Structuration auto-adaptative d'un système de grille pair-à-pair à large échelle

Bassirou Gueye, Olivier Flauzac, Cyril Rabat, Ibrahima Niang

Structuration auto-adaptative d'un système de grille pair-à-pair à large échelle

Bassirou Gueye[1,2] — Olivier Flauzac[1] — Cyril Rabat[1] — Ibrahima Niang[2]

[1] CReSTIC, UFR Sciences Exactes et Naturelles
Université de Reims Champagne Ardenne
FRANCE
{bassirou.gueye, olivier.flauzac, cyril.rabat}@univ-reims.fr

[2] LID, Département de Mathématiques et d'Informatique
Université Cheikh Anta Diop de Dakar
SENEGAL
{bassirou.gueye, ibrahima1.niang}@ucad.edu.sn

RÉSUMÉ. Dans cet article, nous proposons une extension et une implémentation de notre solution de structuration auto-adaptative dans un environnement de grilles P2P à large échelle. La spécification que nous avons proposée permet aussi bien le déploiement, la recherche et l'invocation de services tout en respectant le paradigme des réseaux P2P. De plus, elle est générique, c'est-à-dire applicable sur toute architecture pair-à-pair. Pour garantir cette propriété, étant donné que les systèmes distribués à large échelle ont tendance à évoluer en termes de ressources, d'entités et d'utilisateurs, nous proposons de structurer l'environnement de grille pair-à-pair en communautés virtuelles. Au sein de chaque communauté un nœud appelé PSI (Proxys Système d'Information) joue le rôle de registre de services. Afin de permettre une recherche efficace dans le système, un arbre couvrant constitué uniquement des PSI est maintenu. Les résultats de simulations ont montrés que notre solution garantit un passage à l'échelle en termes de dimensionnement du réseau et aussi de coût de recherches.

ABSTRACT. In this paper, we propose an extension and experimental evaluation of our self-adaptive structuring solution in an large-scale P2P Grid environment. The proposed specification, enables both services deployment, location and invocation of while respecting the P2P networks paradigm. Moreover, the specification is generic i.e. not linked to a particular P2P architecture. The increasing size of resources and users in large-scale distributed systems has lead to a scalability problem. To ensure the scalability, we propose to organize the P2P grid nodes in virtual communities. A particular node called ISP (Information System Proxy) acts as service directory within each cluster. On the other hand, resource discovery is one of the essential challenges in large-scale Grid environment. In this sense, we propose to build a spanning tree which will be constituted by the set of formed ISPs in order to allow an efficient service lookup in the system. An experimental validation, through simulation, shows that our approach ensures a high scalability in terms of clusters distribution and communication cost.

MOTS-CLÉS : Système P2P, Grilles de Services, Algorithmes distribués, Structuration, Arbre couvrant, Oversim.

KEYWORDS : P2P Systems, Grid services, Distributed Algorithms, Clustering, Spanning Tree, Oversim.

1. Introduction

L'évolution des systèmes informatiques se caractérise par une tendance forte vers la décentralisation des services. En effet, la communication, le partage de données et des ressources de calcul et de stockage sont des besoins fortement exprimés par les nouvelles applications informatiques [7]. Pour faire face à ces exigences, des systèmes de partage à large échelle tels que les grilles et les systèmes pair-à-pair se sont imposés.

Une grille de calcul (*Grid Computing*) [18] est une infrastructure constituée d'un ensemble de ressources informatiques appartenant à des entités administratives différentes. Ces ressources, interconnectées par un réseau, sont caractérisées par leur hétérogénéité et leur distribution géographique. Par le biais des grilles, les utilisateurs ont la possibilité d'accéder à des ressources distantes de calcul et de stockage, de lancer des applications qui demandent des ressources non disponibles localement.

L'émergence des Services Web [1] a fourni un cadre qui a initié son alignement avec les technologies de grilles, donnant ainsi naissance à l'OGSA (*Open Grid Service Architecture*) [19]. La spécification de grille de services a pour objectif de normaliser les services composant les grilles, afin de garantir l'interopérabilité de systèmes hétérogènes pour le partage et l'accès à des ressources de calcul et de stockage distribuées [46].

La gestion des ressources réparties géographiquement au sein de plusieurs VOs (Organisations Virtuelles), en particulier, la découverte appropriée de ressources constitue un des défis essentiels dans un environnement de grille [45, 32, 35]. Pour répondre à ces exigences, les grilles n'ont pas cessé d'évoluer en termes d'architectures qui guident le développement de tous les composants d'une application.

Toutefois, la plupart des grilles sont généralement basées sur des architectures centralisées ou hiérarchiques qui présentent un fort degré de centralisation [43, 40, 16, 10, 28, 31, 29]. Cette centralisation implique une gestion unifiée des ressources, mais aussi des difficultés à réagir vis-à-vis des pannes qui impactent la communauté.

Par ailleurs, Foster et Iamnitchi suggèrent que les grilles peuvent fortement tirer profit des technologies pair-à-pair [25]. En effet, les systèmes pair-à-pair (P2P) sont des environnements où chaque entité peut à la fois jouer le rôle de client et de serveur. Ils offrent de nombreux avantages grace à leurs propriétés fondamentales et inhérentes telles que l'auto-organisation, la tolérance aux pannes, le passage à l'échelle, le changement dynamique de topologie, etc. [39, 3].

Plusieurs solutions de découverte de ressources dans un environnement de grille pair-à-pair ont été proposées dans la littérature. Elles peuvent être classé en fonction du degré de centralisation de l'architecture de grille qui impacte grandement sur la tolérance aux pannes, ainsi que de la possibilité pour l'application de passer à l'échelle. On distingue ainsi, les approches basées sur le modèle pair-à-pair décentralisé non-structuré [23, 24, 26, 37, 9], celles basées sur le modèle pair-à-pair décentralisé structuré [38, 34, 41, 27] et enfin les les approches basées sur le modèle pair-à-pair hybride [14, 33, 36, 42].

Chacun de ces modèles bénéficie de plusieurs des avantages qu'offrent les systèmes pair-à-pair. En effet, suivant l'environnement où ces modèles s'exécutent (peu ou très

dynamique, homogène ou hétérogène, large échelle ou échelle moyenne, etc.), certaines architectures sont plus adaptées que d'autres. Cependant, l'inconvénient majeur que présente le modèle d'architecture P2P non-structurée, est l'incomplétude des résultats de recherche. En effet, une recherche peut ne pas être fructueuse même si la ressource demandée se trouve dans le système. En ce qui concerne le modèle d'architecture P2P structurée, les DHTs ne proposent pas une recherche par mot clés. Une adaptation de la DHT sera ainsi nécessaire pour mieux répondre aux utilisateur. Enfin, le modèle d'architecture P2P hybride (ou super-pair) semble être le mieux adapté au contexte de grilles vu que celles-ci vu sont de nature structurées en VOs. Toutefois, le choix optimal des super-pairs n'est pas trivial, plusieurs critères sont à définir selon les besoins de l'application.

Dans ce contexte, nous proposons une solution de structuration auto-adaptative d'un système de grilles P2P à large échelle en communautés ou clusters. Ces travaux viennent compléter notre spécification décrit dans [20, 21]. La spécification est générique, c'est-à-dire applicable sur toute architecture pair-à-pair. En outre, elle permet aussi bien le déploiement, la recherche, l'invocation et l'exécution de services tout en respectant le paradigme des réseaux P2P. Afin de permettre une recherche efficace dans notre système, nous proposons de maintenir un arbre couvrant constitué uniquement des différents responsables de clusters.

Le reste du document est organisé comme suit. Dans la Section 2 nous exposons les travaux connexes. Une vue d'ensemble de notre spécification est décrite dans la Section 3. Dans la Section 4 nous décrivons notre approche de structuration d'un système de grilles P2P à large échelle. Nos résultats expérimentaux sont présentés dans la Section 5. Une conclusion ainsi que des perspectives futures sont données dans la Section 6.

2. Travaux connexes

Les grilles n'ont pas cessé d'évoluer en termes d'architectures qui guident le développement de tous les composants d'une application. En ce qui concerne la découverte de ressources, plusieurs mécanismes, que nous classons en fonction du degré de centralisation de l'architecture, ont été proposées dans la littérature. On peut ainsi distinguer d'une part, les approches basées sur le modèle traditionnel client/serveur et d'autre part, celle basées sur le modèle pair-à-pair.

Les solutions basées sur le modèle client/serveur sont soit centralisées [15, 6, 28, 29] ou hiérarchiques [43, 40, 16, 10, 28, 31]. Toutefois, les solutions basées sur ce modèle sont sensibles aux pannes et généralement inaptes à passer à l'échelle. En effet, le serveur central est un point critique car s'il tombe en panne, l'ensemble de son service devient inaccessible. De plus, ce modèle souffre d'un point de défaillance unique, de goulots d'étranglement dans les environnements hautement dynamiques et grandes échelles.

Pour remédier à ces inconvénients, des solutions de convergence des grilles et des systèmes pair-à-pair ont été proposées [17]. Ces solutions peuvent être classées en trois familles à savoir, les mécanismes basés sur le modèle P2P décentralisé non-structuré [23,

24, 26, 37, 9], ceux basés sur le modèle P2P décentralisé structuré [38, 34, 13, 41, 27] et ceux basés sur le modèle P2P hybride [14, 33, 36, 42].

Les mécanismes de découverte les plus utilisées et basés sur le modèle d'architecture P2P non-structurée, sont l'inondation et la marche aléatoire. La recherche par inondation (*flooding*) consiste à retransmettre récursivement la requête à tous les voisins d'un nœud (sauf celui dont il a reçu la requête) jusqu'à la localisation du service ou l'expiration TTL qui traduit le nombre de retransmissions à effectuer. Cette technique est cependant très coûteux en termes de messages. Des solutions d'améliorations, telles que les marches aléatoires [24, 26], ou encore les marches aléatoires biaisés [37, 44], ont été proposées. La recherche par marche aléatoire consiste à retransmettre récursivement la requête de recherche à un unique voisin choisi aléatoirement. La marche aléatoire est biaisée lorsque la requête est retransmise à un voisin choisi de manière déterministe. Par exemple, dans Gia [12], le choix se base sur le voisin qui a le plus fort degré.

Dans tous les cas, le choix du TTL n'est pas facile à déterminer. Les performances de la recherche dans un tel système sont donc tributaires du TTL qui possède une valeur bornée. En effet, si le TTL est grand, cela peut surcharger le réseau ; et s'il est petit, une recherche peut échouer même si la ressource demandée se trouve dans le système.

Les mécanismes de découverte basés sur le modèle d'architecture P2P structurée reposent généralement sur les DHTs (*Distributed Hash Tables*) [3] qui permettent des recherches rapides et efficaces, s'opérant en temps logarithmique. En outre, la nature distribuée de la table de hachage entre les différents pairs confère à ce type de système une certaine robustesse face aux pannes. Cependant, l'inconvénient majeur est que ces systèmes nécessitent un protocole assez lourd pour la maintenance de leur structure afin de faire face à la dynamique du système. De plus, les DHTs de base permettent uniquement une "recherche exacte" et exigent de ce fait des adaptations pour des expressions de recherche sur des mots-clefs [13, 41, 27].

Dans les mécanismes de découverte basés sur les systèmes P2P hybrides, les nœuds de la grilles n'ont pas les mêmes responsabilités. On distingue en effet, des nœuds appelés *super-nœuds* qui vont jouer le rôle d'annuaire en indexant les méta-données des nœuds rattachés à eux et appelés *nœuds ordinaires*.

Ce modèle semble être le plus adapté à une architecture de grille. En effet, une grille étant constituée d'un ensemble de VOs, chaque *super-nœud* sera responsable d'une VO [14, 33, 36, 42]. De ce fait, la maintenance des méta-données de services d'une VO est gérée par le *super-nœud* responsable de celle-ci.

Les mécanismes de découverte basés sur ces systèmes présentent comme avantage majeur, la réduction du trafic des requêtes. En effet, les *nœuds ordinaires* ne sont pas concernés lors des processus de recherche. En outre, ces systèmes sont relativement tolérants aux pannes car l'indisponibilité d'un *super-nœud* n'affecte que son groupe et pas tout le système. Toutefois, ce type de système est plus complexe à mettre en œuvre. Les performances de la recherche dans un tel système dépendent de la manière dont les *super-nœuds* sont organisés et dont ils communiquent. De plus, le choix optimal des *super-nœuds* n'est pas trivial, plusieurs critères sont à définir selon les besoins de l'application.

3. La spécification P2P4GS

P2P4GS (*Peer-To-Peer For Grid Services*) [20, 21] est une spécification pour la gestion dynamique de services dans un environnement de grille pair-à-pair à large échelle. La spécification présente l'originalité de ne pas lier l'infrastructure pair-à-pair à la plateforme de gestion des services. Elle permet aussi bien le déploiement, la recherche, l'invocation et l'exécution de services tout en respectant le paradigme des réseaux pair-à-pair. En outre, la spécification est générique c'est-à-dire applicable sur toute architecture pair-à-pair. Pour garantir cette propriété, étant donné que les systèmes distribués à large échelle ont tendance à évoluer en termes de ressources, d'entités et d'utilisateurs, nous proposons de structurer l'environnement de grille pair-à-pair en communautés virtuelles. En effet, une structuration efficace d'un système permet de garder les performances satisfaisantes même avec l'augmentation de sa taille [4, 32].

Pour atteindre ces objectifs, nous proposons un modèle d'architecture constitué de quatre couches d'abstraction. Ces couches superposées mettent en évidence les différents mécanismes sous-jacents à l'environnement de grille P2P ainsi que les interactions entre les différentes entités du système. Au niveau de chaque couche, un certain nombre de tâches requises pour le fonctionnement global du système sont réalisées et fournies aux couches supérieures. La figure 1 présente l'architecture de la spécification P2P4GS.

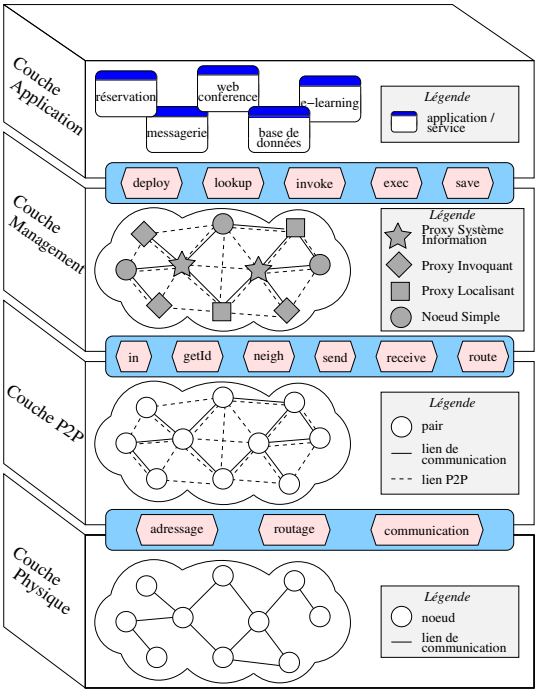


Figure 1 – Architecture de la spécification P2P4GS

Dans ce qui suit, nous allons décrire les différentes couches de l'architecture.

1) La couche physique représente le réseau physique de communication. Ce réseau est généralement Internet. Les nœuds d'une grille peuvent aussi être interconnectés à travers des réseaux haut-débit dédiés. C'est le cas par exemple de Grid'5000 [8] qui utilise une infrastructure réseau dédiée à 10 Gb/s fournie par RENATER.

Nous modélisons cette couche sous la forme d'un graphe orienté et connexe noté : $G_1 = (V, E_1)$ où V représente l'ensemble des nœuds du système (ordinateurs, supercalculateurs, clusters, etc.) et E_1, l'ensemble des liens physiques (les bus, les câbles ou les connexions sans fil) entre les différentes entités du réseau physique de communication.

Ces définitions permettent la prise en compte des différentes spécificités du réseau. Par exemple, les éléments de sécurité comme des pare-feux qui limitent la possibilité de communication bi-directionnelles des nœuds.

Cette couche fournit plusieurs services aux couches supérieures dont les fonctions d'adressage, de routage et de communication.

2) La couche P2P correspond à l'intergiciel pair-à-pair utilisé. Afin d'être exploitable dans notre spécification, cette couche est modélisée sous la forme d'un graphe non-orienté et connexe $G_2 = (V, E_2)$ où V correspond toujours à l'ensemble des nœuds (appelés aussi pairs) du système et E_2 représente l'ensemble des liens virtuels de communication entre les nœuds et établis par le protocole pair-à-pair.

Ces définitions permettent la prise en compte des différentes spécificités du réseau pair-à-pair de recouvrement. En effet, grace aux propriétés inhérentes des systèmes pair-à-pair, tout en faisant abstraction de la complexité du réseau physique de communication sous-jacent, deux nœuds se trouvant derrière des pare-feux peuvent être voisins et ainsi communiquer dans le réseau recouvrant.

Possédant des mécanismes de nommage, d'adressage, de communication, etc., la couche P2P fournit donc ces différentes fonctionnalités ainsi que la maintenance de la topologie du réseau. On suppose que le système pair-à-pair offre les primitives basiques de communication suivantes :

– *in()* cette primitive est exécutée par tout nouveau nœud connecté au système.

– *getId()* pour récuperer son identifiant fourni par la couche P2P ;

– *neigh()* pour récuperer la liste de tous ses voisins ;

– *send(id, message)* pour envoyer un message à un nœud d'identifiant *id* du système ;

– *receive()* pour reveçoir un message en provenance d'un nœud du système ;

– *route(id, message)* pour router le message vers une destination.

On peut remarquer que la primitive *out()* n'est pas prise en compte puisque dans la plupart des systèmes pair-à-pair, un nœud qui quitte le système ne prévient pas ses voisins. Donc, tout système pair-à-pair assurant ces fonctionnalités basiques pourra être exploité par notre spécification.

3) La couche management constitue le cœur de notre spécification. En fait, c'est au niveau de cette couche que toutes les tâches de gestion des services sont définies. Ces tâches vont du déploiement d'un service jusqu'à sa consommation, tout en passant par sa recherche et son invocation.

Vu la nature dynamique d'un environnement de grille pair-à-pair, des mécanismes de gestion des services sont nécessaires pour conserver la consistance d'une telle organisation. Ainsi, au sein de chaque communauté virtuelle, un nœud spécifique appelé PSI (*Proxy Système d'Information*) joue le rôle de registre de services. Un PSI connaît ainsi la localisation de l'ensemble des services partagés par les nœuds membres dits NS (*Nœuds Simples*) de sa communauté.

La gestion des services et précisément du cycle de vie des services inclut plusieurs sous-tâches complexes telles que la gestion du déploiement, la gestion de la localisation et la gestion de l'invocation ainsi que de l'exécution. En vue de ne pas surcharger les PSI, nous proposons de répartir certaines tâches sur d'autres types de nœuds distingués. Ainsi, les tâches d'invocation et d'exécution de services sont déléguées à des nœuds appelés PI (*Proxys Invoquant*). Etant donné que les services n'ont pas les mêmes contraintes d'exécution en termes de CPU, RAM, plateforme d'exécution, etc., un nœud est dit proxy invoquant pour un service S_i donné si et seulement : *i)* il connaît sa localisation ; *ii)* il respecte ses contraintes d'exécution.

Lorsqu'un nœud connait la localisation d'un service S_i (condition *i)*) mais ne satisfait pas la condition *ii)*, alors il sera nommé PL (*Proxy Localisant*) pour ce service.

Remarque 1. *Afin d'éviter une surcharge en mémoire les nœuds PI et PL, nous proposons de supprimer la connaissance sur la localisation d'un service à la fin d'un timeout noté $\mathcal{T}_{Live}$. De ce fait, un service qui est sollicité assez rarement dans le système ne va pas être mémorisé de manière indéfinie. Par contre, un nœud sera toujours proxy invoquant ou proxy localisant pour un service assez fréquemment sollicité du moment où son $\mathcal{T}_{Live}$ sera réinitialisé après chaque appel.*

4) La couche application sert d'interface aux utilisateurs pour l'accès aux services qu'offre l'environnement de grille P2P. En effet, les primitives de la couche sous-jacente (*deploy*, *lookup*, *invoke*, *exec* et *save*) sont exploitées par les différentes plate-formes avec lesquelles elles interagissent afin d'offrir des services à la couche application. Soulignons que l'utilisateur accède de manière transparente aux services de la grille P2P.

4. Structuration auto-adaptative d'un système de grille pair-à-pair

Dans cette section, nous présentons notre solution de structuration auto-adaptative d'un système de grille pair-à-pair à large échelle. Après avoir décrit l'originalité de notre solution, nous définissons les principales notations que nous utilisons dans nos algorithmes. Par la suite, nous exposons notre solution de structuration. Enfin, nous décrivons les mécanismes d'adaption à la dynamique du système.

Comme nous l'avons décrit précédemment, la croissance de la taille des systèmes à large échelle pose la question de la scalabilité. Pour garantir le passage à l'échelle, nous proposons de structurer le sytème en communautés virtuelles.

Notre approche de structuration est complément distribuée. Elle se base uniquement sur le voisinage des nœuds pour l'élections des PSI et ainsi la formation des communautés.

Son originalité est qu'elle se rapproche le plus possible des conditions réelles. En effet, dans un environnement à grande échelle où les nœuds sont géographiquement dispersés, le réseau ne peut pas se former de manière spontanée. Nous considérons ainsi que le réseau n'est pas créé à l'avance. Par conséquent, nous proposons d'élire les PSI au fur et à mesure de la connexion des nœuds en se basant sur leurs voisinages.

D'autre part, pour répondre à la problématique de découverte de ressources, nous proposons de construire un arbre couvrant constitué uniquement des PSI ainsi formés. La particularité de cette approche est que la construction de l'arbre couvrant se fait au moment même de la structuration du système. Ce qui permet de minimiser le coût des communications en termes de messages.

4.1. Définitions et notations

Les notations suivantes sont utilisées dans nos différents algorithmes.

– id_u : identifiant du nœud u.

– $statut_u$: statut du nœud u ; avec $statut_u \in \{UNDEF, NS, PSI\}$.

– $neighTable_u$: table de voisinage du nœud u. Elle reçoit des couples (id, $statut$).

– $updateNeighStatus$: mettre à jour le statut de son voisin.

– $neighList_u$: liste des voisins du nœud u.

– $psiList_u$: liste des voisins PSI du nœud u.

– $serviceList_u$: liste des services hébergés par le nœud u.

– $serviceRegistry_u$: liste des services indexés par le nœud u.

– $routingTable_u$: table de routage du nœud u.

– $gatewayTable_u$: table des PSI passerelles du nœud u.

– $\mathcal{T}_{timeout}$: temps d'attente de certains événements pour décider.

Structure des messages :

– *queryStatus(id, statut)* : demander le statut de son voisin.

– *responseStatus(id, statut)* : répondre à un message *queryManagement*.

– *updateStatus(id, statut)* : envoyer son statut à son voisin.

– *clusterManagement(serviceList, neighList)* : envoyer à son PSI la liste des services indexés ainsi que la liste de ses autres voisins PSI.

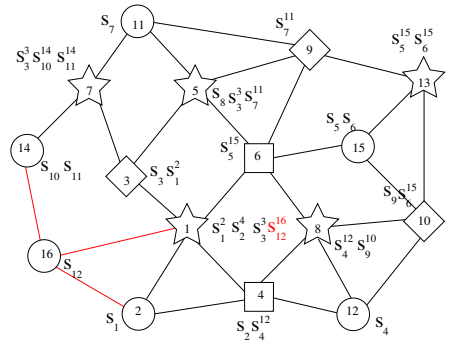
– *masterRouteManagement(id)* : informer au PSI qu'il est ton nœud passerelle.

– *routeDiscovery(id)* (respectivement responseRouteDiscovery(*psiList*)) : découverte de route (respectivement réponse découverte de route).

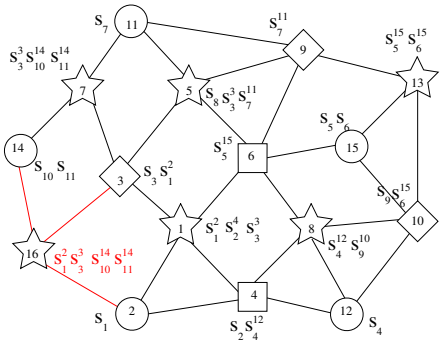
4.2. Algorithmes de structuration

Comme nous l'avons souligner plus haut, notre solution de structuration se base sur le voisinage pour élire les PSIs au fur et à mesure de la connexion des nœuds.

L'approche (naïve) de structuration que nous avons proposée dans [20, 21], se base uniquement sur le voisinage initial des nœuds pour l'élection des PSIs. Ce voisinage dépend protocole P2P sous-jacent et évolue au cours de l'exécution. Donc, lorsqu'un nœud se connecte dans le système et établit la connexion avec ses voisins initiaux, il consulte le statut de ces derniers. Ansi, s'il a au moins un voisin PSI, alors il se déclare NS (Nœud Simple) et informe ses voisins. Ensuite, il envoie les informations sur les services hébergés à ses voisins PSI. La figure 2(a) illustre ce cas. Si par contre le nœud n'a pas de voisin PSI, il devient alors PSI et informe ses voisins (voir figure 2(b)).



(a) Le nœud 16 a comme voisins, les nœuds 1 et 14. Puisqu'il a un voisin PSI (le nœud 1), il devient alors NS. Par la suite, il envoie à son PSI la liste des services indexés (dans cet exemple, les informations sur le service S_{12}). Ce dernier (le nœud 1) met à jour son registre de services.



(b) Le nœud 16 a comme voisins, les nœuds 2, 3 et 14, qui ont tous un statut NS. Ainsi, il devient PSI et informe ses voisins. Par la suite, ses voisins envoient la liste des services indexés (ici, S_1, S_3, S_{10} et S_{11}) au noeud 16 qui met ainsi à jour son registre de service.

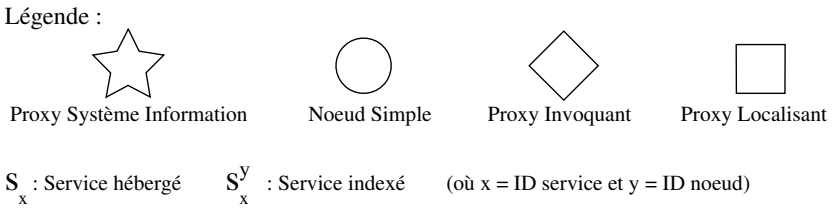


Figure 2 – Connexion d'un nœud dans le système

Nous proposons d'introduire le critère de degré de connexion dans l'approche initiale. Cette contrainte permet au système de se doter des mécanismes de contrôle sur la distribution des clusters (communautés virtuelles).

Ainsi, une fois qu'un nœud se connecte et établit son voisinage initial, il doit attendre d'avoir un nombre de voisins minimal (que nous notons $\Delta_{RequiredMinDegree}$) pour être candidat potentiel à l'élection de PSI. De ce fait, ceux sont les nœuds les plus distingués (plus réputés) et plus stables qui auront plus de chance d'être PSI.

Le principe d'exécution d'algorithme de structuration se déroule ainsi comme suit :

Initialement, le statut d'un nouveau nœud connecté est *UNDEF*. Lorsqu'un nœud u établit un nouveau voisinage avec un nœud v, si le statut de u est PSI ou NS, alors u

envoie son statut (*updateStatus*) au nœud v. Par contre, si le statut de u est *UNDEF* et que son nombre total de voisins atteint le degré minimal requis, alors il envoie un message *queryStatus* à l'ensemble de ses voisins pour demander leur statut.

Lorsqu'un nœud u reçoit un message *responseStatus*, il exécute l'Algorithme 1. Après avoir mis-à-jour le statut de l'émetteur ainsi que son degré (nombre total de voisins) :

– il se déclare PSI si son degré atteint le degré minimal requis ($\Delta_{RequiredMinDegree}$) et qu'il n'a pas de voisin PSI. Il informe ainsi ses voisins de son nouveau statut ; puis déclenche son $\mathcal{T}_{timeout}$. Le PSI attend les messages de ses voisins afin de définir son nœud de passerelle dans l'arbre couvrant.

– il se déclare NS s'il a au moins un voisin PSI. Dans ce cas, il informe ainsi ses voisins de son nouveau statut et s'il a des services hébergés, alors il envoie la liste de leur index (message *clusterManagement*) à l'ensemble de ses voisins PSIs.

Algorithme 1: A la réception d'un message responseStatus(id_v, $status_v$) du nœud v

```
1  updateNeighStatus(id_v, status_v) ; /* Mise à jour du statut de v */
2  if statut_u = UNDEF ∧ numberOfResponses = Δ_RequiredMinDegree then
3      if ∃ k ∈ neighTable / statut_k = PSI then
4          statut_u ⟵ NS;
5          forall the k ∈ neighTable / statut_k = PSI do
6              send(k, clusterManagement(ServiceList_u, neighList_u));
7          end
8      else
9          statut_u ⟵ PSI;
10     end
11     forall the k ∈ neighTable do
12         send(k, updateStatus(id_u, status_u)) ; /* Envoyer à k mon statut */
13     end
14 else
15     if statut_u = NS ∧ statut_v = PSI then
16         send(v, clusterManagement(ServiceList_u, neighList_u));
17     end
18 end
```

Lorsqu'un nœud de statut NS a un nouveau voisin PSI, alors il lui envoie la liste de ses services indexés ainsi que la liste de ses voisins (message *clusterManagement*).

A la réception d'un message *clusterManagement*, le PSI exécute l'Algorithme 2. Il met à jour son registre de services (*serviceRegistry*) et sa table de routage (*routingTable*). Ces mises-à-jour de sont effectuées à chaque réception de ce type de message. Les entrées dans la table *routingTable* correspondent uniquement aux identifiants des PSI récupérés dans $neighList_u$. Les autres identifiants (des NS) contenus dans $neighList_u$ pourront être utilisés par le PSI dans le cas où il souhaite découvrir une route.

A l'expiration de $\mathcal{T}_{timeout}$ (Algorithme 3), le PSI définit sa passerelle dans l'arbre couvrant sur la base des informations contenues dans sa table de routage. Pour ce faire, il choisit dans sa table de routage le PSI qui a numériquement le plus petit identifiant et l'ajoute dans sa table de passerelles (*gatewayTable*).

Algorithme 2: A la réception d'un message clusterManagement($serviceList_v$, $neighList_v$)

forall the $S_i \in serviceList_v$ **do**
  $serviceRegistry \longleftarrow serviceRegistry \cup \{(S_i, id_v)\}$;
end
forall the $q \in neighList_v$ / $statut_q = PSI$ **do**
  if $\nexists\ k = (id_k) \in routingTable$ / $id_k = id_q$ **then**
    $routingTable \longleftarrow routingTable \cup \{id_q\}$;
  end
end

Dans l'exemple de la figure 3 (pour simplifier on a numéroté les nœuds mais dans la pratique, les identifiants sont des adresses IP), le PSI 16, a dans sa table de routage, les PSI 1, 5 et 7. Ainsi, il choisit comme PSI passerelle, le nœud 1.

Le nœud ainsi choisi sera par la suite informé (message *masterRouteManagement*).

Si la table de routage est vide à l'expiration du *timer*, alors cela signifie qu'il n'existe de PSI au voisinage de ses voisins. Ainsi, le PSI se sert des NS contenus dans $neighList_u$ et n'appartenant pas à son voisinage pour la découverte de PSI passerelle.

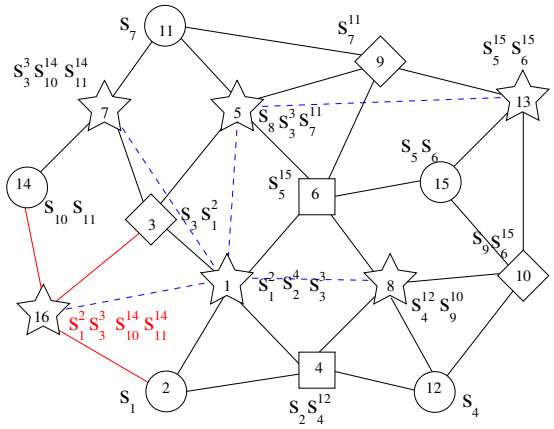


Figure 3 – Evolution de la structuration et choix d'un PSI passerelle

Algorithme 3: A l'expiration du temps $\mathcal{T}_{timeout}$

if $|gatewayTable| = 0 \wedge |routingTable| > 0$ **then**
  $gatewayTable \longleftarrow gatewayTable \cup \{id_k$ / $id_k = \min(routingTable)\}$;
  send(k, $masterRouteManagement(id_u)$);
else
  forall the $k \in neighTable$ **do**
    if $\exists\ q = (id_q) \in neighList_k$ / $id_q \nexists\ neighTable$ **then**
      /* Envoyer à q un message de découverte de route */
      send(q, $routeDiscovery(id_{PSI})$);
    end
  end
end

Lorsqu'un nœud u reçoit un message *masterRouteManagement* d'un nœud v, il exécute l'algorithme 4. Il ajoute alors l'identifiant du nœud v comme une nouvelle entrée dans sa table de passerelles (*gatewayTable*). Ainsi l'arbre couvrant (représenté par les liens en pointillés bleus) se construit au fur-et-à-mesure de la structuration du système.

Algorithme 4: A la réception d'un message masterRouteManagement(id_v) du nœud v

```
1 if ∄ k = (id_k) ∈ gatewayTable / id_k = id_v then
2 |   gatewayTable ⟵ gatewayTable ∪ {id_v};
3 end
```

4.3. Mécanismes d'adaptation à la dynamique du système

Nous décrivons dans cette section les mécanismes d'adaptation face à la dynamique de l'environnement de grille pair-pair.

Au cours de l'évolution d'un système distribué, des modifications topologiques peuvent se produire à tout moment. Une modification topologique se traduit par l'apparition (connexion) ou la disparition (déconnexion) de nœuds. Il est par conséquent nécessaire de mettre en place des mécanismes d'adaptation afin d'assurer l'évolutivité du système.

Notre approche de structuration propose d'élire les PSI au fur et à mesure de la connexion des nœuds en se basant sur leurs voisinages. La spécification s'adapte donc à la connexion de nœuds. Les mécanismes que nous mettons en place, vont ainsi s'intéresser à la déconnexion de nœuds.

Les déconnexions de nœuds peuvent être volontaires dans le cas par exemple d'un service accompli ou bien involontaires quant il s'agit de nœuds défaillants ou de nœuds mobiles quittant une zone de couverture ou le cluster. En outre, les canaux de communication peuvent également être non fonctionnels une période donnée. Par conséquent, nous considérons qu'un nœud est en panne, lorsqu'il n'est plus joignable après un certain temps prédéfini et configurable, dit *délai de garde*.

Toute technique de gestion de pannes comprend un mécanisme chargé de les détecter. Les pannes sont généralement détectées soit par des messages périodiques de type *heartbeats*, ou des messages *ping-pong* [11, 30, 22].

Nous proposons dans nos travaux d'utiliser ces deux types de messages. En effet, les PSI envoient périodiquement des messages *heartbeat* à leurs voisins afin de signaler leur présence. De plus, ils envoient des messages *ping-pong* à leurs PSIs passerelles ainsi qu'à leurs successeurs potentiels. Donc une table (*candidateTable*) devant contenir les candidats successeurs potentiels est maintenue par chaque PSI.

Un nœud est dit successeur potentiel dans une communauté virtuelle donnée, s'il a un nombre de voisins qui atteint le degré minimal requis ($\Delta_{RequiredMinDegree}$) et qu'il n'a pas d'autres voisins PSI. Lorsqu'un nœud remplit ces conditions, alors il envoie un message *candidateManagement* à son PSI.

A la réception d'un message *candidateManagement*, le PSI met à jour sa table *candidateTable* et informe à l'émetteur sa position dans la table. De cette manière, une élection

ne sera pas nécessaire en cas de disparition d'un PSI. En effet, c'est le candidat à la première position (logiquement, le plus ancien entre le candidats et manifestant ainsi un certain degré de stabilité), qui se déclarera automatiquement PSI après le délai de garde.

De même, lorsqu'un PSI envoie un message *ping-pong* à un candidat successeur potentiel v et ne reçoit pas de réponse jusqu'à l'expiration du délai de garde, alors le PSI va considérer que le nœud v est en panne et par conséquent, il sera retiré de la table *candidateTable*. Lorsqu'il y'a modification de la table *candidateTable*, après le retrait élément, les mis-à-jour d'indices sont envoyées aux candidats.

En outre, le PSI envoie son cache aux k premiers nœuds déclaré comme successeurs potentiels. Des messages de mise-à-jour seront aussi envoyées en cas de modification du registre. Notons qu'un cache n'est manipulé par un successeur potentiel que lorsqu'il devient PSI. Donc il représente un *backup* du registre d'un PSI afin de permettre un service continu en cas de panne de celui-ci.

Si, après un certain délai de garde, un NS ne reçoit pas de message *heartbeats* de la part du PSI en panne, il mettra alors à jour sa table de voisinage *neighTable*. Ensuite, le nœud envoie à ses voisins PSI (donc les PSIs passerelles du PSI en panne), la liste à jour de ses voisins. Ces derniers mettront à jour leurs tables *routingTable* et *gatewayTable*.

Remarque 2. *Vu la nature dynamique des environnement de grille pair-à-pair, le système tolère la volatilité d'un nœud simple (NS). Ainsi la déconnexion d'un NS n'est repéré que si son service est demandé. Donc lorsqu'un PSI sollicite son voisin NS pour accéder à un de ses services et que ce voisin ne répond pas alors, le PSI considère que son service est indisponible. Le $\mathcal{T}_{Live}$ est alors déclenché. Si ce dernier expire sans que le nœud soit joignable, alors le PSI supprime les informations sur ce nœud ainsi que ses services.*

Pour assurer la disponibilité d'un service, nous envisageons de le répliquer selon un facteur qui dépendra de la réputation du service.

Remarque 3. *Les statuts PI et PL n'entrent pas en jeu dans le processus de structuration. En effet, c'est suite à une opération de localisation d'un service qu'un nœud, en fonction de ses ressources, devient PI ou PL pour ce service. Il garde le statut pour ce service pour une durée dépendant de la fréquence de sollicitation du service.*

5. Expérimentation et évaluation de performances

Dans cette section, nous donnons une implémentation et une évaluation de notre solution par simulations.

Nous avons utilisé le Framework Oversim [5] d'OMNeT++[1] qui est un simulateur open source à événement discret et hautement modulaire. Plusieurs protocoles P2P (structurés comme non structurés) sont implémentés dans OverSim.

Ainsi, afin d'atteindre nos objectifs, nous avons implémenté notre solution sur des protocoles P2P fonctionnant de manière totalement différente, à savoir Gia [12] qui est un

1. http ://www.omnetpp.org/

overlay non structuré, Pastry [38] qui est un overlay structuré en anneau et Kademlia [34] qui est lui un overlay structuré en hypercube bien que modélisé souvent sous la forme d'un arbre.

Nos simulations sont effectuées sous Grid'5000 [8]. Nous avons utilisé les nœuds de Saint Rémi du Centre de Calcul de Champagne-Ardenne ROMEO[2] et ceux de Sophia[3].

5.1. Métriques de performance et paramètres de simulations

Afin d'analyser les performances de notre solution, nous considérons les métriques d'évaluations suivantes :

– Pourcentage de PSIs : il définit le nombre PSIs formés sur le nombre total de nœud dans le système. En d'autres termes, il correspond au pourcentage de clusters (communautés virtuelles) en fonction de la taille du réseau. Il permet ainsi de définir le dimensionnement du système et a un impact sur le degré de centralisation de l'information.

– Coût de recherche : il définit le diamètre de l'arbre couvrant qui représente le nombre maximal de sauts à effectuer lors de la procédure de découverte de services.

Les paramètres que nous utilisons dans nos différentes simulations sont résumés dans le tableau 1. Nous utilisons les valeurs par défaut proposées dans Oversim.

	Paramètres	**Valeurs**
Paramètres Globaux	Nombre de nœuds	[500, 5000]
	Init Phase Creation Interval	0.1 s
Protocole Gia	GIA Level of Satisfaction	1
	Aggressiveness of Adaptation	256
	Maximum Neighbors	50
Protocole Pastry	Bits Per Digit	4
	Number of Leaves	16
	Proximity Neighbor Selection	On
Protocole Kademlia	Bits Per Digit	1
	Bucket Nodes	16
	Sibling Nodes	8

Tableau 1 – Paramètres de simulation pour l'évaluation de la spécification P2P4GS

5.2. Performances de la première approche de structuration

Dans cette section, nous évaluons de performance de notre première approche de structuration.

Pour évaluer le pourcentage de PSI, nous considérons des réseaux avec un nombre de nœuds variant entre 500 et 5000. La Figure 4 représente, pour chaque protocole P2P sous-jacent (i.e. Gia, Pastry et Kademlia), le pourcentage de PSIs formés en fonction de la taille du système.

2. https ://romeo.univ-reims.fr
3. http ://www-sop.inria.fr/grid5000/

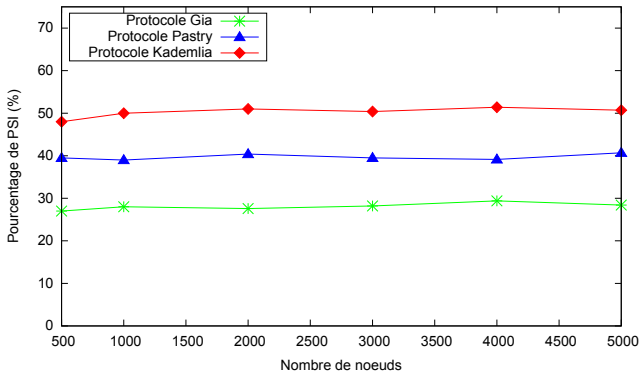


Figure 4 – Percentage of formed ISP according to P2P protocol and the network size

Comme nous pouvons le constater, cette approche de structuration passe à l'échelle puisque le pourcentage de PSIs formés reste relativement constant lorsque la taille du réseau augmente.

Toutefois, le pourcentage de PSI (et donc de clusters) est supérieur à 20% pour les différents protocoles P2P. Comme le précise des travaux dans la littérature [2], nous préconisons un pourcentage de clusters compris entre 5% et 20% pour un meilleur dimensionnement du réseau.

5.3. Impact du degré minimal requis sur les performances du système

Dans cette section, nous étudions l'impact du degré minimal requis ($\Delta_{RequiredMinDegree}$) sur les performances du système.

Nous allons dans un premier temps évaluer le pourcentage de PSIs formés en fonction de la taille du système. Dans un second temps, nous évaluons les coûts des communications en termes de nombre de messages. Pour ce faire, nous mesurons d'une part, le nombre de messages requis pour la formation des groupes virtuels. D'autre part, nous déterminons le diamètre de l'arbre couvrant qui représente le nombre maximal de sauts à effectuer lors de la procédure de découverte de services.

5.3.1. Pourcentage de PSIs en fonction du degré minimal requis

Nous avons introduit le critère de degré minimal requis ($\Delta_{RequiredMinDegree}$) dans le but de mieux contrôler la distribution des groupes virtuels. En effet, la contrainte sur le degré minimal requis permet d'une part, d'éviter la création de clusters singletons (c'est-à-dire contenant qu'un seul nœud et dans ce cas le PSI) ou encore de clusters contenant un nombre insignifiant de nœuds. Cela signifie que $\Delta_{RequiredMinDegree}$ ne doit pas prendre une valeur trop petite.

D'autre part, cette contrainte permet d'éviter la création d'un faible nombre de PSI. En effet, plus le nombre de PSI est petit, plus les PSI ont un grand nombre de voisins et donc plus les clusters sont denses. Ce qui impliquera une plus forte centralisation de l'information pouvant ainsi provoquer une surcharge de PSI et favoriser des goulots d'étranglement

dans le réseau. En outre, les risques d'indisponibilité en cas de panne augmentent. Cela signifie que $\Delta_{RequiredMinDegree}$ ne doit pas prendre une valeur trop grande.

Pour évaluer ainsi le pourcentage de PSI en fonction du degré minimal requis, nous considérons des réseaux avec un nombre de nœuds variant entre 500 et 5000. Pour chaque taille de réseau, nous faisons varié le paramètre $\Delta_{RequiredMinDegree}$ entre 4 et 24.

Les figures 5, 6 and 7 représentent le pourcentage de PSIs formés en fonction du degré minimal requis ($\Delta_{RequiredMinDegree}$) et de la taille du réseau, suivant respectivement le protocole P2P Gia, Pastry et Kademlia.

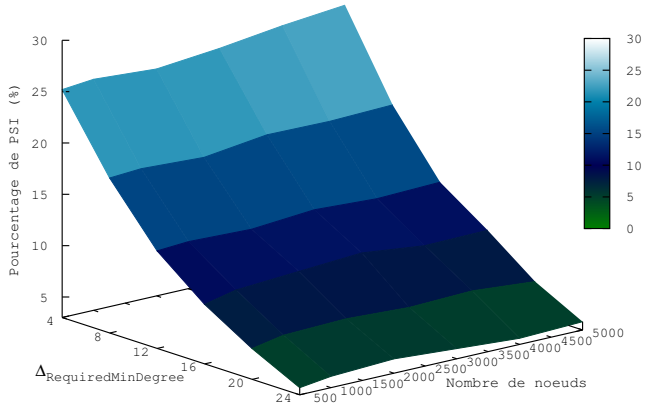


Figure 5 – Protocole Gia : Pourcentage de PSIs formés en fonction du degré minimal requis ($\Delta_{RequiredMinDegree}$) et de la taille du réseau

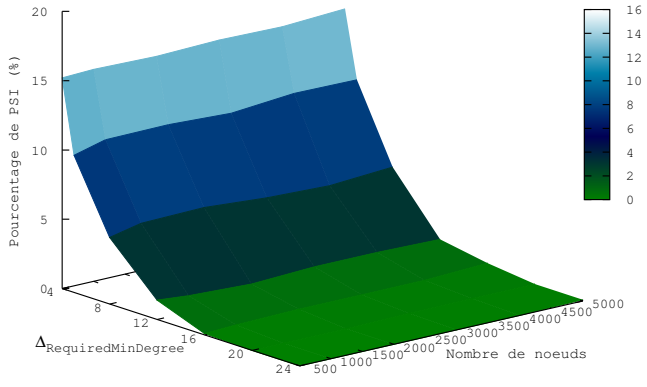


Figure 6 – Protocole Pastry : Pourcentage de PSIs formés en fonction du degré minimal requis ($\Delta_{RequiredMinDegree}$) et de la taille du réseau

Nous constatons d'une part, que pour chaque protocole P2P sous-jacent et pour chaque valeur de $\Delta_{RequiredMinDegree}$, le pourcentage de PSIs formés reste relativement constant

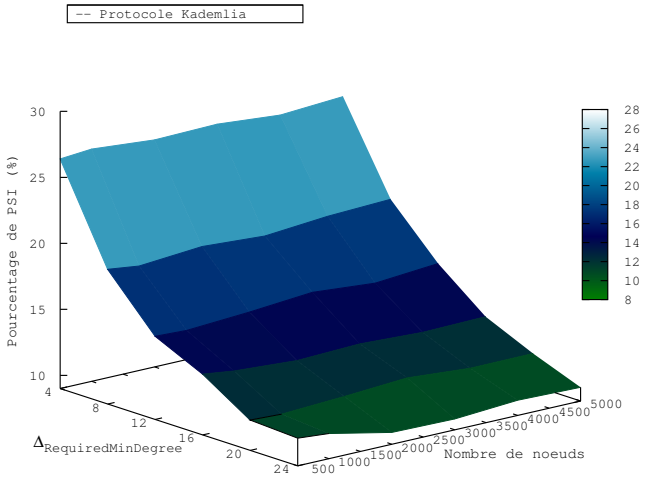


Figure 7 – Protocole Kademlia : Pourcentage de PSIs formés en fonction du degré minimal requis ($\Delta_{RequiredMinDegree}$) et de la taille du réseau

lorsque la taille du réseau augmente. Ce qui confirme le passage à l'échelle de notre solution.

D'autre part, comme nous pouvions l'imaginer, les résultats représentés dans les figures 5, 6 et 7 montrent que le pourcentage de PSIs formés diminue, lorsque le degré minimal requis augmente. Ainsi, en fonction du protocole P2P sous-jacent, nous donnons dans ce qui suit, les valeurs de $\Delta_{RequiredMinDegree}$ qui fournissent un meilleur dimensionnement du système, c'est-à-dire des pourcentages de PSI compris entre 5% et 20%.

Par ailleurs, nous évitons aussi de construire un faible nombre de PSI très denses en vue de ne pas les surcharger et aussi ne pas favoriser des goulots d'étranglement dans le réseau. De ce fait, comme le précise plusieurs travaux dans la littérature, nous préconisons un pourcentage de PSI compris entre 5% et 20% pour un meilleur dimensionnement du système.

– Cas du protocole Gia : la figure 5 montre que le pourcentage de PSI formés varie entre $3,6\%$ et $27,1\%$. Donc, les valeurs du paramètre $\Delta_{RequiredMinDegree}$ qui fournissent une meilleure distribution de clusters sont comprises entre 8 et 20 ;

– Cas du protocole Pastry : dans la figure 6, nous observons que le pourcentage de PSI formés varie entre 0 and $15,33\%$. Ainsi, les valeurs du paramètre $\Delta_{RequiredMinDegree}$ qui fournissent une meilleure distribution de clusters sont comprises entre 4 et 8 ;

– Cas du protocole Kademlia : la figure 7 montre que le pourcentage de PSI formés varie entre $9,83\%$ et $26,6\%$. De ce fait, les valeurs du paramètre $\Delta_{RequiredMinDegree}$ qui offrent une meilleure distribution de clusters sont comprises entre 8 et 24 ;

D'une manière générale, nous constatons que le protocole Pastry fournit un meilleur dimensionnement du système avec des valeurs de $\Delta_{RequiredMinDegree}$ relativement faibles. Contrairement à ce protocole, Kademlia construit des réseaux denses. C'est pour cela que les valeurs de $\Delta_{RequiredMinDegree}$ fournissant un meilleur dimensionnement sont relativement élevées. On remarque enfin que le protocole Gia fournit de meilleures performances en termes de distribution de clusters.

En effet, le protocole Gia assure que les nœuds à plus haut degré sont les nœuds à plus haute capacité en termes de CPU, bande passante, mémoire, etc.. De plus le protocole intègre un mécanisme de contrôle de flux pour éviter la surcharge des nœuds. En outre, chaque nœud calcule indépendamment son niveau de satisfaction (S). Ainsi, dans la mesure où un nœud est pas entièrement satisfaits, l'adaptation de la topologie va continué à rechercher des voisins appropriés pour améliorer le niveau de satisfaction.

5.3.2. Diamètre de l'arbre couvrant en fonction du degré minimal requis

Nous étudions dans cette section l'impact de $\Delta_{RequiredMinDegree}$ sur le diamètre de l'arbre couvrant qui représente le nombre maximal de sauts à effectuer lors de la procédure de découverte de services. Rappelons que l'arbre couvrant est construit lors de la phase de structuration du réseau et est constitué uniquement des PSIs ainsi formés.

Nous considérons ainsi des réseaux avec un nombre de nœuds variant entre 500 et 5000 et pour chaque taille de réseau, nous faisons varié le paramètre $\Delta_{RequiredMinDegree}$ entre 4 et 24, par intervalles de 4.

Les figures 8, 9 and 10 représentent le diamètre de l'arbre couvrant en fonction du degré minimal requis ($\Delta_{RequiredMinDegree}$) et de la taille du réseau, suivant respectivement le protocole P2P Gia, Pastry et Kademlia.

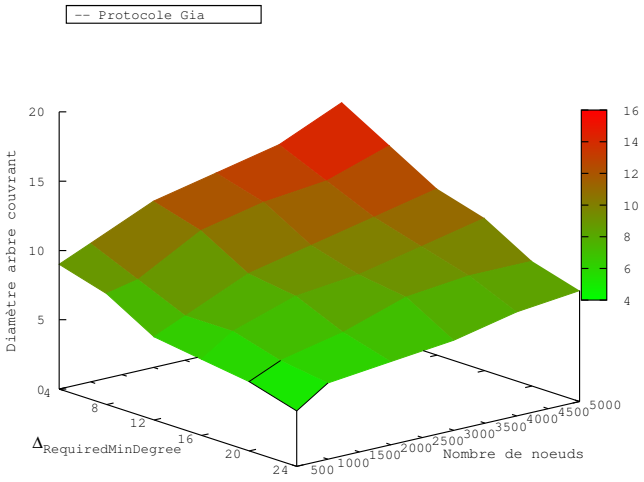


Figure 8 – Protocole Gia : Diamètre de l'arbre couvrant en fonction du degré minimal requis ($\Delta_{RequiredMinDegree}$) et de la taille du réseau

Nous observons d'une part, qu'au niveau des différents protocoles P2P sous-jacents et pour chaque valeur du degré minimal requis, le diamètre de l'arbre couvrant croît de manière logarithmique lorsque la taille du réseau augmente. Ce qui confirme encore une fois le passage à l'échelle de notre solution.

D'autre part, les résultats présentés dans les figures 8, 9 et 10 montrent que le diamètre de l'arbre couvrant diminue, lorsque le degré minimal requis augmente. Ce qui était aussi prévisible. Nous remarquons que que pour chaque taille de réseau, le degré de diminu-

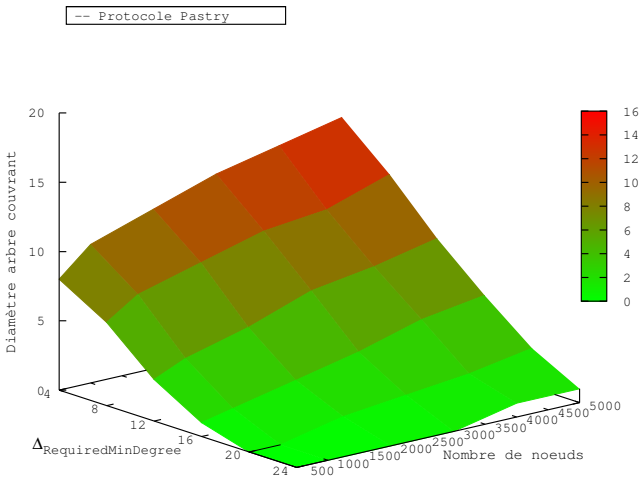


Figure 9 – Protocole Pastry : Diamètre de l'arbre couvrant en fonction du degré minimal requis ($\Delta_{RequiredMinDegree}$) et de la taille du réseau

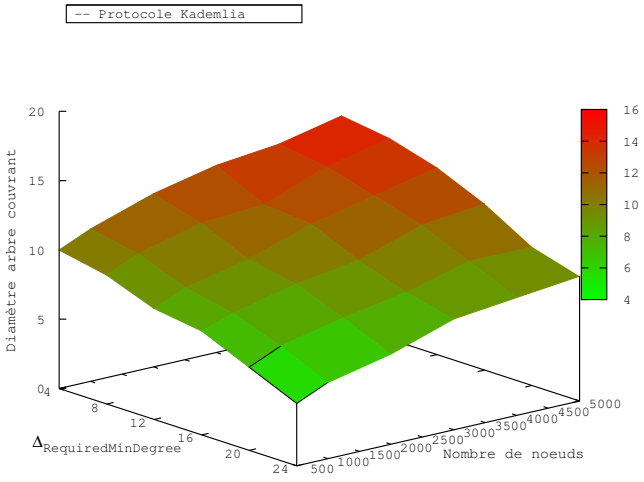


Figure 10 – Protocole Kademlia : Diamètre de l'arbre couvrant en fonction du degré minimal requis ($\Delta_{RequiredMinDegree}$) et de la taille du réseau

tion varie d'un protocole à un autre. En effet, dans le cas du protocole Gia, le diamètre de l'arbre couvrant diminue proportionnellement avec l'augmentation du degré minimal requis. Cette diminution du diamètre de l'arbre est relativement faible si nous considérons le protocole Kademlia. Ce qui atteste du fort degré de connexité des nœuds d'un réseau opérant avec ce protocole. Par contre, le diamètre de l'arbre couvrant diminue très considérablement avec l'augmentation du degré minimal requis dans le cas du protocole Pastry.

6. Conclusion et perspectives

Dans cet article, nous avons proposé une extension et une implémentation de notre solution de structuration auto-adaptative dans un environnement de grilles P2P à large échelle. La spécification P2P4GS que nous avons proposé est générique c'est à dire non liée à une architecture pair-à-pair particulière. Pour garantir le passage à l'échelle, nous avons proposé de structurer l'environnement de grille pair-à-pair en communautés virtuelles. Afin de permettre une recherche efficace dans notre système, nous avons proposé de maintenir un arbre couvrant constitué uniquement des différents PSI. En outre, en utilisant le simulateur OverSim, nous avons validé nos travaux, en évaluant d'une part, pourcentage de PSI formés et d'autre part, le coût des recherches. Pour illustrer la "généricité" de la spécification, nous avons simulé avec des protocoles opérant de manière totalement différentes. Les résultats de simulations ont montrés que notre solution garantit un passage à l'échelle en termes de dimensionnement du réseau et aussi de coût de recherches.

Dans nos futurs travaux, nous évaluons le nombre de saut moyen lors d'un processus de découverte services. Nous prévoyons également d'évaluer les pannes ainsi que le processus déploiement de services suivant les différentes stratégies proposées.

Références

[1] ALONSO, G., CASATI, F., KUNO, H., AND MACHIRAJU, V. *Web services*. Springer, 2004.

[2] AMINI, N., VAHDATPOUR, A., XU, W., GERLA, M., AND SARRAFZADEH, M. Cluster size optimization in sensor networks with decentralized cluster-based protocols. *Computer communications 35*, 2 (2012), 207–220.

[3] ANDROUTSELLIS-THEOTOKIS, S., AND SPINELLIS, D. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys 36*, 4 (2004), 335–371.

[4] BASAGNI, S. Distributed clustering for ad hoc networks. In *IEEE SPAN* (1999), pp. 310–315.

[5] BAUMGART, I., HEEP, B., AND KRAUSE, S. OverSim : A flexible overlay network simulation framework. In *IEEE Global Internet Symposium, 2007* (2007), IEEE, pp. 79–84.

[6] BENSON, E., WASSON, G., AND HUMPHREY, M. Evaluation of uddi as a provider of resource discovery services for ogsa-based grids. In *IPDPS* (2006), IEEE, pp. 9–pp.

[7] BERNHOLDT, D., BHARATHI, S., BROWN, D., CHANCHIO, K., CHEN, M., CHERVENAK, A., CINQUINI, L., DRACH, B., FOSTER, I., ET AL. The earth system grid : Supporting the next generation of climate modeling research. *Proceedings of the IEEE 93*, 3 (2005), 485–495.

[8] BOLZE, R., CAPPELLO, F., CARON, E., DAYDÉ, M., DESPREZ, F., JEANNOT, E., JÉGOU, Y., LANTERI, S., LEDUC, J., MELAB, N., ET AL. Grid'5000 : a large scale and highly reconfigurable experimental grid testbed. *IJHPCA 20*, 4 (2006), 481–494.

[9] BROCCO, A., MALATRAS, A., AND HIRSBRUNNER, B. Enabling efficient information discovery in a self-structured grid. *Future Generation Computer Systems 26*, 6 (2010), 838–846.

[10] CARON, E., AND DESPREZ, F. DIET : A scalable toolbox to build network enabled servers on the grid. *IJHPCA 20*, 3 (2006), 335–352.

[11] CHANDRA, T. D., AND TOUEG, S. Unreliable failure detectors for reliable distributed systems. *JACM 43*, 2 (1996), 225–267.

[12] CHAWATHE, Y., RATNASAMY, S., BRESLAU, L., LANHAM, N., AND SHENKER, S. Making gnutella-like p2p systems scalable. In *ATAPCC* (2003), ACM, pp. 407–418.

[13] CHEEMA, A. S., MUHAMMAD, M., AND GUPTA, I. Peer-to-peer discovery of computational resources for grid applications. In *The 6th IEEE/ACM International Workshop on Grid Computing* (2005), IEEE, pp. 7–pp.

[14] DE MEO, P., MESSINA, F., ROSACI, D., AND SARNÉ, G. M. An agent-oriented, trust-aware approach to improve the qos in dynamic grid federations. *Concurrency and Computation : Practice and Experience 27*, 17 (2015), 5411–5435.

[15] FITZGERALD, S., FOSTER, I., KESSELMAN, C., VON LASZEWSKI, G., SMITH, W., AND TUECKE, S. A directory service for configuring high-performance distributed computations. In *The Sixth IEEE HPDC* (1997), IEEE, pp. 365–375.

[16] FOSTER, I. Globus toolkit version 4 : Software for service-oriented systems. In *Network and parallel computing*. Springer, 2005, pp. 2–13.

[17] FOSTER, I., AND IAMNITCHI, A. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Peer-to-Peer Systems II*. Springer, 2003, pp. 118–128.

[18] FOSTER, I., KESSELMAN, C., AND TUECKE, S. The anatomy of the grid : Enabling scalable virtual organizations. *IJHPCA 15*, 3 (2001), 200–222.

[19] FOSTER, I., KISHIMOTO, H., SAVVA, A., BERRY, D., DJAOUI, A., GRIMSHAW, A., HORN, B., MACIEL, F., SIEBENLIST, F., SUBRAMANIAM, J. TREADWELL, R., ET AL. The open grid services architecture, version 1.0 (2005).

[20] GUEYE, B., FLAUZAC, O., RABAT, C., AND NIANG, I. P2P4GS : A Specification for Services management in Peer-to-Peer Grids. In *INFOCOMP* (2014), IARIA XPS, pp. 41–46.

[21] GUEYE, B., FLAUZAC, O., RABAT, C., AND NIANG, I. A self–adaptive structuring for P2P-based Grids. In *14th IEEE I4CS* (2014), pp. 121–128.

[22] HUAN, W., AND NAKAZATO, H. Failure detection in p2p-grid system. *IEICE Transactions on Information and Systems 98*, 12 (2015), 2123–2131.

[23] IAMNITCHI, A., AND FOSTER, I. On fully decentralized resource discovery in grid environments. In *Grid Computing ?* Springer, 2001, pp. 51–62.

[24] IAMNITCHI, A., AND FOSTER, I. A peer-to-peer approach to resource location in grid environments. In *Grid resource management*. Springer, 2004, pp. 413–429.

[25] IAMNITCHI, A., FOSTER, I., AND NURMI, D. A peer-to-peer approach to resource discovery in grid environments. In *IEEE HPDC* (2002).

[26] JEANVOINE, E., AND MORIN, C. Rw-ogs : an optimized randomwalk protocol for resource discovery in large scale dynamic grids. In *9th IEEE/ACM International Conference on Grid Computing* (2008), pp. 168–175.

[27] JEYABHARATHI, C., AND ANNAMALAI, P. New approaches with chord in efficient p2p grid resource discovery. *CoRR 4*, abs/1401.2008 (2014), 1.

[28] KAUR, D., AND SENGUPTA, J. Resource discovery in web-services based grids. *World Academy of Science, Engineering and Technology 31* (2007), 284–288.

[29] KOVVUR, R., KADAPPA, V., RAMACHANDRAM, S., AND GOVARDHAN, A. Adaptive resource discovery models and resource selection in grids. In *PDGC* (2010), IEEE, pp. 95–100.

[30] LAVINIA, A., DOBRE, C., POP, F., AND CRISTEA, V. A failure detection system for large scale distributed systems. In *CISIS* (2010), IEEE, pp. 482–489.

[31] LYNDEN, S., MUKHERJEE, A., HUME, A. C., FERNANDES, A. A., PATON, N. W., SAKELLARIOU, R., AND WATSON, P. The design and implementation of ogsa-dqp : A service-based distributed query processor. *Future Generation Computer Systems 25*, 3 (2009), 224–236.

[32] MA, T., SHI, S., CAO, H., TIAN, W., AND WANG, J. Review on grid resource discovery : Models and strategies. *IETE Technical Review 29*, 3 (2012), 213–222.

[33] MASTROIANNI, C., TALIA, D., AND VERTA, O. A super-peer model for building resource discovery services in grids : Design and simulation analysis. In *Advances in Grid Computing-EGC 2005*. Springer, 2005, pp. 132–143.

[34] MAYMOUNKOV, P., AND MAZIERES, D. Kademlia : A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[35] NAVIMIPOUR, N. J., RAHMANI, A. M., NAVIN, A. H., AND HOSSEINZADEH, M. Resource discovery mechanisms in grid systems : A survey. *Journal of Network and Computer Applications 41* (2014), 389–410.

[36] PUPPIN, D., MONCELLI, S., BARAGLIA, R., TONELLOTTO, N., AND SILVESTRI, F. A grid information service based on peer-to-peer. In *Euro-Par 2005 Parallel Processing*. Springer, 2005, pp. 454–464.

[37] RAHMEH, O. A., JOHNSON, P., AND TALEB-BENDIAB, A. A dynamic biased random sampling scheme for scalable and reliable grid networks. *INFOCOMP Journal of Computer Science 7*, 4 (2008), 1–10.

[38] ROWSTRON, A., AND DRUSCHEL, P. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*.

[39] SCHOLLMEIER, R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing* (2001), pp. 101–102.

[40] SEYMOUR, K., YARKHAN, A., AGRAWAL, S., AND DONGARRA, J. Netsolve : Grid enabling scientific computing environments. *Advances in Parallel Computing 14* (2005), 33–51.

[41] TALIA, D., TRUNFIO, P., AND ZENG, J. Peer-to-peer models for resource discovery in large-scale grids : a scalable architecture. In *HPCCS-VECPAR*. Springer, 2007, pp. 66–78.

[42] TAN, Y.-H., LÜ, K., AND LIN, Y.-P. Organisation and management of shared documents in super-peer networks based semantic hierarchical cluster trees. *Peer-to-Peer Networking and Applications 5*, 3 (2012), 292–308.

[43] TANAKA, Y., NAKADA, H., SEKIGUCHI, S., SUZUMURA, T., AND MATSUOKA, S. Ninf-G : A reference implementation of RPC–based programming middleware for grid computing. *Journal of Grid computing 1*, 1 (2003), 41–51.

[44] TORKESTANI, J. A. A distributed resource discovery algorithm for p2p grids. *Journal of Network and Computer Applications 35*, 6 (2012), 2028 – 2036.

[45] TRUNFIO, P., TALIA, D., PAPADAKIS, H., FRAGOPOULOU, P., MORDACCHINI, M., PENNANEN, M., POPOV, K., VLASSOV, V., AND HARIDI, S. Peer-to-peer resource discovery in grids : Models and systems. *Future Generation Computer Systems 23*, 7 (2007), 864–878.

[46] TUECKE, S., CZAJKOWSKI, C., FOSTER, I., FREY, J., GRAHAM, S., KESSELMAN, C., VANDERBILT, P., AND SNELLING, D. Grid service specification–draft 11/4/02. ogsi working group, global grid forum, 2002.