Modèle et outils génériques pour la résolution des problèmes liés à la répartition des ressources sur grilles

Julien Gossa

Numéro d'ordre : 2007-INSAL-0085 Année 2007

Institut National des Sciences Appliquées de Lyon
Bâtiment Blaise Pascal, Campus de la Doua, 69621 Villeurbanne Cedex, France.

EDIIS : École Doctorale Informatique Information pour la Société

Modèle et outils génériques pour la résolution des problèmes liés à la répartition des ressources sur grilles

Un petit pas vers la réconciliation des architectures orientées services avec leur infrastructure matérielle

Thèse de Doctorat

Spécialité Informatique

Julien Gossa

Soutenue publiquement le 3/12/2007 devant la commission d'examen composée de

Rapporteurs :	Michelle Sibilla	MdC HDR (IRIT - Université Toulouse III)
	Franck Cappello	DdR (INRIA - Université Paris Sud)
Examinateurs :	Thomas Ludwig	Pr (PVS - Universität Heidelberg) (Président)
	Nouredine Melab	Pr (LIFL - Université des Sciences et Technologies de Lille)
Directeurs :	Lionel Brunie	Pr (LIRIS - INSA Lyon)
	Jean-Marc Pierson	Pr (IRIT - Université Toulouse III)

Laboratoire d'InfoRmatique en Image et Systèmes d'information UMR 5205

Remerciements

Avant d'entrer dans le vif du sujet, je tiens à remercier sincèrement les personnes qui m'ont accompagné durant la longue et douloureuse, mais au combien passionnante réalisation de cette thèse.

Je pense bien sûr à mes encadrants, Lionel et Jean-Marc, qui m'ont donné la chance de découvrir le métier fabuleux d'enseignant-chercheur dont j'espère pouvoir faire mon métier.

Je tiens également à remercier les stagiaires qui ont permis le développement de NDS : Nicolas pour ses beaux logos, Mathieu pour sa parfaite imitation du gars sérieux, Rémi pour avoir trouvé quelques heures pendant la coupe du monde de football et Nipaul pour son bon plan restau combodgien.

Je voudrais aussi remercier mes cothésards : Yann pour ses poils, Rachid pour son bureau, Abdoulaye pour son accent, Ny Haingo pour sa gentillesse, Pascal pour son Linux, Sandro pour sa Napolitude et Romuald sans qui j'aurais du râler, boire et fumer tout seul.

Je n'oublie pas tous les gens avec qui j'ai eu l'occasion de travailler. Toute l'équipe enseignante de l'UFR d'informatique de l'UCBL, j'espère pouvoir encore enseigner le C à la bleusaille. Toute l'équipe administrative pour son travail quotidien et pas toujours marrant. L'ACI « masse de donnée » qui a financé cette thèse ainsi que le projet GGM qui m'a permis de mesurer la difficulté de mener un projet de grande envergure, ainsi que les collègues de ce projet pour les randonnées en raquettes et les bonnes tables. Enfin l'équipe autrichienne qui m'a permis de visiter la superbe ville de Vienne. Et bien sûr toute l'équipe de recherche du LIRIS avec qui j'ai pu lier des liens de sympathie.

Je souhaite également remercier les rapporteurs de cette thèse Michelle Sibilla et Frank Capello pour leurs remarques constructives et pour avoir tenu des délais particulièrement courts, ainsi que les autres membres de la commission d'examen Nouredine Melab et Thomas Ludwig pour avoir témoigner de leur interêt envers mes travaux en acceptant cette charge.

Mais je n'oublie pas non plus mes amis de toujours pour avoir fait semblant d'essayer de comprendre ce que je faisais et mes amis d'ici pour avoir jalousé la flexibilité de mes horaires.

Je pense également à ma famille, qui m'a soutenu durant toutes mes longues et chères études. Mon père pour avoir toujours eu un chien et un PC à la maison. Ma mère pour avoir arrêté de me harceler avec le coiffeur à ma majorité. Mon petit frère pour avoir détourné l'attention en ayant les cheveux longs à la même époque. Mon grand frère pour l'accordéon le dimanche matin à 9 h. Les furets pour leurs beaux bébés et leurs dents acérées. Caroline pour son soutien quotidien et sa cuisine mensuelle mais sophistiquée.

À tous ces gens, et à tous ceux que je ne cite pas mais auxquels je pense, je profite de l'importance de ce manuscrit dans la vie d'un chercheur pour témoigner de toute ma sincère gratitude.

Pour conclure, je voudrais remercier Fleur de Pays pour son tabac de qualité, ainsi que Garth Ennis, Alan Moore et Frank Miller dans le secret espoir qu'ils n'aient jamais été cités dans une thèse (sinon une thèse française, sinon une thèse française d'informatique, sinon une thèse française d'informatique lyonnaise...).

Résumé de la thèse

Depuis plusieurs années, les intergiciels de grille n'ont eu de cesse de gagner en complexité. Grâce à l'adoption des SOA (architectures orientées services), ils permettent aujourd'hui de concevoir des architectures logicielles complexes, de très haut niveau et naturellement distribuées. Les acteurs (administrateurs, développeurs et utilisateurs) sont ainsi quotidiennement confrontés à de nombreux problèmes liés à la nature répartie des ressources : Comment sélectionner une ou plusieurs ressources pour s'acquitter d'une tâche donnée ? Comment déployer une ou plusieurs ressources, voire toute une architecture logicielle ? Comment composer plusieurs ressources ? etc.

La résolution de ces problèmes est difficile car les acteurs travaillent à un niveau très élevé, depuis lequel l'infrastructure matérielle est complètement abstraite. Elle s'avère également particulièrement complexe du fait de la diversité des ressources (du simple fichier à un service web impliquant communications et calculs), de la diversité des objectifs (performances, équilibrage de charge, qualité des réponses, aspects financiers...), de la prise en compte des performances de l'infrastructure et également de la diversité des problèmes (sélection, composition, déploiement...). Or cette résolution est cruciale car elle conditionne les performances des plateformes aussi bien que la quantité de travail des acteurs.

Notre approche propose un service web globus appelé le Network Distance Service. NDS permet de déclarer intuitivement une vaste étendue des problèmes sous forme de graphe grâce à une notion de distance adaptable aux ressources et objectifs et adaptative aux performances de l'infrastructure matérielle. Des algorithmes classiques de la théorie des graphes sont ensuite utilisés pour calculer les solutions : les plus courts chemins sont utilisés pour résoudre les problèmes de sélection et de composition ; un algorithme de clustering est utilisé pour résoudre les problèmes de déploiement.
Deux algorithmes originaux ont également été développé grâce à cette approche : FReDi, capables de piloter dynamiquement des réplicas, et MRKM, capable de prendre les décisions inhérentes au déploiement d'une architecture logicielle complète.
Enfin, un entrepôt de données à été conçu dans le but d'améliorer les capacités d'analyse des outils de surveillance.

Notre approche est illustrée par l'architecture logicielle GGM (http ://liris.cnrs.fr/PROJETS/ggm), ses différentes infrastructures matérielles cibles et les différents problèmes de distribution rencontrés. Des expérimentations grande échelle ont été menées sur la platforme Grid5000 et montrent l'obtention des solutions optimales pour des temps de calcul très court et un travail utilisateur limité.

Mots clés : Grille, Architectures Orientées Services (SOA), surveillance (monitoring), problèmes liés à la répartition (sélection, déploiement, composition)

Thesis abstract

Since many years, ne complexity of grid middlewares never ceased to increase. Thanks to the adoption of SOA (Services Oriented Architectures), they allows to design highly complex, high level, and naturally distributed software architectures. The actors (administrators, developers and users) are thus confronted to many problems related to the distributed nature of resources : How to select one replicated resource to fulfill one given task ? How to deploy one or several resources, or even a complete software architecture ? How to compose several resources ? etc.

Solving these problems is difficult as actors are working at the highest level, from which the infrastructure is fully abstracted. Another difficulty is due to the diversity of resources (from simple files to complex web services implying communications as well as computations), to the diversity of goals (performances, load balancing, response quality, financial aspects...), to the infrastructure performances and to the diversity of problem (selection, deployment, composition...). However, such solving is crucial as it conditions the platform performances as well as the quantity of actors work.

Our proposal is a Globus web services called the Network Distance Service (NDS). NDS allows to model intuitively a large variety of problems as graphs thanks to a notion of distance adaptable to resources and goals and adaptative to infrastructure performances. Classical graph theory algorithms are then used to solve the problems : a shortest-path algorithm is used to solve selection and composition problems ; a clustering algorithm is used to solve deployment problem.
Two genuine algorithms have also been developed thanks to this approach : FReDi, in order to drive dynamically some replicas, and MRKM, in order to deploy a whole software architecture.
Finally, one data warehouse have been designed in order to improve the analysis capabilities of monitoring systems.

Our approach is illustrated by the GGM software architecture (http ://liris.cnrs.fr/PROJETS/ggm), its different target infrastructures and the different distribution problem encountered. Large scale experiments have been conducted on the Grid5000 platform and shows that optimal solutions have been computed in very short solving times and with limited user work.

Key words : grid, Services Oriented Architectures (SOA), monitoring, distribution related problems (selection, deployment, composition)

Table des matières

Première partie

Introduction : problématique et motivations

Les Systèmes d'Information (SI) distribués ont aujourd'hui pris une place cruciale, non seulement dans l'informatique moderne, mais également dans la vie quotidienne des entreprises, des chercheurs et même des particuliers. On peut définir un SI comme un système constitué de l'équipement, des procédures, des ressources humaines et des données qui y sont traitées, dont le but est de fournir de l'information. Les SI distribués sont caractérisés par la distribution spatiale des constituants du SI, qui doivent donc communiquer afin de traiter et fournir l'information. Le moyen de communication le plus connu est le réseau IP dont l'emblème est l'Internet.

Dans cette thèse, nous nous intéressons justement à cette notion de distribution dans ces SI particuliers que sont les grilles informatiques. Ces dernières représentent un enjeu majeur pour l'informatique moderne qui s'explique par deux phénomènes actuels. Premièrement, l'évolution stagnante des technologies de calculs qui limite l'augmentation des puissances CPU et l'envolée des technologies de communications rendent très rentables les architectures matérielles basées sur de nombreuses machines bon marché connectées par un réseau haute performance. Deuxièmement, la mondialisation[1] met l'accent sur des collaborations scientifiques, technologiques et financières à l'échelle planétaire. Ces notions d'exploitation de ressources distribuées et de collaboration sont en effet au cœur même du concept de grille informatique qui cristallise les problématiques majeures des SI distribués.

Dans cette première partie, nous présentons la problématique centrale de cette thèse ainsi que nos motivations. Cette présentation s'articule en trois chapitres. Le Chapitre 1 est une présentation générale des grilles informatiques et de leurs spécificités en terme de problématiques liées à la distribution suivie d'une présentation de trois architectures matérielles cibles de grille, en détaillant la nécessitée de les rendre compatibles par une gestion efficace de la distribution. Le Chapitre 2 détaille les différents problèmes liés à la gestion de cette distribution qui interviennent quotidiennement dans les environnements de grilles. Nous montrerons que la résolution de ces problèmes est critique non seulement pour des aspects de performances, mais également pour favoriser l'adoption des grilles par le plus grand nombre d'utilisateur et assurer la pérennité des plateformes de grille mises en œuvre. De plus, nous identifierons les contraintes et caractéristiques que doivent satisfaire les systèmes de résolution de ces problèmes. Enfin, nous montrerons Chapitre 3 que les solutions présentées dans la littérature ne satisfont pas entièrement ces contraintes, appelant à la conception de nouveaux systèmes de résolution des problèmes liés à la répartition des ressources.

Termes clés :

- *ressources* : toute entité, physique ou logicielle, exploitable présente dans l'environnement, que ce soit de simple fichiers, des applications, des unités de calcul, de stockage ou encore des liens réseau.
- *infrastructure* ou *architecture matérielle* : ensemble des composants physiques présents dans l'environnement. Sommairement, les unités de calcul et de stockage, désignées par hôte, et les unités de communication, désignées par liens réseau.
- *intergiciel* : de l'anglais *middleware*, couche logicielle, souvent composée elle-même de sous-couches internes, servant d'intermédiaire entre les applications et le système d'exploitation afin de fournir des services de haut niveau en environnement distribué (sécurité, indexation, surveillance, etc.) ainsi qu'un niveau d'abstraction supplémentaire.
- *superstructure* ou *architecture logicielle* : ensemble des composants logiques présents dans l'environnement, tels que les données, les services, les applications ou encore les bases de données.

[1] au sens large plutôt qu'au sens économique

- *plateforme* : ensemble infrastructure, intergiciel et superstructure, soit un environnement complet de grille.
- *acteur* ou *utilisateur* : ensemble des intervenants humains, que ce soit les administrateurs, les développeurs ou les utilisateurs finaux.

1

Présentation générale des grilles informatiques

Les grilles informatiques, désignées par « the Grid » par leurs créateurs Foster, Kesselmann et Tuecke dans [Foster 01], font l'objet de recherches intensives depuis le début des années 1990. La Grille était initialement décrite comme « le partage de ressources et la résolution coordonnée de problèmes dans des organisations virtuelles dynamiques et multi-institutionelles ». On ajoute que ce partage ne concerne pas un partage primaire de fichier, mais plutôt un accès direct aux ressources (p. ex. ordinateur, application, donnée ...) comme requis dans un large panel de stratégies de courtage de ressources et de résolution de problèmes, tels qu'il en émerge dans l'industrie, la science et l'ingénierie. Le projet de grille le plus fameux est appelé *Globus*, porté par le *Globus Toolkit* qui est un intergiciel proposant tous les outils nécessaires à la mise en place d'une grille : partage, sécurité, courtage, ordonnancement, surveillance, etc.

Il faut ici rapporter le scénario phare d'utilisation d'une grille : « un fournisseur de service d'application, un fournisseur de service de stockage, un fournisseur de cycles et des consultants sont engagés par un fabriquant de voiture pour réaliser des évaluations de scénarios lors de la planification d'une nouvelle usine ». On voit apparaître dans cet exemple le concept maître des grilles : la collaboration entre plusieurs institutions au sein d'une seule organisation virtuelle.

En effet, le terme « grilles » a récemment gagné en popularité et sert aujourd'hui à désigner une large gamme de SI distribués, tels que le *méta-computing* ou *calcul Internet*. En particulier, on a vu émerger de nombreux projets tels que l'innovant *seti@home*[1] ou le plus récent *World Community Grid*[2]. Aussi désignés par le terme « *Desktop Grid* », en réalité ces environnements ne concernent pas un accès direct aux ressources et ne comprennent pas une notion de collaboration au sein d'organisations virtuelles.

Il faut également noter qu'actuellement le domaine d'exploitation principal des grilles est essen-

[1] http ://setiathome.berkeley.edu/
[2] http ://www.worldcommunitygrid.org/

tiellement scientifique et concerne le partage des ressources de grands centres de calculs composés de grappes informatiques relativement homogènes et extrêmement puissantes. En revanche, les *Desktop Grids* présentent un aspect propre aux grilles qui est d'être basées sur des ressources hétérogènes, potentiellement relativement peu puissantes et partagées.

On peut donc formuler une remarque générale sur la notion de grilles qui a des difficultés à converger vers une unique définition claire et précise, intégrant toutes les problématiques initialement prévues par ses concepteurs.

Dans le cadre de cette thèse, nous distinguons les grilles des autres SI par les définitions suivantes. Le but des grilles informatiques est de favoriser la collaboration à très large échelle entre plusieurs partenaires menant un projet commun nécessitant des calculs très intensifs et/ou sur des volumes de données très importants. De telles collaborations sont supportées par le partage de ressources hétérogènes et dynamiques au sein d'organisations virtuelles (Virtual Organizations - VO) regroupant ces différents partenaires. Ces notions d'*intensivité*, d'*hétérogénéité*, de *dynamicité* et de *VO* sont au cœur de la grille et en font leur particularité.

On peut d'ores et déjà décrire trois problématiques classiques des grilles, qui peuvent également se retrouver dans d'autres SI distribués :

hétérogénéité : les ressources présentes dans les SI distribués présentent des caractéristiques différentes. Par exemple certaines machines peuvent disposer de peu de mémoire vive, de faibles capacités de calcul mais d'une très grande capacité de stockage, alors que d'autres disposent du contraire. Ainsi, connaitre leurs caractéristiques permet de les employer à bon escient.

dynamicité : en plus de leur hétérogénéité, les ressources changent d'état au court du temps. Elles peuvent subir des pannes ou bien simplement être chargées ou même saturées. Dans certains contextes, elles peuvent même apparaître et disparaître spontanément, voire se déplacer physiquement.

intensivité : afin de rentabiliser leur investissement, les ressources peuvent (et même doivent) être soumises à une utilisation intensive, car une ressource non utilisée est une ressource gâchée. Cette problématique revient au problème du passage à l'échelle, aussi désigné par l'anglicisme *scalabilité*. Ce terme désigne la capacité d'une solution à supporter l'augmentation d'une certaine quantité, que ce soit en terme du nombre de ressources, du nombre d'utilisateurs ou encore de taille ou de nombre de données et de tâches.

évolutivité et pérénité : toujours dans un souci de rentabilité, il est important de prendre en compte la propension des plateformes à se retrouver confrontées aux changements futurs. Ces derniers peuvent concerner l'infrastructure matérielle, qui peut être étendue/réduite lors de l'intégration/sortie de nouveaux partenaires et ainsi subir d'importants changements de capacité ou même de nature avec, par exemple, l'intégration d'équipements mobiles sans fil. Ils peuvent également concerner la superstructure logicielle, avec l'ajout de nouveaux composants, ou les utilisateurs avec l'apparition de nouveaux objectifs et autres préoccupations. L'extensibilité et la réutilisabilité sont des aspects majeurs, car ils conditionnent la pérennité de l'environnement de grille, bien qu'ils soient rarement abordés dans les travaux scientifiques.

1.1 Petite histoire des grilles

Rapidement, le monde des grilles s'est partagé en deux groupes : les grilles de calcul et les grilles de stockage. Ces deux mondes ne mettaient pas en jeu les mêmes compétences, les mêmes technologies, les mêmes problématiques ou encore les mêmes matériels. Au début des années 2000, ces deux mondes ont commencé à fusionner, de par l'implication naturelle de ces deux aspects dans les applications réelles (cf. Section 1.2.1) et en grande partie sous l'influence du projet Globus Alliance[3]. Ce dernier, mené par Foster et Kesselman, communément considérés comme les créateurs de la notion de grille, propose un des intergiciels de grille les plus complets, le Globus Toolkit, qui se veut générique et ainsi ne privilégie pas un aspect par rapport à l'autre.

Plusieurs modèles fondamentaux ont été conçus afin de construire des environnements de grille. Parmi les plus populaires, on peut noter le modèle maître-esclaves, aussi appelé *task farming* ou *work queue*. Il comprend une unique machine maître qui détient la liste des tâches à effectuer et les distribue sur un ensemble de machines esclaves selon une stratégie bien définie. Deux implémentations très populaires de ce modèle sont Condor [Thain 05] et XtremWeb [Fedak 01]. Le défaut majeur de cette approche est le goulot d'étranglement et point unique de défaillance (*single point of failure*) que représente le maître, qui risque la saturation et compromet l'ensemble du système en cas de panne.

Le modèle client-agent-serveur, aussi désigné par *GridRPC* (pour *Remote Procedure Call*), est basé sur l'appel de procédure à distance. Dans ce modèle, chaque serveur déclare ses ressources auprès d'un agent, souvent appelé *Registry* ou *Resource Management System* (RMS) ou encore courtier (*broker*). Le client s'adresse alors à l'agent pour connaître les serveurs pouvant satisfaire ses besoins. Il peut ensuite directement contacter un serveur pertinent afin d'exécuter sa tâche. Les implémentations les plus complètes sont NetSolve [Seymour 05], DIET [Caron 06b] et Globus [Foster 01]. Ce modèle ne présente pas l'inconvénient des approches maître-esclaves, mais implique une gestion des ressources plus complexe au niveau de l'intergiciel, du fait de la décentralisation des connaissances et des décisions.

Une des évolutions les plus marquantes concerne l'*Open Grid Services Architecture* (OGSA) proposé par le *Global Grid Forum* (GGF). Initialement, les Grilles étaient conçues pour partager directement les ressources matérielles. Mais cela imposait de graves limitations dans le partage des applications, qui parallèlement a commencé à prendre de plus en plus d'importance. En janvier 2002 eut lieu la première démonstration de l'implémentation d'un service de grille (*Grid Service*) lors du *Globus Toolkit tutorial* tenu au *Argonne National Laboratory*. La notion de service permet de considérer les ressources applicatives au même titre que toutes autres ressources logicielles, type données. Ainsi, les mêmes mécanismes de partage, de gestion et de contrôle pouvaient servir à les manipuler : les partenaires pouvaient désormais partager leurs données, leurs ressources de calculs, mais aussi leurs traitements au sein des VO. Loin d'être neutre, ce changement signifiait une réelle montée en niveau de l'architecture logicielle. En effet, pourquoi l'utilisateur devrait s'intéresser aux ressources bas-niveau que sont les données ou les processeurs lorsqu'il peut manipuler directement les applications qu'il utilise ? Dès lors, une charge supplémentaire s'est mise à peser sur les intergiciels : cacher les ressources bas-niveaux, s'éloignant un peu du contexte initial de partage direct des ressources. OGSA est implémenté dans Globus depuis la version 3 du GT au travers de l'*Open Grid Services Infrastructure* (OGSI).

[3] www.globus.org/

Dans les sections suivantes, nous allons présenter trois déclinaisons des environnements de grilles : les grilles et architectures orientées services, les grilles pervasives et les organisations virtuelles inter-grille. Nous nous efforçons de mettre en perspective les motivations relatives aux problématiques liées à la répartition des ressources, par rapport aux spécificités de chacun d'eux, en insistant tout particulièrement sur les interactions dans le système *utilisateurs / superstructure logicielle / infrastructure matérielle*. L'importance de la présentation de ces trois environnements ne se situe pas tant dans leurs différences que dans leur point commun essentiel : ils sont tous trois conçus pour gérer des tâches intensives issues de collaborations étroites et dynamiques. En cela, ils peuvent être amenés à fusionner, soit par la naissance d'une collaboration entre deux environnements de types différents, soit par l'évolution et l'extension d'un environnement afin de couvrir de nouveaux besoins. De telles fusions impliquent un important nombre de problèmes parmi lesquels les problématiques liées à la répartition des ressources tiennent une place primordiale.

1.2 Les grilles et architecture orientées services

Un problème rapidement identifié est le manque d'ouverture et la spécificité des spécifications et standardisations autour d'OGSA. En effet, jusqu'à présent les grilles restaient des univers fermés inaccessibles au commun des acteurs du monde informatique, qui devaient se former à ces technologies particulières pour y prendre part et ne pouvaient pas réellement réutiliser ces connaissances dans d'autres environnements. C'est pourquoi dès la version 4, le GT adopta les standards développés par l'*Organization for the Advancement of Structured Information Standards* (OASIS). Le cheval de bataille de cette organisation est la standardisation des services web, qui sont des applications utilisables via le web, dont l'interface est décrite en *Web Service Description Langague* (WSDL) et qui utilisent pour communiquer le protocole *Simple Object Access Protocol* (SOAP). Grâce à ces standards, tout développeur web classique peut mettre à disposition ses propres services sur la grille, générant une grande ouverture au prix d'une augmentation de l'écart entre les problèmes intrinsèques de la grille et les préoccupations de ses acteurs.

Mais le plus grand effet de cette ouverture est sans doute le changement de l'infrastructure matérielle sous-jacentes aux grilles. En effet, compte tenu de la particularité des technologies mises en œuvre, les grilles étaient cantonnées aux applications scientifiques très spécifiques sur de puissantes grappes et n'avaient une percée dans le monde industriel que très limitée. Or avec cette nouvelle ouverture, on peut désormais envisager de mettre en place des grilles entre quelques dizaines de machines aux performances modestes, reliées par l'Internet. Mais le cas le plus intéressant concerne les grilles mixtes : l'architecture la plus performante et à même d'être exploitée, est celle qui peut permettre le partage des ressources les plus hétérogènes et les plus dynamiques. En effet, on peut maintenant imaginer des grilles composées non seulement de grappes hautes-performances assurant d'une part le stockage d'énormes volumes de données et d'autre part l'exécution d'énormes volumes de calculs ; mais aussi de machines isolées aux performances limitées et sans garantie de persistance, ne servant de fournisseurs de données et de services, et pouvant être employées pour effectuer certaines tâches modestes.

Finalement, on peut noter que cette démocratisation des grilles va inévitablement avoir un effet sur les utilisateurs potentiels des grilles. De professionnels scientifiques avertis, ils seront bientôt de simples professionnels, utilisant la grille au même titre que le web ou les courriels, réalisant ainsi la vision originaire des concepteurs de la Grille. Mais cela aura également un impact important sur

la grille : il faudra compter sur des utilisateurs plus nombreux, plus exigeants, plus hétérogènes et dynamiques, moins compétents et sensibles aux problématiques... mais surtout moins efficaces en ce qui concerne l'administration et paradoxalement ayant plus de responsabilités. En effet puisque tout un chacun pourra désormais apporter sa pierre (logicielle ou autre) à la grille, tout un chacun en sera responsable. Or les mécanismes actuels de gestion des ressources de la grille sont très bas niveau, et donc très loin des inquiétudes de ces nouveaux utilisateurs. On ne peut malheureusement pas imaginer une adaptation de ces mécanismes internes, tant l'étendue des attentes et des objectifs de ces utilisateurs sera diverse et variée. La grille doit donc se doter au plus vite de solutions de gestion des ressources adaptées à leurs besoins, adaptables à leurs objectifs, adaptatives à l'infrastructure et surtout accessibles.

En définitive, on peut donc observer que les intergiciels de grilles, qui initialement ont été conçus pour assurer les besoins primaires des collaborations (partage, sécurité, etc.), ont maintenant une charge bien plus importante puisqu'ils doivent assurer la gestion intégrale et transparente des ressources bas-niveaux dans un contexte très difficile, puisqu'ils sont maintenant confrontés à une très grande hétérogénéité et une dynamicité toujours croissante des ressources et des utilisateurs, de moins en moins concernés et compétents. Ce qui appelle à la conception de nouvelles solutions de gestion des ressources.

Dans la section suivante, nous présentons un exemple concret de grille.

1.2.1 Un exemple concret : la Grille Géno-Médicale

Le projet GGM, Grille Géno-Médicale, vise à proposer une architecture logicielle orientée service s'appuyant sur l'intergiciel de grille Globus, en vue de gérer des données hétérogènes et dynamiques au sein d'entrepôts de données distribués, à des fins d'analyse et de traitement intensifs, entre autres avec de la fouille de données. Ce projet a été financé par l'ACI « masse de données » pour la période 2004/2007.

Ce défi est particulièrement important dans le cadre des grilles biomédicales. En effet, la diffusion des technologies haut débit en génomique/protéomique et la gestion informatique du dossier médical personnalisé ouvrent des perspectives diagnostiques totalement novatrices. Parce qu'elles exigent une capacité d'analyse et de traitement considérable et un partage d'informations hétérogènes et très volumineuses à grande échelle, ces technologies apparaissent comme des cibles naturelles des grilles. Or, plusieurs solutions nécessaires à leur mise en œuvre effective n'existent pas actuellement :

- les entrepôts de données n'ont pas encore été déployés sur grille de calcul, et des problèmes multiples de gestion de données dans ce contexte (hétérogénéité, dynamicité, sécurité, traçabilité, efficacité d'accès) doivent être résolus. Cet aspect fait l'objet du travail de l'équipe « Systèmes d'Information Spatio-Temporels et Entreposage »[4] du LIRIS, Lyon, composée de Pascal Wehrle, Anne Tchounikine et Maryvonne Miquel, présenté dans [Wehrle 07].
- le portage efficace sur grille de calcul des algorithmes d'extraction de connaissances (ou fouille de donnée) doit être généralisé à des masses importantes de données dynamiques et hétérogènes réparties à grande échelle. Cet aspect est couvert par le travail de l'équipe « Optimisation

[4] http ://liris.cnrs.fr/equipes ?id=51

PArallèle Coopérative »[5] de l'INRIA Dolphin, Lille, composée de Sébastien Cahon, Nouredine Melab et Talbi El-Ghazali, présenté dans [Melab 06].

- un système de caches collaboratifs visant à optimiser/réguler l'utilisation des ressources de stockage et assurer une adaptation des données aux droits et besoins des utilisateurs finaux doivent être développés. Cet aspect est développé par l'équipe « Systèmes d'Information Pervasifs » du LIRIS, Lyon, composée de Yonny Cardenas, Jean-Marc Pierson et Lionel Brunie, et présenté dans [Cardenas 06].
- un système d'exécution distribuée de requêtes d'interrogation de bases de données doit être développé afin de supporter de très grands volumes de données dans un environnement hétérogène et dynamique. Cet aspect est l'objet du travail de l'équipe « Optimisation dynamique de requêtes réparties à grande échelle »[6] de l'IRIT, Toulouse, composée de Mohammed Hussein, Mahmoud El Samad, Franck Morvan et Abdelkader Hameurlain, présenté dans [Hussein 06].

Au-delà du domaine médical, ces problématiques se situent au cœur de l'ouverture des grilles vers la gestion de communautés et d'organisation virtuelles et en conditionnent le déploiement. Par la complexité et le caractère critique des données et des processus impliqués, l'analyse géno-médicale et le diagnostic intégré constituent des champs d'application particulièrement pertinents en termes scientifiques comme en termes d'impact socio-économique. Les détails de la superstructure GGM ont été présentés dans [Pierson 05] et [Pierson 07].

La gestion des ressources vue par les services et leurs développeurs

L'architecture logicielle GGM, présentée Figure 1.1, compte donc globalement quatre services dont les besoins relatifs aux problématiques liées à la répartition des ressources présentent de nombreuses différences :

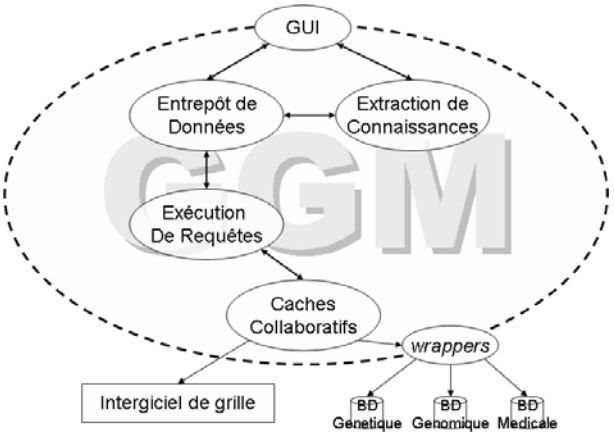


FIG. 1.1 – Architecture Logicielle GGM

- Service d'entrepôt distribué

[5] http ://www2.lifl.fr/OPAC/
[6] http ://www.irit.fr/-Equipe-PYRAMIDE-

Ce service doit permettre le stockage structuré de très grands volumes de données en vue d'analyses complexes. Les problèmes majeurs gérés par les développeurs sont l'indexation des données et les mécanismes d'agrégation distribués. On peut remarquer que de nombreuses questions se posent à eux : quelles données répliquer et comment ? Où les placer ? Comment les sélectionner ? Comment les découper ? Où effectuer les calculs ? Matérialiser ou pas les agrégations ? Or du fait des très grands volumes, ces différents choix conditionnent de façon critique les performances de l'entrepôt et s'imposent en permanence.

- Service d'extraction de connaissances

  Ce service doit permettre la fouille des données contenues dans l'entrepôt. Les principaux objectifs des développeurs sont de définir des algorithmes distribués assurant la pertinence des résultats et de mettre en place des interfaces permettant l'utilisation de ces algorithmes par des non-informaticiens. Là encore, des choix doivent être pris, essentiellement en ce qui concerne le découpage des données à fouiller, le choix des hôtes d'exécution de ces algorithmes et la charge qui leur sera attribuée.

- Service de caches collaboratifs

  Ce service doit prendre en charge le stockage des données. Les objectifs des développeurs sont de gérer l'indexation, les protocoles d'échange et de recherche, ainsi que l'annotation sémantique des données. En revanche, les développeurs doivent également prendre en charge le déploiement et le placement des données. Ces deux problématiques sont rendues très compliquées par le fait que les données stockées sont exploitées par des tiers, et donc avec des objectifs différents. Par exemple, les données de l'entrepôt ne devront pas être gérées de la même manière que les contenus multimédias.

- Service d'exécution distribué de requêtes

  Ce service doit gérer l'exécution optimisée de requêtes SQL du type Sélection-Projection-Jointure (SPJ). L'objectif principal des développeurs est d'embarquer les mécanismes nécessaires dans des agents mobiles capables de prendre des décisions optimisées quant aux lieux d'exécution des opérateurs SQL.

  Ce service est confronté à un problème technique assez fréquent : les spécificités de ses décisions nécessitent une connaissance avancée des performances. Par exemple, il doit disposer des débits des disques, ce qui doit s'acquérir par des outils spécifiques. Ainsi, ce service doit pouvoir utiliser plusieurs outils de surveillance simultanément, ce qui s'avère délicat compte tenu du choix technologique des agents mobiles qui impose des limites dans la taille des codes sources embarqués.

On peut noter un point commun entre tous ces services : le fait que leurs développeurs, en plus des problématiques spécifiques à leurs préoccupations scientifiques, doivent se confronter aux problématiques liées à la répartition des ressources. Il faut également noter que les critères impliqués dans ces décisions sont divers : capacité de stockages, capacités de calculs ou encore capacités de communication et topologie de l'infrastructure. De plus, on peut remarquer que les objectifs de ces gestions sont aussi divers et spécifiques : tantôt de l'équilibrage de charge, tantôt de l'optimisation des temps d'exécution... Et que la nature même des ressources gérées est également diverse et spécifique : des données d'entrepôt, des algorithmes d'extraction, des fichiers annotés ou des relations de base de données. La remarque importante est que ces décisions sont toujours intimement liées aux capacités et à l'état de l'infrastructure matérielle. Malheureusement, trois considérations limitent les connaissances des développeurs sur les infrastructures. Premièrement, les infrastructures sont à différentes échelles dynamiques, ce qui rend de fait leur état inconnu dans l'absolu. De plus, les développeurs

n'ont pas nécessairement connaissance de l'infrastructure cible de leurs applications, laquelle peut de surcroit être en réalité multiple : l'architecture GGM est développée dans une optique de logiciel libre ayant pour vocation d'être rendu disponible sur le web et utilisable par tout un chacun sur sa propre infrastructure. Troisièmement, la réutilisabilité impose qu'une application, même développée spécifiquement pour un environnement cible bien connu et peu dynamique, puisse survivre à une évolution importante ou être par la suite réutilisée dans un cadre différent.

Enfin, un aspect rarement abordé par les développeurs de services à l'heure actuelle, mais qui ne manquera pas d'accompagner une plus grande démocratisation des grilles, est l'aspect financier. En effet, on peut prévoir que l'utilisation de certaines ressources soit soumise à des contre-parties financière. De plus, cette contrepartie risque fort d'être proportionnelle au volume de l'utilisation. Par exemple, on peut imaginer qu'un espace de stockage soit payant en fonction du volume des données stockées et du temps de stockage, ou encore que certaines bases de données externes à la grille facturent le volume de données délivré. Dans ce dernier cas, on peut également noter l'importance d'autres aspects, par exemple la fraîcheur ou la précision des données délivrées. De fait, tous ces aspects peuvent être qualifiés de *sémantiques*[7] et conditionnent également les performances de l'architecture logicielle. Bien que ne concernant pas directement les temps d'exécution ou encore l'équilibrage des charges, on peut noter qu'ils y sont intimement reliés, compte tenu du fait que les ressources payantes sont probablement plus performantes que les ressources gratuites. Ainsi, ces critères font partie intégrante des décisions relatives aux problématiques liées à la répartition des ressources : selon des choix propres aux développeurs, au contexte et à la ressource, il faudra par exemple utiliser une ressource payante mais proposant une prestation supérieure, ou au contraire une ressource gratuite, ou moins chère, au détriment de la qualité.

En conclusion, on peut dire qu'il est fort probable que les développeurs présentent des lacunes face aux problématiques liées à la répartition des ressources, que ce soit de compétence, d'intérêt ou de temps à consacrer face à des contraintes omniprésentes de retour sur investissement. De plus, la gestion de ces problématiques doit être nécessairement automatisées afin de supporter les changements transparents des infrastructures matérielles et de s'adapter aux différentes dynamicités, que ce soit de la topologie ou de l'utilisation de l'infrastructure matérielle, ou de l'utilisation de la ressource développée en soi. En outre, il semble impossible de laisser les unités de gestion des ressources de la grille les prendre en charge, car ceux-ci sont trop bas niveau et n'ont pas été conçus pour considérer tant de diversité et de spécificité, qui plus est dans un cadre de forte dynamicité.

La gestion des ressources vue par les administrateurs

Une fois la phase de développement achevée, commence une autre phase cruciale impliquant la gestion des ressources : l'administration de l'architecture logicielle. Dans un premier temps, celle-ci consiste à effectuer le déploiement initial sur l'infrastructure matérielle. On peut raisonnablement considérer que les administrateurs en charge de cette tâche possèdent une bonne connaissance de l'infrastructure. En revanche, il semble impossible de considérer qu'ils soient également en possession totale des subtilités relatives aux ressources logicielles à déployer. Il faut pourtant qu'ils décident combien d'instances de chacune seront déployées et où elles devront être placées, ce qui ne peut être correctement effectué que sur les bases d'une connaissance solide des ressources, de leurs charges prévues et de leurs interactions potentielles. Dans un second temps, cette administration consistera à

[7] Le terme *sémantique* est à prendre ici au sens large pour désigner tout aspect impliqué dans une prise de décision liée à la distribution des ressources, mais sans rapport direct aux performances.

déployer ou redéployer de nouvelles instances afin d'adapter l'architecture à l'évolution des besoins, à celles de l'infrastructure et aux pannes éventuelles.

On peut donc identifier deux problèmes : le premier concerne la connaissance des ressources gérées qui peut receler des subtilités que seuls leurs développeurs peuvent réellement maîtriser ; le second concerne l'évolution dynamique de la superstructure logicielle qui doit s'adapter à celle de l'infrastructure matérielle.

1.3 Les Grilles pervasives

L'année 2006 a vu l'émergence d'un nouveau type d'architecture appelé *Grilles Pervasives*. Le concept original est d'allier la puissance de calcul et de stockage des grilles avec les capacités de communications et d'interopérabilité propres à l'informatique pervasive [8]. Pour ce faire, différentes philosophies sont envisagées. Certains, comme Pierson dans [Pierson 06], proposent d'intégrer directement les équipements mobiles au cœur de la grille afin de permettre leur participation directe aux calculs. Notre approche se base plutôt sur un scénario de VO présenté dans les travaux initiaux de Foster ainsi que dans la Figure 1.2 :

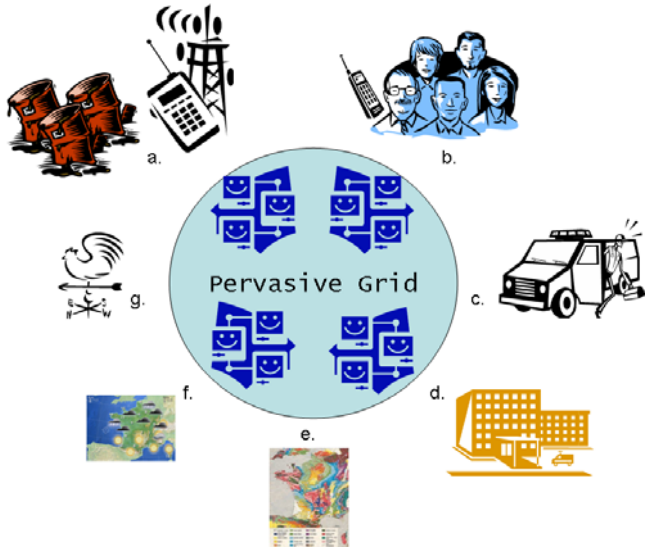


FIG. 1.2 – Exemple d'utilisation de grille pervasive

« *Une équipe de gestion de crise réagit à une fuite de produits chimiques (a) en utilisant la météo locale (f) et les modèles géologiques (e) pour estimer la propagation de la fuite et déterminer son impact en se basant sur la localisation des populations (b) ainsi que sur les caractéristiques géographiques telles que les rivières ou les réservoirs d'eau, afin de créer un plan de migration à court terme (peut-être basé sur des modèles de réactions chimiques), et d'affecter les équipes d'intervention d'urgence (c) en planifiant et coordonnant les évacuations, notifiant les hôpitaux (d), ainsi de suite.* » [Foster 01].

[8] L'informatique pervasive, aussi appelée ubiquitaire ou mobiquitaire, désigne une tendance apparue au début des années 2000 qui consiste à déployer l'informatique dans tout l'environnement des utilisateurs tout en masquant leur présence. Elle est caractérisée par l'utilisation massive d'équipements mobiles sans fil et une très grande transparence des ressources informatiques.

Dans ce type de VO, on peut aisément évaluer les bénéfices d'une ouverture pervasive de la grille, en particulier en ce qui concerne la coordination des équipes d'intervention, la notification des populations locales et des hôpitaux, mais aussi par l'intégration de capteurs (g) déployés sur le lieu de l'accident, aspect non prévu dans le scénario initial.

Il est important de noter que ce type d'utilisation a rapidement été abandonné dans les travaux généraux sur la grille. Ils ont été essentiellement repris dans la branche des *grilles temps-réels* dont la popularité n'est pas comparable avec celles de projets tels que Globus. Ce dernier n'est d'ailleurs pas prêt à supporter une ouverture pervasive. En effet, on peut raisonnablement envisager d'implémenter les solutions de communication, d'interopérabilité et de sécurité propre à l'informatique pervasive en périphérie de la grille afin de permettre physiquement cette fusion. En revanche, les mécanismes internes de gestion des ressources de la grille ne sont pas prévus pour supporter l'impact d'une telle ouverture. En particulier, elle démultiplierait la dynamicité des grilles : des centaines de ressources (capteurs, équipements personnels mobiles...) seraient susceptibles d'être intégrées spontanément en certains points d'accès, déséquilibrant dramatiquement l'infrastructure et son utilisation. Or, il est clair que le maître-mot dans une telle situation est le temps de réponse : on ne peut pas se permettre (comme dans les grilles classiques) de prendre plusieurs heures pour répondre. De plus, de nombreuses ressources devront être déployées sur la grille pour répondre aux nouveaux besoins : au moment même de crise, les différents services d'analyse, ainsi que les modèles ne seront pas nécessairement disponibles sur la grille, mais dans des centres spécialisés, ils devront donc être déployés sur l'infrastructure matérielle de la grille avant d'être utilisés.

Pour pouvoir assurer des temps de réponse acceptables, il est nécessaire de gérer la répartition de ces nouvelles ressources, et éventuellement de re-gérer celles déjà présentes dans la grille, avec une très grande adaptabilité relative à ces ressources et une très grande adaptativité relative à l'infrastructure, afin d'assurer les meilleures performances possibles dans un cadre d'extrême dynamicité et d'hétérogénéité accrue.

1.4 Les organisations virtuelles inter-grilles

Dans la vision originale, la Grille (*the Grid*) désignait une ressource viscéralement pervasive : une entité disponible depuis n'importe quel point de connexion, au même titre que le Web ou l'électricité. Force est de constater que cet impressionnant défi est loin d'être relevé (et peut-être même relevable). On a ainsi vu apparaître de nombreuses solutions et implémentation de grilles, toutes différentes et spécifiques à un problème donné. Bien que positif en ce qui concerne la faisabilité et l'efficacité, cet état de fait implique une grave limitation en ce qui concerne la collaboration : ces grilles indépendantes n'étant pas conçues pour communiquer et étant donc dans l'impossibilité de supporter des collaborations transversales. C'est pourquoi de nouveaux concepts de substitution on fait leur apparition dans le monde des grilles, qui en gagnant un pluriel, en perdait sa majuscule.

On peut par exemple citer le projet GridBus présenté par de Assuncao et Buyya dans [de Assuncao 06]. Ce dernier propose le concept de *Grid Islands* dont le but est de fédérer plusieurs sous-grilles plutôt que d'en considérer une unique à l'échelle de la planète. Le problème majeur de cette approche est qu'elle implique que les superstructures logicielles soient à la base conçues pour l'intergiciel GridBus. Cette contrainte est particulièrement importante, car le choix de l'intergiciel est une étape fondamentale et critique lors de l'élaboration d'une superstructure.

Une seconde approche pour permettre la collaboration inter-grilles consiste à uniformiser les intergiciels. En effet, avec des intergiciels identiques, ou du moins compatibles (ce qui à l'heure actuelle revient à non seulement utiliser le même intergiciel, mais aussi la même version), la gestion interopérabilité se réduit à un problème de configuration et de paramétrage, qui en soit peut déjà être assez complexe. Mais cette approche implique un effort important de concertation et de mise à jour des plateformes. Elle est donc non seulement très coûteuse, mais aussi inadéquate en ce qui concerne les VO dynamiques.

Une troisième approche consiste à développer des mécanismes *ad hoc* afin d'assurer l'interopérabilité entre les différentes grilles. Cette approche est donc moins coûteuse puisque la migration d'intergiciel n'est pas nécessaire. En revanche, elle reste peu adaptée aux VO dynamiques, car elle nécessite toujours d'importants efforts de concertations et de développement.

En revanche, on peut constater que bien que des grilles indépendantes ne puissent pas directement communiquer, un utilisateur peut très bien disposer des droits et de l'équipement logiciel nécessaire à l'utilisation simultanée de plusieurs grilles. Ainsi, l'utilisateur se trouve naturellement au centre des interactions entre différentes grilles indépendantes. L'approche que nous avons adoptée est donc plutôt basée sur la fédération au niveau utilisateur de plusieurs grilles indépendantes.

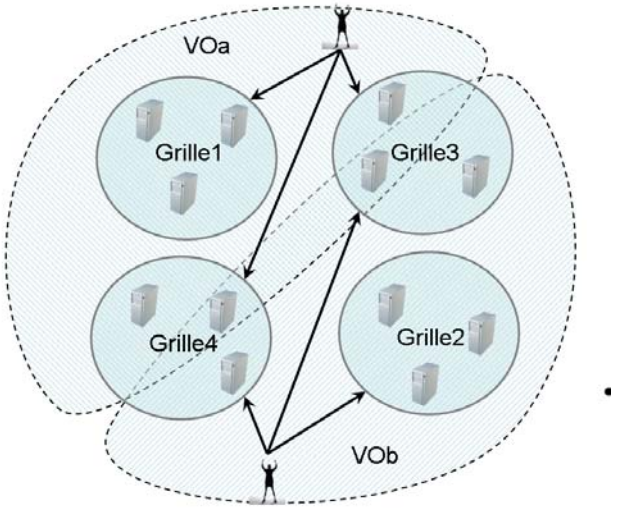


FIG. 1.3 – Un exemple d'organisations virtuelles inter-grilles orientées utilisateur

Nous avons décrit Figure 1.3 un scénario issu du monde médical illustrant cette situation que nous qualifions de *VO inter-grilles orientées utilisateur*. Nous considérons quatre grilles indépendantes : $Grille1$ et $Grille2$ supportent les Système d'Information (SI) de deux centres scientifiques regroupant des chercheurs respectivement en cancérologie et génétique, $Grille3$ et $Grille4$ supportent le SI de deux hôpitaux distincts. Les scientifiques des deux centres souhaiteraient, afin de mener à bien leur recherche, utiliser certaines ressources appartenant aux hôpitaux : il s'agit ici de deux VO VOa pour la cancérologie et VOb pour la génétique. Chacune de ces VO regroupe plusieurs grilles distinctes dont les utilisateurs sont le point commun.

En ce qui concerne les problématiques liées à la répartition des ressources, l'utilisateur est donc en charge de différentes décisions car, dans ce contexte, il est impossible de s'appuyer sur les intergiciels, ceux-ci étant nécessairement indépendants et hétérogènes. Par exemple, l'utilisateur devra

décider sur quelle grille stocker ses données de travail ou encore quelle grille contacter pour s'acquitter d'une tâche. Là encore, ces décisions sont intimement liées aux préférences de l'utilisateur, mais aussi aux performances des différentes infrastructures matérielles et aux caractéristiques des superstructures logicielles. Or ces décisions dépassent les compétences et intérêts des utilisateurs, *a priori* non-informaticiens de formation. De plus, ces décisions risquent de s'avérer aussi nombreuses que fréquentes. Il est donc nécessaire de mettre en place leur automatisation, et ce de façon abordable pour les utilisateurs dont les préoccupations ne comptent pas les problématiques de gestion de la répartition des ressources.

1.5 L'évolution et fusion des environnements, le Darwinisme informatique

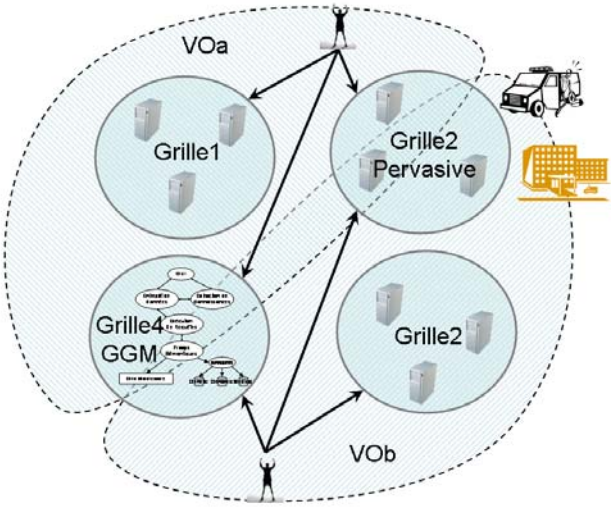


FIG. 1.4 – Un exemple de fusion d'une grille classique avec une grille pervasive dans un contexte inter-grille

L'aspect le plus intéressant dans les trois environnements qui viennent d'être présentés est la pertinence de leur fusion potentielle. En effet, comme il est présenté dans la Figure 1.4 :

- une grille classique telle que GGM peut très bien se doter d'une ouverture pervasive afin de permettre, par exemple, aux équipes d'intervention médicales de consulter et d'annoter les dossiers médicaux des patients.
- elle peut également avoir intérêt à se connecter à d'autres grilles, mais avec un faible couplage : des scientifiques travaillant en cancérologie pourraient avoir à mener des études sur les facteurs génétiques en fournissant leurs propres données à l'entrepôt afin d'y extraire des connaissances spécifiques, sans pour autant mettre leur propre matériel à disposition.
- ces évolutions peuvent se faire de manière totalement transparente dans un contexte de VO inter-grilles.

En plus des problématiques classiques d'interopérabilité des ressources et de gestion de la sécurité, les problématiques liées à la répartition des ressources prennent ici une place cruciale. Effectivement, de telles fusions auraient un impact profond sur le couple *infrastructure matérielle / superstructure logicielle*. Ils se trouveraient subitement modifiés tant dans les faits : extension de la plateforme physique, augmentation du nombre de ressources logiques ; que dans leur finalités : multiplication des objectifs, modification des problématiques principales, introduction de compromis.

On peut remarquer que, dans une certaine mesure, le monde de l'informatique n'échappe pas aux lois de l'évolution édictées par Darwin : les environnements apparaissent et disparaissent au gré des besoins et des contraintes des contextes. Le maître-mot dans le Darwinisme est l'*adaptation*. C'est pourquoi il est d'une importance majeure de concevoir des outils de gestion qui soient tout à la fois génériques, adaptables et adaptatifs, et ce avec un minimum d'intervention utilisateur. Ces outils, loin d'uniquement simplifier la vie des acteurs des grilles, peuvent en réalité permettre d'assurer la pérennité des environnements. Bien que les solutions trop spécifiques remplissent pleinement leur rôle et ce de façon (potentiellement) très optimale, elles ne survivent généralement pas à un changement de contexte ou de problématique, ou alors au prix d'incommensurables efforts manuels d'évolution. Les solutions génériques en revanche sont caractérisées par une très grande adaptabilité, ne demandant pas (ou peu) d'effort lors des évolutions. On peut prendre l'exemple d'Internet qui perdure depuis plusieurs décennies malgré une évolution constante des technologies et de son utilisation. Compte tenu de la vitesse à laquelle le monde informatique est développé, ces évolutions ne relèvent même plus d'un pari mais d'un état de fait, devant désormais être pris en compte dans la conception des plateformes.

1.6 Synthèse

Dans ce chapitre, nous avons présenté le contexte dans lequel se situe notre thèse. Les grilles informatiques sont caractérisées par de grandes dynamicités et hétérogénéités qui se retrouvent tant au niveau de l'infrastructure matérielle, qu'à ceux de la superstructure logicielle, des utilisateurs et des usages, et ce dans l'optique de s'acquitter de tâches très intensives. De plus, on trouve au cœur de la notion de grille, la notion d'organisation virtuelle qui encadre les collaborations inter-organisations, permettant l'exploitation des grilles dans des contextes tant scientifiques qu'industriels.

Nous avons ensuite présenté trois environnements dérivés des grilles et montré leurs besoins en terme de problématiques liées à la répartition des ressources :

- les grilles et architectures orientées services au travers du projet GGM.
- les grilles pervasives au travers d'un scénario de réponse à une situation d'urgence.
- les organisations virtuelles inter-grilles au travers d'un scénario médical.

Leurs caractéristiques et objectifs communs impliquent qu'ils puissent être amenés à fusionner, et ce de façon potentiellement transparente. De telles fusions engendrent des évolutions dramatiques des environnements.

Des solutions de gestion génériques des problématiques liées à la répartition des ressources doivent donc être développées afin de supporter ces évolutions et d'assurer la pérennité des plateformes.

Dans le chapitre suivant, nous étudions en détail les différents problèmes de répartition des ressources qui se rencontrent dans les grilles.

2

Les problématiques de la gestion de distribution

Dans ce chapitre, nous nous intéressons aux différents problèmes de répartition des ressources. Ceux-si se retrouvent dans chacun des environnements de grille présentés. Nous utilisons cette présentation pour identifier les problématiques connexes à la résolution de ces problèmes, ainsi les contraintes et caractéristiques des systèmes de résolution. Nous montrons également le caractère critique de ces problèmes, d'un côté pour les performances de la plateforme et d'un autre pour le travail des développeur et les retours sur investissement.

On peut identifier trois problèmes classiques et quotidiens : le déploiement de ressources, présenté Section 2.1 ; la sélection de ressource, étudié Section 2.2 ; et la composition de ressources, développée 2.3. En plus de ces cas classiques, la répartition des ressources peut intervenir de façon spécifique dans des contextes bien particuliers. Il est impossible de présenter ces cas qui dépendent entièrement du contexte applicatif, mais nous traiterons Section 2.4 de l'agrégation dans un entrepôt de données distribué, qui est un cas emblématique rencontré dans le projet GGM.

2.1 Problème générique du déploiement de ressources

Cette gestion incombe soit aux administrateurs des ressources, soit aux services en charge de la gestion de certaines ressources. Le problème du déploiement de ressources n'est pas ici de déployer à proprement parler les ressources en les mettant physiquement à disposition sur des hôtes donnés, mais plutôt de décider où ces ressources doivent être déployées afin d'optimiser un aspect donné.

2.1.1 Présentation du point de vue de l'infrastructure

La Figure 2.1 montre un exemple d'une telle situation : `client1.athome.com` dispose d'une ressource R (une donnée, un service ou autre) qu'il peut déployer sur trois hôtes mis à sa disposition dans le domaine `grid.com` afin de permettre à certains clients du domaine `athome.com` d'y accéder. Il faut noter ici que la dichotomie client/hôte n'est montrée que pour illustration. En réalité, les clients peuvent (et le font probablement) très bien appartenir à la grille, voire être confondus avec des hébergeurs potentiels.

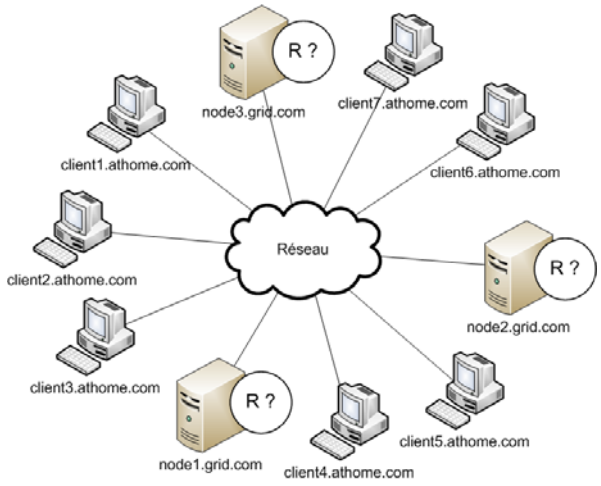


FIG. 2.1 – Le problème du déploiement vu depuis l'infrastructure

2.1.2 Présentation du point de vue de la superstructure GGM

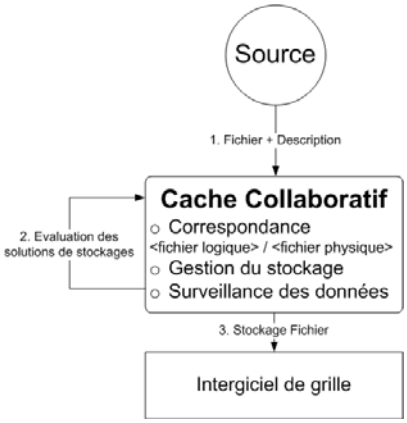


FIG. 2.2 – Le problème du déploiement vu depuis la superstructure GGM

La Figure 2.2 illustre le problème du déploiement d'un fichier vu depuis la superstructure GGM : (1) un client, qui peut être un utilisateur ou un autre composant logiciel de la superstructure, émet une demande de stockage d'un fichier et en fournit une description au service de caches collaboratifs ; (2) ce dernier évalue les différentes solutions de stockage en fonction de critères fournis par l'utilisateur et d'informations disponibles dans le fichier ou dans les informations de surveillance du cache ; (3) le fichier est stocké sur le ou les hôtes sélectionnés.

2.1.3 Analyse

On peut identifier deux problématiques liées à ce déploiement : d'abord la *réplication* et ensuite le *placement*.

La réplication consiste à décider du nombre d'instances, ou réplicas, d'une même ressource devant être déployée sur le réseau. Dans notre illustration, il s'agit de décider si une, deux ou trois instances doivent être déployées. Répliquer une ressource permet :

- d'améliorer sa disponibilité, par exemple en cas de panne de l'hôte d'une instance, une autre est immédiatement disponible.
- de réduire ses temps d'accès, en permettant de *couvrir* tout le réseau afin que tout client aie une instance à proximité.
- d'équilibrer les charges réseau et de calcul, en permettant à plusieurs hôtes de fournir la même ressource simultanément.

En contrepartie, la réplication implique certains aspects négatifs. En effet augmenter le nombre d'instances implique :

- une augmentation des charges disque, qui sont de pures pertes lorsque les ressources ne sont pas utilisées.
- une perte de contrôle sur les ressources, impliquant, entre autre, plus de travail afin d'assurer leur consistance, leur sécurité et leur administration.

Identifier le nombre d'instances optimal pour une ressource donnée est donc un problème de balance entre besoins et contraintes, gains et perte, qui peut s'avérer extrêmement sensible selon les caractéristiques de la ressource. En effet, pour un contenu web de faible taille et nécessitant peu de contrôle, par exemple une simple image non commerciale, une réplication massive sur de nombreux serveurs est tout à fait envisageable. En revanche, pour un contenu vidéo protégé, de grande taille et accessible en flux ce problème devient critique : trop peu d'instances risquent de léser le client final qui n'est pas prêt à accepter une perte de qualité de service pour un contenu payant, alors que de trop nombreuses instances impliquent une utilisation inutile des disques et surtout un risque pour le propriétaire qui ne veut pas que le contenu échappe à son contrôle. Ce problème est d'autant plus difficile qu'il est intimement lié aux demandes utilisateur et revêt donc, dans bien des cas, un caractère dynamique.

Le deuxième problème consiste, une fois que le nombre d'instances est défini, à décider des hôtes où doivent être déployées ces instances. Alors que de bonnes décisions permettent non seulement d'assurer la qualité de service des utilisateurs finaux, mais aussi l'équilibrage des différentes charges, de mauvaises décisions peuvent s'avérer catastrophiques en tous points de vue. Dans notre illustration, il s'agit de décider quels hôtes du domaine `grid.com` sont les plus à même d'héberger les instances.

Ces deux problèmes dépendent de plusieurs aspects :
– l'emplacement des clients et le nombre de leurs requêtes
– les caractéristiques de la ressource (taille, type d'accès, sensibilité, etc.)
– les performances de la portion d'infrastructure matérielle concernée (stockage, calcul, communication, etc.)
– le ou les aspects à optimiser (disponibilité, temps d'exécution, équilibrage des charges, coûts de maintenance, coûts financiers, et.)

La prise en compte de ces différents aspects et leur diversité (non exhaustive puisque tout utilisateur doit pouvoir considérer ses propres objectifs et besoins) rendent ce problème particulièrement difficile face à la contrainte de généricité.

2.2 Problème générique de la sélection de ressources

Une fois les ressources déployées sur l'infrastructure matérielle, il convient lors de chaque utilisation par un client donné, de décider de l'instance la plus à même de répondre à cette utilisation. Une telle décision incombe soit à l'utilisateur final de la ressource, qu'il s'agisse d'un utilisateur humain ou d'une autre ressource, ou encore à un gestionnaire intermédiaire, tel qu'un médiateur ou un courtier.

2.2.1 Présentation du point de vue l'infrastructure

La Figure 2.3 illustre le problème de la sélection avec un client et trois hôtes hébergeant la ressource R demandée. Le problème est le suivant : quelle instance de la ressource R hébergée sur les hôtes du domaine `grid.com`, un utilisateur de `client.athome.com` doit-il utiliser ? Comme pour le problème du déploiement, les deux domaines `grid.com` et `athome.com` ne sont montrés qu'à titre illustratif. De plus, l'utilisateur peut être humain comme une autre ressource active, type service.

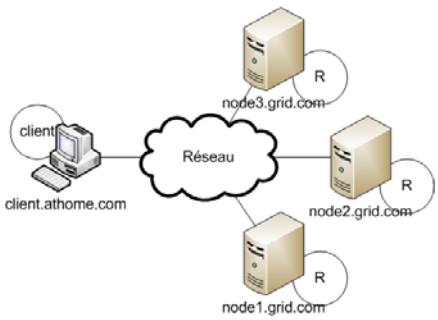


FIG. 2.3 – Le problème de la sélection vu depuis l'infrastructure

2.2.2 Présentation du point de vue de la superstructure GGM

La Figure 2.4 illustre le problème de la sélection vu depuis la superstructure GGM : (1) un client, qui peut être un utilisateur ou un autre composant logiciel de la superstructure, émet une demande pour un fichier en stipulant un nom logique au service de caches collaboratifs ; (2) ce dernier effectue la résolution des noms physiques afin d'établir une liste de tous les réplicas disponibles de ce fichier puis en sélectionne un afin de fournir son identifiant physique au client ; (3) le client accède enfin au fichier demandé.

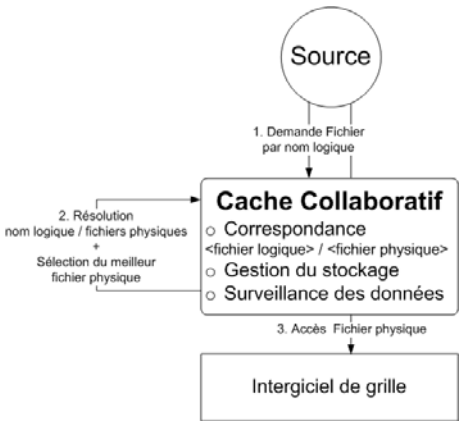


FIG. 2.4 – Le problème de la sélection vu depuis la superstructure GGM

2.2.3 Analyse

Le problème de la sélection est en étroite liaison avec celui du déploiement. Il est également crucial pour les performances de la plateforme : rien ne sert de déployer de façon optimale une ressource, si au moment de l'utilisation un mauvais réplica est systématiquement sélectionné. Par exemple, si le déploiement vise à optimiser les temps d'exécution en couvrant l'ensemble du réseau, mais que les clients ne sélectionnent pas leur réplica le plus proche, alors les performances seront sub-optimales et les charges augmentées et mal équilibrées.

De même que pour le déploiement, le problème de la sélection dépend de trois paramètres :
- l'emplacement du client et des différentes instances de la ressource
- les caractéristiques de la ressource
- les caractéristiques de la portion d'infrastructure matérielle concernée
- le ou les aspects à optimiser

En effet, on pourra décider de privilégier les hôtes les moins chargés afin d'assurer un équilibrage des charges, ou bien de sélectionner les hôtes permettant d'assurer le plus court temps d'exécution, ceux-ci n'étant pas nécessairement les mêmes. De plus, on pourra vouloir prendre également en compte d'autres aspects tels que le coût financier, ou des aspects plus sémantiques tels que la fraîcheur ou l'exactitude. Ainsi, le problème de la sélection est également extrêmement complexe.

2.3 Problème générique de la composition de ressources

On a pu constater avec l'avènement des architectures orientées services que la plupart des tâches relevaient en fait de la composition de plusieurs ressources. En effet, un service étant une application logicielle en charge d'une tâche souvent bien précise et unitaire, les tâches utilisateur correspondent bien souvent à l'utilisation successive de plusieurs ressources assurant chacune des sous-tâches nécessaires. Cela est communément appelé la composition de ressources. Cet aspect a fait l'objet de la thèse de Girma Berhe [Berhe 06] soutenue dans notre équipe.

Le problème de la composition correspond en réalité à deux problèmes. Le premier, appelé ordonnancement, consiste à choisir dans quel ordre les ressources doivent être utilisées lorsque des sous-

tâches sont interchangeables. Le second consiste à sélectionner pour chaque ressource quelle instance doit être utilisée. Dans certains cas, il est possible de résoudre ces deux problèmes indépendamment : on commence par ordonnancer les tâches en fonction d'un graphe sémantique de composition, puis on sélectionne les ressources en fonction de cet ordonnancement. Mais cette approche n'est pas toujours optimale, comme nous allons le voir ci-après.

2.3.1 Présentation du point de vue sémantique

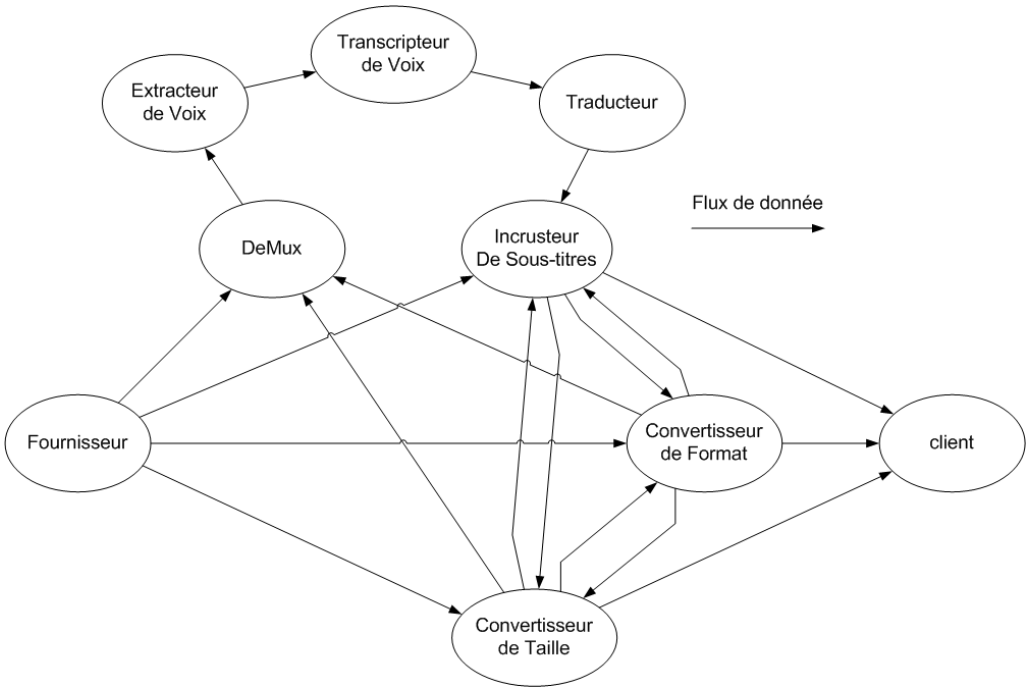


Fig. 2.5 – Exemple de chemins logiques d'adaptation de contenus multimédia.

Fig. 2.6 – Le problème de la sélection du point de vue sémantique

La Figure 2.6 montre un exemple de composition de services permettant l'adaptation d'un contenu multimédia vidéo qui doit être changé de taille, de format et être sous-titré grâce à l'extraction du flux audio, sa transcription en texte, sa traduction puis son incrustation dans le flux vidéo. On parle ici de chemin logique d'adaptation, car on décrit les opérations devant être appliquées au contenu ainsi que leur ordre, sans prendre en compte les services physiques devant être effectivement invoqués pour effectuer ces opérations.

La description des différents services impliqués est :

Fournisseur fournit le contenu vidéo initial

Convertisseur de taille change la taille du contenu vidéo reçu en entrée

Convertisseur de format change le format du contenu vidéo reçu en entrée

DeMux extrait le son du contenu vidéo reçu en entrée

Extracteur de voix extrait les voix du contenu audio reçu en entrée

Transcripteur de voix produit un fichier texte synchronisé transcrivant le contenu audio reçu en entrée

Traducteur traduit un fichier texte reçu en entrée

Incrusteur de sous-titres incruste un fichier texte synchronisé dans un contenu vidéo reçus en entrée

client consomme l'adaptation finale du contenu

Les différents contenus échangés lors de cette composition sont les suivants :

V : Le contenu vidéo initial

Vf : Le contenu vidéo après conversion de format

Vt : Le contenu vidéo après changement de taille

S : Le flux audio du contenu vidéo

S' : Le contenu audio des voix du flux audio

S" : Les sous-titres dans la langue initiale

s : Les sous-titres dans la langue cible

Vs : Le contenu vidéo sous-titré

Nous notons les différents contenus comme des combinaisons des différentes adaptations et les opérations notées entre crochet sont optionnelles. Par exemple Vfts représente le contenu vidéo retaillé, convertit et sous-titré, alors que V[f]s représente deux contenus : soit le contenu vidéo sous-titré et converti, soit le contenu vidéo seulement sous-titré.

2.3.2 Présentation du point de vue l'infrastructure

La Figure 2.7 illustre le problème de la composition de ressource vu depuis l'infrastructure matérielle. Les différents services devant être invoqués entre le fournisseur et le client sont distribués et répliqués sur plusieurs hôtes. Il faut donc prendre en compte cette distribution afin de sélectionner chacun des réplicas en fonction de leur place dans le graphe sémantique.

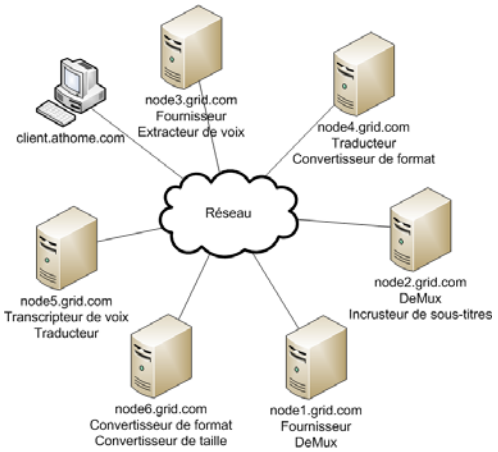


FIG. 2.7 – Le problème de la composition vu depuis l'infrastructure

2.3.3 Présentation du point de vue de la superstructure GGM

La Figure 2.8 illustre le problème de la composition de ressource vu depuis la superstructure GGM : (1) un client, qui peut être un utilisateur ou un autre composant logiciel de la superstructure, émet une demande pour un contenu adapté au service de caches collaboratifs ; (2) ce dernier évalue les différentes solutions d'adaptation en construisant le graphe sémantique d'adaptation et en le confrontant aux services indexés sur la grille ; (4) puis il invoque chacun des services d'adaptation nécessaires en fonction de la solution retenue, soit en connectant directement les services successifs, soit en stockant les résultats intermédiaires selon sa propre stratégie ; (4) le cache stocke le résultat final et en fournit son identifiant physique au client ; (5) le client accède enfin au contenu adapté demandé.

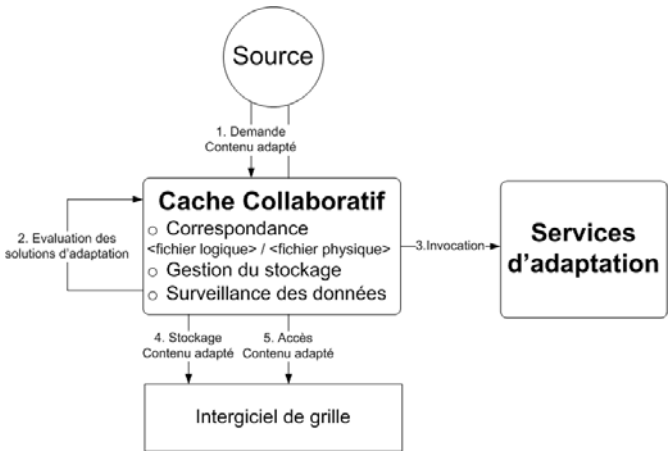


FIG. 2.8 – Le problème de la composition vu depuis la superstructure GGM

2.3.4 Analyse

Une première remarque concerne la parallélisation de la tâche de création des sous-titres qui est possible à trois niveaux différents : parallélisation au plus tôt (le contenu vidéo est envoyé simultanément au service *DeMux* et aux convertisseurs), parallélisation à moitié (le contenu vidéo est envoyé au service *DeMux* après être passé par un des convertisseurs) ou pas de parallélisation (le contenu vidéo est envoyé au service *DeMux* après être passé par les deux convertisseurs). Pour chacun de ces trois cas, les deux convertisseurs de taille et de format sont interchangeables.

On peut donc compter six chemins possibles entre le fournisseur et le consommateur. On peut intuitivement penser que la parallélisation est souhaitable pour améliorer les performances. En réalité, la distribution des ressources rend les choses plus complexes. En effet, si les conversions de tailles et de formats produisent des contenus de plus faibles tailles, et si l'infrastructure matérielle dispose de faibles capacités de communication ou bien que ces communications sont coûteuses (financièrement ou en cas d'utilisation intensive), alors il peut être plus judicieux de commencer par convertir le contenu original avant d'entreprendre son sous-titrage, afin de limiter les coûts de communication.

On peut également remarquer qu'une adaptation paraissant anodine, par exemple l'extraction du flux audio face aux processus de transcription et de traduction, peut en réalité s'avérer être une opération critique si elle est hébergée uniquement sur un hôte présentant des performances très basses.

De plus, les chemins d'adaptation peuvent impliquer d'autres aspects sémantiques tels que la qualité du contenu final produit. En effet, selon l'ordre des opérations, on pourra obtenir un contenu final différent, par exemple en convertissant le contenu original, on perdra en qualité du flux audio impliquant une baisse de la qualité des sous-titres produits, ce qui pourrait être évité en extrayant immédiatement le flux audio du contenu original. Or l'utilisateur pourra vouloir trouver un équilibre entre la qualité et le temps d'exécution, équilibre qui dépend des performances de l'infrastructure matérielle.

Ajoutons à cela que chaque opération logique peut être prise en charge par différents services physiques possiblement répliqués sur le réseau, et nous obtenons un nombre rapidement important de chemins physiques correspondant aux chemins logiques. Par exemple si nous supposons que deux services physiques peuvent fournir chaque opération logique, compte tenu des six chemins logiques possibles, nous obtenons $6 \times 2^8 = 1536$ chemins physiques possibles. Alors que l'exemple pris n'est pas très compliqué, on peut déjà se rendre compte que ce genre de problème n'est pas gérable par des décisions intuitives.

Pour conclure sur cet exemple, nous pouvons donc noter que de nombreux aspects sont impliqués dans la résolution d'un tel problème.
- l'emplacement des instances des différentes ressources à composer
- les caractéristiques des données échangées
- les caractéristiques de la portion d'infrastructure matérielle concernée
- le ou les aspects à optimiser

Lesquels aspects relèvent tantôt de caractéristiques des ressources, des choix et besoins utilisateur et des capacités de l'infrastructure matérielle. Ils obligent donc l'utilisateur à faire face à un problème d'équilibre très sensible et complexe, en particulier compte-tenu du nombre de solutions possibles qui s'offrent à lui, alors qu'il n'a ni les connaissances, ni les compétences pour trancher.

2.4 Problème spécifique de l'agrégation dans un entrepôt de données distribué

Les trois exemples présentés précédemment représentent les trois problèmes classiques liés à la répartition de ressources. En réalité, ils ne représentent que la partie commune à (pratiquement) tous les utilisateurs. Or dans la plupart des applications de grilles, on rencontre des problèmes très spécifiques qui ne rentrent pas exactement dans les trois cas sus-cités (malgré certains points communs).

Nous avons rencontré de nombreux exemples lors du développement de la plateforme logicielle GGM décrite Section 1.2.1. Le plus emblématique concerne l'entrepôt de données distribués. De façon grossière, un entrepôt de données est une base de données dont le schéma prévoit un niveau d'agrégation pour chaque donnée : les données au niveau le plus détaillé proviennent généralement directement des sources, alors que les données de plus haut niveau d'agrégation sont des statistiques calculées sur l'ensemble des données de niveau inférieur. L'exemple classique est celui des ventes dans une chaîne de magasins : au plus bas niveau, on trouve chacune des ventes unitaires par lieu et date, puis au niveau supérieur on trouve la somme des ventes par région ou par jour, et au niveau le plus haut on trouve le total des ventes depuis leur début et sur l'ensemble des magasins.

Une donnée à un certain niveau d'agrégation est appelée agrégat. Lorsqu'un utilisateur demande

un agrégat, l'entrepôt peut soit lui fournir directement s'il est déjà *matérialisé*, soit le calculer à partir des agrégats matérialisés de niveau inférieur. Une hiérarchie simple est présentée Figure 2.9 : les agrégats de niveau 3 (par exemple les ventes d'une année) sont calculés à partir de tous les agrégats de niveau 2 (les ventes des deux semestres de l'année) qui, eux-mêmes, le sont à partir des agrégats de niveau 1 (les ventes des trimestres de l'année).

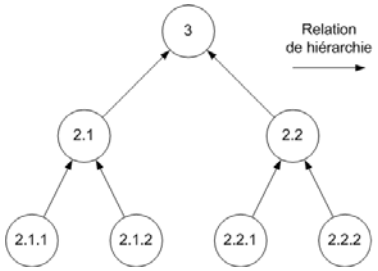


FIG. 2.9 – Exemple de hiérarchie d'agrégation d'entrepôt de donnée

2.4.1 Présentation du point de vue de l'infrastructure

La Figure 2.10 montre un exemple d'une telle situation : `client1.athome.com` veut recevoir l'agrégat de niveau 3. Certains hôtes du domaine `grid.com` hébergent une instance du service d'entrepôt distribué (noté DDW pour *Distributed DataWharehouse*) lequel est capable d'effectuer des agrégations. Les différents agrégats sont stockés par le service de caches collaboratifs disponibles sur d'autres hôtes. Ces agrégats sont inclus dans des fichiers qui peuvent contenir plusieurs agrégats non nécessairement continus. Ces fichiers peuvent être répliqués et les agrégats peuvent être inclus dans plusieurs fichiers différents. Ainsi, un même agrégat peut être inclus dans deux fichiers de tailles très différentes. Or, la structure de ces fichiers impose que leur totalité soit récupérée avant d'extraire un agrégat donné.

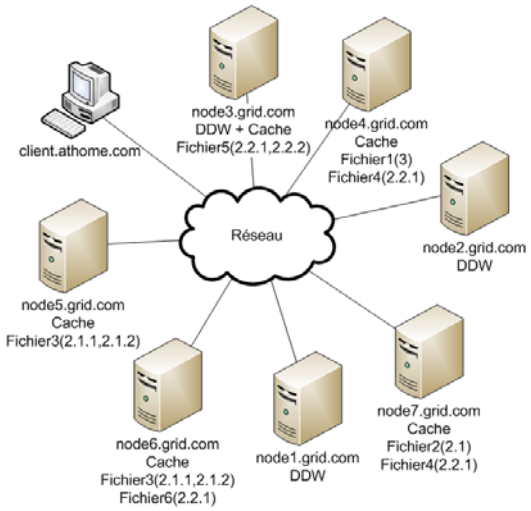


FIG. 2.10 – Le problème spécifique de l'agrégation vu depuis l'infrastructure

L'objectif du problème spécifique de l'agrégation est de décider quelle(s) instance(s) du service d'entrepôt distribué et quel(s) fichier(s) utiliser pour fournir l'agrégat demandé par l'utilisateur. Une telle décision implique des choix du type : vaut-il mieux récupérer directement un agrégat inclus dans

un fichier de grande taille sur un hôte proche ou bien dans un fichier de faible taille mais sur un hôte plus distant, ou encore vaut-il mieux recalculer cet agrégat à partir des agrégats de niveaux inférieurs en fonction des fichiers dans lesquels ils sont inclus.

2.4.2 Présentation du point de vue de la superstructure GGM

La Figure 2.11 illustre le problème spécifique de l'agrégation vu depuis la superstructure GGM : (1) un client, qui peut être un utilisateur ou un autre composant logiciel de la superstructure, émet une requête d'entrepôt dont le résultat est un agrégat ; (2) ce dernier établit la liste des différentes solutions d'agrégation en fonction de la hiérarchie présentée précédemment et de l'indexation des agrégats dans les fichiers logiques ; (3) l'emplacement des différents fichiers logiques est demandé au cache ; (4) ce dernier informe l'entrepôt des emplacements physiques et de la taille des fichiers demandés ; (5) l'entrepôt évalue le coût de chacune des solutions d'agrégation selon ses modèles de coûts propres ; (6) selon la solution retenue, il récupère les fichiers physiques ; (7) puis en extrait les agrégats nécessaires et procède aux agrégations demandée ; (8) enfin il renvoie l'agrégat demandé à l'utilisateur.

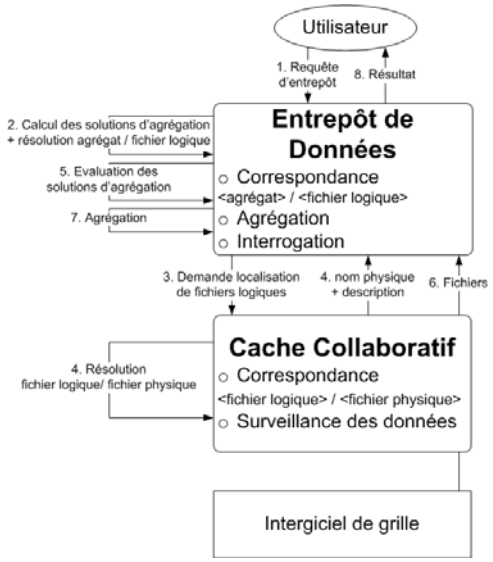


Fig. 2.11 – Le problème spécifique de l'agrégation vu depuis la superstructure GGM

On peut noter une difficulté annexe impliquée par la dissociation des informations impliquée dans la décision. En effet, la décision ne peut être prise qu'en regroupant les informations détenues par l'entrepôt distribué et le service caches collaboratifs. Cela implique donc un fort couplage entre ces deux composants et des problèmes de synchronisation. Par exemple : que se passe-t-il si un fichier ait supprimé après que l'entrepôt est construit son plan d'agrégation ? De plus, seule une étroite collaboration permet à ces deux services d'implémenter des stratégies complémentaires en ce qui concerne le déploiement des fichiers et leur sélection. Par exemple, l'entrepôt peut décider de construire un fichier contenant de très nombreux agrégats compte-tenu de leurs régulières utilisations conjointes, alors que le cache peut décider de découper ce fichier afin de le stocker plus facilement selon l'espace disque disponible sur la grille. On voit ici la mise en œuvre de deux stratégies concurrentes, ce qui risque fort de s'avérer néfaste pour les performances.

2.4.3 Analyse

Dans un entrepôt classique hébergé sur une unique machine, le problème de décider quelles données doivent être matérialisées est déjà assez complexe et relève d'un problème d'équilibre entre charge disque, charge de calcul et temps de réponse. Avec la distribution de l'entrepôt sur plusieurs hôtes différents, non seulement ce problème est exacerbé, mais il oblige également à gérer la réplication de certaines données afin d'assurer leur disponibilité, ce qui renvoie aux problématiques présentées Section 2.1.

De plus lorsqu'une données et demandée, il ne s'agit plus simplement de savoir si elle est matérialisée, car elle peut l'être mais sur un hôte distant dépourvu d'une bonne capacité de communication, alors que les données des niveaux inférieurs permettant son calcul sont sur des hôtes proches, disposant d'une grande capacité de calcul et de communication : parfois l'agrégation sera plus efficace que le simple transfert, ce qui n'était pas le cas dans les entrepôts classiques. Cette agrégation pourra se faire à partir de différentes données répliquées de niveaux différents, présentant ainsi un grand nombre de possibilités. On voit ici apparaître un problème de déploiement/sélection complexe et spécifique aux entrepôts de données distribués.

Qui plus est, la notion de *fraîcheur* prends ici une importance cruciale : lorsqu'un agrégat est matérialisé, il doit être maintenu lors de la mise à jour des agrégats de plus bas niveau. Or ce processus de maintenance peut s'avérer extrêmement consommateur de ressource et peut prendre de longues heures. Un premier impact est que la stratégie de réplication/placement des agrégations s'en trouve complexifiée et doit être élaborée en conséquence. Le second impact est que cette maintenance implique un décalage de fraîcheur entre données agrégées et données détaillées qui doit être pris en compte lors d'une requête : un utilisateur pourra préférer obtenir une réponse rapide au détriment de la fraîcheur, ou le contraire. Enfin, suivant la sensibilité de la demande de certaines agrégations, le concepteur de l'entrepôt pourra délibérément décider de les maintenir plus ou moins fréquemment. On voit ici apparaître un problème complètement nouveau et spécifique à ce contexte.

En résumé, dans cet entrepôt distribué on retrouve les problèmes classiques de déploiement, sélection, composition de ressources, mais dans un contexte particulier soumis à des contraintes et problématiques spécifiques. De plus, on voit apparaître le nouveau problème de la gestion de la maintenance des données agrégées, toujours en relation avec les problématiques de répartition de ressources, mais sans solution disponible dans la littérature. Enfin, on remarque encore que les différents aspects relèvent tour à tour des choix du concepteur de l'entrepôt, de son utilisateur, des caractéristiques des données et des performances de l'infrastructure matérielle.

2.5 Synthèse de la problématique et des motivations

Dans un premier temps, nous avons présenté trois environnements distribués dont les caractéristiques et objectifs communs impliquent qu'ils seront potentiellement amenés à fusionner :

- les grilles et architecture orientée services au travers du projet GGM.
- les grilles pervasives au travers d'un scénario de réponse à une situation d'urgence.
- les organisations virtuelles inter-grilles au travers d'un scénario médical.

Le point commun principal de ces environnements est qu'ils impliquent de nombreuses entités dif-

férentes impliquées dans des problématiques diverses impliquant hétérogénéité et dynamicité. Nous avons identifié les problèmes liés de la répartition des ressources en trois types génériques : déploiement, sélection et composition ; auxquels s'ajoute l'ensemble, malheureusement non fini, des problèmes spécifiques dont nous avons présenté un exemple avec l'agrégation dans un entrepôt de données distribué.

Dans les environnements de grilles, les problèmes liés à la répartition des ressources sont extrêmement importants, car :

- ils interviennent quotidiennement pour toutes les entités de l'environnement distribué, qu'elles soient humaines, logicielles ou matérielles.
- ils conditionnent les performances globales des plateformes et particulières de chaque tâche : de bonnes décisions permettent d'équilibrer les charges de l'infrastructure, de garantir son exploitation au mieux et de garantir que chaque tâche sera exécutée dans les meilleures conditions... Alors que de mauvaises décisions ont des conséquences exactement contraires.
- ils conditionnent la difficulté du travail des décisionnaires : des problèmes difficiles sans support de résolutions obligent les développeurs à s'éloigner de leurs problématiques et préoccupations principales pour concevoir des méthodes de résolution *ad hoc*, limitant ainsi leur retours sur investissements.
- ils conditionnent également la pérennité des plateformes : un ensemble de solutions *ad hoc* à chacun des problèmes rencontrés est difficile à réutiliser, puisqu'à chaque changement de la nature, des besoins ou des objectifs de la plateforme ils doivent être adaptés, voire repensés. En revanche, un système de résolution générique et adaptable permet à la plateforme de supporter ces changements automatiquement et de façon transparente.

En outre, les problèmes liés à la répartition des ressources sur grille sont particulièrement difficiles car :

- ils sont soumis à des besoins et contraintes extrêmement diverses, et parfois même antagonistes, dans lesquels interviennent de nombreux aspects :
  - la nature du gestionnaire : utilisateur final, concepteur, développeur, administrateur, applications ou services...
  - la nature de la ressource gérée : simple fichier, service, flux de données, données d'entrepôt...
  - les performances et la topologie de l'environnement : puissance de calcul, de communication, de stockage, localisation des différentes ressources impliquées...
  - les objectifs à atteindre : équilibrage de charge, temps d'exécution, qualité des données, coûts financiers...

  Leur résolution ne peut donc être satisfaisante que par l'identification d'équilibres non triviaux dans une approche globale.
- les informations impliquées dans la résolution de ces problèmes sont fournis par différentes entités, et donc difficiles à réunir :
  - acteurs :

    Les administrateurs doivent décider des aspects à optimiser lors de la gestion des ressources dont ils ont la charge.

    Les concepteurs et développeurs doivent faire des choix en rapport aux spécificités de leurs développements.

    Les utilisateurs finaux, ou les composants logiciels en charge de leurs décisions, doivent déterminer des aspects à optimiser lors de l'utilisation des ressources.
  - superstructure logicielle :

Les ressources possèdent différentes caractéristiques telles que la taille pour une donnée ou les calculs nécessaires à l'exécution pour une application ou un service.
  - infrastructure matérielle :
    Les différents hôtes proposent des performances différentes en terme de puissance de calcul, de mémorisation ou de stockage.
    L'infrastructure propose également des capacités de communication différentes en terme de latence, bande-passante ou taux d'erreurs.

- tant les utilisateurs, que l'infrastructure matérielle et la superstructure logicielle sont caractérisés par une haute hétérogénéité et une dynamicité omniprésente à court et long terme :
  - l'état des machines évolue dans le temps, étant soumis à des pannes et des charges variée.
  - la topologie de l'infrastructure évolue également suite à des ajouts ou suppression de matériels.
  - les composants logiciels sont fréquemment mis à jour, ajoutés ou modifiés.
  - les usages évoluent naturellement au cours de l'exploitation des plateformes.
  - ...
- cette gestion doit revêtir un caractère générique afin d'être utilisable dans tous les cas qui peuvent se présenter, et aussi afin de pouvoir supporter un changement de la nature même de l'architecture, lors d'évolutions, d'ouvertures ou de fusions.

Enfin pour conclure, nous pouvons identifier les contraintes suivantes pour les solutions génériques de résolution des problèmes liés à la répartition des ressources en environnement de grille :

généricité : pouvoir gérer un large panel de problèmes différents, y compris complètement spécifiques

adaptabilité : prise en compte de différents objectifs déclarés par l'utilisateur, y compris sémantiques ou financiers

adaptativité : prise en compte automatique des caractéristiques de la superstructure

hétérogénéité : prise en compte de matériel de capacités très différentes sur l'infrastructure

dynamicité : prise en compte de fluctuation dans les performances sur les infrastructures partagées

évolutivité : prise en compte de modifications profondes des plateformes suite à l'apparition de nouveaux besoins/objectifs

utilisabilité : facilement compréhensible et utilisable par toute classe d'utilisateur

profitabilité : présenter un rapport coût/profit d'utilisation satisfaisant

3

Travaux Connexes

L'établissement d'un état de l'art exhaustif est bien connu pour ne pas être une tâche aisée, voire même impossible. La démultiplication des équipes de recherches, des laboratoires et des projets a engendré dans les dernières années un tel volume de productions qu'il est de plus en plus difficile de clairement se situer et d'identifier les chercheurs travaillant sur les mêmes domaines.

Notre objectif est de concevoir une solution générique capable de gérer un large panel de problèmes de distribution de ressources dans différents environnements de grille. Il se trouve donc à la croisée de plusieurs domaines scientifiques.

Malheureusement, beaucoup de travaux directement liés à la gestion de la répartition des ressources, tels que les algorithmes de réplication de données, ne considèrent généralement pas les problématiques liées aux grilles. L'environnement Internet, les contenus multimédias ou les serveurs web (pour les applications) sont leurs cibles de prédilection. Or, on ne retrouve pas dans cet environnement exactement les mêmes problématiques, ce qui rend les travaux rapidement inutiles lorsqu'on les étudie de plus près. De plus, ces travaux s'intéressent la plupart du temps à un seul problème, par exemple la sélection de serveurs web, ou encore le déploiement de réplicas de données. D'après ce que nous savons, aucun travaux dans ce domaine ne présente un caractère assez générique permettant de gérer tout problème de distribution de ressources. Leur étude ne concerne donc pas directement les travaux de cette thèse.

Nous nous sommes donc concentrés sur les solutions spécifiques aux environnements de grilles. Il faut rappeler que le défi principal impliqué dans notre objectif est de prendre en compte au sein d'une même solution les caractéristiques de la superstructure et les performances de l'infrastructure. Notre état de l'art se construit donc selon la dualité couches superstructure / infrastructure.

Dans un premier temps, en Section 3.1, nous avons décidé d'étudier un échantillon représentatif de solutions développées au niveau de la superstructure. Ces solutions se basent en général sur un

problème extrêmement précis et identifié dès le départ comme étant primordial pour la superstructure cible. Le problème spécifique, qui est probablement le plus fréquent, et l'un des plus difficiles, concerne le déploiement de ressources en environnement de grille.

Dans un second temps, nous nous sommes intéressés en Section 3.2 aux solutions se situant à un niveau intermédiaire entre la superstructure et l'infrastructure. Ces dernières ne prennent pas nécessairement pour objectif un problème de distribution spécifique bien défini, mais s'intéressent plutôt à la conciliation des caractéristiques de la superstructure et des performances de l'infrastructure en proposant des prédictions de performances. En effet, la prédiction représente une des approches possibles pour gérer la distribution de ressources : être capable de prédire le temps d'exécution d'une tâche en fonction des hôtes candidats permet d'optimiser les performances pour les problèmes de sélection. Et, bien que ce soit rarement abordé, on peut imaginer que de telles prédictions peuvent s'avérer utiles dans d'autres problèmes connexes.

Dans un troisième temps, nous avons étudié en Section 3.3.2 les solutions se concentrant sur l'infrastructure. Plutôt que de résoudre un quelconque problème, ces solutions proposent de collecter et fournir des informations précises sur l'architecture matérielle, avec pour objectif principal la scalabilité, la précision des mesures et/ou l'utilisabilité. Á la charge ensuite de l'utilisateur d'exploiter des informations pour résoudre ses propres problèmes.

3.1 Les solutions aux problèmes spécifiques de la répartition

Parmi les problèmes présentés Chapitre 2, nous nous intéressons plus particulièrement au problème du déploiement de services. En effet, le domaine scientifique est trop vaste pour être complètement couvert. Comme nous nous intéressons ici aux solutions spécifiques, il faudrait présenter les solutions pour chacun des problèmes (déploiement, sélection, composition et spécifique) et ce pour chacune des ressources logicielles des grilles (fichier, contenu multimédia, application, base de données, table de base de donnée, service, etc.). En revanche, les services étant des ressources logicielles relativement nouvelles et surtout caractérisées par une grande généricité, les solutions relatives à leur déploiement sont particulièrement intéressantes. D'autant plus que les services, ou plus simplement les applications, mettent en jeu les capacités de calcul autant que celles de communications, rendant leur gestion plus complexes.

De nombreux outils permettent de gérer le déploiement de services sur un réseau. Le déploiement de services est constitué de plusieurs étapes : (1) la découverte des ressources, qui consiste à lister l'ensemble des hôtes disponibles capables d'héberger le service, que nous désignons par hébergeurs candidats ; (2) la génération du plan de déploiement, qui consiste, en fonction des besoins en calcul et communication, à attribuer chacune des instances de service à un hébergeur candidat ; (3) l'installation des services sur leur hôte d'hébergement ; (4) la configuration ; et enfin (5) le lancement.

Ces différentes étapes étant laborieuses, de nombreux outils ont été développés afin d'assurer leur automatisation. Il faut noter que la partie impliquant à proprement dit une gestion de la répartition des ressources est l'étape (2). Nous allons donc concentrer nos études sur les stratégies d'établissement des plans de déploiement, en tentant de les mettre en perspective avec les aspects d'importance présentés dans l'introduction : respect de l'hétérogénéité, de la dynamicité et de l'évolutivité des ressources et de l'utilisabilité.

3.1.1 ADAGE

Présentation

Automatic Deployment of Applications in Grid Environment est un prototype développé par l'équipe PARIS de l'IRISA Rennes. Son but initial est d'étudier les problèmes liés au déploiement d'applications multi-intergiciel. ADAGE est conçu pour automatiser le déploiement d'applications dans une grille. Sa contribution originale est l'utilisation d'un modèle de description générique appelé GADe (*Generic Application Description*) afin d'assurer une transparence lors de l'utilisation de plusieurs intergiciels.

Afin de réaliser le déploiement d'une application, ADAGE nécessite trois informations : une description de l'application, les références vers un service de surveillance et un fichier de paramètres de contrôle. Le service de surveillance est utilisé afin de découvrir les hôtes candidats. Ce dernier peut être soit MDS2 (voir Section 3.3.2) soit un fichier XML. Les paramètres de contrôle permettent de déclarer des contraintes relatives au déploiement, telles que les latences et bandes-passantes requises vers certains hôtes, ainsi que la politique de déploiement à utiliser, sous forme d'ordonnanceurs implémentés en plug-ins.

Discussion

On peut énoncer deux critiques en ce qui concerne la gestion de la distribution des ressources dans ADAGE. Premièrement, bien qu'extensible puisque sous forme de plug-in, les seuls stratégies de déploiement disponibles à l'heure actuelle sont *aléatoire* et *tourniquet*, ce qui est trop basique pour supporter des applications complexes ou des besoins évolués. Deuxièmement, le fait que seules des contraintes puissent être exprimées est très... contraignant. En effet, on ne peut exprimer le souhait de, par exemple, réduire autant que possible la latence vers un hôte donné tout en augmentant la bande-passante vers un autre, ce qui risque d'empêcher l'obtention du déploiement optimal. De plus, exprimer des contraintes trop importantes risque de réduire l'espace des solutions candidates à néant, si aucun hôte ne leur correspond. D'un autre côté, exprimer des contraintes trop lâches risque d'accroître cet espace au point que le déploiement sera forcément sub-optimal (puisqu'on ne peut déclarer des aspects à optimiser). Ceci laisse présager non seulement d'un travail utilisateur difficile et critique, mais surtout de l'impossibilité de réutiliser les paramètres de contrôle sur différentes infrastructures et d'un mauvais support à leurs changements, qu'ils soient durables ou dus à la dynamicité. En effet, des paramètres de contrôle optimaux pour une grille très haute performance feront apparaître des besoins importants, qui ne peuvent être satsifaits dans une grille basse performance.

En conclusion, on peut dire que ADAGE se concentre sur l'automatisation des mécanismes impliqués dans le déploiement des applications, mais ne dispose pas de l'expressivité nécessaire, ni de politiques suffisamment évoluées pour pouvoir optimiser ce déploiement en toutes circonstances, surtout lorsque l'environnement est hétérogène, dynamique ou évolutif.

3.1.2 GLARE

Présentation

GLARE - Grid-Level Activity REgistration est un *framework* de gestion de composants logiciels développé à l'University of Innsbruck, Autriche. Ses fonctionnalités comprennent l'enregistrement dynamique, le déploiement automatique et la fourniture à la demande de composants logicielles, et ce dans l'objectif d'affranchir les développeurs des considérations relatives à l'infrastructure matérielle et d'optimiser l'utilisation de cette dernière. GLARE se focalise sur l'exécution de *workflows* impliquant plusieurs ressources logicielles, comme c'est le cas dans les SOA (voir Section 1.2).

L'architecture de GLARE comporte trois composants principaux : *Activity Type Registration - ATR*, *Activity Deployment Registry - ADR* et *Registration, Deployment, and Monitoring - RDM*. Les deux registres sont des *WS-Resssources*[1]. Les *Activity Types* correspondent aux abstractions de ressources logicielles et sont organisé de façon hiérarchique. Ils sont définis en terme de domaines, fonctions, arguments et banc d'essai. Ils peuvent pointer de plus vers un fichier décrivant les différentes étapes impliquées dnas le déploiement automatique. Les *Activity Deployment* correspondent à des instanciations de ressources logicielles sur des hôtes donnés. Ils sont définis par le nom d'un exécutable ou d'un service et sont associés à un *Activity Type*. Le composant *Registration, Deployment, and Monitoring* est la principale interface avec les utilisateurs. Il gére les requêtes clients, la surveillance des ressources logicielles et procède aux déploiements. L'ensemble de ces composants est organisé dans un réseau pair à pair conçu pour supporter une grande dynamicité.

Lorsqu'un client émet une requête pour disposer d'un service à un RDM, celui-ci contacte l'ATR à la recherche d'un type correspondant au service demandé. Si ce type n'est pas trouvé, il doit être ajouté par l'utilisateur avant de procéder au déploiement. Une fois le déploiement effectué automatiquement, l'ADR est mis à jour. La sélection des hôtes pour procéder au déploiement est faite en respectant des contraintes données par l'utilisateur, dont les exemples sont : la plateforme (« Intel »), le système d'exploitation (« Linux ») et l'architecture (« 32bit »).

Discussion

L'objectif de supporter les administrateurs des tâches liées au déploiement de ressources logicielles face à une grande dynamicité des ressources logicielles et matérielles, trouve d'intéressantes solutions dans GLARE. Celui-ci fournit une abstraction complexe des ressources logicielles et les organise en hiérarchie afin de simplifier les tâches des utilisateurs lors du déploiement ou de l'utilisation de ressources logicielles. En revanche, le système de sélection des hôtes pour héberger une ressource logicielle donnée est assez limité puisqu'uniquement basé sur des contraintes. La discussion rejoint donc les remarques formulées pour ADAGE.

[1] service web avec état

3.1.3 Sekitei

Présentation

Comme présenté dans les sections précédentes, les solutions de déploiement de ressources logicielles se focalisent sur les mécanismes liés aux déploiements et sur leur automatisation. En revanche, les décisions liées à la gestion de la répartition des ressources représentent une lacune souvent comblée par des solutions basiques. Ainsi, plusieurs algorithmes ont été proposés pour répondre au problème de la construction du plan de déploiement qui conditionne le placement de ressources logicielles sur des hôtes : composants sur les nœuds comme GARA (Globus Architecture for Reservation and Allocation) [Foster 99], PSF [Ivan 02], Conductor [Reiher 00], Ninja [Gribble 01], CANS [Fu 00], et Sekitei. Ce dernier propose un modèle global regroupant les autres algorithmes, nous l'avons donc étudié plus en détail.

Présenté dans [Kichkaylo 04] par Kichkaylo et Karamcheti, Sekitei est un algorithme d'intelligence artificielle répondant au problème de placement de composants (*CPP - Component Placement Problem*). Il est implémenté sous forme d'un module logiciel pouvant être utilisé dans les solutions de déploiement automatique de ressources logicielles telles que ADAGE ou GLARE afin de combler leurs lacunes dans ce domaine.

Le problème CPP vise en particulier les composants ayant des contraintes relatives aux interfaces requises ou implémentées, lesquelles doivent être déclarées par l'utilisateur. Son objectif est donc de gérer le déploiement de plusieurs composants logiciels interdépendants.

Le but de l'algorithme est de placer un composant donné sur un hôte donné. À partir de ce but, Sekitei décide du placement des autres composants en recherchant la configuration minimale. Ce problème correspond au problème de planification (*AI Planning Problem*) avec prise en compte de la disponibilité des ressources : la disponibilité des interfaces et le placement des composants sur les hôtes constitue l'état du système, qui est représenté par des variables booléennes ; des valeurs réelles servent à représenter les capacités et l'état des hôtes et des communications ; des opérateurs expriment le placement d'un composant sur un hôte et la mise à disposition d'une interface sur le réseau. Ainsi le but de CPP est transformé en un but propositionnel.

Le problème est traité en deux étapes : régression et progression. La régression consiste à déterminer le plus petit ensemble d'opérateurs pertinents en prenant en compte leurs préconditions et effets logiques, en d'autres mots, à éliminer les hôtes candidats non pertinents afin de limiter l'espace de déploiement sans prendre en compte les capacités de l'infrastructure. La phase de progression recherche ensuite les solutions dans cet espace : elle vérifie la possibilité d'atteindre le but du CPP en prenant en compte les contraintes exprimées sur l'infrastructure. Dans le cas où aucune solution n'est trouvée, la phase de régression est réexécutée afin d'élargir l'espace de déploiement.

Discussion

Sekitei propose une solution au problème de placement de ressources logicielles sur une grille. Développé en tant que module, il peut aisément être intégré dans les solutions de déploiement afin de leur fournir des décisions plus évoluées. Il prend en compte l'interdépendance entre les ressources

logicielles et les performances de l'infrastructure matérielle.

Un premier problème qui se présente est la contrainte d'avoir un but CPP. Il faut donc commencer par définir l'emplacement d'un composant logiciel afin que Sekitei puisse définir l'emplacement des autres. Bien que cela se défende pour les superstructures présentant un composant principal et plusieurs composants auxiliaires, ce n'est pas le cas de la majorité des superstructures de grilles où plusieurs composants ont une importance primordiale. Par exemple, les composants logiciels de la superstructure GGM ont tous une importance équivalente et sont de plus tous distribués. Il est alors impossible de décider du placement d'une instance d'un des services afin de placer les autres instances.

De plus, l'algorithme proposé est assez complexe, donc difficile à adapter et présente certaines lacunes. Il est par exemple impossible de définir des poids ou priorités pour les composants logiciels afin de les attribuer en priorité aux ressources les plus puissantes. Il ne prend pas correctement en compte la dynamicité de l'infrastructure matérielle et n'est pas utilisable pour le placement dynamique de ressources. Il est aussi impossible de déclarer des objectifs spécifiques (par exemple les coûts financier d'utilisation des ressources), ce qui limite son utilisation dans certains cas.

En conclusion, Sekitei répond aux lacunes des outils de déploiement de ressources logicielles en fournissant des décisions de placement, mais manque d'adaptatabilité et ne peut donc être utilisé dans tous les cas, en particulier lors de besoins spécifiques.

3.1.4 Conclusion

On peut noter enfin que de nombreux outils de déploiement ont été développés pour les grilles. On peut par exemple noter ADAGE [Lacour 05] et GLARE [Siddiqui 05], mais aussi DistAnt [Goscinski 05], Weevil [Wang 05], Jade [Bouchenak 06], GoDiet [Caron 06a] ou JDF [Antoniu 04]. Ces derniers permettent d'automatiser toutes les tâches relatives au déploiement et/ou à l'administration de ressources logicielles en environnement de grille. En revanche, aucun ne propose de politique de déploiement réellement évoluée, et laissent tous la construction du plan de déploiement à la charge de l'utilisateur.

Des modules ont été développés pour combler cette lacune, tels que Sekitei mais aussi GARA (Globus Architecture for Reservation and Allocation), PSF , Conductor, Ninja, CANS. Ces derniers se focalisent sur les performances et sont incapables de prendre en compte des objectifs spécifiques, ce qui limite leur adaptabilité aux besoins utilisateurs et ainsi leur utilisabilité. De plus, ces modules ne répondent qu'à un aspect bien précis de la gestion de la répartition des ressources, et ce manque de généricité menace leur profitabilité compte tenu que leur exploitation peut d'avérer complexe et peuvent nécessiter de larges déploiements, sans pour autant répondre à tous les problèmes rencontrés par les utilisateurs des grilles.

La richesse et la complexité des grilles font qu'une grande part des travaux scientifiques s'intéresse à la conception d'intergiciels ou de superstructures. Dans ces cas, l'ampleur de la tâche implique bien souvent que les problématiques d'optimisation directement liées à la répartition doivent être rapidement résolues par l'utilisation de stratégies brutales. L'exemple de GGM est typique en ce sens où les travaux sur l'entrepôt de données distribué et ceux sur le système de caches collaboratifs doivent se concentrer sur des problèmes de conception tels que des systèmes d'indexation, d'agrégation de données distribuées, ou d'exécution de requêtes réparties, mais ne proposent pas d'implémentation de stratégies de gestion de la répartition particulières.

3.2 Les outils de prédiction de performances

La prédiction des performances est un aspect étroitement lié à la gestion de la répartition des ressources. En effet, être capable de prédire, à moindre coût, le temps nécessaire à un transfert de données ou à un calcul, permet de décider parmi plusieurs ressources candidates laquelle fournira les meilleures performances. En extrapolant un peu, on peut même imaginer de gérer le déploiement de ressources ou leur composition. Ce domaine se situe à l'intermédiaire entre la superstructure et l'infrastructure : les caractéristiques des ressources de la superstructures sont exploitées pour prédire les performances de leur utilisation en fonction des capacités de l'infrastructure.

3.2.1 NetSolve

Présentation

NetSolve est probablement le premier environnement de grille basé sur l'approche GridRPC. Il a été développé à l'Université du Tennessee par Seymour *et al.* et fait l'objet d'un présentation complète dans [Seymour 05]. Il est basé sur une architecture trois-tiers clients, agent et serveurs. L'agent sert d'intermédiaires entre le client, qui lui soumet ses problèmes, et les serveurs, qui lui communiquent régulièrement ses capacités et ses états.

De plus, lors de la soumission d'un problème, cet agent est capable d'estimer le temps d'exécution pour chaque serveur. Cette estimation est basée sur une formule analytique prenant en compte le comportement asymptotique de la fonction invoquée, la taille de l'instance du problème soumis et l'état du serveur et du réseau. L'agent sélectionne ainsi les meilleurs serveurs et en envoie la liste au client, qui peut ensuite les contacter directement.

On peut noter que les dernières versions de NetSolve utilisent NWS pour surveiller l'infrastructure.

Discussion

Le problème majeur de NetSolve, outre le fait qu'il est un environnement de grille, ou du moins de méta-computing, complet et par la même peu générique, réside dans la spécificité du problème géré. En effet, non seulement la formule analytique est imposée et donc restrictive, mais seul le problème de la sélection de serveurs est géré. Il est ainsi impossible d'intégrer ses propres métriques ou encore ses propres problèmes. Il est donc impossible d'utiliser NetSolve pour gérer les problèmes de placement ou encore de composition de ressources, encore moins des problèmes spécifiques.

3.2.2 FAST

Présentation

Fast Agent's System Timer est un outil de modélisation et de prédiction de performances développé par Martin Quinson à l'ENS Lyon et présenté dans [Quinson 02]. Il permet de prédire les temps d'exécution et les besoins mémoire de routine. Il utilise une modélisation avancée de la topologie du réseau afin d'extrapoler à partir des performances théoriques, les temps de transfert de données. De plus, il utilise une technique de macro-calibration des routines dont les résultats sont modélisés par une régression polynomiale qui lui permet de prédire le temps d'exécution de celles-ci sur chacun des hôtes disponibles. Les métriques de surveillances sont acquises à partir de NWS.

L'API de FAST dispose de quatre fonctions principales :
- `comm_time` qui donne le temps nécessaire à un transfert de données
- `comp_time` qui donne le temps nécessaire à l'exécution une routine
- `comp_size` qui donne la taille mémoire nécessaire à l'exécution une routine
- `get_time` qui agrège les résultats précédents.

Discussion

FAST est donc simple d'utilisation grâce à une API concise et complète. De plus il présente des résultats expérimentaux en comparaison à NetSolve très enthousiasmant. Il est utilisé sur la plateforme DIET [Caron 06b] pour réaliser l'ordonnancement des tâches.

Un premier problème est celui de la macro-calibration qui oblige à tester toutes les routines sur tous les types d'hôtes présents sur le réseau, et qui limite *de facto* la dynamicité de l'infrastructure. Mais une fois encore, c'est par le manque de généricité que FAST pêche. Il est en effet impossible de l'utiliser pour d'autres problèmes que la sélection de ressources sans efforts utilisateurs conséquents.

3.2.3 Delphoi

Présentation

Delphoi est un service développé par Maassen *et al.* à la Vrije Universiteit, Amsterdam et présenté dans [Maassen 06]. Son objectif est de permettre l'adaptation des intergiciels à leur infrastructure matérielle, ce qui est très proche de notre but. Il est conçu dans le cadre de GridLab[2] et s'intéresse aux problèmes qui y sont rencontrés :
- L'optimisation de protocole de transfert de données
- La sélection de réplicas
- La visualisation à distance de données
- L'estimation du temps d'attente de jobs.

[2] http ://www.gridlab.org/

Delphoi représente une interface unifiée à tout outil de surveillance sous-jacents. Il fournit ainsi les fonctions `estimateMetric` et `getRawMeasurementData` qui permettent respectivement d'exécuter des tests de performances et de récupérer leur résultats. Il propose aussi des fonctions de plus haut niveau, `estimateTcpOptions` qui propose une configuration Tcp optimale pour les communications entre deux hôtes, et `estimateTransferTime` qui fournit une estimation du temps de transfert d'une donnée. Enfin, il fournit des fonctions relatives aux files d'attente : `getQueueWaitingTime` qui estime le temps d'attente dans une file et `getResourceUtilization` qui estime le niveau d'utilisation d'une ressource donnée. Les prédictions sont basées sur la même librairie que le prédicteur de NWS.

Discussion

Plusieurs aspects sont intéressants dans Delphoi. Premièrement l'utilisation de plusieurs outils de surveillance, qui permet d'obtenir une vue plus complète sur l'infrastructure mais aussi d'ajouter de nouvelles sources (moyennant le développement d'un plugin « glue »). Ensuite l'étendue des problèmes de distribution est plus importante, ne se limitant pas à la sélection de ressources. Malheureusement, il est toujours impossible à l'utilisateur de fournir ses propres problèmes, ce qui représente une limite importante, par exemple pour le développeur de l'entrepôt de donnée de GGM qui ne pourra optimiser son application qu'au prix d'importants efforts.

3.2.4 Conclusion

En conclusion sur les outils de prédiction, on peut dire que ceux-ci manquent d'adaptabilité aux problèmes. Ils permettent en effet d'obtenir des estimations permettant de résoudre le problème de la sélection. En revanche les autres problèmes ne peuvent être résolus qu'avec un grande implication des utilisateurs.

De plus, on peut s'interroger sur l'utilité de fournir des prédictions précises. Les méthodes employées sont plutôt complexes, et outre un surplus dans les coûts de résolution des problèmes, cela implique un manque de flexibilité, les utilisateurs ne pouvant déclarer leur propres métriques sans risquer de compromettre l'intégrité des prédictions. Une approche plus basique basée sur un simple classement des différentes solutions candidates est aussi efficace en ce qui concerne la décision finale et peut s'avérer plus facile à mettre en place et à manipuler par les utilisateurs.

3.3 Les supports génériques de gestion de la répartition

Dans cette section nous présentons les principaux travaux qui se focalisent sur l'infrastructure matérielle. Ces derniers ont pour objectif de collecter et fournir des informations sur l'infrastructure, laissant à la charge de leurs utilisateurs de s'en servir pour gérer la distribution de leurs ressources en fonction de cette infrastructure. Ainsi, ces outils peuvent servir de support « bas niveau » à la gestion de la répartition des ressources.

On peut identifier deux grandes classes d'approches dans ce domaine scientifique : les outils de découverte de la topologie, qui se focalisent sur les aspects purement réseau et la distribution géogra-

phie des hôtes, et les outils de surveillance, qui visent à collecter de nombreuses informations aussi bien à propos du réseau que des hôtes.

3.3.1 Les outils de découverte de la topologie

L'objectif des outils de découverte de la topologie, aussi appelés *DMS - Distance Map Services*, est de fournir une carte du réseau qui soit fidèle aux changements afin de supporter une grande dynamicité tout en limitant le nombre de mesures afin d'assurer le passage à l'échelle.

Présentation

Les deux exemples d'utilisation présentés dans ces travaux sont la sélection d'un réplica parmi plusieurs candidats, le plan indiquant alors le candidat le plus « proche », et l'organisation de réseaux pair à pair, par exemple pour organiser leur topologie ou élire des supers nœuds[3].

L'objectif de la réduction du nombre de mesure se base sur les constats suivants. Pour connaître complètement les valeurs d'une métrique réseau, il faut effectuer n^2 mesures pour n hôtes. Dans un réseau dynamique, ces mesures doivent être effectuées régulièrement. Ainsi, on peut s'attendre à un nombre de mesures très important, mettant en péril les performances de la plateforme.

C'est pourquoi des solutions ont été étudiées afin de réduire ce nombre de mesures tout en garantissant des mesures fidèles aux performances réelles de l'infrastructure. Les deux solutions les plus abouties sont le Global Network Positionning - GNP et Internet Distance Maps - IDMaps. Elles sont toutes deux basées sur la même approche : l'utilisation d'hôtes de référence, appelés *balises* dans GNP et *traceurs* dans IDMaps. Les mesures sont alors limitées à une mesure entre les hôtes cartographiés et les hôtes de référence, ainsi qu'entre les hôtes de référence entre eux. Ensuite, à la demande d'un client, la mesure entre deux hôtes cartographiés est extrapolée.

On voit apparaître ici deux premières difficultés à la charge de l'utilisateur : le nombre d'hôtes références m doit être correctement fixé et les hôtes références correctement positionnés. En effet, plus m est petit, moins des mesures sont nécessaires, mais également moins les extrapolations sont fiables. La solution la plus simple est proposée par les auteurs et consiste à placer un hôte référence par grappe ou par zone géographique.

Les deux propositions diffèrent par la méthode d'extrapolation des mesures manquantes. IDMaps propose une heuristique triangulée : Soit $\{h_1, \ldots, h_n\}$ l'ensemble des hôtes cartographiés et $\{r_1, \ldots, r_m\}$ l'ensemble des hôtes références,

$$\forall i \in [1,n],\ \forall j \in [1,m],\ d(h_i,h_j) = \min_{k=1}^{m} \big(d(h_i,r_k) + d(h_j,r_k)\big)$$

Soit, la distance $d(h_i,h_j)$ entre les hôtes cartographiés h_i et h_j est la somme minimale des distances entre ces hôtes et un hôte référence r_k.

[3]Nœuds d'un réseau pair à pair disposant de plus de privilège ou de tâche supplémentaire par rapport aux autres

On peut ainsi compter, pour n hôtes cartographiés et m hôtes références, $n \times m$ mesures. Par exemple pour $n = 100$ et $m = 10$, on obtient $100 \times 10 = 1000$ mesures, soit une division du nombre de mesures par 10, des mesures complètes ayant été au nombre de 10000.

GNP, pour sa part, utilise un système de coordonnées dans un espace $\mathcal{S}$ à D dimensions. Les coordonnées des hôtes références sont calculées par la résolution d'un problème d'optimisation d'une fonction objective mesurant l'erreur entre la distance mesurée et la distance dans l'espace $\mathcal{S}$. Suite à quoi, chacun des hôtes cartographiés calculent leurs coordonnées dans l'espace $\mathcal{S}$ par rapport aux hôtes références selon la même méthode. Il suffit ensuite d'utiliser la fonction distance de cet espace pour obtenir la distance entre deux hôtes cartographiés.

On peut ainsi compter, pour n hôtes cartographiés et m hôtes références, $n \times m + m^2$ mesures. Par exemple pour $n = 100$ et $m = 10$, on obtient $100 \times 10 + 10^2 = 1100$ mesures, soit un rapport d'économie d'environ 9. La méthode proposée par GNP est donc plus complexes, mais affiche aussi, selon les auteurs, des résultats plus fiables.

GNP a été étendu au travers de nombreux travaux dont les plus importants sont : *NPS - Network Positioning System* [Ng 04] qui ajoute une hiérarchie dans les hôtes références afin réduire leurs charges et d'améliorer le passage à l'échelle, un système de contrôle des hôtes malicieux (mentir sur sa position afin de se faire passer pour plus proche que l'on est, peut être avantageux dans certains réseau pair à pair, au détriment des performances globales) ainsi qu'un système de contrôle des congestions ; *Lighthouse* [Pias 03], qui au contraire permet de ne pas contacter les hôtes références afin d'améliorer le passage à l'échelle dans certaines topologies et de réduire le temps de réponse ; *Vivaldi* [Dabek 04] qui supprime complètement les hôtes références, chaque nœud calculant initialement ses coordonnées par rapport à quelques pairs, et est utilisé dans le protocole d'échange pair à pair BitTorrent[4]

Discussion

Deux problèmes majeurs limitent l'utilisation des outils de découverte de la topologie pour la gestion de la répartition des ressources en environnement de grille.

Tout d'abord ils sont limités à une seule métrique : le RTT[5], qui est la plus simple et la plus légère à mesurer. Ils n'ont pas été testés avec la bande passante et ne sont pas du tout adaptés à la surveillance des métriques hôtes, tels que la puissance CPU. Ils ne donnent donc qu'un aperçu limité de l'infrastructure matérielle, trop limité pour couvrir tous les aspects impliqués dans la gestion de toutes les ressources logicielles présentes dans les grilles. De plus, toutes les mesures sont effectuées grâce au protocole ICMP[6], généralement bloqué par les pare-feux des organisations et donc inutilisables dans les grilles qui en comptent plusieurs.

Ensuite, les méthodes d'extrapolation se basent sur des hypothèses fortes sur les métriques réseau. Ces dernières sont supposées symétriques et doivent respecter l'inégalité triangulaire. Or il a été montré que les routes de l'Internet sont en majorité asymétriques. De plus, le paradigme de *best effort* adopté dans le routage IP ne garantit pas l'inégalité triangulaire. Ces aspects seront discutés plus

[4] http ://www.bittorrent.com/ site officiel essentiellement commercial

[5] RTT : Round Trip Time, temps nécessaire à une donnée de faible taille pour faire un aller-retour entre deux hôtes. Le RTT est généralement mesuré avec l'outil `ping`.

[6] Internet Control Message Protocol http ://www.faqs.org/rfcs/rfc792.html

largement dans le Chapitre 6, mais laissent présager d'une utilisation très risquée, même si l'économie du nombre de ressources paraît intéressante.

3.3.2 Les outils de surveillance

L'approche alternative pour pallier aux limitations constatées sur les outils de découverte consiste à ne pas se focaliser sur le nombre de mesures, mais plutôt sur l'efficacité du stockage et de la livraison des mesures effectuées. C'est l'approche adoptée par les outils de surveillance (ou *monitoring*). Nous allons présenter deux choses dans cette section : le *GMA - Grid Monitoring Architecture*, qui est largement adopté par l'immense majorité des outils de surveillance, et le NWS - *Network weather Service* qui a acquis une importante popularité et sur lequel nos travaux reposent en partie.

GMA - Grid Monitoring Architecture

La surveillance a rapidement été identifiée comme un aspect crucial dans les grilles de calcul. Selon Czajkowski, Kesselman, Fitzgerald et Foster dans l'article fondateur [Czajkowski 01b], les utilisateurs doivent disposer de supports afin de trouver et de garder la trace des ressources auxquelles ils portent intérêt. C'est l'objectif principal des services d'information de grille (*GIS - Grid Information Services*). Ils collectent systématiquement les informations à propos du statut courant des ressources de grille : un processus connu sous l'appellation surveillance ou *monitoring*. Il est crucial dans une large variété de cas tels que l'ordonnancement, le courtage de ressources, la réplication des données, la comptabilité[7], l'analyse des performances, l'optimisation des systèmes distribués ou des applications individuelles, l'auto-réglage des applications...

L'objectif n'est donc pas ici de limiter à tout prix le nombre de mesures afin d'obtenir l'architecture la plus légère possible, mais plutôt de proposer un large choix de métriques différentes, avec une garantie de fiabilité, et accessibles dans les meilleures conditions. Les outils de surveillance sont donc naturellement caractérisés par un coût d'utilisation élevé.

L'identification des métriques d'intérêt pour les environnements de grille, ainsi que les méthodes de mesurement, a été effectuée par le groupe de travail Mesures Réseau (*Network Measurements Working Group*) du *Global Grid Forum* dans [Lowekamp 04]. Les développements récents ont mené à des outils fournissant effectivement l'indexation des hôtes, données et services disponibles sur la grille. Les plus populaires qui sont le *MDS - Monitoring and Discovery Service*[8][Czajkowski 01a] de Globus, *R-GMA - Relationnal Grid Monitoring Architecture* [Cooke 04] et *SCALEA-G* [Truong 04] ont tous adopté l'architecture standard proposée par ce groupe de travail : *GMA - Grid Monitoring Architecture*.

L'architecture de GMA est présentée sur la Figure 3.1. Elle est composée de trois tiers :

- Les producteurs : ce sont les fournisseurs primaires d'informations, le plus souvent les ressources elles-mêmes, qui s'enregistrent auprès des annuaires et publient leur statut.
- Les consommateurs : ce sont les clients de ces informations, le plus souvent des utilisateurs mais aussi des applications ayant besoin de connaissances sur l'infrastructure pour fonctionner.

[7] En vue de facturation, *accounting* en anglais
[8] http ://www.globus.org/mds/

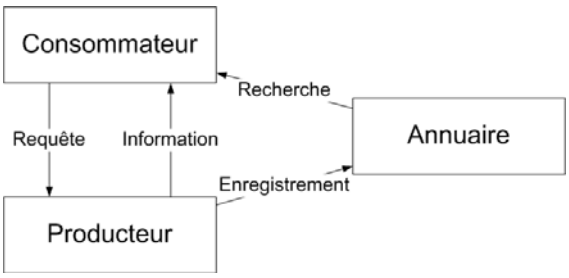


FIG. 3.1 – Architecture Logicielle de GMA

- Les annuaires : ce sont les intermédiaires qui permettent de mettre en relation producteurs et consommateurs en enregistrant les ressources lorsqu'elles sont disponibles et en mettant à disposition des consommateurs des fonctions de recherche.

Le fonctionnement de GMA est relativement simple : les ressources qui apparaissent s'enregistrent auprès d'un annuaire ; lorsqu'un consommateur cherche une ressource, il exprime ses besoins et les soumet à un annuaire qui lui renvoie les référence vers les producteurs demandés ; le consommateur n'a plus qu'à se connecter à ces producteurs pour obtenir les informations voulues.

GMA préconise l'utilisation de protocoles sans état : les ressources qui ne donnent pas signes de vie pendant une durée définie sont considérées disparues et supprimées des annuaires.

De plus, les solutions divergent sur le rôle des annuaires, certains stockant l'ensembles des informations de surveillance et pouvant donc jouer le rôle de fournisseurs. Certains, comme MDS, implémentent une hiérarchie à trois niveaux : les deux plus bas correspondent aux fournisseurs et annuaires, qui sont chapeautés par des annuaires agrégés contenant toutes les informations. Ceci permet d'améliorer le passage à l'échelle de la solution de surveillance, tout en ayant accès aux informations sur l'ensemble des ressources disponibles. Certaines implémentations, comme SCALEA-G, permettent également de surveiller les ressources logicielles. Enfin, les différentes implémentations de GMA diffèrent essentiellement par des choix technologiques : LDAP, XML ou base de données relationnelle. Ces choix influent peu sur l'utilisation finale des données et sont surtout conditionnés par les technologies utilisées sur les plateformes cibles.

Le Network Weather Service : NWS

Le Network Weather Service est un outil de surveillance développé par l'équipe du Professeur R. Wolski de l'Université de Californie, Santa-Barbara (UCSB). Il est caractérisé par une philosophie un peu différente de celles de ses homologues puisqu'au lieu de se concentrer sur l'efficacité des processus de mesure, de stockage et de diffusion des informations de surveillance, son objectif principal est de fournir un ensemble exhaustif de mesures concrètement utilisables par les applications. En effet, contrairement aux nombreux outils basés sur des mesures SNMP, NWS effectue ses mesures en effectuant directement de (petites) tâches à partir du niveau applicatif. Ainsi il peut mesurer la latence et la bande-passante disponibles entre deux nœuds du réseau directement en injectant des paquets et en mesurant leur temps de transfert. Il peut également mesurer le pourcentage de CPU disponible pour un nouveau processus en créant directement un nouveau processus et en mesurant son temps d'exécution. Ainsi il mesure les performances telles qu'elles sont perçues par n'importe quelle

autre application, et propose ainsi des informations pragmatiques sur les capacités des infrastructures matérielles. On peut d'ailleurs noter qu'il ne propose pas de mesures telles que le nombre de hops entre deux nœuds, qui ne représentent rien du point de vue applicatif.

De plus NWS ne se limite pas à des mesures directes du système, mais propose aussi de nombreux traitements statistiques permettant d'affiner les mesures prises et de proposer des prédictions à court et moyen terme. Là encore, NWS ne propose pas un modèle révolutionnaire à la précision parfaite, mais adopte une approche plus efficace qui consiste à utiliser systématiquement plusieurs modèles de prédiction et à confronter leurs résultats avec les mesures réelles prises *a postériori*. La prédiction finale donnée étant celle qui a montré le moins d'écart avec la réalité constatée.

Les différentes mesure que NWS propose sont : la bande passante, la latence et le temps d'ouverture d'une socket de chaque lien TCP reliant deux hôtes abritant des senseurs NWS ; la charge processeur, les espaces mémoire et disque disponibles et le nombre de processeurs pour chaque hôte.

Vue d'ensemble NWS comporte quatre composants dans une architecture logique hiérarchique :

- Les senseurs (S) qui prennent les mesures et sont déployés sur tous les hôtes surveillés.
- Les mémoire (M) qui stockent les résultats des mesures et sont typiquement déployés à raison d'un par site.
- Le serveur de noms (N) qui répertorient les senseurs et mémoires dans un annuaire type LDAP. Ce composant est unique.
- Les prédicteurs (P) qui appliquent les traitements statistiques sur les séries de mesures afin d'en extraire des prédictions. Ils sont compris dans chaque installation, mais ne sont pas inclus dans la hiérarchie de l'architecture puisqu'il ne participent pas directement au processus de surveillance.

Mesures sur les hôtes NWS propose de nombreuses mesures prises sur les hôtes sondés : la charge processeur actuelle, la quote-part disponible pour un nouveau processus, la quantité de mémoire vive et l'espace disque disponible, les vitesses de lecture et d'écriture sur le système de fichier... Ces mesures ne sont pas de simples extractions des informations mises à disposition par les systèmes d'exploitation, mais bien des mesures propres à NWS prises régulièrement. Ainsi on peut compter sur leur uniformité quel que soit le système d'exploitation sous-jacent[9].

Mesures réseaux NWS propose trois mesures des performances réseau : la bande passante, la latence et la durée d'établissement d'une socket pour tout lien TCP/IP ou UDP¨entre deux de ses senseurs.

La latence est une approximation extraite de l'aller-retour (Round-Trip Time) d'un paquet de petite taille : la machine source chronomètre le temps nécessaire pour que la machine destination lui renvoie un paquet de quatre octets. De même, la machine source chronomètre le temps nécessaire entre l'ouverture d'une nouvelle socket et la fin d'un envoi d'un paquet de quatre octets. Le temps nécessaire à l'ouverture de la socket est alors utilisé par NWS pour déduire la durée de la latence. La bande-passante est mesurée selon la même méthode : NWS chronomètre le temps nécessaire à la transmission d'un paquet de taille donnée, par défaut 250 Ko. Il est important de noter que la

[9]On parle ici de distribution Linux, MS Windows n'étant pas supporté

précision de cette mesure de bande-passante dépend du réglage de la taille de cette donnée, ainsi que de celle du tampon utilisé par NWS, par défaut 32 ko. En effet, ces tailles dépendent du réglage des paramètres TCP/UDP/IP et des capacités de l'infrastructure matérielle : 250 ko est en même temps un peu limitée pour des communication très haute performance (supérieure au Gb) et un peu lourde pour du réseau classique (de l'ordre du Mb). De même la taille du tampon dépend du matériel de connexion, les cartes réseaux Gb se démocratisant mais cohabitant encore avec des cartes 100 Mb. On voit ici apparaître un problème qui sera discuté en section 3.3.2.

On peut noter que sur des réseaux très haute performance, même avec un réglage adéquat, NWS pourra fournir des mesures de bande passante maximale de l'ordre de 100 Mb/s. Il ne s'agit pas là d'une erreur, mais d'une fonctionnalité puisque cette valeur représente alors les performances réelles qu'une application utilisant une seule socket peut espérer.

Un autre aspect est que les mesures réseau nécessitent un peu d'organisation. En effet, pour un réseau comportant n hôtes, il faut réaliser $n \times (n-1)$ mesures, les capacités des liens n'étant communément pas symétriques comme montré par les expérimentations à échelle mondiale de He dans [He 05]. De plus, les mesures ne doivent pas entrer en collision sur le même lien au même instant, sous peine de fausser les résultats. NWS gère ce problème grâce au concept de cliques, qui sont un regroupement d'hôtes partageant les mêmes liens réseau. Un système de jeton permet alors aux membres d'une même clique d'être sollicités à tour de rôle pour effectuer leurs mesures, évitant ainsi ces collisions.

Discussion

NWS a su s'imposer comme le service de surveillance de référence, comme en témoignent les différents projets qui l'emploient. Cette popularité vient de sa philosophie particulière qui n'est pas orientée sur la détection de failles et dysfonctionnements, mais bien sur la mesure des performances telles qu'elles sont perçues par les applications. Ces informations peuvent ainsi se rendre utiles pour de nombreuses tâches telles que l'ordonnancement ou le courtage.

Une autre de ses qualités est que NWS effectue ses mesures sans privilèges particuliers. Il peut donc être mis en place et utilisé très facilement, y compris dans les environnements multi-organisationnels sans générer de pénibles, et souvent insolvables, problèmes d'administrations.

Cependant on peut formuler deux critiques d'importance. La première est que la précision des mesures réseau nécessite un réglage uniforme sur toute l'infrastructure. Cela peut poser problème dans les environnements très hétérogènes, car ces réglages seront soit complètement inadaptés pour une sous-partie de l'environnement, soit moyennement adaptés à son ensemble. On peut donc craindre un certain manque de précision dans les mesures dans ces environnements.

La deuxième, et plus importante critique concerne les informations produites qui restent, malgré tout, de bas-niveau. En effet, même si elles peuvent paraître de haut-niveau par rapport à l'infrastructure puisque mesurée au niveau applicatif, elles sont d'un niveau inférieur à la superstructure qui se situe au dessus d'un intergiciel. En particulier, l'avènement des services web au sein des architectures orientées services ont éloigné les utilisateurs de ces considérations, désormais reléguées aux intergiciels. Ainsi le développeur d'un service a d'autres préoccupations que de s'intéresser aux capacités unitaires de chaque lien réseau : les concepts de latence ou encore de charge CPU ne font

pas nécessairement parties de ses compétences. Plus qu'un problème de compétences, le développeur se retrouve face à un problème de retour sur investissement, devant traiter en priorité les contraintes fonctionnelles de ses applications avant de s'intéresser aux optimisations liées à la nature distribuée de son environnement. Pourtant, ces aspects ont un impact direct sur les performances des applications, qui sont également par nature réparties. Ces difficultés se retrouvent d'ailleurs dans le développement de la superstructure GGM, présentée Section 1.2.1.

3.4 Synthèse de l'étude des travaux connexes

Nous avons présenté dans cette partie trois approches relatives à la gestion de la distribution des ressources en environnement de grille : au niveau de la superstructure on trouve des solutions spécifiques, entre la superstructure et l'infrastructure on trouve les système de prédiction des performances, enfin au niveau de l'infrastructure on trouve les outils de découverte de la topologie et les outils de surveillance.

Les systèmes de gestion spécifique présentent des solutions souvent très intéressantes dans un contexte particulier donné pour un problème donné. Par exemple, de nombreux travaux proposent des solutions de placement de ressources logicielles sur l'infrastructure matérielle. Mais ils s'avèrent être caractérisés par une spécificité trop marquée qui limite leur utilisabilité. En effet, une super structure logicielle complexe mettant en jeu de nombreux processus aux besoins divers devra utiliser plusieurs systèmes différents afin de tous les couvrir, ce qui s'accompagne d'études pour choisir lesquelles, d'efforts d'implémentation pour les intégrer et d'une surcharge de la plateforme pour les exécuter. De plus, les développeurs des superstructures sont souvent amenés à rencontrer des problèmes originaux (voir Section 1.2.1), qui par définition n'ont pas été étudiés dans la littérature. Par nature, les solutions spécifiques manquent de généricité pour l'objectif fixé dans cette thèse.

Les outils de prédictions des performances, quant à eux, présentent des capacités intéressantes, essentiellement pour le problème de la sélection, puisqu'ils fournissent des évaluations sur le temps d'exécution des tâches de transfert et de calcul. En revanche, il est impossible à l'utilisateur de soumettre ses propres problèmes, ce qui limite l'adaptabilité des ces solutions. Or, leur utilisation peut s'avérer coûteuse et problématique car ils mettent en œuvre des techniques souvent complexes et gourmandes, tant en terme de ressources qu'en terme d'informations. L'utilisabilité et la profitabilité de ses outils ne sont donc pas satisfaisantes dans le contexte de cette thèse.

Les outils de découverte de la topologie permettent d'obtenir un plan fidèle de l'infrastructure matérielle, et plus particulièrement de son réseau, avec un nombre de mesures limité. Ils présentent ainsi un coût d'utilisation attrayant par sa légèreté. En revanche, ils se basent sur des hypothèses particulièrement fortes quant aux propriétés des métriques fournies, ce qui limite leur utilisation aux environnements ne présentant pas de dynamicité ou d'hétérogénité trop fortes. Mais surtout, les informations fournies sont trop restreintes puisque limitées en pratique à uniquement le *RTT - Round Trip Time*. Or de nombreux aspects plus complexes sont impliqués dans la gestion de la répartition des ressources, par exemple la bande-passante ou les puissances CPU. De plus ces outils ne sont pas, ou peu, extensibles et ne peuvent donc pas intégrer de plus nombreuses métriques. Ils ne sont donc pas adaptés aux besoins fixés dans cette thèse puisqu'ils ne reflètent que partiellement les caractéristiques de l'infrastructure matérielle.

Les outils de surveillances ne sont pas soumis à la même remarque puisque leur objectif est de fournir des informations exhaustives sur l'infrastructure matérielle en se focalisant sur l'efficacité de l'architecture de production / consommation / stockage. En revanche, ces outils sont caractérisés par un coût d'utilisation élevé. De plus les informations délivrées (latence, bande passante, vitesse CPU, etc.) sont loin des préoccupations de leurs utilisateurs finaux, qui vont plutôt s'intéresser au coût de transfert d'une donnée ou à celui d'un calcul. Ainsi, la profitabilité de ces outils n'est pas satisfaisante puisque leur utilisabilté limitée ne justifie pas leurs coûts d'utilisation.

3.4.1 Synthèse récapitulative

Le Tableau 3.1 montre une synthèse des trois approches relatives la gestion de la répartition des ressources sur grille. Les « X » indiquent une parfaite adéquation et les « x » une adéquation limitée aux critères d'évaluation fixés dans cette thèse :

- *généricité* : pouvoir gérer un large spectre de problèmes différents, y compris inattendus
- *adaptabilité* : prise en compte de différents objectifs déclarés par l'utilisateur, y compris sémantique ou financier
- *adaptativité* : prise en compte automatique des caractéristiques de la superstructure
- *hétérogénéité* : prise en compte de matériel de capacités très différentes sur l'infrastructure
- *dynamicité* : prise en compte de la fluctuation des performances
- *évolutivité* : prise en compte de modifications profondes des plateformes suite à l'apparition de nouveaux besoins/objectifs
- *utilisabilité* : facilement compréhensible et utilisable par toute classe d'utilisateur
- *profitabilité* : présenter un rapport coût/profit d'utilisation satisfaisant

Tab. 3.1 – Synthèse des trois approches possibles pour la gestion de la répartition des ressources sur grille par rapport aux critères d'évaluation

approche	généricité	adaptabilité	adaptativité	hétérogénéité	dynamicité	évolutivité	utilisabilité	profitabilité
solutions spécifiques			X	x	x		x	x
outils de prédiction	x		X	X	X			x
outils de surveillance	X			X	X	X		

Ainsi, aucune des approches développées à ce jour ne couvre complètement tous les requis présentés dans l'introduction : profitabilté, utilisabilité, généricité et adaptabilité dans un contexte dynamique et hétérogène. De plus, les aspects qui ne sont pas directement liés aux performances, tels que les coûts financiers, ne sont jamais pris en compte dans les travaux étudiés.

En revanche, ont peut noter que seuls les outils de surveillance résistent à l'évolutivité des plateformes. En effet, comme ces derniers ne considèrent pas la superstructure logicielle, les changements profonds de besoins/objectifs ne les affectent que peu. De plus ils sont conçus pour non seulement supporter, mais surtout refléter les changements de l'infrastructure. Les autres approches sont trop spécifiques à leurs environnements pour assurer la pérénité des plateformes face à une forte évolutivité. C'est pourquoi notre approche consiste à exploiter les outils de surveillance en ajoutant des fonctionnalités adaptatives et adaptables, tout en améliorant leur utilisabilité et leur profitabilité.

Deuxième partie

$\mathcal{CNP}$-Graphe : modélisation et résolution de problèmes liés à la répartition des ressources

Dans cette partie, nous présentons le cœur de notre approche qui constitue également notre contribution principale.

Dans le Chapitre 4, nous montrons comment les problèmes de répartition classiques, sélection, déploiement et composition, peuvent être modélisés génériquement sous leur forme logique grâce à des graphes dits *Computer Network Problem* ou $\mathcal{CNP}$-Graphes. Cette modélisation à l'avantage d'être intuitive et aisément adaptable à toute sorte de problème.

Ensuite, Chapitre 5, nous présentons comment il est possible d'agrémenter ces $\mathcal{CNP}$-Graphes de labels pertinents au travers de fonctions distance. Ces distances permettent de facilement évaluer un très large panel d'objectifs, tels que les performances, en intégrant les coûts de calcul et de transfert, les aspects sémantiques, tels que le fraîcheur des données, ou les aspects opérationnels, tels que les coûts financiers. Ces distances assurent l'adaptation aux différents besoins utilisateurs en modélisant des objectifs d'optimisation variés et sont utilisées pour transformer la modélisation logique des problèmes en modélisation de leur réalité physique.

Puis, Chapitre 6, nous montrons comment la modélisation de la réalité physique des problèmes peut être exploitée par des algorithmes de graphes afin d'en calculer les solutions. Comme de nombreux algorithmes ont été décrits dans la littérature pour résoudre chacun des problèmes présentés, nous montrons comment sélectionner les algorithmes valides en évaluant la satisfaction de leur contraintes aux travers des propriétés des $\mathcal{CNP}$-Graphes labélisés.

Le Chapitre 7 présente l'implémentation de notre approche. Nous y détaillons le service web *Network Distance Service* qui intègre le processus de modélisation et de résolution proposé, sa configuration et son utilisation, ainsi que son interface graphique.

Ensuite, le Chapitre 8 présente les résultats expérimentaux obtenus afin de valider notre approche. Les problèmes de sélection, déploiement et composition ont été résolus sur Grid 5000, un environnement de grille réel. Nous montrons notamment que notre approche permet effectivement de s'adapter aux caractéristiques de la ressources sujette et aux performances de l'infrastructure.

Enfin, le Chapitre 9 présente l'intégration du *Network Distance Service* au sein de la superstructure GGM et détaille les différents avantages de son utilisation.

4

Modélisation des problèmes liés à la répartition des ressources

Nous avons présenté dans l'introduction les problématiques liées à la résolution des problèmes de distribution des ressources en environnement de grille. Une des problématiques les plus importantes est la nécessité d'avoir une approche qui permette de modéliser les problèmes liés à la répartition des ressources qui soit à la fois facile à appréhender, générique et dotée d'une grande expressivité, ce afin de concorder avec les nombreux problèmes auxquels les acteurs peuvent être confrontés. Ces caractéristiques s'appuient naturellement sur une modélisation qui représente la base de notre travail et est présentée dans ce chapitre. Notre modélisation est illustrée dans le cadre des exemples concrets relatifs aux différents problèmes cibles.

Dans la suite, compte-tenu de l'ambigüité du terme *nœud* nous désignerons par *sommets* les éléments propres aux graphes, par *hôtes* les éléments propres aux plateformes. Le terme *nœud* quant à lui désignera les éléments qui correspondent tout à la fois aux graphes et aux plateformes.

Ce chapitre est organisé en trois sections : Section 4.1, nous présentons la modélisation proposée, des exemples sont montrés Section 4.2, puis la modélisation est discutée et le chapitre conclu Section 4.3.

4.1 Définition des $\mathcal{CNP}$-Graphes

Notre modélisation possède deux facettes. Premièrement, la modélisation des problèmes du point de vue logique ou sémantique, c.-à-.d. de la façon dont les acteurs les perçoivent. Cette modélisation est l'objectif des *Computer Network Problem Graph compacts*, notés $c\mathcal{CNP}$-Graphes. Deuxièmement, la modélisation des problèmes du point de vue physique, c.-à-d. leur instanciation compte-tenu de

l'emplacement réel sur l'infrastructure matérielle des différentes ressources et des paramètres effectifs impliqués dans le problème. Cette modélisation est l'objectif des *Computer Network Problem Graph*, notés $\mathcal{CNP}$-Graphes.

En effet, on peut remarquer que plusieurs ressources physiques (matérielles ou logicielles) occupent la même fonction logique. Par exemple plusieurs fichiers (physiques) occupent la même fonction logique si ils sont les réplicas d'une même donnée (logique). Dans les $\mathcal{CNP}$-Graphes compacts, les nœuds représentent les ressources logiques et sont labélisés avec la liste des réplicas physiques de cette ressource. Les arcs représentent alors l'interaction logique de ces ressources logiques. Par interaction logique, nous entendons par exemple l'invocation d'un service ou l'utilisation d'une donnée.

Définition II.1 *$c\mathcal{CNP}$-Graphe : graphe compact de problèmes de distribution de ressources*

Nous appelons $c\mathcal{CNP}$-Graphe, pour compacts Computer Network Problem Graph, un graphe décrivant du point de vue logique ou sémantique un problème de distribution de ressources sur une grille,.

$$c\mathcal{CNP} = (c\mathcal{V}, c\mathcal{E}, TPS, dfS)$$

où

- *$c\mathcal{V}$ est un ensemble fini non vide de sommets. Chaque sommet V est labélisé avec l'ensemble des hôtes représentant la même fonction logique, ou hébergeant la même ressource logique. Cet ensemble d'hôtes est noté $V.H$. Un hôte h issu de cet ensemble est noté $V.h$.*
- *$c\mathcal{E}$ est un ensemble fini d'arcs. Un arc est un couple (ordonné) de sommets $(Vs, Vd) \in c\mathcal{V}^2$.*
- *$TPS = \{tp_1, \ldots tp_n\}$ est un ensemble de variables. Ces variables sont spécifiées par l'utilisateur et décrivent la tâche sujette du problème.*
- *dfS est un ensemble contenant une fonction distance par arc. La fonction distance de l'arc $(Vs, Vd) \in c\mathcal{E}$ est notée $df_{Vs,Vd}$.*

Les fonctions distance et l'ensemble de variables TPS défini par l'utilisateur permettent de modéliser les différents coûts impliqués dans le problème. Tous les détails seront donnés dans le Chapitre 5.

Un $\mathcal{CNP}$-Graphe compact représente ainsi une expression simple de la modélisation d'un problème de répartition. Il peut être transformé en $\mathcal{CNP}$-Graphes, lesquels modélisent la réalité physique du problème dans un contexte donné. Ainsi, un $\mathcal{CNP}$-Graphe peut être compris comme une instanciation particulière d'un problème logique décrit sous forme de $\mathcal{CNP}$-Graphe compact.

Définition II.2 *$\mathcal{CNP}$-Graphe : graphe de problème de distribution de ressources*

Nous appelons Computer Network Problem Graph*, $\mathcal{CNP}$-Graphe, un graphe décrivant du point de vue physique un problème de gestion de répartition des ressources sur grille*

$$\mathcal{CNP} = (\mathcal{V}, \mathcal{E}, d)$$

où

- *$\mathcal{V}$ est un ensemble fini non vide de sommets représentant des hôtes du réseau et labélisés avec leur nom*
- *$\mathcal{E}$ est un ensemble fini d'arcs. Un arc est un couple (ordonné) de sommets $(hs, hd) \in \mathcal{V}^2$.*
- *$d : \mathcal{E} \rightarrow \mathbb{R}$ est une fonction distance servant à labéliser ces arcs. La distance de l'arc $(hs, hd) \in \mathcal{E}$ est notée $d(hs, hd)$*

La distance représente ici le coût réel issu du modèle de coût inclus dans le *c*$\mathcal{CNP}$-Graphes. Nous verrons également Chapitre 5 les détails sur ce sujet.

Le passage d'un $\mathcal{CNP}$-Graphe compact à un $\mathcal{CNP}$-Graphe s'effectue grâce à la fonction *decompact*

Définition II.3 *Fonction decompact mettant en correspondance c*$\mathcal{CNP}$*-Graphe et* $\mathcal{CNP}$*-Graphe associé*

$$
\begin{array}{rlll}
decompact : & c\mathcal{CNP} & \rightarrow \mathcal{CNP} & \\
 & c\mathcal{V} & \mapsto \mathcal{V} = \bigcup_{V \in c\mathcal{V}} V.H & \\
 & c\mathcal{E} & \mapsto \mathcal{E} = \bigcup_{(Vs,Vd) \in c\mathcal{E}} Vs.H \times Vd.H & \\
 & dfS & \mapsto d(Vs.hs, Vs.hd) = df_{Vs.hs,Vd.hd} & : TPS \rightarrow \mathbb{R}
\end{array}
$$

où $df_{Vs.hs,Vd.hd}$ désigne l'instanciation de la fonction distance $df_{Vs,Vd}$ pour l'arc $(Vs.hs, Vd.hd) \in \mathcal{E}$ du CNP-Graphe.

Cette fonction rassemble tous les hôtes présents dans les labels des sommets du $\mathcal{CNP}$-Graphe compact pour former l'ensemble des sommets du $\mathcal{CNP}$-Graphe, les arcs de ce dernier relient les sommets pour lesquels une fonction distance a été déclarée et dont l'instanciation sert à les labéliser. Cette notion d'instanciation sera développée Chapitre 5.

Nous allons maintenant montrer dans la section suivante comment les problèmes présentés dans l'introduction peuvent être modélisés sous forme de *c*$\mathcal{CNP}$-graphes et transformés en $\mathcal{CNP}$-graphes.

4.2 Exemples de $\mathcal{CNP}$-Graphes

Dans cette section, nous nous intéressons uniquement à la structure des $\mathcal{CNP}$-Graphes et leurs versions compactes. Les labels des arcs, l'ensemble TPS et les fonctions distances seront présentés Chapitre 5. De plus, afin de faciliter sa compréhension, nous présentons le processus de modélisation « à l'envers », c.-à-d. en commençant par présenter le problème physique, puis le $\mathcal{CNP}$-Graphe et enfin le $\mathcal{CNP}$-Graphe compact.

4.2.1 Sélection

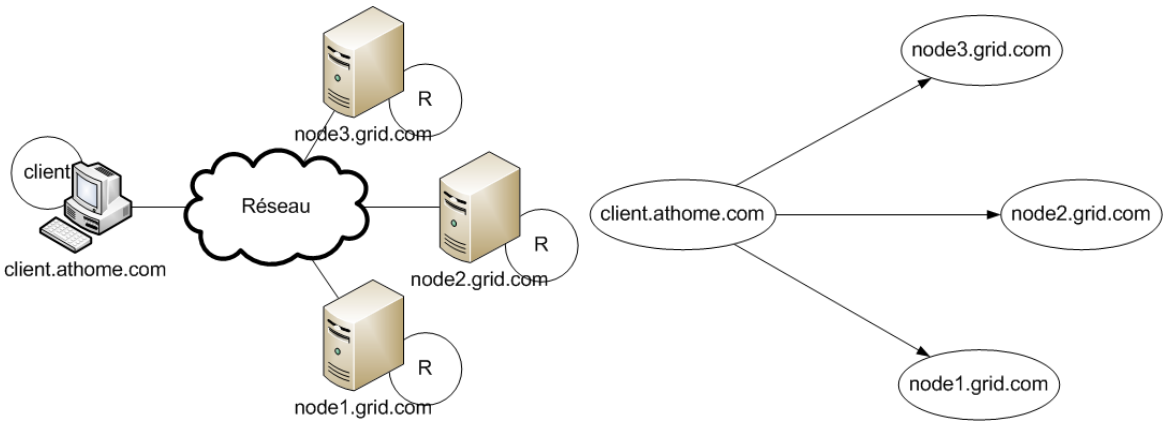


FIG. 4.1 – Illustration du problème de la sélection d'une ressource R par *client* et $\mathcal{CNP}$-Graphe correspondant.

La Figure 4.1 montre le problème tel qu'il a été présenté Chapitre 2 ainsi que le $\mathcal{CNP}$-Graphes correspondant : le client et les hôtes hébergeant la ressource R demandée sont modélisés comme des sommets, les arcs représentent le coût d'utilisation de la ressource R sur l'hôte destination par le client source de l'arc.

On peut noter deux entités logiques dans ce problème : le *client* et la ressource R. Ainsi on peut définir de manière générique les $c\mathcal{CNP}$-Graphes représentatifs d'un problème de sélection d'une ressource R, par $(c\mathcal{V}, c\mathcal{E})$ tels que :

- c$\mathcal{V}$ contient deux sommets : *client* et ressource R à sélectionner, chacun étant labélisé avec les hôtes occupant la fonction logique associée.
- c$\mathcal{E}$ est un singleton : l'arc dont la source appartient à *client* et la destination R car la seule interaction impliquée est celle du client utilisant la ressource.

On peut noter ici que les ensembles d'hôtes associés à *client* et R ne sont pas nécessairement distincts. On peut par exemple vouloir évaluer s'il est plus efficace d'effectuer un calcul intensif sur l'hôte client ou sur une machine plus puissante au prix du transfert des données.

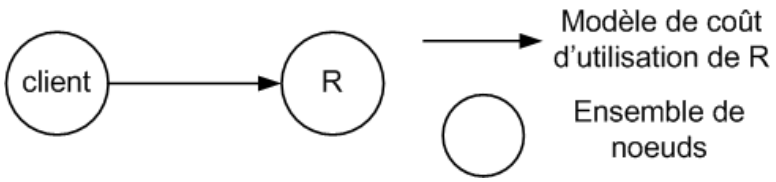


FIG. 4.2 – $\mathcal{CNP}$-Graphe compact correspondant au problème de la sélection d'une ressource répliquée R par un *client*

Le $\mathcal{CNP}$-Graphe compact correspondant est montré Figure 4.2 et défini comme tel :

$$c\mathcal{CNP} = \begin{cases} c\mathcal{V} = & \{client = \{\texttt{client.athome.com}\}, \\ & \quad R = \{\texttt{node1.grid.com}, \texttt{node2.grid.com}, \texttt{node3.grid.com}\}\} \\ c\mathcal{E} = & \{(client, R)\} \end{cases}$$

L'utilisation de la fonction *decompact* présentée précédemment effectue l'opération suivante :

$$\begin{cases} \mathcal{V} = client \cup R \\ \mathcal{E} = client \times R \end{cases}$$

Pour des raisons de lisibilité, *client* et R sont confondus ici avec $client.H$ et $R.H$. Ce sera la cas dans la suite de ce document.

Et permet de retrouver le $\mathcal{CNP}$-Graphe de la Figure 4.1 :

$$\mathcal{CNP} = \begin{cases} \mathcal{V} = & \{\texttt{client.athome.com}, \texttt{node1.grid.com}, \texttt{node2.grid.com}, \texttt{node3.grid.com}\} \\ \mathcal{E} = & \{(\texttt{client.athome.com}, \texttt{node1.grid.com}), \\ & (\texttt{client.athome.com}, \texttt{node2.grid.com}), \\ & (\texttt{client.athome.com}, \texttt{node3.grid.com})\} \end{cases}$$

Un utilisateur peut donc très simplement modéliser un problème de sélection de ressource et stipuler ses différentes instances dans un $c\mathcal{CNP}$-Graphe, puis utiliser la fonction *decompact* afin de générer un $\mathcal{CNP}$-Graphe représentatif de son problème. Il ne lui reste ensuite qu'à utiliser un simple algorithme de sélection du minimum pour trouver le couple $(hs, hd) \in \mathcal{E}$ représentant le couple d'hôtes le plus à même de remplir la tâche envisagée.

4.2.2 Déploiement

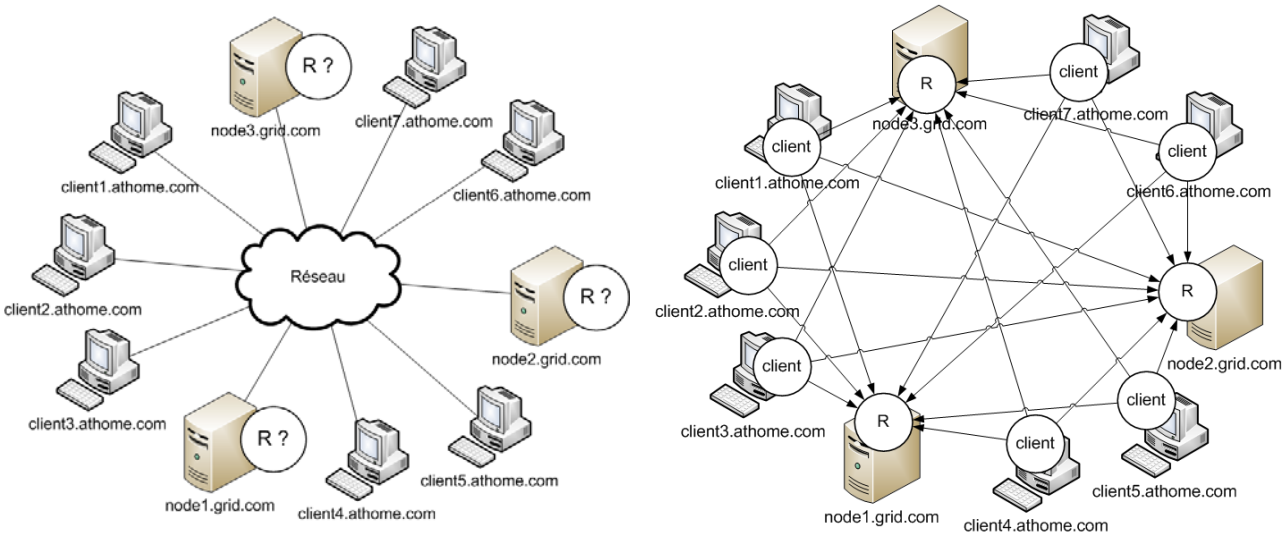


FIG. 4.3 – Illustration du problème du déploiment d'une ressource R par *client* et $\mathcal{CNP}$-Graphe correspondant

La Figure 4.3 montre l'illustration d'un problème de déploiement et le le $\mathcal{CNP}$-Graphe correspondant : une ressource R doit être déployée sur un des hôtes du domaine `grid.com` afin d'être utilisée par un ensemble d'hôtes du domaine `athome.com`.

Les hôtes se répartissent selon deux fonctions logiques : client et hébergeur. Le $c\mathcal{CNP}$-Graphe correspondant à ce problème (Figure 4.4) est donc composé de :

– $c\mathcal{V}$ qui contient deux ensembles *client* et R. *client* contient les clients potentiels de la ressource devant être déployée alors que R contient tous les emplacements potentiels de cette ressource.
– $c\mathcal{E}$ qui contient tous les arcs dont la source appartient à *client* et la destination R car la seule interaction impliquée est celle des clients utilisant la ressource.

Soit :

$$c\mathcal{CNP} = \begin{cases} c\mathcal{V} = & \{client = \{\texttt{client1.athome.com}, \ldots, \texttt{client7.athome.com}\}, \\ & \quad R = \{\texttt{node1.grid.com}, \texttt{node2.grid.com}, \texttt{node3.grid.com}\}\} \\ c\mathcal{E} = & \{(client, R)\} \end{cases}$$

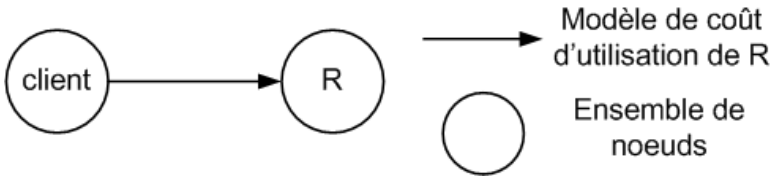


FIG. 4.4 – $\mathcal{CNP}$-Graphe compact correspondant au problème du déploiement d'une ressource R utilisée par des *clients*

L'utilisation de la fonction *decompact* présentée précédemment effectue l'opération suivante :

$$\begin{cases} \mathcal{V} = client \cup R \\ \mathcal{E} = client \times R \end{cases}$$

Et permet de retrouver le $\mathcal{CNP}$-Graphe de la Figure 4.3 :

$$\mathcal{CNP} = \begin{cases} \mathcal{V} = & \{\texttt{client1.athome.com}, \ldots, \texttt{client7.athome.com}, \\ & \texttt{node1.grid.com}, \texttt{node2.grid.com}, \texttt{node3.grid.com}\} \\ \mathcal{E} = & \{(\texttt{client1.athome.com}, \texttt{node1.grid.com}), \\ & (\texttt{client1.athome.com}, \texttt{node2.grid.com}), \\ & (\texttt{client1.athome.com}, \texttt{node3.grid.com}), \\ & \ldots \\ & (\texttt{client7.athome.com}, \texttt{node1.grid.com}), \\ & (\texttt{client7.athome.com}, \texttt{node2.grid.com}), \\ & (\texttt{client7.athome.com}, \texttt{node3.grid.com})\} \end{cases}$$

On peut remarquer que le $\mathcal{CNP}$-Graphe compact, montré Figure 4.4, est identique pour les problèmes de sélection et de déploiement. C'est là une des forces de cette approche : plusieurs problèmes peuvent être traités avec les mêmes graphes, ce qui améliore son utilisabilité, réduisant les efforts imposés aux utilisateurs lors de l'acquisition de la méthode ainsi que lors de son exploitation. La résolution du problème se concentre non sur la structure du graphe de modélisation, mais sur son analyse a l'aide d'algorithmes adaptés.

4.2.3 Composition

La Figure 4.5 montre le $\mathcal{CNP}$-Graphe compact du problème de composition de ressources présenté Section 2.3. On peut remarquer qu'il est strictement identique au graphe de composition sémantique. C'est une autre force de notre approche : beaucoup de processus sont déjà modélisés sous forme de graphe, pouvant ainsi être réutilisés directement, sans surcharge de travail particulière pour l'utilisateur.

Le $\mathcal{CNP}$-Graphe est donc composé de :
- $c\mathcal{V}$ contient les différents services impliqués dans l'adaptation du contenu.
- $c\mathcal{E}$ contient tous les arcs dont la source et la destination peuvent directement interagir en échangeant des contenus intermédiaires à adapter.

Ainsi :

$$c\mathcal{CNP} = \begin{cases} c\mathcal{V} = \{ & Fournisseur,\ Convertisseur\_de\_taille,\ Convertisseur\_de\_format, \\ & DeMux,\ Extracteur\_de\_voix,\ Transcripteur\_de\_voix, \\ & Traducteur,\ Incrusteur\_de\_soustitres,\ client\} \\ \\ c\mathcal{E} = \{ & (Fournisseur,\ DeMux), \\ & (Fournisseur,\ Incrusteur\_de\_soustitres), \\ & (Fournisseur,\ Convertisseur\_de\_taille), \\ & (Fournisseur,\ Convertisseur\_de\_format), \\ & (DeMux,\ Extracteur\_de\_voix), \\ & (Extracteur\_de\_voix,\ Transcripteur\_de\_voix), \\ & (Transcripteur\_de\_voix,\ Traducteur), \\ & (Traducteur,\ Incrusteur\_de\_soustitres), \\ & (Incrusteur\_de\_soustitres,\ Convertisseur\_de\_taille), \\ & (Incrusteur\_de\_soustitres,\ Convertisseur\_de\_format), \\ & (Convertisseur\_de\_format,\ Incrusteur\_de\_soustitres), \\ & (Convertisseur\_de\_format,\ Convertisseur\_de\_taille), \\ & (Convertisseur\_de\_format,\ client), \\ & (Convertisseur\_de\_taille,\ Incrusteur\_de\_soustitres), \\ & (Convertisseur\_de\_taille,\ Convertisseur\_de\_format), \\ & (Convertisseur\_de\_taille,\ client)\} \end{cases}$$

On peut noter que ce $\mathcal{CNP}$-Graphe compact est assez complexe et aboutira donc à un $\mathcal{CNP}$-Graphe qui l'est encore plus. Dans les faits, il est rare qu'un utilisateur humain construise lui-même le chemin d'adaptation dont il est le consommateur. Cette tâche est assumée par le gestionnaire de ressources et le système de composition. Or comme le $\mathcal{CNP}$-Graphe compact correspond exactement au graphe de composition, sa création ne représente en fait pratiquement pas d'effort.

Après l'utilisation de la fonction *decompact*, on peut utiliser un algorithme de recherche du plus court chemin traversant un élément de chaque sous-ensemble de $c\mathcal{V}$ pour optimiser la composition des ressources.

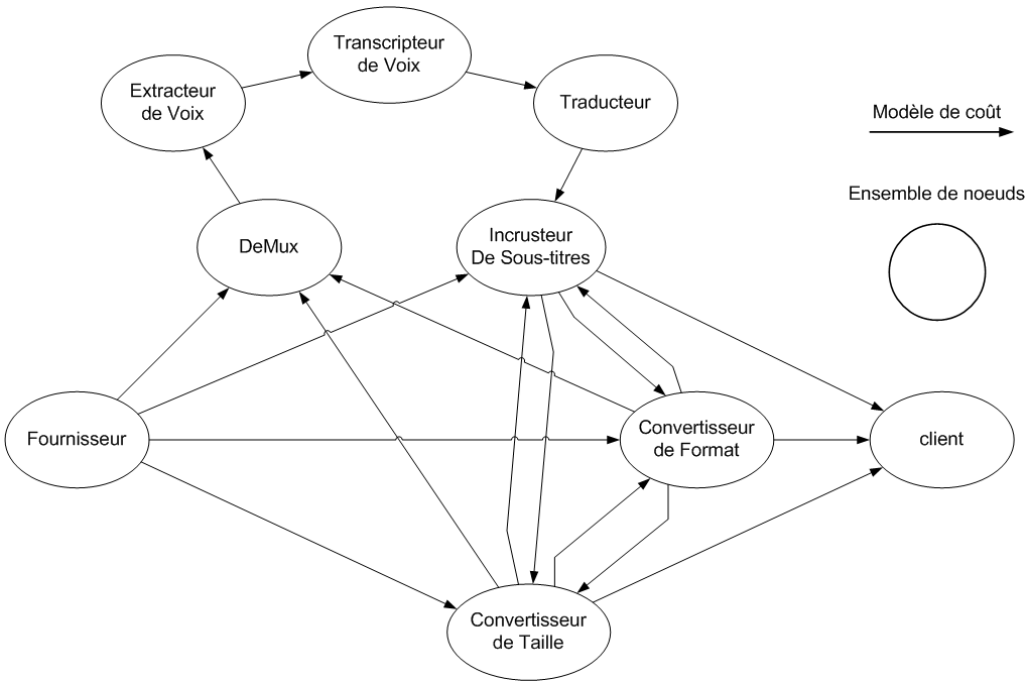


FIG. 4.5 – $\mathcal{CNP}$-Graphe illustrant le problème de composition de ressources

4.2.4 Problème spécifique de l'agrégation dans un entrepôt de données distribué

L'exemple de l'agrégation des données dans un entrepôt de données distribué sur grille présenté Section 2.4 est illustré Figure 4.6. Le problème consiste à évaluer le meilleur moyen de matérialiser un agrégat sur un hôte donné *client*. Pour cela, on peut soit transférer directement cette donnée si elle est déjà matérialisée, ou bien la recalculer à partir d'agrégats de niveau inférieur. Dans le $\mathcal{CNP}$-Graphe compact correspondant, on retrouve la hiérarchie d'agrégation originale à laquelle on a ajouté le ou les hôtes *client* devant recevoir l'agrégat et les hôtes pouvant calculer les agrégats vX ou X désigne le niveau d'agrégation. On a de plus enlevé les agrégats n'étant pas matérialisés (en l'occurrence 2.2).

Ainsi :
- $c\mathcal{V}$ contient le ou les clients ainsi que les hôtes hébergeant chacun des niveaux d'agrégation et ceux permettant leur calcul. On peut remarquer que ces sous-ensembles ne sont pas nécessairement distincts, en particulier les vX sont très probablement tous égaux puisqu'ils représentent de simples hôtes de calculs.
- $c\mathcal{E}$ contient touts les arcs représentant une utilisation d'agrégat.

Soit :

$$c\mathcal{CNP} = \begin{cases} c\mathcal{V} = & \{client,\ 3,\ 2.1,\ 2.1.1,\ 2.1.2,\ 2.2.1,\ 2.2.2,\ v3,\ v2.1,\ v2.2\} \\ c\mathcal{E} = & \{(client, 3),\ (client, v3),\ (v3, 2.1),\ (v3, v2.1),\ (v3, v2.2), \\ & (v2.1, 2.1.1),\ (v2.1, 2.1.2),\ (v2.2, 2.2.1),\ (v2.2, 2.2.2)\} \end{cases}$$

Il suffit alors, après avoir appliqué la fonction *decompact*, de trouver le sous-arbre minimum couvrant les niveaux nécessaires (i.e. ayant un nœud dans soit X soit vX et ses sous-niveaux) pour obtenir la solution minimisant les coûts de calcul de l'agrégat demandé.

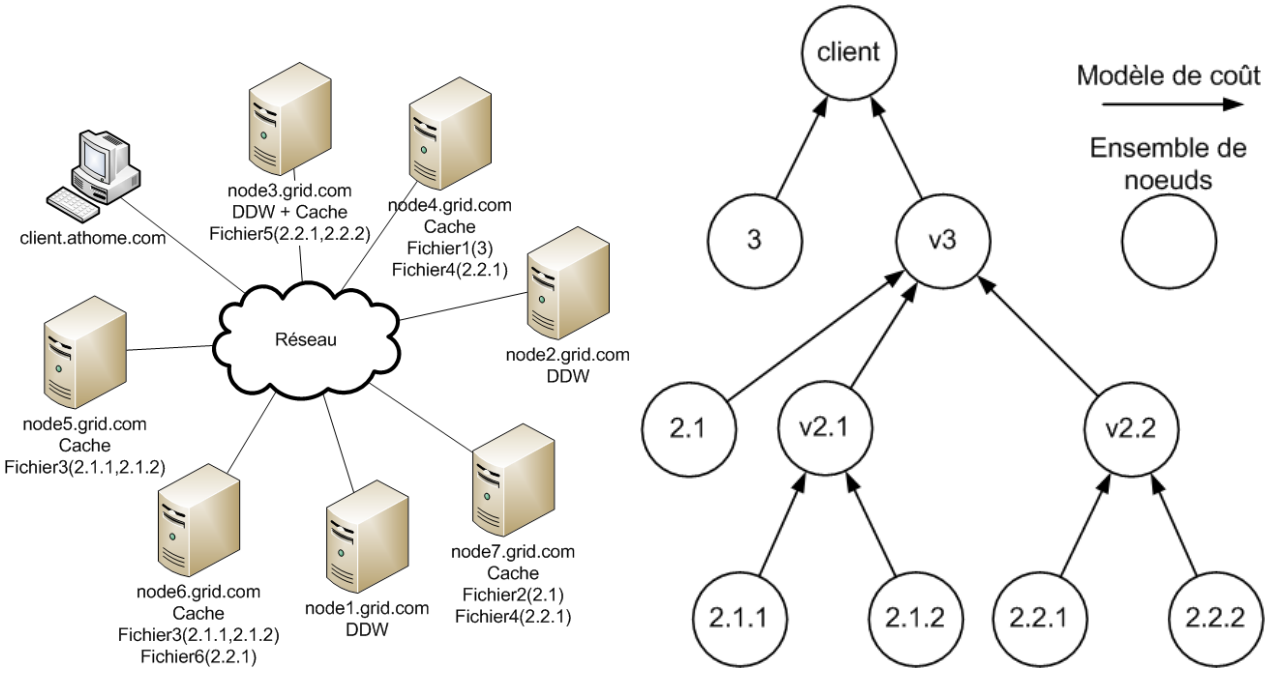


FIG. 4.6 – Illustration du problème spécifique d'agrégation de données dans un entrepôt de données distribués et $\mathcal{CNP}$-Graphe compact correspondant

4.3 Remarques et synthèse sur la modélisation des problèmes

Nous avons montré dans ce chapitre, au travers des différents exemples présentés précédemment, comment les problèmes rencontrés dans la grille peuvent être modélisés sous forme de $\mathcal{CNP}$-Graphes et leur version compacte. Cette modélisation a l'avantage d'être intuitive et donc facile à acquérir et utiliser. Certains problèmes peuvent mener à des $\mathcal{CNP}$-Graphes compacts relativement complexes, aboutissant à des $\mathcal{CNP}$-Graphes qui le sont encore plus. Mais dans la plupart des cas, le problème fait déjà l'objet d'une modélisation sous forme de graphe, qui peut être réutilisée presque directement.

Chacun des problèmes liés à la répartition des ressources présentés Chapitre 2 peut être modélisé sous forme d'un graphe. On peut également citer de façon non exhaustive d'autres représentations de problème de gestion de la répartition des ressources par des graphes :

- les bases de données : on peut représenter un modèle relationnel de données par un graphe orienté dont les sommets représentent les relations et les arcs leurs dépendances. En particulier, un schéma relationnel issu d'un processus de normalisation est désigné par « graphe sémantique normalisé ». On peut, par exemple, utiliser ce graphe pour résoudre le problème du déploiement des fragments d'une base de données distribuée.

- les ontologies : présentées comme un aspect fondamental du web sémantique, les ontologies se décrivent comme un ensemble de concepts et de relations qui sont naturellement représentées sous forme d'un graphe orienté. Ce graphe peut, par exemple, être utilisé pour optimiser la navigation dans des données distribuées décrites par une ontologie.

- le parallélisme : largement exploité dans les techniques d'optimisation d'algorithme et d'ordonnancement, le parallélisme est souvent représenté par des graphes de dépendance de flots d'instructions ou de données, lesquelles correspondent aux sommets et sont reliés par des

arcs représentant les dépendances temporelles. Par exemple, ce graphe peut être utilisé pour distribuer les différentes tâches sur les différents processeurs disponibles.

Une telle modélisation permet d'utiliser les algorithmes issus de la théorie des graphes afin de résoudre ces problèmes. L'avantage d'une telle approche est qu'elle repose sur une théorie depuis longtemps éprouvée et surtout extrêmement riche : de nombreux problèmes sont traités par de nombreux algorithmes. En effet, un problème se décline souvent en de nombreux sous-problèmes, lesquels sont résolus par plusieurs algorithmes présentant des caractéristiques différentes en terme de fonctionnalités, de complexités et de précision.

On peut également remarquer que la plupart des problèmes concrets de gestion de la répartition des ressources correspondent à des problèmes NP-difficiles. Par exemple :

- problème de localisation discrète : placer des ressources sur un réseau.
- arbre couvrant minimum ou problème du postier chinois : organiser un réseau logique (par exemple pair à pair) afin de minimiser le coût de livraison d'un message à un ensemble d'hôtes.
- *job-shop scheduling* ou *minimum makespan* : distribuer un ensemble de tâches à un ensemble de ressources afin de minimiser le temps de réalisation.
- problème du sac à dos : trouver une solution de déploiement d'un ensemble de ressources sur un ensemble d'hôtes.
- plus courts chemins : trouver un chemin de combinaison de ressource optimal.
- problème d'édition de forêts : trouver un moyen optimal d'obtenir une architecture cible à partir de l'architecture actuelle.

De plus, la modélisation des problèmes sous forme de graphes présente le grand avantage d'être facilement compréhensible et manipulable par les utilisateurs humains. Ces derniers peuvent utiliser une représentation graphique des problèmes modélisés sous forme de $\mathcal{CNP}$-Graphes afin d'identifier visuellement leurs caractéristiques.

En revanche, la résolution des problèmes s'appuie sur l'utilisation d'algorithmes de graphe exploitant les labels des arcs des $\mathcal{CNP}$-Graphes. Or ces labels sont issus de fonctions distances devant refléter le coût des interactions représentées. Dans le chapitre suivant, nous montrons comment de telles fonctions distances peuvent être construites.

5

Modélisation des coûts dans les $\mathcal{CNP}$-Graphes

L'utilisation de fonctions distances entre les nœuds des $\mathcal{CNP}$-Graphes, ou de façon plus pragmatique entre les hôtes de la grille, est au cœur de notre proposition. En effet, il convient que les labels employés par les algorithmes de résolution soient représentatifs des aspects à optimiser.

Nous avons vu dans l'introduction que de nombreux aspects doivent être reflétés en fonction du contexte, de la tâche, des choix et besoins utilisateur. Ces aspects concernent aussi bien les performances, que la sémantique. Les aspects de type sémantique tels que la fraîcheur des données ou les coûts financiers d'utilisation des ressources ne peuvent être qu'à la charge du décisionnaire et font d'ailleurs partie de leurs compétences.

En revanche, les aspects directement liés aux performances de l'infrastructure matérielle sont généralement hors de leurs préoccupations. De plus, ces aspects sont assez complexes à appréhender. C'est pourquoi dans ce chapitre, nous allons commencer par proposer une modélisation de l'infrastructure matérielle afin de la rendre aisément manipulable et de faciliter l'appréhension des notions de performances par l'utilisateur. Puis nous montrerons comment construire des fonctions distances, avec un minimum d'efforts, afin qu'elles reflètent les aspects à optimiser.

Il est important à ce stade de remarquer que notre objectif n'est pas d'élaborer des fonctions distances permettant d'obtenir une estimation de temps d'exécution. Cette approche est fort attrayante et ainsi souvent exploitée dans la littérature. En revanche, elle implique la mise en place de techniques de prédiction complexes, souvent basées sur des méthodes statistiques et de régression, rendant du même coup leur compréhension par l'utilisateur difficile et limitant la flexibilité de leur exploitation. Notre objectif étant de permettre à tout utilisateur d'utiliser ses propres modèles de coût en déclarant ses propres fonctions distance, l'utilisation de ces techniques n'est pas pertinente. De plus, la plupart des problèmes sont résolus en trouvant la meilleure solution (ou du moins une solution satisfaisante). C'est pourquoi nous avons adopté une approche basée sur le classement des différentes solutions candidates, plutôt que sur une prédiction de leur coût réel. Cette approche sera confortée dans les expérimentations, Chapitre 8, par la comparaison du classement des ressources candidates

obtenu par notre approche avec le classement basé sur les temps d'exécution réel.

5.1 Modélisation de l'infrastructure matérielle

5.1.1 Modélisation globale

Du point de vue des utilisateurs et de la superstructure logicielle, l'infrastructure matérielle est composée uniquement de machines situées en périphérie. En effet, les machines internes au réseau, telles que les routeurs, n'ont aucune importance dans la mesure où elles ne sont pas accessibles directement et totalement transparentes d'utilisation. Ainsi, nous modélisons cette infrastructure matérielle comme un ensemble de nœuds périphériques, que nous appelons hôtes. De plus, nous modélisons chaque couple d'hôtes pouvant communiquer directement par des liens logiques. Ces liens logiques ne représentent pas nécessairement un lien physique, mais plutôt les performances perçues par les applications sur l'ensemble des liens physiques entre l'hôte source et l'hôte destination.

Cette modélisation est donc un graphe $\mathcal{CN}$ (pour Computer Network) :

Définition II.4 *Modélisation d'un graphe de réseau d'ordinateurs (Computer Network)*

$$\mathcal{CN} = (\mathcal{H}, \mathcal{L}, M)$$

où

- $\mathcal{H}$ *est un ensemble fini non vide d'hôtes.*
- $\mathcal{L}$ *est un ensemble fini non fide de liens logiques. Un lien est un couple ordonné d'hôtes de* $\mathcal{H}$*.*
- M *est un ensemble de métriques servant à labéliser aussi bien les éléments de* $\mathcal{H}$ *que de* $\mathcal{L}$*.*

Dans un environnement ouvert où tout hôte peut accéder à tous les autres, le graphe $\mathcal{CN}$ est complet. En revanche, la présence de politiques de sécurité restrictives peut impliquer que certains hôtes n'aient pas accès à la totalité de la plateforme. Ces restrictions sont facilement modélisables puisqu'il suffit de ne pas inclure les arcs correspondants dans $\mathcal{L}$.

5.1.2 Modélisation des métriques de performance

Selon le *Network Monitoring Working Group* du *Global Grid Forum* dans [Lowekamp 04] :

Définition II.5 *« Une métrique est une quantité en relation avec les performances ou la fiabilité de l'Internet ».*

Nous appelons indifféremment *observation* ou *mesure* la valeur correspondante à une métrique donnée à une instant donné.

Nous avons regroupé les métriques de surveillance dans un ensemble :

Définition II.6 *Nous notons mM l'ensemble des métriques de surveillance (monitoring metrics) disponibles.*

On peut identifier deux domaines différents pour ces métriques : $mM = mM_{\mathcal{H}} \cup mM_{\mathcal{L}}$, où :

- *$mM_{\mathcal{H}}$ est l'ensemble des métriques dont le domaine est $\mathcal{H}$. Ce sont les métriques relatives aux hôtes telles que la vitesse du CPU ou l'espace disque. L'observation d'une métrique $m \in mM_{\mathcal{H}}$ pour l'hôte i est notée m_i.*
- *$mM_{\mathcal{L}}$ est l'ensemble des métriques dont le domaine est $\mathcal{L}$. Ce sont les métriques relatives aux liens logiques (c.-à-d. au réseau d'interconnexion) telles que la latence ou la bande passante. L'observation d'une métrique $m \in mM_{\mathcal{L}}$ depuis l'hôte i vers l'hôte j est notée $m_{i,j}$.*

De plus, on peut identifier deux codomaines différents pour ces métriques : $mM = mM_{\mathbb{R}} \cup mM_{\mathbb{S}}$, où :

- *$mM^{\mathbb{R}}$ est l'ensemble des métriques à valeurs dans $\mathbb{R}$.*
- *$mM^{\mathbb{S}}$ est l'ensemble des métriques à valeurs dans $\mathbb{S}$, l'ensemble des chaînes de caractères. $mM^{\mathbb{S}}$ permet notamment de représenter les métriques qualitatives telles que l'architecture matérielle ou le type du système d'exploitation.*

Ainsi, les métriques de performance que nous utilisons principalement sont :

- $CPUs \in M^{\mathbb{R}}_{\mathcal{H}}$: pour *CPUspeed*, la vitesse du processeur, la cadence fondamentale en cycles par secondes à laquelle la plus basique opération peut être exécutée, en Hertz.
- $CPUa \in M^{\mathbb{R}}_{\mathcal{H}}$: pour *CPUavailability*, la disponibilité processeur, le ratio de temps pendant lequel le processeur ne fait rien, dans l'intervalle $[0, 1]$
- $CPUarchi^{\mathbb{S}}_{\mathcal{H}}$: l'architecture du processeur, par exemple $Intel^{\text{TM}}$ou AMD^{TM}.
- $diskSpace \in M^{\mathbb{R}}_{\mathcal{H}}$: l'espace disque total en Mo.
- $freeDiskSpace \in M^{\mathbb{R}}_{\mathcal{H}}$: l'espace disque disponible en Mo.
- $L \in M^{\mathbb{R}}_{\mathcal{L}}$: la latence réseau, le temps qu'un paquet met pour atteindre la destination depuis la source, en seconde.
- $BW \in M^{\mathbb{R}}_{\mathcal{L}}$: la bande-passante réseau, la quantité de données qui peut atteindre la destination depuis la source dans un intervalle de temps, en octets par seconde.

On peut remarquer que les métriques de $L_{\mathcal{L}}$ présentent des valeurs particulières : celles correspondantes à un hôte vers lui-même[1]. Ces valeurs doivent être configurées pour chaque métrique utilisée. Par convention

$$\forall h \in \mathcal{H},\ BW_{h,h} = \infty \text{ et } L_{h,h} = 0$$

en clair, le coût d'un transfert local de données est nul.

[1] Aussi appelée *boucle locale*

Les outils de surveillance proposent très souvent plusieurs types d'observations dont les plus utiles sont l'observation actuelle (c.-à-d. la dernière observation disponible) et la prévision. Alors que les observations actuelles sont pertinentes pour des problèmes à court terme tels que la sélection de ressources, les prévisions représentent en fait les performances sur lesquelles on peut compter (et non pas l'observation dans un futur plus ou moins proche) et sont pertinentes pour les problèmes à long terme tels que le placement de ressources. Les prévisions d'une métrique $m \in M$ sont notées m'.

On peut noter ici que ces labels sont multiples, ce qui rend le graphe $\mathcal{CN}$ inexploitable par les algorithmes de graphe classiques qui fonctionnent souvent avec un label unique.

5.1.3 Modélisation des métriques composées

Une remarque importante sur les métriques présentées précédemment est qu'elles sont fort éloignées des préoccupations de la superstructure logicielle. C'est pourquoi nous avons développé des métriques composées. Celles-ci sont plus proches de ces préoccupations et donc plus facilement utilisables. Les deux tâches les plus basiques de la superstructure logicielle sont de transférer des données et d'effectuer des calculs, c'est pourquoi nous avons mis en place deux métriques composées : *Data Transfers Cost* et *Computation Task Cost* qui sont regroupées dans un ensemble nommé CM (pour *Compound Metrics*) :

Définition II.7 *Métrique composée*

Une métrique composée est une combinaison de métriques de surveillance et de métriques composées.

L'ensemble des métriques composées est noté CM.

DTC : Data Transfer Cost

La métrique composée DTC évalue le coût de transfert d'un certain volume de données d'un hôte vers un autre. Selon les concepteurs de NWS, Faerman et al., « le modèle brut de bande-passante (*raw bandwidth model* en anglais) peut être utilisé pour ordonner différents ordonnancements candidats » [Faerman 99]. Ce modèle est le suivant :

Définition II.8 *Raw Bandwidth Model*

$$\forall (hs, hd) \in \mathcal{L},\ \frac{dataSize}{BW_{hd,hs}}$$

où $dataSize$ désigne la taille de la donnée à transférer exprimée en octet.

Cependant, nous avons remarqué que sur certaines architectures matérielles, en particulier en présence de machines passerelles aux performances très limitées, ou selon les mécanismes de sécurité

mis en place, que le temps d'établissement des connexions réseau pouvait s'avérer particulièrement important. Ainsi, il se peut que latence et bande passante ne soient pas proportionnelles ni même ordonnées de la même manière : soit trois hôtes hi, hj et hk, il est possible que $BW_{hi,hj} < BW_{hi,hk}$ alors que $L_{hi,hj} > L_{hi,hk}$. Dans ce cas, le modèle brut de bande passante fausse le classement des hôtes pour les données de très petite taille pour lesquelles la latence est le critère principal.

C'est pourquoi, afin de faire apparaître la latence dans ce modèle de coût, nous avons ajouté les trois aller-retour nécessaires à l'établissement et la fermeture d'une connexion TCP/IP :

Définition II.9 *Métrique composée Data Transfer Cost*

$$\forall (hs, hd) \in \mathcal{E},\ DTC_{hs,hd}(dataSize) = 3 \times (L_{hs,hd} + L_{hd,hs}) + \frac{dataSize}{BW_{hs,hd}}$$

CTC : Computation Task Cost

CTC évalue le coût d'un calcul de taille donnée sur un hôte donné. Elle prend en compte non seulement la cadence du CPU, mais aussi sa disponibilité. Elle est construite sur le même modèle que CTC :

Définition II.10 *Métrique composée Computation Task Cost*

$$\forall h \in \mathcal{V},\ CTC_h(nb\_cycles) = \frac{nb\_cycles}{CPUs_h \times CPUa_h}$$

On peut remarquer que la quantité *nb_cycles* n'est pas toujours facile à déterminer. En effet, l'évaluation du nombre de cycles impliqués dans l'exécution d'une application dépend de nombreux paramètres tels que le nombre d'opérations unitaires et le nombre de cycles pour chaque type d'opération unitaire.

Deux approches sont exploitables pour définir cette quantité. La première consiste à utiliser des outils de profilage de code tels que *gprof* du projet *GNU*[2] qui permet de mesurer le temps passé dans chaque routine de l'application, ou encore *CodeAnalyst*™d'AMD™[3] qui permet d'émuler les processeurs AMD afin de mesurer exactement le nombre de cycles impliqués dans chaque routine en fonction des processeurs.

La deuxième approche est basée sur les techniques de calibration. En substance, cela consiste à mesurer le temps global nécessaire à l'exécution d'une application avec plusieurs ensembles de paramètres, puis à modéliser ce temps en fonction des paramètres, le plus souvent grâce à des techniques de régression. Nous discuterons cet aspect Section 10.3.

[2] http ://www.gnu.org/
[3] http ://developer.amd.com/

Remarques sur les métriques composées

La première des remarques est que les métriques composées peuvent être, sur le modèle des métriques de performances, déclinées sous forme de prédictions. Dans ce cas, les métriques brutes qui y sont représentées sont également déclinées, lorsque c'est possible, sous forme de prédiction. Par exemple

$$CTC'_h(nb\_cycles) = \frac{nb\_cycles}{CPUs_h \times CPUa'_h}$$

car si la disponibilité peut être prédite, la vitesse en revanche est une métrique statique pour laquelle cela n'est pas pertinent.

Une autre remarque concerne les caractéristiques avancées qui ne sont pas représentées. Par exemple, DTC ne fait pas apparaitre le taux de perte de paquets ou encore le nombre de hops. En réalité, ces caractéristiques sont déjà prises en compte dans les observations de latence et bande passante fournies par les outils de surveillance.

La plus importante remarque est que la finalité de ces métriques composées n'est pas d'évaluer le temps dans une unité précise, d'un transfert ou d'un calcul, ce qui s'apparenterait à de la prédiction de performance. Leur finalité est uniquement de permettre le classement des différents candidats à une même tâche. Nous montrerons aux travers des expérimentations, Chapitre 8, la pertinence de cette approche.

5.2 Modélisation des métriques sémantiques

Les métriques exposées dans la section précédente sont toutes relatives aux performances de l'infrastructure matérielle. En revanche, on peut noter l'existence de métriques d'une autre nature, relatives aux caractéristiques des ressources. Nous prenons ici le mot *sémantique* au sens large pour désigner toutes les informations qui ne sont pas directement liées aux performances. Cela comprendra donc des informations comme la fraîcheur des données ou le coût d'utilisation d'une ressource, en réalité l'ensemble des métadonnées, attributs et descripteurs des ressources. Nous conserverons le terme « *métrique* » bien qu'il ne soit pas tout à fait approprié (ces informations ne sont pas nécessairement mesurées), car ces informations sont fournies par des logiciels tiers (tels que le service de caches collaboratifs) et interviennent de la même manière que les métriques de surveillance dans notre approche. Le problème principal est que ces informations, dans le monde des grilles, n'ont pas donné lieu à une normalisation au même titre que les métriques de surveillance.

Définition II.11 *Nous notons sM (semantic Metrics) l'ensemble des métriques sémantiques.*

On ne peut identifier de domaines pour ces métriques, car le sujet de ces informations peut être n'importe quelle entité présente sur la grille. Mais on peut identifier deux codomaines différents : $sM = sM^{\mathbb{R}} \cup sM^{\mathbb{S}}$, où :

- *$sM^{\mathbb{R}}$ est l'ensemble des métriques à valeurs dans $\mathbb{R}$.*
- *$sM^{\mathbb{S}}$ est l'ensemble des métriques à valeurs dans $\mathbb{S}$, l'ensemble des chaînes de caractères.*

Nous pouvons définir quelques métriques sémantiques classiques :

- $fileSize \in sM^{\mathbb{R}} : \mathcal{S} \rightarrow \mathbb{R}$ qui associe à un identifiant de fichier, sa taille en octet.
- $queryRate \in sM^{\mathbb{R}} : \mathcal{S} \times \mathcal{H} \rightarrow \mathbb{R}$ qui associe à un identifiant de ressource et à un hôte, le taux de demandes journalier de cette ressource par cet hôte.
- $freshness \in sM^{\mathbb{R}} : \mathcal{S} \rightarrow [0, 1]$ qui associe à un identifiant de donnée, un indice de fraîcheur. Ce dernier atteint 1 lorsque la donnée correspond à la dernière mise à jour et 0 lorsqu'elle correspond à la plus ancienne version.
- $storageFinancialCost \in sM^{\mathbb{R}} : \mathcal{H} \rightarrow \mathbb{R}$ qui associe un hôte avec le coût financier d'utilisation de son espace de stockage en euros par méga-octet.
- $cpuFinancialCost \in sM^{\mathbb{R}} : \mathcal{H} \rightarrow \mathbb{R}$ qui associe un hôte avec le coût financier d'utilisation de ses capacités de calcul en euros par méga-cycles.

Pour chacune de ces métriques, nous supposons qu'un logiciel tiers est capable de nous fournir leurs valeurs. On peut noter que cette hypothèse n'est pas du tout irréaliste : la taille des données est communément fournie par les systèmes d'exploitation, la fraîcheur et le taux de demande par un hôte d'un fichier sont communément surveillés par les systèmes de caches, et les loueurs de ressources informatiques fournissent toujours leurs tarifs. Cette liste n'est pas exhaustive, elle peut être étendue de la façon la plus intuitive et la plus accessible possible, comme nous le montrerons Chapitre 7. Enfin, il est à noter que le concept d'observation actuelle / prédiction est pertinent pour certaines métriques sémantiques au même titre que pour les métriques de performance : par exemple, *queryRate* peut être déclinée en $queryRate'$ pour désigner la valeur sur laquelle on peut compter plutôt que la dernière observation.

5.3 Modélisation des propriétés de tâche

En plus des métriques de surveillance et des métriques sémantiques, une troisième catégorie de paramètres doit être prise en compte dans la modélisation des distances entre hôtes. Il s'agit des informations propres au problème modélisé et aux différentes tâches qui le compose. Nous désignons de telles informations par ensemble des propriétés de tâche, noté TPS (*Tasks Properties Set*).

Définition II.12 *TPS (Tasks Properties Set) ensemble des propriétés de tâche*

Ensemble de variables représentant les propriétés impliquées dans le problème modélisé. Chaque propriété est nommée et peut être évaluée comme une constante réelle ou une chaîne de caractères, ou encore être associées à une fonction de distribution distrib.

$$TPS = \{< tp_1[= v_1|, distrib_1, lb_1, ub_1, s_1] >, \ldots < tp_n[= v_n|, distrib_n, lb_n, ub_n, s_n] >$$

où $\forall i \in [1, n]$

- $tp_i \in \mathbb{S}$ *est le nom de la propriété*

- $v_i \in \mathbb{R}$ *est la valeur de la propriété, de type réel ou chaîne de caractères (optionnelle)*
- $distrib_i : \mathbb{R} \mapsto \mathbb{R}$ *est une fonction de distribution des valeurs réelles de la propriété, accompagnée de ses bornes inférieure* lb_i *et supérieure* ub_i*, ainsi que d'un pas* s_i *(voir Section 7.2.4 pour plus de détails) (optionnelle)*

Un exemple très simple de TPS est la taille d'une donnée. Si le problème concerne le placement ou la sélection d'une donnée précise dont la taille est 100 Mo, l'ensemble des propriétés de tâche est :

$$TPS = \{< size = 100Mo >\}$$

Si le problème concerne le placement d'un service de stockage qui contient des fichiers dont les tailles se situent entre 10 Mo et 1 Go et respectent une distribution logarithmique, avec un pas de 100Mo, l'ensemble des propriétés de tâche est :

$$TPS = \{< size,\ log,\ 10Mo,\ 1Go, 100Mo >\}$$

D'autres exemples commentés de TPS sont donnés Section 5.4.2.

5.3.1 Remarques sur les propriétés de tâche

On peut remarquer que les propriétés de tâche sont en réalité des métriques brutes au même titre que les métriques de performances ou sémantique. La différence se situe au niveau du fournisseur des observations qui dans le cas des métriques brutes est un logiciel tiers, alors que dans le cas des propriétés de tâche le fournisseur est l'utilisateur (qu'il soit humain ou logiciel).

En outre, on peut déclarer des distributions dans les propriétés de tâche, ce qui n'est pas prévu dans les métriques brutes. Nous n'avons donc pas jugé pertinent de le faire, car à notre connaissance les logiciels tiers concernés ne fournissent pas ces informations. En réalité, il est tout à fait envisageable qu'un jour des logiciels tiers fournissent ce type d'information. Leur prise en compte ne pose aucun problèmes et se résume à quelques modifications mineures de notre modélisation des métriques brutes.

Enfin, nous allons voir dans la section suivante qu'il existe une différence supplémentaire entre métrique et propriété de tâche quant à leur statut dans la création des fonctions distance.

5.4 Modélisation des fonctions distance

Un $\mathcal{CNP}$-Graphe compact ($c\mathcal{CNP}$) représente un problème lié à la répartition des ressources sous son point de vue logique. L'instanciation d'un $c\mathcal{CNP}$ en un $\mathcal{CNP}$-Graphe $\mathcal{CNP}$ représentant la réalité physique de ce problème se fait d'une part en utilisant les labels des sommets de $c\mathcal{CNP}$ pour créer les sommets de $\mathcal{CNP}$, et d'autres part en évaluant les fonctions distance de $c\mathcal{CNP}$ pour labéliser les arcs de $\mathcal{CNP}$.

Nous avons défini le concept de fonction distance comme une fonction de coût. Cette dernière est donc la fonction qui permet d'attribuer une valeur réelle que nous appelons *distance*[4] aux arcs de

[4]Aussi désignée par *poids* dans la théorie des graphes

$\mathcal{CNP}$ en vue d'une exploitation par des algorithmes. Pour ce faire, il convient d'intégrer d'une part les information de surveillance de la plateforme, lesquelles ont été modélisées sous forme d'un graphe $\mathcal{CN}$, et d'autre par les propriétés de l'interaction logique entre les entités logicielle, lesquelles ont été modélisées en tant que valeurs TPS.

Avec la déclaration des métriques de performance, des métriques sémantiques, des métriques composées et des propriétés de tâche, l'utilisateur possède tous les outils de base pour définir aisément des fonctions distances reflétant les aspects qu'il compte optimiser. Plus formellement :

Définition II.13 *Fonction Distance La fonction distance utilisée pour labéliser l'arc* $(Vs, Vd) \in c\mathcal{E}$ *d'un* $\mathcal{CNP}$*-Graphe compact est notée :*

$$df_{Vs,Vd} \in \mathcal{F}^{\mathbb{R}}_{TPS}$$

où $\mathcal{F}^{\mathbb{R}}_{TPS}$ *désigne l'ensemble des fonctions de* $tp_1 \times \ldots \times tp_n$ *dans* $\mathbb{R}$*. Ces fonctions peuvent contenir des métriques de performance et sémantiques de* M*, des métriques composées de* CM *et des propriétés de tâche de* TPS*.*

Par convention, les valeurs des fonctions distance tendent vers 0 lorsque l'aspect évalué tend vers la perfection, puisque les distances sont considérées comme des coûts et que les algorithmes tendent généralement à les minimiser.

De plus, les métriques chaînes de $\mathbb{S}$ ne peuvent être utilisées qu'incluses dans des tests (voir Section 5.4.2).

Un exemple très simple est la modélisation des coûts de transfert impliqués dans l'adaptation d'un fichier : un fournisseur *provider* envoie une fichier de taille *size* à un service d'adaptation *converter* qui renvoie un fichier de taille $size/2$ à un *client* :

$$\begin{cases} TPS = \{< size >\} \\ dfS = \{df_{provider,converter}(size) = DTC_{provider,converter}(size), \\ \qquad df_{converter,client}(size) = DTC_{converter,client}(size/2)\} \end{cases}$$

Veuillez noter que, dans la suite, l'ensemble des fonctions distance dfS n'est pas stipulé car nous ne présentons pas ici des $\mathcal{CNP}$-Graphes compacts, mais seulement des exemples de TPS et de fonctions distance, indépendamment de tout problème logique. C'est d'ailleurs une des garanties de souplesse de notre approche : la modélisation de la logique des problèmes est décorrélés de la modélisation des coûts impliqués dans ce problèmes. Ces deux modélisations peuvent donc être modifiées et réutilisées indépendamment, y compris par des acteurs différents. Ceci sera discuté Section 5.5.

5.4.1 Évaluation des fonctions distance

Afin d'attribuer les labels aux arcs d'un $\mathcal{CNP}$-Graphe, il convient d'évaluer les fonctions distances du $c\mathcal{CNP}$-Graphe correspondant.

Le label de l'arc $(Vs.hs, Vd.hd) \in Vs \times Vd$ est calculé en instanciant $df_{Vs,Vd}$.

Définition II.14 *Instanciation des fonctions distances*

L'instanciation de $df_{Vs,Vd}$ pour l'arc $(Vs.hs, Vd.hd) \in Vs \times Vd$ est notée :

$$df_{Vs.hs,Vd.hs} : tp_1 \times \ldots \times tp_n \rightarrow \mathbb{R}$$

Par exemple, les instanciations des fonctions distance montrées précédemment sont :

$$df_{provider.hs,converter.hd}(size) = DTC_{hs,hd}(size)$$

$$df_{converter.hs,client.hd}(size) = DTC_{hs,hd}(size/2)$$

Le label d'un arc $(Vs.hs, Vd.hd) \in Vs \times Vd$ est noté $d(Vs.hs, Vd.hd)$ et correspond à l'évaluation de $df_{Vs.hs,Vd.hd}$ en tenant compte des valeurs effectives des variables déclarée dans TPS.

Cette évaluation peut se faire selon deux modes distincts : pour les problèmes à court terme et ceux à long terme.

Évaluation pour les problèmes à court terme

Pour les problèmes à court terme, tels qu'une sélection ou une composition de ressources pour une exécution immédiate, la dernière observation des métriques sera utilisée ainsi que les valeurs déclarée des propriétés de tâche.

$$d(Vs.hs, Vd.hd) = df_{Vs.hs,Vd.hd}(v_1, \ldots, v_n)$$

Par exemple :

$$\begin{cases} TPS \ni < size = 100Mo > \\ df_{provider,converter}(size) = DTC_{provider,converter}(size) \end{cases}$$

permettra de labéliser les arcs $(provider.hs, converter.hd) \in provider \times converter$ par :

$$d(provider.hs, converter.hd) = DTC_{hs,hd}(100Mo)$$

Évaluation pour les problèmes à long terme

Pour les problèmes à long terme, tels qu'un placement de ressource, la prédiction des métriques sera utilisée ainsi que l'intégration des valeurs des propriétés de tâche en fonction des déclaration relatives à leur distribution :

$$d(Vd.hs, Vd.hd) = \int_{lb_1}^{ub_1} \dots \int_{lb_n}^{ub_n} \left[\left(\prod_{k=1}^{n} distrib_k(tp_k) \right) \times df_{Vs.hs,Vd.hs}(tp_1, \dots tp_n) \right] dtp_1 \dots dtp_n$$

Par exemple :

$$\begin{cases} TPS \ni < size,\ log,\ 10Mo,\ 1Go, 100Mo > \\ df_{provider,converter}(size) = DTC_{provider,converter}(size) \end{cases}$$

permettra de labéliser les arcs $(provider.hs, converter.hd) \in provider \times converter$ par :

$$d(provider.hs, converter.hd) = \int_{10Mo}^{1Go} \left[log(size) \times DTC'_{hs,hd}(size) \right] \ dsize$$

Le calcul de cette intégrale sera détaillé Section 7.2.4.

5.4.2 Exemples de propriétés de tâche et de fonctions distance

Afin de mieux fixer les idées, voici quelques exemples classiques de propriétés de tâche et de fonctions distance.

Transfert d'une donnée d'une source vers une destination pour minimiser le temps de transfert

Dans le cas d'un transfert de donnée et lorsqu'on ne s'intéresse qu'aux performances, la seule propriété de tâche est la taille des données transférées, notée *dataSize*. La donnée étant transférée de la source vers la destination, on peut utiliser directement la métrique composée $DataTransferCost$ comme suit :

$$\begin{cases} TPS \ni < dataSize > \\ df_{Vs,Vd}(dataSize) = DTC_{Vs,Vd}(dataSize) \end{cases}$$

Transfert d'une donnée d'une source vers une destination pour optimiser l'équilibrage de l'espace disque

Lorsqu'on s'intéresse à l'équilibrage des charges pour stocker une donnée, on utilise les métriques $freeDiskSpace$ et *diskSpace* pour calculer le pourcentage d'espace libre. Les arcs vers les hôtes ne disposant pas assez d'espace sont labélisés avec la valeur infinie, les autres avec l'inverse de l'espace disque disponible. Ainsi, les hôtes présentant le pourcentage d'espace disque libre le plus élevé seront destination des arcs les plus courts :

$$\begin{cases} TPS \ni < dataSize > \\ df_{Vs,Vd}(dataSize) = \begin{cases} \dfrac{freeDiskSpace_{Vd} - dataSize}{diskSpace_{Vd}} & \text{si } freeDiskSpace_{Vd} > dataSize \\ \infty & \text{sinon} \end{cases} \end{cases}$$

Transfert d'un fichier d'une source vers une destination pour minimiser le temps de transfert et la fraîcheur, à partir de son identifiant

Il est possible de prendre en compte plusieurs aspects dans une même distance. Par exemple, lors de la récupération d'un fichier désigné par l'identifiant $fileId$, on peut vouloir sélectionner les réplicas en fonction des temps de transfert, mais en privilégiant les versions les plus récentes. On combine alors la métrique composée $DataTransferCost$ et la métrique sémantique $freshness$, considérée ici comme un *malus* arbitraire : lorsque l'indice de fraîcheur est au plus bas (0), la distance est doublée.

$$\begin{cases} TPS \ni < fileId > \\ df_{Vs,Vd}(fileId) = DTC_{Vd,Vs}(fileSize(fileId)) * (2 - freshness(fileId, Vd)) \end{cases}$$

On peut également considérer la fraîcheur comme métrique principale et le coût du transfert comme un modificateur de celle-ci, si le besoin en est.

Invocation d'un service par une source sur une destination pour minimiser le temps d'exécution

L'invocation d'un service comporte trois parties : l'envoi du client vers le service des données à traiter de taille *input*, leur traitement nécessitant $nCycles$ opérations élémentaires de CPU, puis le retour des résultats de taille *output* du service vers le client. Il faut donc combiner les métriques composées $DataTransferCost$ pour les transferts aller et retour et $ComputationTaskCost$ pour le traitement par le service :

$$\begin{cases} TPS \supseteq \{< input >, < nCycles >, < output >\} \\ df_{Vs,Vd}(input, nCycles, output) = DTC_{Vs,Vd}(input) + CTC_{Vd}(nCycles) + DTC_{Vd,Vs}(output) \end{cases}$$

Invocation d'un service par une source sur une destination en sélectionnant un type d'architecture

On peut utiliser les métriques chaînes de $\mathbb{S}$, telles que l'architecture du processeur, dans des tests :

$$\begin{cases} TPS \supseteq \{< input >, < cycles >, < output >\} \\ df_{Vs,Vd}(input, cycles, output) \\ \qquad = \begin{cases} DTC_{Vs,Vd}(input) + CTC_{Vd}(cycles) + DTC_{Vd,Vs}(output) & \text{si } CPUarchi = \text{“}Intel\text{”} \\ \infty & \text{sinon} \end{cases} \end{cases}$$

Invocation d'un service par une source sur une destination pour minimiser le coût financier

On peut également vouloir minimiser le coût financier d'utilisation d'un service. Dans ce cas, on utilisera la métrique sémantique *cpuFinancialCost* qui permettra d'obtenir des arcs d'autant plus longs que le coût financier sera élevé.

$$\begin{cases} TPS \ni < cycles > \\ df_{Vs,Vd}(cycles) = cpuFinancialCost(Vd) \times cycles.10^{-6} \end{cases}$$

On peut également combiner les deux derniers exemples afin de trouver une solution d'équilibre entre les performances et les coûts financiers, à la manière de la combinaison entre fraîcheur et temps de transfert.

5.5 Discussion

On peut remarquer que, lors du processus de déclaration des $c\mathcal{NP}$-Graphes, de nombreuses tâches sont laissées à la discrétion de l'utilisateur : déclaration des nœuds, des propriétés de tâches et des fonctions distance. Ceci confère une grande souplesse d'utilisation à notre proposition qui peut ainsi être employée dans de nombreux contextes et à de nombreuses fins. Il faut noter que ces tâches sont proches des préoccupations des utilisateurs et ne représentent donc pas une surcharge de travail ou des efforts supplémentaires pour l'utilisateur. À l'inverse, la manipulation des performances de l'infrastructure matérielle, qui est l'aspect le plus complexe à appréhender depuis la superstructure logicielle, est rendue accessible grâce au système de métriques composées. De plus, l'ensemble des métriques composées peut être agrémenté d'autant de métriques que nécessaire.

Ainsi, un côté important de notre approche est que notre proposition sert de lien entre les différents acteurs de la superstructure logicielle. En effet, les développeurs peuvent déclarer les métriques composées pertinentes pour leurs applications afin que les administrateurs les emploient lors du déploiement et de la gestion de ces applications. Les concepteurs des applications peuvent également proposer des métriques que les développeurs pourront ensuite employer avec un effort limité. Les administrateurs pourront de leur côté déclarer des métriques adaptées aux environnements dont ils ont la charge afin que les concepteurs et développeurs puissent facilement adapter leurs applications aux environnements cibles. Enfin, une métrique composée telle que le coût d'exécution d'une application pourra être raffinée, ou déclinée en plusieurs versions. De plus, ces modifications peuvent se faire de

façon transparente : si la métrique modifiée possède le même nom que la métrique originale, ses utilisateurs n'auront pas besoin d'en être avertis, tout en accomplissant au niveau de la superstructure logicielle un réel niveau d'adaptation à l'infrastructure matérielle.

5.6 Synthèse sur la modélisation des coûts dans les $\mathcal{CNP}$-Graphes

La modélisation des fonctions distance et leur transformation en distances réelle labélisant les arcs conclue la définition des $\mathcal{CNP}$-Graphes. Tous les outils sont maintenant en place afin de mener à bien le processus de résolution des problèmes liés à la répartition des ressources dans un environnement de grille : un problème peut être modélisé sous forme de $\mathcal{CNP}$-Graphe compact, lequel, en intégrant les informations de surveillance de la plateforme modélisées dans un graphe $\mathcal{CN}$, peut être instancié en un $\mathcal{CNP}$-Graphe modélisant la réalité physique de problème.

L'étape suivant consiste à exploiter ce $\mathcal{CNP}$-Graphe dans un algorithme afin d'en extraire les solutions au problème modélisé.

6

Exploitation algorithmique des $\mathcal{CNP}$-Graphes

Nous avons montré dans la partie précédente comment les problèmes liés à la répartition des ressources sur une grille pouvaient être modélisés sous forme de $\mathcal{CNP}$-Graphes. Une fois un tel graphe créé, la méthode la plus naturelle pour résoudre effectivement les problèmes est d'utilisation des algorithmes de graphes. Ceux-ci peuvent dans certains cas être extrêmement simples, comme lors de la résolution d'un problème de sélection simple : il suffit alors d'une simple recherche de l'arc de poids minimum. Dans d'autres cas, ils peuvent s'avérer plus compliqués. Parmi les plus compliqués, on trouve les problèmes liés au déploiement de ressources. Ces problèmes trouvent leurs solutions dans la théorie de la localisation discrète.

Une particularité de notre approche est qu'elle permet à l'utilisateur d'exploiter l'algorithme qui correspond le mieux à ses besoins pour résoudre son problème. Ceci implique une phase de sélection de cet algorithme par l'utilisateur. Les deux paramètres principaux qui doivent être pris en compte sont d'un côté la complexité, qui conditionne le temps d'exécution, d'un autre le facteur d'approximation, qui conditionne la précision des solutions obtenues.

Par exemple, face à un problème de taille modérée mais crucial au fonctionnement du système, on sélectionnera un algorithme exact mais gourmand en ressources. *A contrario*, pour un problème de grande taille mais peu sensible, on sélectionnera un algorithme d'approximation peu gourmand en ressources. Á noter ici que la notion de « gourmand » est relative aux performances de l'infrastructure matérielle au temps d'exécution de l'algorithme, et peut être quantifiée grâce à notre proposition. Cet compromis précision/coût relève strictement des préférences utilisateur et nous ne pouvons proposer plus d'aide que la quantification du temps d'exécution de l'algorithme en fonction de sa complexité.

En revanche, les algorithmes possèdent également des contraintes relatives aux caractéristiques du graphe sur lequel ils sont exécutés. Ces caractéristiques sont en général présentées en tant qu'hypothèses de l'algorithme. La vérification de ces caractéristiques incombe à notre système puisque notre but est de fournir un environnement de gestion de distribution des ressources avec un minimum d'effort utilisateur. Ce dernier, pour pouvoir utiliser l'algorithme de son choix, doit donc pouvoir facilement

vérifier que cet algorithme est valide sur le $\mathcal{CNP}$-Graphe cible.

Ces caractéristiques sont relatives à deux aspects des graphes.

Le premier aspect concerne la classe du graphe : orienté, complet, arbre, forêt, treillis, digraphe, etc. Ainsi que certaines statistiques telles que le nombre de nœuds, leur degré, le diamètre, etc. Ces aspects ont été largement étudiés dans la théorie des graphes et ne nécessitent donc pas d'implication supplémentaire.

Le deuxième aspect concerne les caractéristiques des labels des arcs. Cet aspect n'a pas été étudié dans la littérature car ces labels sont souvent représentatifs d'un problème issu de notre réalité physique, et possèdent donc des propriétés naturelles non discutée. Par exemple, les algorithmes issus de la théorie de la localisation discrète prennent en général comme cas d'usage le placement de bâtiments, tels que des entrepôts ou des casernes de pompier, dans le but de réduire la distance kilométrique qui les séparent de leurs clients. Cette distance kilométrique est donc utilisée pour labéliser les arcs et possèdent naturellement des propriétés que les concepteurs des algorithmes sont amenés à exploiter. Dans le cadre de notre proposition, les distances sont comprises sur un réseau, et ne sont donc pas soumises aux mêmes propriétés. De plus, les utilisateurs sont en capacité de déclarer un large panel de fonctions distance, sans contraintes particulière. Ces dernières peuvent être très variées. Ainsi, certains algorithmes peuvent être inappropriés à leur exploitation.

Dans un premier temps, nous allons nous intéresser aux algorithmes de résolution des problèmes de placement en Section 6.1. Dans cette section, nous montrerons qu'il existe différentes classes de problèmes dont la pertinence dépend des objectifs de l'utilisateur. Nous nous intéresserons de plus près au problème des k-médians et nous montrerons qu'il existe pour un même problème, plusieurs classes d'algorithmes dépendant des propriétés des labels.

Dans un second temps, en Section 6.2, nous détaillons notre approche permettant de quantifier la satisfaction de ces propriétés afin de permettre à l'utilisateur de sélectionner la classe d'algorithme la plus appropriée à la résolution de son problème.

6.1 Algorithmes de résolution des problèmes de déploiement

Un des problèmes génériques auxquels nous nous sommes particulièrement intéressés est celui du placement de ressources. Ce problème trouve ses principales solutions issues de la théorie des graphes dans la théorie de la localisation discrète. Cette théorie est composée de trois problèmes primaires, en anglais : *clustering*, *Uncapacitated Facility Location* ou UFL et *Quadratic Assignment Problem* ou QAP.

Clustering vise à partitionner les nœuds en exactement k groupes minimisant un critère donné. L'ensemble des centroïdes de ces groupes représente la solution au problème de placement.

Uncapacitated Facility Location (UFL) utilise des coûts associés au déploiement de nouvelles ressources et vise à définir le nombre de ressources ainsi que leur position dans l'objectif de minimiser le coût de déploiement et la qualité de service client.

Quadratic Assignment Problem (QAP) qui avec k ressources et k nœuds, assigne chaque res-

source à un nœud dans le but de minimiser les flux entre elles.

On peut ainsi remarquer que UFL est pertinent lorsque les déploiements sont très coûteux face à l'utilisation des ressources. Il correspond donc à un problème de placement de ressources au temps d'exécution pour une utilisation ponctuelle impliquant relativement peu de communications. Par exemple le déploiement d'un ensemble de services fortement découplés pour une seule utilisation.

QAP quant à lui est pertinent lorsque l'accent doit être mis sur les communications entre ressources. C'est le cas de services fortement couplés, par exemple dans une architecture dédiée à l'adaptation de contenus multimédias. En revanche, on peut noter un certain manque de souplesse, le nombre de nœuds et de ressources devant être identique. Ce manque de souplesse est d'ailleurs confirmé par les algorithmes de résolution de QAP qui sont bien souvent basés sur la programmation linéaire, technique très efficace mais assez difficile à contrôler ou régler.

Le problème de clustering par contre est très vaste et très souple : il peut aisément être adapté grâce au critère qui peut être défini en fonction des besoins.

6.1.1 Problème du clustering

Le problème du clustering peut être formalisé comme suit :

Définition II.15 *Problème de clustering*

Soit $\mathcal{V}$ un ensemble de points d'un espace équipé d'une fonction distance df, partitionner $\mathcal{V}$ en k clusters distincts $C_1, \ldots, C_k$ et déterminer ses centroides $\mu = \{\mu_1, \ldots, \mu_k\} \subset V$ tels qu'une métrique « critere » est minimisée.

Dans notre cas, k est le nombre d'instances de la ressource[1] à placer, tandis que les centroïdes $\mu_1, \ldots, \mu_k$ sont leur position optimale.

Il comporte ainsi plusieurs variantes dont les plus populaires sont décrites ci-après.

- *k-median* ou *min-sum clustering* vise à définir exactement k positions afin de minimiser la somme de la distance entre des clients et leur ressource la plus proche. Dans notre contexte, cela revient à optimiser globalement l'aspect évalué par la fonction distance.

$$critere = \sum_{i \in \mathcal{V}} \min_{j=1}^{k} d(i, \mu_j)$$

- *k-center* ou *min-max clustering* vise à définir exactement k positions afin de minimiser le maximum de la distance entre les clients et leur ressource la plus proche, soit le diamètre maximal des clusters. Dans notre contexte, cela revient à éviter qu'un client soit lésé par rapport à l'aspect évalué par la fonction distance.

[1] Le terme ressource est à prendre ici à son sens large, il peut s'agir également des tâches d'un programme parallèle

$$critere = \max_{i \in \mathcal{V}} \min_{j=1}^{k} d(i, \mu_j)$$

Ces deux critères sont basés sur la qualité de service client, mais on peut également imaginer des critères basés sur l'équilibrage des charges des ressources.

- *min-var clustering* vise à définir exactement k positions et les membres des clusters, notés $C_1, \ldots, C_k$ afin de minimiser la variance de la somme des distances entre des clients et leur ressource attitrée dans le cluster. Dans notre contexte, cela revient à équilibrer l'aspect évalué par la fonction distance sur l'ensemble des ressources.

$$critere = var_{j=1}^{k} \sum_{i \in C_j} d(i, \mu_j)$$

Nous nous sommes concentrés sur le problème des k-médians, car il est celui qui représente le mieux les besoins classiques exprimés lors du déploiement d'une ressource : optimiser la qualité de service client, tout en minimisant la charge globale des ressources. Nous nous sommes donc intéressés aux différents algorithmes résolvant ce problème.

6.1.2 Le problème des k-médians et les propriétés des labels

Reese est l'auteur d'une bibliographie annotée [Reese 06] mise à jour chaque année sur les différents travaux menés sur ce thème. Avec plus de 120 citations, on peut mesurer l'étendue des solutions disponibles pour ce problème. Nous avons pu constater dans une grande majorité de ces travaux (comme dans beaucoup d'autres dans le champ scientifique des graphes) l'hypothèse récurrente que les nœuds des graphes se situent dans un espace euclidien doté d'une distance euclidienne (aussi appelés espace métrique et distance métrique).

Ces espaces, et leur distance associée d, sont caractérisés par quatre propriétés :

Soit E un ensemble de couples de sommets labélisés avec une fonction distance d,

Nonnégativité d *nonnegative* $\Leftrightarrow \ \forall (i,j) \in \mathcal{E},\ d(i,j) \geq 0$

Séparabilité : d *separability* $\Leftrightarrow \forall (i,i) \in \mathcal{E},\ d(i,i) = 0$

Symétrie : d *symmetric* $\Leftrightarrow \forall (i,j) \in \mathcal{E},\ d(i,j) = d(j,i)$

Inégalité triangulaire : d *triangle* $\ \Leftrightarrow \forall (i,j) \in \mathcal{V},\ \forall\ k \in \mathcal{E},\ d(i,j) \leq d(i,k) + d(k,j)$

En réalité, il est presque impossible de concevoir des algorithmes efficaces sans heuristiques, lesquelles doivent se baser sur des hypothèses respectées par les données d'entrée. La popularité de l'espace euclidien est expliquée par son intuitivité et sa satisfaction dans notre monde physique. Comme les algorithmes sont à l'origine conçus pour résoudre les problèmes de ce monde, il est naturel que ces hypothèses soient exploitées.

De plus, on peut remarquer que ces hypothèses sont employées dans la description des problèmes aussi bien que dans le déroulement et les preuves des algorithmes. Aussi, l'impact de leur insatisfaction

ne peut être déterminé que par une étude précise et spécifique de chaque algorithme en particulier. En règle générale, elle se traduit par une perte de la précision des solutions obtenues, mais elle peut également impacter les temps d'exécution.

Il est nécessaire avant d'aller plus loin dans notre réflexion, d'analyser dans ces hypothèses et leur exploitation dans les algorithmes de graphe.

- La nonnégativité est très souvent employée et peu discutée.
- La séparabilité, de même, est peu discutée car elle implique la présence de boucles dans le graphe, ce qui est souvent négligé.
- La symétrie, en revanche, est souvent discutée car elle mène à différentes classes de problèmes et algorithmes. Ceci s'explique par le fait qu'elle peut être présente dans certains problèmes de notre monde physique, l'exemple classique étant la durée nécessaire pour se déplacer d'un point à un autre qui n'est pas nécessairement symétrique de par le relief (monté dans un sens, descente dans l'autre) ou par le trafic (sens des départs en vacances).
- L'inégalité triangulaire mène rarement à différentes classes d'algorithmes. Par contre elle sert souvent de base aux heuristiques. Dans la plupart des cas, elle doit donc être satisfaite.

Dans le cas particulier du problème des k-médians, nonnégativité et réflexité sont effectivement peu (voire pas) citées. En revanche, l'inégalité triangulaire est absolument obligatoire dans le sens où si elle n'est pas satisfaite « même décider si l'optimum est fini est NP-difficile » [Archer 01]. La symétrie, quant à elle, mène à deux classes d'algorithmes : la classe symétrique, traitée par exemple dans les travaux de Jain et Vazirani dans [Jain 99], Bartal, Charikar dans [Bartal 01] ou encore Ostrovsky et Rabani dans [Ostrovsky 02] ; et la classe asymétrique, traitée par exemple par Archer dans [Archer 01], Gortz et Wirth dans [Gortz 06] ou encore Panigrahy et Vishwanathan dans [Panigrahy 98].

Il est donc nécessaire de pouvoir qualifier les distances des CNP-Graphes afin de pouvoir sélectionner les algorithmes applicables.

Il est également nécessaire de spécifier que quelle que soit la satisfaction des propriétés des distances et des graphes, il est toujours possible de résoudre n'importe quel problème par un algorithme *trivial* (aussi désigné par *force brutale*). Ce type d'algorithmes énumère et compare toutes les solutions candidates au problème donné. Leur complexité est généralement exponentielle, en particulier pour les problèmes NP-difficiles. En revanche, leur mise en œuvre est aisée, ce qui leur confère une grande souplesse d'utilisation, et les solutions obtenues sont exhaustives, ce qui permet non seulement de sélectionner la ou les meilleures, mais en plus de les situer dans l'espace des solutions.

Enfin, les problèmes rencontrés sur les réseaux sont bien souvent de taille modeste comparés aux problèmes cibles de la théorie des graphes (qui se chiffrent souvent en millions de solutions). Ainsi, l'utilisation d'heuristiques complexes n'est réellement utile que dans des cas très précis caractérisés par une fréquence d'utilisation élevée plutôt que par une grande taille. L'utilisation d'algorithmes triviaux ne doit donc pas être négligée.

6.2 Satisfaction des propriétés euclidiennes

6.2.1 Satisfaction des propriétés euclidiennes dans les réseaux IP

Dans une première approche, nous avons évalué dans quelle mesure les propriétés des distances euclidiennes pouvaient être satisfaites par les métriques de surveillance dans les réseaux IP, en particulier les métriques réseau, qui par définition concernent un couple d'hôtes (ou points du réseau).

- la nonnégativité peut être considérée comme satisfaite, aucune métrique à notre connaissance ne présentant de valeurs négatives.
- la séparabilité concerne la distance d'un point à lui-même, et ne concerne donc pas directement les métriques de surveillance.
- la symétrie en revanche a été étudiée à plusieurs reprises et sous plusieurs angles dans les réseaux IP. Une première remarque concerne les connexions asymétriques, telles que l'ADSL (*Asymmetric Digital Subscriber Line*), qui par définition ne la respectent pas. De plus, He dans « How Asymmetric is Internet Routing ? A Systematic Approach » [He 05] montre lors d'expérience à échelle mondiale qu'une large majorité des routes Internet sont asymétriques. On peut donc difficilement considérer que les métriques réseau soient symétriques. Cela n'exclut pas qu'elles le soient sur certaines infrastructures matérielles, par exemple avec des liens de communications symétriques, privés, non partagés et peu utilisés (ce qui représente beaucoup de conditions).
- l'inégalité triangulaire est également difficile à satisfaire dans les réseaux IP de par le paradigme de « *best effort* », qui implique que les paquets ne soient pas routés nécessairement de façon optimale. De plus, ce routage est bien souvent basé sur le nombre de sauts (*hops*) entre sources et destinations, et ne reflète donc pas toujours les performances perçues par les applications. Par exemple si nous avons une route optimale entre le point A et le point B ainsi qu'entre le point B et le point C, mais que la route de A à C n'est pas optimale ou encore subit une saturation ou une panne, il se peut que l'inégalité triangulaire ne soit pas satisfaite ($d(A,B) + d(B,C) < d(A,C)$).

6.2.2 Satisfaction des propriétés euclidiennes dans les $\mathcal{CNP}$-Graphes

Une première approche pour quantifier la satisfaction des propriétés euclidienne dans les $\mathcal{CNP}$-Graphes consiste à adopter une approche formelle en modélisant la satisfaction des propriétés pour chacun des composants potentiels des fonctions distances, ainsi que la transmission de ces propriétés pour chacune des opérations de combinaison disponibles. Outre l'ampleur de la tâche, cette approche présente un manque important de souplesse, par exemple lors d'ajout de nouvelles métriques, qui la rend inadéquate à notre contexte.

De plus, une propriété peut être exploitée si elle est *suffisamment* satisfaite : par exemple, une *légère* insatisfaction peut mener à une perte de précision des solutions proposées par un algorithme, en revanche ce dernier peut présenter un gain de performance assez important pour justifier son utilisation. Il convient donc de définir des indices de satisfaction pour chaque propriété. L'utilisateur

peut ainsi se baser sur ces indices afin de déterminer les pertes potentielles de précision, de les comparer aux gains de performances et ainsi de faire ses propres choix en pleine connaissance des faits.

De plus, en fonction de leur utilisation, les propriétés peuvent n'être satisfaites qu'*approximativement*. Par exemple, l'imprécision des mesures rend la stricte satisfaction de la symétrie impossible à obtenir en pratique : il semble irréaliste d'espérer obtenir exactement la même observation dans les deux sens d'une même communication même lorsque les routes et performance des liens sont symétriques.

Enfin, la déclaration des fonctions distance est laissée entièrement libre. Ainsi, il est possible d'obtenir n'importe quelle valeur de label sur les arcs des $\mathcal{CNP}$-Graphes.

C'est pourquoi nous avons décidé d'adopter une approche numérique qui consiste à quantifier la satisfaction des contraintes en se basant directement sur les valeurs effectives des distances entre nœuds des $\mathcal{CNP}$-Graphes après leur calcul.

Définition II.16 *Indices de satisfaction des propriétés euclidiennes*

Soit d la fonction réelle labélisant les arcs d'un $\mathcal{CNP}$-Graphe $(\mathcal{V}, \mathcal{E})$

$$\forall (i,i) \in \mathcal{E}, \quad separability_d(i,i) = \begin{cases} 1 & si\ d(i,i) = 0 \\ 0 & sinon \end{cases}$$

$$\forall (i,j) \in \mathcal{E}, \quad nonnegativity_d(i,j) = \begin{cases} 1 & si\ d(i,j) \geq 0 \\ 0 & sinon \end{cases}$$

$$\forall (i,j) \in \mathcal{E}, \quad symmetry_d(i,j) = \begin{cases} 1 & si\ d(i,j) = d(j,i) \\ 1 - \dfrac{|d(i,j) - d(j,i)|}{|d(i,j)| + |d(j,i)|} & sinon \end{cases}$$

$$\forall (i,j) \in \mathcal{E}, \quad triangle_d(i,j) = \begin{cases} 1 & si\ card(\mathcal{E}) < 2 \\ \dfrac{card(\mathcal{E}^t_{i,j})}{card(\mathcal{E})} & sinon \end{cases}$$

$$où\ \mathcal{E}^t_{i,j} = \{k \in \mathcal{E}\ /\ d(i,j) \leq d(i,k) + d(k,j)\}$$
$$et\ \forall i \in \mathcal{V},\ \forall j \in \mathcal{V},\ d(i,j) = \infty\ si\ (i,j) \notin \mathcal{E}$$

Une première remarque concerne $separability_d$ qui aurait pu être raffinée afin de rendre compte plus précisément du niveau de la satisfaction de la séparabilité. Mais cela est superflu dans le sens où les algorithmes qui l'exploite considère en fait des graphes sans boucle, ce qui peut être obtenu avec une simple précaution dans la déclaration des c$\mathcal{CNP}$-Graphes ou encore en élaguant directement les boucles des $\mathcal{CNP}$-Graphes. De plus, ces indices sont assez simples à appréhender, ce qui les rend faciles à manipuler et aisément extensibles dans les cas spécifiques où d'autres propriétés s'avèrent nécessaires. Les indices de satisfaction *separability* et *nonegativity* sont en effet vraiment simples et ne seront pas discutés plus en détail. $triangle_d(i,j)$ est en réalité assez simple également puisque qu'il

s'agit du nombre de sommets intercalables entre i et j tels que l'inégalité triangulaire est respectée divisé par le nombre de sommets intercalables total. En revanche, nous avons vérifié la pertinence de *symmetry*, c.-à-d. que nous avons démontré que *symmetry* tendait bien vers 1 lorsque les distances étaient symétriques et vers 0 dans le cas contraire.

Propriété 1 *limites de* $symmetry_d$

$$\lim_{\Delta \to 0} symmetry_d(i,j) \to 1, \quad \lim_{\Delta \to +-\infty} symmetry_d(i,j) \to 0$$

Démonstration 1 *de la propriété des limites de* $symmetry_d$

Supposons $d(j,i) = d(i,j) + \Delta$. *Si* $\Delta \neq 0$ *alors :*

$$symmetry_d(i,j) = 1 - \frac{|d(i,j) - (d(i,j) + \Delta)|}{|d(i,j)| + |d(i,j) + \Delta|} = 1 - \frac{|\Delta|}{|d(i,j)| + |d(i,j) + \Delta|}$$

Si $\big(d(i,j) \geq 0$ *et* $d(i,j) + \Delta \leq 0\big)$ *ou* $\big(d(i,j) \leq 0$ *et* $d(i,j) + \Delta \geq 0\big)$ *alors :*

$$symmetry_d(i,j) = 1 - \frac{|\Delta|}{|\Delta|} = 0$$

sinon :

$$symmetry_d(i,j) = 1 - \frac{|\Delta|}{|2 \times d(i,j) + \Delta|}$$

et donc

$$\lim_{\Delta \to 0} symmetry_d(i,j) \to 1, \quad \lim_{\Delta \to +-\infty} symmetry_d(i,j) \to 0$$

Ces indices permettent d'évaluer la satisfaction des propriétés de l'espace euclidien entre deux nœuds des $\mathcal{CNP}$-Graphes. On peut donc calculer une large variété d'indicateurs de satisfaction globale sur l'ensemble des arcs présents dans les $\mathcal{CNP}$-Graphes (minimum, maximum, moyenne, variance, etc.), mais aussi, si un seuil de satisfaction acceptable est défini pour un algorithme donné, le nombre d'arcs satisfaisant ce seuil. Ainsi, l'utilisateur est équipé d'indices lui permettant de valider les $\mathcal{CNP}$-Graphes en fonction de leurs hypothèses. Ces indices représentent donc une aide substantielle dans le processus de sélection de l'algorithme adéquat à la résolution des problèmes.

6.3 Optimisation

Dans les sections précédentes, nous avons exposé l'importance des propriétés de graphe dans l'exploitation des algorithmes. Il s'avère que bon nombre de problèmes proviennent de la présence d'hôtes dans plusieurs ensembles lors de la déclaration des $\mathcal{CNP}$-Graphes. Cela implique la présence de boucles et de couples d'arcs opposés (la destination de l'une étant la source de l'autre et *vice versa*). Ces types d'arc sont les principales causes de non-satisfaction des propriétés de l'espace euclidien, en particulier dans les $\mathcal{CNP}$-Graphes simples, comme le montre l'exemple suivant.

Soit le $\mathcal{CNP}$-Graphe compact :

$$\begin{cases} c\mathcal{V} = & C = \{h1, h2, h3\} \cup R = \{h1, h2, h3\} \\ c\mathcal{E} = & C \times R \end{cases}$$

Le $\mathcal{CNP}$-Graphe correspondant est montré Figure 6.1 :

$$\begin{cases} \mathcal{V} = & \{h1, h2, h3\} \\ \mathcal{E} = & \{(h1, h1), (h1, h2), (h1, h3), \\ & (h2, h1), (h2, h2), (h2, h3), \\ & (h3, h1), (h3, h2), (h3, h3)\} \end{cases}$$

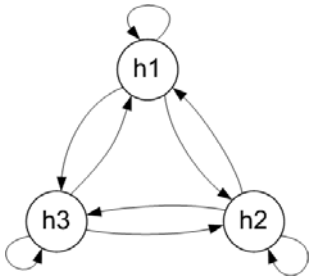


FIG. 6.1 – « Dessine-moi un $\mathcal{CNP}$-Graphe frisé comme un mouton »

On peut donc noter trois boucles (hi, hi) qui menacent la séparabilité et six couples d'arcs opposés (hi, hj) et (hj, hi) qui menacent la symétrie, qui engendrent ainsi une multitude de cycles (hi, hj), (hj, hk) et (hk, hi) menaçant l'inégalité triangulaire (et rendant le graphe inexploitable par les algorithmes travaillant sur des graphes acycliques).

Un moyen très simple, mais néanmoins efficace, pour éviter cela est de créer un nœud différent pour chaque instance d'un hôte dans un ensemble d'hôtes. Un hôte est ainsi représenté par plusieurs nœuds dans le $\mathcal{CNP}$-Graphe. Pour les différencier, nous les préfixons avec le nom de l'ensemble (séparé par le caractère | afin d'éviter toute confusion avec le nom réel de l'hôte). Ainsi le $\mathcal{CNP}$-Graphe correspondant devient :

Le $\mathcal{CNP}$-Graphe produit par cette technique est montré Figure 6.2 :

$$\begin{cases} \mathcal{V} = & \{C|h1, C|h2, C|h3, R|h1, R|h2, R|h3\} \\ \mathcal{E} = & \{(C|h1, R|h1), (C|h1, R|h2), (C|h1, R|h3), \\ & (C|h2, R|h1), (C|h2, R|h2), (C|h2, R|h3), \\ & (C|3, R|h1), (C|h3, R|h2), (C|h3, R|h3)\} \end{cases}$$

Ce $\mathcal{CNP}$-Graphe est bi-partite : il ne présente plus ni boucle, ni arc opposé, ni cycle et satisfait ainsi les principales propriétés au prix d'une augmentation du nombre de nœuds. Cette technique permet donc d'assurer la compatibilité des $\mathcal{CNP}$-Graphes simples avec la plupart des algorithmes évolués. Il est à noter que si des cycles apparaissent dans un $\mathcal{CNP}$-Graphe compact, ceux-ci seront répercutés dans le $\mathcal{CNP}$-Graphe correspondant, malgré l'utilisation de cette technique.

Par défaut, le calcul des $\mathcal{CNP}$-Graphes applique cette technique.

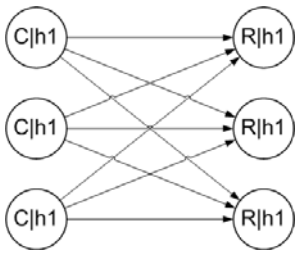


FIG. 6.2 – $\mathcal{CNP}$-Graphe bi-partite complet

6.4 Synthèse sur l'exploitation algorithmique des $\mathcal{CNP}$-Graphes

Dans ce chapitre, nous avons présenté l'importance de la phase de sélection de l'algorithme utilisé pour résoudre un problème modélisé sous forme de $\mathcal{CNP}$-Graphe. Cette sélection est soumise d'un côté à des contraintes utilisateurs liées au temps d'exécution de l'algorithme et au facteur d'approximation des solutions obtenues. D'un autre côté, les caractéristiques du $\mathcal{CNP}$-Graphe doivent être considérées. En particulier, les propriétés des distances issues de l'évaluation des fonctions distance doivent être soigneusement évaluées.

Les propriétés de l'espace euclidien étant les plus populaires dans la théorie des graphes, nous avons présenté une méthode d'évaluation de leur satisfaction au sein d'un $\mathcal{CNP}$-Graphe. Cette méthode est basée sur la définition d'indices de satisfaction pour chacune de ces propriétés.

Avec ce dernier ajout, l'utilisateur est maintenant en capacité de modéliser, dans un $\mathcal{CNP}$-Graphe compact, un problème lié à la répartition des ressource sous sa forme logique, puis de l'instancier en un $\mathcal{CNP}$-Graphe modélisant sa réalité physique, et enfin de sélectionner l'algorithme utilisé pour résoudre le problème en toute connaissance de cause.

L'intégralité de ce processus a été développé au sein d'un service web présenté dans le chapitre suivant.

7

Implémentation : NDS, le *Network Distance Service*

Le processus de déclaration des $\mathcal{CNP}$-Graphes compacts, leur transformation en $\mathcal{CNP}$-Graphes et la résolution des problèmes classiques liès à la répartition des ressources ont été implémentés dans service web appelé *Network Distance Service*, ou plus simplement NDS.

NDS a été développé en JAVA avec le Globus Toolkit 4.2[1] sous Eclipse[2] et compte environ 10000 lignes de codes utiles. Il est interrogeable soit par l'API WSDL classique au travers de requêtes SOAP, soit par une interface graphique conçue pour faciliter son utilisation par les utilisateurs humains.

Les objectifs de cet développement sont centrés sur l'utilisabilité et l'extensibilité. Les efforts de conception et d'implémentation ont mis l'accent sur la clarté des interfaces, de telles façon que l'acquisition du fonctionnement de NDS soit le plus simple possible. De plus, la configuration de NDS a été pensée de telle façon qu'elle soit facilement modifiable, par exemple lors de l'ajout de métriques de surveillance ou de la création de nouvelle métriques composées.

Ce chapitre présente les détails techniques de cette implémentation. En Section 7.1, nous présentons le fichier de configuration de NDS. Puis nous présentons en Section 7.2 l'interrogation de NDS, ainsi qu'un exemple concret de requête en JAVA et les routines dédiées à son traitements. Enfin, nous présentons le client graphique JAVA de NDS en Section 7.3 et nous concluons en Section 7.4.

7.1 Configuration du service web NDS

Dans cette section, nous détaillons la méthode d'accès aux outils de surveillance implémentée dans NDS, ainsi que la configuration des métriques de surveillance. Ensuite, nous détaillons notre système

[1] http ://www.globus.org
[2] http ://www.eclipse.org/

de configuration des métriques composées.

7.1.1 Accès aux outils de surveillance

L'accès aux outils de surveillance, tels qu'NWS, est effectué par l'utilisation de leur API en lignes de commande. Cela implique que l'hôte d'exécution de NDS possède l'équipement logiciel adéquat à l'interrogation de ces outils qui ne sont pas intégrés en tant que librairie dans l'empaquetage logiciel de NDS. Par exemple, NWS devra être installé au préalable afin de pouvoir utiliser ses routines d'interrogation. Cette approche primaire engendre une légère dégradation des performances. En revanche, elle garantit une bonne flexibilité pour l'utilisateur et le configurateur. En effet, ces lignes de commande sont déclarées dans le fichier de configuration JNDI (*Java Naming and Directory Interface*[3]) d'initialisation du service web NDS. L'intégration d'une nouvelle métrique ou même d'un nouvel outil se fait donc par le simple ajout d'une ligne en texte clair dans ce fichier de configuration, suivi d'un simple redémarrage du conteneur Globus. Le redéploiement du service n'est donc pas nécessaire. En outre, les messages d'erreur rencontrés lors de l'exécution de ces lignes de commandes sont remontés au service web sous forme d'exceptions JAVA renseignées, ce qui facilite leur débogage .

Les fichiers de configuration JNDI adoptent un format XML avec un schéma strict et basique formé d'une collection de paramètres formant des couples nom/valeur. La déclaration d'une métrique se fait donc sous forme d'un paramètre *metrics* dont la valeur est composée de cinq lignes :

1. nom de la métrique
2. description de la métrique et des paramètres
3. ligne de commande avec paramètres remplacés par $\%Pi$
4. type de la métrique : *double*, *integer* ou *string*
5. expression rationnelle[4] permettant d'extraire l'observation du flux de sortie
6. multiplicateur permettant d'harmoniser les unités

La déclaration de la métrique latence fournie par NWS est donnée à titre illustratif Figure 7.1.

7.1.2 Déclaration des métriques composées

Les métriques composées sont déclarées en suivant la même méthode que pour les métriques brutes. Il est ainsi très aisé de les raffiner ou d'en ajouter de nouvelles, ce de façon lisible et sans redéploiement du service NDS.

Elles sont déclarée comme un ensemble :

1. nom de la métrique composée
2. description de la métrique composée et de ses paramètres (en nombre infini)
3. définition de la métrique composée
4. type de la métrique composée : *double*, *integer* ou *string*

A titre d'exemple, la déclaration de la métrique $DataTransferCost$ introduite dans la section 5.1.3 est donnée Figure 7.2.

[3] java.sun.com/products/jndi/
[4] Aussi appelée expression régulière

FIG. 7.1 – Déclaration de la métrique brute $NWSlatencyTcp$ dans le fichier de configuration JNDI de NDS

```
<parameter>
 <name>
   metric
 </name>
 <value>
   NWSlatencyTcp
   The amount of time to transmit a TCP message ; Params: %P1,%P2 = IPs/names of targets
   nws_extract latencyTcp -n1 -h0 -fMea %P1 %P2
   double
   -(\ +|-)?((\ .[0-9]+)|([0-9]+(\ .[0-9]*)?))
   0.001
 </value>
</parameter>
```

FIG. 7.2 – Déclaration de la métrique composée DTC dans le fichier de configuration JNDI de NDS

```
<parameter>
   <name>
      compoundMetric
   </name>
   <value>
      NWSDTC
      Data Transfer Cost based on NWS mea. ; Params: %P1,%P2: targets; %P3: size of data
      3*( NWSlatencyTcp(%P1,%P2) + NWSlatencyTcp(%P2,%P1) ) + %P3/NWSbandwidthTcp(%P1,%P2)
      double
   </value>
</parameter>
```

7.2 Interrogation du service web NDS

Dans cette section, nous présentons la composition des requêtes d'interrogation de NDS, un exemple concret en JAVA et les routines utilisées pour traiter ces requêtes.

7.2.1 Requêtes NDS

Les requêtes NDS visent à déclarer les informations nécessaires au calcul de $\mathcal{CNP}$-Graphes compacts, tels que définis Section 4.1. Elles sont composés de trois parties :

$$\begin{cases} \text{Les ensembles nommés d'hôtes du } \mathcal{CNP}\text{-Graphe compact : } c\mathcal{V} \\ \text{L'ensemble des propriétés de tâche : } TPS \\ \text{L'ensemble des fonctions distances : } dfS \end{cases}$$

Les arcs éléments de l'ensemble $c\mathcal{E}$ correspondent aux couples de sommets de $c\mathcal{V}$ pour lesquels une fonction distance à été déclarée dans dfS.

7.2.2 Exemple de requête NDS pour le problème de la sélection de ressource présenté Section 4.2.1

Cet exemple s'appuie sur le problème de la sélection d'une ressource R hébergée par les hôtes `node1.grid.com`, `node2.grid.com` et `node3.grid.com` et devant être utilisée par l'hôte `client.athome.com`. Cette ressource est un service et l'objectif concerne les performances, la fonction distance est celle présentée Section 5.4.2.

$$\begin{cases} c\mathcal{V} = & \{client = \{\texttt{client.athome.com}\}, \\ & \quad R = \{\texttt{node1.grid.com}, \texttt{node2.grid.com}, \texttt{node3.grid.com}\}\} \\ TPS = & \{< input = 1000 >, < nCycle = 500 >, < output = 2000 >\} \\ dfS = & \{df_{client,R} = DTC_{client,R}(input) + CTC_R(cycle) + DTC_{R,client}(output)\} \end{cases}$$

On peut noter qu'une telle requête est non seulement très simple, mais surtout uniquement composée d'informations parfaitement compréhensibles par n'importe quel utilisateur.

7.2.3 Exemple de code client JAVA

Le code JAVA correspondant à cette invocation est donné Figure 7.3.

Ce code est assez verbeux du fait des spécifications *WSDL* qui, dans l'implémentation du GT4, imposent de nombreux emboitement d'objets en grande partie à cause de la gestion des collections qui est limitée. En revanche, l'utilisation de fonctions distances complexes ou de nombreuses métriques ne rend pas la requête plus compliquée. En réalité, le corps de cette requête peut être copié-collé et seules les lignes concernant les déclarations des sources/destinations, des valeurs TPS et de la fonction distance à utliser ont besoin d'être adaptées.

7.2.4 Évaluation des distances

L'évaluation des distances utilisées pour labéliser les arcs des $\mathcal{CNP}$-Graphes est faite en trois étapes :

1. la résolution des métriques composées
2. la résolution des observations des métriques brutes
3. l'évaluation de la fonction distance à proprement dit

Cette opération est effectuée grâce à un analyseur syntaxique développé spécifiquement. Pour cette opération, les fonctions distance sont considérées comme de simples chaînes de caractères.

FIG. 7.3 – Code client en JAVA d'invocation du service NDS

```
// Déclaration du message d'invocation
CSrcDestTaskPropertiesDistanceFunction ftsidcf
 = new CSrcDestTaskPropertiesDistanceFunction();

// Déclaration des ensembles d'hôtes
CHostSets fts = new CHostSet[2];
String sClient[]={"client.athome.com"};
String sR[]={"node1.grid.com", "node2.grid.com", "node3.grid.com"};
fts[0] = new CHostSet("client", sClient);
fts[1] = new CHostSet("R", sR);
ftsidcf.setHostSet(fts);

// Déclaration des valeurs TPS
CNamedValue ids[] = new CNamedValue[3];
ids[0] = new CNamedValue("input", 1000);
ids[1] = new CNamedValue("cCycle", 500);
ids[2] = new CNamedValue("output", 2000);
ftsidcf.setTaskProperties(new CTaskProperties(ids));

// Déclaration des fonctions distances
String dfS[]={"DTC(client,R,input)+CTC(R,cycle)+DTC(R,client,output)"}
ftsidcf.setDistanceFunctionSet(dfS);

// Invocation
CDistance ftd[] = ndsPT.getDistances(ftsidcf).getDistance();

// Exploitation du CNP-Graphe
int bestDest;
if ( ftd[0].getValue() < ftd[1].getValue() ) bestDest=0;
else bestDest=1;
System.out.println(ftd[bestDest].getDestination() + " is the best");
```

Résolution des métriques composées

La résolution des métriques composées consiste à remplacer dans la fonction distance les métriques composées par leur définition. C'est un traitement récursif qui détecte une sous-chaîne de la fonction distance qui correspond à un nom de métrique composée suivie de ses paramètres. Si le nombre de paramètres est incorrect, une exception est levée. Sinon la sous-chaîne est remplacée par la définition de la métrique composée, telle que spécifiée dans le fichier configuration de NDS. Les paramètres $\%Pi$ sont ensuite remplacés par les paramètres effectifs. Ce traitement est appliqué tant qu'une sous-chaîne est détectée, ainsi les métriques composées peuvent contenir d'autres métriques composées.

À la fin de ce traitement, la fonction distance ne contient plus de métrique composée.

Résolution des observations des métriques brutes

La résolution des observations des métriques brutes consiste à remplacer les métriques brutes par leur observation, soit des valeurs réelles pour les métriques de $M^{\mathbb{R}}$, soit des chaînes de caractères pour les métriques de $M^{\mathbb{S}}$. C'est un traitement itératif qui consiste à détecter les sous-chaînes de la fonction distance qui correspondent à un nom de métrique brute suivi de ses paramètres. Si le nombre de paramètres est incorrect, une exception est levée. Ensuite, la ligne de commande déclarée dans le fichier de configuration de NDS est exécutée après avoir remplacé les paramètres $\%Pi$ par les paramètres effectifs. Si cette exécution échoue ou si son résultat n'est pas analysable selon l'expression rationnelle stipulée, une exception est levée. Sinon, la sous-chaîne est remplacée par la valeur obtenue.

À la fin de ce traitement, la fonction distance ne contient plus de métrique.

Évaluation de la fonction distance

L'évaluation de la fonction distance consiste à calculer sa valeur réelle, afin de pouvoir labéliser l'arc du $\mathcal{CNP}$-Graphe correspondant. À ce stade, seuls les noms tp_i de valeur TPS présents dans la fonction distance empêchent son évaluation.

Dans un premier temps, les sous-chaînes correspondantes à un tp_i sont détectées. Si une valeur réelle v_i est stipulée, elle est substituée à tp_i. Sinon, la distribution $distrib_i(tp_i)$ est ajoutée à la fonction distance en tant que multiplieur (voir Section 5.4.1). Enfin, on calcule la valeur de l'intégrale par la somme sur les intervales $[lb_i, ub_i]$ avec les pas s_i des évaluations de la chaîne obtenue après avoir remplacé chaque tp_i par les valeurs v_i. En d'autres termes, on calcule :

$$d(hs, hd) = \sum_{tp_1=lb_1}^{ub_1} \dots \sum_{tp_n=lb_n}^{ub_n} \left[\prod_{k=1}^{n} (s_k \; distrib_k(tp_k)) \times df_{i,j}(tp_1, \dots tp_n) \right]$$

où $\sum_{tp_i=lb_i}^{ub_i} X$ représente la somme discrète de X pour toutes les valeurs de tp_i dans l'intervalle $[lb_i, ub_i]$ en utilisant le pas s_i, que l'on peut aussi représenter par $\sum_{x_i=0}^{\frac{ub_i-lb_i}{s_i}} X$ avec $tp_i = lb_i + x_i \; s_i$.

Enfin, l'évaluation de la chaîne obtenue est faite grâce à la librairie *JAVA Math Expression Parser*[5]. Cette librairie supporte tous les opérateurs et fonctions mathématiques classiques, ainsi que les tests booléens et les chaînes de caractères. L'utilisation de cette librairie confère, à la déclaration des fonction distance, une grande souplesse qui permet une expression précise des besoins par un moyen intuitif et sans contrainte particulière.

7.2.5 Algorithmes

Les $\mathcal{CNP}$-Graphes produits peuvent être exploités directement par l'utilisateur avec ses propres algorithmes. Néanmoins, plusieurs algorithmes ont été implémentés dans NDS : un algorithme de tri afin de résoudre le problème de la sélection, un algorithme de plus court chemin afin de résoudre le

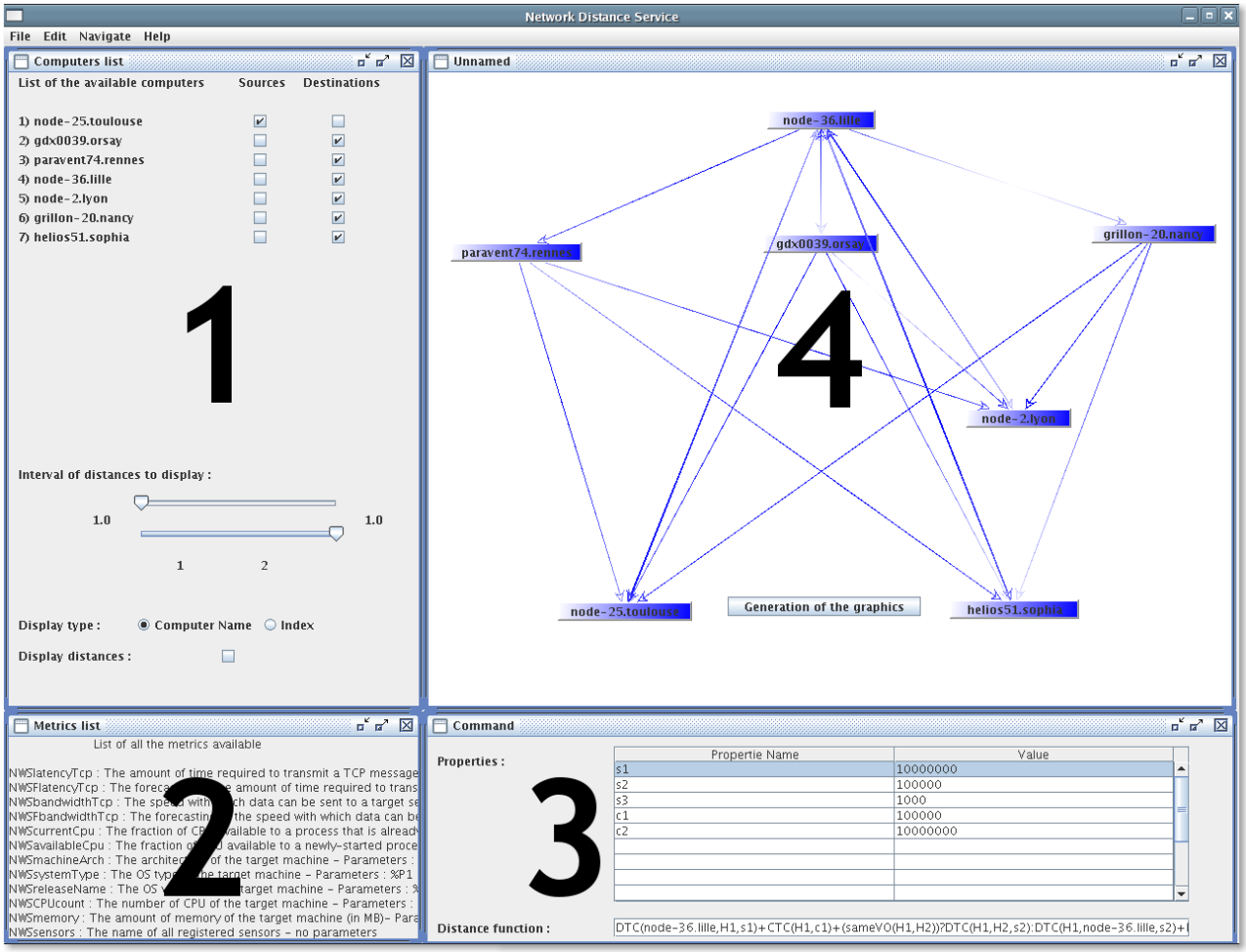
[5] www.singularsys.com/jep/

problème de la composition et un algorithme trivial résolvant le problème des k-médians. Ce dernier est présenté Section 11.1.1.

Ces algorithmes sont ainsi utilisables directement par l'utilisateur. Lorsque le problème modélisé fait partie des cas classiques présentés (sélection, déploiement et composition) et s'inscrit dans un cadre classiques, sans contrainte particulière de temps d'exécution, sa résolution peut être obtenue directement depuis NDS, limitant grandement les efforts utilisateurs. De plus, ces algorithmes représentent une première étape vers une bibliothèque plus complète et mieux renseignée, par exemple en ce qui concerne la complexité, le facteur d'approximation et les contraintes relatives aux propriétés des $\mathcal{CNP}$-Graphes. De plus, afin de faciliter encore plus la résolution des problèmes, nous envisageons d'étudier une méthode de sélection automatique de l'algorithme le plus pertinent en fonction de besoins déclarés par l'utilisateur, de la satisfaction des contraintes et de la taille du problème.

7.3 Client graphique JAVA

FIG. 7.4 – Capture d'écran du client graphique JAVA de NDS.



Afin de faciliter l'utilisation du service web NDS par les acteurs humains, nous avons développé un client graphique lourd en JAVA. Cette interface est composée de quatre fenêtres :

1. Sélection des sommets du $\mathcal{CNP}$-Graphe compact dans la liste des hôtes surveillés (récupérée dynamiquement)
2. Affichage de la liste des métriques supportées (récupérée dynamiquement)
3. Déclaration des valeurs TPS et de l'ensemble des fonctions distance dfS
4. Affichage du $\mathcal{CNP}$-Graphe généré

La barre d'outils permet d'accéder aux options de sauvegarde/chargement des CNP-Graphes, ainsi qu'à l'exécution des différents algorithmes disponibles directement dans NDS.

L'affichage initial des CNP-Graphes est une représentation classique de graphes bi-partites complets, car cette classe correspond aux graphes représentatifs des problèmes de sélection et de déploiement de ressources. Chaque nœud du graphe peut être déplacé interactivement par glissement afin que l'utilisateur puisse l'organiser selon ses besoins.

De plus, un code couleur a été adopté pour les arcs : plus le rapport de la taille de l'arc affiché sur la valeur de son label est important, plus l'arc est foncé. Deux seuils peuvent être fixés afin de n'afficher que les arcs dans un intervalle de valeurs de label donné.

7.4 Synthèse sur l'implémentation

Dans ce chapitre, nous avons abordé les différents détail relatif à l'implémentation de notre approche dans un service web dénommé *Network Distance Service* (NDS).

Nous avons apporté beaucoup de soin à ce développement afin d'obtenir, au delà d'une preuve de concept, une application réellement utilisable. Nous avons ainsi eu une attention particulière pour l'extensibilité et la facilité d'utilisation de NDS : métriques brutes et métriques composées sont aisément ajoutables et modifiables grâce à un fichier de configuration clair et lisible, les requêtes NDS sont facilement manipulables et adaptables avec peu d'efforts de programmation et une interface graphique simple et intuitive permet aux utilisateurs humains de visualiser et manipuler leur $\mathcal{CNP}$-Graphes afin de mieux comprendre les caractéristiques de leurs problèmes. Enfin, NDS étant un service web classique respectant les standards en vigueur, il est très facilement déployable et exploitable dans toute architecture orientée services.

Dans le chapitre suivant, nous allons présenter les différentes expérimentation que nous avons réalisées grâce à ce service.

8

Expérimentations du service web NDS

Ces expérimentations valident l'approche basée sur les $\mathcal{CNP}$-Graphes. Leur objectif est double. D'un côté, nous cherchons à montrer la pertinence des recommandations fournies par NDS dans différents problèmes de répartition des ressources. En effet, ces recommandations doivent tenir compte des performances de l'infrastructure ainsi que des caractéristiques des ressources sujettes du problème. D'un autre côté, nous cherchons à montrer la profitabilité d'NDS qui doit être utilisable avec peu d'efforts grâce à des requêtes simples et ne doit engendrer qu'une faible consommation des ressources, avec des temps d'exécution très courts.

Ces expérimentation s'articulent autour de trois ressources de type différents, qui servent à montrer la capacité d'adaptativité à la superstructure. En Section 8.1, nous présentons ces ressources, ainsi que l'environnement utilisés lors des tests. Puis, en Sections 8.2, 8.3 et 8.4, nous montrons les résultats obtenus pour les trois problèmes génériques sélection, déploiement et composition de ressources. Enfin, nous montrons les résultats d'une expérience portant sur le point d'équilibre en coût de calculs et coûts de communications Section 8.5.

8.1 Cadre expérimental

8.1.1 Environnement

Les expériences sont menées sur la plateforme expérimentale Grid 5000[1] décrite dans [Cappello 05]. Celle-ci permet de réserver des hôtes afin de conduire des expériences dans différents domaines. Sa topologie étendue à l'échelle de la France est montrée Figure 8.1. Elle dispose d'un réseau hautes-

[1] https ://www.grid5000.fr/

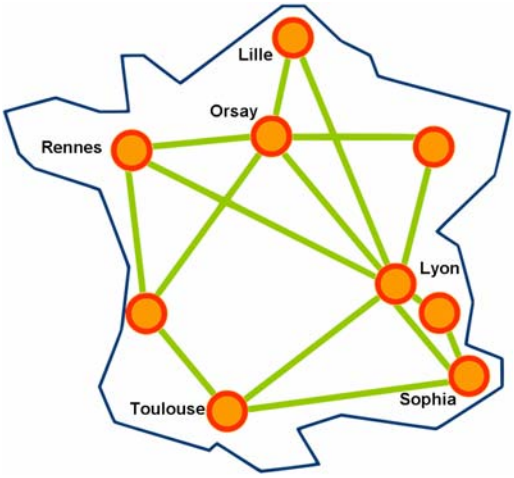


FIG. 8.1 – French Grid 5000 topology

performances (de 1 à 10 giga bits par seconde) réservé à son utilisation sur le réseau RENATER[2] mais partagé entre tous ses utilisateurs. Les hôtes ne sont pas partagés durant les réservations et différentes machines sont disponibles (AMD, Intel, 32 et 64 bits, simple et double cœurs). Ainsi, Grid 5000 représente un environnement réellement hétérogène, dynamique et imprédictible. Les métriques sont fournies par NWS, version 3.0. Les recommandations sont calculées sur un Intel P4 3GHz, 1 MB cache, 1 GB RAM, exécutant une distribution Linux Fedora Core 4.

Les expérimentations sont menées à l'échelle de la France métropolitaine : elles impliquent 16 hôtes répartis sur 6 sites : 2 sur 2 grappes différentes de `Lille`, 2 sur `Lyon`, 2 sur `Orsay`, 4 sur 2 grappes différentes de `Rennes`, 4 sur 2 grappes différentes de `Sophia` et 2 sur `Toulouse`.

Valeur des métriques de surveillance

La liste des noms des hôtes de l'expérimentation est donnée Table 8.1.

Les observations des métriques de surveillance fournies par NWS sont données en annexe Table 1.

8.1.2 Objectif

Ces expérimentations se focalisent sur l'optimisation du temps d'exécution de différentes tâches impliquant une prise de décision. En effet, les aspects sémantiques tels que les coûts financiers d'utilisation des ressources ou la fraîcheur des données peuvent être aisément pris en compte dans les $\mathcal{CNP}$-Graphes afin de procéder à leur optimisation. En revanche, les métriques composées relatives aux performances de l'infrastructure matérielle, à défaut de pouvoir faire l'objet d'une preuve d'efficacité formelle, doivent montrer leur efficacité expérimentale. Ainsi, nos expérimentations permettent de montrer la pertinence des métriques composées ainsi que l'efficacité de notre approche.

[2]http ://www.renater.fr/

```
1         node-11.lille
2         node-62.lille
3   sagittaire-11.lyon
4   sagittaire-49.lyon
5         gdx0116.orsay
6         gdx0140.orsay
7       parasol23.rennes
8       parasol48.rennes
9      paravent05.rennes
10     paravent20.rennes
11       helios01.sophia
12       helios48.sophia
13        node-15.sophia
14        node-65.sophia
15        node-42.toulouse
16        node-51.toulouse
```

TAB. 8.1 – Liste des noms des hôtes de l'expérimentation

Notre approche est validée par l'évaluation des performances des solutions recommandées par NDS et par leur classement dans l'ensemble des solutions possibles. Cela est fait en trois étapes : (1) NDS calcule une recommandation en fonction d'une requête ; (2) Les performances expérimentales de *toutes* les solutions candidates sont évaluées ; (3) La recommandation fournie par NDS est classée parmi toutes les solutions candidates.

Les performances expérimentales sont des temps d'exécution. Ils sont évalués grâce à un *fake service* composé d'un client et d'un serveur dont les paramètres de fonctionnement sont *input*, *comp* et *output*. Son exécution est la suivante : le *fake client* envoie *input* octets de données aléatoires au *fake serveur* ; ce dernier effectue *comp* divisions d'une variable de type `double` ; finalement, *output* octets de données aléatoires sont retournées soit au client, soit à un autre *fake serveur*. On peut ainsi émuler un large panel de comportements d'application : depuis un simple contenu digital type fichier (sans donnée envoyée du client au serveur, ni calculs) jusqu'à des services nécessitant de nombreux calculs avec des quantités de données en entrée et sortie paramétrables.

Ce *fake service* a été calibré sur la machine chargée des calculs des recommandations, laquelle ne fait pas partie de Grid5000. Ce calibrage simple a eu pour but d'évaluer le nombre de cycles CPU consommés en fonction du paramètre *comp*. Nous avons simplement exécuté une fois le *fake serveur* avec $comp = 1\text{E+}6$ et nous avons obtenu un temps d'exécution $T = 0,029613s$, ce qui a permis de déduire que le facteur de calibration C était égal à :

$$\mathcal{C} = \frac{T \times CPUs}{comp} = \frac{0,029613 \times 2996\text{E+}6}{1\text{E+}6} = 88,72$$

Les temps expérimentaux présentés dans les résultats sont des moyennes sur 10 exécutions indépendantes. compte-tenu du faible écart observé d'une exécution à l'autre (<1%), ce nombre est largement suffisant.

8.1.3 Ressources sujettes

Nous avons mené les expériences avec des ressources logicielles de plusieurs types. L'obtention de recommandations différentes pour les différents types a pour objectif de montrer l'adaptabilité de notre approche aux spécificités de la superstructure logicielle, alors que la pertinence de ces solutions a pour objectif de montrer son adaptabilité aux caractéristiques de l'infrastructure matérielle.

Les trois ressources que nous avons émulées grâce au *fake service* sont les trois services principaux de la superstructure GGM : le service de caches collaboratifs, le service de fouille et l'entrepôt de données distribué. Ces trois services sont représentatifs des différents comportements applicatifs que l'on peut trouver dans une superstructure, ils permettent donc de couvrir un spectre varié d'applications.

Les modèles de coût utilisés pour chacune de ces ressources sont les suivants :

Service de cache collaboratif

Le service de cache collaboratif a pour objectif principal de fournir les données exploitées dans la superstructure. Le coût principal impliqué dans son fonctionnement est donc représenté par le transfert de données :

$$\begin{cases} TPS = & \{< dataSize >\} \\ dfS = & \{df_{Vs,Vd} = DTC_{Vd,Vs}(dataSize)\} \end{cases}$$

où $dataSize$ représente la taille de la donnée fournie.

Service de fouille de données

Le service de fouille de données a pour objectif de découvrir des relations dans un ensemble de données. Ces données sont composées d'un ensemble d'attributs de 100 octets au nombre de $nAtt$. Après avoir effectué des calculs à hauteur de 100 $nAtt^2$, il renvoie un rapport de 10 Ko. Son coût est donc modélisé comme suit[3] :

$$\begin{cases} TPS = & \{< nAtt >\} \\ dfS = & \{df_{Vs,Vd} = DTC_{Vs,Vd}(100\ nAtt) \\ & \qquad + CTC_{Vd}(\mathcal{C} \times 100\ nAtt^2) \\ & \qquad + DTC_{Vd,Vs}(10\text{E+3})\} \end{cases}$$

[3]Veuillez noter l'utilisation de la notation ingénieur xE+y $= x \times 10^y$. De plus, une remarque mineure concerne les notations des unités multiples du binaire : selon la normalisation des préfixes binaires de 1998 par la Commission électrotechnique internationale, le kilo octet, noté Ko, désigne 1000 octets alors que le kibi (kilo binaire) octet, noté Kio, désigne 1024 octets. La différence n'étant pas pertinente dans nos expérimentations, nous nous baserons sur le kilo octet et ne considèrerons que des puissances de 10. Pour plus d'informations, voir http ://www.iec.ch/zone/si/si_bytes.htm

Entrepôt de données distribué

L'entrepôt de données distribué a pour objectif d'agréger les données en fonction d'une requête de taille moyenne 1000 octets. Dans ces expérimentation, nous considérons le cas où le service stocke tous les agrégats nécessaires pour répondre à la requête. Lorsque l'agrégat demandé est déjà matérialisé, son travail se résume à le transférer à son client. En revanche, si cet agrégat n'existe pas, le service doit d'abord le calculer à partir des agrégats matérialisés et disponibles localement. Un agrégat est caractérisé par sa taille *aggsize* qui indique le nombre d'agrégats élémentaires qu'il contient. La taille d'un agrégat élémentaire est de 10 ko.

Le nombre d'agrégats élémentaires inclus dans l'agrégat demandé influe sur la part d'agrégats déjà disponibles et celle devant être calculée. En effet, plus la taille de l'agrégat augmente, moins il a de chance d'être déjà matérialisé. Un exemple de ratio d'agrégats matérialisé en fonction du nombre d'agrégats élémentaires est montré Figure 8.2. Il suit une distribution de Gauss $gauss_{1,20000}(x) = e^{-\frac{(x-1)^2 0000}{20000^2}}$ où 1 est la valeur de x au pic et 20000 est la largeur à mi-hauteur ou FWHM (*Full Width at Half Maximum*).

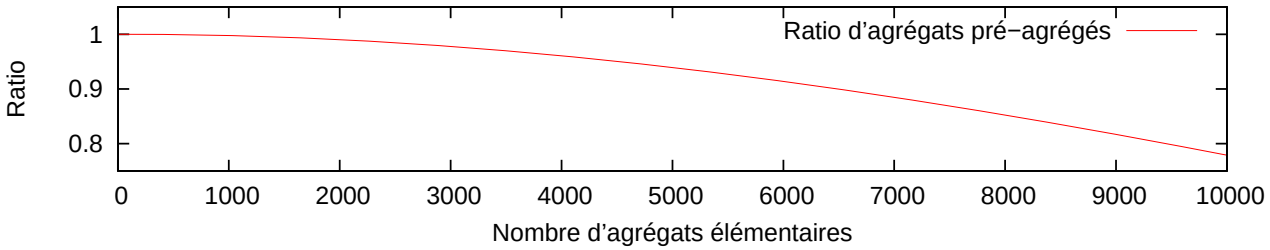


FIG. 8.2 – Ratio d'agrégats matérialisés en fonction du nombre d'agrégats élémentaires

La matérialisation des agrégats est une opération réputée très coûteuse. Nous l'avons arbitrairement évaluée comme la puissance 3 du nombre d'agrégats élémentaires impliqués. Son modèle de coût est alors :

$$\begin{cases} TPS = & \{< aggSize >\} \\ dfS = & \{df_{Vs,Vd} = DTC_{Vs,Vd}(1000) \\ & \quad + CTC_{Vd}(\mathcal{C} \times (aggSize\ (1 - gauss_{1,20000}(aggSize)))^3) \\ & \quad + DTC_{Vd,Vs}(aggSize\ \text{10E+3})\} \end{cases}$$

8.1.4 Conclusion

Dans cette section, nous avons détaillé les ressources sujettes que nous avons utilisées pour expérimenter NDS. Les TPS et dfS décrits sont, en principes, déclarés par les développeurs de ces ressources. Ces déclaration sont intuitives et donc facilement accessibles et modifiables.

Dans les trois sections suivantes, nous montrons les résultats obtenus pour les trois scénarios présentés dans l'introduction, Chapitre 2 : sélection, déploiement et composition de ressources. Les TPS et dfS utilisés sont repris pour former des requêtes NDS complètes. Ces requêtes sont vouées à

être soit générées automatiquement par les applications qui les émettent, soit créées par les utilisateurs humain de NDS.

La présentation de ces expérimentations suit donc le processus de résolution des problèmes liés à la répartition des ressources tel qu'il est perçu dans une utilisation réelle de NDS : la modélisation des coûts relatifs aux ressources est faite par les développeurs qui connaissant ces ressources, puis ces coûts sont réutilisés dans la modélisation et la résolution des problèmes par ceux qui y sont confrontés.

8.2 Sélection

Le problème de la sélection consiste à décider, parmi toutes les instances d'une ressource répliquée, laquelle doit être utilisée afin d'optimiser un aspect donné, dans notre cas le temps d'exécution. Ce problème peut être résolu par la simple sélection de l'arc de label minimum dans le $\mathcal{CNP}$-Graphe.

Le client est l'hôte `node-11.lille` alors que les réplicas sont positionnés sur tous les hôtes de l'expérience.

De plus, nous menons les expériences pour les trois ressources sujettes, lesquelles sont de type différent : communications intensives, calculs intensifs et équilibré.

8.2.1 $\mathcal{CNP}$-Graphes

Service de caches collaboratifs

Le service de caches collaboratifs est un exemple typique de ressources de type communications intensives. Le problème qui se pose est de sélectionner parmi les instances du service qui stocke les réplicas d'une donnée de 100 Mo, celle qui présentera les meilleures performances.

Ainsi le $\mathcal{CNP}$-Graphe que nous utilisons est généré par le $\mathcal{CNP}$-Graphe compact suivant :

$$c\mathcal{CNP} = \begin{cases} c\mathcal{V} = & \{C = \texttt{node-11.lille}\},\ R = \{\texttt{*}\}\} \\ c\mathcal{E} = & \{(C,\ R)\} \\ TPS = & \{< dataSize = 100\text{E+}6 >\} \\ dfS = & \{df_{C,R} = DTC_{R,C}(dataSize)\} \end{cases}$$

Veuillez noter l'utilisation de $*$ pour remplacer tous les hôtes disponibles sur le réseau.

Les 16 valeurs des évaluations réelles des distances des 16 solutions candidates modélisées dans le $\mathcal{CNP}$-Graphe généré sont données en annexe, Table 2.

Service de fouille de données

Le service de fouille de données est un exemple typique de ressources de type calculs intensifs : les données échangées ne sont pas très importantes, mais les temps de calcul sont généralement critiques. Le problème est de sélectionner l'instance de ce service pouvant traiter le plus rapidement 1000 attributs. Ainsi le $\mathcal{CNP}$-Graphe que nous utilisons est généré par la requête NDS :

$$c\mathcal{CNP} = \begin{cases} c\mathcal{V} = & \{C = \texttt{node-11.lille}\},\ R = \{\texttt{*}\}\} \\ c\mathcal{E} = & \{(C,\ R)\} \\ TPS = & \{< nAtt = 1000 >\} \\ dfS = & \{df_{C,R} = DTC_{C,R}(100\ nAtt) + CTC_R(\mathcal{C} \times 100\ nAtt^2) + DTC_{R,C}(\text{10E+3})\} \end{cases}$$

Les 16 valeurs des évaluations réelles des distances des 16 solutions candidates modélisées dans le $\mathcal{CNP}$-Graphe généré sont données en annexe Table 2.

Entrepôt de données distribué

L'entrepôt de données distribué est un exemple typique de ressources de type équilibré : il implique aussi bien des transferts de données que des calculs. La problème est de sélectionner l'instance qui pourra fournir un agrégat de taille 200 le plus rapidement possible. Ainsi le $\mathcal{CNP}$-Graphe que nous utilisons est généré par la requête NDS :

$$c\mathcal{CNP} = \begin{cases} c\mathcal{V} = & \{C = \texttt{node-11.lille}\},\ R = \{\texttt{*}\}\} \\ c\mathcal{E} = & \{(C,\ R)\} \\ TPS = & \{< aggSize = 200 >\} \\ dfS = & \{df_{C,R} = DTC_{C,R}(1000) \\ & \quad + CTC_R(\mathcal{C} \times (aggSize \times (1 - gauss_{1,20000}(aggSize)))^3) \\ & \quad + DTC_{R,C}(aggSize \times \text{10E+3})\} \end{cases}$$

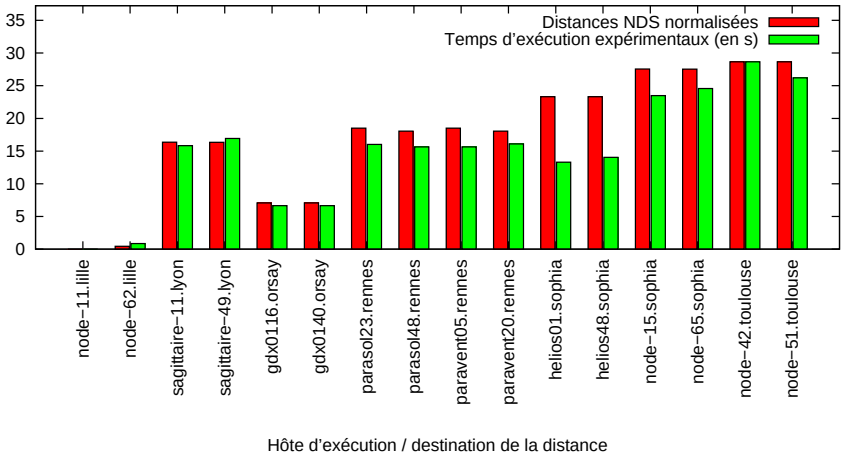
Les 16 valeurs des évaluations réelles des distances des 16 solutions candidates modélisées dans le $\mathcal{CNP}$-Graphe généré sont données Table 2.

8.2.2 Comparaison des résultats expérimentaux

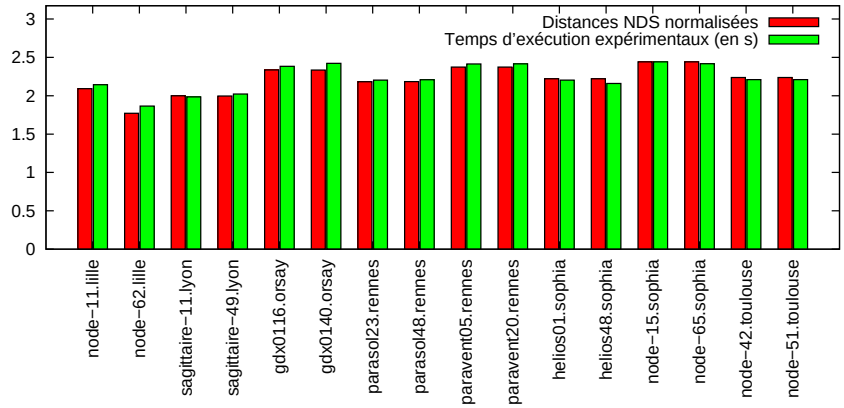
La Figure 8.3 montre l'ensemble des temps d'exécution expérimentaux (en secondes) et les distances obtenues en tant que label du $\mathcal{CNP}$-Graphe (en unité arbitraire) en fonction de la localisation du réplica. Les distances NDS ont été normalisées afin de faire correspondre leur maximum avec le maximum des temps expérimentaux.

Temps de décision NDS : $31\mu s$ ($25\mu s$ pour la récupération des métriques brutes, $6\mu s$ pour le calcul des distances)

Type communication intensives :



Type calculs intensifs :



Type équilibré :

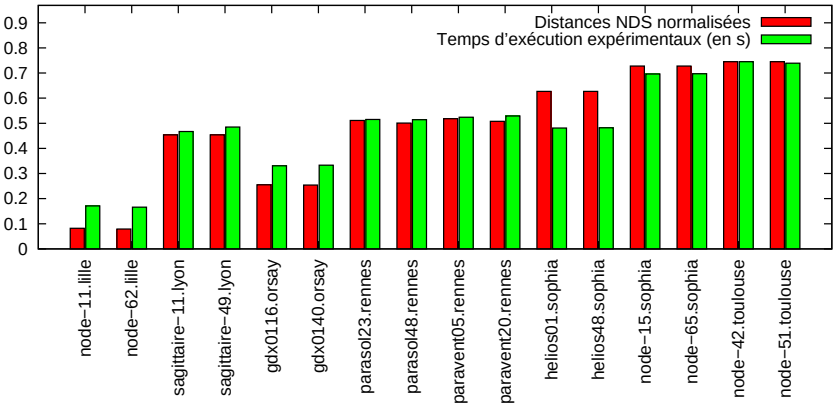


FIG. 8.3 – Performances expérimentales et distances NDS entre `node-11.lille` et tous les autres hôtes

On peut tout d'abord remarquer que le classement des différentes solutions candidates est pratiquement identique en se basant sur les distances des $\mathcal{CNP}$-Graphes et en se basant sur les performances expérimentales : ceci prouve la pertinence des distances qui permettent effectivement d'évaluer les coûts des tâches. Plus particulièrement les recommandations fournies par NDS, désignées par les distances minimales, correspondent effectivement aux meilleures performances expérimentales. Par exemple, la ressource de type calculs intensifs présente des performances optimales sur `node-62.lille` avec un temps d'exécution de $1,768s$ qui est attesté avec une distance de $3,423$ alors que le deuxième meilleurs résultat se constate sur `sagittaire11.lyon` avec un temps d'exécution de $1,998s$ pour une distance de $3,869$.

De plus, le classement des distances, et plus particulièrement les recommandations, varient selon le type de la tâche. Par exemple, la sélection du réplica hébergé par `node-11.lille` représente la meilleure décision pour le type communication intensive. Même si ce résultat est trivial puisqu'il correspond au réplica local, on peut constater que `node-62.lille` représente la meilleure décision pour les types calculs intensifs. L'aspect important ici est que, pour un même problème, NDS fournit des recommandations différentes en fonction de la nature de la ressource sujette.

On peut également remarquer que les hôtes des sites `Orsay` et `Lyon` représentent des performances inégales. Pour les ressources de type calculs intensifs, `Lyon` représente le meilleur choix. Pour les ressources de type communications intensives, `Orsay` représente le meilleur choix. Or, les caractéristiques des machines de ces deux sites sont équivalentes sur le papier[4], que ce soit en matière de CPU (AMD Opteron 246) ou de liens physiques de communication vers le client situé à Lille (10 Gb/s).

C'est là un des intérêts de notre approche, qui permet d'aller au delà des caractéristiques théoriques de l'infrastructure et d'exploitant les mesures dynamiques reflétant la réalité des performances telles qu'elles sont perçues par les applications. Ainsi, notre approche montre une double capacité d'adaptation : d'un côté aux capacités de l'infrastructure matérielle, que ce soit en terme de calcul ou de communication, et d'un autre aux caractéristiques des ressources sujettes aux décisions.

Plus précisément, le modèle de sélection est différent selon les types de tâche : alors que celui des tâches de type communications intensives montre bien une prise en compte de la topologie de l'infrastructure matérielle avec de grandes variations de performances selon la solution candidate, celui des tâches du type calculs intensifs n'est pas affecté par cette topologie. En effet, ces dernières n'impliquant pas ou peu de transferts de données, seules les capacités CPU importent. De plus, le modèle de sélection pour le type équilibré est effectivement équilibré entre les deux modèles extrêmes. Ceci montre la réelle capacité d'adaptation de notre approche aux spécificités de la superstructure logicielle : les distances étant différentes selon les caractéristiques des ressources et tâches, les décisions qui s'en suivent le sont aussi.

Il y a une sous-estimation remarquable des hôtes de la grappe `helios` du site `sophia` lorsque des communications sont impliquées. Cela est dû à une configuration TCP spécifique sur ce cluster qui accélère le traitement des très petits paquets et introduit des erreurs dans les mesures NWS. Ceci peut être corrigé très simplement en configurant NWS de manière adaptée. Nous avons utilisé la configuration par défaut et laissé apparître ce problème pour montrer la dépendance de notre solution aux mesures de surveillance et modérer son importance, la plupart des distances étant tout de même parfaitement pertinentes, ainsi que les recommandations qui en découlent.

[4] voir www.grid5000.fr

De plus, les recommandations représentent de réelles optimisations étant donné que les temps d'exécution varient de, pour chacun des trois types, $1s$ à $30s$, $1,8s$ à $2,4s$, et $0,1s$ à $0,7s$. Ces écarts montrent également que notre approche est pertinente aussi bien pour des tâches longues que courtes, et avec des écarts entre les différentes solutions aussi bien importants que très faibles.

Enfin, nous notons que le temps de décision, qui intègre le calcul des distances ainsi que la sélection du minimum, est très court : $31\mu s$, dont $25\mu s$ pour la récupération des métriques brutes et $6\mu s$ pour le calcul des distances. Ceci montre la profitabilité satisfaisante de notre approche pour les problèmes de sélection, y compris pour des tâches caractérisées par des temps d'exécution très courts.

8.3 Déploiement

Le second problème auquel nous nous sommes intéressés est la construction d'un plan de déploiement. Ce dernier consiste à dimensionner et placer une ressource : c.-à-d. décider du nombre d'instances devant être déployées et des emplacements de chacune d'elles. L'objectif est de minimiser le temps d'exécution global pour tous les clients, ce qui correspond au problème des k-médians présenté Section 6.1. Nous supposons que tous les hôtes sont clients et hébergeurs potentiels.

8.3.1 Prise en compte de la distribution des clients

Dans la plupart des cas, la distribution des requêtes clients n'est pas homogène : les demandes pour la ressource ne sont pas équiprobables quel que soit le client. C'est pourquoi nous utilisons une fonction de distribution de ces requêtes par client. Cette information peut être extraite par une analyse des logs ou par une analyse des relations entre clients et ressources. Nous intégrons cette information dans les distances en tant qu'une métrique $QD \in M_{\mathcal{H}}^{\mathbb{R}}$ (pour *Query Distribution*) donnant la quantité moyenne de requêtes pour chaque hôte dans un laps de temps donné. La Table 3, donnée en annexe, montre la distribution aléatoire que nous avons utilisée, donnant le nombre de requêtes par heure. Nous avons arbitrairement fixé la durée de l'expérience à une heure par client, soit 16 heures.

8.3.2 Prise en compte de la variation des valeurs TPS

Comme le déploiement est un problème à long terme (c.-à-d. les ressources, une fois déployées, seront utilisées sur une longue période), nous utilisons le deuxième mode d'évaluation des distances. Ce dernier utilise l'intégration de la fonction distance sur les propriétés de tâche afin de considérer un intervalle de valeurs plutôt qu'une unique valeur fixe donnée. Nous avons utilisé une distribution Gaussienne (ou Normale) classique :

$$gauss_{a,b}(x) = e^{-\frac{(x-a)^2}{b^2}}$$

où a est la valeur de x au pic et b est la largeur à mi-hauteur ou FWHM (*Full Width at Half Maximum*). Ces distributions ont été choisie arbitrairement et sont montrées Figure 8.4. La distribution gaussienne a été choisie pour sa grande popularité. Au demeurant, NDS permet de déclarer toute distribution jugée pertinente par l'utilisateur (voir Section 5.3). Ainsi, le choix de cette distribution n'a que peu d'impact sur la pertinence de l'évaluation de notre proposition.

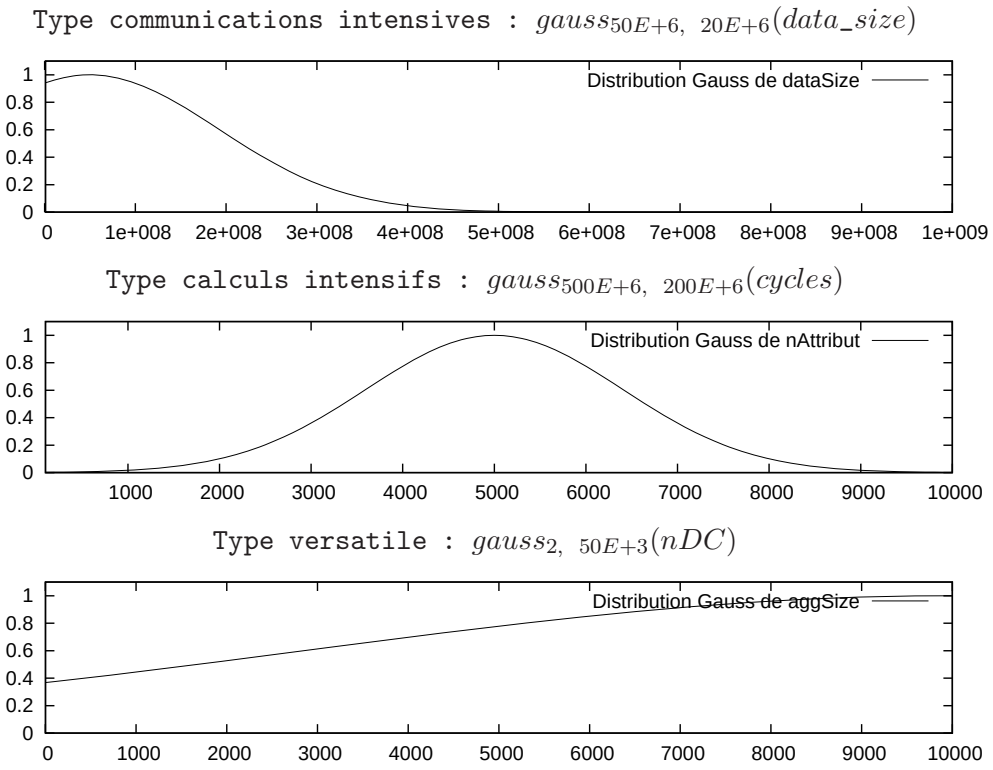


FIG. 8.4 – Distribution des valeurs TPS

8.3.3 $\mathcal{CNP}$-Graphes

Nous avons mené cette expérience pour trois types de services : type communications intensives, type calculs intensifs et type versatile, qui le cas le plus difficile à gérer.

Service de caches collaboratifs

La ressource de type communications intensives est le service de caches collaboratifs. Il fournit des données dont la taille varie entre 1 Mo et 1 Go, avec une majorité de 50 Mo et un FWHM de 200 Mo. Ainsi la requête NDS est :

$$c\mathcal{CNP} = \begin{cases} c\mathcal{V} = & \{C = \{*\},\ R = \{*\}\} \\ c\mathcal{E} = & \{(C,\ R)\} \\ TPS = & \{< dataSize,\ gauss_{50E+6,\ 200E+6},\ \text{1E+6},\ \text{1000E+6} >\} \\ dfS = & \{df_{C,R} = QD_C(datasize) \times DTC_{R,C}(dataSize)\} \end{cases}$$

Service de fouille de données

La ressource de type calculs intensifs est le service de fouille de données. Le nombre d'attributs varie de 100 à 10000, avec une majorité de 5000 et un FWHM de 2000. Ainsi la requête NDS est :

$$c\mathcal{CNP} = \begin{cases} c\mathcal{V} = & \{C = \{*\},\ R = \{*\}\} \\ c\mathcal{E} = & \{(C,\ R)\} \\ TPS = & \{< nAtt,\ gauss_{5000,\ 2000},\ 100,\ 10000) >\} \\ dfS = & \{df_{C,R} = DTC_{C,R}(100\ nAtt) \\ & \qquad + CTC_R(\mathcal{C} \times 100\ nAtt^2) \\ & \qquad + DTC_{R,C}(10\text{E+}3)\} \end{cases}$$

Entrepôt de données distribué

Les deux ressources pour lesquelles les résultats viennent d'être présentés ont des comportements clairs quelles que soient les valeurs TPS. En revanche, certaines ressources changent de comportement en fonction des valeurs TPS. Ce dernier est un cas bien plus subtil que nous appelons type versatile.

Un exemple de ce type, est un service de comparaison de fichiers utilisant un algorithme à complexité exponentielle en fonction du nombre de fichiers. Lorsque ce nombre est faible, le traitement de comparaison est très rapide, le coût principal est alors celui de transfert des fichiers et du compte rendu. En revanche, lorsque le nombre de fichiers augmente, le traitement de comparaison prend de plus en plus de temps, jusqu'à un certain point où il dépasse le coût de transfert.

Mais ce type implique une subtilité supplémentaire. En effet, il est très probable que la somme des coûts impliqués dans la comparaison de très nombreux fichiers soit bien plus importante que la somme des coûts impliqués dans la comparaisons d'un faible nombre de fichiers. Ainsi, la prise en compte globale des coûts implique que ce service soit considéré du type calculs intensifs.

Le type versatile est donc rencontré lorsque la ressource change de comportement en fonction des valeurs TPS *et* que la distribution de ces valeurs privilégie un grand nombre de requêtes d'un coût faible pour un certain type, contre un faible nombre de requêtes d'un coût élevé pour un autre type.

Ce type est donc très difficile à appréhender et en pratique impossible à gérer par des techniques intuitives, car le comportement de la ressource est imprévisible et, comme montré dans les expériences de sélection, les modèles sont différents selon le type.

Un autre cas de ressources de type versatile est lorsque certaines composantes de coût sont fixes alors que d'autres varient d'une position mineure vers une position majeure. Ce cas concerne essentiellement les ressources dont le comportement est imprédictible, par exemple lorsque le temps de calcul n'est pas fonction d'une valeur TPS et peut sensiblement varier.

La ressource de type versatile que nous considérons est l'entrepôt distribué : ce dernier est essentiellement de type communications intensives lorsque la taille de l'agrégat est basse, puis lorsqu'elle

augmente il devient progressivement de type calculs intensifs. Il est donc très difficile de fixer *a priori* le type de cette ressource face à des tailles d'agrégats variables. Dans la suite, nous considèrerons que la taille des agrégats demandés est entre 1 et 10000, avec une majorité d'agrégats de taille importante 10000 et un FWHM de 10000.

$$c\mathcal{CNP} = \begin{cases} c\mathcal{V} = & \{C = \{*\},\ R = \{*\}\} \\ c\mathcal{E} = & \{(C,\ R)\} \\ TPS = & \{< aggSize,\ gauss_{10000,\ 10000},\ 1,\ 10000 >\} \\ dfS = & \{df_{C,R} = DTC_{C,R}(1000) \\ & \quad + CTC_R(\mathcal{C} \times (aggSize\ (1 - gauss_{1,20000}(aggSize)))^3) \\ & \quad + DTC_{R,C}(aggSize\ \text{10E+3})\} \end{cases}$$

8.3.4 Satisfaction des propriétés euclidiennes

Comme nous l'avons montré Chapitre 6, plusieurs classes d'algorithmes sont disponibles pour résoudre les problèmes de déploiement. Ces différentes classes dépendent de la satisfaction des propriétés de l'espace euclidien par les distances utilisée dans le $\mathcal{CNP}$-Graphe.

La Table 8.2 montre les indices de satisfaction des propriétés de l'espace euclidien calculés sur les distances des $\mathcal{CNP}$-Graphes et sur les résultats expérimentaux. La première observation concerne la correspondance entre les indices expérimentaux et ceux obtenus sur les $\mathcal{CNP}$-Graphes , qui atteste de la pertinence de notre approche et des résultats obtenus par NDS.

Tab. 8.2 – Indices de satisfaction des propriétés de l'espace euclidien par les distances et par les temps d'exécution expérimentaux

`Type communications intensives :`

Propriété	nonnegativité		séparabilité		symétrie		inégalité triangulaire	
	NDS	Expe.	NDS	Expe.	NDS	Expe.	NDS	Expe.
Mean	1.00	1.00	1.00	1.00	0.52	0.47	0.80	0.81
Max	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Min	1.00	1.00	1.00	1.00	0.01	0.01	0.21	0.14

`Type calculs intensifs :`

Propriété	nonnegativité		séparabilité		symétrie		inégalité triangulaire	
	NDS	Expe.	NDS	Expe.	NDS	Expe.	NDS	Expe.
Mean	1.00	1.00	0.00	0.00	0.54	0.54	0.95	0.95
Max	1.00	1.00	0.00	0.00	1.00	1.00	1.00	1.00
Min	1.00	1.00	0.00	0.00	0.01	0.01	0.86	0.79

`Type versatile :`

Propriété	nonnegativité		séparabilité		symétrie		inégalité triangulaire	
	NDS	Expe.	NDS	Expe.	NDS	Expe.	NDS	Expe.
Mean	1.00	1.00	0.00	0.00	0.53	0.52	0.90	0.91
Max	1.00	1.00	0.00	0.00	1.00	1.00	1.00	1.00
Min	1.00	1.00	0.00	0.00	0.01	0.01	0.43	0.50

Plus précisément, la séparabilité est satisfaite seulement lorsque peu de calculs sont impliqués. Ceci est dû au fait que le coût d'un transfert local est considéré nul (voir la définition de DTC Section 5.1.3) et non celui d'un calcul local. De plus, la symétrie et l'inégalité triangulaire ne sont pas satisfaites. Comme vu Section 6.1, cela implique des risques dans l'utilisation de certains algorithmes de graphe. Cependant, l'inégalité triangulaire atteint un niveau plus satisfaisant pour la ressource de type calcul intensif. Dans le cas où l'utilisateur considèrerait ce niveau satisfaisant en fonction de l'algorithme qu'il compte utiliser, alors il doit choisir des algorithmes considérant des distances asymétriques, tels que pour les k-médians : [Archer 01, Gortz 06, Panigrahy 98].

8.3.5 Résultats expérimentaux

Dans la suite, nous montrons comment utiliser les résultats d'un algorithme résolvant le problème des k-médians afin de décider combien d'instances des ressources sont nécessaires et où les placer. De plus, afin d'analyser l'impact de l'utilisation de la distribution des requêtes et de l'intégration des valeurs TPS, nous montrons les performances des recommandations NDS dans quatre cas :

- (1) sans intégration, ni distribution des requêtes
- (2) sans intégration, mais avec distribution des requêtes
- (3) avec intégration, mais sans distribution des requêtes
- (4) avec intégration et distribution des requêtes

Lorsque la distribution des requêtes n'est pas utilisée, nous considérons que chaque client envoie une quantité équivalente de requêtes. Lorsque l'intégration des valeurs TPS n'est pas utilisée, nous utilisons le pic de la distribution gaussienne comme valeur fixe dans la requête NDS.

Les matrices des évaluations réelles des distances dans le cas (4) sont montrées en annexe Tables 4, 6 et 8.

Les Tables 5, 7 et 9 données en annexe montrent en détail les résultats de l'exécution de l'algorithme trivial des k-médians sur les $\mathcal{CNP}$-Graphes des trois ressources de l'expérimentation. Ces résultats sont donnés pour les quatre cas de calculs de distances étudiés, ainsi que pour les cas expérimentaux correspondant aux solutions meilleures, pires et médianes. Les résultats sont donnés sous trois formes : le critère d'évaluation expérimental de la solution (dont la formule est rappelée ci-après), la précision qui est calculée comme le ratio entre le critère pour la solution concernée et le critère de la meilleure solution, et les détails des hôtes sélectionnés pour héberger les ressources (désignés par μ_j dans la formule du ci-dessous).

$$critere = \sum_{i \in \mathcal{V}} \min_{j=1}^{k} \; d(i, \mu_j)$$

Pour plus de lisibilité, nous avons inclus ces résultats dans la Figure 8.5, qui montre le temps d'exécution expérimental moyen par client comme une fonction du nombre de réplicas. Les barres représentent les performances des recommandations NDS dans les quatre cas étudiés. Les courbes pleines représentent l'intervalle des performances de toutes les solutions candidates en deux parties : meilleure et pire que la performance médiane. La frontière basse de cette courbe indique donc les

meilleures performances atteignables alors que la frontière supérieure représente les pires possibles. De plus, les solutions comportant plusieurs réplicas sur le même site étant peu pertinentes, elles ont été filtrées des solutions candidates.

Comme on peut le voir, les recommandations dans le cas (4) correspondent aux meilleurs résultats, tandis que les solutions candidates présentent un large intervalle de performances. Le temps de décision NDS étant presque $1s$, notre approche est également largement profitable, surtout en considérant que les temps donnés le sont pour une unique requête.

De plus, les résultats NDS évaluent le facteur d'accélération relatif à l'ajout d'un réplica supplémentaire. Ainsi, on peut décider du nombre d'instances nécessaires en fonction des besoins spécifiques de la ressource. En effet, on peut remarquer qu'il est intéressant de déployer des ressources du type communications intensives jusqu'à 6 instances, mais on peut décider de se limiter à 3 si cela représente une accélération suffisante. En revanche, la réplication des ressources de type calculs intensifs n'apporte pas d'amélioration des temps de réponse en régime normal (c.-à-d. hors saturation). Comme nous l'avons montré dans les expériences sur la sélection, la topologie de l'architecture matérielle importe peu pour ce type de ressources, il est donc inutile de distribuer ces ressources près des clients[5].

Un pendant logique de cette observation est que les décisions NDS sont améliorées par la prise en compte de la distribution des requêtes uniquement lorsque la topologie doit être prise en compte, lorsque la composante communication est importante dans les coûts, comme l'atteste le cas (2) qui ne représente une amélioration que pour le type communications intensives.

L'intégration des valeurs TPS n'est utile que pour les ressources de type versatile : les cas (1) et (2) montrent de mauvais résultats car l'absence d'intégration implique que la ressource est considérée de type calculs intensifs et les décisions prises surtout en fonction du coût des calculs. L'intégration des valeurs TPS permet de détecter le type réel de la ressource qui est fait communications intensives, comme l'atteste le cas (3) qui présente une légère amélioration mais surtout le cas (4) qui présentent des résultats optimaux.

Finalement, une remarque importante à propos de cette expérience est que le nombre de requêtes qu'un hôte ou une ressource peut supporter dans un temps donné n'est pas pris en compte. Or comme les $\mathcal{CNP}$-Graphes de deux ressources similaires seront pratiquement identiques, les deux plans de déploiement obtenus pour ces deux ressources par l'exécution successive de résolutions des k-médians seront également identiques. Il se peut alors que les mêmes hôtes soient sélectionnés pour héberger ces deux ressources alors qu'il aurait été judicieux de les placer sur des hôtes différents afin d'équilibrer les charges. Ce problème fait l'objet d'un algorithme présenté Section 11.

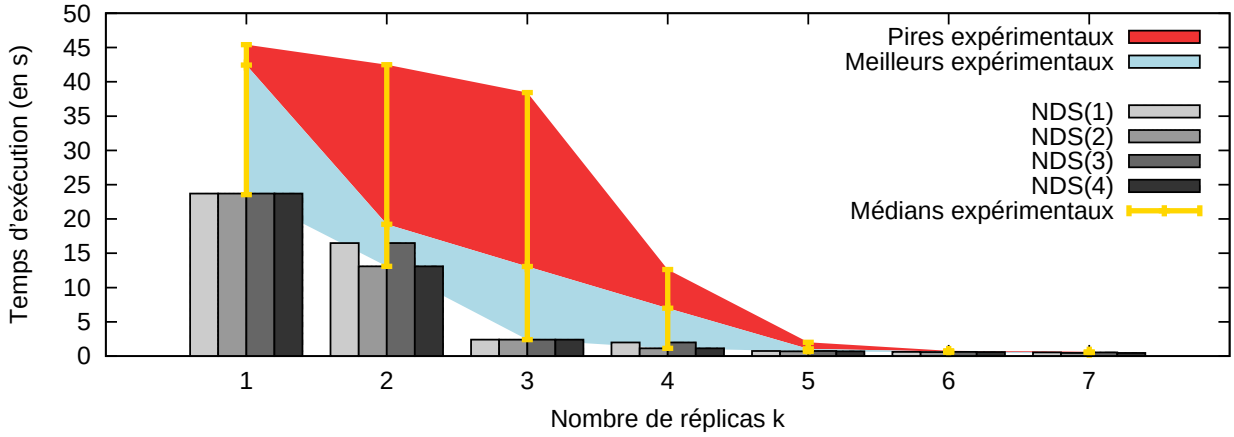
8.4 Composition de ressources

Comme introduit Section 2.3, la composition de ressources est au cœur des architectures orientées services. L'exemple typique déjà présenté concerne l'adaptation d'un contenu numérique multimédia qui doit être traité par plusieurs services en suivant un chemin d'adaptation entre le fournisseur et le client. Cette tâche implique des prises de décision relatives à la distribution particulièrement com-

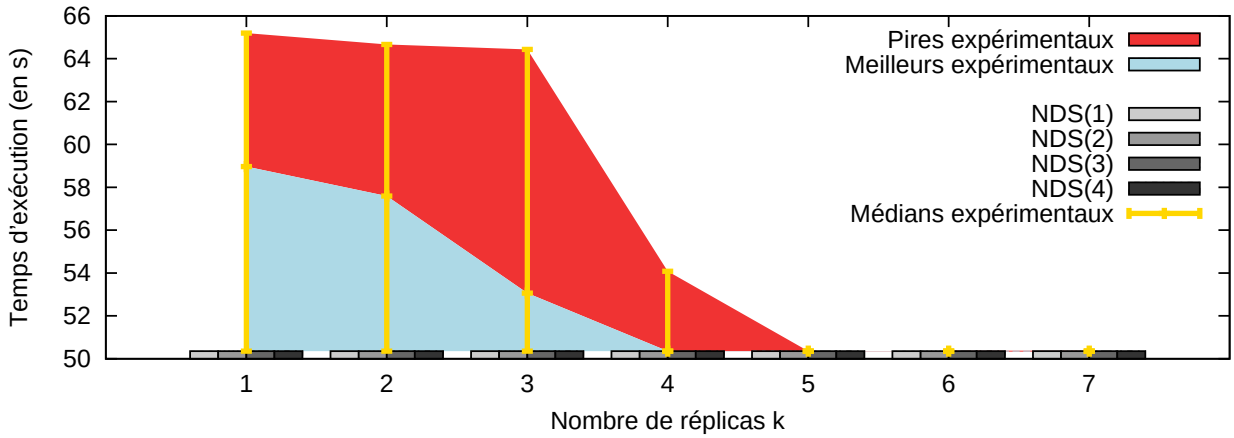
[5] Cela n'empêche pas de décider de répliquer ces ressources en fonction de leur capacité de traitement et du nombre de requêtes attendues sur la plateforme

Temps de décision : 971 097$\mu s \approx 1s$ (611μs pour la récupération des métriques brutes, 608μs pour le calcul du $\mathcal{CNP}$-Graphe, 178μs pour le calcul des propriétés, 969 700μs pour la résolution des k-médians avec $k \in [1, 16]$)

Type communications intensives :



Type calculs intensifs :



Type versatile :

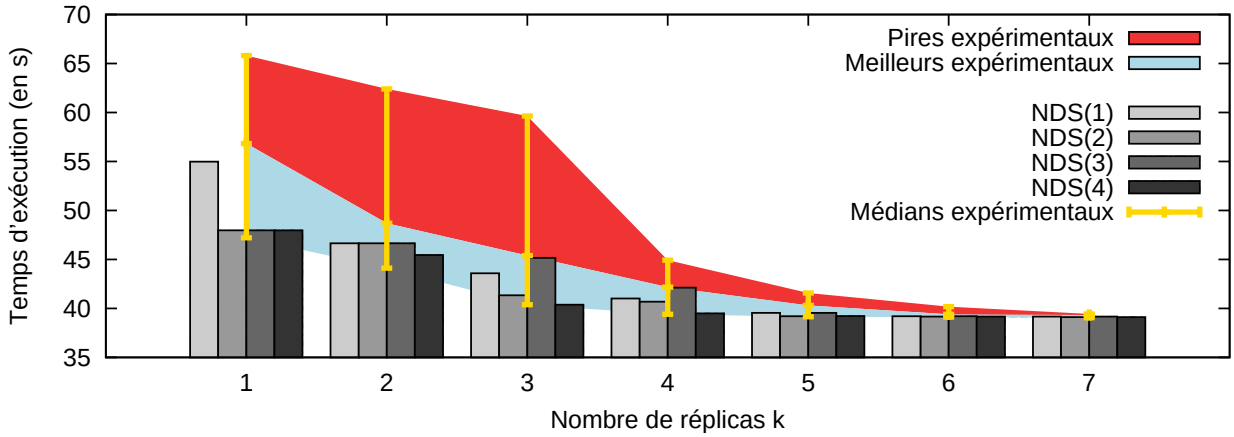


FIG. 8.5 – Temps d'exécution expérimentaux moyens en secondes en fonction du nombre de réplicas pour le problème du déploiement

plexes. D'une part, plusieurs chemins logiques (aussi appelés sémantiques) peuvent correspondre à l'adaptation finale voulue. Par exemple, sur la Figure 8.6 on peut remarquer que les opérations de conversion de taille et de format sont interchangeables. D'autre part, chaque service d'adaptation peut être répliqué, augmentant le nombre de combinaisons possibles pour former le chemin physique du fournisseur au client. En conséquence, le nombre de solutions candidates est généralement extrêmement élevé, rendant les décisions impossibles par des moyens intuitifs et risquées par des heuristiques hasardeuses.

Notre approche permet de prendre simultanément tous les aspects en compte afin d'optimiser le chemin logique et physique en fonction des caractéristiques de la superstructure logicielle et des capacités et topologie de l'infrastructure logicielle.

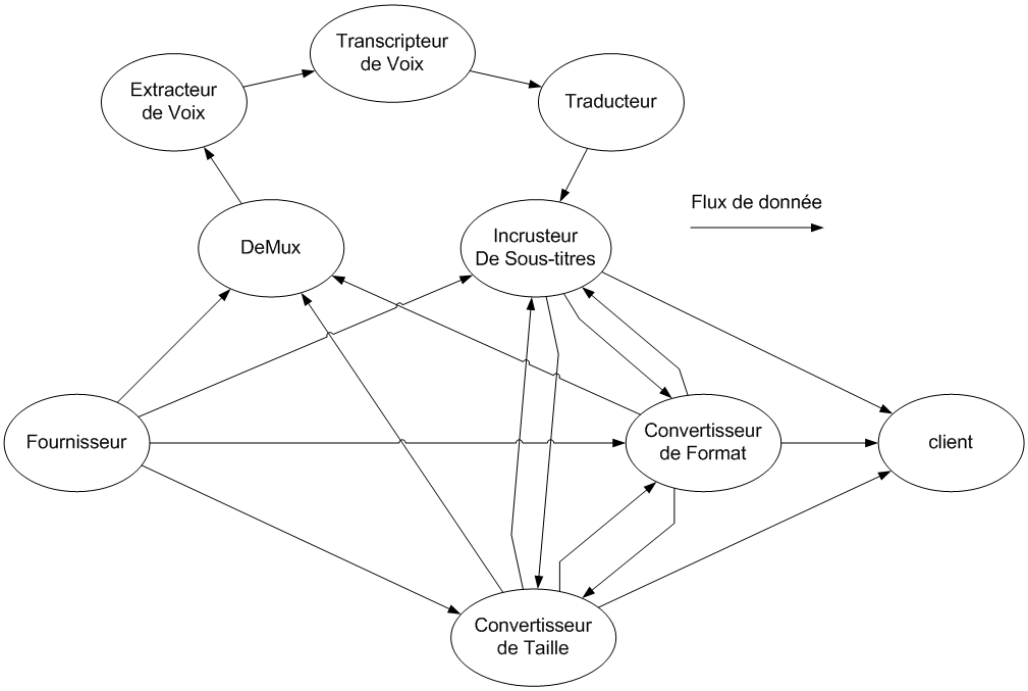


Fig. 8.6 – Exemple de chemins logiques d'adaptation de contenus multimédia.

Pour rappel, la Figure 8.6 illustre un exemple de composition de services permettant l'adaptation d'un contenu multimédia vidéo qui doit être changé de taille, de format et être sous-titré grâce à l'extraction du flux audio, sa transcription en texte, sa traduction puis son incrustation dans le flux vidéo. On parle ici de chemin logique d'adaptation, car on décrit les opérations devant être appliquées au contenu ainsi que leur ordre, sans prendre en compte les services physiques devant être effectivement invoqués pour effectuer ces opérations.

La description des différents services impliqués est :

Fournisseur fournit le contenu vidéo initial

Convertisseur de taille change la taille du contenu vidéo reçu en entrée

Convertisseur de format change le format du contenu vidéo reçu en entrée

DeMux extrait le son du contenu vidéo reçu en entrée

Extracteur de voix extrait les voix du contenu audio reçu en entrée

Transcripteur de voix produit un fichier texte synchronisé transcrivant le contenu audio reçu en entrée

TAB. 8.3 – Temps d'exécution expérimentaux pour la composition de ressources

Solution candidate	NDS	Meilleur	Médian	Pire
Temps d'exécution expérimental (s)	28,65	28,65	67,83	121,89

Temps de décision : 3065μs (618μs pour la récupération des métriques brutes, 2447μs pour le calcul du $\mathcal{CNP}$-Graphe)

Traducteur traduit un fichier texte reçu en entrée

Incrusteur de sous-titres incruste un fichier texte synchronisé dans un contenu vidéo reçu en entrée

client consomme l'adaptation finale du contenu

Les différents contenus échangés lors de cette composition sont les suivants :

V : Le contenu vidéo initial (100 Mo)

Vf : Le contenu vidéo après conversion de format (60 Mo)

Vt : Le contenu vidéo après changement de taille (50 Mo)

Vft : Le contenu vidéo après conversion de format et changement de taille (30 Mo)

S : Le flux audio du contenu vidéo (1 Mo)

S' : Le contenu audio des voix du flux audio (1 Mo)

S" : Les sous-titres dans la langue initiale (1 Ko)

s : Les sous-titres dans la langue cible (1 Ko)

Nous considérons que l'incrustation des sous-titres ne change pas la taille de la vidéo. De plus, nous notons les différents contenus comme des combinaisons des différentes adaptations et les opérations notées entre crochet sont optionnelles. Par exemple Vfts représente le contenu vidéo retaillé, convertit et sous-titré, alors que V[f]s représente deux contenus : soit le contenu vidéo sous-titré et converti, soit le contenu vidéo seulement sous-titré.

En considérant que chacun des 8 services est répliqué deux fois sur la grille, et compte-tenu des 6 chemins logiques possibles, on obtient le nombre de solutions candidates : $6 \times 2^8 = 1536$.

Le $\mathcal{CNP}$-Graphe est produit avec la requête NDS détaillée Section 4.2.3. Il correspond au graphe de la Figure 8.6, avec chaque nœud dupliqué afin de représenter les deux instances des services, et les arcs labélisés avec les valeurs pertinentes issues des compositions de DTC et CTC. compte-tenu du nombre de solutions possibles, il est impossible de montrer l'ensemble des résultats expérimentaux. De plus, les détails sur le déploiement des instances et sur la solution recommandée par NDS présentent peu d'intérêt. La Table 8.3 montre donc le résumé des temps d'exécutions expérimentaux en secondes, obtenus grâce à l'utilisation du *fake service*.

Comme on peut le remarquer, la recommandation fournie par NDS correspond à la meilleure performance (28s), alors que les solutions médianes (les plus probables) montrent des temps d'exécution plus de deux fois plus importants et la pire solution monte à quatre fois plus. De plus, le temps de décision est seulement 3065μs, ce qui atteste d'une très bonne profitabilité de notre approche dans les problèmes de composition de ressources, y compris pour des tâches ayant des temps d'exécution très courts.

8.5 Identification des points d'équilibre

Dans les sections précédentes, nous avons montré que NDS permettaient de calculer des solutions précises et adaptées aux caractéristiques de la superstructure. Cette adaptabilité a été montrée en utilisant trois ressources de types différents dont dépend la pertinence des différents coûts. Mais il est important de noter que cette pertinence n'est pas absolue, mais dépend fortement des capacités de l'infrastructure matérielle. En particulier, l'entrepôt de données distribué, qui dans nos tests est une ressource de type équilibré et versatile, pourrait s'avérer du type communications intensives dans une grille très large échelle basée sur Internet et dont les capacités de communication sont plus limitées. À l'inverse, cette ressource pourrait s'avérer du type calcul intensif à l'intérieur d'une grappe disposant d'un réseau vraiment très haut débit.

Il existe donc un point d'équilibre entre les deux coûts principaux : de calcul et de communication. Ce point d'équilibre correspond à la quantité de données transférées et la quantité de calculs pour lesquelles le temps de transfert est équivalent au temps de calcul. Nous avons mis au point une méthode de recherche simple afin d'identifier ce point d'équilibre. La première étape consiste à évaluer le coût d'un transfert de données moyenné sur l'ensemble des hôtes de l'expérience, et ce pour plusieurs quantités de données différentes (de 1 ko à 100 Mo). Ensuite, pour chacune de ces quantités, nous avons cherché la quantité de calculs menant à un coût équivalent.

L'évaluation des coûts est faite d'une part expérimentalement avec les temps d'exécution du *fake service*, paramétré pour ne faire d'abord que du transfert, puis que du calcul ; et d'autre part grâce aux métriques composées DTC et CTC. Alors que la première évaluation permet d'identifier le point d'équilibre réel de la plateforme, la deuxième permet d'identifier le point d'équilibre perçu par NDS.

Nous avons mené cette expérience, d'une part, sur Grid 5000 avec les nœuds de l'expérimentation, et d'autre part sur une grille de fortune composée de deux machines situées à Toulouse et Lyon. Ces machines sont plus puissantes que celles de Grid 5000 ($CPUs \approx$ 3GHz contre 2GHz). En revanche elles ne disposent pas d'un accès privilégié au réseau RENATER et disposent donc de performances réseau inférieures à celles de Grid 5000 ($BW \approx$ 2.5Mb contre 25Mb inter-grappe).

Le résultat de cette expérience est montré Figure 8.7 montrant les points d'équilibres en deux fois deux courbes, pour les deux cas : expérimental et NDS ; et sur les deux plateformes : Grid 5000 (g5k) et la grille de fortune (GdF). Les quantités de données transférées sont représentées en abscisse, et les quantités de calculs en ordonnée.

On peut, tout d'abord, remarquer que les points d'équilibre évalués par NDS sont sensiblement identiques à ceux observés expérimentalement avec le *fake service*. Ceci prouve l'efficacité des métriques composées pour détecter le point d'équilibre afin de s'adapter à l'infrastructure. On peut néanmoins remarquer un léger écart entre les points d'équilibre réels et évalués. Cet écart marque la zone dans laquelle les métriques composées fournissent un point d'équilibre imparfait. Cependant, même lorsqu'on se situe dans cet écart, les distances calculées ne font pas nécessairement l'objet d'erreurs importantes, car l'influence des deux coûts y est pratiquement équivalente.

De plus, on peut remarquer sur ces courbes deux régimes de part et d'autre des valeurs 10 ko à 100 ko. En effet, le temps de transfert d'une donnée de 1000 octets est égal à celui d'une donnée de

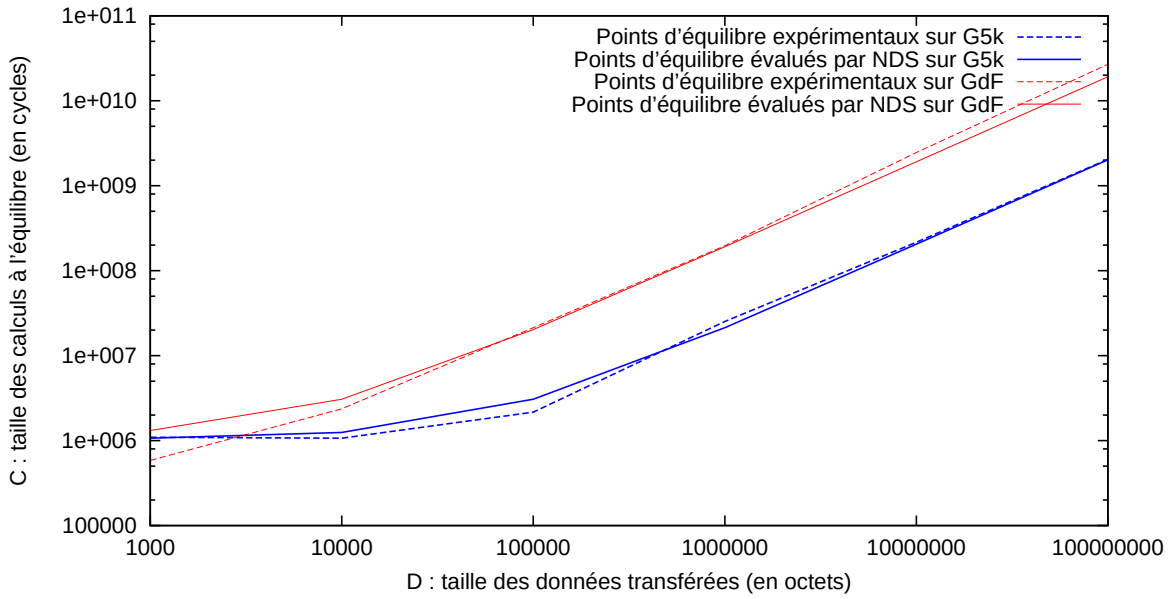


FIG. 8.7 – Point d'équilibre entre coût de transfert et coût de calcul, évalués expérimentalement et par NDS, sur deux plateformes distinctes.

10 ko. 10 ko est en fait la taille généralement observée de la MTU[6] sur l'Internet. Ainsi, quelle que soit la taille de la donnée transmise, si elle est inférieure à la MTU, cela prendra un paquet, et donc le même temps. L'existence de ces deux régimes est prise en compte dans la métrique composée DTC grâce à l'ajout des latences (voir Section 5.1.3) et bien évaluée par NDS. De plus, Grid 5000 possède un réseau très haute performance, la valeur de MTU s'y situe entre 10 ko et 100 ko. Alors que notre grille de fortune, avec son réseau premier prix, a une MTU de 10ko classique. Cette différence est clairement visible sur la figure : les courbes de Grid 5000 ont un régime initial plus étendu.

C'est là le point le plus important de cette expérience : Grid 5000 et notre grille de fortune présentent des points d'équilibre différents : le rapport nombre de cycles par octet transféré est plus important sur notre grille de fortune que sur Grid5000, ce qui est logique, mais bien détecté par NDS. L'adéquation des points d'équilibre expérimentaux et évalués par NDS, et ce dans deux environnements différents, prouve que notre approche permet de s'adapter de façon transparente aux performances de l'infrastructure, et donc à leur dynamicité et à leur évolutions à long terme.

8.6 Synthèse

Dans ce chapitre, nous avons présenté les expérimentations réalisées sur le service web NDS qui implémente la création des $\mathcal{CNP}$-Graphes et des algorithmes capables de les exploiter afin de fournir des recommandations pour différents problèmes liés à la répartition des ressources.

Nous avons couvert un spectre représentatif de problématique en expérimentant trois problèmes : sélection, déploiement et composition de ressources ; et trois types de ressources sujettes : communications intensives, calculs intensifs et équilibré/versatile.

[6]Maximum Transmission Unit : taille maximum qu'un paquet IP peut transporter

Afin d'évaluer la pertinence des recommandations fournies par NDS, nous avons décidé d'évaluer expérimentalement toutes les solutions candidates sur Grid 5000, une grille réelle large échelle.

Les recommandations fournies par NDS se sont avérées très pertinentes, en obtenant systématiquement les meilleurs temps d'exécution expérimentaux. Cette pertinence est due à une double capacité d'adaptation : d'une part à l'infrastructure matérielle, en sélectionnant les hôtes les plus appropriés en fonction de leurs capacités et de leur position dans le réseau ; et d'autre part à la superstructure, en fournissant des recommandations différentes en fonction de la ressource sujette.

De plus, les requêtes NDS utilisées pour résoudre les problèmes sont très simples, ce qui atteste de sa bonne utilisabilité. Enfin, les temps d'exécution de NDS se sont avérés très faibles, ce qui atteste de sa bonne profitabilité.

9

Intégration de NDS dans une superstructure

NDS peut être vu comme une surcouche-interface aux outils de surveillance tels qu'NWS et MDS. De plus, sa capacité à résoudre les problèmes liés à la distribution des ressources le rend utile aux composants logiciels de tous les niveaux de la superstructure. Ainsi, l'intégration de NDS dans une superstructure est transversale.

Ce chapitre illustre l'intégration de NDS dans une superstructure en reprenant les cas d'usages du projet GGM présentés Chapitre 2. Ces derniers sont modifiés pour prendre en compte l'utilisation de NDS. Il s'agit la de spécifications, seule l'intégration avec le service d'entrepôt distribué a été effectivement implémentée. Trois intérêts sont afférents à l'intégration de NDS dans une superstructure.

Le premier intérêt est d'unifier l'accès aux outils de surveillance en ajoutant une couche d'abstraction. Cet intérêt est illustré par l'interfaçage avec le système d'exécution distribuée de requête, Section 9.2.

Le deuxième intérêt est que les développeurs, et leur applications, sont déchargés des phases d'évaluation des différentes solutions candidates. Cet aspect est illustré Section 9.3 avec le service de cache distribué.

Le troisième intérêt est qu'il renforce la souplesse et l'évolutivité de la plateforme en faisant le lien entre les différents composants et leurs développeurs et en permettant de s'adapter automatiquement, sinon facilement, aux évolutions de l'infrastructure et à l'apparition de nouveaux besoins, en terme de répartition des ressources, au niveau de la superstructure. Cet intérêt est illustré par l'intégration avec l'entrepôt distribué, Section 9.4.

9.1 Vue d'ensemble de l'intégration d'NDS dans la superstructure GGM

La Figure 9.1 montre la place que prend NDS dans la superstructure : NDS est central, tous les services sont amenée à interagir avec lui : le service de caches collaboratifs afin de prendre les décisions de déploiement et sélection des données stockées, l'entrepôt distribué afin de résoudre le problème de l'agrégation, le service d'exécution des requêtes afin d'accéder aux informations de surveillance et, bien que cela n'ai pas été spécifié, le service de fouille peut utiliser NDS pour découper ses données et distribuer ses calculs.

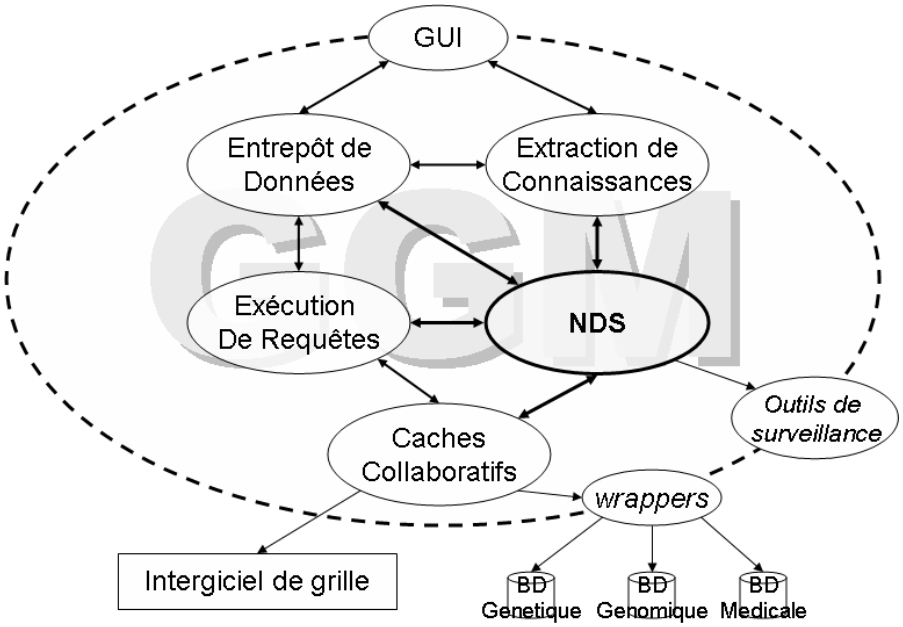


FIG. 9.1 – Vue générale de l'intégration d'NDS dans la superstructure GGM

Cette place centrale montre l'importance de l'intégration d'NDS au sein d'une super structure : la même importance que tiennent les problématiques liées à la répartition des ressources dans les environnements distribués.

9.2 Intégration avec le service d'exécution des requêtes

La Figure 9.2 montre l'intégration de NDS avec le service d'exécution distribué de requêtes. Le problème est ici que ce service est implémenté grâce à des agents mobiles qui embarquent des modèles de coût servant à décider des réplicas de données à utiliser et des processeurs à utiliser pour effectuer les calculs. Or, ces modèles de coût sont basés sur des informations spécifiques telles que le débit des disques. Ces informations ne sont pas surveillées par les outils de surveillance classiques. L'utilisation de plusieurs d'entre eux est donc nécessaire. Or, les agents mobiles, pour des raison de taille, ne peuvent embarquer les interfaces pour ne nombreux outils de surveillance. De plus, sa nature distribuée est peu efficace pour maintenir à jour la liste des outils à même de fournir chacune des informations. L'intégration d'NDS permet de résoudre ces problèmes en présentant une interface unifiée à tous les outils de surveillance.

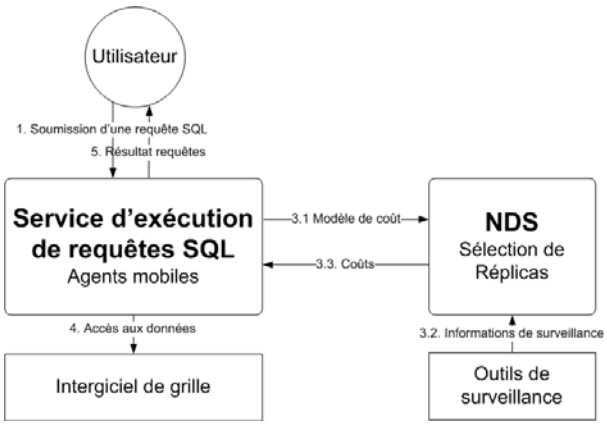


FIG. 9.2 – Intégration du service d'exécution distribuée de requête SQL avec NDS

9.3 Intégration avec le service de caches collaboratifs

La Figure 9.3 rappelle le scénario initial déclenché par la demande d'un utilisateur de stocker des données dans le service de caches collaboratifs. Elle montre le même scénario après intégration d'NDS dans la superstructure. Le service de caches collaboratifs est maintenant déchargé de la phase de décision et peut utiliser les recommandations fournies par NDS afin de stocker les données selon ses propres modèles de coûts sans se soucier de leur évaluation.

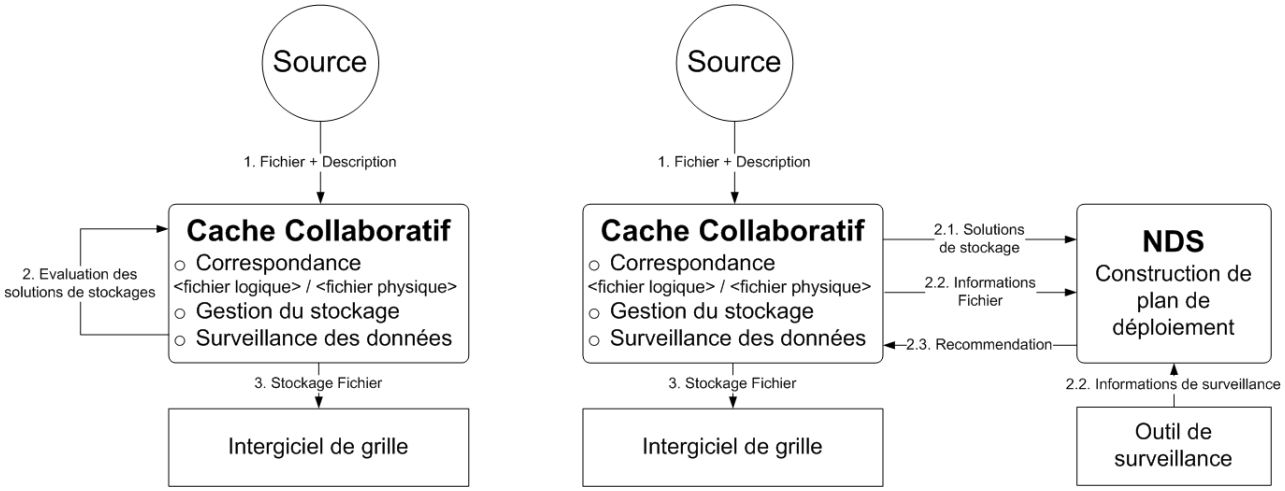


FIG. 9.3 – Le problème du déploiement vu depuis la superstructure GGM, avant et après intégration de NDS

Les Figures 9.4 et 9.5 illustrent l'utilisation de NDS par le service de cache pour les problèmes de la sélection et de la composition de services d'adaptation de contenus multimédias : la phase d'évaluation des différentes solutions est remplacée par l'interrogation d'NDS.

On peut noter que cette intégration est parfaitement identique pour les deux problème. Ceci est un point fort de notre approche : une fois NDS intégré et son fonctionnement acquis, il peut être utilisé pour résoudre indifféremment une vaste étendue de problèmes, déchargeant totalement son utilisateur de ces considérations. L'utilisation de NDS permet donc aux différents composants de la superstructure de d'adapter plus facilement aux évolutions possibles de la plateforme, pas seulement

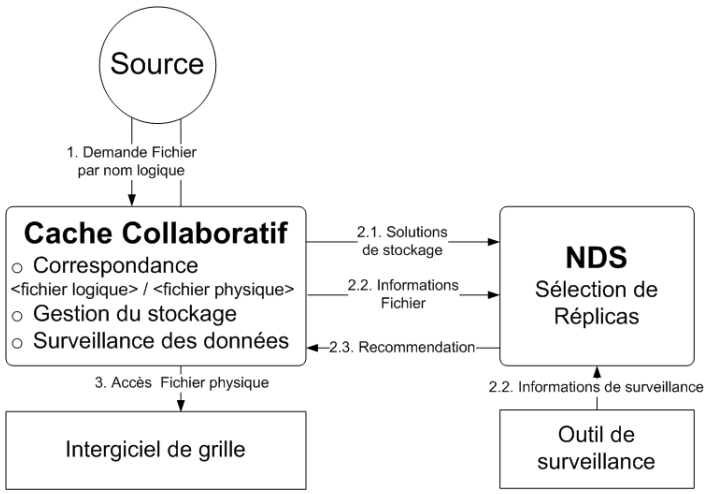


FIG. 9.4 – Le problème de la sélection vu depuis la superstructure GGM après intégration de NDS

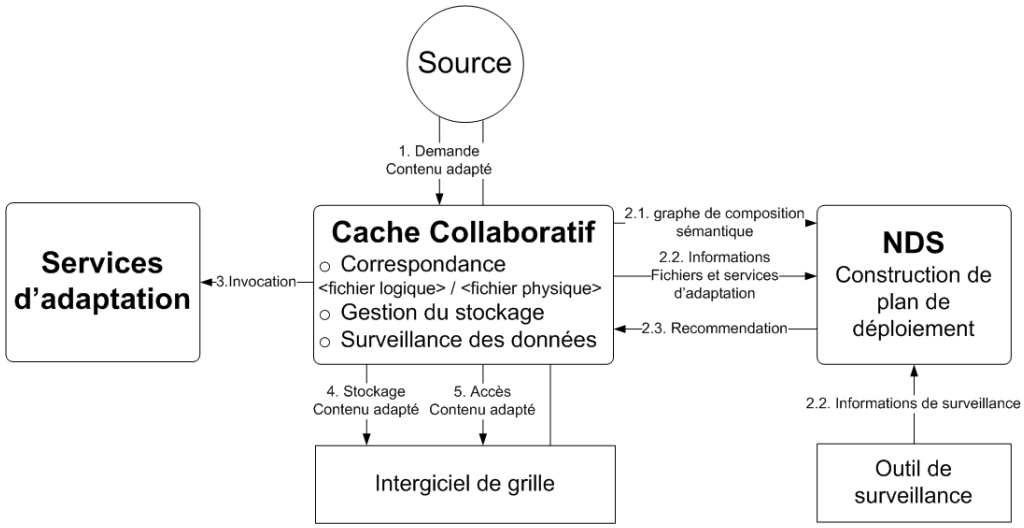


FIG. 9.5 – Le problème de la composition vu depuis la superstructure GGM après intégration de NDS

en ce qui concerne les performances de l'infrastructure, mais aussi en cas d'apparition de nouvelles problématiques liées à la répartition des ressources.

9.4 Intégration avec l'entrepôt de données distribué

La Figure 9.6 rappelle le cas d'usage déclenché par une demande d'agrégat auprès de l'entrepôt de données distribué. Elle montre également ce même cas d'usage après l'intégration de NDS. Dans ce cas, non seulement NDS décharge l'entrepôt de la phase de choix du plan d'agrégation, mais en plus il permet de limiter ses interactions avec le service de caches collaboratifs. Cet aspect présente deux avantages. Le premier est qu'il limite le couplage et les efforts d'intégration entre ces deux services, favorisant ainsi leur réutilisabilité et leur développement. Le deuxième est qu'il concentre le lieu des décisions lié à la répartition des ressources. Ainsi, le risque de stratégies concurrentes, voire antagonistes, est limité. De plus, ces stratégies étant formulées au travers des $\mathcal{CNP}$-Graphes,

la collaboration entre ces deux services est grandement facilitée. Par exemple, le développeur de l'entrepôt de données distribué pourra intégrer ses propres métriques dans NDS, lesquelles seront réutilisées par le service de caches collaboratifs pour placer les agrégats. Il pourra également les modifier de façon totalement transparente pour leurs utilisateurs. On favorise ainsi l'homogénéité des stratégies de gestion de la distribution des ressources en limitant les efforts de concertation de la part des différents acteurs.

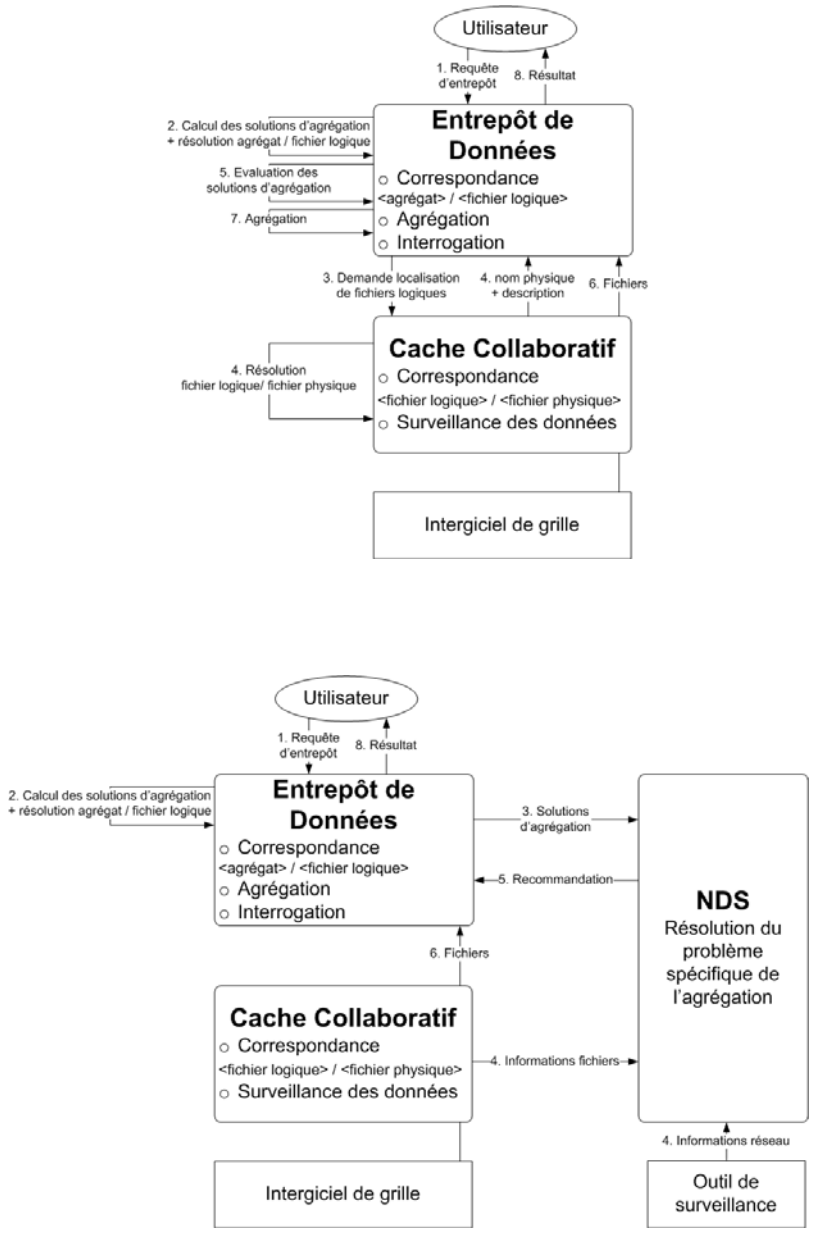


FIG. 9.6 – Le problème de l'agrégation vu depuis la superstructure GGM, avant et après l'intégration de NDS

9.5 Synthèse

Dans ce chapitre, nous avons détaillé l'intégration d'NDS dans la supertructure GGM, ainsi que la modification des différents cas d'usage.

L'intérêt de cette intégration s'articule autour de trois axes :

- NDS représente une interface unifiée à tout outil de surveillance, ce qui s'avère très utile lorsque les composants de la superstructure ont des besoins spécifiques impliquant l'utilisation de plusieurs de ces outils.
- NDS décharge les développeurs et leurs composants des phases d'évaluation des différentes solutions candidates aux problèmes liés à la répartition des ressources. En plus d'un gain en terme de temps de développement, et donc de retour sur investissement, cette décharge permet de favoriser l'adaptabilité de la plateforme face à la dynamicité de l'infrastructure, ainsi que son évolutivité face à l'apparition de nouveaux besoins en terme de problématiques liées à la répartition des ressources.
- NDS permet de faire le lien entre les différents composants et leurs développeurs respectifs, favorisant ainsi une collaboration transparente et limitant les efforts de concertation nécessaires à la mise en place des stratégies liées à la répartition des ressources.

Enfin, cette intégration est grandement facilitée par l'interface d'NDS qui se résume à deux fonctions, la première permettant d'accéder aux informations de surveillance et la deuxième permettant de générer les $\mathcal{CNP}$-Graphes.

Dans le chapitre suivant, nous discutons des avantages et inconvénients de l'approche proposée tout au long de cette partie.

10

Discussion et conclusion de la résolution des problèmes liés à la répartition des ressources par l'utilisation de $\mathcal{CNP}$-Graphes

Dans cette partie, nous avons présenté le cœur de notre approche : une modélisation de la logique des problèmes liés à la répartition des ressources sous forme de $\mathcal{CNP}$-Graphes compacts, la transformation de ces derniers en $\mathcal{CNP}$-Graphes modélisant la réalité physique des problèmes, et leur exploitation par des algorithmes. Des expérimentations en environnement réels de grille ont montré la pertinence des solutions obtenues par ces algorithmes.

Dans ce chapitre, nous discutons notre approche en détaillant quatre aspects : la capacité de notre système à passer à l'échelle en Section 10.1, en section 10.2 les contraintes relatives à l'utilisation d'outils de surveiallance, et enfin en section 10.3 de l'impact du calibrage et des difficultés d'identifier les informations servant à modéliser les problèmes.

10.1 De la complexité et du passage à l'échelle

Une critique qui s'impose concerne le nombre d'hôtes utilisé dans les expérimentations (16) qui est trop limité pour prouver la scalabilité de notre approche. Ce nombre a du être limité afin de pouvoir évaluer effectivement les temps d'exécution expérimentaux de toutes les solutions candidates. On peut en effet redouter une augmentation des temps de décision avec l'augmentation du nombre d'hôtes impliqués, en particulier lors de l'utilisation d'algorithmes à complexité exponentielle. Nous pouvons cependant formuler quelques remarques :

- Cela n'est pas un problème pour les décisions en temps linéaire tels que la sélection ou la

composition de ressources.

- En revanche, cet aspect doit être discuté pour le problème du déploiement, car le nombre de solutions candidates augmente significativement en fonction du nombre de candidats à l'hébergement. Une première observation est que plus nombreux sont les hôtes impliqués, plus des décisions précises seront profitables, pour lesquelles on peut se permettre des temps de décision relativement importants.
- Des techniques simples permettent de produire des $\mathcal{CNP}$-Graphes satisfaisant les propriétés euclidiennes. Par exemple grâce à l'optimisation expliquée Section 6.3, on peut générer des $\mathcal{CNP}$-Graphe bipartites. Ces derniers sont des graphes simples et acycliques, qui satisfont donc par définition la réflexivité, la symétrie et l'inégalité triangulaire. On peut alors utiliser sans crainte des algorithmes avancés exploitant ces propriétés afin de réduire la complexité et donc les temps de décision.
- Enfin, un autre moyen de réduire les temps de décisions est de produire des hypergraphes dans lesquels chaque nœud représente en fait un ensemble d'hôtes homogènes hautement connectés. Considérer ainsi les grappes présents dans la grille, plutôt que chaque hôte individuellement, permet de réduire le nombre de nœuds du $\mathcal{CNP}$-Graphe et donc les temps de décision. Cette approche n'est pas pertinente dans les grilles hautement hétérogènes et éparses, telles que les *Desktop Grids* utilisant les machines d'utilisateurs volontaires issus du grand public. En revanche, elle est extrêmement réaliste dans les grilles industrielles et scientifiques actuelles, telles que Grid 5000 qui a pour but de réunir 5000 machines mais compte aujourd'hui *seulement* 9 sites et 26 grappes.

10.2 De l'utilisation de services de surveillance

Un autre aspect de notre approche devant être discuté concerne l'utilisation d'outils de surveillance fournissant les métriques brutes. Or, ces outils sont réputés pour leur consommation de ressources. La discussion sur la nécessité pour une architecture de grille de disposer d'un tel outil est hors de nos prérogatives, nous pouvons seulement souligner les grands avantages qu'elle confère : suivi des pannes, connaissance temps réel des performances dynamiques telles que perçues par les applications... Il ne fait alors aucun doute quant au fait qu'une grille surveillée puisse tirer profit de l'utilisation de NDS, comme celui-ci est léger et fournit des décisions précises et adaptées avec peu d'efforts utilisateur. L'objectif initial qui était de combler l'écart entre la superstructure logicielle et l'infrastructure logicielle, connue au travers des informations de surveillance, est donc bien atteint.

De plus, comme on peut le remarquer sur les temps de décisions donnés Section 8, la récupération des métriques brutes tient une place important dans les temps de résolution des problèmes. L'utilisation de techniques de cache pourrait être envisagée pour réduire ce coût. Le code de NDS en est aujourd'hui à sa version de test, ce qui permet d'espérer des améliorations avec des efforts d'implémentation supplémentaires. Ces efforts permettrons d'augmenter son rendement, par exemple afin de le rendre également compétitif sur les machines obsolètes qui peuvent être mise en production sur les *grilles de fortune*, souvent utilisées par les équipe scientifiques pour tester leur travaux.

Pour l'instant, notre approche n'intègre pas la sécurité. Cet aspect est clairement hors de nos considérations, mais peut prendre de l'importance dans cadre d'utilisation industrielle. Il peut présenter

certains problèmes, car les informations de surveillance peuvent être considérées comme critiques dans certains environnements. NDS peut se coupler avec le gestionnaire de sécurité de la grille afin d'accéder aux services de surveillance sécurisés tels que le MDS de Globus. Cependant, cet aspect peut poser plus de problèmes avec les outils ne gérant pas la sécurité eux-mêmes, tels que NWS.

10.3 Du calibrage et de la définition des propriétés de tâches et fonctions distance

Nous avons déjà décrit les difficultés qui pouvaient survenir lors du processus d'identification des valeurs TPS, et en particulier du nombre de cycles consommés par une application et servant à renseigner la métrique composée CTC. Bien que les utilisateurs puissent s'appuyer sur des outils et techniques existantes pour évaluer ces valeurs, il est possible qu'une erreur d'appréciation dégrade la précision des distances calculées par NDS.

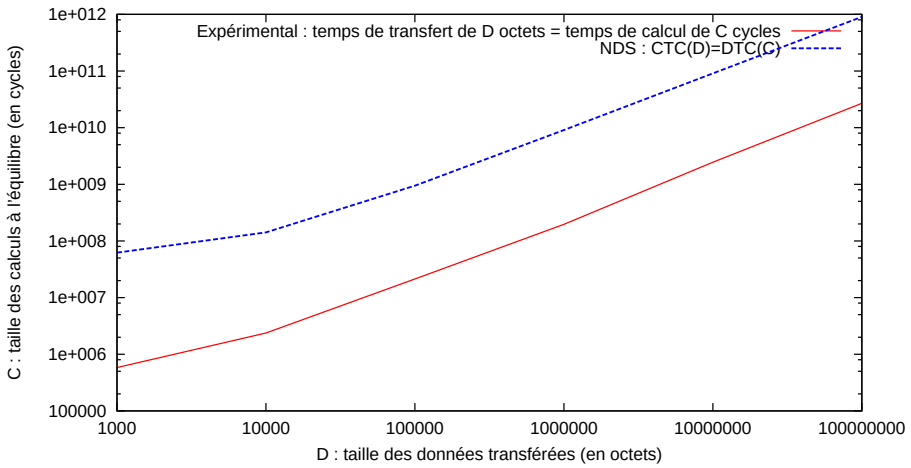


FIG. 10.1 – Point d'équilibre entre coût de transfert et coût de calcul sans calibrage

Pour nos expérimentations, nous avons utilisé une facteur de calibration $\mathcal{C}$. Afin d'évaluer l'importance de ce facteur, nous avons effectué l'identification des points d'équilibre, de la même manière qu'en Section 8.5, mais sans utiliser le facteur de calibrage. Les résultats sont montrés Figure 10.1. Sans utilisation du facteur de calibration, on peut observer une augmentation de l'écart entre les points d'équilibre réels identifiés expérimentalement et ceux identifiés par NDS. Il est clair que cet écart devrait être d'autant plus important que l'évaluation du nombre de cycles est imprécise. Néanmoins, les ressources se situant hors de l'écart de ces deux courbes feront l'objet de distances pertinentes.

En effet, notre solution évalue le classement des différentes solutions candidates plutôt que la prédiction de leurs temps d'exécution réels. Le principal inconvénient de cet approche est qu'il est difficile de déclarer des contraintes du type « temps d'exécution < x secondes ». En revanche, si les techniques de prédiction sont souvent complexes et basées sur des informations précises et complètes, les techniques de classement sont beaucoup plus souples et tolérantes, et peuvent être mises en place sur des informations partielles ou approximatives, du moment qu'elles sont uniformes. Cela confère à notre approche une certaine tolérance aux imprécisions dans les paramètres utilisés pour construire les fonctions distances.

Enfin, la définition des propriétés de tâche et des fonctions distance dans les ensembles TPS et dfS peut s'avérer être un processus délicat dans les cas les plus complexes. Pour alléger ce processus, nous prévoyons d'étudier les techniques permettant de définir ces informations automatiquement. Un effort important est d'ailleurs fourni dans ce sens au travers du standard *Web Service Distributed Management* (WSDM)[1] développé par OASIS afin d'embarquer ces informations directement dans chacun des services de la superstructure.

10.4 Synthèse de la discussion à propos des $\mathcal{CNP}$-Graphes

Ce chapitre de discussion nous a permis d'éclaircir quatre points concernant notre contribution principale.

Le premier point concerne son passage à l'échelle en terme de nombre de sommets des $\mathcal{CNP}$-Graphes. En réalité, lorsque ce nombre de sommets est important et l'algorithme de résolution complexe, cela signifie que le problème est d'un importance cruciale pour la plateforme. On peut donc se permettre des temps de résolution plus importants. De plus, des techniques simples permettent de limiter ce nombre de sommets, par exemple en modélisant les grappes au lieu de chaque hôte individuellement.

Le second point concerne le calibrage afin d'obtenir des informations précises sur le comportement des applications, et plus particulièrement sur la mesure du nombre de cycles CPU qu'elles consomment. Notre système permet une certaine tolérance dans l'imprécision de ces mesures, en particulier pour les applications présentant des besoins clairement orientés en terme de calcul ou bien de transfert. En revanche, cette phase de calibration garde une certaine importance pour les recommandations obtenues par l'utilisation d'NDS lorsque le comportement de l'application sujette n'est pas clair (c.-à-d. lorsque les coûts de calcul et de communication sont en concurrence).

Le troisième point concerne l'utilisation d'outils de surveillance de l'infrastructure, dont l'utilisation peut s'avérer relativement coûteuse. L'observation principale est que si cette utilisation est jugée pertinente, alors l'utilisation de NDS ne fait aucun doute. De plus, NDS améliore la profitabilité et l'utilisabilité des informations de surveillance, et aide donc à rentabiliser la mise en production d'un ou plusieurs outils de surveillance.

Enfin, le quatrième point aborde la difficulté de définir les informations de description des ressources sujettes, dans les ensembles TPS et dfS. Bien que notre approche fasse preuve d'une certaine tolérance dans ces informations, l'étape de leur définition est importante et obligatoire. Nous étudierons dans les travaux futurs les méthodes qui permettent de les définir automatiquement et de les inclure dans les informations standardisées de description des services.

[1] http ://www.oasis-open.org/committees/wsdm/

10.5 Synthèse à propos des $\mathcal{CNP}$-Graphes

Cette section conclue la partie principale de cette thèse. Dans cette partie, nous avons présenté une méthode de modélisation générique des problèmes liés à la répartition des ressources sur grille. Cette modélisation fait l'objet de graphes appelé *Computer Network Problème Graphs* ($\mathcal{CNP}$-Graphes). Nous avons montré comment un utilisateur pouvait, avec peu d'effort et une expertise limitée, modéliser un large spectre de problèmes sous leur forme logique, ce qui confère à notre approche une bonne utilisabilité. Nous avons détaillé la modélisation des coûts, qui permet de définir aux utilisateurs de représenter aisément un large choix de besoins et d'objectifs, mêlant performances grâce aux métriques composées, aspects sémantiques ou encore financiers. Ainsi, notre approche est grandement adaptable. Nous avons également montré comment ces deux modélisations pouvaient être utilisées pour générer automatiquement un graphe représentatif de la réalité physique des problèmes et comment ces graphes pouvaient être validés pour l'exploitation par un algorithme donné.

Nous avons ensuite détaillé l'implémentation de notre solution dans un service web appelé NDS, pour *Network Distance Service*. Cette implémentation met l'accent sur l'expressivité et la facilité d'extension et d'utilisation de notre approche. Ce service a servi à expérimenter notre approche dans un environnement réel de grille, pour les trois problèmes génériques liés à la répartition des ressources sur grille : sélection, déploiement et composition de ressources. L'utilisation de différents types représentatifs de ressource nous a permis de montrer que NDS est adaptatif puisqu'il permet d'obtenir des solutions différentes en fonction des caractéristiques de la superstructure. Nous avons également montré que ces solutions tiennent compte de la topologie de l'infrastructure matérielle puisqu'elles correspondent systématiquement aux meilleures performances expérimentales. De plus, les requêtes utilisées pour résoudre les problèmes sont très simples, ce qui montre l'utilisabilité de notre système. Enfin, les temps d'exécution extrêmement courts de NDS témoignent de sa bonne profitabilité.

Pour conclure, nous pouvons dire que notre approche permet de rendre l'utilisation des services de surveillance beaucoup plus profitable, en mettant à la disposition des différents acteurs des grilles, un outil permettant de résoudre aisément et de façon générique, adaptable et adaptative les problèmes liés à la répartition des ressources. De plus, notre outil permet de rendre transparent les changements, même importants, qui peuvent survenir au niveau de l'infrastructure, favorisant ainsi la pérennité des plateformes. Notre approche constitue donc un pas vers la réconciliation des architectures logicielles de très haut niveau que sont les architectures orientées services et leurs infrastructures matérielles.

Dans la partie suivante, nous détaillerons plusieurs algorithmes que les $\mathcal{CNP}$-Graphes ont permis de concevoir et développer.

Troisième partie

Algorithmes de placement de ressources

Nous avons vu dans l'introduction l'importance des tâches de déploiement de ressources logicielles sur l'infrastructure. Nous avons également vu dans l'état de l'art que l'établissement des plans de déploiement était une des étapes critiques du déploiement qui était le moins bien traité par les outils de déploiement. Or les plans de déploiement conditionnent fortement les performances des superstructures, un bon plan de déploiement garantissant la pleine utilisation des capacités de la plateforme et une qualité de service optimale.

On peut identifier deux problématiques sous-jacentes à l'établissement d'un plan de déploiement : premièrement le dimensionnement, qui consiste à définir pour chaque ressource quel est le nombre optimal de réplicas à déployer ; deuxièmement le placement, qui consiste à sélectionner un hôte destiné à héberger un de ces réplicas.

Nous avons également montré dans le chapitre précédent comment on pouvait exploiter les $\mathcal{CNP}$-Graphes en réutilisant des algorithmes issus de la théorie de la localisation discrète afin de dimensionner et placer une ressource donnée. Un avantage supplémentaire de notre approche est que la modélisation des problèmes liés à la répartition des ressources sous forme de graphe ouvre de grandes perspectives algorithmiques. En effet, les processus de conception d'un système de résolution de ces problèmes sont généralement longs et complexes. Disposer d'une modélisation adaptée de l'infrastructure et des problèmes permet de grandement faciliter ces processus.

Dans cette partie nous présentons deux algorithmes de placement que nous avons développés grâce à l'approche des $\mathcal{CNP}$-Graphes. Le premier, présenté Chapitre 11, est conçu pour générer des plans de déploiement d'une superstructure de grille complète, donc composée de plusieurs ressources interdépendantes. Le second, présenté Chapitre 12 est conçu pour gérer un dimensionnement et un placement dynamique de contenus dont la réplication est fortement contrainte. Il est particulièrement adapté aux environnements de grilles pervasives et a été amélioré , Chapitre 13 par l'intégration de prédictions de déplacements de clients mobiles.

11

MRKM : Multi-Ressources k-Médians

11.1 Motivation

Nous avons montré comment l'utilisation d'un algorithme trivial résolvant le problème des k-médians, présenté dans [Reese 06], permettait de dimensionner et placer optimalement une ressource sur une infrastructure avec une grande souplesse dans le choix des critères à optimiser. Des expériences en environnement réel de grille ont montré son efficacité.

Néanmoins, la principale critique adressable au problème des k-médians, dont l'Algorithme 1 est la résolution triviale, est qu'il ne prend en compte qu'une seule ressource. Or, un des scénarios critique de déploiement consiste à déployer une partie, voire toute une structure logicielle, ce qui implique de considérer non pas des ressources indépendantes, mais plusieurs ressources différentes interdépendantes.

11.1.1 Algorithme trivial des k-médians

La Figure 11.1 montre un graphe simple représentatif d'un problème de déploiement d'une ressource : $Vd = \{S1, S2\}$ contient les emplacements possibles, $Vs = \{C1, C2\}$ les clients. Les arcs sont labélisés avec le coût d'utilisation par le client source de la ressource sujette à l'empalcement destination. Avec $k = 1$, la k-partition de Vd est $\mathcal{P}_k(Vd) = \{\{S1\}, \{S2\}\}$, elle contient toutes les k-combinaisons de Vd, soit toutes les solutions candidates du problème (c.-à-d.. toutes les solutions de placement possibles). Le

L'Algorithme 1 est une résolution triviale du problème des k-médians, qui vise à minimiser le critère suivant :

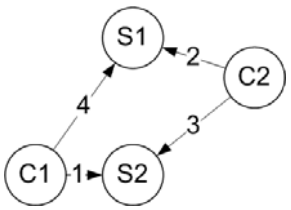


FIG. 11.1 – Exemple de graphe d'un problème de déploiement

$$critere(solution) = \sum_{Ci \in Vs} \min_{Sj \in solution} d(Ci, Sj)$$

Sur l'exemple de la Figure 11.1, on peut voir que $critere(\{S1\}) = d(C1, S1)+d(C2, S1) = 4+2 = 6$ et $critere(\{S2\}) = d(C1, S2)+d(C2, S2) = 1+3 = 4$. $\{S2\}$ est donc la solution optimale à ce problème.

On peut imaginer la conception d'un plan de déploiement de plusieurs ressources comme la somme de plusieurs plans de déploiement indépendants, calculé successivement pour chacune des ressources. Cette approche présente deux faiblesses.

Tout d'abord, elle ne prend pas en compte la charge des hôtes après déploiement. Le risque encouru est que toutes les ressources soient assignées à la machine la plus puissante de la plateforme et non pas répartie uniformément sur son ensemble. Ceci représente un choix optimal dans une logique de décisions indépendantes, mais un problème évident de déséquilibre des charges. On peut éviter ce problème en attendant, après le déploiement de chaque ressource, que l'infrastructure matérielle se stabilise afin que les outils de surveillance reflètent l'impact de ce nouveau déploiement et que les décisions suivantes soient prises en conséquence. Mais cette méthode présente un problème de performance et de passage à l'échelle en terme du nombre de ressources différentes à déployer. Il faut donc prendre en compte cet aspect directement dans l'algorithme de création du plan de déploiement.

De plus, un aspect qui prend une importance accrue dans les architectures orientées services est que les ressources ne sont pas des éléments isolés, mais sont amenées à collaborer et interagir tout au long de leur cycle de vie. Prendre en compte ces interactions est primordial afin de ne pas placer des ressources en forte collaboration à des points trop éloignés de l'infrastructure matérielle.

C'est pourquoi nous avons développé une adaptation de cet algorithme, baptisé $MRKM$ pour *Multi-Ressources k-Médians*. Le but de celui-ci est de prendre en compte la surcharge impliquée par chaque déploiement ainsi que les interactions potentielles entre ressources.

11.2 Modélisation

- $\mathcal{CNP}_r = (Vs \cup Vd,\ Vs \times Vd,\ d_r)$ le $\mathcal{CNP}$-Graphe relatif à la ressource r
  - Vs l'ensemble des origines des arêtes, représentant les clients
  - Vd l'ensemble des destinations des arêtes, candidats à l'hébergement

Algorithme 1 kmedians(k, Vs, Vd, d) : $recom$

Entrées :
k : Entier, le nombre de ressources à déployer
Vs : ensemble de nœuds, origines des arcs, clients
Vd : ensemble de nœuds, destinations des arcs, candidats à l'hébergement
d : matrice de $|Vs \times Vd|$ Réels, distance réelle labélisant les arcs

Sorties :
$recom$: ensemble de k Nœuds, représentation des k recommandations de placement

Variables Locales :
$critere$, $recomCritere$: Réel, les critères d'évaluation des solutions
$solution$: ensemble de k nœuds, représentation les solutions candidates
hs, hd : Entier, id. de nœuds
$distMin$: Réel

Description :
pour tout $solution$ **faire**
  Calculer $critere$ de $solution$
  si $critere$ est minimal **alors**
    Mémoriser $solution$ dans $recom$
  finsi
fin pour
Retourner $recom$.

Détails :
$recomCritere \leftarrow \infty$
//pour toute les k-combinaisons de Vd)
pour tout $solution \in \mathcal{P}_k(Vd)$ **faire**
  $critere \leftarrow 0$ //pour tous les clients
  pour tout $hs = 1$ à $|Vs|$ **faire**
//rechercher le plus proche serveur de la $solution$ candidate
    $distMin \leftarrow \infty$
    pour tout $hd = 1$ à k **faire**
      si $d(hs, solution(hd)) < distMin$ **alors**
        $distMin \leftarrow d(hs, solution(hd))$
      finsi
    fin pour
//ajouter la distance au plus proche serveur à $critere$
    $critere \leftarrow critere + distMin$
  fin pour
//si $critere$ minimum
  si $critere < recomCritere$ **alors**
//mémoriser la $solution$
    $recomCritere \leftarrow critere$
    $recom \leftarrow solution$
  finsi
fin pour
Retourner $recom$.

– $d_r(hs, hd)$ la distance réelle pour la ressource r labélisant l'arc $(hs, hd) \in Vs \times Vd$
On peut remarquer que Vd et Vs sont communs aux $\mathcal{CNP}$-Graphes de toutes les ressources.

- $R = \{r1, \ldots, rn\}$ est l'ensemble des n ressources à déployer.
- $amount(r)$ le nombre d'instances de la ressource r devant être déployées.
- $cost(r)$ est le coût associé à la ressource r.
- $cost'(r)$ est le coût relatif associé à la ressource R.
- $load(h)$ la charge potentielle associée à l'hôte h.
- $aff(ri, rj)$ représente l'affinité de la ressource r_i avec la ressource r_j.
- $aff'(ri, rj)$ représente l'affinité relative de la ressource r_i avec la ressource r_j.
- $aff''(r, h)$ représente l'affinité potentielle de la ressource r avec l'hôte h.
- $recom(r)$ est un ensemble d'hôtes représentant les recommandations de $MRKM$ pour le placement de la ressource r.
- $depl(h)$ est l'ensemble des ressources figurant dans les recommandations de placement sur l'hôte h.

11.2.1 Charge potentielle

Plutôt que de procéder au déploiement réel des ressources pour évaluer leur impact sur l'infrastructure matérielle, nous avons décidé de définir une notion de *charge potentielle* reflétant la charge additionnelle impliquée par les recommandations de placement de $MRKM$. Cette charge additionnelle est fonction du coût de chaque ressource déployée, de telle façon que déployer plusieurs ressources légères soit équivalent à déployer une ressource lourde.

Coût d'hébergement d'une ressource

Cette notion de charge potentielle est basée sur la notion de coût d'hébergement d'une ressource. Ce coût doit ête défini indépendamment des capacités de l'infrastructure. Par exemple, on peut définir ce coût comme la somme des valeurs TPS décrivant la ressource, telles que définie dans le $\mathcal{CNP}$-Graphes compact initial (voir Section 5.3) : la taille d'une donnée représente bien son coût d'hébergement ; la somme des tailles de données en entrée et sortie, ainsi que le nombre de cycles CPU. C'est donc une fonction associant chaque ressource à une valeur réelle :

Définition III.1 *Coût d'hébergement d'une ressource*

$$cost : R \rightarrow \mathbb{R}$$

Coût relatif d'hébergement d'une ressource

Un problème important qui se pose ici concerne la méthode d'évaluation du coût de chaque ressource. Il faut bien préciser ici qu'on ne considère pas le coût de déploiement, mais le coût d'hébergement de la ressource une fois déployée. Cette évaluation peut être complexe et comporte un risque important d'imprécision. C'est pourquoi nous avons défini la notion de *coût relatif*.

Définition III.2 *Coût relatif d'une ressource*

$$\begin{aligned} cost' : R &\rightarrow [0,1] \\ r &\mapsto \frac{cost(r)}{\sum_{r_i \in R} cost(r_i)} \end{aligned}$$

Ce coût relatif permet de résorber les imprécisions de la méthode employée pour définir les coûts d'hébergement et permet ainsi sa simplification. Il suffit en effet que cette méthode soit uniforme pour l'ensemble des ressources et respecte la proportion des coûts pour que les coûts relatifs soient corrects. Il faut tout de même noter que même cette simplification ne rend pas triviale la définition des coûts d'hébergement. Cette opération doit tout de même être conduite précautionneusement.

Charge potentielle d'un hôte

On peut utiliser la notion de coût relatif d'hébergement des ressources pour définir la notion de charge potentielle des hôtes. La charge potentielle d'un hôte correspond à la somme des coûts relatifs des ressources figurant dans les recommandations d'hébergement sur cet hôte :

Définition III.3 *Charge potentielle de l'hôte h*

$$load(h) = \sum_{R_i \in depl(h)} cost'(R_i)$$

Un aspect important ici est que nous ne nous intéressons pas aux prédictions des charges réelles des hôtes après déploiement. Ainsi, nous n'avons pas de garantie qu'un hôte ne puisse être saturé. Il faut donc noter que si cette infrastructure ne peut supporter la charge induite par les ressources à déployer, nous ne serons pas en capacité de le détecter. En revanche, nous équilibrerons les charges en attribuant une charge potentielle plus importante au machines plus performantes/disponibles, et ce en respect avec la proportion de la charge globale des ressources à déployer et des performances globales offertes pas l'infrastructure matérielle. Ainsi, nous repousserons autant que possible la saturation de la plateforme.

11.2.2 Affinité entre ressources

Le deuxième aspect d'importance que nous devons prendre en compte est celui de l'interaction entre les ressources. Comme déjà discuté précédemment Section 1.2, les tâches utilisateurs dans les

architectures orientées services sont bien souvent basées sur la composition de plusieurs ressources. Par exemple, des données sont amenées à être utilisées ensemble plus que d'autres, des services interagiront avec certains autres ou utiliseront telle base de donnée. Les ressources de la superstructure sont donc amenées à interagir ensemble durant leur cycle de vie. Nous avons donc défini une notion d'affinité entre ressources.

Affinité entre ressources

La notion d'affinité entre ressources permet de quantifier les interactions entre chaque couple de ressources. Une approche basique consiste à définir l'affinité entre deux ressources comme la fréquence des requêtes échangées entre ces deux ressources. C'est donc une fonction qui associe un réel à chaque couple de ressources.

Définition III.4 *Affinité entre ressources*

$$aff : R \times R \rightarrow \mathbb{R}$$

Ici encore le problème de la précision des valeurs d'affinités se pose. C'est pourquoi nous proposons également une notion d'affinité relative.

Affinité relative entre ressources

De même que pour le coût des ressources et pour les mêmes raisons, nous définissons la notion d'affinité relative entre ressources.

Définition III.5 *Affinité relative entre deux ressources r_i et r_j*

$$\begin{array}{rccc} aff' : & R \times R & \rightarrow & [0,1] \\ & (r_i, r_j) & \mapsto & \dfrac{aff(r_i, r_j)}{\displaystyle\sum_{r_k \in R} \sum_{r_l \in R} aff(r_k, r_l)} \end{array}$$

De plus, l'affinité entre les ressources doit être déclinée en une troisième notion qui est l'affinité potentielle entre une ressource et un hôte.

Affinité potentielle entre ressources et hôtes

La notion d'affinité potentielle permet d'évaluer l'importance des interactions entre une ressource et un hôte, compte-tenu des ressources qui doivent y être déployées. Si l'affinité est la fréquence des requêtes échangées entre les ressources, l'affinité potentielle est la somme des fréquences relatives potentielles de requêtes échangées entre une ressource et toutes les ressources recommandées à l'hébergement sur un hôte.

Définition III.6 *Affinité potentielle entre ressources r et hôte h*

$$\begin{array}{rccc} aff'' : & R \times \mathcal{H} & \rightarrow & [0,1] \\ & (r,h) & \mapsto & \sum_{r_i \in depl(h)} aff'(r, r_i) \end{array}$$

compte-tenu que la somme des affinités relatives est égale à 1 et qu'une même ressource ne sera pas déployée deux fois sur le même hôte, on a l'assurance que l'affinité potentielle ne dépassera jamais 1.

11.2.3 Modulation des distances

Ainsi, nous sommes équipés de deux quantités, charge et affinité potentielles, qui doivent être prises en compte dans le placement des ressources à déployer.

Nous pouvons constater que dans le problème des k-médians, les solutions sont composées des sommets destinations des arcs aux labels les plus bas. Ainsi, l'augmentation des labels défavorise la destination dans le processus de sélection des solutions. La Figure 11.2 montre l'impact de l'augmentation (d'un facteur 5) des labels des arcs dont la destination est $S2$: $critere(\{S2\}) = 5 + 15 = 20 > critere(\{S1\}) = 4 + 2 = 6$. Ainsi, cette augmentation rend la solution $S1$ plus avantageuse que $S2$. C'est pourquoi nous augmentons les labels des arcs en fonction de la charge potentielle de leur destination.

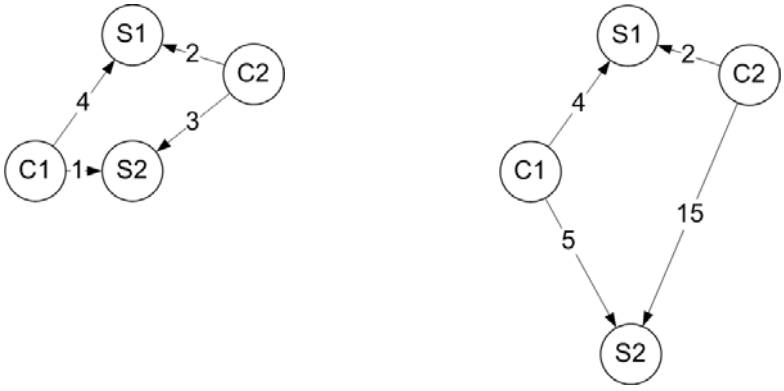


FIG. 11.2 – Exemple de graphe d'un problème de déploiement avant et après augmentation des labels des arcs vers $S2$

De plus, les sommets qui sont la source des arcs aux labels les plus importants représentent les clients stratégiques, ceux qui influent le plus sur la qualité des solutions. Ainsi, les meilleures solutions dépendent directement de l'emplacement de ces sommets. La Figure 11.3 montre l'impact de l'augmentation (d'un facteur 5) des labels des arcs dont la source est $C2$: $critere(\{S2\}) = 1+15 = 16 > critere(\{S1\}) = 4 + 10 = 14$. Cette modification augmente le poids de $C2$ dans les critères de solution, rendant ainsi $S1$ plus avantageuse que $S2$. C'est pourquoi nous augmentons les labels des arcs dont la source présente une importante affinité potentielle avec la ressource sujette au déploiement.

Nous pouvons donc, pour une ressource donnée, moduler les distances d'un $\mathcal{CNP}$-Graphe en fonction de la charge potentielle des destinations et de l'affinité potentielle des sources avec cette

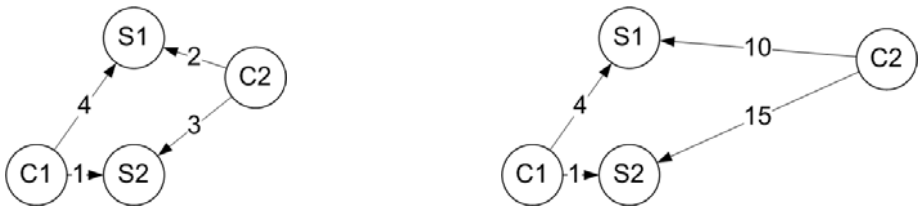


FIG. 11.3 – Exemple de graphe d'un problème de déploiement avant et après augmentation des labels des arcs issus de $C2$

ressource. Nous appelons cette notion *distance modulée*

Définition III.7 *Distance modulée entre l'hôte h_i et hôte h_j pour la ressource r*

$$\begin{array}{rccc} d' : & \mathcal{H} \times \mathcal{H} & \rightarrow & \mathbb{R} \\ & (hi, hj) & \mapsto & \alpha(d(hi, hj), load(hj), aff''(r, hi)) \end{array}$$

où $\alpha : \mathbb{R} \times [0,1] \times [0,1] \rightarrow \mathbb{R}$ est une fonction servant à régler l'influence des charges potentielles et des affinité potentielles par rapport aux contraintes utilisateurs. Ce réglage permet à l'utilisateur d'adapter le comportement de l'algorithme $MRKM$. Par exemple, afin de favoriser l'équilibrage des charges, on prévoira une grande influence des charges potentielle au détriment de celle des affinités, alors que pour favoriser les performances, on prévoira le contraire.

11.2.4 Algorithme MRKM

L'Algorithme 2 présente le fonctionnement de $MRKM$. Il est construit sur une logique gloutonne, à la manière dont on résout le problème des sacs à dos par approximation : une liste de ressources à déployer est fournie par l'utilisateur, l'algorithme pioche une ressource, exécute l'algorithme trivial des k-médians sur son $\mathcal{CNP}$-Graphe modulé, met à jour les charges potentielles et affinités potentielles en fonction des recommandations obtenues, puis pioche la ressource suivante et ainsi de suite jusqu'à ce qu'il n'y ait plus de ressource.

Comme dans tout algorithme glouton, l'ordre dans lequel les ressources sont traitées influe sur le résultat final. Il peut donc être judicieux de trier la liste des ressources avant de la soumettre à l'algorithme $MRKM$. Les deux stratégies évidentes sont : (1) de trier cette liste par ordre décroissant de coût, les ressources les plus coûteuses seront alors avantagées, ce qui favorise l'équilibrage des charges ; et (2) de la trier par ordre décroissant de la somme des affinités, les ressources provocant le plus d'interactions seront alors avantagées, ce qui favorisera les performances de la superstructure.

11.3 Expérimentation

Afin de vérifier l'efficacité de l'algorithme MRKM, nous avons décidé de comparer qualitativement son résultat avec celui obtenu par l'utilisation successive de l'algorithme trivial des k-médians pour chacune des ressources à déployer. En effet, les résultats de l'utilisation des k-médians qui ont été

Algorithme 2 MRKM(nR, Vs, Vd, dS, kS, $cost$, aff) : $recom$

Entrées :

n : Entier, nombre de ressources à déployer
Vs : ensemble de nœuds, origines des arcs, clients
Vd : ensemble de nœuds, destinations des arcs, candidats à l'hébergement
dS : vecteur de n matrices de $|Vs \times Vd|$ Réels, distances labels pour chacune des ressources
kS : vecteur de n entiers, les nombres d'instances des ressources devant être déployées
$cost$: vecteur de n réels, coûts d'utilisation des ressources
aff : matrice de $|n \times n|$ réels, affinités entre les ressources

Sorties : $recom$: ensemble d'ensembles d'Entier, id. de nœuds, les recommandations de placement des n ressources

Variables Locales :

h, hs, hd : Entier, id. de nœuds
r, ri, rj : Entier, id. de ressources
$cost'$: vecteur de n réels, coûts relatifs des ressources aff' : matrice de $n \times n$ réels, affinité relative des ressources aff'' : matrice de $n \times |Vd|$ réels, affinité potentielle entre ressources et hôtes

//Initialisations
pour tout $ri = 1$ à n **faire**
  $cost'(ri) = cost(ri) / \sum_{rj \in R} cost(rj)$
  pour tout $rj = 1$ à n **faire**
    $aff'(ri, rj) = aff(ri, rj) / \sum_{rx=1}^{n} \sum_{ry=1}^{n} aff(rx, ry)$
  fin pour
  pour tout $hj = 1$ à $|Vd|$ **faire**
    $aff''(rj, hj) = 0$
    $load(hj) = 0$
  fin pour
fin pour
// Pour toutes les ressources
pour tout $r = 1$ à n **faire**
// Moduler les distances
  pour tout $hi = 1$ à $|Vs|$ **faire**
    pour tout $hj = 1$ à $|Vd|$ **faire**
      $d'(hi, hj) = \alpha(dS(r)(hi, hj), load(hj), aff''(r, hi))$
    fin pour
  fin pour
// Exécuter k-médians
  $recom(r) = kmedian(k(r), Vs, Vd, d')$
// Mettre à jour charges et affinités potentielles
  pour tout $h = 1$ à $k(r)$ **faire**
    $load(recom(r)(h)) = load(recom(r)(h)) + cost'(r)$
    pour tout $ri = 1$ à n **faire**
      $aff''(ri, h) \leftarrow aff''(ri, h) + aff'(ri, r)$
    fin pour
  fin pour

fin pour
Retourner $recom$ // Retourner les recommandations de placement

montrés Section 8.3 permettent de penser que les performances expérimentales seront satisfaisantes, sinon parfaites.

Les ressources que nous avons utilisées sont celles de la superstructure GGM et que nous avons utilisées lors des expérimentation précédentes Chapitre 8 : (1) le service de caches collaboratifs de type communications intensives, (2) le service de fouille de donnée de type calculs intensifs et (3) l'entrepôt distribué de type versatile.

Les matrices contenant le nombre d'instances de chacune des ressources, leurs affinités et leur coûts sont montrées Table 11.1.

Le nombre d'instances a été fixé arbitrairement à 2 pour le service de cache, 3 pour l'entrepôt et 1 pour le service de fouille.

Le déroulement d'une utilisation typique de l'architecture GGM est le suivant : un utilisateur navigue dans l'entrepôt distribué, lequel interagit fréquemment avec le service de caches collaboratifs pour calculer les agrégats demandés. Une fois l'utilisateur satisfait par l'agrégat obtenu, les informations à fouiller sont extraites de cet agrégat et envoyées au service de fouille. Il y a donc deux cas d'interactions : l'entrepôt avec le cache, qui a été jugée 3 fois plus importante que celle de l'entrepôt avec le service de fouille.

Le coût des ressources a été calculé comme la moyenne des labels des arcs de leur $\mathcal{CNP}$-Graphe respectif, multiplié par la somme de leurs affinités. Ainsi, le coût des ressources reflète leurs coûts de transfert et de calcul, ainsi que la fréquence de leurs utilisations. Ainsi, en triant la liste des ressources en fonction de leur coût, nous prenons tous les aspects en compte.

$$Cost(r) = \frac{\sum_{(hs,hd)\in Vs\times Vd} d(hs,hd)}{card(Vs \times Vd)} \times \sum_{ri\in R} Aff(ri,r)$$

Tab. 11.1 – Matrice des k, affinités et coûts des ressources

$$k = \begin{pmatrix}2\\3\\1\end{pmatrix} \quad Aff = \begin{pmatrix}0.00 & 3.00 & 1.00\\3.00 & 0.00 & 0.00\\1.00 & 0.00 & 0.00\end{pmatrix} \quad Cost = \begin{pmatrix}261.61\\159.76\\58.81\end{pmatrix}$$

Enfin, la fonction de réglage que nous avons utilisée est volontairement exagérée afin que l'on puisse pleinement observer l'influence des charges et affinités dans les résultats :

$$\alpha(d,l,a) = d\ (1 + 100 \times l)\ (1 + 100 \times a)$$

Avec des valeurs plus modérées, nous obtenons moins de changements, mais de meilleures performances expérimentales. En effet, il est très difficile, pour des raisons techniques, d'évaluer les temps d'exécution sur des machines saturées (en réalité, rien uqe la recherche du point de saturation est déjà un problème à part entière). Notre évaluation est donc faites sur des machines disponibles, les performances obtenues ne sont donc pas pertinentes pour la validation d'$MRKM$ et donc pas présentées.

Les résultats obtenus avec la simple utilisation de l'algorithme trivial des k-médians et l'algorithme MRKM sont montrés Table 11.2. On peut formuler une observation pour chacune des ressources :

- les recommandations de placement pour la ressource 1 correspondent à celle obtenues par la résolution d'un k-médian classique, car aucune recommandation n'a encore été calculée, les décisions ne sont donc pas encore influencées par le placement d'autres ressources.
- les recommandations de placement pour la ressource 2 sont clairement influencées par celles de la ressource 1 : les hôtes de `toulouse` et `sophia` ont été remplacés par des hôtes de `lyon` et `lille`, se rapprochant des instances de la ressource 1 sous l'influence de l'affinité potentielle, tout en évitant de se mettre sur les mêmes hôtes grâce à l'influence de la charge potentielle.
- les recommandations de placement pour la ressource 3 sont clairement influencées par celles des ressources 1 et 2 : l'hôte de `lille`, déjà sélectionné pour héberger la ressource 1, a été remplacé par un hôte de `sophia`. L'aspect important ici est que ni `lille` ni `lyon` n'ont été sélectionnés malgré l'affinité entre les ressources 1 et 3 car cette dernière est de type calcul intensif. Le coût des communications et, par extension, la topologie de l'infrastructure importent donc peu dans son placement. On peut donc se permettre de « l'éloigner » pour l'héberger sur un hôte disposant d'une grande capacité de calcul. On peut remarquer sur cette figure que les hôtes les plus efficaces pour héberger ce type de ressources sont, dans l'ordre, ceux de `lille`, `lyon` puis `sophia`. Or les deux hôtes de `lyon` et les deux de `lille` sont déjà chargés des ressources 1 et 2. `sophia` est donc sélectionné.

Tab. 11.2 – Résultats de l'algorithme MRKM

	Ressource 1	Ressource 2	Ressource 3
k-médians	`sagittaire-49.lyon`, `node-62.lille`,	`node-42.toulouse`, `node-65.sophia`, `parasol48.rennes`,	`node-62.lille`,
MRKM	`sagittaire-49.lyon`, `node-62.lille`,	`parasol48.rennes`, `sagittaire-11.lyon`, `node-11.lille`,	`helios48.sophia`,

11.4 Discussion

La première remarque que nous pouvons formuler concerne la complexité de l'algorithme $MRKM$. En effet, cette dernière est élevée puisque $MRKM$ résout le problème des k-médians pour chacune des ressources, lequel est NP-difficile. En réalité, cette complexité ne représente pas réellement un problème, car les ressources sont généralement déployées pour une utilisation à long terme. Ainsi, le temps de décision impliqué par l'utilisation d'un algorithme centralisé, même complexe, ne doit pas être considéré comme une surcharge au fonctionnement des ressources. De plus, un placement judicieux des ressources est généralement extrêmement profitable puisqu'il permet d'équilibrer les charges et/ou d'optimiser les performances sur un long terme.

On peut également remarquer que l'algorithme $MRKM$ a été présenté dans le cadre d'un déploiement à partir d'une superstructure vierge. En fait, l'existence de ressources déjà déployées est parfaitement envisageable. D'abord, ces ressources influent les performances mesurées sur l'infrastructure, et donc les $\mathcal{CNP}$-Graphes. Elles sont donc déjà comprises d'une certaine façon en tant que charge. L'affinité en revanche nécessite qu'on connaisse l'emplacement de ces ressources afin d'initialiser correctement l'affinité potentielle entre ressources et hôtes. De plus, $MRKM$ a été présenté comme une

extension des k-médians, mais il est tout a fait envisageable de l'utiliser pour d'autres problèmes de placement : il suffit de remplacer dans l'algorithme de $MRKM$ la résolution des k-médians par la résolution d'un autre problème, tel que présenté Section 6.1, pour obtenir un comportement semblable avec un objectif différents.

De plus, l'algorithme $MRKM$ nécessite de nombreuses données en entrée : nombre d'instances, coût et affinité des ressources. Or, ces données ne sont pas nécessairement disponibles et peuvent être complexes à récupérer. Cet aspect rejoint la discussion sur la définitions des valeur TPS, Section 10.3, et amène aux même remarques. Même si notre calcul des affinités et charges potentielles confère une certaine tolérance dans l'imprécision des entrées, il reste que leur acquisition reste un point sensible de notre approche. Il faut par ailleurs noter que les définitions que nous avons adoptées pour les coûts et affinités ne sont pas strictes. Si besoin en est, elles peuvent être raffinées. Par exemple, si les ressources à déployer sont inactives (c.-à-d. qu'elles n'envoient pas de requêtes), type données, on peut définir l'affinité comme le nombre d'utilisations communes de ces ressources.

Enfin, il est important de noter que notre algorithme ne prend pas en compte la capacité des hôtes à héberger une certaine charge de ressource. En réalité, cela ne limite pas la précision de ses résultats, puisque l'équilibrage des charges fait partie de ses fonctionnalités. Ainsi, on a l'assurance que les ressources seront équitablement réparties entre les hôtes en fonction de leurs performances. En revanche, si on cherche à déployer trop de ressources pour la plateforme, notre algorithme est incapable de le détecter et utiliser ses recommandations mènera à une surcharge, qui néanmoins sera uniforme sur l'ensemble des hôtes de la plateforme.

11.5 Conclusion

Dans ce chapitre, nous avons présenté l'algorithme $MRKM$ - *Multi Resources k-Medians*. Ce dernier est conçu pour calculer des plans de déploiement pour plusieurs ressources vouées à interagir les unes avec les autres. L'approche que nous avons adoptée consiste à réutiliser l'algorithme trivial de résolution du problème des k-médians en modulant les distances par rapport à deux quantités que nous avons définies : charge potentielle et affinité potentielle. Cette modulation permet non seulement d'équilibrer les charges en évitant de sélectionner systématiquement les hôtes les plus performants dans les plans de déploiement, mais aussi de prendre en compte le niveau d'interaction entre les ressources afin de minimiser les coûts de communication lorsque c'est nécessaire. De plus, la clarté de l'algorithme $MRKM$ et l'utilisation d'une fonction réglage confère à notre approche une grande souplesse d'utilisation qui lui permet d'être adapté aux principaux besoins utilisateurs. Des expériences menées sur Grid 5000 avec les ressources de la superstructure GGM ont montré le bon comportement de l'algorithme $MRKM$ et la pertinence des solutions obtenues.

12

Placement dynamique : FReDi

Les environnements hautement dynamiques comme les grilles pervasives présentent des caractéristiques particulières. La dynamicité concerne non seulement l'infrastructure matérielle avec ses performances et sa topologie, mais également les ressources dont l'utilisation peut varier fortement d'un instant à l'autre. Ces dernières sont également caractérisées par une hétérogénéité accrue : leur durée de vie peut être longue, mais également extrêmement courte ; leurs utilisateurs changent fréquemment et peuvent être mobiles ; certaines peuvent nécessiter un très grand niveau de contrôle, d'autres sont libres d'accès et de distribution. De plus, la grande dynamicité de l'infrastructure matérielle peut rendre inadaptées les solutions centralisées, aussi bien en ce qui concerne l'algorithme de décisions de placement que les outils de surveillance mis en place.

Face à cette grande dynamicité et à cette hétérogénéité accrue, les solutions de placement présentées précédemment ne sont pas adaptées, car elles sont prévues pour un cadre statique et les prises de décisions sont centralisées. En effet, même si on peut imaginer exécuter l'algorithme de calcul des recommandations régulièrement afin de garder un bon niveau d'adaptation du placement aux besoins, ces recommandations risquent fort de d'avérer sub-optimales car la dynamicité ne sera pas réellement prise en compte.

C'est pourquoi nous nous sommes intéressés au placement dynamique de ressources. Il faut ici regarder du coté des solutions en ligne (*on line* en anglais) qui ne considèrent pas toutes les requêtes connues, mais propose plutôt de réagir en fonction uniquement de chaque requête au moment de son émission.

12.1 Positionnement

Dans la suite des explications de cet algorithme, nous nous limiterons à considérer des ressources de type données sous forme de fichiers. Ces fichiers sont potentiellement très nombreux et caractérisés par de grandes dynamicité et hétérogénéité. Ainsi leur placement se doit d'être adapté et dynamique, ce qui se traduit par des décisions différentes d'un fichier à l'autre et un processus de décision exécuté en continu. De plus, certains de ces fichiers, dits *sensibles*, sont caractérisés par le besoin d'un haut niveau de contrôle qui se traduit par la limitation de leur réplication afin de pouvoir les surveiller en temps réel. C'est le cas, par exemple, de contenus multimédias commerciaux : leurs détenteurs ne souhaitent pas qu'ils soient répliqués à tout va, car cela augmente le risque de piratage ; ou encore le cas de données sujettes à des mécanismes passant mal à l'échelle, tel qu'une maintenance fréquente de la consistance, ou une surveillance critique des accès. Enfin, l'utilisation de ces fichiers est caractérisée par une forte imprédictabilité, rendant difficile le développement d'heuristiques générique et obligeant à adopter des méthodes fortement réactives. Entre autres, l'approche basée sur les k-médians est inadéquate car elle considère un placement statiques, alors qu'un placement dynamique serait plus pertinent.

De plus, ces fichiers sont stockés sur le réseau de caches collaboratifs décrit Section 1.2.1. Nous considérons que ceux-ci font office de proxy pour les clients mobiles ou non : à chaque instant, chacun d'entre eux est connecté à proxy-cache chargé de relayer ses requêtes. De plus, ces proxy-caches présentent des fonctionnalités utiles :

- ils sont organisés en un réseau pair-à-pair connexe : chaque proxy-cache maintient une liste de proxy-caches voisins et peut atteindre tout autre proxy-cache par propagation de proche en proche.
- ils permettent d'annoter les réplicas des fichiers stockés avec des méta-données éditables.
- ils assurent l'indexation des fichiers présents dans le réseau de caches : chaque proxy-cache permet de savoir combien de réplicas d'un fichier sont présents et leur localisation.
- ils surveillent l'utilisation des fichiers en interceptant et enregistrant chaque demande et en proposant un système d'abonnement aux notifications des évènements de migration, duplication et suppression de fichiers donnés.

Enfin, nous considérons que nous nous trouvons dans le cadre des petits mondes (*small worlds*, en anglais) : la topologie de l'infrastructure matérielle, la localisation des clients et l'utilisation des ressources n'est pas aléatoire. En fait, les utilisateurs présentant les mêmes centres d'intérêt sont regroupés en petits mondes qui correspondent à des grappes physiques hautement connectées. Chacune de ces grappes est en revanche faiblement connectée aux autres. Cette hypothèse est réaliste et ouvre ne nombreuses opportunités algorithmiques, comme l'a montré Kleinberg dans [Kleinberg 00].

12.2 Avoir ou ne pas avoir un algorithme de placement, telle est la question

Il faut ici s'interroger sur la pertinence d'utiliser un algorithme de réplication des ressources. En effet, dans [Karlsson 02], Karlsson présente les résultats d'une étude comparative de nombreux algorithmes de placement de réplicas sur des traces réelles de serveurs web. Sa conclusion est que

dans tous les cas étudiés, la mise en place d'un algorithme de placement n'était pas justifiée, car les performances s'avéraient identiques, sinon plus faibles, que celles obtenues avec la plus simple des stratégies de cache, *Least Recently Used*. Or dans le cas particulier que nous avons présenté, les techniques de caches ne peuvent s'appliquer de par la sensibilité de certaines ressources qui doivent nécessiter un certain niveau de contrôle. Un nombre maximum de réplicas doit donc être fixable, et dans ce cas, un algorithme le positionnement de ces réplicas prend toute son importance.

12.3 Identification du problème

En réalité le problème de placement présenté ici se rapproche plus de celui des k-serveurs, qui se place dans un cadre dynamique en ligne, que de celui des k-médians, qui se place dans un cadre statique hors ligne.

Définition III.8 *Problème des k-serveurs*

Soit k serveurs mobiles sur un arbre composé de N nœuds, des requêtes apparaissent successivement sur cet arbre. Une requête est servie lorsqu'un des serveurs l'atteint en se déplaçant jusqu'à elle. Le coût de ce problème est représenté par la distance totale parcourue par l'ensemble des serveurs pour servir l'ensemble des requêtes.

Ce problème NP-difficile est traité par une solution d'approximation en ligne présentée par Borodin et El-Yaniv dans [Borodin 98]. Cet algorithme appelé arbre de double couverture - DC-Tree (*Double Coverture - Tree*) est décrit comme suit :

Définition III.9 *Algorithme DC-Tree*

Initialement, k serveurs sont placés aléatoirement sur l'arbre. Puis, à chaque instant, tous les serveurs dans le voisinage de la requête se déplacent vers elle à vitesse constante.

Le voisinage d'une requête est défini par les premiers serveurs rencontrés en parcourant l'arbre depuis la requête.

La Figure 12.1 montre le comportement de DC-Tree sur un exemple de pilotage de trois serveurs S1, S2 et S3, devant répondre à une requête. La difficulté principale est de détecter les serveurs dans le voisinage de la requête. Initialement, S1 est hors du voisinage puisqu'il se trouve « derrière » S2 sur le chemin depuis la requête. S1 ne se déplace donc pas. S2 et S3 sont dans le voisinage et se déplacent alors à vitesse constante vers la requête. Lorsque ils se sont déplacés de la distance 1, S3 passe « devant » S2. S2 est alors exclu du voisinage et cesse de se déplacer. S3 est le dernier serveur dans le voisinage de la requête, il achève donc son déplacement et répond à la requête.

Cette méthode est (N-1)k-compétitive, ce qui signifie que son comportement dans le pire des cas s'avère être (N-1)k plus lourd que la solution optimale (qui peut être calculée en connaissant

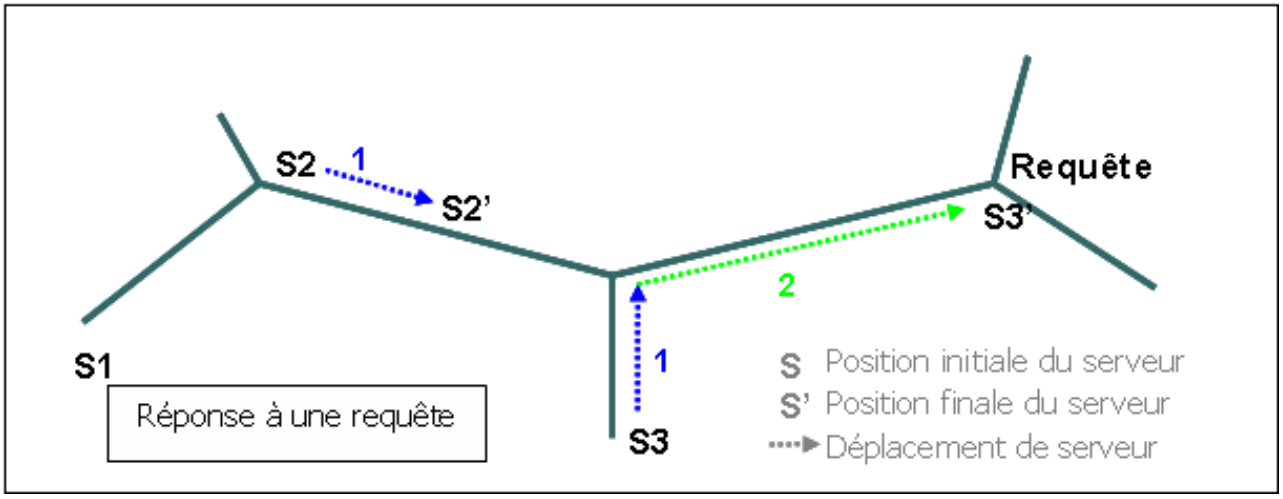


FIG. 12.1 – Illustration du pilotage de serveurs par DC-Tree

la séquence des requêtes à l'avance). Heureusement, selon les auteurs cette méthode s'avère bien meilleure dans le cas général.

En fait, en s'appuyant sur la théorie des petits mondes, on espère des résultats effectivement très compétitifs en situation réelle, puisque la séquence des requêtes n'est pas aléatoire, mais conditionnée par les utilisateurs, et présente ainsi des propriétés de « concentration » des demandes dans différents foyers d'attraction.

De plus, les auteurs affirment que cet algorithme fonctionne également sur un graphe quelconque, et pas seulement sur un arbre.

En revanche, cet algorithme nécessite d'être adapté aux environnements distribués : même si le concept de distance présenté dans la partie précédente permet de créer un graphe représentatif du problème et ce de façon adaptée à chaque ressource, le concept de « vitesse » reste à être adapté et afin de respecter les contraintes du cadre, la mise en œuvre de l'algorithme doit être distribuée.

12.4 Reformulation adaptée au placement de réplicas

Alors que le nombre de serveurs k est fixe dans le problème original, dans le contexte présenté ici le nombre de réplicas doit être dynamique afin de suivre les variations des besoins d'utilisation. Une autre différence entre le comportement d'un réplica et celui d'un serveur dans le problème initial est qu'un réplica ne doit pas nécessairement se déplacer sur un proxy-cache attitré du client demandeur. Il peut en effet être accédé par le client final sur un autre proxy-cache. On peut formaliser ce problème comme suit :

Définition III.10 *Problème des k-serveurs adapté au placement de réplicas*

Soit un maximum de kmax réplicas d'un même fichier sur un réseau en arbre de N proxy-caches, des clients émettent des requêtes relayées par leur proxy-cache le plus proche. Le coût de ce problème

est représenté par la somme sur l'ensemble des requêtes, des distances entre le proxy-cache du client émetteur et le réplica le plus proche.

Ainsi, nous voulons positionner k réplicas au cours de leur utilisation pour optimiser dans le temps la distance globale entre ces réplicas et l'ensemble des clients émetteurs de requête, tout en gardant k inférieur à $kmax$ et le plus bas possible afin de limiter l'utilisation des disques et de permettre le contrôle des réplicas sensibles.

De plus, dans l'algorithme original, les réplicas sont censés pouvoir se déplacer de façon continue sur les arêtes des graphes. Or dans le cas des $\mathcal{CNP}$-Graphes, les arêtes sont des liens de communication, ce qui rend ce déplacement impossible. En revanche, on peut définir une notion de position virtuelle sur cette arête : plutôt que de déplacer physiquement les réplicas, on définit pour chacun d'eux un vecteur contenant la position qu'il devrait avoir selon l'algorithme original. De plus, cette position virtuelle n'est pas nécessairement unique. Ainsi, l'algorithme DC-tree peut être adapté comme suit :

Définition III.11 *Algorithme DC-Tree adapté au placement de réplicas Lors de leur déploiement initial, un réplica initial est placé sur un des proxy-caches, a priori celui attitré à sa source.*

À l'apparition d'une requête, tous les réplicas dans son voisinage vont se déplacer virtuellement à vitesse constante dans sa direction. Lorsqu'un réplica possède plusieurs positions virtuelles différentes, il est dupliqué en plusieurs réplicas correspondant à ces positions. Lorsqu'un réplica n'est plus utilisé pendant une durée déterminée, il est supprimé.

12.5 Adaptation à un environnement distribué

L'algorithme original fonctionne dans un environnement entièrement centralisé. La topologie complète du réseau (distances entre nœud, positions des réplicas et requêtes„ ...) est connue et utilisée à chaque prise de décision. De plus, cette topologie est fixe. Dans le cadre d'un environnement dynamique et compte-tenu de la fréquence et du nombre des décisions, il semble impossible de centraliser l'algorithme de décision au risque d'une surcharge et d'un goulot d'étranglement sur l'hôte l'hébergeant.

De plus, on pourrait imaginer garder un algorithme de décision centralisé, mais répartir la charge des décisions sur l'ensemble des proxy-caches en leur attitrant à chacun la gestion d'un nombre équivalent de fichiers. Mais cette approche est rendue inefficace du fait de la dynamicité et de l'imprédictabilité de l'utilisation des fichiers, qui sont largement accrues dans les environnements pervasif comparés aux environnements classiques : de façon aléatoire, un proxy-cache pourrait subitement se retrouver en charge d'aucun fichier, ou pire en charge de fichiers très éloignés. Ce qui pourrait être résolu par la mise en place d'un mécanisme supplémentaire d'équilibrage régulier des charges de décision, au prix d'une charge supplémentaire, pas nécessairement pertinente.

En outre, cela ne résoudrait pas le problème de la connaissance globale des performances et de la topologie de l'infrastructure matérielle qui peut s'avérer très coûteuse dans un environnement dynamique.

L'environnement de décision est donc limité à l'échelle très restreinte d'un proxy-cache, dans une approche pair-à-pair. Chaque proxy-cache détient seulement une connaissance locale du graphe d'interconnexion. Il convient donc de prendre les bonnes décisions, en terme de placement de réplica, en se basant sur ces données partielles, dans un cadre de fonctionnement pair à pair afin de garder une architecture souple, adaptative, résistante aux pannes et passant à l'échelle. Nous avons ainsi adopté une approche de réplicas qui prendront sur leur lieu d'hébergement, et de façon complètement autonome, les décisions de migration, duplication et suicide (suppression) afin d'optimiser globalement le coût reflété par leur distance associée.

12.5.1 Méta-données

Le système de gestion des méta-données des fichiers implémenté dans le service de caches collaboratifs s'avère très utile. En plus des méta-donnée spécifiques à chaque fichier (propriétaire, description, etc.), nous l'utilisons pour stocker et maintenir à jour les données spécifiques à notre algorithme :

- $kmax$ le nombre maximum de réplicas autorisés.
- k le nombre actuel global de réplicas dans le service de cache.
- df la fonction distance à utiliser pour ce fichier, telle que définie dans Chapitre 5
- TTS *Time To Suicide*, le durée d'inutilisation d'un réplica avant sa suppression
- $D2A$ *Distance To Attraction*, le rapport distance/attraction ajoutée à chaque requête
- AD *Attraction Degradation*, la quantité dont l'attraction est réduite par unité de temps
- TTL *Time To Live*, la durée restante de temps avant suppression
- DTS *Distance To Self*, la distance considérée d'un proxy-cache vers lui-même.
- AV *Attraction Vector*, le vecteur d'attraction représentant la position virtuelle.

Chacune de ces définitions sera expliquée et positionnée dans leur contexte d'utilisation dans la suite.

12.5.2 Positions virtuelles

L'objectif est de permettre aux différents réplicas de se « déplacer à vitesse constante » sur le CNP-Graphe. Or, le concept de « vitesse constante », comprise dans l'algorithme comme une vitesse dans un repère euclidien, n'est pas du tout transposable au monde des réseaux. Elle n'a aucun rapport avec la vitesse de transfert ou de réponse. De plus ces déplacements étant continus, les réplicas sont censés pouvoir s'arrêter au beau milieu d'une liaison entre deux sites, ce qui une fois de plus est irréalisable dans le monde des réseaux.

Pour ce faire, nous transposons le concept de « déplacement à vitesse constante » vers le concept de « positions virtuelles » relatives aux distances représentées dans le CNP-Graphe, qui garde tout son sens sur un réseau. Nous avons développé deux mécanismes complémentaires pour assurer le placement des réplicas : la maintenance des positions virtuelles et leur concrétisation.

12.5.3 Les positions virtuelles - vecteur d'attraction

Chaque proxy-cache maintient une position virtuelle des réplicas qu'il gère, en fonction de sa connaissance de la topologie voisine. Cette position virtuelle est représentée par un vecteur contenant une valeur réelle, appelée *attraction*, pour chacun de ses voisins ainsi que pour lui-même. Initialement, ou après une concrétisation, le vecteur attraction est nul. Il va ensuite évoluer dans le temps selon les requêtes reçues. Une attraction vers un lien est augmentée lorsque le réplica est sollicité sur ce lien. La valeur de l'augmentation est la distance fournie lors de cette sollicitation, qui doit correspondre au déplacement prévu dans l'algorithme original, multiplié par un facteur paramétrique $D2A$ (*Distance To Attraction*).

Protocole NCasting inter proxy-cache

Nous avons élaboré un protocole de communication inter proxy-cache permettant de maintenir l'état des vecteurs d'attraction. Nous avons baptisé ce mécanisme le NCasting pour NeighbourCasting. L'algorithme 3 décrit son fonctionnement en pseudo-code.

Algorithme 3 Protocole NCasting

Exécuté sur le proxy-cache $localPC$ après réception d'un message $NCasting(R, dist)$ envoyé par le proxy-cache $senderPC$

Entrées :

R : identifiant de réplica

$dist$: Réel, la distance reçue

Sorties :

Mise à jour du vecteur d'attraction AV du réplica R

Variables Locales :

$localPC, closestPC$, pc : Identifiants de Proxy-Caches

$neighbourList$: liste de proxy-caches.

// Si R est stocké par le proxy-cache récepteur

si $localPC.store(R) = VRAI$ **alors**

$R.AV[senderPC] \leftarrow R.AV[senderPC] + R.D2A \times dist$ // M.à.j. du vecteur d'attraction local de R

// Sinon

sinon

$closestPC \leftarrow localPC.select(R)$ // Récupérer la position du plus proche réplica

$localPC.send(closestPC, NCasting(R, dist))$ // Lui envoyer un $NCasting$

// Puis pour tous les autres voisins

pour tout $pc \in localPC.getNeighbourList()$ **faire**

si $pc \neq closestPC$ ET $pc \neq senderPC$ **alors**

$localPC.send(pc, NCasting(R, d(localPC, closestPC)))$ // Envoyer $NCasting$

finsi

fin pour

finsi

Lors d'une demande initiale d'un fichier, son proxy-cache attitré $localPC$ redirige le client vers le réplica le plus proche stocké par $closestPC$ et initialise la mise à jour des vecteurs d'attraction par la réception locale d'un message $NCasting(R, d(localPC, closestPC))$ si $localPC$ et $closestPC$

sont différents. Si $localPC$ et $closestPC$ sont égaux, le proxy-cache possède déjà le réplica. Dans ce cas, il n'émet pas de $NCasting$ et met à jour le vecteur d'attraction local de R comme suit : $R.AV[localPC] \leftarrow R.AV[localPC] + R.D2A\ R.DTS$. Ainsi, les requêtes locales sont également prises en compte dans les vecteurs d'attraction.

Illustration du comportement du protocole $NCasting$

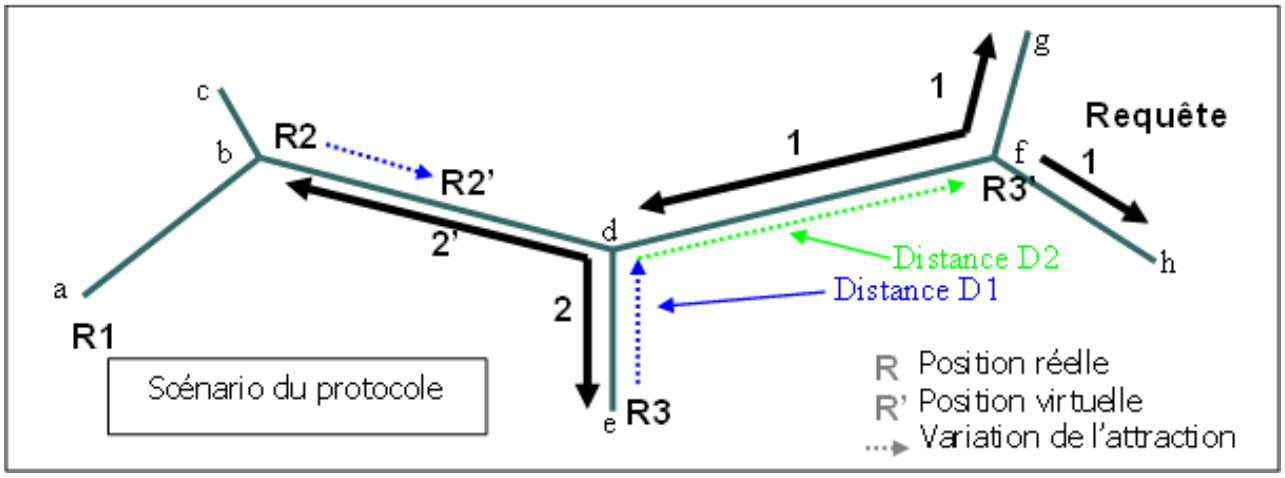


FIG. 12.2 – Illustration du protocole NCasting

La Figure 12.2 montre le comportement du protocole NCasting suite à une requête. Les détails sont les suivants :

1. initialement : le proxy-cache f demande un contenu R stocké sur a, b et e ; la distance au réplica $R3$ le plus proche est $d(f,e) = D1 + D2$; il déclenche donc une inondation de :
   (1) $NCasting(R, D1 + D2)$
2. d reçoit $NCasting(R, D1 + D2)$ et ne possède pas le réplica donc il renvoie :
   (2) $NCasting(R, D1 + D2)$ vers e, possédant $R3$, le réplica le plus proche.
   (2') $NCasting(R, D1)$ vers b, $D1 = d(d,e)$ étant la distance au réplica le plus proche
3. e reçoit $NCasting(R, D1 + D2)$ et possède $R3$:
   il ajoute $D1 + D2$ au vecteur d'attraction de $R3$ vers d
4. b reçoit $NCasting(R, D1)$ et possède $R2$:
   il ajoute $D1$ au vecteur d'attraction de $R2$ vers d

On peut donc voir qu'après l'exécution de ce protocole, les positions virtuelles des différents réplicas correspondent bien aux positions prévues dans l'algorithme original *DC-Tree*, (voir Figure 12.1) :

- la position virtuelle de $R3$, notée $R3'$, a atteint la position de la requête.
- la position virtuelle de $R2$, notée $R2'$, s'est déplacée tant qu'elle se situait dans le voisinage de la requête : après un déplacement de $D1$, $R3'$ atteint le proxy-cache d, sortant $R2'$ du voisinage.
- la position virtuelle de $R1$ n'est pas modifiée car $R1$ est situé derrière $R2$ et donc hors du voisinage de la requête depuis le début.

De plus, cette exemple considère une valeur *Distance To Attraction* $D2A = 1.0$, pour illustrer le comportement de l'algorithme, mais il clair que cette valeur permet de définir le nombre de requêtes nécessaires pour obtenir exactement le même comportement que l'algorithme original. Par exemple $D2A = 0.5$ permettra d'obtenir des positions virtuelles étant éloignées de moitié par rapport à celle de l'algorithme original. Ce paramètre permet donc de ne pas déplacer systématiquement les réplicas vers la position de chaque requête.

Complexité des communications

Soit un arbre de N nœuds. Le pire des cas est lorsque le voisinage de la requête équivaut à l'arbre complet. Une requête donnera alors lieu à autant de message qu'il y a d'arc dans l'arbre : $N - 1$ messages. Soit une complexité de messages en $O(N)$.

Cette complexité est la même que celle d'une inondation, donc très lourde. Elle peut néanmoins être significativement réduite en exploitant la connaissance de la position de tous les réplicas de la donnée et en limitant l'envoi de message vers ces destinations. De plus, une optimisation pour les fichiers les plus convoités est d'accumuler les distances reçues puis de n'envoyer qu'un seul $NCasting$ récapitulatif, afin d'économiser des messages.

Ce protocole peut également être mis en place sur un graphe quelconque. Deux méthodes sont envisageables. La première est d'ajouter un contrôle de boucle. La technique la plus simple consistant à estampiller les messages $NCasting$ et refuser ceux qui ont déjà été reçus. Cette technique entraine un léger surcoût puisque des messages $NCasting$ inutiles seront émis. La seconde méthode consiste à modéliser le réseau des proxy-caches collaboratifs sous forme d'un $\mathcal{CNP}$-Graphe. Ce dernier peut alors être utilisé pour extraire un arbre couvrant minimum servant à organiser le réseau pair à pair des proxy-caches. On peut également imaginer des mises à jour régulières de ce réseau pour coller à la dynamicité de l'infrastructure. Ces mises à jour n'ont par ailleurs pas besoin d'être extrêmement fréquentes puisque les caches sont exécutés sur l'infrastructure câblée qui est largement plus stable que l'infrastructure de communication sans-fil. De plus, organiser ce réseau permet d'optimiser les différentes communications internes au système de caches collaboratifs, et s'avère donc profitable au-delà de la seule utilisation de FReDi.

12.5.4 Opération de maintenance

Nous appelons *concrétisation* les actions de migration/duplication et suicide des réplicas en fonction de leur vecteur d'attraction. Les opérations de concrétisation seront déclenchées lors de la maintenance des vecteurs d'attraction. Cette opération de maintenance est régulièrement exécutée par chaque proxy-cache pour chacun de ses réplicas. Lors de cette maintenance, on réduit les valeurs selon un paramètre de dégradation temporelle AD(*Attraction Degradation*), qui permet de limiter l'impact des requêtes en fonction de leur ancienneté et on décidera des opérations de migration/duplication/suicide.

En réalité, le vecteur d'attraction représente la popularité directionnelle du réplica. Ainsi lorsque

deux valeurs deviennent très importantes, cela signifie que le réplica peut être utile dans deux directions différentes. Dans les cas où la duplication est possible, le réplica se duplique sur les deux proxy-caches voisins indiqués par l'attraction. Sinon, le réplica ne se copie pas et tente de migrer pour trouver un compromis.

Le même mécanisme s'applique au suicide. Lorsque le vecteur d'attraction reste nul pendant une période paramétrable TTS (*Time To Suicide*), nous considérons que le réplica n'a plus aucune utilité locale et doit être supprimé.

L'Algorithme 4 décrit le fonctionnement de l'opération de maintenance.

Scénario d'illustration

La Table 12.1 détaille le comportement de réplicas d'une donnée pilotée par FReDi : elle illustre les différentes opérations de migration, duplication et suppression en fonction de l'état des vecteurs d'attraction.

12.6 Expérimentations

L'évaluation des performances de FReDi comprends deux métriques orthogonales :

- NRA : le nombre de réplicas autorisé, le but de FReDi étant de le garder le plus bas possible.
- *critere* : la moyenne arithmétique des distances entre requêtes et leur plus proche réplica.

De plus, nous comparons les performances de FReDi à trois stratégies classiques de placement :

- Serveur unique : pas de réplication, donc un contrôle total sur les réplicas mais un *critere a priori* élevé. Le choix du serveur unique étant crucial, nous avons sélectionné le meilleur emplacement compte-tenu de la positions de toutes les requêtes, ce qui en soit est déjà difficilement réalisable en environnement réel étant dépourvu de cette connaissance.
- Cache après la première requête : réplication systématique sans suppression.
- Pré-cache : réplication initiale sur tous les proxy-caches.

Les expérimentations menées sur les distances en environnement réel ayant déjà montré la pertinence de cette partie de notre approche, nous avons donc expérimenté FReDi sur un simulateur. Ceci nous a permis d'utiliser des traces très importantes sur une topologie très étendue.

Nous avons développé un simulateur du protocole de NCasting et du mécanisme de concrétisation. Ce dernier a été utilisé pour exploiter les traces d'utilisation de 20 contenus vidéos et audio hébergés par un serveur web privé depuis mai 2004 jusqu'a septembre 2005. Ce serveur diffuse des contenus multimédias créés par ses utilisateurs et téléchargés dans le monde entier : les contenus les plus demandés ont été téléchargés depuis plus d'une centaine de pays différents, les contenus les moins populaires se limitent à une vingtaine de pays. De plus, l'utilisation des contenus est très hétérogène : les plus sollicités atteignent plusieurs dizaines de requêtes par jour, les moins sollicités quelques requêtes par mois. Le modèle d'utilisation est également hétérogène : une très forte utilisation quotidienne en France, et

Algorithme 4 Maintenance des vecteurs d'attraction

Exécuté toutes les périodes p sur les proxy-caches pour chaque réplica R

Entrées :

p : Réel, la période d'exécution de la maintenance
$R.AV$: vecteur réel, contenant les attractions de R
$R.TTS$: Réel, le temps d'inutilisation de R avant sa suppression
$R.TTL$: Réel, le temps restant à R avant sa supression
$R.kmax$: Entier, le nombre maximum de réplicas
$R.k$: Entier, le nombre de réplicas actuel
$R.AD$: Réel, le facteur de dégradation des valeurs d'attraction

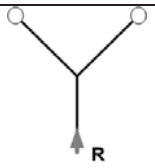
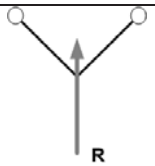
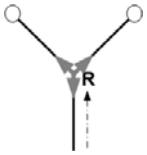
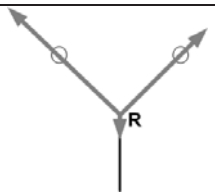
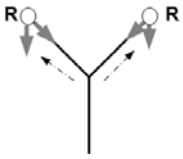
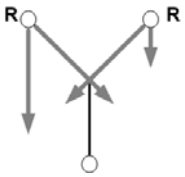
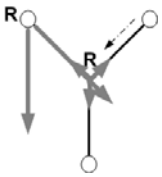
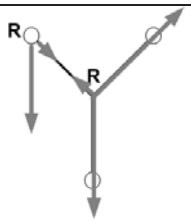
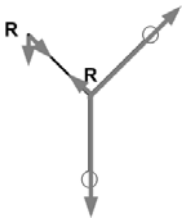
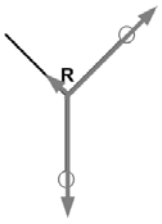
Sorties :

Le déclenchement des opérations de migration/duplication/suicide et la dégradation des attractions.

Variables Locales :

NDA : Entier, le nombre de duplications autorisée
L : liste des liens pour lesquels l'attraction est plus forte que la distance, triée par ordre décroissant d'attraction
i : Entier
$flagSuicide$: Booléen

```
                    //Déclenchement des opérations de duplication tant que possible, puis migration
NDA ← R.kmax − R.k
pour i = 1 à NDA faire
  si L[i] ≠ localPC alors
    localPC.copy(R, L[i])
    R.AV[L[i]] ← 0
  finsi
fin pour
si L[NDA + 1] ≠ localPC alors
  localPC.move(R, L[NDA + 1]
finsi
                                  //Application du facteur de dégradation sur les attractions
flagSuicide ← VRAI
pour tout v ∈ R.AV faire
  v ← v − p × R.AD
  si v ≠ 0 alors
    flagSuicide ← FAUX
  finsi
fin pour
                                                  //Gestion du temps qui reste à vivre
si flagSuicide = VRAI alors
  R.TTL ← R.TTL − p
sinon
  R.TTL ← R.TTS
finsi
si R.TTL = 0 alors
  localPC.delete(R)
finsi
```

	Avant maintenance	Après maintenance
Étape 0		
À l'étape initiale, il existe deux foyers d'utilisation massive et un seul réplica, son vecteur d'attractions est nul.		
Étape 1 : Migration vers foyers		
Au fils des NCastings, l'attraction grandit vers les foyers. R est donc migré vers la seule direction d'attraction.		
Étape 2 : Duplication vers foyers		
Puis les attractions grandissent vers les deux foyers. Le nombre maximal de réplicas est deux. Il n'est pas atteint. R est donc dupliqué vers les deux foyers.		
Étape 3 : Migration vers point d'équilibre		
Un troisième foyer apparait. Les attractions grandissent donc vers ce foyer. Les attractions locales grandissent également puisque les réplicas sont situés sur des foyers. Le nombre de réplicas maximum étant atteint, on ne peut dupliquer. L'attraction locale du réplica gauche l'empêche de migrer. En revanche, le réplica droite décide de migrer vers une position d'équilibre.		
Étape 4 : Dégradation		
Le réplica gauche est maintenant hors du voisinage des foyers gauche. Son attraction vers eux n'est donc plus augmentée. Le facteur de dégradation réduit alors cette valeur.		
Étape 5 : Supression		
Le foyer gauche disparait. Il est toujours hors du voisinage des deux foyers restants. Son attraction n'est donc plus augmentée et le facteur de dégradation réduit son vecteur d'attraction à zéro. Son temps à vivre atteint ensuite zéro et il est supprimé. Suite à quoi, le réplica restant pourra à nouveau se dupliquer vers les deux foyers.		

TAB. 12.1 – Scénario d'illustration du pilotage des réplicas par FReDi

des « vagues » d'utilisations dans les pays étrangers, probablement déclenchés par la découverte de certains contenus et leur diffusion au sein de communauté. Les fichiers ne sont d'ailleurs pas indexés par d'autres moyens que les moteurs de recherche classiques. Le modèle d'utilisation correspond donc bien à celui des petits mondes puisqu'il fonctionne essentiellement sur le bouche à oreille au sein de communautés.

La topologie utilisée est composée de 235 proxy-caches couvrant la surface de la Terre, à raison d'un proxy-cache par pays. Chaque requête des 20 contenus a été supposée relayée par le proxy-cache attitré au pays d'origine de la requête, extrait des traces grâce à la librairie GeoIP[1]. Nous avons utilisé la distance kilométrique pour créer la topologie du réseau, ce qui ne pose pas de problème, car l'objectif est d'évaluer les performances de FReDi, quelle que soit la distance utilisée, et car la pertinence des distances des $\mathcal{CNP}$-Graphes a été montrée Section 8. De plus, ces distances ont été normalisées dans l'intervalle $[0, 1]$

Le paramétrage de FReDi est le suivant :
- $DTS = 0.5$: la distance d'un proxy-cache à lui-même est la valeur médiane des distances entre tous les pays. Les requêtes locales ont donc une importance médiane.
- $DTA = 1.1$: le rapport distance - attraction est important : toutes les requêtes peuvent potentiellement déclencher une opération de migration ou duplication.
- $AD = 1$: en revanche, le facteur de dégradation des attractions permettra de rapidement « oublier » les requêtes passée, le comportement des réplicas est donc très réactif.
- $TTS = 1mois$: les réplicas sont conservés un mois avant suppression. Cela est pertinent pour nos traces car les fichiers peuvent être utilisé « sporadiquement », c.-à-d. à plusieurs jours d'intervalle au sein d'une même communauté.

Ensuite, nous avons mené des expériences avec le nombre maximum de réplicas $kmax$ variant dans l'intervalle $[0, 235]$. compte-tenu de la convergence rapide de nos résultats, nous ne les présentons que dans l'intervalle $[0, 60]$.

Finalement, nous avons comparé nos résultats à ceux des stratégies concurrentes. On peut remarquer que le *critere* de la stratégie « pré-cache » est 0 puisque nous ne pouvons pas placer les réplicas plus proches de leurs clients.

Comme on peut le constater sur la figure 12.3, avec seulement un unique réplica autorisé, nous obtenons déjà une amélioration des résultats par rapport à la stratégie « serveur unique ». La convergence rapide des résultats montre que le comportement de FReDi est très pertinent. De plus, nous obtenons des résultats quasiment optimaux (équivalents à la solution « pré-cache ») avec seulement 10 réplicas pour 235 proxy-caches.

La Table 12.2 montre les mêmes résultats normalisés : 100% représente la meilleure efficacité et 0% la pire. La qualité de service est représentée par la distance moyenne entre requêtes et leur réplica le plus proche divisé par la valeur obtenue par la stratégie « serveur unique ». L'indice de contrôle sur les réplicas est représenté par le nombre maximum de réplicas divisé par le nombre de proxy-caches émettant au moins une requête pour le réplica. Ainsi, la stratégie « serveur unique » présente une qualité de service minimale mais un contrôle optimum, alors que la stratégie « pré-cache » présente des résultats inverses.

[1] www.maxmind.com

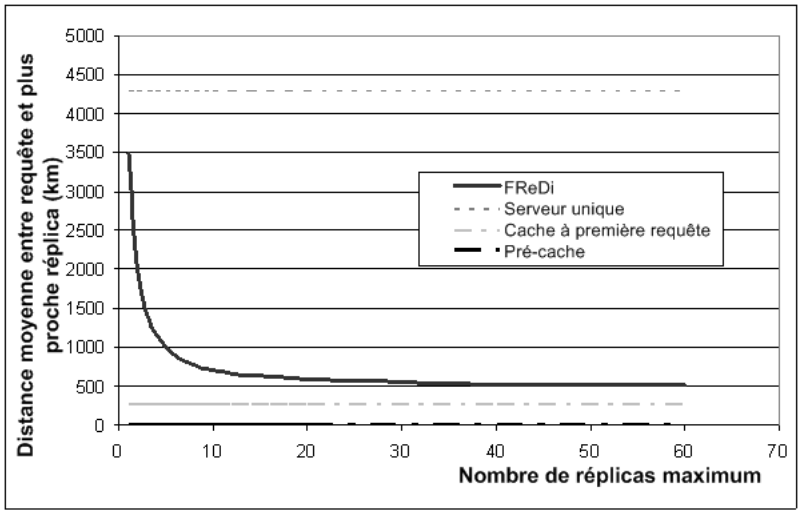


FIG. 12.3 – Distance moyenne entre les requêtes et leur plus proche réplica en fonction du nombre maximum de réplicas.

Strategie	qualité de service	indice de contrôle
Serveur unique	0%	100%
Cache après première requête	94%	0%
Pré-cache	100%	0%
FReDi avec 1 réplica	19%	100%
FReDi avec 10 réplicas	84%	74%
FReDi avec une infinité de réplicas	88%	0%

TAB. 12.2 – Résultats expérimentaux normalisés de la comparaison de FReDi avec les stratégies concurrentes

Les résultats importants sont : qu'avec 1 réplica, FReDi améliore de 20% la qualité de service, pour un indice de contrôle équivalent à celui du serveur unique ; qu'avec 10 réplicas, FReDi présente une qualité de service et un contrôle très satisfaisants (respectivement 84% et 74%), alors que la stratégie « cache après première requête » montre une qualité de service légèrement plus importante (94%), mais pour un contrôle extrêmement bas ; qu'une quantité infinie de réplicas n'est pas nécessaire puisque l'écart avec les résultats obtenus avec 10 réplicas est faible (88% contre 84%).

12.7 Discussion

L'algorithme distribué de placement FReDi présente plusieurs aspects devant être discutés.

Premièrement, FReDi est conçu pour piloter des réplicas de contenus sensibles utilisés dans un contexte de petits mondes. En effet, en cas d'utilisation anarchique, ou si les contenus peuvent être répliqués à tout va, il est clair que des stratégies de caches plus simples s'avèreront autant, sinon plus efficaces, comme exposé Section 12.2.

Deuxièmement, FReDi dispose d'un ensemble important de paramètres conditionnant son fonc-

tionnement. Cet ensemble confère à notre approche une grande flexibilité et lui permet d'être utilisable et efficace pour de nombreuses catégories d'utilisations et de contenus. En revanche, le réglage de ces différents paramètres afin d'obtenir un pilotage optimal des réplicas peut s'avérer être un processus délicat. En effet, des paramètres « classiques » peuvent être relativement aisément définis en se basant sur des informations de surveillance des utilisations des réplicas, telles que la fréquence des requêtes, et sur des informations sur la plateforme, telles que la taille du réseau de proxy-cache et le nombre de clients. En revanche, obtenir un pilotage absolument optimal pour tous les réplicas risque de s'avérer plus compliqué. Un système de paramétrage dynamique sera envisagé dans les travaux futurs afin de parer aux problèmes classiques de pilotage dynamique, tels que les effets de « va et vient » entre deux positions, et de décharger l'utilisateur de cette tâche fastidieuse.

Troisièmement, les expérimentations ont été menées dans le cadre de contenus multimédias. En réalité, comme nous l'avons montré Chapitre 5, les $\mathcal{CNP}$-Graphes peuvent être représentatifs d'un large choix de ressources. Ainsi, FReDi peut également être utilisé pour le pilotage dynamique de réplicas de ressources de différents types, par exemple des services web.

Enfin, FReDi est caractérisé par un pilotage complètement réactif : les décisions ne sont prises qu'en fonction des requêtes passées. Ceci peut s'avérer être une limitation très importante dans un contexte d'utilisateurs mobiles très dynamiques puisque les réplicas accuseront toujours un « retard » sur leurs utilisateurs. Cet aspect peut-être amélioré par l'utilisation de prédiction des requêtes : savoir quelles requêtes seront émises dans un futur plus ou moins proche permet de piloter les réplicas afin de les positionner sur les proxy-caches demandeurs avant même que ceux-ci aient émis une requête. Cet aspect fait l'objet du chapitre suivant.

12.8 Conclusion

Dans ce chapitre, nous avons présenté un algorithme de placement dynamique de réplicas de contenus sensibles dans un contexte de petits mondes. Les contenus sensibles sont caractérisés par un besoin accru de contrôle, pour des raisons de maintien de la consistance, de maîtrise de leur utilisation ou de sécurité. Tous ces aspects sont grandement facilités lorsque le nombre de réplicas est gardé faible. Notre approche est donc de piloter un faible nombre de réplicas afin de servir tous les clients au mieux. Ainsi, on permet de garder le contrôle nécessaire sur les réplicas, tout en assurant une qualité de service satisfaisante pour les clients.

Le concept principal de notre approche est celui des vecteurs d'attraction, qui indiquent pour chaque réplica les migrations qui les rapprocheraient de leurs clients. L'exploitation de ces vecteurs se fait grâce à un protocole appelé *NCasting* qui permet leur mise à jour en fonction des requêtes, et à une routine de maintenance qui permet de prendre les décisions de migration, duplication et surpression. Ces mécanismes sont entièrement distribués, ce qui rend notre algorithme résistant aux pannes et passable à l'échelle.

De plus, FReDi est l'adaptation distribuée d'un algorithme de graphe en ligne, ce qui assure son efficacité et son adaptabilité face à l'apparition de nouveaux usages dans un contexte dynamique. FReDi est également entièrement paramétrable, ce qui lui permet d'être adaptable à toute sorte de besoins. Son efficacité a été prouvée sur un simulateur exploitant les traces d'un serveur web réel sur de nombreux contenus téléchargés dans le monde entier.

Enfin, les travaux futurs incluent un système d'aide au paramétrage et la possibilité d'un paramétrage dynamique sera étudiée. De plus, nous verrons dans le chapitre suivant comment des prédictions peuvent être exploitées afin d'améliorer la précision des décisions de pilotage de FReDi.

13

Proactivité FReDi

L'algorithme FReDi présenté dans la section précédente présente des caractéristiques intéressantes de flexibilité face à l'hétérogénéité et la dynamicité de l'utilisation des ressources. En revanche, son fonctionnement est complètement réactif : les décisions sont prises après chaque requête et en fonction de leur historique.

Or, la proactivité est une technique très populaire dans les environnements très dynamiques, tels que les grilles pervasives. Elle consiste en l'exploitation de prédictions, le plus souvent des requêtes et de leur localisation, sur un futur plus ou moins proche, afin d'adapter les aspects d'importance aux nouveaux besoins avant même leur apparition.

Ainsi, une approche pertinente permettant de pallier à la réactivité de l'algorithme FReDi consiste à'utiliser des prédictions afin de rendre le comportement des réplicas proactif. En d'autres termes, l'objectif est de placer des réplicas en fonction des requêtes futures plutôt que les requêtes passées afin que ces requêtes soient servies au mieux. On peut ainsi placer un réplica sur un proxy-cache avant même que celui-ci ne relaye une première requête.

C'est pourquoi nous avons établi une collaboration avec une équipe du groupe de recherche « Distributed Multimedia Systems » de l'Université de Vienne, Autriche, composée par Hummel, Gansterer, Jacenek et Adrowitzer qui travaille sur la prédiction de la localisation de clients mobiles. Ils ont présenté dans [Hummel 05] un outil de prédiction des coordonnées géographiques de clients mobiles. L'objet de cette collaboration est d'interfacer FReDi avec leur outil de prédiction. L'objectif est double : pour l'équipe Vienoise, il s'agit de montrer l'utilité pragmatique de leur prédicteur pour une utilisation pragmatique ; pour nous, il s'agit de vérifier si les décisions de FReDi peuvent être améliorées par l'utilisation de prédiction.

13.0.1 Modèle de prédiction

Le modèle de prédiction est basé sur une stratégie de comparaison avec un historique d'entrainement : étant donné une position à un temps t_0, les m précédentes positions sont comparées aux séquences dans l'historique d'entrainement. Le paramètre m détermine de combien on remonte dans l'historique. On ne peut donc pas prédire la position d'un taxi sans historique de position.

En fonction de la précision avec laquelle les séquences dans l'historique correspondent à la séquence actuelle, un ensemble de n prédictions de positions sont calculées avec une probabilité spécifique pour les f étapes temporelles futures. Si toutes les correspondances parfaites s'accordent sur la même position, celle-ci est assignée d'une probabilité de 1, sinon la probabilité est ajustée en fonction de la qualité des correspondances de chacune des séquences historiques.

Les notations que nous avons adoptées, pour un client donné, sont :

- T l'intervalle des temps
- $posC_t$ la position réelle du client au temps t
- $posR_t$ la position du réplica au temps t
- $predC_t^i$ la prédiction $i \in [1, n]$ de position du client au temps t
- $probC_t^i$ sa probabilité

13.0.2 Intégration des prédictions dans FReDi

13.1 Expérimentations

13.1.1 Cas d'usage et méthodologie

Cette collaboration s'appuie sur les données réelles de localisation GPS de taxis viennois. Nous considérons que Vienne est couverte par un réseau de proxy-caches gérant des contenus. Les positions des taxis par rapport à ce réseau sont utilisées comme positions de clients émettant des requêtes pour un contenu répliqué. L'objectif est d'évaluer si le placement de ces contenus peut être optimisé par l'utilisation des prédictions des positions des taxis plutôt que par leurs positions réelles.

Nous avons utilisé les traces de 400 taxis, à raison d'une position toutes les 20 secondes. Ces taxis embarquent des équipements mobiles sans fil. Une grille de point d'accès, type hot-spots WIFI, couvre la ville de Vienne. Cette grille régulière est composée de N points d'accès, N pouvant varier, comme illustré sur la Figure 13.1. Ces points d'accès sont équipés de proxy-caches collaboratifs pouvant communiquer par un réseau câblé. Ainsi, chaque position de taxi a été utilisée comme position d'exécution d'une requête sur le point d'accès le plus proche (nous ne supposons aucun recouvrement). La distance utilisée pour établir la grille de points d'accès et déterminer les points d'accès les plus proche des positions des taxis est la distance loxodromique qui permet de transformer les coordonnées GPS exprimées en latitude et longitudes en coordonnées cartésiennes exprimées en x et y.

Nous avons testé l'utilité des prédictions dans deux cas complémentaires : le cas global et le cas

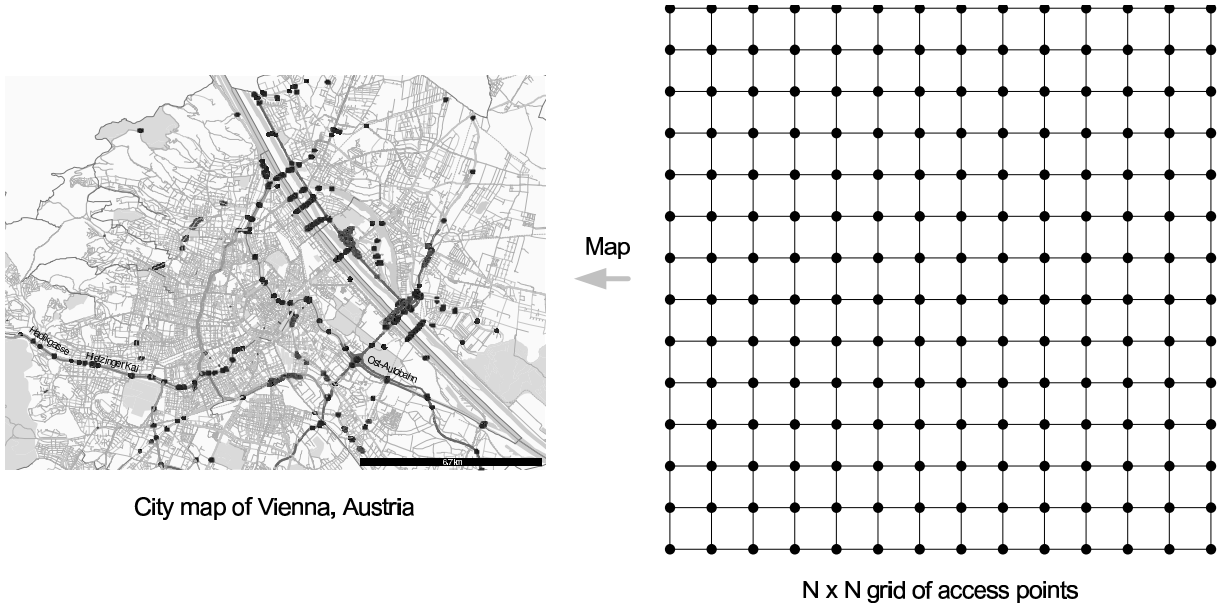


FIG. 13.1 – Correspondance des N^2 points d'accès sur la ville de Vienne, Autriche

individuel. Dans le cas global, l'ensemble des 400 taxis envoie des requêtes pour un même contenu. Il s'agit donc ici d'optimiser globalement la position des réplicas par rapport aux positions de l'ensemble des taxis. Les expériences ont montré que les prédictions n'apportaient pas de gain dans le cas global. Ceci est dû aux caractéristiques des traces utilisées qui ne présentent pas de phénomène de concentration de tous les clients en petits mondes à certains instants donnés, mais plutôt une couverture uniforme, presque aléatoire, d'une grande partie de la surface modélisée. Ceci rejoint les remarques faites dans la discussion sur l'algorithme FReDI. De plus, une analyse comparative approfondie des résultats afin de faire émerger des critères d'influence est impossible à moins de créer plusieurs sous-ensembles de taxis.

Dans le cas individuel, chaque taxi demande un contenu différent. Il s'agit donc d'optimiser la position des réplicas par rapport aux positions d'un seul taxi donné à la fois. Il est important d'observer que dans ce cas, il n'est pas pertinent de considérer plusieurs réplicas, mais plutôt de positionner avec précision un unique contenu. De plus, dans ce cas on va pouvoir mener une analyse comparative approfondie taxi par taxi afin d'identifier les critères qui influent sur les résultats.

Nous avons expérimenté quatre stratégies de placement :

`A` - **Actuelle :** Cette stratégie est basée sur les positions réelles des taxis. Elle correspond donc au comportement réactif initial de FReDi et sert de base de comparaison pour les autres stratégies.
A chaque temps t, le proxy-cache à la coordonnée $posC_t$ initialise le protocole NCasting avec un message

$$NCasting(R, d(posC_t, posR_t))$$

`PP` - **Prédiction parfaite :** Cette stratégie utilise la position réelle des taxis au temps $t+1$. Elle correspond donc à l'utilisation d'un prédicteur parfait ne présentant aucune erreur.
A chaque temps t, le proxy-cache à la coordonnée $posC_{t+1}$ initialise le protocole NCasting avec un message

$$NCasting(R, d(posC_{t+1}, posR_t))$$

`P` - **Prédiction :** Cette stratégie s'appuie uniquement sur les prédictions de position au temps $t+1$.
A chaque temps t, les proxy-caches aux coordonnées prédites $\forall i \in [1,n],\ predC_{t+1}$ initialisent le protocole NCasting avec un message

$$NCasting(R, d(predC^i_{t+1}, posR_t)/probC^i_{t+1})$$

`AP` - **Actuelle et Prédiction :** Cette stratégie combine les stratégies `A` et `P`.
A chaque temps t, les messages décrits pour les stratégies `A` et `P` sont émis.

Nous avons défini une métrique *critere* évaluant la précision du placement comme la moyenne arithmétique de la distance de Manhattan entre une requête et le réplica le plus proche, pour un client donné :

$$critere = \frac{\sum_{t \in T} d(posC_t, posR_t)}{|T|}$$

Une valeur particulière est $critere = 1$ qui correspond à un réplica stocké sur une proxy-cache directement voisin de celui du taxi.

Le simulateur développé pour les expérimentations de FReDi a été utilisé avec pour topologie la grille de points d'accès et pour traces de requête les positions réelles et prédites des taxis.

Résultats globaux

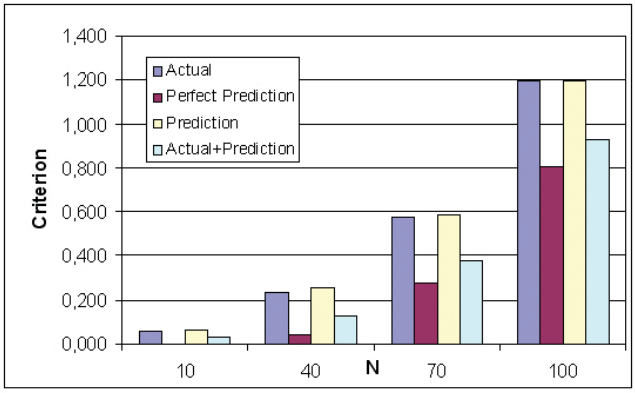


FIG. 13.2 – *critere* moyen des différentes stratégies en fonction de la taille de la grille N

La Figure 13.2 montre la moyenne de *critere* sur l'ensemble des 400 taxis, pour chaque stratégie et en fonction de N. Nous pouvons faire quelques observations :

- Tous les *criteres* augmentent avec N : car *critere* est calculé en fonction de la distance sur la grille de proxy-caches. Or, pour deux point de coordonnées GPS donné, la distance dans la grille est d'autante plus importante que N est grand. En revanche, les résultats comparatifs des différentes stratégies semble homogène quelqeu soit N.
- Le résultat `A` montre que FReDi est très précis : certains critères sont supérieurs à 1 uniquement lorsque $N = 100$, ce qui rends les résultats difficiles à améliorer.

- Le résultat PP montre que des prédictions parfaites sont tout de même utiles avec des améliorations allant de 100% pour $N = 10$ à 33% pour $N = 100$. En effet, il est plus difficile d'améliorer FReDi lorsque N est grand car cela implique plus de proxy-caches intermédiaires pour un même déplacement. Or, FReDi est basé sur des réplicas qui se déplace de proche en proche. Les réplicas mettent donc plus de temps à se déplacer.
- Le résultat P montre que la seule utilisation des prédictions fournies par le prédicteur ne représente pas une réelle amélioration.
- Le résultat AP n'est pas aussi bon que le PP, présente tout de même des améliorations intéressantes : de 50% pour $N = 10$ à 20% pour $N = 100$. Ces deux derniers aspects seront discutés plus précisément dans la section 13.1.1.

Distribution des améliorations :

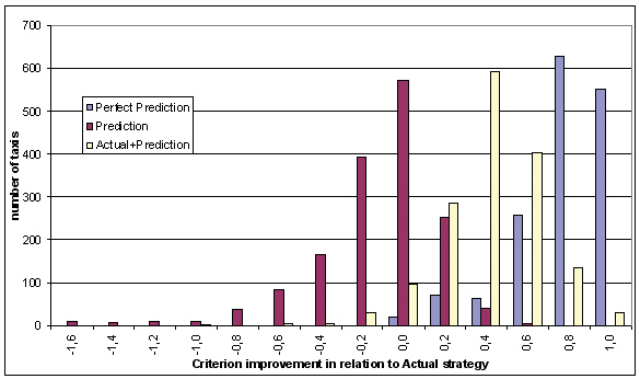


FIG. 13.3 – Distribution des améliorations pour les 400 taxis

L'amélioration de la stratégie X est :

$$amelioration(\texttt{X}) = \frac{critere(\texttt{A}) - critere(\texttt{X})}{critere(\texttt{A})}$$

Par exemple : $amelioration(\texttt{X}) = 0$ reflète que la stratégie X montre les même performances que la stratégie A ; $amelioration(\texttt{X}) > 0$ montre de meilleures performances ; $amelioration(\texttt{X}) < 0$ montre de moins bonnes performances. Plus précisément $amelioration(\texttt{X}) = 1,0$ correspond à l'amélioration parfaite : tous les éloignements entre client et réplica ont été corrigés ; $amelioration(\texttt{X}) = -1,0$ correspond au doublement de ces éloignements (ce qui n'est pas nécessairement catastrophique, par exemple pour $critere = 1$ qui devient $critere = 2$).

La Figure 13.3 montre la distribution des améliorations : chaque barre correspond à un nombre de taxis dans un intervalle d'amélioration donné. Par exemple l'intervalle noté $0,0$ correspond à $]-0.1, 0.1]$. On peut faire quelques observations :

- La stratégie PP montre la meilleur amélioration, suivie par la stratégie AP.
- La plupart des résultats de la stratégie P sont neutres.
- La stratégie AP montre une excellente amélioration : très peu de cas de dégradations ou neutre, de nombreux cas d'amélioration avec un pic autour de la valeur $0,5$.

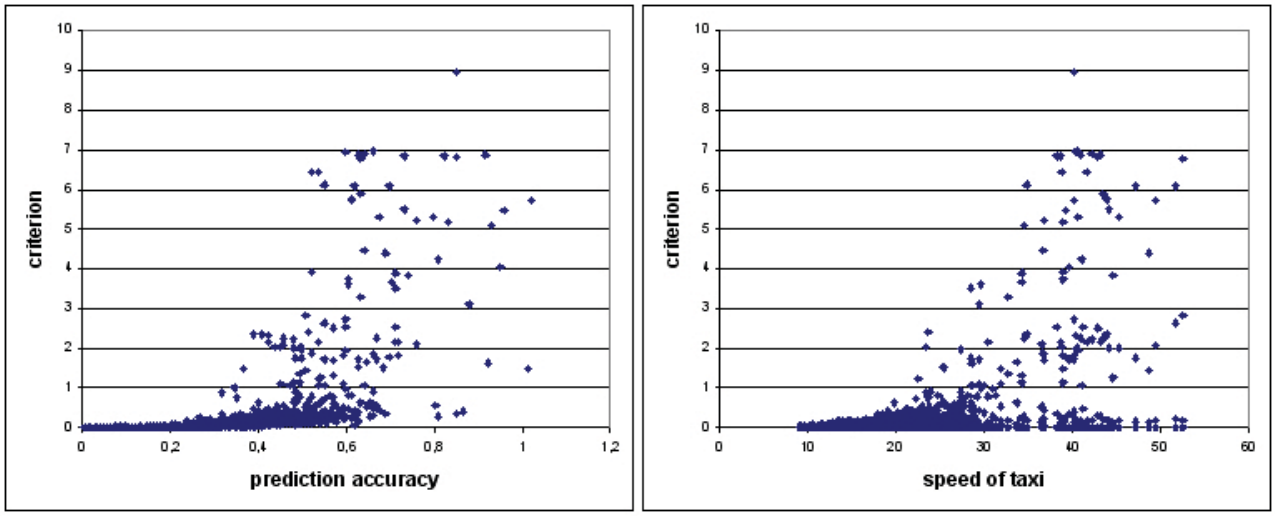


FIG. 13.4 – *critere* en fonction de (1) la précision des prédictions et (2) la vitesse des taxis

Influence :

La Figure 13.4 montre chaque *critere* de la stratégie `P` en fonction de : (1) la précision des prédictions (2) la vitesse moyenne des taxis.

La précision des prédiction est calculée par la moyenne arithmétique des différences entre la position réelle des taxis et la prédiction au même temps.

D'un côté, la précision du placement de la stratégie `P` est fortement corrélée avec la précision des prédictions : lorsque les prédictions sont bonnes ($< 0,3$), le placement est très précis, sinon le placement va de bon à très mauvais.

D'un autre côté, la stratégie `P` est précise lorsque la vitesse du taxi est faible (< 20), sinon elle va également de bon à très mauvais.

En fait, ces corrélations sont difficiles à analyser, car de nombreux paramètres sont liés : la précision du placement dépend de la précision des prédictions, laquelle dépend de la taille de la grille, mais cette dernière influence aussi la vitesse des taxis dans la grille, laquelle vitesse influence la précision des prédictions.

Analyse

Notre analyse des différents résultats mène à deux conclusions :

En premier lieu, compte-tenu des résultats des stratégies `PP` et `P`, la précision des prédictions est un facteur clé pour la précision des placements. Nous avons remarqué que sur certaines traces, à partir d'un temps donné, les prédictions accusaient un décalage avec les positions réelles. Ce décalage n'étant jamais résorbé, les placements suivants sont tous biaisés. Ce phénomène a été observé sur les

taxis roulant le plus vite. Ceci explique que la stratégie `P` ne présente pas de réelles améliorations globalement.

Ensuite, la stratégie hybride `AP` est pertinente pour limiter ce phénomène : elle n'atteint pas les résultats de la stratégie `P`, mais représente tout de même une amélioration intéressante avec pratiquement pas de cas de dégradation des performances.

13.2 Conclusion

Dans ce chapitre, nous avons montré comment FReDi pouvait être couplé avec un système de prédiction de position de clients mobiles afin d'améliorer la précision de ses décisions de placement. Le système de prédiction utilisé a été fourni par son équipe de développement située à Vienne, Autriche. Nous avons évalué notre approche grâce à des traces réelles issues de GPS de taxis viennois.

Il est apparu que la précision des prédictions était cruciale pour le fonctionnement de FReDi. En effet, des prédictions parfaites peuvent aller jusqu'à obtenir un placement parfait. En revanche, le prédicteur utilisé (comme tout prédicteur réel) propose des prédictions erronées qui limitent les améliorations observées. On peut limiter l'impact de ces erreurs en ne considérant pas seulement les positions prédites, mais également les positions réelles dans les décisions de FReDi.

Les travaux futurs pour l'équipe viennoise concerneront l'amélioration de la précision de leur prédicteur, dont nous avons montré le caractère critique pour le comportement de FReDi. De notre côté, nous voulons avancer dans l'analyse des résultats afin de découvrir plus précisément quels sont les critères d'influence sur le placement dans le cas de clients mobiles. De plus, nous allons réfléchir à une extension de FReDi permettant d'attribuer à un même réplica plusieurs stratégies de placement. En effet, il se peut qu'une stratégie soit plus efficace qu'une autre pour un taxi donné. Le calcul à la volée des critères d'évaluation des stratégies, éventuellement couplé à un système de paramétrage dynamique de FReDi, permettra alors d'adapter automatiquement les décisions de pilotage en fonction du comportement de chaque client individuellement. Notre objectif est de se rapporcher d'une autonomie adaptative de chacun des composant de FReDi.

Quatrième partie

GR-OLAP

14

DW, OLAP et surveillance

14.1 Administration de la grille

Les administrateurs sont responsables de l'infrastructure matérielle sous-jacente à la grille. La grille étant multi-institutionnelle, plusieurs groupes d'administrateurs agissent sur la grille, chacun n'étant responsable que d'une partie bien définie de l'infrastructure, correspondant généralement à une organisation traditionnelle. Ainsi, même les administrateurs disposant de la plus haute autorité sur les ressources, ne disposent en réalité pas de l'accès à toutes les informations sur l'ensemble de l'infrastructure. Néanmoins, ces administrateurs sont en charge de nombreuses décisions qui concernent le déploiement de la superstructure logicielle aussi bien que les évolutions matérielles. Or, ces décisions doivent se baser sur une connaissance avancée de l'état et de la topologie de l'infrastructure matérielle.

En fait, nous pouvons identifier trois axes d'analyse de l'infrastructure de la grille qui doivent être pris en compte lors des décisions administrateurs :

- le temps
  Les informations de surveillance ne concernent que les quelques dernières minutes écoulées, ce qui est suffisant pour prendre des décisions opérationnelles (p. ex. la sélection d'un hôte pour une tâche donnée à un instant donné). En revanche, les administrateurs peuvent être intéressés par la connaissance de l'évolution de la grille sur plusieurs mois, voire années pour les systèmes les plus pérennes. En fait, ils peuvent aussi bien avoir à s'intéresser à l'état de la grille minute par minute pour toute la durée d'un jour donné il y a plusieurs semaines afin d'analyser un incident, qu'aux capacités globales de la grille depuis sa création, mois par mois afin d'analyser sont évolution. De plus, ils doivent pouvoir prendre les cycles de vie humains en compte dans leur analyse : heures ouvrées, jours ouvrés, week-end, jours fériés, périodes de vacances...
- l'infrastructure matérielle (ou espace)

Les grilles sont composées de plusieurs sites sous différentes administrations, lesquels sont composés de sous-réseaux (noté *LAN*), chacun regroupant un certain nombre d'hôtes. De plus, les organisations virtuelles (VO) constituent des sous-ensembles de ces entités, regroupant par exemple un site entier, avec une LAN d'un autre site plus quelques hôtes isolés. Un exemple et montré Figure 14.1. Les administrateurs peuvent donc avoir besoin d'informations aussi bien au niveau global sur l'ensemble de la grille, ou au niveau global sur l'ensemble d'une VO, qu'au niveau d'un hôte donné ou à n'importe quel niveau intermédiaire. En outre, l'infrastructure est composée, en plus des hôtes, des liens de communication entre eux. Ainsi, le décisionnaire est intéressé dans des statistiques telles que le minimum, le maximum, la moyenne ou la variance des capacités de calcul, stockage ou autre d'un hôte donné aussi bien que celles d'une VO donnée, mais aussi celles des capacités de communications entre ces hôtes.

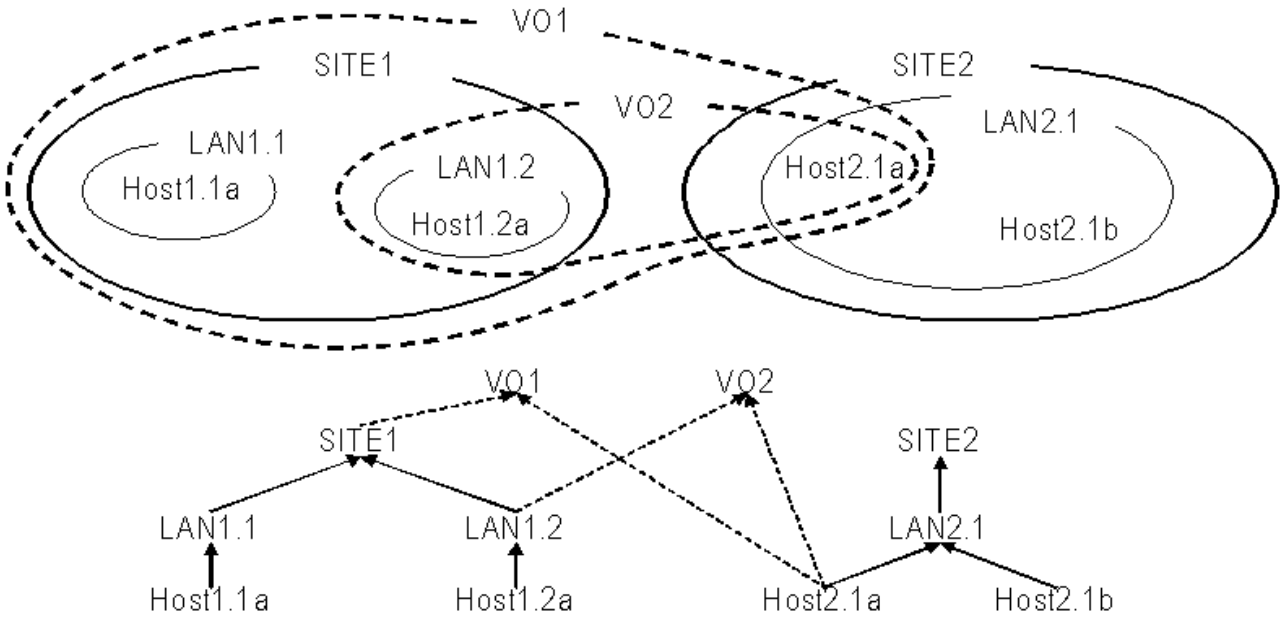


FIG. 14.1 – Exemple d'infrastructure hiérarchique de grille avec VO

- les services

  En se basant sur le paradigme des architectures orientées services, les applications, que nous désignons de façon un peu restrictive par services, sont d'une grande importance pour les décisionnaires. Les services peuvent être classifiés selon plusieurs taxonomies, dont les plus courantes sont thématiques ou sémantiques et se retrouvent dans les annuaires de services tels qu'UDDI[1] de OASIS. Malheureusement, ces classifications sont des outils professionnels, d'autant plus difficiles à obtenir qu'ils sont coûteux à développer. Nous nous sommes donc basés sur un échantillon utilisé à titre de preuve de concept, comme montré Figure 14.2. Une hiérarchie sémantique complète, telle que celle présentée dans la Thèse de David Coquil [Coquil 06], devra bien sûr être intégrée dans l'application finale.

[1] http ://www.uddi.org/

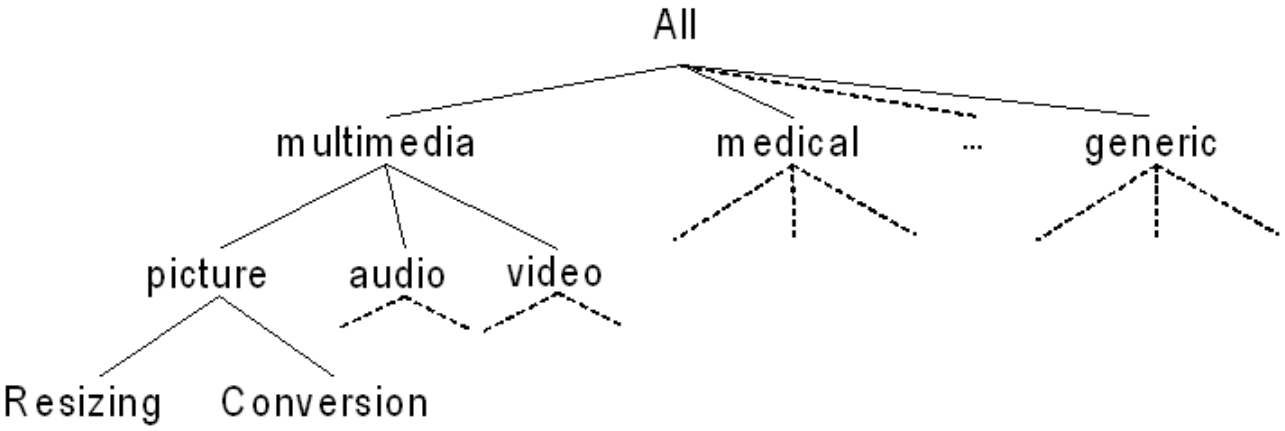


FIG. 14.2 – Exemple d'arbre hiérarchique de classification sémantique de services

14.1.1 Limites des outils de surveillance

Les outils de surveillance existants sont dépourvus des capacités d'analyse nécessaires aux besoins des administrateurs tels qu'exposés dans la section précédente. En effet, l'expressivité de leur système d'interrogation est limitée seulement au niveau de granularité le plus détaillé. Leurs utilisateurs ne peuvent donc pas obtenir de vues globales sur l'état ou les capacités de la grille, car les informations stockées ne concernent que les états actuels. Par exemple, ils supportent des requêtes du type : « Quelle est la vitesse et la charge du CPU de chaque hôte de la grille ? ». Ce qui ne permet pas de détecter, par exemple, les périodes de sur/sous utilisation des ressources, ou encore les sites disposant d'une grande capacité de stockage. En résumé, les solutions existantes ne permettent pas d'obtenir des vues globales ou agrégées sur les ressources, et par la même ne permettent pas leur analyse.

De plus, les interfaces des outils de surveillance actuels sont souvent limitées à des API (Application Programming Interface), éventuellement accompagnées d'interfaces web de présentation. Ces interfaces sont inadaptées à des fins d'analyse : les interfaces graphiques sont bien souvent développées pour être inclues dans des clients web légers et présentent des capacités de présentation limitées pour la plupart à des affichages textuels accompagnés de graphiques sommaires (courbe du nombre de réservations/jobs en fonction du temps, graphiques en secteurs (ou camembert) pour le ratio de machines réservées ou actives). Mais même ces interfaces graphiques manquent de fonctionnalités permettant la « naviguation » interactive dans les informations, qui est une base importante des recherches nécessaires à l'extraction de connaissances et à la prise de décision.

Notre approche est de mettre à profit la similarité entre les axes d'analyse présentés précédemment et les modèles d'analyse multi-dimentionnelle afin d'améliorer l'expressivité des requêtes sur les informations de surveillance et d'ainsi permettre leur analyse à des fins de décision. Ces modèles sont mis en œuvre dans les entrepôts de données et les processus OLAP, que nous allons présenter dans la section suivante.

14.2 Datawarehouse , OLAP

14.2.1 Définition et usages

L'entreposage de donnée, additionné des technologies OLAP (*On Line Analytical Processing*), est un outil innovant d'aide à la décision en découverte de connaissance et en intelligence économique. Il est désormais un sujet majeur dans l'organisation économique aussi bien que dans les communautés scientifiques. La motivation principale est de tirer bénéfice de l'énorme volume de donnée présent dans les bases de données hétérogène et distribuées afin d'améliorer l'analyse des données et les prises de décision [Kimball 02].

Un entrepôt est une collection de données orientée métier, intégré, non volatile, archivée et incrémentale, en règle générale stocké sur un seul site dépôt et collectées depuis de multiples sources [Inmon 96]. Les informations d'un entrepôt sont organisées selon un modèle multidimentionnel ayant pour objectif de permettre le précalcul et l'accès rapide aux données résumées afin de servir d'aide à la décision. Ce modèle multidimentionnel organise les données en axes d'analyse appelés dimensions. Les faits analysés sont caractérisés par des métriques appelées mesures. Les dimensions peuvent être organisées selon un schéma hiérarchique, permettant ainsi la navigation dans les différents niveaux d'analyse.

Un serveur OLAP calcule et optimise l'hypercube, c.-à-d. l'ensemble des valeurs de fait pour toutes les combinaisons d'instances de dimension (appelé membre). Dans le but d'optimiser l'accès aux données, le résultat des requêtes est précalculé sous forme d'agrégat. Par exemple, si l'on considère l'axe temporel, cinq agrégats peuvent être précalculés : pour le jour, la semaine, le mois, l'année et le temps complet. Ceci permet au décisionnaire d'explorer les différentes dimensions à différentes granularités. Ce processus d'analyse est conduit par la navigation dans le cube multidimentionnel grace aux opérateurs OLAP classiques. Ces opérateurs permettent de naviguer intuitivement dans les données en accédant interactivement à différents niveaux de détails et en changeant les axes d'analyse à la volée.

Finalement, une interface utilisateur interactive (OLAP client) permet de supporter la découverte de connaissance en encourageant la nature itérative des processus d'analyse. Les clients OLAP représentent visuellement la structure multi-dimentionnelle de l'hypercube et formule les requêtes multi-dimentionnelles. Le paradigme le plus populaire est celuit des tables Pivot : une feuille deux dimensions associant totaux et sous-totaux, supportant les données complexes en emboitant plusieurs dimensions sur les axes x ou y et en affichant les données sur plusieurs pages. Des exemples concrets seront montrés Section 14.4.

L'architecture trois-tiers entrepôt de données, serveur OLAP et client OLAP permet effectivement de conduire des analyses multidimensionnelles.

Dans la suite, nous nous basons sur le modèle multidimensionnel conceptuel MultiDimEr présenté par Malinowski dans [Malinowski 06]. Les détails de ce modèle sont présentés Figure 14.3.

Ce modèle a le mérite d'être relativement intuitif à comprendre sur un exemple concret, lequel est donné Figure 14.4.

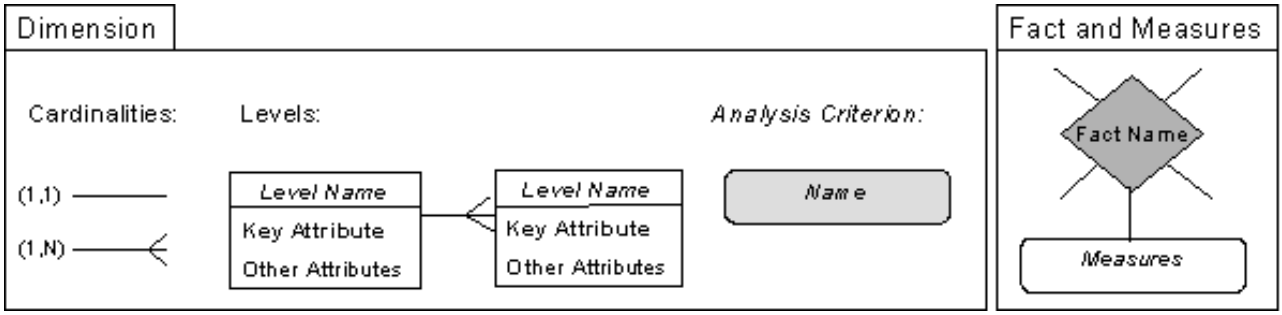


FIG. 14.3 – modèle multidimensionnel conceptuel MultiDimEr

14.3 GR-OLAP

Notre proposition pour répondre aux besoins énoncés dans la section précédente n'est pas de concevoir une nouvelle solution de surveillance, mais plutôt d'utiliser les données collectées par les outils existants une architecture trois-tiers dédiée à leur analyse multi-dimensionnelle. Ainsi, les données de surveillance servent à approvisionner un entrepôt de données, lequel permet l'utilisation des processus d'analyses OLAP au travers de client OLAP élaborés permettant une réelle navigation interactive dans les informations sur la grille. Le problème principal consiste à concevoir le modèle multidimensionnel adapté aux informations de surveillance. Comme décrit dans la section suivante, la conception de l'hypercube soulèvent de nombreux problèmes tels que la conception de hiérarchie non-stricte et non-couvrante, l'utilisation de mesure non-additive ou semi-additive, l'ensemble des faits menant à un schéma en constellation d'étoiles... C'est pourquoi une conception précise de l'hypercube s'impose afin d'améliorer les capacité d'analyse des informations de surveillance de grille.

La collaboration avec Sandro Bimonte prend ici son importance. Sandro a pris en charge ces problématiques peu familières, lesquelles sortent par ailleurs du champ scientifique de cette thèse. GR-OLAP, pour *GRid-OLAP*, est né de cette collaboration étroite où chacun de nous a apporté son expertise : Sandro s'est occupé d'identifier les problèmes intrinsèques aux entrepôts, lesquels problèmes ont été résolus ensembles.

14.3.1 Modèle conceptuel Multi-dimensionnel

La représentation conceptuelle basée sur MultiDimEr (cf. Figure 14.3) de l'application GR-OLAP est montrée Figure 14.4. Les dimensions et faits sont présentés ci-dessous.

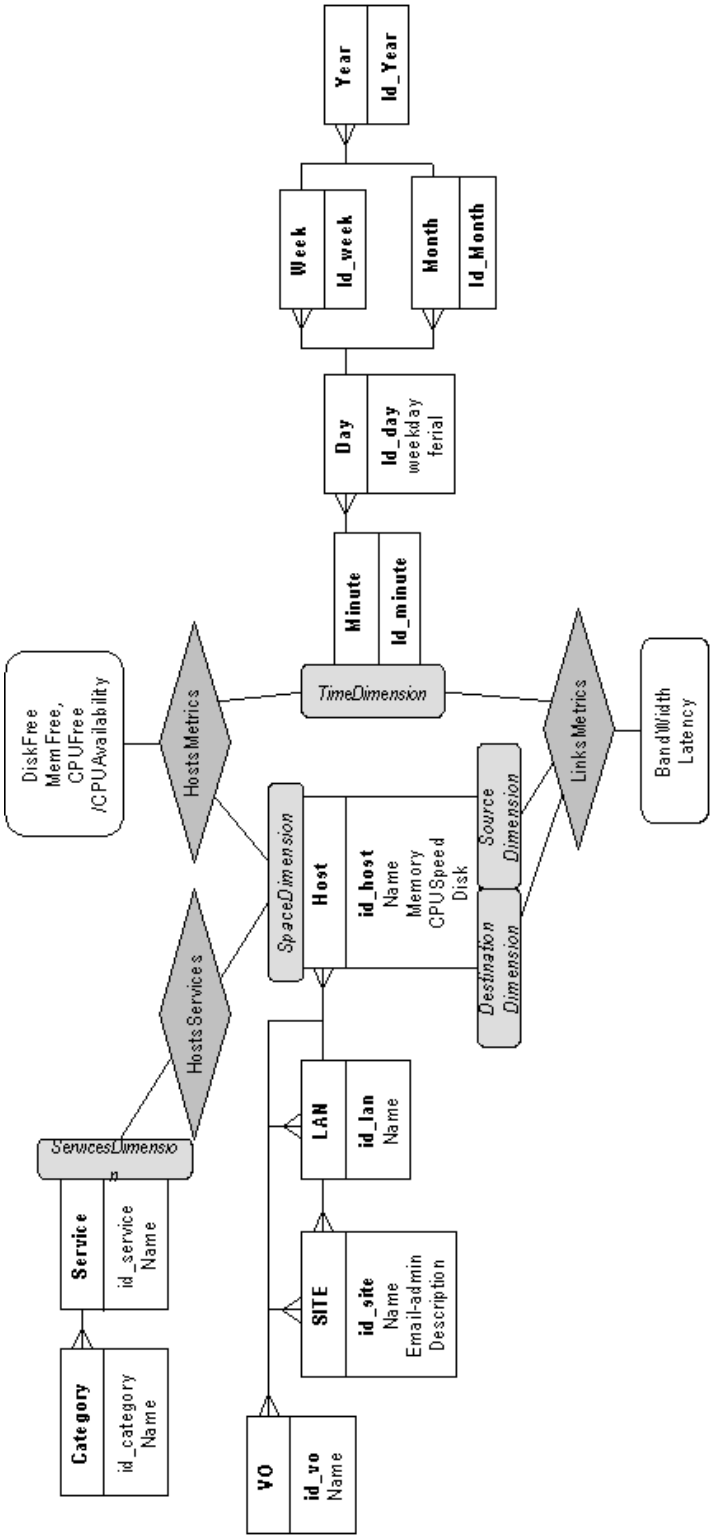


FIG. 14.4 – Modèle conceptuel Multi-dimensionnel de GR-OLAP

Dimensions

- SpaceDimension
  Cette dimension est une hiérarchie non-couvrante et non-stricte dont les différents niveaux sont : les hôtes (notés Host), les sous-réseaux (notés LAN), les sites (notés SITE) et les organisations virtuelles (notées VO). Elle est non-stricte du fait de la modélisation des VO qui implique des relations n-n entre membres de différents niveaux : un hôte et/ou une LAN et/ou un SITE peuvent appartenir à plusieurs VOs simultanément. Dans ce type de hiérarchie, les membres peuvent « sauter » certains niveaux : Host peut sauter les niveaux LAN et SITE, LAN peut sauter le niveau SITE afin d'être directement relié à une VO. Un exemple de cette hiérarchie est montrée Figure 14.1 : Host21a est directement relié à VO1 et VO2 ; LAN12 est directement connectée à VO2.
- TimeDimension
  Cette dimension multiple présente deux hiérarchies alternatives correspondant aux calendriers classiques : minute, jour, semaine et année ; minute, jour, mois et année. De plus, les jours fériés et les jours ouvrés sont marqués. Ainsi, l'utilisateur peut naviguer à sa guise dans les informations selon le calendrier le mieux adapté à son investigation.
- ServiceDimension
  Cette dimension est une hiérarchie simple comme montrée Figure 14.2.

Fact table

- HostsMetrics
  HostsMetrics contient les informations dynamiques de surveillance des hôtes : le ratio de temps CPU oisif (CPUFree), la quantité de mémoire vive libre (MemFree) et l'espace disque libre (DiskFree). De plus, nous avons ajouté une mesure dérivée représentant la capacité de calcul disponible (CPUAvailability). Ses dimensions sont l'espace (SpaceDimension) et le temps (TimeDimension).
  Les mesures sont agrégées par la moyenne, le minimum et le maximum sur la dimension temps et avec la somme, la moyenne, le minimum et le maximum sur la dimension espace. Ainsi, nous faisons face à des données semi-additives[2], l'addition des données de surveillance au cours du temps ne faisant pas sens. Cet aspect est précisé car il sort du contexte classiques des analyses OLAP et peut ainsi poser problème selon la solution d'implémentation retenue.
  Les métriques statiques sont stockées en tant qu'attribut des membres Host. La modélisation de ces attributs est indispensables non seulement pour la navigation dans l'hypercube, mais aussi pour le calcul des mesures dérivées telles que CPUAvailability.
  Grâce à cette table de fait, les administrateurs peuvent par exemple :
  – évaluer la distribution des capacités des ressources et de leur charge.
    Afficher la somme des DiskFree pour chaque VO au 08-2006 avec mise en évidence des minimums
  – découvrir les périodes durant lesquelles les ressources sont sous-utilisées afin de planifier des tâches très coûteuses.
    Afficher la somme des CPUAvailability pour SITE1 pour chaque semaine de 2006 avec mise en évidence des maximums

[2] les données semi-additives ne peuvent être additionnées que selon certaines dimensions

– évaluer l'évolution de l'infrastructure
Afficher la moyenne des MemFree pour chaque SITE pour chaque semaine depuis la création de la grille

- LinksMetrics

  LinksMetrics contient les mesures des métriques réseau : nous avons intégré la latence (Latency) et la bande-passante (Bandwidth), mais d'autres métriques pourront être ajoutées si besoin. Ses dimensions sont le temps (TimeDimension) et l'espace (SourceDimension et DestinationDimension). Ces deux dernières dimensions empruntent leur hiérarchie à SpaceDimension et représentent la source et la destination des liens de communication. L'exploitation de cette hiérarchie permet d'interroger le modèle multi-dimensionnel sur les liens entre des groupes d'hôtes (par exemple entre une LAN et un SITE).

  Les mesures sont agrégées avec la moyenne, le minimum et le maximum selon le temps et l'espace.

  Grâce à cette table de faits, les administrateurs peuvent par exemple :
  – comparer les capacités de communication internes des différents sous-réseaux.
  Afficher la moyenne des Bandwidth et Latency pour chaque LAN au O8-2006
  – évaluer l'évolution de la constance des capacités de communication entre deux sites.
  Afficher le minimum et le maximum des Bandwidth et Latency depuis SITE1 vers SITE2 pour chaque jour de 2006
  – sélectionner un sous-réseau pour stocker une très large collection de données distribuées en vue de l'utiliser sur un hôte donné
  Afficher la moyenne de Bandwidth depuis chaque LAN vers Host21a au 24-08-2006 avec mise en évidence des minimums et maximums

- HostServices

  HostsServices contient l'ensemble des services exécutés sur les hôtes de la grille. Elle peut être utilisée soit pour sélectionner ces hôtes en fonction d'une classe de services, soit pour trouver les services exécutés sur un ensemble d'hôtes.

  Ses dimensions sont l'espace (SpaceDimension) et la taxonomie sémantique des services (ServicesDimension). C'est une table de fait sans fait, ainsi la seule agrégation est un compteur (COUNT) du nombre de service. Il faut noter que le but ici n'est pas de découvrir les services à des fins d'utilisation immédiate (ce que font déjà les outils d'information de grille, tels qu'MDS), mais d'analyser sémantiquement la répartition spatio-temporelle des services.

  Grâce à cette table de faits, les administrateurs peuvent par exemple :
  – découvrir quels sites présentent un intérêt particulier dans les applications médicales.
  Afficher le nombre de services dans la classe Medical exécutés sur chaque SITE
  – vérifier la distribution des applications intéressant les VO
  Afficher le nombre de services pour chaque catégorie pour chaque VO

14.3.2 Navigation dans l'hypercube

La plus grande puissance de notre approche réside dans la possibilité de naviguer dans tous les faits simultanément. Chaque table de faits couvre un des aspects de l'interrogation multi-dimensionnelle. Comme montré Figure 14.4, HostsMetrics et LinksMetrics partagent des dimensions communes. L'application de l'opérateur *drill across* et les membres des dimensions communes permettent de mettre

en perspectives les relations entre faits. Par exemple, pour sélectionner les meilleurs sous-réseaux pour participer à une nouvelle organisation virtuelle ou des services médicaux vont produire de larges quantités de données à transmettre à des services multimédias pour des traitements très coûteux :

- *Afficher le nombre de services dans la classe Multimedia et la somme des CPUAvailability, MemFree et DiskFree pour chaque LAN au 08-2006, avec mise en évidence des maximums et minimums*
- *Afficher le nombre de services dans la classe Medical et la somme des DiskFree pour chaque SITE au 08-2006, avec mise en évidence des maximums*
- *Afficher les Latency et Bandwidth entre les LAN des deux requêtes précédentes avec mise en évidence des maximums*

Il est important de rappeler ici que les requêtes présentées sont cruciales pour les tâches des administrateurs bien qu'elles ne soient absolument pas supportées par les outils de surveillance actuels.

14.4 Implémentation de l'application GR-OLAP

Le modèle multi-dimensionnel de l'application GR-OLAP a été implémenté en utilisant le système de gestion de base de données Oracle[3], le serveur OLAP Mondrian[4] et le client web OLAP JPivot[5].

L'entrepôt a été approvisionné avec des échantillons basés sur de vraies informations de surveillance mesurées par NWS sur notre grille test composée de 7 hôtes très hétérogènes répartis sur trois sites (Lille, Toulouse et Lyon) connectés via l'Internet, pendant une période de 3 heures. Ces données ont répliquées avec des changements aléatoires de 10% maximum de la valeur d'origine afin de simuler une surveillance sur deux mois à cheval sur deux années, dans le but d'obtenir plusieurs membres par niveaux de la hiérarchie temporelle (c.-à-d. plusieurs années et mois représentés).

En réalité, l'approvisionnement de l'entrepôt GR-OLAP avec les données issues de systèmes de surveillance en production ne présente aucun problème. En effet, le standard Grid Monitoring Architecture (GMA), décrit section ***, est absolument compatible avec les processus ETL (*Extract-Transform-Load*) communément mis en place dans les solutions OLAP et qui consiste à extraire et transformer les données depuis des sources réparties hétérogènes afin de les charger dans l'entrepôt. Du point de vue de l'architecture logicielle, plusieurs sources hétérogènes (appelées producteurs dans GMA et sources dans les processus ETL) peuvent être intégrées sans peine grâce au système d'abonnement aux flux prévus dans GMA : le processus ETL et l'entrepôt ne sont alors que des consommateurs classiques de l'architecture GMA. Du point de vue de l'interopérabilité, les langages d'interrogation des systèmes de surveillance sont, pour la plupart, basés sur des standards ordinaires (SQL, XML ou LDAP), ce qui réduit leur intégration à de la simple configuration. On peut affirmer que l'application GR-OLAP peut être déployée sans plus d'effort que n'importe quelle application OLAP compatible avec son environnement.

Dans les paragraphes suivants, nous présentons trois différentes captures d'écrans du client web JPivot montrant les capacités d'analyse multi-dimensionnelle de GR-OLAP. La barre d'outils fournit

[3] http ://www.oracle.com/
[4] http ://mondrian.pentaho.org/
[5] jpivot.sourceforge.net/

le déclenchement interactif des opérateurs OLAP.

14.4.1 Table de faits HostsMetrics

GR-OLAP Application

		Mesures			
TimeDimension	SpaceDimension	CPUFREE	MEMFREE	DISKFREE	CPUAVAILABILITY
−All TimeDimensions	+PHY	4.77	5,018.53	1,437,780.30	44,832.11
	+VO1	3.82	4,897.63	1,399,978.51	33,271.40
	+VO2	2.83	3,989.84	604,999.40	13,322.47
−06	+PHY	4.77	5,018.53	1,437,780.30	44,832.11
	+VO1	3.82	4,897.63	1,399,978.51	33,271.40
	+VO2	2.83	3,989.84	604,999.40	13,322.47
+08-06	+PHY	4.76	5,016.96	1,437,664.46	44,777.79
	−VO1	3.82	4,895.93	1,399,880.99	33,220.34
	−SITE1	2.88	4,774.88	1,362,115.70	23,002.64
	+LAN11	0.99	907.81	795,611.57	3,945.29
	+LAN12	1.89	3,867.08	566,504.13	7,556.03
	−vSITE1	0.94	121.05	37,765.29	660.17
	−vLAN21	0.94	121.05	37,765.29	660.17
	HOST21a	0.94	121.05	37,765.29	660.17
	+VO2	2.83	3,988.12	604,269.42	13,310.90
−09-06	+PHY	4.78	5,022.62	1,438,081.72	44,973.44
	+VO1	3.84	4,902.05	1,400,232.26	33,404.26
	+VO2	2.84	3,994.32	606,898.92	13,352.55
+01-09-06	+PHY	4.81	5,005.06	1,440,516.67	45,231.62
	+VO1	3.86	4,884.71	1,402,741.67	33,621.88
	+VO2	2.85	4,005.94	608,200.00	13,403.81
+02-09-06	+PHY	4.76	5,041.36	1,435,484.44	44,698.04
	+VO1	3.81	4,920.56	1,397,555.56	33,172.13
	+VO2	2.83	3,981.93	605,511.11	13,297.87

FIG. 14.5 – Requête multi-dimensionnelle sur la table de faits HostsMetrics

La Figure 14.5 montre l'affichage résultant d'une requête multi-dimensionnelle sur la table de faits HostsMetrics. L'utilisateur peut naviguer dans la structure physique de l'infrastructure matérielle de la grille (membre PHY) ou des différentes organisations virtuelles (i.g. VO1) grâce à la dimension SpaceDimension. La modélisation des VOs implique deux difficultés. Premièrement, elle rend cette dimension non-couvrante : certains niveaux peuvent être sautés, par exemple un hôte peut être directement relié à une VO, sautant les niveaux LAN et SITE. Nous avons donc décidé de l'implémenter avec des remplaçants virtuels (p. ex. vLAN21) qui permettent de conserver la hiérarchie malgré les sauts de niveau. Deuxièmement, elle rend la dimension non-stricte : les hôtes, sous-réseaux et sites peuvent appartenir à plusieurs VO. Ceci peut impliquer des problèmes d'imprécisions tels que le comptage multiple ou la répartition des valeurs de mesures [Kimball 02]. Heureusement, l'agrégation des mesures entre différentes VOs n'est pas pertinente et les mesures associées à un hôte participent entièrement aux agrégations au niveau VO. Ainsi, les problèmes d'imprécision sont naturellement évités dans l'application GR-OLAP.

Un exemple d'extraction de connaissance sur la Figure 14.5 est que la capacité globale de calcul de la V1 est deux fois supérieure à celle de la VO2. Analyse se réalisant de façon simple et intuitive

grâce à GR-OLAP, et ce sans connaissance des rapports administratifs des différentes évolutions de l'infrastructure matérielle.

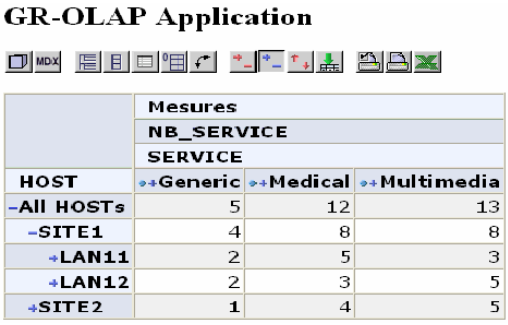
14.4.2 Table de faits LinksMetrics

GR-OLAP Application

		Mesures							
		AVG_LATENCY		AVG_BANDWIDTH		MIN_BANDWIDTH		MAX_LATENCY	
		GRIDD		GRIDD		GRIDD		GRIDD	
TimeDimension	GRIDS	+SITE1	+SITE2	+SITE1	+SITE2	+SITE1	+SITE2	+SITE1	+SITE2
-08-06	+SITE1	1.24	6.01	297.62	7.48	50.0	5.0	1.98	7.92
	-SITE2	5.99	1.48	7.47	74.53	5.0	50.0	7.92	1.98
	+LAN21	5.99	1.48	7.47	74.53	5.0	50.0	7.92	1.98
+25-08-06	+SITE1	1.17	6.13	331.83	7.26	52.0	5.0	1.98	7.6
	-SITE2	6.24	1.59	7.74	65.75	5.1	51.0	7.68	1.78
	+LAN21	6.24	1.59	7.74	65.75	5.1	51.0	7.68	1.78
+26-08-06	+SITE1	1.25	6.04	298.45	7.39	50.0	5.0	1.98	7.92
	-SITE2	6.02	1.54	7.45	74.79	5.0	50.0	7.92	1.98
	+LAN21	6.02	1.54	7.45	74.79	5.0	50.0	7.92	1.98
+27-08-06	+SITE1	1.25	6.03	298.83	7.40	50.0	5.0	1.98	7.92
	-SITE2	5.99	1.48	7.46	73.01	5.0	51.0	7.92	1.98
	+LAN21	5.99	1.48	7.46	73.01	5.0	51.0	7.92	1.98
+28-08-06	+SITE1	1.23	5.99	292.63	7.38	50.0	5.0	1.98	7.92
	-SITE2	5.93	1.46	7.37	74.53	5.0	50.0	7.92	1.96
	+LAN21	5.93	1.46	7.37	74.53	5.0	50.0	7.92	1.96
+29-08-06	+SITE1	1.24	6.04	296.69	7.66	50.0	5.0	1.98	7.92
	-SITE2	5.93	1.46	7.64	75.97	5.0	50.0	7.92	1.98
	+LAN21	5.93	1.46	7.64	75.97	5.0	50.0	7.92	1.98
+30-08-06	+SITE1	1.23	5.94	300.05	7.59	50.0	5.0	1.98	7.92
	-SITE2	6.07	1.47	7.42	74.73	5.0	50.0	7.92	1.98
	+LAN21	6.07	1.47	7.42	74.73	5.0	50.0	7.92	1.98

FIG. 14.6 – Requête multi-dimensionnelle sur la table de faits LinksMetrics

La Figure 14.6 montre l'affichage résultant d'une requête multi-dimensionnelle sur la table de faits LinksMetrics : les mesures associées aux connexions entre les sites SITE1 et SITE2 ainsi qu'entre ces sites et le sous-réseau LAN2 par mois et jour. On peut ainsi très aisément visualiser que le réseau de SITE1 est plus rapide que celui de SITE2. Mais on peut également détecter que l'écart sur les bandes passantes (respectivement 297,62 et 74,53, soit un rapport d'environ 4) est beaucoup plus important que l'écart constaté sur les latences (respectivement 1,24 et 1,48, soit un rapport d'environ 1.2). Ainsi, la latence n'est pas représentative des performances réseau sur ces deux sites. En conséquence, les outils basés essentiellement sur cette seule métrique (voir Chapitre 3) ne sont pas adaptés à cette infrastructure.



	Mesures		
	NB_SERVICE		
	SERVICE		
HOST	Generic	Medical	Multimedia
-All HOSTs	5	12	13
-SITE1	4	8	8
+LAN11	2	5	3
+LAN12	2	3	5
+SITE2	1	4	5

FIG. 14.7 – Requête multi-dimensionnelle sur la table de faits HostServices

14.4.3 Table de faits HostServices

Finalement, la Figure 14.7 montre l'affichage résultant d'une requête multi-dimensionnelle sur la table de faits HostsServices. On peut par exemple visualiser que SITE1 exécute autant de services médicaux que multimédias. Ce site est donc probablement dédié au traitement de contenus multimédias médicaux. On peut également voir que leur distribution varie au niveau des sous-réseaux : LAN11 contient 5 services médicaux pour 3 multimédias alors que LAN12 est à l'inverse. Un administrateur cherchant à réorganiser la superstructure pourra alors décider d'homogénéiser les sous-réseaux en répartissant les services de telle façon à ce qu'il y en ait 4 de chaque type sur chacun des sous-réseaux, ou *a contrario* décider de les spécialiser en mettant les 8 services médicaux sur un sous-réseau, et les 8 services multimédias sur l'autre.

14.4.4 Évaluation de l'espace disque consommé par GR-OLAP

Une facétie envisageable consiste à utiliser l'entrepôt de données distribué sur grille développé dans le cadre du projet GGM pour stocker les données de GR-OLAP. Afin de valider cette approche, nous avons étudié la taille des données devant être stockée dans notre application. Les tables de faits HostsMetrics et LinksMetrics sont les seules tables concernées par des insertions fréquentes, notre étude se concentre donc sur elles.

Soit h le nombre d'hôtes surveillés, d le délai entre deux mesures et p la période de temps qui concerne les données stockées. Si les identifiants des hôtes sont encodés sur 4 octets (une adresse IPv4) et que CPUFree, MemoryFree, DiskFree, Latency et Bandwidth sont également encodés sur 4 octets (types entier et réel standards), alors la taille cumulée s des deux tables est :

$$\begin{aligned} s &= \left[\text{taille des mesures sur les hôtes} + \text{taille des mesures sur le réseau}\right] \times \frac{periode}{delai} \\ &= \left[h \times sizeof(HostsMetrics) + h \times (h-1) \times sizeof(LinksMetrics)\right] \times \frac{p}{d} \\ &= \left[h \times (4 \times 4) + h \times (h-1) \times (3 \times 4)\right] \times \frac{p}{d} \end{aligned}$$

Ainsi, avec $h = 1000$, $p = 1\ mois$ et $d = 5\ minutes$: $s \approx 100Go$.

La difficulté principale est la taille de LinksMetrics qui croit en $O(h^2)$. Même sans considérer toutes les tables, ni les agrégations, ni les volumes spécifiques au système de gestion de base de données, la taille peut s'avérer, en première approche, problématique à l'échelle d'une année. En revanche, on peut remarquer qu'une fréquence de 5 minutes est généralement inutile à des fins d'analyse et doit être limitée à des cas besoins bien spécifiques. Dans la plupart des cas, on pourra se limiter à des fréquences de l'ordre de l'heure voire de la journée. De plus, les informations au niveau le plus détaillé peuvent être supprimée après un certain délai : les données heure par heure sont probablement inutiles après un mois, les données par semaines après une année, etc.

avec $pj = 1$ *jour* et $dj = 1$ *heure* les périodes et délais du jour, $ps = 1$ *semaine* et $ds = 1$ *jour* ceux d'une semaine, $pm = 1$ *mois* et $dm = 1$ *semaine* ceux d'un mois et $pa = 1$ *an* et $da = 1$ *mois* ceux d'une année, on obtient :

$$\begin{aligned} s &= \left[h \times (4 \times 4) + h \times (h-1) \times (3 \times 4)\right] \times (\frac{pj}{dj} + \frac{ps}{ds} + \frac{pm}{dm} + \frac{pa}{da}) \\ &= \left[h \times (4 \times 4) + h \times (h-1) \times (3 \times 4)\right] \times (24 + 7 + 4 + 12) \\ &\approx 500Mo \end{aligned}$$

On peut donc envisager de garder le volume des données dans des ordres de grandeur raisonnables sans pour autant affecter leur pertinence ou leur précision. L'utilisation de l'entrepôt de donnée du projet GGM n'est donc pas une obligation, mais pourra être envisagée dans les cas particuliers nécessitant un niveau de détail important sur un très long terme.

Il est important de noter que nous n'abordons pas les aspects de performance tels que la surcharge de maintenance des données, car notre application est en fin de compte une application OLAP absolument classique, ne comprenant que des données incrémentales (c.-à-d. sans aucune mise à jour de données déjà insérées) et on peut donc s'appuyer sur les solutions classiques pour prendre en charge ces coûts efficacement.

Nous concluons donc en disant que l'application GR-OLAP peut être mise en place dans un environnement classique d'analyse OLAP, sur un seul serveur puissant disposant d'une importante mais rationnelle capacité de stockage. Elle pourra néanmoins être mise ne place dans un environnement plus perfectionné, par exemple avec un entrepôt de données distribué, dans des cas particuliers nécessitant des données détaillées à haute fréquence devant être conservées sur une période étendue.

14.5 Discussion et travaux futurs

La principale limite de la solution d'analyse des informations de surveillance GR-OLAP concerne la dynamicité des dimensions. En réalité, sites, sous-réseaux et hôtes peuvent apparaitre et disparaitre, changer d'attributs ou de structure, et peuvent joindre te quitter des VOs à tout instant. Cette dynamicité affecte le schéma de la dimension espace et les utilisateurs sont intéressés par les traces de ces modifications. Le même problème s'applique ne moindre mesure à la dimension services. Malheureusement, les modifications à la volée des dimensions sont difficiles à gérer et implique des aspects de modélisation temporelle. Selon Sandro, ces dernières font l'objet de travaux actuels en analyse OLAP et certaines solutions sont proposées, mais au prix de baisse de performances ou de restrictions sur

l'expressivité des dimensions. L'investigation de ces solutions fait partie des travaux futurs connectés à l'amélioration de l'application GR-OLAP.

Deuxièmement, il serait intéressant d'inclure les aspects de surveillance d'utilisation dans GR-OLAP : quel utilisateur accède à quelle ressource et à quel moment. l'intégration de ces informations permettrait de faire émerger des profils d'utilisation typiques afin de planifier des évolutions adaptées de l'utilisation de la plateforme. Par exemple, dans le cas présenté Section 14.4.3, les informations sur les utilisations des services permettraient de savoir quel service est généralement couplé avec tel autre. Si les services sont relativement indépendants, l'administrateur pourra décider de spécialiser les sous-réseaux afin d'obtenir une architecture claire (tel sous réseau est dédié au multimédia, tel autre au médical) et d'adapter le dimensionnement de l'infrastructure en conséquence. Si les services sont interdépendants, l'administrateur pourra placer sur les mêmes sous-réseaux les services qui font l'objet du plus d'utilisation commune, indépendamment de leur classe sémantique.

Troisièmement, les cartes interactives et les affichages graphiques sont les principaux instruments des processus d'analyse spatio-temporelle efficaces. Ils ne révèlent pas seulement les relations et tendances spatio-temporelles, mais stimulent aussi les processus cognitifs de l'utilisateur. En effet, une représentation cartographique des données multi-dimensionnelles permet de visualiser les données sur un plan et de découvrir les corrélations spatio-temporelles entre faits et membres des dimensions. L'exemple typique étant la découverte des périodes de sur/sous utilisations des ressources en fonction du calendrier et des sites. C'est pourquoi une perspective consiste à utiliser un client OLAP spatial tel que GéOLAP [Bimonte 05].

Quatrièmement, la principale limite de notre client JPivot est que les différentes tables pivot ne sont pas interconnectées : l'utilisateur ne peut naviguer sur toutes les tables simultanément et doit même parfois utiliser MDX ou SQL afin de lier ses différentes navigations. C'est pourquoi une extension de JPivot doit être développée afin de naviguer dans plusieurs tables simultanément. Cet aspect va être investigué dans nos travaux futurs. Il se peut que des efforts modestes suffisent à intégrer plusieurs tables JPivot interconnectées au sein d'une même page web. Dans le cas contraire, une solution de rechange devra être choisie afin de pallier à cette lacune.

Enfin, les informations de surveillance sont par nature produites en temps réel. Les processus d'alimentation[6] et d'interrogation temps-réel des entrepôts de données sont des perspectives ouvertes à l'heure actuelle dont les solutions sont pleines de promesses pour l'application GR-OLAP.

14.6 Conclusion

Dans cette partie, nous avons présenté comment les technologies OLAP peuvent être appliquées aux informations de surveillance. Cette application innovante améliore l'expressivité des interrogations sur ces données afin de répondre aux besoins des administrateurs dont le travail est rendu difficile par la nature multi-organisationnelle de la grille.

Alors que le modèle conceptuel présente quelques particularités rendant son implémentation difficile sur les solutions existantes d'analyse OLAP, nous avons montré comment les principales difficultés pouvaient être contournées par des subtilités d'implémentation. Nous avons ainsi montré, au travers

[6]ETL dans le jargon du domaine

d'une implémentation, la faisabilité de notre approche ainsi que sa compatibilité avec les solutions existantes de surveillance. Nous avons de plus étudié la faisabilité de sa mise en place au travers d'une étude de l'espace disque consommé. Mais un des aspects les plus importants de notre contribution concerne les différentes analyses rendues possibles par l'application GR-OLAP, et leur importance dans la vie quotidienne des administrateurs.

Finalement, ce travail ouvre de nombreuses perspectives dans les utilisations pouvant être faites afin de pouvoir mieux comprendre, analyser et gérer les grilles dans une approche comprenant aussi bien l'infrastructure matérielle que la superstructure logicielle.

Cinquième partie

Conclusion

15

Conclusion et travaux futurs

15.1 Conclusion

Afin de conclure la rédaction de cette thèse, nous commençons par récapituler nos contributions en fonction des critères établis dans l'introduction : généricité, adaptabilité, adaptativité, hétérogénéité, dynamicité, évolutivité, utilisabilité et profitabilité.

15.1.1 Généricité face à différents problèmes

Le critère de généricité est central dans notre approche. Il désigne la capacité de pouvoir gérer un large panel de problèmes différents. Nous avons montré que ce critère était d'une importance majeure pour les superstructures de grille car, si certains problèmes génériques (sélection, déploiement et composition) se retrouvent dans de nombreuses applications, il existe également des problèmes spécifiques, tels que l'agrégation dans un entrepôt de donnée. Ces problèmes spécifiques ne peuvent être exhaustivement listés et peuvent également apparaitre lors d'évolutions de la plateforme. Il est donc indispensable que la résolution des problèmes de répartition soit générique afin de pouvoir prendre en compte un large spectre.

Certains de ces problèmes peuvent néanmoins être traités dans la littérature par des solutions spécifiques. Mais comme nous l'avons montré dans l'étude des travaux connexe, leur intégration n'est pas toujours possible et représente souvent de très importants efforts. En revanche, les outils de prédiction des performances et les outils de surveillance sont génériques dans le sens où ils ne résolvent pas directement des problèmes de répartition.

Afin de pouvoir résoudre les problèmes spécifiques, nous avons proposé une modélisation générique de l'aspect logique des problèmes liés à la répartition des ressources. Cette modélisation fait

l'objet des $\mathcal{CNP}$-Graphes compacts (Computer Network Problem Graph). Ces graphes sont composés de sommets qui représentent les fonctions logiques impliquées dans le problème à résoudre. Par exemple, pour un problème de sélection, le $\mathcal{CNP}$-Graphe est composé de deux sommets : un sommet représentant le ou les clients, et l'autre les réplicas de la ressource devant être sélectionnée. De plus, les sommets de ces graphes qui ont une interaction logique sont reliés par un arc labélisé avec une fonction distance permettant de modéliser le coût impliqué par cette interaction. Enfin, nous avons montré l'utilisation de ces graphes permettait d'obtenir des solutions très précises et faciles à analyser pour les trois problèmes génériques : sélection, déploiement et composition de ressources.

Cette modélisation est très simple à comprendre et manipuler, de sorte qu'un développeur se retrouvant face à un problème connu, mais sans solution, pourra le modéliser et le résoudre grâce à notre approche, sans avoir à développer une solution de résolution *ad hoc*.

15.1.2 Adaptabilité à différents objectifs

L'adaptabilité concerne la capacité à prendre en compte différents objectifs déclarés par l'utilisateur, y compris sémantique ou financier. En effet, nous avons montré que la seule considération des performances était trop restrictive pour couvrir tous les besoins des utilisateurs des grilles. Par exemple, un problème classique consiste à décider s'il est plus rentable de récupérer une donnée plus fraîche ou précise, mais avec un temps de récupération plus long, ou au contraire de récupérer rapidement une donnée de moins bonne qualité. L'identification des critères et de leur pondération dépend entièrement de l'expertise et des choix de l'utilisateur. En revanche, l'évaluation de chacun des critères impliqués doit être comprise dans le système de résolution des problèmes.

Notre étude des travaux connexes a montré qu'aucune solution ne s'intéressant aux problèmes de répartition ne permettait de prendre en compte un large spectre de critères combinés par l'utilisateur : alors que les solutions spécifiques ne permettent pas aux utilisateurs de déclarer leurs propres objectifs, les outils de prédiction et de surveillance des performances ne s'intéressent... qu'aux performances.

Notre proposition repose sur la déclaration de fonctions distances servant à labéliser les nœudes des $\mathcal{CNP}$-Graphes compacts. Ces fonctions distances permettent de prendre en compte aussi bien les critères de performance des performances, grâce à des métriques composées proches des préoccupations des développeurs, que tout autre critère tel que les coûts financiers. Ces critères peuvent être aisément combinés à la guise de l'utilisateur. Ensuite, ces $\mathcal{CNP}$-Graphes compacts sont transformés automatiquement en $\mathcal{CNP}$-Graphes. Ces derniers représentent la réalité physique des problèmes : leurs sommets sont des hôtes de l'infrastructure et leurs arcs sont labélisés avec des réels représentant le coût effectif de leur utilisation. Enfin, nous avons montré, sur de nombreux exemples, qu'un large choix d'objectifs était modélisable.

Les fonctions distances permettent donc bien aux utilisateurs de déclarer des modèles de coût reflétant des objectifs adaptés à leurs besoins.

15.1.3 Adaptativité aux caractéristiques de la superstructure

L'adaptativité concerne la prise en compte automatique des caractéristiques de la superstructure. Elle est hautement liée à l'hétérogénéité des ressources logicielles. En effet, la solution d'un problème de répartition est différente en fonction des caractéristiques des ressources. L'exemple le plus concret étant qu'une tâche consommant beaucoup de temps processeur, mais peu de communications devra être assignée à un hôte puissant, alors qu'une tâche calculant peu mais communiquant beaucoup devra être assignée à un hôte hautement connecté. L'adaptabilité est donc cruciale pour la pertinence des solutions obtenues par le processus de résolution des problèmes.

Nous avons montré que l'adaptativité est relativement bien traitée par les solutions spécifiques et les outils de prédiction des performances. En revanche, elle n'était pas prise en compte dans les outils de surveillance, car ils ne s'intéressent qu'aux performances de l'infrastructure, sans considérer la superstructure.

Nous avons proposé, pour répondre à ce critère d'adaptativité, une modélisation des caractéristiques des ressources sous forme de variables déclarées par l'utilisateur et réunies dans un ensemble *Task Properties Set* (TPS). Ces variables sont intégrées dans le calcul des coûts modélisés par les fonctions distances, et impacte sur les labels des $\mathcal{CNP}$-Graphes et donc sur les solutions obtenues par notre approche. De plus, ces variables ne nécessitent pas toujours une valeur réelle fixe, mais peuvent être décrites par une fonction de distribution, permettant d'identifier finement les caractéristiques des ressources soumises à des intervalles de paramètres conditionnant leur comportement. Nous avons également prouvé la pertinence de notre approche en montrant dans nos expérimentations que des solutions différentes étaient adoptées pour trois types de ressources différentes : communications intensives, calculs intensifs et équilibré ou versatile.

15.1.4 Hétérogénéité et dynamicité de l'infrastructure matérielle

L'hétérogénéité concerne la prise en compte d'infrastructures très différentes en terme de capacité et de topologie, ainsi que la présence dans d'une même infrastructure de matériels présentant des performances différentes. En plus des problèmes classiques de panne, capacité et topologie sont plus ou moins dynamiques en fonction de l'environnement : alors que les infrastructures de grille dédiées sont relativement stables, les environnements tels que les grilles pervasives et hautement partagées sont très dynamiques. En fonction de l'utilisation de la plateforme (nombre de tâches en cours, nombre d'utilisateurs connectés, etc.), les performances peuvent sensiblement varier. Cet aspect est crucial dans la résolution des problèmes liés à la répartition des ressources, car, dès lors que les performances sont prises en compte, les solutions dépendent fortement des capacités de l'infrastructure.

Nous avons montré que cet aspect était plutôt bien couvert dans les travaux connexes, en particulier par les outils de surveillance qui fournissent des informations exhaustives sur les différentes capacités et performances des matériels de calcul, stockage et communication.

La précision des résultats obtenus lors de nos expérimentations a permis de montrer que l'utilisation des informations de surveillance confère à notre approche une réelle adaptation aux caractéristiques et à l'état de l'infrastructure. De plus, nous avons conçu deux algorithmes exploitant les $\mathcal{CNP}$-Graphes et visant à déployer des ressources dans un cadre dynamique et hétérogène.

Le premier algorithme, dénommé $MRKM$ pour *Multi-Resources K-Médians*, résout une extension du problème des k-médians : il s'agit de prendre les décisions de placement de plusieurs ressources différentes. Nous avons intégré la prise en compte de deux aspects. Le premier est la charge impliquée par la décision de placement d'une ressource, le deuxième est l'affinité entre les ressources. Nous avons montré dans des expérimentations en environnement réel de grille que $MRKM$ permettaient effectivement de répartir équitablement les charges sur l'ensemble des hôtes disponibles, tout en gardant proches les ressources qui interagissent fréquemment. $MRKM$ peut donc être utilisé pour déployer un sous-ensemble, voire la totalité, d'une superstructure en tenant compte de l'hétérogénéité de la plateforme, grâce à un équilibrage des charges très précis.

Le deuxième algorithme, appelé $FReDi$ pour *Flexible Replica Displacer*, est un algorithme de pilotage de réplicas dynamiques. Il est adapté aux environnements extrêmement dynamiques nécessitant un haut niveau d'adaptation, tels que les environnements pervasifs. C'est une adaptation distribuée de l'algorithme d'approximation en ligne *DC-Tree*. Il est capable de régler dynamique le nombre et le placement de réplicas d'une ressource en fonction des requêtes. Il est conçu pour fonctionner sur un réseau de proxy-caches et est basé sur des mécanismes entièrement distribués, ce qui favorise le passage à l'échelle et la résistance aux pannes. L'efficacité de $FReDi$ a été prouvée avec des expérimentations sur un simulateur exploitant les traces d'un serveur web réel. De plus, $FReDi$ a fait l'objet d'une collaboration avec une équipe autrichienne, visant à améliorer la précision de ses décisions par l'intégration de prédictions. FReDi permet de déployer des ressources dans des environnements extrêmement dynamiques, en terme d'usage et de performances, grâce à un placement et un dimensionnement dynamique.

Hétérogénéité et dynamicité de l'infrastructure sont donc correctement prises en compte dans notre approche, y compris lorsqu'elles atteignent un seuil critique grâce à deux algorithmes développés spécifiquement.

15.1.5 Évolutivité des plateformes

L'évolutivité désigne les modifications profondes des plateformes. En effet, la durée de vie des plateformes peut facilement atteindre plusieurs années. compte-tenu de l'évolution technologique des matériels informatiques, les plateformes évoluent naturellement en terme de performances. Mais elles peuvent également évoluer de façon encore plus dramatique en fonction des collaborations et partenariats, tel qu'il est prévu dans les environnements collaboratifs de grille comprenant une notion d'organisation virtuelle. Dans ce cas, la plateforme peut se trouver subitement étendue ou diminuée. De plus, il est naturel que lors d'une exploitation à long terme, de nouveaux besoins et objectifs fassent leurs apparitions, par exemple lorsque plusieurs projets s'enchainent ou lorsque des décisions d'orientation sont prises. Enfin, les plateformes cibles ne sont pas nécessairement connues lors du développement des différents composants logiciels. Il est donc nécessaire de pouvoir adapter ces composants à leur infrastructure une fois qu'ils y sont déployés. Prendre en compte l'évolutivité des plateformes est crucial dans le développement des superstructures, puisque cela conditionne leur capacité à s'adapter aux changements ainsi que leur réutilisabilité dans de nouveaux contextes et donc leur pérennité.

Nous avons montré que les solutions spécifiques et les outils de prédiction des performances prenaient trop en compte les spécificités de leur contexte applicatif cible. Ils sont donc difficilement réutilisables, et doivent faire l'objet d'efforts d'adaptation importants lors de changements des plateformes. En revanche, les outils de surveillance sont conçus pour justement s'adapter à un vaste panel

d'infrastructures et refléter leurs caractéristiques.

Nous avons montré que notre approche permettait de s'adapter effectivement et de façon transparente à l'infrastructure, la première preuve étant la précision des solutions obtenues lors des expérimentations. Une deuxième preuve a été apportée en identifiant les points d'équilibre en coûts de calcul et coûts de communications sur deux plateformes différentes. Ces deux plateformes présentent des points d'équilibre très différents qui sont effectivement détectés par notre approche : notre système permet donc bien de s'adapter d'une part aux caractéristiques de la superstructure, mais surtout de les mettre en perspective avec les caractéristiques de l'infrastructure afin de supporter un large spectre de plateformes différentes, ainsi que leurs évolutions futures.

Mais aussi, nous avons conçu et implémenté un système de métriques composées facilement extensible. Ce système permet à des utilisateurs de déclarer et modifier des métriques composées d'une part, pendant que d'autres utilisateurs les exploitent dans leurs applications. Un changement dans les coûts modélisés pour une application peut donc être répercuté sur toutes les décisions basées sur ce coût, sans pour autant que le décisionnaire ait à intervenir. Ce système permet donc de supporter des changements profonds d'objectif. Par exemple, si, à un instant donné, des coûts financiers d'utilisation des ressources de calculs font leur apparition, il suffit alors que l'administrateur de la plateforme modifie la métrique composée évaluant le coût des calculs pour prendre en compte ce coût financier en plus des performances. Toutes les décisions basées sur cette métrique composée prendront alors en compte ce nouvel aspect, sans qu'aucun composant logiciel ne soit modifié.

15.1.6 Utilisabilité

L'utilisabilité désigne la capacité d'un système à être facilement compréhensible et utilisable par toute classe d'utilisateur. L'utilisabilité est cruciale dans le sens où elle conditionne, d'une part, l'adoption du système (un système trop complexe est rarement populaire), et d'autre part, les retours sur investissements suite à l'adoption de ce système. Or, cette notion de retours sur investissement est cruciale dans le monde des grilles, pas seulement d'un point de vue économique, mais aussi d'un point de vue scientifique. Par exemple, nous avons vu que tous les composants logiciels de la superstructure GGM présentent des besoins en terme de répartition des ressources, mais que ces besoins étaient secondaires face aux autres problématiques spécifiques à chacun des domaines. Or, la répartition des ressources conditionne fortement les performances de leurs composants. Il est donc très important de fournir aux développeurs de ces composants un système capable de résoudre les problèmes liés à la répartition des ressources, qui soit utilisable avec le moins d'efforts, que ce soit en terme d'acquisition ou d'intégration.

Notre étude des travaux connexes a montré qu'aucune des solutions proposées ne permettait réellement une acquisition et une intégration rapide : les solutions spécifiques doivent souvent être adaptées alors que les outils de de prédiction et de surveillance des performances fournissent des informations de trop bas niveau, qui demandent un effort important avant de pouvoir effectivement résoudre un problème de répartition.

L'utilisabilité de notre approche repose sur trois points. Premièrement, notre modélisation est souple et extensible, elle est donc très facile à comprendre et à manipuler. En particulier, la modélisation des problèmes sous forme de $\mathcal{CNP}$-Graphes compacts est intuitive, et peut souvent réutiliser des modélisations existantes : c'est le cas par exemple des chemins d'adaptation qui sont représentés par

des graphes correspondant exactement aux $\mathcal{CNP}$-Graphes compacts du problème de compositions de ressources. De plus, la déclaration des coûts au travers des fonctions distance peut être appréhendée de façon intuitive et les métriques composées permettent aux utilisateurs de prendre en compte des coûts sans savoir réellement comment ils sont calculés. Nous avons montré comment calculer des indices de satisfaction des propriétés euclidiennes, permettant de valider les $\mathcal{CNP}$-Graphes en vue de leur exploitation par différentes classes d'algorithmes. Ces indices peuvent être exploités par l'utilisateur pour le guider dans son choix des algorithmes de résolution, limitant également son travail sur cet aspect. Enfin, les $\mathcal{CNP}$-Graphes produits représentent des hôtes selon leur fonction logique dans le problème, ce qui les rend accessibles aux utilisateurs, qui peuvent les visualiser et les manipuler intuitivement. Cette modélisation permet donc l'acquisition facile de notre approche.

Le deuxième point est l'implémentation de notre approche dans le service web *Network Distance Service* (NDS). En plus de l'effort visant à rendre ce service facilement extensible, son intégration dans une superstructure est très facile : il possède une interface restreinte et une configuration facile à modifier. *A priori*, aucun effort de développement n'est nécessaire à son intégration. Cette implémentation rend donc notre approche facilement intégrable dans un large spectre de superstructures.

Enfin, nous avons proposé une conception et une implémentation d'un entrepôt de données, nommé GR-OLAP pour *GRid-On Line Analytical Processing*, capable de stocker les informations de surveillance issues de l'infrastructure matérielle d'une grille. Cet entrepôt permet d'augmenter les capacités d'analyse des outils de surveillance grâce au processus d'analyse OLAP et permet une navigation interactive dans les informations de surveillance grâce aux tables pivots. Il est particulièrement adapté à l'analyse spatio-temporelle de l'évolution de l'infrastructure et permet d'extraire des connaissances répondant à des besoins cruciaux d'administration. Cet entrepôt permet de rendre les informations de surveillance réellement utilisables par les administrateurs.

15.1.7 Profitabilité

Enfin, les systèmes de résolution des problèmes de la répartition doivent présenter, pour des raisons évidentes, un rapport coût/profit d'utilisation satisfaisant.

En particulier, nous avons montré que les outils de surveillance, bien que souvent indispensables, ne présentent pas une profitabilité satisfaisante : ils consomment généralement une part non négligeable des ressources, mais ne présentent en comparaison qu'un service limité, puisque les informations fournies doivent être retraitées par les utilisateurs afin de résoudre effectivement les problèmes.

Or, notre approche est générique, adaptable, adaptative et donc utilisable pour un vaste spectre de problèmes, objectifs et ressources. Il prend également en compte l'hétérogénéité et la dynamicité des plateformes, favorise leur pérennité et est facilement utilisable. Notre approche permet donc d'améliorer sensiblement les bénéfices de l'utilisation d'un service de surveillance en limitant fortement cette phase de retraitement. Enfin, nous avons montré que les temps d'exécution de notre service sont très courts et que les solutions obtenues étaient très précises, sa profitabilité est donc bien satisfaisante.

15.2 Un peu de recul

Sous un jour pragmatique, le projet GGM est une superstructure logicielle de grille conséquente et représentative. L'étude de cette superstructure a permis de faire émerger un besoin pratique commun à l'ensemble des développeurs et de leurs développements : le besoin de prendre des décisions de répartition de ressources en fonction de l'état et de la capacité de l'infrastructure. Nous avons étudié les différentes infrastructures matérielles cibles et identifié les contraintes connexes : généricité pour prendre en compte différents problèmes, adaptativité pour prendre en compte la dynamicité de l'infrastructure, utilisabilité pour être facilement intégrable et profitabilité. Nous avons alors développé une modélisation et des outils capables de répondre au besoin de résolution des problèmes de répartition en respectant ces contraintes. Nous avons validé ces travaux sur des expérimentations et ils peuvent être réutilisés dans tous les contextes présentant les mêmes caractéristiques, donc la plupart des superstructures de grille, en particulier celles basées sur une architecture orientée services. De plus, nos travaux permettent une adaptation transparente de la superstructure à l'infrastructure, favorisant ainsi leur réutilisabilité dans plusieurs environnements et leur résistance à des changements importants de nature de la plateforme.

Sous un jour plus fondamental, nous avons identifié un vide scientifique entre superstructures de haut niveau, telles que les architectures orientées services, et les infrastructures sur lesquelles elles sont implémentées. Parmi les approches proposées dans la littérature, les outils de surveillance de l'infrastructure seuls proposent des informations suffisamment complètes pour pouvoir résoudre efficacement les problèmes liés à la répartition des ressources. Mais ces outils sont de très bas niveau puisqu'ils se situent à la base des intergiciels, voire en dessous. Les superstructures quant à elles se situent au-dessus des intergiciels. Or, les intergiciels ont décuplé de taille et de complexité au cours des dernières années. À tel point qu'il est aujourd'hui très difficile de concilier le développement des superstructures avec la prise en compte de l'infrastructure. La contribution scientifique majeure de notre thèse est un moyen de réconcilier superstructure et infrastructure. Cette réconciliation est réalisée grâce à une modélisation par les utilisateurs des problèmes de la superstructure d'une part et une modélisation automatique de l'infrastructure d'une autre. Ces deux modélisations sont utilisées pour identifier dans des $\mathcal{CNP}$-Graphes, de façon transparente et intuitive pour l'utilisateur, la réalité physique des problèmes et ainsi pouvoir les résoudre aisément. De plus, cette approche étant clairement opérationnelle, nous avons également développé un entrepôt de donnée permettant aux administrateurs de naviguer intuitivement dans les informations de surveillance des infrastructures dont ils ont la charge afin de mieux en cerner la nature et de pouvoir en suivre l'évolution à long terme. Notre proposition est donc une modélisation et des outils permettant de réconcilier les superstructures de haut niveau avec leurs infrastructures.

15.3 Travaux futurs

En plus des différentes améliorations et investigations déjà présentées dans chacune des sous-sections, un premier effort concerne l'extension des outils que nous avons développés en vue de leur mise en production. Ensuite, une perspective concerne l'étude de deux aspects fondamentaux : plonger les problèmes de superstructure plus en profondeur dans la théorie des graphes, et exploiter les particularités des mesures des métriques de surveillance. Enfin, une dernière perspective concerne l'adaptation et l'intégration de nos travaux dans le cadre des systèmes autonomes.

15.3.1 Développement, intégration et mise en production d'une suite logicielle orientée utilisateur

La capacité d'expression des problèmes et de modélisation de l'infrastructure des outils proposés dans cette thèse représente une base solide à l'élaboration d'une suite logicielle permettant aux utilisateurs finaux et aux administrateurs de mieux maîtriser leurs environnements de travail. En effet, étant donné que toutes les décisions et gestions ne peuvent être déléguées aux intergiciels, la démocratisation des plateformes de grille ne peut se faire sans la mise en place d'outils conviviaux, compréhensibles et utilisables par les utilisateurs non-informaticiens. La cible que nous étudions actuellement concerne les organisations virtuelles inter-grilles. Dans un tel environnement, de nombreuses décisions sont laissées à la discrétion de l'utilisateur, qui n'est pas nécessairement informaticien. De plus, les administrateurs ne comprennent pas forcément toutes les subtilités relatives aux applications exécutées sur leurs plateformes. L'objectif est ici beaucoup plus ambitieux, puisqu'il ne s'agit plus de réconcilier la superstructure avec son infrastructure, mais de concilier les utilisateurs issus de tout domaine scientifique avec les plateformes de grille. NDS et GR-OLAP représentent deux premiers composants de cette suite logicielle, mais d'autres composants doivent être étudiés et développés : un gestionnaire personnel des ressources accordées à un utilisateur, couplé à NDS, permettra à chacun de décider des ressources à utiliser pour mener à bien une tâche ; couplé à un système de médiation, il permettra l'accès transparent aux ressources hétérogènes ; enfin, couplé à un système d'acompte, il permettra le suivi individuel des consommations et besoins. Cette suite logicielle permettra l'accompagnement des utilisateurs tout au long de la durée de vie des plateformes, depuis la conception jusqu'à la maintenance.

De plus, une telle suite logicielle implique que nous nous intéressions à la résolution des problèmes liés à la répartition des ressources *matérielles*. En particulier, nous voulons traiter le problème suivant : étant donnés une infrastructure, une superstructure, un budget et les tarifs de différents composants, définir les ajouts de composants matériels qui permettront d'obtenir les meilleures performances. La résolution de ce problème représente une aide cruciale pour les administrateurs puisqu'elle permettra de répondre à des questions telles que : vaut-il mieux changer les disques, les mémoires ou les cartes réseaux, et pour quels matériels ? La suite logique de ce problème est d'inverser le processus de résolution : plutôt que de définir la meilleure superstructure pour une infrastructure donnée, nous voulons pouvoir définir la meilleure infrastructure pour une superstructure donnée. Un exemple d'application cible est : étant donnés un réseau et une superstructure, définir les capacités des hôtes afin d'obtenir une plateforme équilibrée en ce qui concerne les coûts de communications et de calculs. Cette application implique que les points d'équilibres, identifiés dans cette thèse pour valider la pertinence des métriques composées, soient exploités directement dans la résolution des problèmes.

Finalement, nous envisageons d'intégrer à cette suite logicielle un outil d'évaluation des stratégies liées à la répartition des ressources. Cet outil sera basé sur la modélisation des ressources et de l'infrastructure matérielle proposée dans cette thèse. En effet, cette modélisation, ainsi que le service NDS, ne permettent pas seulement de résoudre les problèmes liés à la répartition des ressources, mais permettent également d'évaluer les performances d'une stratégie ou d'une solution donnée à un problème lié à la répartition des ressources. On pourra par exemple les utiliser pour tester des stratégies de sélection / composition, ou encore pour évaluer l'efficacité d'une solution de déploiement, en fonction des infrastructures matérielles cibles. Notre objectif n'est pas de concevoir un nouveau simulateur de grille ou réseau, car nous ne voulons pas mettre en place de simulation d'exécution, mais seulement d'évaluation de performances. La conception d'un tel outil impose que l'on étudie l'intégration des solutions, en plus des problèmes, dans notre service. De plus, nous envisageons de construire une base de données contenant les informations de surveillances de plusieurs plateformes, et de développer un

outil interactif permettant de modifier ou assembler ces informations pour construire des infrastructures virtuelles. Ceci permettra de faciliter l'utilisation de la suite logicielle en la découplant d'une infrastructure réelle. Les développeurs pourront alors tester leurs stratégies sur différentes infrastructures cibles, et les administrateurs pourront évaluer les gains d'une réorganisation de la superstructure comme tester l'impact de modifications de l'infrastructure.

15.3.2 Extension des capacités de résolution et exploitation des particularités des mesures de surveillances

Les perspectives plus fondamentales portent sur l'étude détaillée des différents champs d'application de la théorie des graphes afin d'y plonger les problèmes pragmatiques des superstructures. Nous envisageons de coupler NDS avec une librairie d'algorithmes de graphe afin d'augmenter sa capacité de résolution et d'établir une liste des algorithmes correspondant à des problèmes liés à la répartition des ressources. De plus, notre modélisation ouvre des perspectives pour le développement de nouveaux algorithmes grâce à la mise en évidence de nouveaux problèmes. En effet, la modélisation sous forme de $\mathcal{CNP}$-Graphes permet de visualiser la réalité physique des problèmes. Elle représente donc une aide cruciale dans le processus d'identification des caractéristiques des problèmes et de conception de méthodes de résolution adaptées. Le processus d'adaptation que nous avons suivi pour concevoir $MRKM$ et $FReDi$ est pertinent pour un vaste panel d'applications. Il sera donc reconduit pour, au moins, deux autres problèmes. Le premier concerne l'adaptation du problème des sacs à dos (*bin packing*) afin de déployer une superstructure. Deux approches seront étudiées : utiliser la totalité de la superstructure pour déployer un maximum de réplicas des composants de la superstructure, en d'autres termes remplir n sacs à dos avec autant de réplicas possibles de m ressources ; ou au contraire déployer une superstructure complète sur un minimum d'hôtes, en d'autres termes remplir un minimum de sacs à dos avec m ressources. Le deuxième problème est celui de l'édition de forêt (*forest editing problem*) qui a pour objectif de définir les opérations minimales permettant d'obtenir un graphe cible à partir d'un graphe initial. La résolution de problème sur un $\mathcal{CNP}$-Graphe permettra de définir les opérations minimales à effectuer sur une superstructure pour la réorganiser.

Une perspective également fondamentale concerne l'exploitation des imprécisions des informations de surveillance. En effet, ces informations sont issues de mesures, donc approximatives. Un facteur de précision est d'ailleurs généralement fourni par les outils de surveillance. Cette imprécision n'est actuellement pas exploitée dans la résolution des problèmes, seule la valeur la plus pertinente est exploitée. Or, labéliser les arcs des $\mathcal{CNP}$-Graphes, non pas avec cette unique valeur, mais avec un intervalle des valeurs possibles en fonction des facteurs de précision ouvre d'intéressantes perspectives algorithmiques, car elle permet d'obtenir plusieurs solutions équivalentes pour un même problème. Chacune de ces solutions pouvant être pertinente dans un contexte donné, cette approche pourra être adoptée pour les problèmes mettant en jeu plusieurs objectifs. Un exemple d'application cible est l'organisation d'un réseau pair à pair pour la diffusion de données. Ce problème possède deux facettes : d'un côté de minimiser la consommation des ressources de communication, ce qui correspond au calcul de l'arbre couvrant minimum du réseau pair à pair ; d'un autre côté, minimiser le temps de diffusion entre les pairs, ce qui revient à calculer les plus courts chemins. Dans une approche classique, l'arbre couvrant minimum est unique. Mais en tenant compte de l'imprécision des mesures, on peut obtenir plusieurs arbres couvrants minimums. On peut alors mettre en correspondance ces arbres avec les plus courts chemins pour définir l'arbre couvrant minimum en fonction du pair émetteur des données à diffuser. Une autre application cible concerne les services collaboratifs distribués, tels que le service de caches collaboratifs : d'un côté il s'agit de couvrir au mieux le réseau afin de servir les clients, ce

qui correspond à un problème des k-médians, d'un autre côté il faut minimiser le coût des communications entre les différentes instances du service. Résoudre les k-médians en tenant compte de la marge d'erreur permettra d'obtenir plusieurs solutions équivalentes, parmi lesquelles certaines minimiseront également le temps de communication entre les instances. À notre connaissance, l'utilisation d'intervalles afin de labéliser un graphe n'a jamais été utilisée dans ce contexte.

15.3.3 Adaptation de nos travaux pour les systèmes autonomes

Une dernière perspective concerne le domaine des systèmes autonomes (*autonomic computing*). Ces systèmes ont pour objectif de limiter au maximum les besoins d'administration et les interventions humaines sur la superstructure. Le défi principal consiste à doter les superstructures d'une capacité de réaction ou proaction automatique aux évènements, tels que les pannes (*self healing*) ou à l'apparition de nouveaux besoins. Or sa capacité d'adaptation et de résolution des problèmes de répartition, fait de notre approche un candidat pertinent pour gérer les ressources dans un système autonome. Nous étudierons les techniques permettant de détecter les pannes et autres problèmes de performances. Cette détection permettra de déclencher automatiquement des opérations de réorganisation de la superstructure en temps voulu afin d'assurer la continuité de l'exécution de la plateforme. Cette cible applicative nécessite aussi, dans une perspective de complète autonomie, le développement d'une méthode de sélection automatique des algorithmes les plus pertinents pour résoudre un problème donné en fonction des algorithmes disponibles et des caractéristiques des $\mathcal{CNP}$-Graphes. L'objectif final étant de pouvoir définir tous les composants d'une superstructure, puis de laisser le système se dimensionner et s'organiser de façon autonome en fonction des besoins et de l'état de l'infrastructure. Un problème majeur que nous devrons de plus étudier concerne la déclaration automatique des problèmes. À l'heure actuelle, notre approche implique qu'un utilisateur déclare le problème, même si cette déclaration est adaptable et incluse dans une application. Dans un système autonome, la superstructure doit pouvoir identifier et déclarer elle-même ses propres problèmes en complète autonomie.

Nous envisageons également d'étudier les possibilités de conception de services web de grille, adaptables aux performances (*performances aware web services*). En effet, dans les environnements de grilles orientées services, le développement est le plus souvent dirigé par les fonctionnalités, et les performances, sans être pour autant négligées, ne sont pas au cœur des processus de développement. Les travaux de cette thèse s'attachent à adapter la superstructure. Or, l'adaptation autonome de chacun des services est une étape importante dans la mise en place de systèmes autonomes. Nous voulons donc étudier la possibilité de développer des applications automatiquement adaptables en fonction des performances. De la même manière que les services s'adaptent aux terminaux (capacité d'affichage, de calcul, d'autonomie, etc.) dans les environnements pervasifs, on peut imaginer que certains services s'adaptent en fonction des performances, par exemple en décidant de compresser les données si le réseau est encombré mais les processeurs disponibles, ou au contraire en sollicitant plus d'instances lorsque le réseau est disponible mais les hôtes saturés. Cette approche est connexe à l'effort de définition du standard WSDM (*Web Service Distributed Management*) fourni par OASIS pour identifier et intégrer des métriques de surveillance des services. Ceci permettra non seulement de mieux supporter la dynamicité des plateformes, mais surtout de s'adapter à un large panel d'infrastructures et à leurs évolutions. NDS représente une base pertinente dans le sens où il est capable d'évaluer les coûts en fonction des ressources, mais il reste à étudier comment inclure ces informations directement dans le processus de développement et dans l'exécution des services, plutôt que seulement dans la résolution des problèmes de répartition.

Ces perspectives représentent chacune un petit pas de plus sur le chemin de la réconciliation des superstructures logicielles avec leurs infrastructures matérielles... Mais « *le chemin est long du projet à la chose* » [Molière].

This is the end...

Sixième partie

Annexes

$$CPU_s = \begin{pmatrix} 2193.5 & 2593.6 & 2391.4 & 2393.6 & 1995.1 & 1995.1 & 2192.5 & 2191.1 & 2009.3 & 2009.3 & 2193.8 & 2193.8 & 1994.1 & 1994.1 & 2192.7 & 2193.0 \end{pmatrix}$$

$$L = \begin{pmatrix}
000.00 & 000.09 & 010.16 & 010.16 & 004.34 & 004.34 & 011.21 & 011.21 & 011.21 & 011.21 & 016.75 & 016.75 & 016.74 & 016.74 & 017.43 & 017.43 \\
000.09 & 000.00 & 010.16 & 010.16 & 004.34 & 004.34 & 011.21 & 011.21 & 011.21 & 011.21 & 016.76 & 016.79 & 016.74 & 016.74 & 017.43 & 017.43 \\
010.17 & 010.16 & 000.00 & 000.11 & 009.12 & 009.12 & 012.58 & 012.58 & 012.58 & 012.58 & 007.24 & 007.23 & 007.20 & 007.20 & 007.89 & 007.89 \\
010.17 & 010.17 & 000.13 & 000.00 & 009.12 & 009.12 & 012.58 & 012.58 & 012.58 & 012.58 & 007.24 & 007.24 & 007.21 & 007.21 & 007.89 & 007.89 \\
004.35 & 004.35 & 009.12 & 009.12 & 000.00 & 000.12 & 008.85 & 008.84 & 008.85 & 008.84 & 020.25 & 020.28 & 020.19 & 020.19 & 015.51 & 015.51 \\
004.35 & 004.35 & 009.12 & 009.12 & 000.13 & 000.00 & 008.85 & 008.84 & 008.85 & 008.84 & 020.23 & 020.20 & 020.19 & 020.19 & 015.51 & 015.50 \\
011.22 & 011.21 & 012.58 & 012.58 & 008.85 & 008.85 & 000.00 & 000.10 & 000.10 & 000.12 & 019.18 & 019.17 & 019.16 & 019.16 & 019.85 & 019.85 \\
011.22 & 011.21 & 012.58 & 012.58 & 008.84 & 008.84 & 000.10 & 000.00 & 000.10 & 000.12 & 019.25 & 019.25 & 019.16 & 019.16 & 019.85 & 019.85 \\
011.22 & 011.21 & 012.58 & 012.58 & 008.85 & 008.85 & 000.10 & 000.10 & 000.00 & 000.12 & 019.18 & 019.17 & 019.16 & 019.16 & 019.85 & 019.85 \\
011.22 & 011.21 & 012.58 & 012.58 & 008.84 & 008.84 & 000.10 & 000.10 & 000.12 & 000.00 & 019.25 & 019.25 & 019.16 & 019.16 & 019.85 & 019.85 \\
016.81 & 016.82 & 007.20 & 007.20 & 020.31 & 020.32 & 019.17 & 019.17 & 019.19 & 019.19 & 000.00 & 000.19 & 000.20 & 000.20 & 014.57 & 014.57 \\
016.82 & 016.82 & 007.20 & 007.20 & 020.31 & 020.20 & 019.17 & 019.17 & 019.19 & 019.19 & 000.20 & 000.00 & 000.19 & 000.20 & 014.57 & 014.57 \\
016.75 & 016.74 & 007.20 & 007.21 & 020.19 & 020.19 & 019.16 & 019.16 & 019.16 & 019.16 & 000.12 & 000.11 & 000.00 & 000.11 & 014.47 & 014.47 \\
016.75 & 016.74 & 007.21 & 007.20 & 020.19 & 020.19 & 019.16 & 019.16 & 019.16 & 019.16 & 000.15 & 000.15 & 000.11 & 000.00 & 014.47 & 014.47 \\
017.43 & 017.43 & 007.89 & 007.89 & 015.51 & 015.52 & 019.85 & 019.84 & 019.85 & 019.85 & 014.55 & 014.56 & 014.47 & 014.47 & 000.00 & 000.10 \\
017.44 & 017.43 & 007.91 & 007.89 & 015.51 & 015.50 & 019.85 & 019.84 & 019.85 & 019.85 & 014.50 & 014.52 & 014.47 & 014.47 & 000.11 & 000.00
\end{pmatrix}$$

$$BW = \begin{pmatrix}
\infty & 815.51 & 020.92 & 022.04 & 047.34 & 048.51 & 018.85 & 019.01 & 019.11 & 019.25 & 013.45 & 013.44 & 011.59 & 011.59 & 011.14 & 011.14 \\
811.33 & \infty & 021.55 & 020.93 & 048.42 & 049.79 & 020.04 & 019.02 & 020.04 & 020.04 & 013.17 & 013.45 & 010.13 & 010.71 & 011.14 & 011.14 \\
021.51 & 021.41 & \infty & 813.34 & 023.19 & 023.19 & 016.98 & 016.98 & 016.98 & 016.98 & 029.23 & 029.17 & 022.29 & 026.78 & 024.52 & 024.52 \\
021.52 & 021.63 & 814.31 & \infty & 023.18 & 023.20 & 016.98 & 016.98 & 016.98 & 016.98 & 030.72 & 030.72 & 026.78 & 023.29 & 024.52 & 024.52 \\
049.70 & 048.37 & 023.22 & 024.34 & \infty & 771.96 & 024.48 & 024.55 & 024.48 & 024.55 & 011.12 & 011.12 & 009.54 & 009.56 & 012.44 & 012.44 \\
049.73 & 048.50 & 023.15 & 023.15 & 772.39 & \infty & 025.03 & 025.04 & 025.03 & 025.04 & 011.12 & 011.12 & 009.57 & 009.57 & 012.44 & 012.44 \\
019.01 & 019.42 & 016.97 & 016.98 & 024.48 & 025.03 & \infty & 805.62 & 805.51 & 803.45 & 011.79 & 011.79 & 008.22 & 009.30 & 009.80 & 009.80 \\
019.48 & 019.03 & 016.97 & 016.97 & 024.55 & 025.04 & 805.37 & \infty & 805.98 & 803.93 & 011.79 & 011.79 & 009.14 & 009.12 & 009.80 & 009.80 \\
019.01 & 019.42 & 016.97 & 016.98 & 024.48 & 025.03 & 805.62 & 805.51 & \infty & 803.45 & 011.79 & 011.79 & 008.22 & 009.30 & 009.80 & 009.80 \\
019.48 & 019.03 & 016.97 & 016.97 & 024.55 & 025.04 & 805.37 & 805.98 & 803.93 & \infty & 011.79 & 011.79 & 009.14 & 009.12 & 009.80 & 009.80 \\
015.09 & 015.09 & 026.60 & 026.38 & 012.47 & 012.48 & 009.65 & 009.64 & 009.64 & 009.64 & \infty & 717.26 & 712.47 & 712.22 & 012.18 & 012.18 \\
015.09 & 015.09 & 026.38 & 026.38 & 012.47 & 012.47 & 009.64 & 009.64 & 009.64 & 009.64 & 697.55 & \infty & 713.09 & 712.72 & 012.17 & 012.18 \\
012.77 & 012.78 & 029.39 & 029.39 & 010.55 & 010.56 & 011.18 & 011.18 & 011.18 & 011.18 & 741.76 & 737.20 & \infty & 872.42 & 013.41 & 013.41 \\
012.77 & 012.78 & 029.39 & 029.39 & 010.56 & 010.56 & 011.18 & 011.18 & 011.18 & 011.18 & 741.29 & 742.70 & 872.01 & \infty & 013.41 & 013.41 \\
012.28 & 012.28 & 026.92 & 026.92 & 013.71 & 013.71 & 010.80 & 010.80 & 010.80 & 010.80 & 014.73 & 015.52 & 011.68 & 012.97 & \infty & 870.75 \\
012.28 & 012.28 & 026.92 & 026.92 & 013.71 & 013.71 & 010.80 & 010.80 & 010.80 & 010.80 & 015.53 & 014.73 & 012.06 & 012.05 & 870.19 & \infty
\end{pmatrix}$$

TAB. 1 – Observations des métriques de surveillance fournies par NWS

Service de caches collaboratifs (type communications intensives)

$$\begin{pmatrix} 0.000 & 0.986 & 37.216 & 37.194 & 16.106 & 16.096 & 42.105 & 41.099 & 42.105 & 41.099 & 53.057 & 53.057 & 62.682 & 62.676 & 65.197 & 65.196 \end{pmatrix}$$

Service de fouille de données (type calculs intensifs)

$$\begin{pmatrix} 2.136 & 1.808 & 2.042 & 2.038 & 2.385 & 2.384 & 2.229 & 2.230 & 2.423 & 2.423 & 2.268 & 2.268 & 2.492 & 2.492 & 2.285 & 2.285 \end{pmatrix}$$

Entrepôt de données distribué (type équilibré)

$$\begin{pmatrix} 0.171 & 0.165 & 0.942 & 0.941 & 0.527 & 0.527 & 1.058 & 1.038 & 1.074 & 1.053 & 1.299 & 1.299 & 1.509 & 1.508 & 1.545 & 1.545 \end{pmatrix}$$

TAB. 2 – Valeurs des évaluations réelles des distances des $\mathcal{CNP}$-Graphes pour le problème de la sélection

TAB. 3 – $QD \in M_{\mathcal{H}}^{\mathbb{R}}$: Nombre de requêtes par client sur une heure

$node$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
QD_{node}	1	7	22	29	0	10	54	62	25	25	35	44	77	80	52	53

$$d = \begin{pmatrix}
000.00 & 000.00 & 000.07 & 000.07 & 000.03 & 000.03 & 000.08 & 000.08 & 000.08 & 000.08 & 000.10 & 000.10 & 000.12 & 000.12 & 000.12 & 000.12 \\
000.01 & 000.00 & 000.53 & 000.52 & 000.23 & 000.23 & 000.58 & 000.60 & 000.58 & 000.60 & 000.75 & 000.75 & 000.89 & 000.89 & 000.92 & 000.92 \\
001.73 & 001.68 & 000.00 & 000.04 & 001.56 & 001.56 & 002.13 & 002.13 & 002.13 & 002.13 & 001.36 & 001.37 & 001.23 & 001.23 & 001.34 & 001.34 \\
002.17 & 002.29 & 000.06 & 000.00 & 001.97 & 002.07 & 002.82 & 002.82 & 002.82 & 002.82 & 001.82 & 001.82 & 001.63 & 001.63 & 001.78 & 001.78 \\
000.02 & 000.02 & 000.03 & 000.03 & 000.00 & 000.00 & 000.03 & 000.03 & 000.03 & 000.03 & 000.06 & 000.06 & 000.08 & 000.08 & 000.06 & 000.06 \\
000.33 & 000.32 & 000.69 & 000.69 & 000.02 & 000.00 & 000.64 & 000.64 & 000.64 & 000.64 & 001.29 & 001.29 & 001.52 & 001.52 & 001.17 & 001.17 \\
004.81 & 004.52 & 005.34 & 005.34 & 003.70 & 003.62 & 000.00 & 000.11 & 000.11 & 000.11 & 009.39 & 009.40 & 008.11 & 008.11 & 008.39 & 008.39 \\
005.49 & 005.49 & 006.15 & 006.15 & 004.25 & 004.17 & 000.13 & 000.00 & 000.13 & 000.13 & 010.83 & 010.83 & 009.34 & 009.34 & 009.67 & 009.67 \\
002.19 & 002.09 & 002.46 & 002.46 & 001.71 & 001.67 & 000.05 & 000.05 & 000.00 & 000.05 & 004.34 & 004.34 & 003.74 & 003.74 & 003.87 & 003.87 \\
002.22 & 002.13 & 002.52 & 002.52 & 001.74 & 001.71 & 000.05 & 000.05 & 000.05 & 000.00 & 004.43 & 004.43 & 003.82 & 003.82 & 003.96 & 003.96 \\
004.35 & 004.45 & 002.00 & 001.91 & 005.27 & 005.26 & 004.97 & 004.97 & 004.97 & 004.97 & 000.00 & 000.08 & 000.08 & 000.08 & 003.98 & 003.77 \\
005.46 & 005.45 & 002.51 & 002.39 & 006.60 & 006.59 & 006.22 & 006.22 & 006.22 & 006.22 & 000.10 & 000.00 & 000.10 & 000.10 & 004.73 & 004.98 \\
011.13 & 012.73 & 005.79 & 004.82 & 013.53 & 013.48 & 015.69 & 014.12 & 015.69 & 014.12 & 000.18 & 000.18 & 000.00 & 000.15 & 011.05 & 010.70 \\
011.55 & 012.50 & 005.00 & 005.75 & 014.00 & 013.98 & 014.39 & 014.68 & 014.39 & 014.68 & 000.19 & 000.19 & 000.15 & 000.00 & 010.32 & 011.11 \\
007.78 & 007.78 & 003.54 & 003.54 & 006.97 & 006.97 & 008.85 & 008.85 & 008.85 & 008.85 & 007.12 & 007.12 & 006.46 & 006.46 & 000.00 & 000.10 \\
007.96 & 007.96 & 003.62 & 003.62 & 007.13 & 007.13 & 009.05 & 009.05 & 009.05 & 009.05 & 007.28 & 007.28 & 006.61 & 006.61 & 000.10 & 000.00
\end{pmatrix}$$

TAB. 4 – Valeurs des évaluations réelles des distances des $\mathcal{CNP}$-Graphes pour le problème du déploiement de la ressource de type communications intensives

k	NDS(1)		NDS(2)		NDS(3)		NDS(4)		Meilleur		Médian		Pire	
	critere	précision	*critere*	précision	*critere*	précision	*critere*	précision	*critere*	précision	*critere*	précision	*critere*	précision
1	23.7	1.01	23.7	1.01	23.7	1.01	23.7	1.01	23.5	1.00	42.4	1.80	45.4	1.93
	sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-11.lyon		parasol48.rennes		gdx0116.orsay	
2	16.5	1.26	13.1	1.00	16.5	1.26	13.1	1.00	13.1	1.00	19.2	1.47	42.5	3.25
	parasol48.rennes		node-65.sophia pa-		parasol48.rennes		node-65.sophia pa-		node-65.sophia pa-		node-65.sophia		paravent20.rennes	
	sagittaire-49.lyon		rasol48.rennes		sagittaire-49.lyon		rasol48.rennes		rasol48.rennes		node-62.lille		parasol23.rennes	
3	2.4	1.00	2.4	1.00	2.4	1.00	2.4	1.00	2.4	1.00	13.1	5.43	38.4	15.97
	node-42.toulouse		node-51.toulouse		node-42.toulouse		node-51.toulouse		node-51.toulouse		node-65.sophia		paravent20.rennes	
	node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-15.sophia		helios01.sophia		parasol23.rennes	
	parasol48.rennes		parasol48.rennes		parasol48.rennes		parasol48.rennes		parasol48.rennes		paravent20.rennes		gdx0116.orsay	
4	2.0	1.74	1.2	1.00	2.0	1.74	1.2	1.00	1.1	1.00	7.0	6.08	12.6	11.01
	node-42.toulouse		node-51.toulouse		node-42.toulouse		node-51.toulouse		node-51.toulouse		node-65.sophia		node-65.sophia	
	node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		parasol23.rennes		paravent05.rennes	
	parasol48.rennes		parasol48.rennes		parasol48.rennes		parasol48.rennes		parasol48.rennes		gdx0116.orsay		parasol48.rennes	
	node-62.lille		sagittaire-49.lyon		node-62.lille		sagittaire-49.lyon		sagittaire-11.lyon		sagittaire-11.lyon		node-62.lille	
5	0.7	1.05	0.7	1.00	0.7	1.05	0.7	1.00	0.7	1.00	1.1	1.52	2.0	2.87
	node-42.toulouse		node-51.toulouse		node-42.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse	
	node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		helios48.sophia	
	parasol48.rennes		parasol48.rennes		parasol48.rennes		parasol48.rennes		parasol48.rennes		helios48.sophia		paravent20.rennes	
	sagittaire-49.lyon		gdx0140.orsay		sagittaire-49.lyon		gdx0140.orsay		gdx0140.orsay		parasol23.rennes		parasol48.rennes	
	node-62.lille		sagittaire-49.lyon		node-62.lille		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-11.lyon		node-62.lille	
6	0.6	1.07	0.6	1.05	0.6	1.07	0.6	1.05	0.6	1.00	0.7	1.20	0.8	1.35
	node-42.toulouse		node-51.toulouse		node-42.toulouse		node-51.toulouse		node-51.toulouse		node-42.toulouse		node-51.toulouse	
	node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-15.sophia		helios01.sophia	
	parasol48.rennes		parasol48.rennes		parasol48.rennes		parasol48.rennes		node-15.sophia		paravent20.rennes		paravent20.rennes	
	gdx0140.orsay		gdx0140.orsay		gdx0140.orsay		gdx0140.orsay		parasol48.rennes		parasol23.rennes		gdx0116.orsay	
	sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		gdx0140.orsay		gdx0140.orsay		sagittaire-49.lyon	
	node-11.lille		node-62.lille		node-11.lille		node-62.lille		sagittaire-49.lyon		sagittaire-49.lyon		node-62.lille	
7	0.5	1.17	0.5	1.00	0.5	1.17	0.5	1.00	0.5	1.00	0.6	1.24	0.6	1.31
	node-42.toulouse		node-51.toulouse		node-42.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse	
	node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-15.sophia	
	helios01.sophia		node-15.sophia		helios01.sophia		node-15.sophia		node-15.sophia		parasol48.rennes		helios48.sophia	
	parasol48.rennes		parasol48.rennes		parasol48.rennes		parasol48.rennes		parasol48.rennes		gdx0140.orsay		helios01.sophia	
	gdx0140.orsay		gdx0140.orsay		gdx0140.orsay		gdx0140.orsay		gdx0140.orsay		sagittaire-49.lyon		parasol48.rennes	
	sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-11.lyon		gdx0116.orsay	
	node-11.lille		node-62.lille		node-11.lille		node-62.lille		node-62.lille		node-11.lille		sagittaire-11.lyon	

TAB. 5 – Résultats de l'exécution de l'algorithme trivial de résolution du problème des k-médians sur le $\mathcal{CNP}$-Graphe de la ressource de type communications intensives

$$d = \begin{pmatrix}
000.09 & 000.08 & 000.08 & 000.08 & 000.10 & 000.10 & 000.09 & 000.09 & 000.10 & 000.10 & 000.09 & 000.09 & 000.10 & 000.10 & 000.09 & 000.09 \\
000.68 & 000.57 & 000.62 & 000.62 & 000.75 & 000.75 & 000.68 & 000.68 & 000.74 & 000.74 & 000.68 & 000.68 & 000.75 & 000.75 & 000.68 & 000.68 \\
002.17 & 001.84 & 001.98 & 001.98 & 002.38 & 002.38 & 002.17 & 002.17 & 002.37 & 002.37 & 002.17 & 002.17 & 002.38 & 002.38 & 002.17 & 002.17 \\
002.87 & 002.43 & 002.63 & 002.62 & 003.16 & 003.16 & 002.88 & 002.88 & 003.14 & 003.14 & 002.87 & 002.87 & 003.16 & 003.16 & 002.87 & 002.87 \\
000.05 & 000.04 & 000.04 & 000.04 & 000.05 & 000.05 & 000.05 & 000.05 & 000.05 & 000.05 & 000.05 & 000.05 & 000.05 & 000.05 & 000.05 & 000.05 \\
000.96 & 000.81 & 000.88 & 000.88 & 001.06 & 001.06 & 000.96 & 000.96 & 001.05 & 001.05 & 000.97 & 000.97 & 001.06 & 001.06 & 000.97 & 000.97 \\
005.44 & 004.60 & 004.99 & 004.99 & 005.97 & 005.97 & 005.42 & 005.42 & 005.91 & 005.91 & 005.45 & 005.45 & 006.01 & 006.00 & 005.46 & 005.46 \\
006.27 & 005.31 & 005.75 & 005.75 & 006.88 & 006.88 & 006.24 & 006.25 & 006.81 & 006.81 & 006.28 & 006.28 & 006.92 & 006.92 & 006.30 & 006.29 \\
002.51 & 002.13 & 002.31 & 002.30 & 002.76 & 002.76 & 002.50 & 002.50 & 002.73 & 002.73 & 002.52 & 002.52 & 002.77 & 002.77 & 002.52 & 002.52 \\
002.57 & 002.17 & 002.36 & 002.35 & 002.82 & 002.82 & 002.56 & 002.56 & 002.79 & 002.79 & 002.57 & 002.57 & 002.83 & 002.83 & 002.58 & 002.58 \\
003.52 & 002.98 & 003.22 & 003.22 & 003.87 & 003.87 & 003.53 & 003.53 & 003.85 & 003.85 & 003.50 & 003.50 & 003.85 & 003.85 & 003.52 & 003.52 \\
004.41 & 003.73 & 004.03 & 004.03 & 004.85 & 004.85 & 004.42 & 004.43 & 004.82 & 004.82 & 004.38 & 004.38 & 004.82 & 004.82 & 004.41 & 004.41 \\
007.76 & 006.57 & 007.10 & 007.09 & 008.54 & 008.54 & 007.77 & 007.78 & 008.48 & 008.48 & 007.71 & 007.71 & 008.48 & 008.48 & 007.76 & 007.76 \\
008.05 & 006.82 & 007.36 & 007.35 & 008.86 & 008.86 & 008.07 & 008.07 & 008.80 & 008.80 & 008.00 & 008.00 & 008.80 & 008.80 & 008.05 & 008.05 \\
005.22 & 004.42 & 004.77 & 004.77 & 005.73 & 005.73 & 005.23 & 005.23 & 005.70 & 005.70 & 005.21 & 005.21 & 005.74 & 005.73 & 005.18 & 005.18 \\
005.34 & 004.52 & 004.88 & 004.87 & 005.86 & 005.86 & 005.35 & 005.35 & 005.83 & 005.83 & 005.33 & 005.33 & 005.87 & 005.87 & 005.30 & 005.30
\end{pmatrix}$$

TAB. 6 – Valeurs des évaluations réelles des distances des $\mathcal{CNP}$-Graphes pour le problème du déploiement de la ressource de type calculs intensifs

k	NDS(1)		NDS(2)		NDS(3)		NDS(4)		Meilleur		Médian		Pire	
	critere	précision	*critere*	précision	*critere*	précision	*critere*	précision	*critere*	précision	*critere*	précision	*critere*	précision
1	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	59.0	1.17	65.2	1.29
	node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		parasol23.rennes		gdx0140.orsay	
2	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	57.6	1.14	64.7	1.28
	node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		helios48.sophia		paravent20.rennes	
	node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		gdx0116.orsay		gdx0140.orsay	
3	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	53.1	1.05	64.4	1.28
	node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		parasol23.rennes		node-15.sophia	
	node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		sagittaire-11.lyon		paravent20.rennes	
	node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-11.lille		gdx0140.orsay	
4	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	54.1	1.07
	node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse	
	node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		gdx0140.orsay		paravent20.rennes	
	node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		sagittaire-49.lyon		gdx0116.orsay	
	node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		sagittaire-49.lyon	
5	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00
	node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		paravent05.rennes	
	node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		helios01.sophia		parasol23.rennes	
	node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		parasol48.rennes		gdx0116.orsay	
	node-15.sophia		node-15.sophia		node-15.sophia		node-15.sophia		node-15.sophia		parasol23.rennes		sagittaire-11.lyon	
	node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille	
6	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00
	node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-15.sophia	
	node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		helios01.sophia	
	node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		paravent05.rennes	
	node-15.sophia		node-15.sophia		node-15.sophia		node-15.sophia		node-15.sophia		node-15.sophia		parasol23.rennes	
	helios48.sophia		helios48.sophia		helios48.sophia		helios48.sophia		helios48.sophia		parasol48.rennes		sagittaire-49.lyon	
	node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille	
7	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00	50.4	1.00
	node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse	
	node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		node-42.toulouse		node-65.sophia		node-42.toulouse	
	node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-65.sophia		node-15.sophia		helios48.sophia	
	node-15.sophia		node-15.sophia		node-15.sophia		node-15.sophia		node-15.sophia		paravent20.rennes		paravent20.rennes	
	helios48.sophia		helios48.sophia		helios48.sophia		helios48.sophia		helios48.sophia		paravent05.rennes		parasol48.rennes	
	helios01.sophia		helios01.sophia		helios01.sophia		helios01.sophia		helios01.sophia		gdx0116.orsay		sagittaire-49.lyon	
	node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille	

TAB. 7 – Résultats de l'exécution de l'algorithme trivial de résolution du problème des k-médians sur le $\mathcal{CNP}$-Graphe de la ressource de type calculs intensifs

$$d = \begin{pmatrix}
000.06 & 000.05 & 000.09 & 000.09 & 000.08 & 000.08 & 000.10 & 000.10 & 000.11 & 000.10 & 000.11 & 000.11 & 000.12 & 000.12 & 000.12 & 000.12 \\
000.46 & 000.39 & 000.68 & 000.67 & 000.62 & 000.62 & 000.74 & 000.75 & 000.78 & 000.79 & 000.82 & 000.82 & 000.93 & 000.93 & 000.91 & 000.91 \\
002.30 & 002.05 & 001.34 & 001.36 & 002.36 & 002.36 & 002.49 & 002.50 & 002.63 & 002.63 & 002.12 & 002.12 & 002.20 & 002.20 & 002.11 & 002.11 \\
002.99 & 002.75 & 001.80 & 001.77 & 003.08 & 003.13 & 003.31 & 003.31 & 003.48 & 003.48 & 002.82 & 002.82 & 002.92 & 002.92 & 002.80 & 002.80 \\
000.04 & 000.04 & 000.05 & 000.05 & 000.04 & 000.04 & 000.05 & 000.05 & 000.05 & 000.05 & 000.06 & 000.06 & 000.07 & 000.07 & 000.06 & 000.06 \\
000.81 & 000.71 & 000.93 & 000.93 & 000.72 & 000.71 & 000.96 & 000.96 & 001.02 & 001.02 & 001.27 & 001.27 & 001.45 & 001.45 & 001.22 & 001.22 \\
005.99 & 005.29 & 005.95 & 005.94 & 005.82 & 005.78 & 003.66 & 003.72 & 004.05 & 004.05 & 008.22 & 008.22 & 007.96 & 007.96 & 007.73 & 007.73 \\
006.88 & 006.23 & 006.85 & 006.85 & 006.70 & 006.66 & 004.28 & 004.22 & 004.66 & 004.66 & 009.47 & 009.47 & 009.17 & 009.17 & 008.91 & 008.91 \\
002.75 & 002.44 & 002.75 & 002.74 & 002.69 & 002.67 & 001.71 & 001.72 & 001.84 & 001.87 & 003.79 & 003.80 & 003.67 & 003.67 & 003.57 & 003.57 \\
002.80 & 002.50 & 002.81 & 002.80 & 002.74 & 002.73 & 001.75 & 001.75 & 001.91 & 001.88 & 003.88 & 003.88 & 003.76 & 003.76 & 003.65 & 003.65 \\
004.48 & 004.16 & 003.14 & 003.09 & 005.16 & 005.16 & 004.78 & 004.78 & 004.99 & 004.99 & 002.36 & 002.40 & 002.64 & 002.64 & 004.30 & 004.20 \\
005.61 & 005.15 & 003.94 & 003.87 & 006.46 & 006.46 & 005.98 & 005.99 & 006.25 & 006.25 & 003.01 & 002.96 & 003.31 & 003.31 & 005.26 & 005.38 \\
010.62 & 010.59 & 007.59 & 007.11 & 012.29 & 012.27 & 012.83 & 012.07 & 013.30 & 012.54 & 005.30 & 005.30 & 005.73 & 005.80 & 010.58 & 010.40 \\
011.01 & 010.64 & 007.38 & 007.74 & 012.74 & 012.73 & 012.39 & 012.54 & 012.88 & 013.03 & 005.49 & 005.49 & 006.02 & 005.95 & 010.42 & 010.80 \\
007.28 & 006.74 & 004.93 & 004.93 & 007.23 & 007.23 & 007.80 & 007.80 & 008.12 & 008.12 & 006.96 & 006.96 & 006.99 & 006.99 & 003.50 & 003.55 \\
007.45 & 006.89 & 005.04 & 005.04 & 007.40 & 007.40 & 007.98 & 007.98 & 008.31 & 008.31 & 007.12 & 007.12 & 007.15 & 007.15 & 003.63 & 003.58
\end{pmatrix}$$

TAB. 8 – Valeurs des évaluations réelles des distances des $\mathcal{CNP}$-Graphes pour le problème du déploiement de la ressource de type versatile

k	NDS(1)		NDS(2)		NDS(3)		NDS(4)		Meilleur		Médian		Pire	
	critere	précision	*critere*	précision	*critere*	précision	*critere*	précision	*critere*	précision	*critere*	précision	*critere*	précision
1	55.0	1.16	48.0	1.02	48.0	1.02	48.0	1.02	47.2	1.00	56.8	1.20	65.8	1.39
	node-62.lille		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-11.lyon		node-42.toulouse		gdx0140.orsay	
2	46.6	1.06	46.6	1.06	46.6	1.06	45.4	1.03	44.1	1.00	48.7	1.10	62.4	1.41
	sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		parasol48.rennes		helios48.sophia		node-15.sophia pa-		paravent20.rennes	
	node-62.lille		node-62.lille		node-62.lille		sagittaire-49.lyon		sagittaire-11.lyon		rasol23.rennes		gdx0140.orsay	
3	43.6	1.08	41.3	1.02	45.2	1.12	40.4	1.00	40.4	1.00	45.4	1.12	59.6	1.48
	helios01.sophia		node-51.toulouse		parasol23.rennes		node-51.toulouse		node-42.toulouse		node-42.toulouse		paravent05.rennes	
	sagittaire-49.lyon		helios48.sophia		sagittaire-49.lyon		helios48.sophia		helios48.sophia		gdx0116.orsay		parasol48.rennes	
	node-62.lille		node-62.lille		node-62.lille		parasol23.rennes		parasol23.rennes		sagittaire-49.lyon		gdx0140.orsay	
4	41.0	1.04	40.7	1.03	42.1	1.07	39.5	1.00	39.4	1.00	42.2	1.07	44.9	1.14
	node-51.toulouse		node-51.toulouse		helios01.sophia		node-51.toulouse		node-42.toulouse		node-51.toulouse		node-65.sophia	
	helios01.sophia		helios48.sophia		parasol23.rennes		helios48.sophia		helios48.sophia		node-65.sophia		helios01.sophia	
	sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		parasol48.rennes		parasol23.rennes		parasol23.rennes		gdx0140.orsay	
	node-62.lille		node-62.lille		node-62.lille		sagittaire-49.lyon		sagittaire-11.lyon		gdx0140.orsay		sagittaire-49.lyon	
5	39.5	1.01	39.2	1.00	39.5	1.01	39.2	1.00	39.2	1.00	40.3	1.03	41.6	1.06
	node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-42.toulouse		node-51.toulouse		node-15.sophia	
	helios01.sophia		helios48.sophia		helios01.sophia		helios48.sophia		helios48.sophia		helios48.sophia		helios48.sophia	
	parasol23.rennes		parasol23.rennes		parasol23.rennes		parasol48.rennes		parasol23.rennes		helios01.sophia		parasol23.rennes	
	sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-11.lyon		parasol23.rennes		sagittaire-11.lyon	
	node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		gdx0116.orsay		node-62.lille	
6	39.2	1.00	39.2	1.00	39.2	1.00	39.2	1.00	39.1	1.00	39.4	1.01	40.2	1.03
	node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-42.toulouse		node-42.toulouse		node-51.toulouse	
	helios48.sophia		helios48.sophia		helios48.sophia		helios48.sophia		helios48.sophia		node-65.sophia		node-15.sophia	
	helios01.sophia		parasol48.rennes		helios01.sophia		parasol48.rennes		parasol48.rennes		helios48.sophia		helios48.sophia	
	parasol23.rennes		parasol23.rennes		parasol23.rennes		parasol23.rennes		parasol23.rennes		parasol23.rennes		paravent20.rennes	
	sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-11.lyon		sagittaire-49.lyon		parasol23.rennes	
	node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		sagittaire-11.lyon		node-11.lille	
7	39.2	1.00	39.1	1.00	39.2	1.00	39.1	1.00	39.1	1.00	39.3	1.01	39.4	1.01
	node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-51.toulouse		node-42.toulouse	
	node-42.toulouse		node-42.toulouse		helios48.sophia		node-42.toulouse		node-42.toulouse		helios48.sophia		helios01.sophia	
	helios48.sophia		helios48.sophia		helios01.sophia		helios48.sophia		helios48.sophia		parasol48.rennes		parasol48.rennes	
	helios01.sophia		parasol48.rennes		parasol48.rennes		parasol48.rennes		parasol48.rennes		gdx0116.orsay		parasol23.rennes	
	parasol23.rennes		parasol23.rennes		parasol23.rennes		parasol23.rennes		parasol23.rennes		sagittaire-49.lyon		gdx0140.orsay	
	sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-49.lyon		sagittaire-11.lyon		sagittaire-11.lyon		sagittaire-11.lyon	
	node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-62.lille		node-11.lille		node-62.lille	

TAB. 9 – Résultats de l'exécution de l'algorithme trivial de résolution du problème des k-médians sur le $\mathcal{CNP}$-Graphe de la ressource de type versatile

Table des figures

Liste des tableaux

Publications

[]

Revue internationale

[1] Jean-Marc Pierson, Julien Gossa, Pascal Wehrle, Yonny Cardenas, El Samad Mahmoud, Cahon Sebastien, Lionel Brunie, Clarisse Dhaenens, Hameurlain Abdel kader, Nouredine Melab, Maryvonne Miquel, Franck Morvan, Talbi El gazali, and Anne Tchounikine. Ggm efficient navigation and mining in distributed geno-medical data. *IEEE Transactions on NanoBioscience*, May 2007.
(Présentation de la superstructure GGM)

[2] Mahmoud El Samad, Julien Gossa, Franck Morvan, Abdelkader Hameurlain, Jean-Marc Pierson, and Lionel Brunie. Monitoring service for large scale dynamic query optimization in grid environment. *International Journal of Web and Grid Services (IJWGS)*, 2007.
(Présentation des résultats de l'intégration de NDS avec le service de requêtes optimisées)

Chapitres d'ouvrages

[3] Julien Gossa and Sandro Bimonte. Gr-olap : On line analytical processing of grid monitoring information, August 2007. Encyclopedia of Database Technologies and Applications, Second Edition, Eds. Laura C. Rivero, Jorge Horacio Doorn,Viviana E. Ferraggine. IDEA Group Inc. (Présentation de GR-OLAP)

[4] Julien Gossa and Jean-Marc Pierson. NDS : Network Distance Service. In *Extraction et Gestion Parallèles Distribuées de Connaissances (EGPDC 06)*, January 2006.
(Présentation des spécifications de NDS)

Conférences internationales

[5] Julien Gossa, Jean-Marc Pierson, and Lionel Brunie. Adaptable Distance-Based Decision-Making Support in Dynamic Cross-Grid Environment. In LNCS, editor, *The 13th International Euro-Par Conference European Conference on Parallel and Distributed Computing*, 2007. (Présentation de l'environnement inter-grille et de ses besoins en matière de gestion des ressources)

[6] Julien Gossa, Jean-Marc Pierson, and Lionel Brunie. Fredi : Flexible replicas displacer. In IEEE CS Press, editor, *5th International Conference on Networking*, April 2006. (Présentation de FReDi)

[7] Julien Gossa, Jean-Marc Pierson, and Lionel Brunie. Adapted distance-based decision-making support, a mandatory tool to make the grid pervasive. a.k.a. When the Grid becomes pervasive, even Mith Bucchanon needs decision-making support In IEEE, editor, *IEEE International Conference on Pervasive Services (ICPS'07)*, July 2007. (Présentation des grilles pervasives et de leurs besoins en terme de gestion de ressources)

[8] Julien Gossa. End-to-end distance computation in grid environment by NDS, the Network Distance Service. In IEEE, editor, *4th European Conference on Universal Multiservice Networks (ECUMN'07)*, February 2007.
(Présentation de NDS et de résultats expérimentaux préliminaires)

Atelier international

[9] Julien Gossa. Evaluation of network distances properties by nds, the network distance service. In Create-Net IEEE, editor, *GridNets'06*, October 2006.
(Présentation de la satisfaction des propriété euclidienne dans un réseau IP)

Conférences nationales

[10] Julien Gossa, Jean-Marc Pierson, and Lionel Brunie. (dé)placement dynamique de réplicas dans un environnement pervasif. In *1ere Journées Francophones Mobilité et Ubiquité, UbiMob'04*, 2004.
(Présentation préliminaire de FReDi)

Soumissions en cours d'évaluation :

[11] Julien Gossa, Jean-Marc Pierson, and Lionel Brunie. A comprehensive framework for decision-making support in grid and soa environment putting weights on computer network graphs. *Future Generation Computing Systems (FGCS)*, 2007. soumission. (Digest de cette thèse)

Bibliographie

[Antoniu 04] Gabriel Antoniu, Luc Bougé, Mathieu Jan & Sébastien Monnet. *Large-Scale Deployment in P2P Experiments Using the JXTA Distributed Framework*. In Marco Danelutto, Marco Vanneschi & Domenico Laforenza, editeurs, Euro-Par, volume 3149 of *Lecture Notes in Computer Science*, pages 1038–1047. Springer, 2004.

[Archer 01] Aaron Archer. *Two O (log* k)-Approximation Algorithms for the Asymmetric k-Center Problem*. In Proceedings of the 8th International IPCO Conference on Integer Programming and Combinatorial Optimization, pages 1–14, London, UK, 2001. Springer-Verlag.

[Bartal 01] Yair Bartal, Moses Charikar & Danny Raz. *Approximating min-sum k-clustering in metric spaces*. In STOC '01 : Proceedings of the thirty-third annual ACM symposium on Theory of computing, pages 11–20, New York, NY, USA, 2001. ACM Press.

[Berhe 06] Girma Berhe. *Accès et adaptation de contenus dans les systèmes d'information pervasifs*. PhD thesis, LIRIS, INSA Lyon, France, 2006.

[Bimonte 05] S. Bimonte, A. Tchounikine & M. Miquel. *Towards a spatial multidimensional model*. In DOLAP '05 : Proceedings of the 8th ACM international workshop on Data warehousing and OLAP, pages 39–46, New York, NY, USA, 2005. ACM Press.

[Borodin 98] Allan Borodin & Ran El-Yaniv. Online computation and competitive analysis. Cambridge University Press, 1998.

[Bouchenak 06] Sara Bouchenak, Noel De Palma, Daniel Hagimont & Christophe Taton. *Autonomic Management of Clustered Applications*. In CLUSTER. IEEE, 2006.

[Cappello 05] Franck Cappello, Eddy Caron, Michel J. Daydé, Frédéric Desprez, Yvon Jégou, Pascale Vicat-Blanc Primet, Emmanuel Jeannot, Stéphane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Benjamin Quétier & Olivier Richard. *Grid'5000 : a large scale and highly reconfigurable grid experimental testbed*. In GRID, pages 99–106. IEEE, 2005.

[Cardenas 06] Yonny Cardenas, Jean-Marc Pierson & Lionel Brunie. *Temporal Storage Space for Grids*. In Michael Gerndt & Dieter Kranzlmüller, editeurs, HPCC, volume 4208 of *Lecture Notes in Computer Science*, pages 803–812. Springer, 2006.

[Caron 06a] Eddy Caron, Pushpinder Kaur Chouhan & Holly Dail. *GoDIET : A Deployment Tool for Distributed Middleware on Grid'5000*. In IEEE, editeur, EXPGRID workshop. Experimental Grid Testbeds for the Assessment of Large-Scale Distributed Apllications and Tools. In conjunction with HPDC-15., pages 1–8, Paris, France, June 19th 2006.

[Caron 06b] Eddy Caron & Frédéric Desprez. *DIET : A Scalable Toolbox to Build Network Enabled Servers on the Grid*. International Journal of High Performance Computing Applications, vol. 20, no. 3, pages 335–352, 2006.

[Cooke 04] A. Cooke, A. Gray & et al. *The relational grid monitoring architecture : Mediating information about the grid*. 2004.

[Coquil 06] David Coquil. *Conception et mise en oeuvre de proxies sémantiques et coopératifs*. PhD thesis, LIRIS, INSA Lyon, France, 2006.

[Czajkowski 01a] Karl Czajkowski, Carl Kesselman, Steven Fitzgerald & Ian Foster. *Grid Information Services for Distributed Resource Sharing*. hpdc, vol. 00, page 0181, 2001.

[Czajkowski 01b] Karl Czajkowski, Carl Kesselman, Steven Fitzgerald & Ian T. Foster. *Grid Information Services for Distributed Resource Sharing*. In HPDC, pages 181–194. IEEE Computer Society, 2001.

[Dabek 04] Frank Dabek, Russ Cox, M. Frans Kaashoek & Robert Morris. *Vivaldi : a decentralized network coordinate system*. In Raj Yavatkar, Ellen W. Zegura & Jennifer Rexford, editeurs, SIGCOMM, pages 15–26. ACM, 2004.

[de Assuncao 06] Marcos Dias de Assuncao & Rajkumar Buyya. *A Case for the World Wide Grid*. Rapport technique GRIDS-TR-2006-1, GRIDS Laboratory, Mel bourne University, Australia, February 2006.

[Faerman 99] Marcio Faerman, Alan Su, Rich Wolski & Francine Berman. *Adaptive Performance Prediction for Distributed Data-Intensive Applications*. In ACM/IEEE SC99 Conference on High Performance Networking and Computing, Portland, OR, USA, 1999.

[Fedak 01] Gilles Fedak, Cecile Germain, Vincent Neri & Franck Cappello. *XtremWeb : A Generic Global Computing System*. In CCGRID '01 : Proceedings of the 1st International Symposium on Cluster Computing and the Grid, page 582, Washington, DC, USA, 2001. IEEE Computer Society.

[Foster 99] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt & A. Roy. *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation*. In Proceedings of the International Workshop on Quality of Service, 1999.

[Foster 01] Ian Foster, Carl Kesselman & Steven Tuecke. *The Anatomy of the Grid : Enabling Scalable Virtual Organizations*. The International Journal of High Performance Computing Applications, vol. 15, no. 3, pages 200–222, Fall 2001.

[Fu 00] Xiadong Fu, Weisong Shi, Anatoly Akkerman & Vijay Karamcheti. *CANS : Composable, Adaptive Network Services Infrastructure*. Rapport technique, New York, NY, USA, 2000.

[Gortz 06] Inge Li Gortz & Anthony Wirth. *Asymmetry in k-center variants*. Theor. Comput. Sci., vol. 361, no. 2, pages 188–199, 2006.

[Goscinski 05] Wojtek Goscinski & David Abramson. *Application Deployment over Heterogeneous Grids using Distributed Ant*. In e-Science, pages 361–368. IEEE Computer Society, 2005.

[Gribble 01] Steven D. Gribble, Matt Welsh, Rob von Behren, Eric A. Brewer, David Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, B. Zhao & Robert C. Holte. *The Ninja architecture for robust Internet-scale systems and services373423*. Comput. Networks, vol. 35, no. 4, pages 473–497, 2001.

[He 05] Yihua He. *How Asymmetric is Internet Routing ? A Systematic Approach*. In Special Interest Group on Data Communications (SIGCOMM), 2005.

[Hummel 05] Karin A. Hummel. *Mobility-Aware Coordination in a WLAN Hot-Spot Area*. In Thomas Magedanz, Ahmed Karmouch, Samuel Pierre & Iakovos S. Venieris, editeurs, MATA, volume 3744 of *Lecture Notes in Computer Science*, pages 294–304. Springer, 2005.

[Hussein 06] Mohammad Hussein, Franck Morvan & Abdelkader Hameurlain. *Dynamic Query Optimization : from Centralized to Decentralized*. In Gregory D. Peterson, editeur, ISCA PDCS, pages 273–279. ISCA, 2006.

[Inmon 96] W. H. Inmon. *The Data Warehouse and Data Mining.* Commun. ACM, vol. 39, no. 11, pages 49–50, 1996.

[Ivan 02] Anca-Andreea Ivan, Josh Harman, Michael Allen & Vijay Karamcheti. *Partitionable Services : A Framework for Seamlessly Adapting Distributed Applications to Heterogeneous Environments*. In HPDC '02 : Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, page 103, Washington, DC, USA, 2002. IEEE Computer Society.

[Jain 99] Kamal Jain & Vijay V. Vazirani. *Primal-Dual Approximation Algorithms for Metric Facility Location and k-Median Problems*. In FOCS '99 : Proceedings of the 40th Annual Symposium on Foundations of Computer Science, pages 2–13, Washington, DC, USA, 1999. IEEE Computer Society.

[Karlsson 02] Magnus Karlsson & Mallik Mahalingam. *Do We Need Replica Placement Algorithms in Content Delivery Networks*. In 7th International Workshop on Web Content Caching and Distribution (WCW), August 2002.

[Kichkaylo 04] Tatiana Kichkaylo & Vijay Karamcheti. *Optimal Resource-Aware Deployment Planning for Component-Based Distributed Applications*. In HPDC '04 : Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, pages 150–159, Washington, DC, USA, 2004. IEEE Computer Society.

[Kimball 02] Ralph Kimball. The data warehouse toolkit : practical techniques for building dimensional data warehouses (seconde édition). John Wiley & Sons, Inc., New York, NY, USA, 2002.

[Kleinberg 00] Jon Kleinberg. *The small-world phenomenon : an algorithm perspective*. In STOC '00 : Proceedings of the thirty-second annual ACM symposium on Theory of computing, pages 163–170, New York, NY, USA, 2000. ACM Press.

[Lacour 05] S. Lacour, C. Perez & T. Priol. *Generic Application Description Model : Toward Automatic Deployment of Applications on Computational Grids*. In GRID '05 : Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, pages 284–287, Washington, DC, USA, 2005. IEEE Computer Society.

[Lowekamp 04] Bruce Lowekamp, Brian Tierney, Les Cottrell, Richard Hughes-Jones, Thilo Kielmann & Martin Swany. *A Hierarchy of Network Performance Characteristics for Grid Applications and Services*. In Global Grid Forum, june 2004.

[Maassen 06] Jason Maassen, Rob V. van Nieuwpoort, Thilo Kielmann, Kees Verstoep & Mathijs den Burger. *Middleware adaptation with the Delphoi service : Research Articles*. Concurr. Comput. : Pract. Exper., vol. 18, no. 13, pages 1659–1679, 2006.

[Malinowski 06] Elzbieta Malinowski & Esteban Zimányi. *Hierarchies in a multidimensional model : From conceptual modeling to logical representation.* Data Knowl. Eng., vol. 59, no. 2, pages 348–377, 2006.

[Melab 06] Nouredine Melab, Sébastien Cahon & El-Ghazali Talbi. *Grid computing for parallel bioinspired algorithms.* J. Parallel Distrib. Comput., vol. 66, no. 8, pages 1052–1061, 2006.

[Ng 04] T. S. Eugene Ng & Hui Zhang. *A network positioning system for the internet.* In ATEC'04 : Proceedings of the USENIX Annual Technical Conference 2004 on USENIX Annual Technical Conference, pages 11–11, Berkeley, CA, USA, 2004. USENIX Association.

[Ostrovsky 02] Rafail Ostrovsky & Yuval Rabani. *Polynomial-time approximation schemes for geometric min-sum median clustering.* J. ACM, vol. 49, no. 2, pages 139–156, 2002.

[Panigrahy 98] Rina Panigrahy & Sundar Vishwanathan. *An O(log* n) Approximation Algorithm for the Asymmetric p-Center Problem.* Journal of Algorithms, vol. 27, no. 2, pages 259–268, 1998.

[Pias 03] Marcelo Pias, Jon Crowcroft, Steve R. Wilbur, Tim Harris & Saleem N. Bhatti. *Lighthouses for Scalable Distributed Location.* In M. Frans Kaashoek & Ion Stoica, editeurs, IPTPS, volume 2735 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 2003.

[Pierson 05] Jean-Marc Pierson, Lionel Brunie, Maryvonne Miquel, Anne Tchounikine, Clarisse Dhaenens, Nouredine Melab, Talbi El ghazali, Abdelkader Hameurlain & Franck Morvan. *Grid for Geno-Medicine.* BioGrid'05, may 2005.

[Pierson 06] Jean-Marc Pierson. *A Pervasive Grid, from the data side.* Rapport technique RR-LIRIS-2006-015, LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/Ecole Centrale de Lyon, June 2006.

[Pierson 07] Jean-Marc Pierson, Julien Gossa, Pascal Wehrle, Yonny Cardenas, El Samad Mahmoud, Cahon Sebastien, Lionel Brunie, Clarisse Dhaenens, Hameurlain Abdel kader, Nouredine Melab, Maryvonne Miquel, Franck Morvan, Talbi El gazali & Anne Tchounikine. *GGM Efficient Navigation and Mining in Distributed Geno-Medical Data.* IEEE Transactions on NanoBioscience, vol. 6, no. 2, pages 110–116, May 2007.

[Quinson 02] Martin Quinson. *Dynamic Performance Forecasting for Network-Enabled Servers in a Metacomputing Environment.* In International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS'02), in conjunction with IPDPS'02, April 15-19 2002.

[Reese 06] J. Reese. *Solution methods for the p-median problem : An annotated bibliography.* Networks, vol. 48, no. 3, pages 125–142, 2006.

[Reiher 00] P. Reiher, R. Guy, M. Yarvis & A. Rudenko. *Automated Planning for Open Architectures.* In OPENARCH, Tel-Aviv, March 2000.

[Seymour 05] Keith Seymour, Asim YarKhan, Sudesh Agrawal & Jack Dongarra. *NetSolve : Grid Enabling Scientific Computing Environments.* In Lucio Grandinetti, editeur, Grid Computing : The New Frontier of High Performance Computing (Volume 14). Elsevier, 2005. Also available as CoreGRID TR-0001.

[Siddiqui 05] Mumtaz Siddiqui, Alex Villazon, Jurgen Hofer & Thomas Fahringer. *GLARE : A Grid Activity Registration, Deployment and Provisioning Framework.* In SC '05 : Proceedings of the 2005 ACM/IEEE conference on Supercomputing, page 52, Washington, DC, USA, 2005. IEEE Computer Society.

[Thain 05] Douglas Thain, Todd Tannenbaum & Miron Livny. *Distributed computing in practice : the Condor experience.* Concurrency - Practice and Experience, vol. 17, no. 2-4, pages 323–356, 2005.

[Truong 04] Hong-Linh Truong & Thomas Fahringer. *SCALEA-G : a Unified Monitoring and Performance Analysis System for the Grid.* vol. 12, no. 4, pages 225–237, Fall 2004.

[Wang 05] Yanyan Wang, Matthew J. Rutherford, Antonio Carzaniga & Alexander L. Wolf. *Automating experimentation on distributed testbeds*. In David F. Redmiles, Thomas Ellman & Andrea Zisman, editeurs, ASE, pages 164–173. ACM, 2005.

[Wehrle 07] Pascal Wehrle, Maryvonne Miquel & Anne Tchounikine. *A Grid Services-Oriented Architecture for Efficient Operation of Distributed Data Warehouses on Globus*. In AINA, pages 994–999. IEEE Computer Society, 2007.