Algorithmique itérative pour l'équilibrage de charge dans les réseaux dynamiques

Flavien Vernier

N° d'ordre 1046
Année 2004

THÈSE

Présentée à

Université de Franche-Comté
UFR Sciences et Techniques
Laboratoire d'Informatique de l'université Franche-Comté

Pour obtenir le

GRADE DE DOCTEUR DE L'UNIVERSITÉ DE FRANCHE-COMTÉ
Spécialité Informatique Automatique et Productique

Algorithmique itérative pour l'équilibrage de charge dans les réseaux dynamiques

par
Flavien VERNIER

Soutenu le 03 décembre 2004 devant la commission d'examen :

Rapporteurs	Bernard Toursel	Professeur à l'Université de Lille-1
	Christian Michel	Professeur à l'Université Louis Pasteur Strasbourg
Directeur de Thèse	Jacques Bahi	Professeur à l'Université de Franche-Comté
Co-encadrant	Raphaël Couturier	Maître de conférences à l'Université de Franche-Comté
Examinateurs	Noufissa Mikou	Professeur à l'Université de Bourgogne
	Michel Tréhel	Professeur émérite à l'Université de Franche-Comté

Remerciements

Tout au long de ces trois années de thèse, certaines personnes m'ont soutenu et aidé. Je tiens à les remercier tout particulièrement.

Je voudrais tout d'abord te remercier Jacques, pour avoir dirigé cette thèse. Merci pour ta disponibilité et ton soutien permanent. Pour ces heures de recherche de preuve et tous les travaux réalisés ensemble.

Merci à toi Raphaël, toi sans qui mon travail de thèse n'aurait pas aussi bien évolué, merci pour ta confiance et la pertinence de tes conseils et pour ta patience au cours de ces interminables heures de programmation et surtout de débugage.

Je tiens également à remercier Bernard Toursel et Christian Michel de m'avoir fait l'honneur d'accepter d'être rapporteurs de cette thèse, ainsi que Noufissa Mikou et Michel Tréhel pour avoir accepté d'être membres de mon jury.

Merci à toi Kamel pour toutes ces poses "Kawa" tellement agréables, ces moments de détente et de discussion. Pour les travaux que nous avons réalisés ensemble, mais surtout pour ton amitié.

Merci à tous les membres de l'équipe pour votre accueil, votre disponibilité et pour cette ambiance de travail toujours joviale.

Merci à tous mes proches, famille et amis, qui m'ont soutenu et encouragé tout au long de ces trois années de thèse.

Enfin, merci à toi Sylviane, pour tout le temps que tu as consacré à relire et corriger ce mémoire, mais surtout merci de m'avoir offert le plus beau cadeau du monde : notre fille Mandoline.

Table des matières

Table des figures

Liste des tableaux

Notations

Pour l'ensemble de la thèse les notations suivantes ont été respectées. Elles sont regroupées selon le thème auquel elles se rapportent. Les différents thèmes considérés sont les graphes, les réseaux, l'équilibrage de charge et les notions purement mathématiques.

Notations relatives aux graphes

G :
: est un graphe représentant de manière ensembliste un réseau statique. G est composé des deux ensembles V et E ($G = (V, E)$).

V :
: est l'ensemble des nœuds d'un graphe, donc l'ensemble des processeurs du réseau représenté.

E :
: est l'ensemble des arcs d'un graphe, donc l'ensemble des liens de communication du réseau représenté.

n :
: correspond au nombre de nœuds (processeurs) dans un graphe ($|V| = n$).

m :
: correspond au nombre d'arcs (liens de communication) dans un graphe ($|E| = m$).

(i, j) :
: représente l'arc liant les nœuds i et j.

$d(i)$:
: est le degré du nœud i, soit son nombre de voisins.

$d(G)$:
est le degré du graphe G, soit le degré maximal de ses nœuds ($d(G) = \max_i(d(i)), i \in V$)

$G^{(t)}$:
est un graphe représentant de manière ensembliste un réseau dynamique. $G^{(t)}$ est composé des trois ensembles V, E et $E_B^{(t)}$ ($G^{(t)} = (V, E, E_B^{(t)})$).

$E_B^{(t)}$:
est l'ensemble des arcs cassés à l'instant t, donc l'ensemble des liens de communication inutilisables à l'instant t.

c_i :
est le poids du nœud i, soit la puissance de calcul du processeur i.

f_i :
est le poids de l'arc i, soit les caractéristiques du lien de communication i.

$G(t, t + L)$:
représente le graphe de communication superposé entre les instants t et $t + L$. Ce graphe est composé de tous les nœuds du graphe G et de tous les arcs utilisés pour l'équilibrage de charge entre les instants t et $t + L$.

Notations relatives aux réseaux

D :
est le nombre de dimensions d'un hypercube, ces dimensions sont numérotées de 1 à D

d :
est une dimension d'un hypercube ($1 \leq d \leq D$)

$k_1 \times k_2 \times \ldots k_n$:
définit une grille ou un tore. n donne le nombre de dimensions et k_i le nombre de nœuds dans la dimension i ; ainsi la grille (ou le tore) 4×3 est définie sur deux dimensions avec $k_1 = 4$ et $k_2 = 3$; elle comporte donc 12 nœuds.

$n_1 \times n_2 \times \ldots$:
idem $k_1 \times k_2 \times \ldots k_n$.

Notations relatives à l'équilibrage

$w_i^{(t)}$:
représente la charge du processeur (nœud) i à l'instant t ; cette charge est par définition positive ou nulle.

$W^{(t)}$:
est le vecteur des charges de chaque processeur (nœud) du système à l'instant t ; notons que $W^{(t)}(i) = w_i^{(t)}$.

$\alpha_{(ij)}$:
est un paramètre d'échange dans le cas d'algorithmes d'équilibrage de charge de type diffusion ; il représente une fraction de la différence de charge entre les processeurs i et j.

α_{opt} :
est la valeur optimale de $\alpha_{(ij)}$ lorsque $\alpha_{(ij)}$ est constant pour tout i et tout j ; cette valeur donne la vitesse de convergence maximal à l'algorithme d'équilibrage de charge.

λ :
est le paramètre d'échange dans le cas d'algorithmes d'équilibrage de charge de type "Dimension Exchange".

λ_{opt} :
est la valeur optimale de λ ; cette valeur donne la vitesse de convergence maximale à l'algorithme d'équilibrage de charge.

M :
représente la matrice de diffusion dans le contexte des réseaux statiques.

m_{ij} :
correspond à la valeur de $M(i, j)$ et est égale à $\alpha_{(ij)}$ si i et j sont différents.

$M^{(t)}$:
représente la matrice de diffusion à l'instant t dans le contexte des réseaux dynamiques.

$m_{ij}^{(t)}$:
correspond à la valeur de $M^{(t)}(i, j)$ et est égale à $\alpha_{(ij)}$ si i et j sont différents et que l'arc (i, j) n'appartient pas à l'ensemble $E_B^{(t)}$.

M_d :
correspond à la matrice de diffusion relative à la dimension d d'un hypercube dans le cas d'algorithmes d'équilibrage de charge de type "Dimension Exchange".

c :
représente une couleur, dans les algorithmes de type "Dimension Exchange" pour réseaux quelconques ; chaque arc possède une couleur qui correspond à une dimension.

k :
est le nombre de couleurs utilisées dans un graphe coloré.

M_c :
correspond à la matrice de diffusion relative à la couleur (dimension) c dans le cas d'algorithmes de type "Dimension Exchange".

$l_{ij}^{(t)}$:
représente la charge échangée entre les processeurs i et j à l'instant t ; cette valeur est positive si j envoie de la charge à i, négative dans le cas inverse.

β :
est un paramètre de pondération utilisé dans les algorithmes de relaxation et de second ordre ; sa valeur est toujours comprise entre 0 et 2.

β_{opt} :
correspond à la valeur optimale de β.

$\beta_{max}^{(t)}$:
correspond à la valeur maximale de β à l'instant t ; cette valeur permet de ne pas générer de charge négative sur les nœuds.

$\beta_{opt}^{(t)}$:
est le paramètre de relaxation optimal à l'instant t ; il n'intervient que dans le contexte des réseaux dynamiques.

$\beta_{Che}^{(t)}$:
est la valeur optimale de β à l'instant t pour l'algorithme de Chebyshev.

$\beta^{(t)}$:
correspond à la valeur de β à l'instant t ; cette valeur est définie comme le minimum entre β_{opt} ou $\beta_{opt}^{(t)}$ et $\beta_{max}^{(t)}$

$R^{(t)}$:
est la fonction qui définit $\beta_{max}^{(t)}$ dans l'algorithme de diffusion relaxée.

$\eta_i^{(t+1)}$:
est la quantité de charge qui apparaît sur le processeur i à l'instant $t+1$; ceci n'intervient que dans le contexte d'équilibrage de charge dynamique où la quantité de charge globale du système peut varier.

c :
est la charge consommée par un processeur ; cette valeur est constante dans le temps pour un processeur donné et correspond à sa puissance de calcul ; elle n'intervient que dans le contexte d'équilibrage de charge dynamique.

Notations mathématiques

s :
représente la plus petite valeur propre de M.

l :
représente la seconde plus grande valeur propre de M.

$s^{(t)}$:
représente la plus petite valeur propre de $M^{(t)}$.

$l^{(t)}$:
représente la seconde plus grande valeur propre de $M^{(t)}$.

γ :
représente la plus grande valeur propre différente de 1 et en valeur absolue d'une matrice de diffusion.

μ_i :
est une valeur propre de M ; les μ_i sont ordonnées de la manière suivante : $1 = \mu_1 > \mu_2 > \cdots > \mu_n > -1$. Notons que $\mu_2 = l$ et que $\mu_n = s$.

L :
est le laplacien du graphe G.

$\widetilde{L}$:
est le laplacien généralisé du graphe G ; il tient compte de l'hétérogénéité du graphe, ce que ne fait pas L.

λ_i :
est une valeur propre de $\widetilde{L}$; les λ_i sont ordonnées de la manière suivante : $0 = \lambda_1 < \lambda_2 < \cdots < \lambda_n$.

Introduction

Avec l'évolution rapide des capacités de traitement des stations de travail et le déploiement massif de clusters ou autres réseaux de stations, le calcul distribué a grandement évolué au cours de cette dernière décennie. Étant toujours à la recherche de plus de puissance, les utilisateurs du calcul distribué se sont tout d'abord tournés vers les réseaux de stations, plus facilement évolutifs et meilleur marché que les machines parallèles dédiées. Un nouveau pas a été franchi ces dernières années, toujours dans le but d'accroître la puissance disponible pour réaliser du calcul distribué : le monde de la recherche s'intéresse aux réseaux de clusters, au "peer to peer" et à n'importe quel moyen d'utiliser toute puissance de calcul accessible depuis Internet. Ainsi, différentes méthodes de coopération se sont développées : GRID computing, Meta computing, "peer to peer" ... Dans ces nouveaux contextes d'exécution, différents problèmes d'hétérogénéité sont apparus, tant au niveau des machines utilisées qu'au niveau du réseau reliant ces machines. De nombreuses bibliothèques de communications telles que PM2 [59], Heterogeneous MPI [5] et CORBA, ou des systèmes plus complexes permettant le déploiement et la résolution de problèmes, comme Globus [36], NetSolve [1], Ninf [58] ou encore DIET [24], ont été développés afin de répondre aux nouvelles contraintes induites par ces contextes d'exécution.

Les applications développées sur ces systèmes sont de deux catégories. Soit elles sont centralisées de type clients-serveur : une machine - le serveur - centralise les requêtes des différents clients. Soit elles sont distribuées (ou réparties) : chaque machine du système ne travaille qu'avec ses voisins directs. Pour notre part, nous nous focalisons sur les systèmes distribués, et plus particulièrement sur les problèmes que peut engendrer l'hétérogénéité. La recherche d'un maximum de puissance dans les systèmes distribués passe nécessairement par une répartition ou un équilibrage de charge efficace. Ceci est d'autant plus vrai lorsqu'on se trouve dans un contexte hétérogène. Travailler avec un grand nombre de machines perd de l'intérêt si le système est mal équilibré, car une partie de la puissance totale disponible est inutilisée. L'équilibrage de charge a pour but de tirer le meilleur parti des différentes machines coopératives.

Dans le domaine de l'équilibrage (ou répartition) de charge, différentes approches

existent et de nombreuses études ont été réalisées. Au-delà des études, les systèmes d'équilibrage de charge sont utilisés dans différents domaines. Google, pour ne citer que lui, utilise un système de répartition de charge pour traiter au mieux les requêtes qui lui sont soumises : une requête entrante est redirigée vers un des serveurs les moins chargés, et la résolution de la requête elle-même est effectuée en parallèle avec un système implémentant de l'équilibrage de charge. La plupart des serveurs web qui traitent un grand nombre de requêtes fonctionnent également sur ce principe. L'équilibrage de charge peut également intervenir au niveau réseau. Dans ce cas, il gère la transmission des paquets pour ne pas surcharger un lien de communication et utiliser au mieux le réseau. De manière générale, l'équilibrage de charge est utilisé dans toute application ou architecture distribuée et intervient aussi bien au niveau des processus [53, 47, 46, 48, 65] qu'au niveau des données. L'enjeu de l'équilibrage de charge est d'obtenir des performances optimales pour le système qu'il gère.

D'un point de vue algorithmique, il existe dans la littérature deux grands domaines : les algorithmes d'équilibrage de charge centralisés et ceux décentralisés. Dans le premier cas, la centralisation permet une connaissance globale des différents systèmes coopératifs. Ceci facilite grandement la répartition puisqu'on sait exactement où la charge doit être placée. Le choix de la charge à déplacer n'en reste pas moins un problème non trivial. Dans [17] les auteurs s'intéressent à ce problème dans le contexte des clusters hétérogènes. Dans le second cas - les algorithmes d'équilibrage de charge décentralisés - chaque système coopératif ne connaît que les caractéristiques de ses voisins directs. Sous ces conditions, les algorithmes utilisés sont itératifs et équilibrent de proche en proche à chaque pas de temps. Ce type d'algorithmes d'équilibrage de charge a été introduit par Cybenko dans [25]. Dans cet article, il présente deux algorithmes, la diffusion et "Dimension Exchange", qui ont été largement étudiés [56] et enrichis ou dérivés par la suite [38, 43, 49, 2]. C. Flament [35] a considéré des approches de type diffusion dans le domaine des sciences sociales et la propagation des idées. D'autres algorithmes d'équilibrage de charge distribués sont étudiés selon une approche différente dans [42], ou selon des contextes particuliers [39, 40, 41]. Dans [57, 68] les auteurs proposent des classifications de ces algorithmes.

Pour notre part, nous nous intéressons aux algorithmes itératifs d'équilibrage de charge distribués dans un contexte particulier, celui des réseaux dynamiques. Pour définir de manière succincte un réseau dynamique, il s'agit d'un réseau dans lequel certains liens de communication peuvent être temporairement perdus ou surchargés. Afin de rester le plus générique possible, nous ne définissons pas la notion de charge, car elle se définit en fonction du système à équilibrer (la charge peut être définie comme un nombre de processus, comme une quantité de données ou peut avoir une autre définition). Ceci permet d'appliquer les algorithmes étudiés à tout système

nécessitant de l'équilibrage de charge. Avant cette thèse, seuls les auteurs de [16] se sont intéressés à ce domaine particulier de l'équilibrage de charge. Ce travail pose des contraintes fortes à l'algorithme d'équilibrage, nous l'avons donc repris et modifié afin que les contraintes imposées par l'algorithme soit en concordance avec une application réelle. Nous l'avons également élargi à d'autres algorithmes. Une étude très récente, relative à ce contexte particulier de l'équilibrage de charge, proposée par Monien et al. dans [33], vient confirmer nos résultats.

Ce mémoire de thèse comporte trois parties : tout d'abord, nous nous intéressons aux algorithmes de premier ordre, puis à ceux de second ordre, et pour finir, nous présentons une application d'un algorithme d'équilibrage de charge sur un problème concret.

Algorithmes de premier ordre

La première partie de ce mémoire est consacrée exclusivement aux algorithmes d'équilibrage de charge de premier ordre. On parle d'algorithme de premier ordre, dans le domaine de l'équilibrage de charge, lorsqu'un algorithme n'utilise que la répartition de charge courante du système pour définir les transferts à effectuer. Cette première partie est composée de trois chapitres : le premier présente, pour les réseaux statiques, les algorithmes de premier ordre existants, et introduit un nouvel algorithme. Le second développe, pour ces algorithmes, notre travail sur réseaux dynamiques. Et le dernier illustre le comportement de ces algorithmes par des simulations sur réseaux statiques et dynamiques.

Chapitre 1 : Ce premier chapitre présente les différents algorithmes existants sur lesquels nous travaillons, et en introduit un nouveau que nous nommons : la diffusion relaxée. Ces algorithmes sont tous dérivés de la diffusion de premier ordre introduite par Cybenko dans [25]. Le nouvel algorithme que nous proposons a pour but d'accélérer la convergence de la diffusion classique. Ce chapitre s'intéresse également aux contextes les plus généraux, c'est à dire aux charges dynamiques et aux réseaux hétérogènes. On entend par charge dynamique la charge globale du système ; cela signifie que la charge globale du système à équilibrer peut varier au court du temps : des quantités de charge peuvent disparaître ou apparaître lors de l'équilibrage. Par opposition, on parle de charge statique lorsque la charge globale du système reste la même tout au long de l'exécution de l'algorithme d'équilibrage. Quant à l'hétérogénéité du réseau, elle influe sur les paramètres de l'algorithme d'équilibrage. Dans la suite du document nous nous restreignons à une charge statique et à des réseaux homogènes, ceci dans l'unique but de simplifier les algorithmes. Nous verrons que

leur adaptation à un contexte plus général est triviale.

Chapitre 2 : Dans un second temps, nous abordons le problème des réseaux dynamiques. Nous définissons dans ce chapitre la notion de réseaux dynamiques et nous présentons pour chaque algorithme vu précédemment son adaptation à ce type de réseaux. Le point le plus important de ce chapitre est la preuve de la convergence de ces algorithmes sur réseaux dynamiques. Ces algorithmes ont été modifiés et convergent sous certaines conditions qui sont cohérentes avec la réalité. Ces adaptations ont été publiées dans [12, 11, 10] et l'article [13], qui récapitule tous nos travaux sur les algorithmes de premier ordre, est en soumission.

Chapitre 3 : Pour finir cette partie, différentes simulations illustrent le comportement des algorithmes sur plusieurs types de réseaux classiques. Ces simulations sont réalisées avec une charge système virtuelle afin de montrer l'évolution des algorithmes sans interaction avec un système réel à équilibrer. Elles permettent de mettre en évidence l'influence du dynamisme et du type de réseaux sur la convergence des algorithmes.

Diffusion de second ordre

La seconde partie s'intéresse aux algorithmes de diffusion de second ordre. Contrairement à ceux étudiés en première partie, ces algorithmes déterminent les transferts de charge à effectuer en fonction de la répartition de charge courante et de celle précédente. Ces algorithmes utilisent une mémoire afin de conserver l'état du système à l'itération précédente. Il en résulte généralement une vitesse de convergence nettement supérieure à celle obtenue avec un algorithme de premier ordre. Cette partie comporte deux chapitres. Le premier rappelle les algorithmes de second ordre sur réseaux statiques et introduit une nouvelle contrainte sur la détermination de leurs paramètres. Le second présente l'application de ces algorithmes sur les réseaux dynamiques.

Chapitre 4 : De la même manière que pour la première partie, nous commençons par présenter la modélisation des algorithmes que nous étudions. Les deux algorithmes auxquels nous nous intéressons sont la diffusion de second ordre et la méthode de Chebyshev, qui est une variante du premier. Ces algorithmes sont introduits dans [44]. Nous apportons à ces deux algorithmes une contrainte qui n'est présentée dans aucun des articles trouvés. Ces algorithmes convergent naturellement vers une répartition uniforme de la charge, mais ne garantissent pas la positivité de la charge d'un nœud à tout instant. Or il est inconcevable qu'un processeur ait une charge négative. Cette contrainte non négligeable a

donc été ajoutée à la détermination des paramètres de ces algorithmes. Ce chapitre se termine par différentes simulations qui illustrent le comportement de ces algorithmes sur réseaux statiques.

Chapitre 5 : L'application de ces algorithmes aux réseaux dynamiques est présentée dans ce chapitre. Nous donnons les différentes modifications apportées à ces algorithmes afin qu'ils prennent en compte le dynamisme du réseau. Ces modifications en entraînent d'autres sur la contrainte de positivité de la charge ; nous donnons donc également, pour les réseaux dynamiques, les contraintes sur la détermination de paramètres de second ordre. Nous montrons expérimentalement que les deux algorithmes de second ordre que nous étudions convergent vers l'équilibre malgré le dynamisme du réseau. La preuve théorique est nettement plus complexe que dans le cas d'algorithmes de premier ordre et nous ne sommes, à l'heure actuelle, pas parvenus à la finaliser. Pour finir, différentes simulations sur réseaux statiques ou dynamiques illustrent, comme pour les algorithmes de premier ordre, le comportement et l'influence du dynamisme sur ces algorithmes.

De la Théorie à la Pratique

Nous nous sommes jusqu'à présent restreints à une étude théorique et des simulations sans charge réelle des algorithmes d'équilibrage de charge. Cette dernière partie étudie la mise en pratique d'un algorithme d'équilibrage de charge sur une application réelle. L'application répartie que nous cherchons à équilibrer est la résolution d'une équation différentielle partielle (EDP) modélisant le problème d'advection-diffusion qui correspond à un problème de cinétique chimique. Dans les parties précédentes, nous utilisons toujours une notion abstraite de la charge et nous ne l'avons jamais définie de manière concrète, mais cette mise en pratique nous oblige à la définir concrètement en fonction de l'application. Cette dernière partie est composée de deux chapitres : le premier présente l'application d'advection-diffusion et la mise en pratique d'un algorithme d'équilibrage de charge sur un réseau statique. Le second s'attache à la mise en place de l'application avec équilibrage de charge sur un réseau dynamique.

Chapitre 6 : Dans un premier temps, nous présentons l'application d'advection-diffusion dans une version séquentielle et nous donnons sa modélisation, nécessaire à son implantation. Suite à ceci, nous détaillons le choix de l'algorithme d'équilibrage de charge à utiliser en fonction des différentes contraintes induites par l'application. Afin d'appliquer l'algorithme d'équilibrage choisi sur l'implantation d'advection-diffusion, nous définissons la charge de l'applica-

tion ainsi que la puissance de calcul disponible sur chaque processeur. Ces définitions sont réalisées de la manière la plus simple possible pour ne pas ralentir l'application. Suite à ces définitions et au choix de l'algorithme à utiliser, plusieurs expérimentations sont réalisées sur des réseaux statiques. Ces différents points mettent en évidence les problèmes rencontrés lors de la mise en pratique d'un algorithme d'équilibrage de charge.

Chapitre 7 : Dans un second temps, nous appliquons ce problème sur un réseau dynamique. Ceci implique différentes modifications au niveau du problème lui-même, ainsi qu'au niveau de l'équilibrage de charge. La version du problème utilisée précédemment est synchrone, ce qui n'est pas compatible avec le dynamisme du réseau. Une version asynchrone du problème a donc été utilisée pour ce type de réseaux. Nous montrons également que notre définition de la charge ne fonctionne pas avec une exécution asynchrone, elle est donc redéfinie en conséquence. Suite à ces modifications nous déployons plusieurs expériences qui présentent concrètement l'apport de l'équilibrage de charge pour cette application.

Les différents résultats obtenus au cours de cette thèse vont nous permettre de conclure sur les contributions que nous apportons au domaine de l'équilibrage de charge. Nous dégageons aussi les apports des algorithmes d'équilibrage de charge sur réseaux dynamiques. Enfin, nous présentons les différents axes d'étude que nous ouvrons afin d'élargir ce travail.

Première partie

Algorithmes de Premier Ordre

Chapitre 1

Algorithmes pour Réseaux Statiques

1.1 Introduction

Un des problèmes les plus importants dans l'élaboration de programmes parallèles est la répartition de la charge de travail sur les différentes machines coopératives. L'objectif des méthodes de répartition de charge est de tirer le plus grand parti de la puissance de calcul disponible. De nombreuses méthodes (ou algorithmes) ont déjà été étudiées depuis plusieurs années : allocations statiques ou dynamiques, algorithmes itératifs... Dans notre cas nous nous restreignons aux algorithmes itératifs d'équilibrage de charge. Ce genre d'algorithmes, adapté à l'équilibrage de charge, a été introduit par George Cybenko dans [25]. Ces algorithmes ont l'avantage d'être totalement distribués et n'opèrent que de proches en proches. De manière générale, un algorithme itératif d'équilibrage de charge effectue, par itérations successives, des transferts de charge entre les différents processeurs voisins afin de converger vers l'équilibre des charges du système.

Dans ce chapitre, nous nous intéressons aux algorithmes de premier ordre. Ceux-ci calculent les transferts de charge à effectuer uniquement en fonction de la répartition de charge courante. Dans la section 1.2, nous présentons les algorithmes de premier ordre existants dans la littérature. Tout d'abord, l'algorithme de diffusion est présenté sous sa forme générale pour une charge dynamique puis sous sa forme restreinte à une charge statique. On considère que la charge d'un système est statique lorsque la charge globale du système à équilibrer est la même à chaque instant ; à l'inverse, s'il y a consommation et création de charge au cours de l'évolution du système, on parle de charge dynamique. Nous présentons par la suite, les algorithmes de type "Dimension Exchange". Ce type d'algorithmes est une variante

de l'algorithme de diffusion, la différence tenant au fait que, dans le cas de "Dimension Exchange", l'équilibrage s'effectue par paires de processeurs voisins. Nous présentons plus particulièrement deux algorithmes : "Dimension Exchange" classique (DE) dédié aux hypercubes et "Generalized Dimension Exchange" (GDE) adapté à tout type de graphe. Pour ces trois algorithmes, nous donnons les paramètres optimaux et nous proposons deux exemples illustrant le choix de ces paramètres. Pour finir cette section nous nous intéressons à l'application de ces algorithmes sur des réseaux hétérogènes et à la méthode générale pour déterminer les paramètres de diffusion optimaux. Ce chapitre ce termine par la section 1.3 dans laquelle nous introduisons un nouvel algorithme : la diffusion relaxée. Cet algorithme est une variante de la diffusion de premier ordre, dans ce dernier, nous accélérons la vitesse d'équilibrage de la diffusion par un système de relaxation.

1.2 État de l'art : diffusion de premier ordre

Les algorithmes itératifs d'équilibrage de charge ont été introduits par George Cybenko dans [25]. Dans cet article, l'auteur propose un algorithme itératif pour l'équilibrage de charge : la diffusion de premier ordre que nous appelons par abus de langage "diffusion". Cet algorithme est à la base dédié aux réseaux statiques et homogènes.

L'algorithme de diffusion effectue, par itérations successives, des transferts de charge entre les différents processeurs pour converger vers un équilibre uniforme des charges du systèmes.

1.2.1 Algorithme de diffusion de premier ordre

Afin de présenter les différents algorithmes d'équilibrage de charge, nous devons formaliser la représentation d'un réseau. Nous considérons que le réseau est homogène, connecté et non-bipartite et que les liens de communication sont bidirectionnels. Classiquement, la topologie d'un réseau est représentée par un graphe $G = (V, E)$ avec V l'ensemble des nœuds et E l'ensemble des arcs du réseau. Notons que E est un sous-ensemble de $V \times V$. Chaque processeur du réseau est représenté par un nœud dans le graphe et tout lien de communication entre deux processeurs i et j est représenté par l'arc (i, j) appartenant à E. Par définition, chaque nœud est numéroté entre 1 et n, ainsi nous avons $|V| = n$ et nous posons que $|E| = m$.

Afin d'exprimer l'algorithme de diffusion, nous notons par $w_i^{(t)}$ la charge du processeur i à l'instant t. Nous considérons dans un premier temps, que cette charge $w_i^{(t)}$ est infiniment divisible : elle est modélisée par un réel positif. Cette contrainte

sur la charge est généralement utilisée pour l'étude des algorithmes d'équilibrage de charge. Dans une application pratique, la charge est toujours entière et positive. Avec ces définitions, l'algorithme de diffusion s'écrit de la manière suivante :

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j \alpha_{ij}(w_j^{(t)} - w_i^{(t)}) + \eta_i^{(t+1)} - c, \tag{1.1}$$

où α_{ij} est une constante positive, $\eta_i^{(t+1)}$ est la charge de travail qui est créée sur le processeur i à l'instant t et c est une constante représentant la charge de travail consommée par un processeur en une itération. Notons que nous sommes dans un cadre homogène, donc chaque processeur effectue la même quantité de travail c en une itération.

Il est aisé de remarquer que la charge de travail échangé $l_{ij}^{(t)}$ entre deux processeurs i et j voisins est donnée par :

$$l_{ij}^{(t)} = \alpha_{ij}(w_j^{(t)} - w_i^{(t)}).$$

La charge échangée est donc une fraction α_{ij} de la différence de charge entre i et j. On notera que pour cet algorithme $\alpha_{ij} = \alpha_{ji}$. De ces remarques découlent différentes contraintes sur les constantes α_{ij} : α_{ij} est compris entre 0 inclus et 1 inclus ; α_{ij} est égal à 0 si et seulement si l'arc (i, j) n'existe pas dans E. $\sum_j \alpha_{ij}$ est compris entre 0 exclu et 1 inclus ; en d'autres termes, un nœud i est toujours connecté à au moins un nœud j et il ne peut pas donner plus de charge qu'il n'en possède.

Cet algorithme opère avec une charge dynamique, c'est à dire que la charge global du système ($\sum_i w_i^{(t)}$) évolue au cours du temps (il y a consommation et génération de charge) [52]. Afin d'analyser l'évolution de l'équation 1.1, nous devons la simplifier et l'exprimer pour une charge statique : charge qui sera constante entre chaque itération ($\sum_i w_i^{(t)} = \sum_i w_i^{(t+1)}$). Le modèle pour charge statique s'exprime simplement en considérant que $\eta_i^{(t+1)} = 0$ pour tout instant t et $c = 0$. Dans ce cas, l'équation 1.1 s'écrit :

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j \alpha_{ij}(w_j^{(t)} - w_i^{(t)}). \tag{1.2}$$

En regroupant les termes $w_i^{(t)}$, nous obtenons l'équation suivante :

$$w_i^{(t+1)} = (1 - \sum_j \alpha_{ij})w_i^{(t)} + \sum_j \alpha_{ij} w_j^{(t)}, \tag{1.3}$$

ce qui permet de mettre en évidence le fait que l'équation 1.2 est linéaire. La mise à jour de tout le système peut alors s'écrire sous la forme d'une équation vectorielle :

$$W^{(t+1)} = MW^{(t)}, \tag{1.4}$$

où $W^{(t)}$ est le vecteur de dimension n contenant la charge de tous les processeurs à l'instant t et M est une matrice que nous nommons matrice de diffusion définie par m_{ij} telle que :

$$m_{ij} = \begin{cases} \alpha_{ij} & \text{si } i \neq j, \\ 1 - \sum_j \alpha_{ij} & \text{si } i = j. \end{cases}$$

Notons que $\sum_i m_{ij} = 1$, de plus $m_{ij} = m_{ji}$, la matrice M est donc symétrique et doublement stochastique. Il est prouvé dans [25] que cet algorithme converge vers une répartition uniforme W^* de la charge telle que :

$$w_i^* = \frac{\sum_{j=1}^{n} w_j^{(0)}}{n}.$$

Notons que pour le reste de ce document, nous nous restreignons à une charge statique afin de simplifier l'étude des algorithmes.

1.2.2 Détermination de la matrice de diffusion

Le point le plus important dans l'algorithme de diffusion est la détermination de la matrice de diffusion M. C'est elle qui définie la vitesse de convergence de l'algorithme, c'est à dire la vitesse à laquelle l'algorithme équilibre la charge du système. De nombreuses études montrent que la matrice de diffusion optimale dépend intégralement de la topologie du réseau sur lequel l'équilibrage est appliqué. Dans la littérature on trouve deux principales approches afin de déterminer les valeurs de α_{ij} (donc M), l'approche globale [74] qui nécessite une connaissance globale du réseau, c'est à dire une connaissance des valeurs propres du Laplacien du réseau considéré, et l'approche locale [21], [60] qui détermine α_{ij} uniquement en fonction du degré du nœud i et des degrés de ses voisins j, le degré d'un nœud étant son nombre de voisins.

L'approche locale, nommée ADF (Average Diffusion) ou choix de Boillat [21], définit α_{ij} comme

$$\alpha_{ij} = \frac{1}{\max\left(d(i), d(j)\right) + 1}, \tag{1.5}$$

où $d(i)$ est le degré du nœud i. Dans le cas de graphes réguliers, tel un tore ou un hypercube, le degré de chaque nœud est identique ; l'expression de α_{ij} peut donc se simplifier par

$$\alpha_{ij} = \frac{1}{d(G) + 1}, \tag{1.6}$$

où $d(G)$ est le degré du graphe G ($d(G) = \max_i d(i)$). Ce choix de α_{ij} est également appelé choix de Cybenko [25]. En ce qui concerne les graphes quasis réguliers, telle

une grille de grande taille, il est fréquent de considérer que le graphe est régulier et d'affecter à tout α_{ij} la valeur $\frac{1}{d(G)+1}$. Cette considération a pour but de simplifier la détermination de la matrice de diffusion M. Cette approche locale a l'avantage d'être simple et facile à mettre en place.

L'approche globale quant à elle, nécessite une étude préalable du réseau considéré, elle est plus précisément appelée ODF (Optimally-tuned Diffusion). Cette méthode consiste à choisir α_{ij} constant ($\alpha_{ij} = \alpha$) tel que cette valeur maximise la vitesse de convergence de l'algorithme tout en conservant les caractéristiques de M. La vitesse de convergence est déterminée en fonction de la seconde valeur propre γ de M et est égale à $-\ln\gamma$ [69] ; le calcul de α s'effectue en minimisant γ. On notera que cette méthode n'est pas directe et nécessite une nouvelle étude pour chaque graphe considéré. Néanmoins, différentes études ont été réalisées pour des graphes classiques ainsi :

$$\begin{array}{ll} \alpha = \frac{1}{2} & \text{pour une ligne} \\ \alpha = \frac{1}{2n} & \text{pour une grille } k_1 \times k_2 \times \ldots k_n \\ \alpha = \frac{1}{2n+1-\cos(\frac{2\pi}{k})} & \text{pour un tore } k_1 \times k_2 \times \ldots k_n \text{ avec } k = \max k_i \\ \alpha = \frac{1}{n+1} & \text{pour un hypercube de dimension } n \end{array} \tag{1.7}$$

Ces deux méthodes ont leurs propres avantages : ADF est simple à mettre en place et garde un caractère distribué, et ODF fournie une valeur optimale pour α.

1.2.3 Illustration du comportement de l'algorithme de diffusion

Des simulations complètes de la diffusion sur différents types de réseaux sont présentées dans le chapitre 3.

Considérons tout d'abord le graphe quelconque donné par la figure 1.1. Selon les

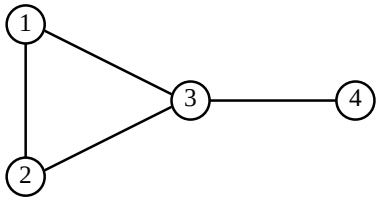


FIG. 1.1 – Graphe quelconque.

méthodes présentées dans la section précédente, la matrice de diffusion M peut être

$$\begin{bmatrix} 5/12 & 1/3 & 1/4 & 0 \\ 1/3 & 5/12 & 1/4 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 1/4 & 3/4 \end{bmatrix} \qquad \begin{bmatrix} 1/2 & 1/4 & 1/4 & 0 \\ 1/4 & 1/2 & 1/4 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 1/4 & 3/4 \end{bmatrix}$$

(a) choix de Boillat. (b) choix de Cybenko.

TAB. 1.1 – Matrices de diffusion relatives au graphe 1.1.

déterminée en fonction du degré de chaque nœud, en fonction du degré du graphe ou de manière optimale. Selon le degré des nœuds, M est définie par

$$\begin{cases} m_{ij} = \frac{1}{\max(d(i),d(j))+1} & \text{pour } i \neq j, \\ m_{ii} = \sum_{j \neq i} m_{ij}. \end{cases}$$

Ce qui donne M présentée dans la table 1.1(a). En ce qui concerne le choix de Cybenko, selon le degré du graphe, M est définie par

$$\begin{cases} m_{ij} = \frac{1}{d(G)+1} & \text{pour } i \neq j, \\ m_{ii} = \sum_{j \neq i} m_{ij}. \end{cases}$$

Dans ce cas, M est donnée par la table 1.1(b). Pour la méthode optimale, la détermination de M nécessite une étude complète du graphe. Cette étude n'ayant pas d'intérêt particulier dans notre cas, M optimale n'est pas donnée pour le graphe 1.1.

Considérons dans un second cas le graphe régulier (grille 3x3) donné par la figure 1.2. De même que pour le graphe quelconque, la matrice de diffusion peut être déterminée selon trois stratégies. la matrice M, dans le cas d'une grille 3x3 et selon le choix de Boillat, est donnée dans la table 1.2(a). De manière simplifiée, c'est à dire en fonction du degré du graphe, M est présentée dans la table 1.2(b). Le graphe étudié étant standard, la valeur optimale de α est connue. Dans notre cas, une grille $2d$, α optimal est $1/4$, la matrice de diffusion optimale peut ainsi être déterminée, elle est donnée dans la table 1.3.

On peut remarquer que la détermination des α_{ij} est plus ou moins évidente selon les cas considérés. Les cas les plus simples sont lorsque α_{ij} est constant : choix de Cybenko ou choix optimal. Néanmoins, ces deux cas nécessitent des informations globales, ce qui peut être contraire au caractère distribué de l'algorithme. Le choix de la méthode à utiliser pour déterminer les α_{ij} dépend donc des informations connues avant l'exécution de l'algorithme.

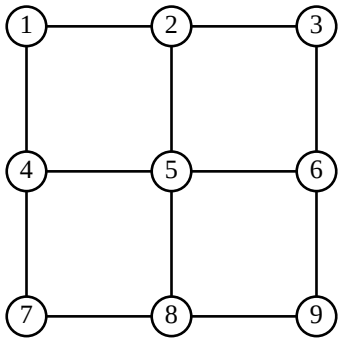


FIG. 1.2 – Grille 3x3.

1.2.4 Algorithme de dimension exchange

Les algorithmes de type "Dimension Exchange" sont dérivés de l'algorithme de diffusion. Ils sont basés sur des communications uni-ports, c'est à dire qu'un nœud i ne peut s'équilibrer qu'avec un seul de ses voisins j à un instant t donné.

L'algorithme "Dimention Exchange" (DE), spécifique aux hypercubes, a été proposé par G. Cybenko dans [25]. Il montre que cet algorithme est une application particulière de la diffusion. A la différence de la diffusion, on considère qu'un processeur ne peut communiquer qu'avec un seul de ses voisins à la fois et qu'il effectue l'équilibrage voisin après voisin. L'algorithme n'équilibrant que deux par deux les processeurs, α_{ij} est choisi tel que l'équilibre entre les paires de processeurs s'effectue de manière uniforme :

$$\alpha_{ij} = \frac{1}{2}.$$

Sous ces conditions, chaque processeur peut communiquer simultanément sur une seule dimension de l'hypercube à un instant t donné. L'algorithme DE s'écrit donc :

$$w_i^{(t+1)} = w_i^{(t)} + \frac{1}{2}\left(w_j^{(t)} - w_i^{(t)}\right), \tag{1.8}$$

où j est voisin de i sur une dimension d donnée. De même que pour la diffusion, la mise à jour de la charge de tout le système peut s'écrire sous forme vectorielle

$$W^{(t+1)} = M_d W^{(t)}, \tag{1.9}$$

où M_d est la matrice de diffusion pour la dimension d de l'hypercube. La valeur de d est choisie en fonction de t de manière à faire le cycle des D dimensions. Considérons que les dimensions de l'hypercube sont numérotées de 1 à D ($1 \leq d \leq D$), d est défini par

$$d = t \bmod D + 1$$

$$\begin{bmatrix}
1/2 & 1/4 & \ddots & 1/4 & & & & & \\
1/4 & 3/10 & 1/4 & \ddots & 1/5 & & & 0 & \\
\ddots & 1/4 & 1/2 & 0 & \ddots & 1/4 & & & \\
1/4 & \ddots & 0 & 3/10 & 1/5 & 0 & 1/4 & & \\
 & 1/5 & \ddots & 1/5 & 1/5 & 1/5 & \ddots & 1/5 & \\
 & & 1/4 & 0 & 1/5 & 3/10 & 0 & \ddots & 1/4 \\
 & & & 1/4 & \ddots & 0 & 1/2 & 1/4 & \ddots \\
 & 0 & & & 1/5 & \ddots & 1/4 & 3/10 & 1/4 \\
 & & & & & 1/4 & \ddots & 1/4 & 1/2
\end{bmatrix}$$

(a) choix de Boillat.

$$\begin{bmatrix}
3/5 & 1/5 & \ddots & 1/5 & & & & & \\
1/5 & 2/5 & 1/5 & \ddots & 1/5 & & & 0 & \\
\ddots & 1/5 & 3/5 & 0 & \ddots & 1/5 & & & \\
1/5 & \ddots & 0 & 2/5 & 1/5 & 0 & 1/5 & & \\
 & 1/5 & \ddots & 1/5 & 1/5 & 1/5 & \ddots & 1/5 & \\
 & & 1/5 & 0 & 1/5 & 2/5 & 0 & \ddots & 1/5 \\
 & & & 1/5 & \ddots & 0 & 3/5 & 1/5 & \ddots \\
 & 0 & & & 1/5 & \ddots & 1/5 & 2/5 & 1/5 \\
 & & & & & 1/5 & \ddots & 1/5 & 3/5
\end{bmatrix}$$

(b) choix de Cybenko.

TAB. 1.2 – Matrices de diffusion relatives au graphe 1.2.

Cet algorithme étant limité aux hypercubes, il est plus fréquent de considérer sa forme généralisée.

$$\begin{bmatrix}
1/2 & 1/4 & \ddots & 1/4 & & & & & \\
1/4 & 1/4 & 1/4 & \ddots & 1/4 & & & 0 & \\
\ddots & 1/4 & 0 & 0 & \ddots & 1/4 & & & \\
1/4 & \ddots & 0 & 1/4 & 1/4 & 0 & 1/4 & & \\
 & 1/4 & \ddots & 1/4 & 0 & 1/4 & \ddots & 1/4 & \\
 & & 1/4 & 0 & 1/4 & 1/4 & 0 & \ddots & 1/4 \\
 & & & 1/4 & \ddots & 0 & 1/2 & 1/4 & \ddots \\
 & 0 & & & 1/4 & \ddots & 1/4 & 1/4 & 1/4 \\
 & & & & & 1/4 & \ddots & 1/4 & 1/2
\end{bmatrix}$$

TAB. 1.3 – Matrice de diffusion relative au graphe 1.2 avec un choix optimal.

1.2.5 Algorithme de dimension exchange généralisé

L'algorithme "Generalized Dimension Exchange" (GDE) est l'application de DE sur réseaux quelconques [50],[67]. Dans un premier temps, Hosseini et al. dans [50] proposent d'utiliser un graphe coloré afin de représenter les différentes dimensions sur un graphe quelconque. Dans un second temps, Xu et Lau dans [67] utilisent un paramètre λ à l'image de α_{ij} pour la diffusion. Ainsi, pour un graphe $G = (V, E)$ chaque arc (i, j) est associé à une couleur de sorte que pour un nœud donné, tous ses arcs aient une couleur différente. Le réseau est donc représenté par le graphe coloré $G_k = (V, E_k)$ avec E_k l'ensemble des triplets $(i, j; c)$ où i et j sont les sommets de l'arc (i, j) et c la couleur associée à l'arc. Notons que les couleurs sont numérotées de 0 à $k-1$, où k est le nombre de couleurs du graphe. Si $d(i)$ représente le degré du nœud i et $d(G)$ le degré du graphe ($d(G) = \max_i d(i)$), il est prouvé que le nombre minimum de couleurs k utiles à G_k est $d(G) \leq k \leq (d(G) + 1)$ [34].

Le graphe ainsi défini, l'algorithme GDE pour un nœud i donné s'exécute suivant l'équation (1.10).

$$\begin{array}{rcll} w_i^{(t+1)} & = & w_i^{(t)} + \lambda \left(w_j^{(t)} - w_i^{(t)} \right) & \text{si } \exists j | (i, j; c) \in E_k \wedge c = t \bmod k, \\ & = & w_i^{(t)} & \text{sinon,} \end{array} \tag{1.10}$$

avec λ le paramètre de transfert. Notons que dans le cas de DE, $\lambda = \frac{1}{2}$. La mise à jour de la charge de tout le système s'effectue comme pour DE :

$$W^{(t+1)} = M_c W^{(t)}, \tag{1.11}$$

où M_c est la matrice de diffusion pour la couleur (dimension) c. La valeur de c est choisie en fonction de t, de manière à faire le cycle des k couleurs. Les couleurs étant numérotées de 0 à $k-1$ ($0 \leq c \leq (k-1)$), c est défini par

$$c = t \bmod k$$

1.2.6 Détermination de λ

Comme dans l'algorithme de diffusion, le point le plus important dans GDE est la détermination de λ. λ définit la vitesse de convergence de l'algorithme, donc la vitesse à laquelle l'algorithme équilibre la charge du système. On trouve dans la littérature deux méthodes afin de déterminer λ : la méthode innée et la méthode optimale.

La première méthode est basée sur une simple observation : l'algorithme équilibre les processeurs deux à deux, λ est donc choisi de manière à équilibrer les paires de processeurs uniformément.

$$\lambda = \frac{1}{2}.$$

Ce choix de λ est celui utilisé dans DE.

La méthode optimale est basée sur une étude approfondie de GDE est du réseau à équilibrer. Considérons un cycle d'équilibrage de GDE, en d'autres termes l'application successive de $M_0, M_1, \ldots M_{k-1}$. La mise à jour du système suite à un cycle se note :

$$W^{(t+k)} = M_{k-1} \times M_{k-2} \times \cdots \times M_0 W^{(t)} = M(\lambda) W^{(t)},$$

où $M(\lambda)$ est la matrice GDE. $M(\lambda)$ correspond à la matrice de diffusion M de l'algorithme de diffusion. La détermination de λ optimal consiste à choisir sa valeur telle que la vitesse de convergence soit maximale. La vitesse de convergence est déterminée en fonction de la seconde valeur propre γ de $M(\lambda)$ et est égale à $-\ln \gamma$ [69] ; le calcul de λ s'effectue en minimisant γ. On notera que cette méthode nécessite une nouvelle étude pour chaque graphe considéré. Néanmoins, différentes études [67, 70] ont été réalisées pour des graphes classiques ainsi :

$$\begin{array}{ll} \lambda = \frac{1}{2} & \text{pour un hypercube,} \\ \lambda = \frac{2-\sqrt{2(1-\cos(\frac{2\pi}{n}))}}{1+\cos(\frac{2\pi}{n})} & \begin{cases} \text{pour un tore } 2n_1 \times 2n_2 \text{ avec } n = \max n_1, n_2, \\ \text{pour un anneau de dimension } 2n, \\ \text{pour une grille } n_1 \times n_2 \text{ avec } n = \max n_1, n_2, \\ \text{pour une ligne de dimension } n. \end{cases} \end{array} \tag{1.12}$$

Ces deux méthodes ont leurs propres avantages : la première donne une valeur simple et efficace de λ, et la seconde fournit une valeur optimale pour λ.

1.2.7 Illustration du comportement de l'algorithme GDE

Reprenons les graphes étudiés en section 1.2.3 afin de déterminer les matrices de diffusion pour chaque couleur. Dans le cas de GDE le réseau considéré doit être représenté par un graphe coloré. Une coloration possible est donnée par le graphe 1.3 Ce graphe contient trois couleurs (0), (1) et (2), ce qui nous donne les trois matrices

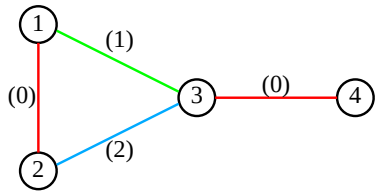


FIG. 1.3 – Graphe quelconque coloré.

M_0, M_1 et M_2 présentée dans la table 1.4 pour le choix inné. En ce qui concerne le

$$M_0 \begin{bmatrix} .5 & .5 & 0 & 0 \\ .5 & .5 & 0 & 0 \\ 0 & 0 & .5 & .5 \\ 0 & 0 & .5 & .5 \end{bmatrix} \quad M_1 \begin{bmatrix} .5 & 0 & .5 & 0 \\ 0 & 1 & 0 & 0 \\ .5 & 0 & .5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M_2 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & .5 & .5 & 0 \\ 0 & .5 & .5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

TAB. 1.4 – Matrices de diffusion relatives au graphe coloré 1.3.

choix optimal de λ, une étude approfondie est nécessaire. Cette étude n'ayant pas d'intérêt particulier dans notre cas, nous ne donnons pas les matrices optimales de diffusion pour le graphe quelconque.

Considérons à présent le cas de la grille 3x3. Comme précédemment nous devons définir une coloration de la grille. Ce choix étant arbitraire, considérons le graphe coloré 1.4 pour représenter notre réseau. Pour la grille considérée, le nombre minimum de couleurs est de 4, ce qui donne 4 matrices. A titre d'exemple, nous ne donnons que la matrice relative à la première dimension (couleur 0), les autres étant facilement déterminable. Ainsi dans le cas inné, la matrice de diffusion pour la couleur 0 est représentée dans la table 1.5(a). Dans le cas optimal, nous savons que pour une grille $n_1 \times n_2$, λ est donné par :

$$\lambda = \frac{2 - \sqrt{2(1 - \cos(\frac{2\pi}{n}))}}{1 + \cos(\frac{2\pi}{n})}$$

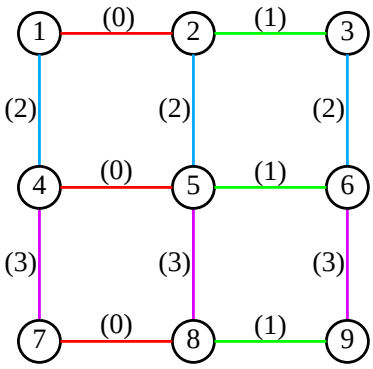


FIG. 1.4 – Grille 3x3 coloré.

avec $n = \max(n_1, n_2)$, soit 3 dans notre cas. Ce qui donne $\lambda = 0.5359$ La matrice de diffusion pour la couleur 0 est donnée dans la table 1.5(b).

Comme pour les algorithmes étudiés précédemment, la détermination des paramètres d'équilibrage dépend entièrement des informations dont nous disposons sur le graphe. Des simulations de GDE sur différents types de réseaux sont présentées dans le chapitre 3.

1.2.8 Diffusion sur réseaux hétérogènes

L'application des algorithmes itératifs d'équilibrage de charge aux réseaux hétérogènes a été étudiée dans différents papiers [30, 31, 62, 54]. La méthode proposée est générique et peut s'appliquer à tous les algorithmes vus dans ce chapitre.

Commençons par définir un réseau hétérogène. Un réseau hétérogène est un réseaux pour lequel les machines n'ont pas la même puissance de calcul et les liens de communication ont des caractéristiques différentes. Ce dernier se présente, d'un point de vue graphe, comme un réseau homogène. Le graphe représentatif est donc $G = (V, E)$ avec V l'ensemble des nœuds du réseau et E l'ensemble des arcs du réseau avec $|V| = n$ et $|E| = m$. L'hétérogénéité du réseau est introduite par la pondération des arcs et des nœuds. Les poids représentent, de manière relative, les caractéristiques d'un lien réseau et la puissance d'un nœud de calcul. Ainsi, les vecteurs C et F sont introduits et représentent respectivement les poids des nœuds et les poids des arcs. Notons que $|C| = n$ et que $|F| = m$. Chaque nœud se voit associer un poids c_i et chaque arc (i, j) un poids f_{ij}, avec $c_i, f_{ij} \in \mathbb{R}^+$. f_{ij} représente plus précisément le coût de la communication sur l'arc (i, j). Si f_{ij} est faible, ceci indique que l'arc (i, j) a un débit élevé.

L'étude du problème d'hétérogénéité pour les algorithmes de diffusion de premier

$$\begin{bmatrix} .5 & .5 & & & & & & & \\ .5 & .5 & 0 & & & & & 0 & \\ & 0 & 1 & 0 & & & & & \\ & & 0 & .5 & .5 & & & & \\ & & & .5 & .5 & 0 & & & \\ & & & & 0 & 1 & 0 & & \\ & & & & & 0 & .5 & .5 & \\ & 0 & & & & & .5 & .5 & 0 \\ & & & & & & & 0 & 1 \end{bmatrix}$$

(a) choix de inné.

$$\begin{bmatrix} 0.4641 & 0.5359 & & & & & & & & \\ 0.5359 & 0.4641 & 0 & & & & & & 0 & \\ & 0 & 1 & 0 & & & & & & \\ & & 0 & 0.4641 & 0.5359 & & & & & \\ & & & 0.5359 & 0.4641 & 0 & & & & \\ & & & & 0 & 1 & 0 & & & \\ & & & & & 0 & 0.4641 & 0.5359 & & \\ & 0 & & & & & 0.5359 & 0.4641 & 0 & \\ & & & & & & & 0 & 1 & \end{bmatrix}$$

(b) choix de optimal.

TAB. 1.5 – Matrices de diffusion relatives au graphe coloré 1.4.

ordre est proposé dans [31, 62]. Dans ces papiers, les auteurs modifient l'algorithme de telle sorte que l'équilibre soit relatif aux poids des nœuds :

$$w_i^* = \frac{\sum_{i=1}^n w_i}{\sum_{i=1}^n c_i} c_i.$$

Afin d'atteindre cet équilibre relatif, ils pondèrent la charge d'un nœud par le poids de ce nœud. La charge échangée entre deux processeurs i et j est donnée par

$$l_{ij}^{(t)} = \alpha_{ij} \left(\frac{w_i^{(t)}}{c_i} - \frac{w_j^{(t)}}{c_j} \right). \tag{1.13}$$

La modélisation 1.13 permet d'atteindre un équilibre relatif, mais ne prend pas en compte l'hétérogénéité des liens de communication. Afin de favoriser les liens les

plus rapides, les poids f_{ij} des arcs sont appliqués au coefficient de transfert α_{ij}. La forme généralisée de l'algorithme de diffusion 1.2 est donc :

$$\begin{array}{lcl} l_{ij}^{(t)} & = & \frac{\alpha_{ij}}{f_{ij}} \left(\frac{w_i^{(t)}}{c_i} - \frac{w_j^{(t)}}{c_j} \right) \\ \text{et} & & \\ w_i^{(t+1)} & = & w_i^{(t)} - \sum_j l_{ij}^{(t)}. \end{array} \tag{1.14}$$

L'expression vectorielle de la mise à jour de tout le système nécessite une représentation plus générale de l'algorithme de diffusion vu en section 1.2.1. Considérons l'échange de charge entre i et j, dans le cas homogène, sous la forme

$$l_{ij}^{(t)} = \alpha_{ji} w_i^{(t)} - \alpha_{ij} w_j^{(t)},$$

avec $\alpha_{ji} = \alpha_{ij}$. Ainsi, dans le cas hétérogène, l'équation 1.14 se récrit

$$w_i^{(t+1)} = w_i^{(t)} - \sum_j \left(\frac{\alpha_{ji}}{c_i f_{ij}} w_i^{(t)} - \frac{\alpha_{ij}}{c_j f_{ij}} w_j^{(t)} \right).$$

Il est simple de voir à présent, que la mise à jour du système est donnée par

$$W^{(t+1)} = M W^{(t)}$$

avec

$$m_{ij} = \begin{cases} \frac{\alpha_{ij}}{c_j f_{ij}} & \text{si } i \neq j \\ 1 - \sum_j \frac{\alpha_{ji}}{c_i f_{ij}} & \text{si } i = j \end{cases}$$

Dans un cas hétérogène, la matrice de diffusion M n'est pas doublement stochastique mais simplement stochastique colonne. Elle respecte toutefois la contrainte suivante : $m_{ij} c_j = m_{ji} c_i$ pour tout couple (i, j).

1.2.9 Détermination de α optimal

Dans le cas de réseaux hétérogènes, la détermination de α optimal nécessite, comme pour le cas homogène, une étude complète du réseau [31]. La différence étant que pour les topologies classiques, il n'existe pas d'étude donnant α optimal. Nous donnons donc dans cette section la méthode générale de détermination de α optimal.

Tout d'abord, considérons que pour tout couple de voisins (i, j), α_{ij} est constant et est égal à α. Posons $\mathcal{C}$ la matrice carrée $n \times n$ composée des c_i sur la diagonale et 0 ailleurs et A la matrice d'incidence nœud-arc du graphe G. A est définie telle que

$A \in \{-1, 0, 1\}^{n \times m}$. Chaque colonne comporte uniquement deux entrées non nulles "1" et "-1"pour les deux nœuds relatifs à l'arc. Le signe défini le sens de l'arc mais n'a pas grande importance. Soit B la matrice d'adjacence du graphe ($B \in \{0, 1\}^{n \times n}$). Chaque entrée (i, j) de B est à 1 si i est voisin de j ou 0 sinon ; le graphe n'étant pas orienté, B est symétrique. Notons pour finir $\mathcal{F}$, la matrice diagonale $m \times m$ avec $\mathcal{F}_{ee} = \sqrt{f_e}$ avec e l'arc (i, j), et $\widetilde{L}$ le laplacien généralisé du graphe où $\widetilde{L} = \widetilde{A}\widetilde{A}^T\mathcal{C}^{-1}$ avec $\widetilde{A} = A\mathcal{F}^{-1}$.

La vitesse de convergence de l'algorithme de diffusion est fonction de la plus grande valeur propre γ, différente de 1 et en valeur absolue, de la matrice de diffusion : $\gamma = \max|\mu_2|, |\mu_n|$ avec μ_2 et μ_n respectivement la seconde plus grande et la plus petite valeur propre de M ($1 = \mu_1 > \mu_2 > \cdots > \mu_n > -1$). Sachant que $M = I - \alpha\widetilde{L}$, nous avons $\mu_2 = 1 - \alpha\lambda_2$ et $\mu_n = 1 - \alpha\lambda_n$, avec λ_2 et λ_n respectivement la seconde plus petite et la plus grande valeur propre du laplacien généralisé ($0 = \lambda_1 < \lambda_2 < \cdots < \lambda_n$). La vitesse de convergence étant $-\ln\gamma$ [69], sa valeur maximale est atteinte pour une valeur minimum de γ, on en déduit donc que la valeur optimale de α est donnée par :

$$\alpha_{opt} = \frac{2}{\lambda_2 + \lambda_n}.$$

α_{opt} ainsi défini ne garanti pas la positivité de la diagonale de M, Il est aisé de voire que $m_{ii} \geq 0$ si $\alpha \leq \frac{1}{\widetilde{L}_{ii}}$. α est donc choisi tel que :

$$\alpha = \min_i \left(\alpha_{opt}, \frac{1}{\widetilde{L}_{ii}} \right)$$

Cette méthode est générique et s'applique à tous les algorithmes vus précédemment. Notons que pour GDE, la matrice de diffusion nécessaire à la détermination de α_{opt} (λ_{opt}) est la matrice $M(\lambda)$. Dans le cas homogène, $\mathcal{C}$ et $\mathcal{F}$ sont l'identité. Le laplacien généralisé est donc simplement L, où $L = AA^T$.

Afin de faciliter la compréhension, reprenons notre exemple de graphe quelconque 1.1 et pondérons le de manière arbitraire (cf. 1.5). Commençons par donner les diverses matrices nécessaires au calcul de α_{opt} ($\mathcal{C}$, A, B et $\mathcal{F}$).

$$\mathcal{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

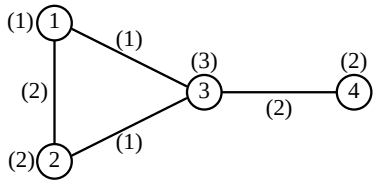


FIG. 1.5 – Graphe quelconque pondéré.

$$B = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathcal{F} = \begin{bmatrix} \sqrt{2} & 0 & 0 & 0 \\ 0 & \sqrt{1} & 0 & 0 \\ 0 & 0 & \sqrt{1} & 0 \\ 0 & 0 & 0 & \sqrt{2} \end{bmatrix}$$

Ces matrices nous permettent de déterminer le laplacien généralisé $\widetilde{L}$.

$$\widetilde{A} = A\mathcal{F}^{-1} = \begin{bmatrix} 1/\sqrt{2} & 1 & 0 & 0 \\ -1/\sqrt{2} & 0 & 1 & 0 \\ 0 & -1 & -1 & 1/\sqrt{2} \\ 0 & 0 & 0 & -1/\sqrt{2} \end{bmatrix}$$

$$\widetilde{L} = \widetilde{A}\widetilde{A}^T\mathcal{C}^{-1} = \begin{bmatrix} 1.5 & -0.25 & -1/3 & 0 \\ -0.5 & 0.75 & -1/3 & 0 \\ -1 & -0.5 & 5/6 & -0.25 \\ 0 & 0 & -1/6 & 0.25 \end{bmatrix}$$

On en déduit les valeurs propres suivantes :

$$\begin{bmatrix} 1.851 & 0 & 1.176 & 0.306 \end{bmatrix},$$

et la valeur optimale de α :

$$\alpha_{opt} = 0.927.$$

Comme indiqué précédemment, cette valeur de α ne garantit pas la positivité de M. Dans notre cas, grâce à $\widetilde{L}$, on déduit que $\min_i \left(\frac{1}{\widetilde{L}_{ii}}\right) = 0.666$. La valeur considérée comme optimale est donc :

$$\alpha = 0.666.$$

1.3 Contribution : la diffusion relaxée

Nous proposons dans cette section un nouvel algorithme de premier ordre basé sur la diffusion, nommé diffusion relaxée. Il permet l'accélération par relaxation de la vitesse d'équilibrage de la diffusion.

1.3.1 Algorithme de diffusion relaxée

Le principe de ce nouvel algorithme est d'intégrer un paramètre de relaxation β positif dans l'algorithme de diffusion (1.2), afin d'accélérer la convergence de ce dernier. La diffusion de type relaxée s'écrit :

$$w_i^{(t+1)} = w_i^{(t)} + \beta \sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) \tag{1.15}$$

Par manipulation de l'équation 1.15, on obtient :

$$\begin{aligned} w_i^{(t+1)} &= (1-\beta)\, w_i^{(t)} + \beta w_i^{(t)} + \beta \textstyle\sum_j \alpha_{ij} w_j^{(t)} - \beta \sum_j \alpha_{ij} w_i^{(t)} \\ &= (1-\beta)\, w_i^{(t)} + \beta \left(1 - \textstyle\sum_j \alpha_{ij} \right) w_i^{(t)} + \beta \sum_j \alpha_{ij} w_j^{(t)} \\ &= (1-\beta)\, w_i^{(t)} + \beta \left[\left(1 - \textstyle\sum_j \alpha_{ij} \right) w_i^{(t)} + \sum_j \alpha_{ij} w_j^{(t)} \right], \end{aligned} \tag{1.16}$$

qui fait ressortir, en partie, l'expression de la diffusion. L'équation 1.16 peut donc se récrire sous la forme vectorielle :

$$W^{(t+1)} = (1-\beta)\, W^{(t)} + \beta M W^{(t)}. \tag{1.17}$$

La convergence d'un tel algorithme est un résultat connu, toutefois il est évident que β doit être correctement choisi afin d'accélérer la convergence et de respecter les contraintes sur la charge. La première condition est satisfaite si β est supérieur à 1. La seconde se limite à une seule contrainte à satisfaire : la positivité de la charge ; une charge $w_i^{(t+1)}$ négative n'a pas de sens. β doit donc être choisi, pour un i donné, tel que :

$$w_i^{(t)} + \beta \sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) \geq 0.$$

Il en découle que :

$$\beta \sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) \geq -w_i^{(t)}$$

Trois cas sont donc possibles :

– $\sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) > 0$
Dans ce cas :

$$\beta \sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) \geq -w_i^{(t)}$$
$$\beta \geq \frac{-w_i^{(t)}}{\sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right)}$$

Ceci implique que β doit être supérieur à une valeur négative, or β est défini positif.

– $\sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) = 0$
Dans ce cas, la positivité de $w_i^{(t+1)}$ ne dépend pas de β, puisque sous ces conditions : $w_i^{(t+1)} = w_i^{(t)}$.

– $\sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) < 0$
Dans ce cas :

$$\beta \sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) \geq -w_i^{(t)}$$
$$\beta \leq \frac{w_i^{(t)}}{\sum_j \alpha_{ij} \left(w_i^{(t)} - w_j^{(t)} \right)}$$
$$\beta \leq \frac{w_i^{(t)}}{\sum_j \alpha_{ij} \left(w_i^{(t)} - \min_j \left(w_j^{(t)} \right) \right)} \qquad \text{par minoration de } w_j^{(t)}$$
$$\beta \leq \frac{w_i^{(t)}}{(1 - M_{ii}) \left(w_i^{(t)} - \min_j \left(w_j^{(t)} \right) \right)}$$
$$\beta \leq \frac{w_i^{(t)}}{(1 - M_{ii}) \left(w_i^{(t)} - w_{min}^{(t)} \right)} \qquad \text{avec } w_{min}^{(t)} = \min_j \left(w_j^{(t)} \right)$$

Cette borne supérieure est donc la seule que nous considérons, les deux cas précédents n'intervenant pas dans le choix de β. De plus, nous remarquons que cette borne dépend du temps. Jusqu'à présent, nous nous sommes restreints à un nœud i donné, pour la détermination de β. Pour le réseau complet, il en découle naturellement que :

$$\beta \leq \min_i \frac{w_i^{(t)}}{(1 - M_{ii}) \left(w_i^{(t)} - w_{min}^{(t)} \right)} \quad \forall i | \sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) < 0. \tag{1.18}$$

Posons :

$$R^{(t)} = \min_i \frac{w_i^{(t)}}{(1 - M_{ii}) \left(w_i^{(t)} - w_{min}^{(t)} \right)} \quad \forall i | \sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) < 0.$$

La fonction $R^{(t)}$ étant croissante et monotone, une borne déterminée à l'instant t est toujours valable à tout instant $t + n$ ($n \in N^+$). Une valeur de β déterminée à l'instant 0 est donc correcte à tout autre instant.

Remarque 1 *Ceci n'est vrai que dans le cas de charge statique. Dans un contexte de charge dynamique, $R^{(t)}$ n'est pas monotone, β devient donc fonction du temps et doit être recalculé à chaque itération.*

Notons pour finir, que si $\beta = 1$, la diffusion relaxée est équivalente à la diffusion de premier ordre.

1.3.2 Détermination du paramètre de relaxation β optimal

Nous avons donné précédemment la borne supérieure de β afin que $w^{(t+1)}$ reste positif. Le paramètre de relaxation définissant l'accélération que l'on donne à l'algorithme de diffusion, il doit être choisi de manière optimale. β est défini positif, mais il est connu que β doit être supérieur à 1 pour accélérer la diffusion.

La détermination de β optimal (β_{opt}) pour un algorithme de relaxation et sous nos conditions est définie dans [18]. Si M est une matrice stochastique, le paramètre optimal de relaxation est

$$\beta_{opt} = \frac{2}{2 - (s + l)},$$

où s et l sont respectivement la plus petite et la seconde plus grande valeur propre de M.

Cette valeur optimale de β ne respecte pas forcément les conditions définies précédemment ($1 \leq \beta \leq R^{(t)}$). La borne supérieure de β dépendant du temps, notons donc $\beta_{opt}^{(t)}$ la valeur optimale de β pour chaque instant t. $\beta_{opt}^{(t)}$ est ainsi défini par

$$\beta_{opt}^{(t)} = \min\left(R^{(t)}, \frac{2}{2 - (s + l)}\right) \tag{1.19}$$

Notons pour finir, avec $\beta_{opt}^{(t)} = \beta_{opt}$, la vitesse de convergence de la diffusion relaxée est donnée par $-\ln \frac{l-s}{2-(s+l)}$ [18].

1.3.3 Conditions et preuve de convergence

L'algorithme de diffusion relaxée est dérivé de la diffusion de premier ordre et peut être vu comme un cas particulier de la diffusion où la matrice de diffusion est définie par

$$(1 - \beta^{(t)})Id + \beta^{(t)}M,$$

avec Id la matrice identité aux dimensions de M.

Proposition 1 *Avec β choisi selon l'équation 1.19, l'algorithme de diffusion relaxée converge.*

Preuve 1 *Il suffit d'appliquer les résultats présentés dans [18].* □

1.3.4 Illustration du comportement de l'algorithme diffusion relaxée

Reprenons les cas étudiés en section 1.2.3 : tout d'abord le graphe quelconque figure 1.1, puis la grille 3x3 figure 1.2. Déterminons, pour les différents M considérés, la valeur de β_{opt}. Tout d'abord avec le graphe quelconque et α_{ij} défini selon le degré des nœuds. Dans ce cas, les valeurs propres de M sont données dans la table 1.6(a). Ce qui donne une valeur optimale pour β_{opt} de $\frac{2}{2-0.75} = 1.6$. Cette valeur ne tient

$$\left[\begin{array}{cccc} 0 & 0.0833 & 0.75 & 1 \end{array}\right]. \qquad \left[\begin{array}{cccc} 0 & 0.25 & 0.75 & 1 \end{array}\right].$$

(a) choix de Boillat. (b) choix de Cybenko.

TAB. 1.6 – Valeurs propres de M relatives au graphe 1.1.

pas compte de la répartition de la charge. Pour le cas étudié, on obtient donc :

$$\beta_{opt}^{(t)} = \min\left(R^{(t)}, 1.6\right)$$

En ce qui concerne la détermination de M selon le degré du graphe, on obtient les valeurs propres présentées dans la table 1.6(b). On remarque que les valeurs de s et l sont identiques à celles précédentes. L'expression de $\beta_{opt}^{(t)}$ est donc la même que pour M défini selon le choix de Boillat.

Considérons à présent le cas de la grille 3x3. Nous obtenons respectivement pour les trois choix de M étudiés (Boillat, Cybenko et optimal), les valeurs propres données dans la table 1.7. Pour chaque M considéré, les valeurs de s et l sont différentes, nous avons donc une valeur de $\beta_{opt}^{(t)}$ par cas. Dans le cas du choix de Boillat :

$$\beta_{opt}^{(t)} = \min\left(R^{(t)}, 1.29\right)$$

Dans le cas du choix de Cybenko :

$$\beta_{opt}^{(t)} = \min\left(R^{(t)}, 1.43\right)$$

Et enfin dans le cas optimal :

$$\beta_{opt}^{(t)} = \min\left(R^{(t)}, 1.14\right)$$

$$\left[\begin{array}{cccccccccc} -0.316 & 0.032 & 0.032 & 0.3 & 0.316 & 0.5 & 0.767 & 0.767 & 1 \end{array}\right],$$

(a) choix de Boillat.

$$\left[\begin{array}{cccccccccc} -0.2 & 0.2 & 0.2 & 0.4 & 0.4 & 0.6 & 0.8 & 0.8 & 1 \end{array}\right],$$

(b) choix de Cybenko.

$$\left[\begin{array}{cccccccccc} -0.5 & 0 & 0 & 0.25 & 0.25 & 0.5 & 0.75 & 0.75 & 1 \end{array}\right].$$

(c) choix de optimal.

TAB. 1.7 – Valeurs propres de M relatives au graphe 1.2.

Notons pour finir, que la diffusion relaxée ne peut pas garder un caractère distribué. La détermination de β nécessite au minimum une connaissance de la répartition initiale de la charge afin de calculer $R^{(0)}$. La détermination de $\beta_{opt}^{(t)}$ quant à elle demande une étude plus poussée du réseau et de M. Des simulations sur différents types de réseaux sont présentées dans le chapitre 3.

1.4 Conclusion

Ce premier chapitre présente, dans un premier temps, différents algorithmes d'équilibrage de charge existants. Il ressort deux grands types d'algorithmes : les algorithmes de type diffusion et les algorithmes de type "Dimension Exchange". Nous montrons, via deux exemples, les difficultés qui peuvent intervenir lors de la détermination des paramètres d'équilibrage. De manière générale, la détermination de ces paramètres dépend des connaissances sur le graphe. Dans le cas de graphes quelconques, les méthodes de détermination ne nécessitant pas d'étude sont les plus simples à employer. Pour ce qui concerne les graphes classiques (grille, tore ...), les méthodes optimales, nécessitant une étude précise du graphe, peuvent facilement être utilisées du fait que les résultats de ces études sont connus. Ces algorithmes étant, à la base, dédiés aux réseaux homogènes, nous présentons dans un second temps une étude qui permet d'adapter les algorithmes de diffusion aux réseaux hétérogènes. Nous montrons pour finir, comment déterminer de manière générale les paramètres de diffusion optimaux. Ce chapitre se termine par l'introduction d'un nouvel algorithme : la diffusion relaxée. Ce dernier algorithme permet l'accélération

par relaxation de la vitesse d'équilibrage de la diffusion.

Tous les algorithmes vus dans ce chapitre sont dédiés aux réseaux statiques, homogènes ou non. C'est à dire, des réseaux pour lesquels l'architecture ne change pas. Avec l'évolution du calcul distribué et l'utilisation d'Internet comme support de communication, les liens reliant les nœuds de calcul ne sont pas fiables et l'architecture du réseau peut changer. Ces changements interviennent lorsqu'un lien est temporairement, ou non, coupé ou qu'il est surchargé ; on parlera de réseau dynamique. Ce dynamisme peut donc entraîner divers problèmes au sein des algorithmes d'équilibrage. Le chapitre suivant étudie ce problème pour les algorithmes vus précédemment.

Chapitre 2

Algorithmes pour Réseaux Dynamiques

2.1 Introduction

Dans le contexte du calcul sur grille, du "meta computing" ou de tout autre modèle utilisant Internet comme support de communication, les transferts de messages ne sont pas garantis à 100%. Certains liens de communication peuvent être temporairement ou non, perdus ou surchargés. On parle alors de réseaux dynamiques. Pour notre étude, nous nous limitons au dynamisme d'un point de vue liens de communication, un nœud du réseau ne peut ni spontanément apparaître ni définitivement disparaître au cours de l'exécution de l'algorithme. Sous ces conditions les algorithmes d'équilibrage de charge vus précédemment ne peuvent s'exécuter correctement.

Nous proposons dans cette section leur adaptation aux réseaux dynamiques, nous avons publié ces adaptations dans [12, 11, 10]. A notre connaissance, avant le début de cette thèse, aucune étude sur ce sujet n'avait été publiée. La première étude de l'application de ces algorithmes aux réseaux dynamiques est proposée dans [16], mais cette étude contraint le réseau à être infiniment souvent connecté ; une nouvelle étude très ressente est présentée dans [33], cette dernière confirme les résultats que nous présentons. A des fins de simplification et de preuve, nous nous restreignons à une charge statique du système et à des réseaux homogènes. Bien que ces algorithmes soient présentés de manière restreinte, il est facile de les généraliser en se référant au chapitre 1.

Ce chapitre est organisé de la façon suivante : une première section (2.2), introduit l'application de l'algorithme de diffusion aux réseaux dynamiques. Cette section définit de manière formelle la notion de réseaux dynamiques et présente

l'application de l'algorithme de diffusion de premier ordre sur ce dernier. Elle se termine en donnant les conditions de convergence et la preuve de cette convergence. La section suivante 2.3 se charge des algorithmes de type "Dimension Exchange", c'est à dire ceux pour lesquels l'équilibrage s'effectue par paires de processeurs. Elle présente l'adaptation de l'algorithme "Dimension Exchange" dédié aux hypercubes et GAE (Generalized Adaptative Exchange) qui est une variante de GDE pour les réseaux dynamiques. Pour ces deux algorithme, nous faisons le lien avec la preuve présentée pour la diffusion de premier ordre. Le dernier algorithme que nous traitons en section 2.4 est la diffusion relaxée, comme pour les algorithmes précédents, nous donnons son adaptation aux réseaux dynamiques et nous prouvons sa convergence sur ce type de réseaux. Pour finir ce chapitre, la section 2.5 illustre par un exemple la détermination des paramètres de diffusion.

2.2 Algorithmes de diffusion de premier ordre

Dans un premier temps, étudions l'adaptation de l'algorithme de diffusion sur les réseaux dynamiques. Cette étude a été présentée dans [11].

2.2.1 Modélisation de l'algorithme

Avant de présenter l'algorithme de diffusion de premier ordre pour réseaux dynamiques, nous devons adapter le graphe représentatif de ce dernier afin qu'il prenne en compte le dynamisme du réseau. De manière générale, un réseau est modélisé par un graphe $G = (V, E)$ où V est l'ensemble des nœuds et E est l'ensemble des arcs du réseau (cf. section 1.2.1). Le dynamisme est introduit par l'ensemble $E_B^{(t)}$ qui contient les arcs (i, j) inutilisables à l'instant t. $E_B^{(t)}$ est un sous-ensemble de E. Un réseau dynamique est donc représenté par le graphe $G^{(t)} = (V, E, E_B^{(t)})$. Notons que si $E_B^{(t)}$ est vide $G^{(t)}$ est équivalent au graphe G.

Le réseau ainsi modélisé, nous pouvons donner l'application de l'algorithme présenté section 1.2 sur les réseaux dynamiques. L'échange de charge entre deux processeurs i et j voisins est donné par l'équation 2.1.

$$l_{ij}^{(t)} = \begin{cases} \alpha_{ij}(w_j^{(t)} - w_i^{(t)}) & \text{si } (i,j) \in E \wedge (i,j) \notin E_B^{(t)}, \\ 0 & \text{sinon.} \end{cases} \tag{2.1}$$

En d'autres termes, deux processeurs i et j s'équilibrent si et seulement si ils sont voisins et le lien qui les relie est opérationnel. L'évolution de la charge d'un nœud i entre deux instants est donné par la charge du processeur i plus la somme des

charges échangées avec ses voisins (cf. équation 2.2).

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j l_{ij}^{(t)}. \tag{2.2}$$

Aux conditions près, l'équation d'échange de charge entre deux processeurs est la même que pour la diffusion sur réseaux statiques. La différence est liée au fait que l'algorithme ne s'applique pas sur un lien cassé, ceci implique que la mise à jour du système dépend de l'ensemble $E_B^{(t)}$ et donc du temps. La matrice de diffusion est donc fonction du temps. L'équation vectorielle, modélisant la mise à jour de tout le système entre deux instants t et $t+1$ s'écrit, dans le cas de réseaux dynamiques, selon l'équation 2.3.

$$W^{(t+1)} = M^{(t)} W^{(t)}, \tag{2.3}$$

où $M^{(t)}$ est la matrice de diffusion à l'instant t, définie par

$$m_{ij}^{(t)} = \begin{cases} \alpha_{ij} & \text{si } (i,j) \in E \wedge (i,j) \notin E_B^{(t)} \wedge i \neq j, \\ 1 - \sum_k \alpha_{ik} & \forall k | (i,k) \in E \wedge (i,k) \notin E_B^{(t)} \wedge i = j \\ 0 & \text{sinon.} \end{cases}$$

Notons que pour $m_{ij}^{(t)}$ ainsi défini, la matrice de diffusion $M^{(t)}$ est doublement stochastique comme M mais n'est pas forcément connectée.

La détermination des paramètres de diffusion α_{ij} s'effectue avec les mêmes méthodes que dans le cas de réseaux statiques. Par contre, la détermination de paramètres optimaux ne peut pas être réalisée avec exactitude, du fait du dynamisme du réseau. Le paramètre α_{ij} optimal est donc toujours calculé à partir du réseau sans coupure.

2.2.2 Conditions de convergence sur réseaux dynamiques

Les caractéristiques de la matrice de diffusion induites par la détermination des paramètres de diffusion (doublement stochastique, connectée et non-bipartite), suffisent pour prouver, dans le cas de réseaux statiques, la convergence de ces algorithmes. Dans le cas présent, la perte de liens de communication peut rendre la matrice de diffusion non-connectée. Ces algorithmes convergent donc sous certaines conditions. Avant de préciser ces conditions, nous devons définir la notion de graphe de communication superposé.

Définition 1 *A chaque instant, le graphe de communication dédié à l'équilibrage de charge est le graphe contenant seulement les arcs utilisés pour les transferts de charge à cet instant. Le graphe de communication superposé dédié à l'équilibrage de charge noté $G_{t,t+n}$, est le graphe qui contient uniquement les arcs utilisés pour les*

transferts de charge entre les instants t et $t+n$. Le graphe de communication superposé est une représentation imagée de la superposition des graphes de communication entre deux instants t et $t+n$.

Avec cette définition, nous pouvons donner les conditions de convergence.

Théorème 1 *L'algorithme de diffusion de premier ordre sur réseaux dynamiques converge vers une répartition uniforme de la charge si et seulement si pour tout instant t correspond un instant $t+L$ tel que le graphe de communication superposé $G_{t,t+L}$ soit connecté.*

La preuve de ce théorème se trouve dans la section suivante. Notons d'une part que ce théorème n'implique pas qu'un arc doit absolument être opérationnel à un instant t quelconque ni que tous les arcs doivent être utilisable simultanément, et d'autre part que ces conditions sont proches de la réalité.

2.2.3 Preuve de la convergence de la diffusion sur réseaux dynamiques

Lemme Technique

A l'instant t, la matrice d'incidence $I^{(t)}$ de la matrice de diffusion $M^{(t)}$ est définie par

$$I_{ij}^{(t)} = \begin{cases} 1 \text{ si } (i \text{ et } j \text{ communique à l'instant } t) \text{ ou si } (i=j) \\ 0 \text{ sinon} \end{cases}$$

Définissons la matrice d'incidence du graphe de communication superposé $G_{t,T}$ entre les instants t et T par

$$I_{ij}^{(t,T)} = I_{ij}^{(t)} + I_{ij}^{(t+1)} + ... + I_{ij}^{(T-1)} + I_{ij}^{(T)}$$

où $1+1=1$; $1+0=0+1=1$; $0+0=0$.

Lemme 1 *Pour m matrices d'incidence $I^1, ..., I^m$ nous avons*

$$I^{(1)} + I^{(2)} + ... + I^{(m)} \leq I^{(m)} \times I^{(m-1)} \times ... \times I^{(1)}$$

où nous utilisons un produit matriciel standard et où la somme de valeurs booléennes est définie précédemment.
La relation d'ordre $\leq$ entre deux matrices est un ordre partiel élément par élément.

Preuve 2 *Par induction : il est suffisant de prouver que si* $\left(I^{(k)} \times I^{(k-1)}\right)_{ij} = 0$ *alors* $\left(I^{(k)} + I^{(k-1)}\right)_{ij} = 0$. *Nous avons :* $\left(I^{(k)} \times I^{(k-1)}\right)_{ij} = 0 \Leftrightarrow \sum_{l=1}^{n} \left(I^{(k)}\right)_{il} \left(I^{(k-1)}\right)_{lj}$ $= 0 \Leftrightarrow \left(I^{(k)}\right)_{il} \left(I^{(k-1)}\right)_{lj} = 0$, *pour tout* $l \in \{1, ..., n\}$, *particulièrement pour* $l = j$ *et* $l = i$ *donc*

$$\left(I^{(k)}\right)_{ij} \left(I^{(k-1)}\right)_{jj} = 0 \quad et \quad \left(I^{(k)}\right)_{ii} \left(I^{(k-1)}\right)_{ij} = 0$$

Comme toutes les entrées diagonales d'une matrice d'incidence sont 1, *nous en déduisons que* $\left(I^{(k)}\right)_{ij} = 0$ *et* $\left(I^{(k-1)}\right)_{ij} = 0$, *ainsi* $\left(I^{(k)} + I^{k-1}\right)_{ij} = \left(I^{(k)}\right)_{ij} + \left(I^{(k-1)}\right)_{ij} = 0$ □

Lemme 2 *Si le graphe de communication superposé* $G_{t,t+m}$, *avec pour matrices de diffusion* $M^{(t)}, M^{(t+1)}, ..., M^{(t+m)}$ *est connecté, alors* $M = M^{(t+m)} \times M^{(t+m-1)} \times ... \times M^{(t)}$ *est une matrice irréductible.*

Preuve 3 *Puisque la matrice d'incidence de* M *est* $I = I^{(t+m)} \times ... \times I^{(t)}$, *et que la matrice d'incidence du graphe superposé est* $I^{(t)} + I^{(t+1)} ... + I^{(t+m)}$, *grâce au lemme 1, nous déduisons que pour toute entrée du graphe superposé à* 1, *l'entrée correspondante de* I *est également* 1. *Ainsi, si le graphe superposé est connecté, alors* M *est irréductible.* □

Preuve du théorème 1

Condition suffisante : Par hypothèse, il existe toujours un instant $t \in N$, tel que le graphe superposé soit connecté et selon le lemme 2, pour tout instant $t > t_0$ il existe toujours des matrices irréductibles $T^{(p_i)}$ telles que :

$$M^{(t)} \times M^{(t-1)} \times ... \times M^{(1)} = M^{(t)}..M^{(t-L)} \times T^{(p_\alpha)} \times T^{(p_{\alpha-1})} \times ... \times T^{(p_1)}$$

où

$$\begin{array}{lcl} T^{(p_\alpha)} & = & M^{(t-L-1)} \times ...M^{(L_\alpha)}, \\ T^{(p_{\alpha-1})} & = & M^{(L_\alpha-1)} \times ...M^{(L_{\alpha-1})}, \\ & ... & , \\ T^{(p_1)} & = & M^{(L_2-1)} \times ...M^{(L_1)}, \end{array}$$

L et L_i sont des entiers finis (donc quand t tend vers l'infini α tend également vers l'infini et $\lim_{\alpha \to \infty} p_\alpha = \infty$).

Il est aisé de voir que les matrices $T^{(p_i)}$ sont doublement stochastiques, non-bipartites (-1 n'est pas une valeur propre) et irréductibles. Nous savons que si une

matrice A est irréductible et n'a pas -1 comme valeur propre, alors il existe l tel que A^l est une matrice positive. Cette déduction implique que

$$\forall i, \lim_{k\to\infty}(T^{p_i})^k = Q = \frac{1}{n}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

Soit $\gamma^{(p_j)}$ la seconde plus grande valeur propre de $T^{(p_j)}$, alors

$$0 \leq \gamma^{(p_j)} < 1$$

Rappelons que pour une matrice T stochastique ligne, nous avons

$$\|T\|_\infty = 1$$

où $\|T\|_\infty$ est la norme max, et si T est symétrique et doublement stochastique

$$\|T\|_2 \leq \sqrt{\|T\|_1 \|T\|_\infty} = \sqrt{\|T'\|_\infty \|T\|_\infty} = 1 \tag{2.4}$$

où T' est la matrice transposée de T et $\|T\|_1$ et $\|T\|_2$ sont respectivement la norme l_1 et la norme Euclidienne.

Soit $W^{(t)}$ le vecteur de charge du réseau à l'instant t, $W^{(0)}$ la répartition de charge initiale et $W^* = (\frac{\sum_i w_i^{(0)}}{n}, ..., \frac{\sum_i w_i^{(0)}}{n})$ la répartition uniforme de la charge, alors

$$\begin{aligned}
\left\|W^{(t+1)} - W^*\right\|_2 & \\
&= \left\|M^{(t)}...M^{(t-L)} \times T^{(p_\alpha)} \times T^{(p_{\alpha-1})} \times ... \times T^{(p_1)} W^{(0)} - W^*\right\|_2 \\
&\leq \left\|M^{(t)}\right\|_2 ... \left\|M^{(t-L)}\right\|_2 \times \left\|T^{(p_\alpha)} \times T^{(p_{\alpha-1})} \times ... \times T^{(p_1)} W^{(0)} - W^*\right\|_2 \\
&\leq \left\|T^{(p_\alpha)} \times T^{(p_{\alpha-1})} \times ... \times T^{(p_1)} W^{(0)} - W^*\right\|_2 \text{ selon (2.4)} \\
&= \left\|T^{(p_\alpha)}(T^{(p_{\alpha-1})} \times ... \times T^{(p_1)} W^{(0)}) - T^{(p_\alpha)}(W^*)\right\|_2 \\
&\leq \gamma^{(p_\alpha)} \left\|T^{(p_{\alpha-1})}(T^{(p_{\alpha-2})} \times ... \times T^{(p_1)} W^{(0)}) - T^{(p_{\alpha-1})}(W^*)\right\|_2 \\
&\quad \vdots \\
&\leq \gamma^{(p_\alpha)} \cdots \gamma^{(p_1)} \left\|W^{(0)} - W^*\right\|_2
\end{aligned}$$

ainsi

$$\lim_{t\to\infty} \left\|W^{(t+1)} - W^*\right\|_2 \leq \lim_{\alpha\to\infty} \gamma^{(p_\alpha)} \cdots \gamma^{(p_1)} \left\|W^{(0)} - W^*\right\|_2$$

Le nombre d'arcs étant fini, le nombre de graphes de connections l'est aussi. Ainsi il existe k tel que pour tout $j \in N$, $\gamma^{(p_j)} \leq \gamma^{(p_k)} < 1$, et donc $\lim_{\alpha\to\infty} \gamma^{(p_\alpha)} \cdots \gamma^{(p_1)} \leq \lim_{\alpha\to\infty} \left(\gamma^{(p_k)}\right)^\alpha = 0$.

L'inéquation précédente implique que $\lim_{t\to\infty} \left\|W^{(t)} - W^*\right\|_2 = 0$, donc

$$\forall i \in \{1, ...n\}\, w_i^{(t)} \to w_i^*$$

en d'autres termes, la charge de chaque processeur tend vers une répartition uniforme de la charge $w_i^* = \frac{\sum_i w_i^{(0)}}{n}$.

Condition nécessaire : Elle est évidente, si un processeur n'est jamais atteint, sa charge ne peut pas être équilibrée.

2.3 Algorithmes de type "dimension exchange"

Étudions à présent l'adaptation aux réseaux dynamiques des algorithmes de type "dimension exchange" ("dimension exchange" et "dimension exchange" généralisé), c'est à dire les algorithmes qui équilibrent les processeurs deux à deux.

2.3.1 Dimension "Exchange"

L'algorithme de dimension "Exchange" classique est dédié aux hypercubes. Cet algorithme est présenté, pour les réseaux statiques, en section 1.2.4. Rappelons tout d'abord sa forme générale :

$$w_i^{(t+1)} = w_i^{(t)} + \frac{1}{2}\left(w_j^{(t)} - w_i^{(t)}\right),$$

où j est voisin de i sur une dimension d donnée. Rappelons également que la dimension d utilisée à un instant t donné est calculée par $d = t \bmod D + 1$ où D est le nombre de dimensions de l'hypercube. Le principe de son application sur réseaux dynamiques a été présenté dans [12] et est équivalent à celui de la diffusion. Si, à un instant t, deux processeurs i et j doivent s'équilibrer ensemble et que le lien qui les relie est cassé, alors ils ne s'équilibrent pas et gardent leur charge respective. L'équation 2.5 modélise donc la charge échangée entre deux processeurs.

$$l_{ij}^{(t)} = \begin{cases} \frac{1}{2}(w_j^{(t)} - w_i^{(t)}) & \text{si } (i,j) \notin E_B^{(t)}, \\ 0 & \text{sinon,} \end{cases} \tag{2.5}$$

avec i voisin de j sur une dimension d donnée. On en déduit que l'évolution de la charge d'un processeur i entre deux instants est décrite par l'équation 2.6.

$$w_i^{(t+1)} = w_i^{(t)} + l_{ij}^{(t)} \tag{2.6}$$

Comme pour DE sur réseaux statiques, l'équilibrage s'effectue dimension par dimension. Ainsi, à chaque dimension correspond une matrice de diffusion qui est, dans le contexte de réseaux dynamiques, fonction du temps. La mise à jour du système s'exécute donc suivant l'équation 2.7.

$$W^{(t+1)} = M_d^{(t)} W^{(t)}, \tag{2.7}$$

avec $M_d^{(t)}$ la matrice de diffusion de la dimension d à l'instant t définie par m_{ij} tels que

$$m_{ij(i\neq j)}^{(t)} = \begin{cases} 0.5 & \text{si } (i,j) \notin E_B^{(t)}, \\ 0 & \text{sinon,} \end{cases}$$

$$m_{ii}^{(t)} = \begin{cases} 0.5 & \text{si } (i,j) \notin E_B^{(t)}, \\ 1 & \text{sinon,} \end{cases}$$

j étant le voisin de i sur la dimension d, dans le cas contraire $m_{ij} = 0$.

2.3.2 Generalized Adaptative Exchange

L'algorithme "Generalized Adaptative Exchange" (GAE) est l'application de GDE sur les réseaux dynamiques, vu en section 1.2.5, avec un degré de liberté quant au choix du voisin. Cet algorithme est présenté dans le rapport de recherche [10]. Dans l'algorithme GDE, les processeurs s'équilibrent par paires suivant un ordre défini par la coloration du graphe. Cet ordre imposé peut être un frein à l'équilibrage lorsque GDE opère sur un réseau dynamique. Considérons à un instant donné qu'un nœud i doive s'équilibrer, selon la coloration, avec son voisin j ; cependant, l'arc (i,j) n'est pas opérationnel. Sous ces conditions et selon le mode opératoire de GDE, i et j ne s'équilibrent pas. Toutefois, il est possible que i ait un voisin k qui ne s'équilibre avec personne et pour lequel le lien (i,k) soit opérationnel. Il semble donc plus judicieux de ne pas respecter l'ordre imposé et d'équilibrer i avec k. Nous proposons donc un nouvel algorithme GAE, dérivant de GDE, pour lequel le choix des paires de voisins n'est pas imposé dans l'algorithme. Ce choix doit cependant être réalisé selon une stratégie (aléatoire, arbitraire (GDE) ou plus sophistiquée) respectant les contraintes des algorithmes de type DE.

D'un point de vue modélisation, nous considérons que GAE est l'algorithme de diffusion dans lequel chaque nœud a au plus un arc non cassé. La stratégie de choix définit pour un nœud i, à un instant t, un voisin j tel que, d'une part, l'arc (i,j) soit opérationnel et d'autre part, j ne s'équilibre pas déjà avec un autre nœud. Tout autre arc (i,k) différent de (i,j) est considéré comme cassé : $(i,k) \in E_B^{(t)}$. La charge

échangée entre deux processeurs se calcule donc selon l'équation 2.8.

$$l_{ij}^{(t)} = \begin{cases} \lambda(w_j^{(t)} - w_i^{(t)}) & \text{si } (i,j) \notin E_B^{(t)}, \\ 0 & \text{sinon.} \end{cases} \tag{2.8}$$

La mise à jour de la charge d'un nœud i donné se calcule selon l'équation 2.9.

$$w_i^{(t+1)} = w_i^{(t)} + l_{ij}^{(t)}. \tag{2.9}$$

Comme pour tous les algorithmes précédents, la mise à jour du système entre deux instants s'écrit sous la forme vectorielle donnée par l'équation 2.10.

$$W^{(t+1)} = M^{(t)} W^{(t)}, \tag{2.10}$$

où $M^{(t)}$ est la matrice de diffusion définie par :

$$m_{ij}^{(t)} = \begin{cases} \lambda & \text{si } (i,j) \in E \wedge (i,j) \notin E_B^{(t)} \wedge i \neq j, \\ 1-\lambda & \exists k | (i,k) \in E \wedge (i,k) \notin E_B^{(t)} \wedge i = j \\ 1 & \nexists k | (i,k) \in E \wedge (i,k) \notin E_B^{(t)} \wedge i = j \\ 0 & \text{sinon.} \end{cases} \tag{2.11}$$

En d'autres termes, $m_{ij}^{(t)} = \lambda$ et $m_{ii}^{(t)} = 1 - \lambda$ si i communique avec un de ses voisins j à l'instant t, pour tout autre nœud k, $m_{ik}^{(t)} = 0$. Si i ne s'équilibre avec aucun de ses voisins à l'instant t, $m_{ii}^{(t)} = 1$ et $m_{ij}^{(t)} = 0$ pour tout nœud j. La section 2.5.2 propose un exemple de détermination des matrices de diffusion selon une stratégie quasi aléatoire.

2.3.3 Conditions et preuve de convergence

Nous savons que les algorithmes de type "dimension exchange" sont dérivés de la diffusion de premier ordre.

Corollaire 1 *GAE converge sous les conditions définies dans le théorème 1.*

Preuve 4 *Nous somme dans un cas particulier du théorème 1 où la matrice de diffusion M est définie par l'équation 2.11* □

2.4 Algorithme de diffusion relaxée

Pour finir avec la mise en place des algorithmes de premier ordre sur réseaux dynamiques, intéressons nous à la diffusion relaxée.

2.4.1 Modélisation de l'algorithme

La modélisation de l'algorithme de diffusion relaxée pour des réseaux dynamiques, s'effectue en introduisant le paramètre de relaxation $\beta^{(t)}$ dans l'algorithme de diffusion dédié à ces réseaux. C'est à dire, de la même manière que pour les réseaux statiques (cf. section 1.3). Ainsi, la charge échangée entre deux processeurs est donnée par l'équation 2.12.

$$l_{ij}^{(t)} = \begin{cases} \beta^{(t)}\alpha_{ij}(w_j^{(t)} - w_i^{(t)}) & \text{si } (i,j) \in E \wedge (i,j) \notin E_B^{(t)}, \\ 0 & \text{sinon.} \end{cases} \tag{2.12}$$

Comme pour le cas de la diffusion sur réseaux dynamiques, il n'y a échange de charge que si i et j sont voisins et que le lien qui les relie est opérationnel. La mise à jour d'un nœud donné i s'effectue de la même manière que pour la diffusion, le paramètre de relaxation étant inclus dans l'équation de la charge échangée. La charge de i à l'instant $t+1$ est calculée suivant l'équation 2.13 ; de manière littérale cette charge est celle de i à l'instant t plus la somme des charges échangées.

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j l_{ij}^{(t)}. \tag{2.13}$$

Les modifications engendrées sur l'équation vectorielle sont les mêmes qu'entre la diffusion sur réseaux statiques et sur réseaux dynamiques : la matrice M devient fonction du temps. L'évolution du système avec la diffusion relaxée est donc régie par l'équation 2.14.

$$W^{(t+1)} = \left(1 - \beta^{(t)}\right) W^{(t)} + \beta^{(t)} M^{(t)} W^{(t)}, \tag{2.14}$$

avec $M^{(t)}$ définie en section 2.2.1.

2.4.2 Paramètre de relaxation β optimal

La détermination du paramètre β optimal est, dans le cas de réseaux statiques, définie en section 1.3.2 par l'équation 1.19 que nous rappelons ici :

$$\beta_{opt}^{(t)} = \min\left(R^{(t)}, \frac{2}{2-(s+l)}\right)$$

Dans cette expression, nous avons $R^{(t)}$ qui est uniquement fonction de la répartition de charge à l'instant t et $\frac{2}{2-(s+l)}$ qui dépend de la matrice de diffusion. Dans le contexte de réseaux dynamiques, la matrice diffusion peut être différente à chaque

instant. Notons donc, $s^{(t)}$ et $l^{(t)}$ respectivement la plus petite et la seconde plus grande valeur propre de $M^{(t)}$. La valeur de β_{opt} peut ainsi être déterminée à tout instant par l'équation 2.15.

$$\beta_{opt}^{(t)} = \min\left(R^{(t)}, \frac{2}{2-(s^{(t)}+l^{(t)})}\right) \tag{2.15}$$

Bien que cette expression donne la valeur optimale de β, elle nécessite une connaissance de la matrice de diffusion à chaque instant, ce qui est contraire au caractère distribué de l'algorithme. Nous verrons par la suite comment appliquer cet algorithme dans un contexte réel. La section 2.5.1 propose un exemple de détermination de ce paramètre.

2.4.3 Conditions et preuve de convergence

L'algorithme de diffusion relaxée est dérivé de la diffusion de premier ordre et peut être vu comme un cas particulier de la diffusion où la matrice de diffusion est définie par

$$(1-\beta^{(t)})Id + \beta^{(t)}M^{(t)},$$

avec Id la matrice identité aux dimensions de $M^{(t)}$.

Proposition 2 *Avec β choisi selon l'équation 2.15, et sous les conditions définies dans le théorème 1, l'algorithme de diffusion relaxée converge.*

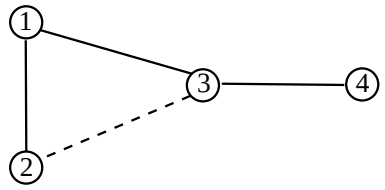
Preuve 5 *Il suffit d'appliquer les résultats présentés dans [18].* □

2.5 Illustrations des algorithmes de premier ordre sur réseaux dynamiques

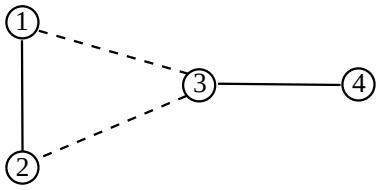
Afin d'illustrer la mise en place de ces algorithmes, nous présentons dans cette section quelques exemples sur réseaux dynamiques. Des simulations sur différents types de réseaux sont présentés dans le chapitre 3.

2.5.1 Illustration des algorithmes de type diffusion sur réseaux dynamiques

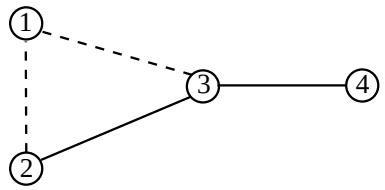
Considérons, tout d'abord, une évolution possible donnée en figure 2.1 du réseau représenté par la figure 1.1. Les liens en pointillés sont considérés inutilisables



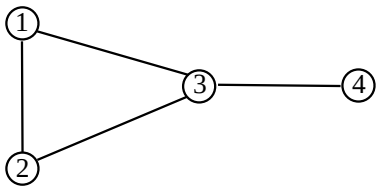
(a) Temps t, $E_B^{(t)} = \{(2,3)\}$.



(b) Temps $t+1$, $E_B^{(t+1)} = \{(2,3)(1,3)\}$.



(c) Temps $t+2$, $E_B^{(t+2)} = \{(1,2)(1,3)\}$.



(d) Temps $t+3$, $E_B^{(t+3)} = \{\}$.

FIG. 2.1 – Évolution d'un graphe quelconque.

(cassés) à l'instant correspondant. Afin de déterminer simplement les matrices de diffusion pour chaque instant, utilisons un choix de Cybenko, $\alpha_{ij} = \frac{1}{d(G)+1}$. Nous obtenons, pour l'évolution choisie, quatre matrices différentes (voire tableau 2.1), une par instant. On peut remarquer que sur l'intervalle de temps choisi, le théorème

$$M^{(t)} \begin{bmatrix} 1/2 & 1/4 & 1/4 & 0 \\ 1/4 & 3/4 & 0 & 0 \\ 1/4 & 0 & 1/2 & 1/4 \\ 0 & 0 & 1/4 & 3/4 \end{bmatrix} \quad M^{(t+1)} \begin{bmatrix} 3/4 & 1/4 & 0 & 0 \\ 1/4 & 3/4 & 0 & 0 \\ 0 & 0 & 3/4 & 1/4 \\ 0 & 0 & 1/4 & 3/4 \end{bmatrix}$$

$$M^{(t+2)} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3/4 & 1/4 & 0 \\ 0 & 1/4 & 1/2 & 1/4 \\ 0 & 0 & 1/4 & 3/4 \end{bmatrix} \quad M^{(t+3)} \begin{bmatrix} 1/2 & 1/4 & 1/4 & 0 \\ 1/4 & 1/2 & 1/4 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 1/4 & 3/4 \end{bmatrix}$$

TAB. 2.1 – Matrices de diffusion relatives aux graphes 2.1.

1 est vérifié : à l'instant t le graphe est connecté, le graphe superposé $G(t+1, t+2)$

l'est aussi, ainsi que le graphe à $(t+3)$. Notons que pour un instant t donné, on peut considérer le graphe superposé $G(t,t)$. Les matrices de diffusion données dans le tableau 2.1 sont aussi bien celles pour l'algorithme de diffusion de premier ordre que celles pour l'algorithme de diffusion relaxée.

En ce qui concerne la diffusion relaxée, il reste à déterminer β_{opt} à chaque instant. Commençons par donner les valeurs propres de chaque matrice de diffusion (cf Tab.2.2). Rappelons que β_{opt} se calcule de la manière suivante dans le cas de

$$\begin{bmatrix} 0.146 & 0.5 & 0.853 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0.5 & 0.5 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0.25 & 0.75 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0.25 & 0.75 & 1 \end{bmatrix}$$

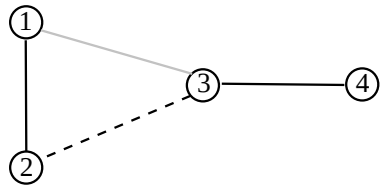
TAB. 2.2 – Valeurs propres relatives aux matrices 2.1.

réseaux dynamiques : $\beta_{opt}^{(t)} = \min\left(R^{(t)}, \frac{2}{2-(s^{(t)}+l^{(t)})}\right)$. Ainsi, nous obtenons pour les quatre instants considérés :

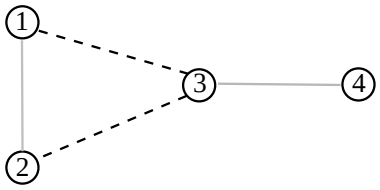
$$\begin{array}{rcl} \beta_{opt}^{(t)} & = & \min\left(R^{(t)}, 1.998\right) \\ \beta_{opt}^{(t+1)} & = & \min\left(R^{(t)}, 2\right) \\ \beta_{opt}^{(t+2)} & = & \min\left(R^{(t)}, 2\right) \\ \beta_{opt}^{(t+3)} & = & \min\left(R^{(t)}, 1.6\right) \end{array}$$

2.5.2 Illustration de l'algorithme GAE sur réseaux dynamiques

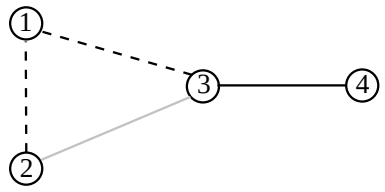
Reprenons l'évolution du réseau quelconque figure 1.1 donnée par la figure 2.1 et appliquons GAE sur ce dernier. Il faut tout d'abord déterminer une stratégie pour le choix des paires de processeurs. Pour l'exemple, nous optons pour une stratégie aléatoire qui ne crée pas deux fois de suite les mêmes paires de processeurs : une paire créée à l'instant t n'est pas recréée à l'instant $t+1$. Les liens grisés sur la figure 2.2 représentent les paires de processeurs choisies à chaque instant. On peut remarquer que le choix respecte bien les différentes conditions : un processeur ne s'équilibre qu'avec un seul de ses voisins et un processeur ne s'équilibre pas deux fois de suite avec le même voisin. Notons également que sur la période présentée,



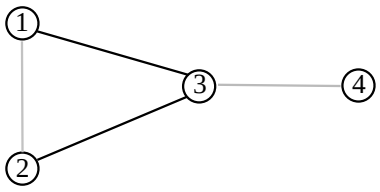
(a) Temps t, $E_B^{(t)} = \{(2,3)\}$.



(b) Temps $t+1$, $E_B^{(t+1)} = \{(2,3)(1,3)\}$.



(c) Temps $t+2$, $E_B^{(t+2)} = \{(1,2)(1,3)\}$.



(d) Temps $t+3$, $E_B^{(t+3)} = \{\}$.

FIG. 2.2 – Évolutions d'un graphe quelconque et des paires de processeurs utilisées par GAE.

le théorème 1 se vérifie : les graphes superposés $G(t, t+1)$ et $G(t+2, t+3)$ sont connectés.

L'algorithme GAE considère que les arcs valides mais non utilisés pour l'équilibrage sont cassés, nous avons donc pour l'exécution de l'algorithme :

$$\begin{aligned} E_B^{(t)} &= \{(2,3)(1,2),(3,4)\} \\ E_B^{(t+1)} &= \{(2,3)(1,3)\} \\ E_B^{(t+2)} &= \{(1,2)(1,3)(3,4)\} \\ E_B^{(t+3)} &= \{(1,3)(2,3)\} \end{aligned}$$

En supposant que $\lambda = 1/2$, les ensembles E_B permettent de déterminer pour chaque instant, les matrices de diffusion données dans le tableau 2.3. Notons pour finir que la valeur optimale de λ ne peut pas être calculée du fait du choix aléatoire des paires de processeurs : λ_{opt} dépend de $M(\lambda)$ donc de l'ordre d'application des matrices $M^{(t)}$.

$$M^{(t)} \begin{bmatrix} 1/2 & 0 & 1/2 & 0 \\ 0 & 1 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M^{(t+1)} \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/4 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \end{bmatrix}$$

$$M^{(t+2)} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M^{(t+3)} \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/4 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \end{bmatrix}$$

TAB. 2.3 – Matrices de diffusion de GAE relatives aux graphes 2.2.

2.6 Conclusion

Ce chapitre introduit les méthodes permettant d'appliquer les algorithmes vus au chapitre 1 sur des réseaux dynamiques. On parle de réseaux dynamiques lorsque la topologie du réseau change au cours de l'exécution. Ces changements sont dûs à des coupures ou des surcharges du réseau. Dans notre cas nous ne considérons le dynamisme qu'au niveau communication, c'est à dire qu'un nœud ne peut ni apparaître spontanément ni définitivement disparaître au cours de l'exécution. Nous traitons plus particulièrement dans ce chapitre les deux grands types d'algorithmes de premier ordre : les algorithmes de type diffusion et ceux de type " Dimension Exchange". En ce qui concerne les algorithmes diffusion de premier ordre, diffusion relaxée et DE, leur application sur réseaux dynamiques est quasiment identique à celle sur réseaux statiques, la seule différence étant que si un lien de communication est cassé, il n'est pas utilisé pour l'équilibrage. Dans le cas de GDE, la coloration du graphe peut être un frein à l'équilibrage, nous proposons donc une nouvelle approche (GAE), sans coloration, permettant de s'adapter au dynamisme du réseau.

Nous avons vu jusqu'ici différents algorithmes dédiés aux réseaux statiques ainsi que leurs adaptations pour les réseaux dynamiques. Il semble naturel à présent d'étudier leurs comportements sur différentes topologies et de mettre en évidence l'influence du dynamisme du réseau sur leur convergence. Le chapitre suivant présente plusieurs simulations permettant de comparer tous ces algorithmes.

Chapitre 3

Simulations

3.1 Introduction

Nous avons vu jusqu'ici différents algorithmes et leur adaptation sur réseaux dynamiques. Il est intéressant d'étudier à présent leur comportement sur différents réseaux avec pertes de liens de communication. Différentes comparaisons entre certains de ces algorithmes peuvent être trouvées dans [74, 72, 4]. Ce chapitre présente différentes simulations réalisées avec scilab, logiciel scientifique dédié au calcul numérique et développé par l'INRIA [22]. Afin que la comparaison soit la plus réaliste possible, nous déterminons les topologies de telle sorte que le nombre de nœuds soit identique. Ainsi, les simulations sont réalisées sur une ligne et un anneau de 64 nœuds, sur une grille 2D et un tore 8×8, sur une grille 3D $4 \times 4 \times 4$ et enfin sur un hypercube de dimension 6. Dans le but d'étudier le comportement pur de ces algorithmes, c'est à dire sans interaction avec l'application à équilibrer, nous représentons la charge de chaque processeur par un réel. Nous considérons pour les simulations, que la charge est infiniment divisible. Le système est considéré équilibré lorsque la différence de charge entre le nœud le plus chargé et celui le moins chargé est inférieure à 1. A l'initialisation du système, nous nous plaçons dans une répartition de charge la plus défavorable possible. En d'autres termes, à l'itération 0, toute la charge du système est placée sur le processeur 0, tous les autres ayant une charge nulle.

Une première section 3.2 se cantonne aux réseaux statiques. Elle présente d'une part l'évolution des algorithmes en fonction de la topologie sur laquelle ils sont appliqués, et d'autre part l'influence des paramètres de diffusion pour les algorithmes de type diffusion, puis pour ceux de type DE. Une seconde section 3.3 présente quant à elle l'interaction entre le dynamisme du réseau et la vitesse de convergence des algorithmes. Tout d'abord pour GAE avec différentes stratégies pour le choix des

paires de processeurs, puis pour les algorithmes de type diffusion. Elle se termine par une comparaisons, tout algorithme confondu, selon la méthode de détermination des paramètres de diffusion.

3.2 Comparaison sur réseaux statiques

Commençons par étudier le comportement des algorithmes vus précédemment sur des réseaux statiques.

3.2.1 Influence de la topologie

Voyons dans un premier temps l'évolution de ces algorithmes sur diverses topologies de réseaux. Il nous faut tout d'abord déterminer les paramètres de diffusion pour les différents algorithmes. Notre objectif étant de mettre en évidence l'influence de la topologie, nous choisissons les paramètres les plus simples possibles. En ce qui concerne la diffusion, nous optons pour un choix de Cybenko où $\alpha = \frac{1}{d(G)+1}$. Dans le cas de GDE, nous choisissons *lambda* de manière innée, c'est à dire : $\lambda = 1/2$. Le dernier paramètre à déterminer est celui de relaxation. Il n'existe pas de méthode immédiate donnant la valeur de β, il nous faut donc la calculer selon l'équation 1.19. Le tableau 3.1 récapitule les valeurs de α et β selon les différentes topologies. En ce

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
α	1/3	1/3	1/5	1/7	1/5	1/7
β	1.5	1.496	1.25	1.166	1.164	1

TAB. 3.1 – α et β déterminés selon les différentes topologies statiques.

qui concerne β, l'équation 1.19 donne sa valeur optimale $\beta_{opt}^{(t)}$ pour chaque instant. Cette valeur pose un problème non négligeable : elle doit être recalculée à chaque instant avec une connaissance globale de la répartition de charge. Évoluant dans un système distribué, il n'est pas concevable d'effectuer ce calcul à chaque itération. Pour la diffusion relaxée, β est donc calculé une seule fois avant l'exécution de l'algorithme et est utilisé pour toutes les itérations. Le tableau 3.1 donne ainsi la valeur de $\beta_{opt}^{(0)}$.

Les paramètres déterminés, nous pouvons donner les résultats des simulations. Le tableau 3.2, présente selon l'algorithme utilisé, diffusion de premier ordre (FOS), diffusion relaxée (RFOS) et "dimension exchange" généralisé (GDE) et selon la topologie, le nombre d'itérations nécessaires afin d'obtenir une répartition uniforme

de la charge. Les résultats donnés dans ce tableau mettent en évidence l'effet de la

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
FOS	6595	1648	193	72	49	20
RFOS	4395	1185	151	55	43	20
GDE	4395	1098	150	55	36	6

TAB. 3.2 – Nombre d'itérations nécessaires à la convergence de FOS, RFOS et GDE.

topologie sur les différents algorithmes. On remarque en premier lieu que plus la topologie est connectée, plus l'équilibrage s'effectue rapidement. Ceci est vrai pour les trois algorithmes. En ce qui concerne plus particulièrement les algorithmes de type diffusion, FOS et RFOS, comme nous pouvions nous y attendre RFOS est plus rapide que FOS. En observant les résultats de plus près, on peut voir que plus le réseau est connecté, plus *beta* est proche de 1 et moins le gain apporté par la relaxation est élevé. Dans le cas particulier de l'hypercube, nous avons $beta = 1$, la relaxation est donc équivalente à la diffusion classique et n'apporte rien de plus. En comparant RFOS et GDE, on peut observer que la vitesse de convergence est quasi identique dans la majeure partie des cas, avec toutefois un léger avantage pour GDE. Cet avantage intervient dans le cas de topologies régulières : anneau, tore et hypercube. Pour finir, dans le cas de l'hypercube, GDE s'exécute comme DE, le nombre d'itérations pour atteindre l'équilibre est égal au nombre de dimensions de l'hypercube.

3.2.2 Influence des paramètres de diffusion

Intéressons nous à présent à l'influence des différents paramètres sur la convergence des algorithmes.

Algorithmes de type diffusion

Nous avons choisi, dans la section précédente, d'utiliser les paramètres les plus simples possibles. En ce qui concerne la diffusion, il existe trois méthodes permettant de déterminer α. La première est celle utilisée précédemment $\alpha = \alpha_{ij} = \frac{1}{d(G)+1}$ (choix de Cybenko), la seconde consiste à calculer α en fonction du degré de chaque nœud tel que $\alpha_{ij} = \frac{1}{\max(d(i),d(j))+1}$ (choix de Boillat) et enfin, la dernière méthode consiste à calculer α optimal en fonction du graphe $\alpha = \alpha_{ij} = \alpha_{opt}$. Nous les nommons respectivement α_{Cyb}, α_{Boi} et α_{opt}. Le tableau 3.3 donne les valeurs de α_{Cyb} et α_{opt} en fonction du graphe. Les valeurs de α_{Boi} dépendant du nœud considéré dans un

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
α_{Cyb}	1/3	1/3	1/5	1/7	1/5	1/7
α_{Boi}	1/3	1/3	-	-	1/5	1/7
α_{opt}	1/2	2/4.01	1/4	1/6	1/4.29	1/7

TAB. 3.3 – α en fonction de la topologie et de la méthode de calcule.

graphe donné, elles ne sont données dans ce tableau que pour les topologies où α_{Boi} est constant. Notons que α_{opt} pour l'anneau étudié est proche de 1/2, sa valeur ne peut toutefois pas être arrondie à 1/2 sinon l'anneau est bipartite et l'algorithme de diffusion ne converge pas. Nous savons que dans le cas de la diffusion relaxée, β dépend de M donc de α. Puisque nous avons trois méthodes différentes permettant de déterminer α, nous avons pour chaque cas une valeur différente de β. Le tableau 3.4 donne les valeurs β_{Cyb}, β_{Boi} et β_{opt} calculées respectivement en fonction de α_{Cyb}, α_{Boi} et α_{opt}. Toutes ces valeurs de β sont calculées de manière optimale suivant l'équation 1.19. On remarque que la plupart des valeurs de β_{opt} sont égales à 1, la

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
β_{Cyb}	1.5	1.496	1.274	1.292	1.164	1
β_{Boi}	1.5	1.496	1.263	1.197	1.164	1
β_{opt}	1	1	1.019	1.108	1	1

TAB. 3.4 – β en fonction de la topologie et de la méthode de calcule.

relaxation n'apporte donc rien de plus pour ces cas. Pour les grilles 2D et 3D, les valeurs de β_{opt} sont proches de 1. Le gain engendré par la relaxation est donc faible relativement à la diffusion avec α_{opt}

Voyons à présent les résultats de simulations donnés dans le tableau 3.5. Comparons tout d'abord les différents choix de α pour la diffusion. Pour les topologies étudiées, le choix de Boillat donne les mêmes valeurs de α qu'avec celui de Cybenko, sauf pour les grilles 2D et 3D. Les résultats sont donc les mêmes en ce qui concerne la ligne, l'anneau, le tore et l'hypercube. Pour les deux cas où α est différent, le choix de Boillat est plus avantageux que celui de Cybenko. La détermination optimale de α donne, comme on peut s'y attendre, de meilleurs résultats que les deux autres choix, hormis pour la grille 3D. Dans ce cas, le choix de Boillat est plus efficace qu'un choix optimal. Ceci s'explique simplement : le choix optimal détermine α tel que tous les α_{ij} soient constants ($\alpha_{ij} = \alpha_{opt} \forall i, j$), α_{ij} non constant peut donner une vitesse d'équilibrage supérieure à α_{opt}.

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
FOS_{Cyb}	6595	1648	193	72	49	20
FOS_{Boi}	6595	1648	180	60	49	20
FOS_{opt}	4395	1185	154	61	43	20
$RFOS_{Cyb}$	4395	1185	151	55	43	20
$RFOS_{Boi}$	4395	1185	142	49	43	20
$RFOS_{opt}$	4395	1185	151	55	43	20

TAB. 3.5 – Nombre d'itérations nécessaires à la convergence de FOS et RFOS selon la méthode de calcule des paramètres.

Étudions à présent les résultats obtenus avec la diffusion relaxée. On observe en premier lieu que la relaxation basée sur un choix de Cybenko est équivalente à celle basée sur un choix optimal. De plus, comme précisé précédemment, la relaxation n'apporte quasi rien de plus que la diffusion avec un choix optimal de α. Concernant β calculé selon un choix de Boillat et avec une topologie en grilles, l'apport de la relaxation n'est pas négligeable (gain de 6 et 11%). Notons pour finir que dans le cas de l'hypercube, quel que soit l'algorithme et le choix de α, la vitesse de convergence est la même. De manière générale, le choix le plus judicieux semble être d'effectuer de la relaxation basée sur un choix de Boillat pour α_{ij}. Au pire il est équivalent aux autres choix et dans certain cas il peut être plus rapide.

Algorithmes GDE

Dans le cas de GDE, le choix de λ se limite à deux méthodes : la méthode innée qui nous donne $\lambda = \lambda_{inn} = \frac{1}{2}$ et la méthode optimale qui donne $\lambda = \lambda_{opt}$. Le tableau 3.6 donne les différentes valeurs de λ selon la méthode de calcul et la topologie considérée. On remarque tout d'abord que dans le cas de l'hypercube, la

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
λ_{inn}	1/2	1/2	1/2	1/2	1/2	1/2
λ_{opt}	0.95	0.91	0.72	0.58	0.58	1/2

TAB. 3.6 – λ inné et optimal selon la topologie.

méthode innée est équivalente à la méthode optimale. Ce tableau montre également que plus le graphe est connecté, plus λ_{opt} est proche de $\frac{1}{2}$. Les résultats obtenus avec ces différentes valeurs de λ sont donnés dans le tableau 3.7. Ces résultats sont

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
GDE_{inn}	4395	1098	150	55	36	6
GDE_{opt}	182	89	44	32	22	6

TAB. 3.7 – Nombre d'itérations nécessaires à la convergence de GDE selon λ choisi.

sans équivoques, le gain apporté par GDE_{opt} est tel qu'aucun algorithme de premier ordre vu dans ce chapitre ne peut le concurrencer. Notons toutefois que, lorsque λ est proche de 1, l'algorithme effectue des transferts de charge conséquents, ce qui peut être pénalisant lors d'une application réelle.

3.3 Comparaison sur réseaux dynamiques

Nous avons vu précédemment l'influence des paramètres de diffusion sur la convergence des algorithmes. Voyons à présent comment ils réagissent sur des réseaux dynamiques.

3.3.1 GAE : influence de la stratégie de choix

Afin de mettre en évidence l'influence de la stratégie de choix des paires de voisins, ainsi que celle du dynamisme du réseau pour GAE, nous reprenons les six topologies étudiées précédemment, auxquelles nous appliquons un pourcentage de pertes sur les liens de communication. Les tableaux 3.9, 3.10 et 3.11 présentent respectivement les résultats pour des réseaux avec 10, 30 et 50% de pertes. Avant d'étudier ces tableaux en détail, focalisons nous sur l'influence de la stratégie dans le cas de réseaux statiques. Ces résultats sont donnés dans le tableau 3.8. Nous proposons dans ce tableau trois stratégies différentes. Une première arbitraire de type GDE notée *arbi*, avec cette stratégie l'ordre paires est défini avant l'exécution de l'algorithme. Une seconde aléatoire notée *rand*, dans ce cas les paires sont déterminées dynamiquement et aléatoirement à chaque itération . Et en fin une dernière notée $M2LL$ ("Most to Least Loaded") qui consiste à créer des paires de manière à toujours essayer de décharger en premier lieu les processeurs les plus chargés vers leur voisin le moins chargé. Pour chacune de ces stratégies, nous avons deux simulations : une consistant à choisir λ de manière innée (*inn*) et une autre pour laquelle λ optimal est utilisé (*opt*). On remarque tout d'abord que la stratégie aléatoire n'est pas efficace, on ne l'approfondira donc pas dans le contexte de réseaux dynamiques. L'utilisation de λ optimal reste dans tous les cas le paramètre le plus efficace ; notons

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
$GAE_{arbi_{inn}}$	4395	1098	150	55	36	6
$GAE_{arbi_{opt}}$	182	89	44	32	22	6
$GAE_{rand_{inn}}$	6674	1652	218	81	67	36
$GAE_{rand_{opt}}$	5809	1327	138	65	52	32
$GAE_{M2LL_{inn}}$	4395	1098	135	55	34	6
$GAE_{M2LL_{opt}}$	243	102	54	36	30	6

TAB. 3.8 – Nombre d'itérations nécessaires à la convergence de GDE selon différentes stratégies et selon λ choisi avec 0% de perte.

qu'il l'est d'autant plus que la stratégie est arbitraire (de type GDE). Ceci est tout à fait logique, puisque λ_{opt} est calculé à partir de cette stratégie. Pour finir, on peut remarquer que la stratégie $M2LL$ n'apporte rien de plus que celle arbitraire avec λ_{inn} dans le contexte de réseaux statiques.

Penchons nous à présent sur l'impact du dynamisme du réseau sur l'évolution de GAE. Comme indiqué précédemment, nous nous limitons aux stratégies arbitraire (*arbi*) et $M2LL$. Au regard des résultats présentés dans les tableaux 3.9, 3.10 et

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
$GAE_{arbi_{inn}}$	5267	1317	185	70	46	17
$GAE_{arbi_{opt}}$	673	225	78	49	35	25
$GAE_{M2LL_{inn}}$	5635	1394	161	56	41	17
$GAE_{M2LL_{opt}}$	1380	352	70	40	28	14

TAB. 3.9 – Nombre d'itérations nécessaires à la convergence de GDE selon différentes stratégies et selon λ choisi avec 10% de perte.

3.11, on remarque que l'impacte du dynamisme varie selon la stratégie déterminant les paires de voisins. Dans le cas de λ_{inn}, la stratégie arbitraire est plus sensible aux pertes de liens que celle $M2LL$, ce qui donne un avantage à $M2LL$ lorsque le réseau est dynamique. Cette sensibilité au dynamisme se retrouve également lorsque λ_{opt} est utilisé. Cependant $M2LL$ n'est pas toujours plus avantageux dans ce cas que la stratégie arbitraire. Ceci s'explique par le fait qu'avec λ_{inn} $M2LL$ est équivalent à *arbi* dans un contexte statique alors qu'avec λ_{opt} *arbi* est le plus rapide. La stratégie $M2LL$ devient donc avantageuse selon le pourcentage de pertes et le réseau considéré.

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
$GAE_{arbi_{inn}}$	7275	1883	262	120	68	35
$GAE_{arbi_{opt}}$	1789	560	134	74	51	32
$GAE_{M2LL_{inn}}$	7068	1798	185	59	47	20
$GAE_{M2LL_{opt}}$	2530	653	91	41	33	16

TAB. 3.10 – Nombre d'itérations nécessaires à la convergence de GDE selon différentes stratégies et selon λ choisi avec 30% de perte.

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
$GAE_{arbi_{inn}}$	9989	2563	346	155	95	47
$GAE_{arbi_{opt}}$	3109	889	188	113	76	48
$GAE_{M2LL_{inn}}$	8617	2184	217	69	55	20
$GAE_{M2LL_{opt}}$	3422	912	111	54	39	19

TAB. 3.11 – Nombre d'itérations nécessaires à la convergence de GDE selon différentes stratégies et selon λ choisi avec 50% de perte.

3.3.2 Influence du pourcentage de pertes sur la diffusion

Nous nous sommes penchés sur l'interaction entre le dynamisme du réseau et GAE, voyons à présent comment se comporte les algorithmes de diffusion et de diffusion relaxée sur ces réseaux. Comme précédemment, nous étudions leur comportement avec différents pourcentages de pertes de liens de communication. Les tableaux 3.12, 3.13 et 3.14 donnent respectivement les résultats avec 10, 30 et 50% de pertes de liens. Pour ces simulations, nous ne considérons que deux méthodes

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
FOS_{Boi}	7251	1816	200	66	54	22
FOS_{opt}	4828	1212	170	68	46	22
$RFOS_{Boi}$	4839	1213	158	55	46	22
$RFOS_{opt}$	4838	1210	166	62	46	22

TAB. 3.12 – Nombre d'itérations nécessaires à la convergence de FOS et RFOS selon la méthode de calcule de α et avec 10% de perte.

de détermination de α : le choix de Boillat (Boi) et celui optimal (opt). Celle basée

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
FOS_{Boi}	8775	2223	240	82	67	28
FOS_{opt}	5845	1489	206	83	56	28
RFOS_{Boi}	5835	1480	191	67	58	27
RFOS_{opt}	5838	1490	202	76	58	27

TAB. 3.13 – Nombre d'itérations nécessaires à la convergence de FOS et RFOS selon la méthode de calcule de α et avec 30% de perte.

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
FOS_{Boi}	10750	2728	299	101	82	33
FOS_{opt}	7169	1832	253	104	70	35
RFOS_{Boi}	7155	1822	233	83	69	34
RFOS_{opt}	7159	1822	247	95	68	35

TAB. 3.14 – Nombre d'itérations nécessaires à la convergence de FOS et RFOS selon la méthode de calcule de α et avec 50% de perte.

sur le degré du graphe, choix de Cybenko, est équivalente à (Boi) dans la plupart des réseaux considérés. Les résultats obtenus avec différents pourcentages de pertes nous montrent que le dynamisme influe considérablement sur la vitesse de convergence des algorithmes. Cette influence est toutefois uniforme est n'avantage ni un algorithme ni un choix de α en particulier. On en tire donc les mêmes conclusions que sur un réseau statique : dans le cas de la diffusion simple, un choix optimal est le plus avantageux, mais de manière générale la diffusion relaxée basée sur un choix de Boillat donne des résultats équivalents à FOS_{opt} et RFOS_{opt} voire meilleurs pour certaines topologies.

3.3.3 Étude comparative des différents algorithmes

Nous avons vu l'interaction entre les différents algorithmes et les réseaux dynamiques. Comparons à présent, dans le cas particulier d'une grille 2D, leur comportement selon, d'une part un choix simple des paramètres de diffusion, puis d'autre part selon un choix optimal. Nous nous limitons à une grille 2D pour cette comparaison puisque ce réseaux est le plus classique et pour ne pas surcharger le document. Les courbes présentées dans les figures 3.1 et 3.2 montrent le nombre d'itérations nécessaires à la convergence en fonction du pourcentage de pertes. A l'origine ces

courbes sont exponentielles, cependant elles sont quasi linéaires sur les taux de perte étudiés, elles ont donc été linéarisées afin de mettre en évidence l'impact du dynamisme sur la vitesse d'équilibrage. Les algorithmes utilisés sont : FOS et RFOS avec un choix de Boillat, et GAE avec une stratégie arbitraire et M2LL. Les graphiques 3.1 et 3.2 présentent respectivement ces algorithmes avec un choix simple et un choix optimal de leur paramètre de diffusion. Considérons tout d'abord le cas des choix

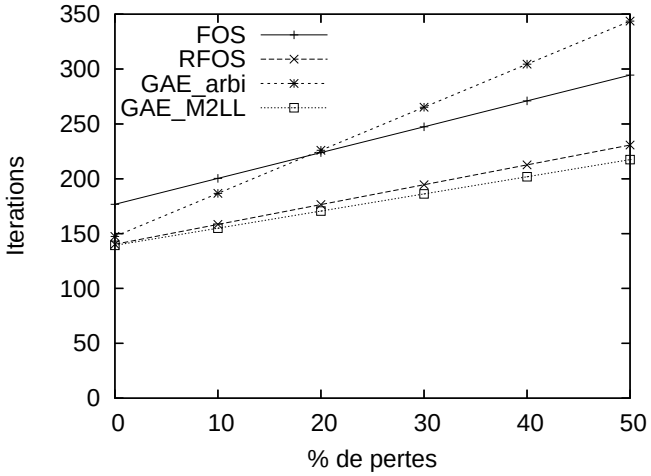


FIG. 3.1 – Évolution du nombre d'itérations en fonction du pourcentage de perte pour une grille 2D avec une détermination simple des paramètres.

simples. On peut remarquer que RFOS et GAE_{M2LL} sont quasi équivalents avec la même sensibilité au dynamisme du réseau. Remarquons également que GAE_{arbi} est très sensible aux pertes de liens et devient moins efficace que la diffusion. Dans le cas d'un choix optimal, GAE est toujours plus rapide que la diffusion. Notons que cette courbe montre bien que FOS et RFOS sont équivalents. On remarque pour finir que GAE_{arbi} est également très sensible aux pertes de liens avec λ_{opt} mais reste dans ce cas plus efficace qu'un algorithme de type diffusion.

3.4 Conclusion

Ce chapitre présente différentes simulations dans différents contextes et illustre les influences des paramètres de diffusions, du dynamisme du réseau et dans le cas de GAE la stratégie de choix pour les paires de processeurs. Suite à ces simulations, on peut remarquer que, hormis GAE avec λ_{opt}, le choix de tel ou tel algorithme

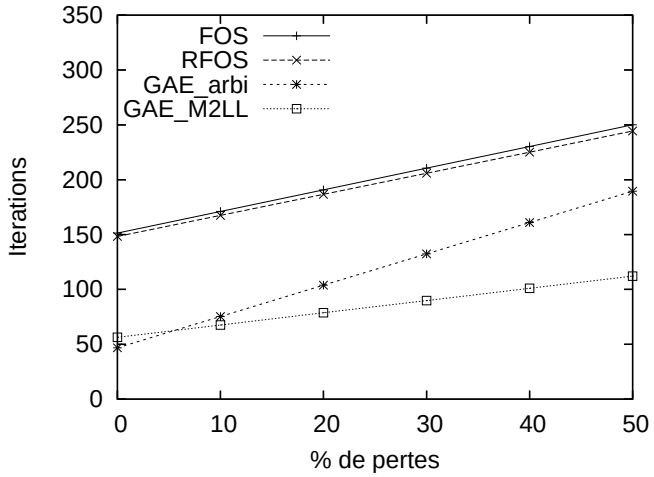


FIG. 3.2 – Évolution du nombre d'itérations en fonction du pourcentage de perte pour une grille 2D avec une détermination optimale des paramètres.

afin d'équilibrer un système, dépend de la topologie et du pourcentage de pertes sur les liens de communication. En ce qui concerne GAE avec λ_{opt} et une stratégie M2LL, l'équilibrage est le plus efficace avec une grande tolérance aux pertes de liens. Cependant il peut être pénalisant car il effectue, pour certaines topologies, des transferts de charge très importants à chaque étape. Ces diverses simulations ainsi que les remarques que nous en avons tirées nous permettent de déterminer quel algorithme doit être utilisé suivant les conditions induites par le système à équilibrer.

Ce chapitre termine ainsi notre première partie dédiée aux algorithmes de premier ordre. Avant de passer à une expérimentation concrète - l'équilibrage d'un programme réparti de calcul numérique - nous nous intéressons aux algorithmes de N-ième ordre et plus particulièrement à l'algorithme de diffusion de second ordre.

Deuxième partie

Diffusion de Second Ordre

Chapitre 4

Second Ordre pour Réseaux Statiques

4.1 Introduction

Nous savons que le rôle des algorithmes d'équilibrage de charge est d'équilibrer le travail réparti sur différentes machines. Mais ceci doit être réalisé de manière la plus rapide possible et sans pénaliser le système à équilibrer. Dans le contexte des algorithmes d'équilibrage de charge distribués, et plus particulièrement dans celui des algorithmes de diffusion, une approche utilisant une mémoire des échanges passés a été développée. Ce type d'algorithme est appelé de n-ième ordre selon les n niveaux de mémoire utilisés [44, 26, 32]. Dans ce chapitre nous étudions en particulier les algorithmes de type diffusion de second ordre pour l'équilibrage de charge, qui ont été introduits dans [44]. Nous nous penchons plus précisément sur l'algorithme de diffusion de second ordre proprement dit [44] et sur une de ses variantes : l'algorithme de Chebyshev [26]. Ces algorithmes étant de type second ordre, les échanges de charge sont calculés en fonction de la répartition de charge courante et de celle à l'itération précédente.

Ce chapitre se restreint à une application de ces algorithmes sur des réseaux statiques. Une première section 4.2 présente les principes des algorithmes de diffusion de second ordre. Nous donnons également les conditions de convergence relatives aux paramètres de l'algorithme ainsi que celles relatives à l'équilibrage et enfin leurs valeurs optimales. Pour finir cette section nous présentons la variante de Chebyshev. Une seconde section 4.3 présente notre contribution à ces algorithmes : nous donnons les bornes des paramètres garantissant la positivité de la charge. Une dernière section 4.4 illustre l'application de ces algorithmes ainsi que la détermination des différents paramètres et présente des simulations de ces algorithmes.

4.2 État de l'art : algorithme de type diffusion de second ordre

Présentons tout d'abord les principes des algorithmes de type second ordre, puis le cas particulier de la méthode de Chebyshev.

4.2.1 Diffusion de second ordre

L'algorithme de diffusion de second ordre consiste à équilibrer la charge en fonction de la charge courante et en fonction des transferts de charge effectués à l'itération précédente. Il utilise un niveau de mémoire, les transferts à l'itération $t-1$, afin d'accélérer la convergence de la diffusion de premier ordre. Ainsi les transferts de charge entre deux processeurs i et j sont donnés par l'équation 4.1

$$l_{ij}^{(t)} = \begin{cases} \alpha_{ij}\left(w_j^{(0)} - w_i^{(0)}\right) & t = 0, \\ (\beta-1)l_{ij}^{(t-1)} + \beta\alpha_{ij}\left(w_j^{(t)} - w_i^{(t)}\right) & t > 0, \end{cases} \tag{4.1}$$

où β pondère le gradient et la mémoire. On peut remarquer que le second ordre est bien basé sur le premier ordre puisque si β est égal à 1 les deux algorithmes sont équivalents. De plus on retrouve dans l'équation 4.1 les différentes parties du premier ordre. Notons également qu'à la première itération cet algorithme applique le premier ordre, ceci permet de créer la mémoire nécessaire à l'exécution de l'algorithme.

Comme pour les algorithmes vus en première partie, la mise à jour du système peut s'écrire sous forme vectorielle. L'écriture des algorithmes de premier ordre sous cette forme est triviale, mais elle ne l'est pas pour le second ordre, nous la détaillons donc ci-dessous. Pour un nœud i donné, sa mise à jour entre deux instants est donnée par :

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j l_{ij}^{(t)},$$

où j sont les voisins de i. Afin de donner la forme vectorielle du second ordre, développons cette expression pour t supérieur à 0.

$$\begin{aligned} w_i^{(t+1)} = \; & w_i^{(t)} + \textstyle\sum_j l_{ij}^{(t)} \\ & w_i^{(t)} + \textstyle\sum_j \left((\beta-1)l_{ij}^{(t-1)} + \beta\alpha_{ij}\left(w_j^{(t)} - w_i^{(t)}\right)\right) \\ & w_i^{(t)} + (\beta-1)\textstyle\sum_j l_{ij}^{(t-1)} + \beta\textstyle\sum_j \alpha_{ij}\left(w_j^{(t)} - w_i^{(t)}\right). \end{aligned} \tag{4.2}$$

Rappelons que $\sum_j l_{ij}^{(t-1)} = \left(w_i^{(t)} - w_i^{(t-1)}\right)$, nous avons donc :

$$\begin{aligned} w_i^{(t+1)} = \quad & w_i^{(t)} + (\beta - 1)\left(w_i^{(t)} - w_i^{(t-1)}\right) + \beta \sum_j \alpha_{ij}\left(w_j^{(t)} - w_i^{(t)}\right) \\ & \beta w_i^{(t)} + (1-\beta) w_i^{(t-1)} + \beta \sum_j \alpha_{ij}\left(w_j^{(t)} - w_i^{(t)}\right) \\ & \beta w_i^{(t)} + (1-\beta) w_i^{(t-1)} + \beta \sum_j \alpha_{ij} w_j^{(t)} - \beta w_i^{(t)} \sum_j \alpha_{ij} \\ & (1-\beta) w_i^{(t-1)} + \beta \sum_j \alpha_{ij} w_j^{(t)} + \beta(1 - \sum_j \alpha_{ij}) w_i^{(t)} \\ & (1-\beta) w_i^{(t-1)} + \beta\left(\sum_j \alpha_{ij} w_j^{(t)} + (1 - \sum_j \alpha_{ij}) w_i^{(t)}\right). \end{aligned}$$

Ainsi on déduit facilement que pour t supérieur à 0 l'évolution du système est donnée par $W^{(t+1)} = (1-\beta)W^{(t-1)} + \beta M W^{(t)}$ où M est la matrice de diffusion définie de la même manière que pour le premier ordre, c'est à dire :

$$m_{ij} = \begin{cases} \alpha_{ij} & \text{si } i \neq j, \\ 1 - \sum_j \alpha_{ij} & \text{si } i = j. \end{cases}$$

La matrice M est donc symétrique, doublement stochastique est irréductible. De manière générale, la mise à jour du système est donnée par l'équation 4.3.

$$W^{(t+1)} = \begin{cases} MW^{(0)} & t = 0, \\ \beta M W^{(t)} + (1-\beta) W^{(t-1)} & t > 0. \end{cases} \tag{4.3}$$

En ce qui concerne β, il est connu que l'algorithme converge si β est compris entre 0 et 2 exclus [66].

4.2.2 Méthode de Chebyshev

La méthode de Chebyshev est présentée dans [26]. Cette méthode est une variante de l'algorithme de second ordre. La différence intervient au niveau de β qui est, dans le cas de Chebyshev, fonction du temps t. Ainsi l'équation 4.4 donne les valeurs de β pour chaque instant, afin de le différencier de $\beta^{(t)}$ du second ordre, nous le nommons $\beta_{Che}^{(t)}$.

$$\begin{aligned} \beta_{Che}^{(1)} &= 1, \\ \beta_{Che}^{(2)} &= \tfrac{2}{2-\mu_2^2}, \\ \beta_{Che}^{(t)} &= \tfrac{4}{4-\mu_2^2 \beta_{Che}^{(t-1)}} \quad t > 2. \end{aligned} \tag{4.4}$$

Le fait que β dépende du temps implique une légère modification, introduite dans l'équation 4.5, au niveau de l'équation 4.3.

$$W^{(t+1)} = \begin{cases} MW^{(0)} & t = 0, \\ \beta_{Che}^{(t)} M W^{(t)} + (1 - \beta_{Che}^{(t)}) W^{(t-1)} & t > 0. \end{cases} \tag{4.5}$$

La méthode de Chebyshev est souvent préférée au second ordre. La vitesse de convergence de Chebyshev est généralement supérieure à celle du second ordre.

4.3 Contribution : détermination du paramètre de second ordre β

Nous savons que β doit être compris entre 0 et 2. De manière plus précise, dans [44], les auteurs montrent que si β est compris entre 0 et 1, l'utilisation de la mémoire n'apporte rien à l'algorithme de premier ordre, et peut même ralentir la convergence. Le paramètre β est donc choisi entre 1 et 2, afin d'accélérer la vitesse d'équilibrage. Pour que l'accélération engendrée par β soit maximale, sa valeur doit être déterminée en fonction de M. Nous savons selon [45] que β optimal (β_{opt}) est donné par l'équation 4.6 dans le cas du second ordre, ou par l'équation 4.4 dans le cas de Chebyshev.

$$\beta_{opt} = \frac{2}{1 + \sqrt{1 - \mu_2^2}}, \tag{4.6}$$

où μ_2 est la seconde plus grande valeur propre de M. De manière générale ces valeurs optimales de β sont toujours utilisées, toutefois elles ne garantissent pas la positivité de la charge.

Donnons les conditions sur β afin que cette contrainte soit satisfaite. Considérons un nœud i donné, nous cherchons à déterminer β tel que $w_i^{(t+1)}$ soit positif, $w_i^{(t+1)}$ négatif n'ayant pas de sens.

$$\begin{aligned} w_i^{(t+1)} &\geq 0 \\ (1-\beta)w_i^{(t-1)} + \beta\left(\textstyle\sum_j \alpha_{ij} w_j^{(t)} + (1 - \sum_j \alpha_{ij}) w_i^{(t)}\right) &\geq 0 \end{aligned}$$

posons $w_{i(fos)}^{(t+1)} = \left(\sum_j \alpha_{ij} w_j^{(t)} + (1 - \sum_j \alpha_{ij}) w_i^{(t)}\right)$, ceci étant l'expression d'un pas de premier ordre (FOS) pour un nœud i donné. Nous obtenons donc :

$$\begin{aligned} (1-\beta)w_i^{(t-1)} + \beta w_{i(fos)}^{(t+1)} &\geq 0 \\ \beta\left(w_{i(fos)}^{(t+1)} - w_i^{(t-1)}\right) &\geq -w_i^{(t-1)} \end{aligned}$$

Trois cas sont donc possibles :

- $w_{i(fos)}^{(t+1)} - w_i^{(t-1)} > 0$
  Dans ce cas :
  $$\beta\left(w_{i(fos)}^{(t+1)} - w_i^{(t-1)}\right) \geq -w_i^{(t-1)}$$
  $$\beta \geq \frac{-w_i^{(t-1)}}{w_{i(fos)}^{(t+1)} - w_i^{(t-1)}}.$$

Ceci implique que β doit être supérieur à une valeur négative, or β est par définition positif.

– $w_{i(fos)}^{(t+1)} - w_i^{(t-1)} = 0$
Dans ce cas, la positivité de $w_i^{(t+1)}$ ne dépend pas de β, puisque sous ces conditions : $w_i^{(t+1)} = w_i^{(t-1)}$.

– $w_{i(fos)}^{(t+1)} - w_i^{(t-1)} < 0$
Dans ce cas :

$$\begin{aligned}
\beta\left(w_{i(fos)}^{(t+1)} - w_i^{(t-1)}\right) &\geq -w_i^{(t-1)} \\
\beta &\leq \frac{-w_i^{(t-1)}}{w_{i(fos)}^{(t+1)} - w_i^{(t-1)}} \\
\beta &\leq \frac{w_i^{(t-1)}}{w_i^{(t-1)} - w_{i(fos)}^{(t+1)}} \\
\beta &\leq 1 + \frac{w_{i(fos)}^{(t+1)}}{w_i^{(t-1)} - w_{i(fos)}^{(t+1)}}.
\end{aligned}$$

Cette borne supérieure est donc la seule que nous considérons, les deux cas précédents n'intervenant pas dans le choix de β. De plus, nous remarquons que cette borne dépend du temps. Jusqu'à présent, nous nous sommes restreints à un nœud i donné pour la détermination de β. Pour le réseau complet, il en découle naturellement que la valeur maximale de β à chaque instant est :

$$\beta_{max}^{(t)} = \min_i \left(1 + \frac{w_{i(fos)}^{(t+1)}}{w_i^{(t-1)} - w_{i(fos)}^{(t+1)}}\right) \quad \forall i | w_{i(fos)}^{(t+1)} - w_i^{(t-1)} < 0. \tag{4.7}$$

La valeur optimale de β, pour le second ordre, est donc fonction du temps et est définie par l'équation 4.8, dans le cas de Chebyshev, la valeur optimale de β est donnée par l'équation 4.9.

$$\beta^{(t)} = \min_i \left(\beta_{max}^{(t)}, \beta_{opt}\right), \tag{4.8}$$

$$\beta^{(t)} = \min_i \left(\beta_{max}^{(t)}, \beta_{Che}^{(t)}\right). \tag{4.9}$$

A la différence de la diffusion relaxée, l'équation donnant la valeur maximale de β n'est pas monotone, elle doit donc être calculée à chaque pas de temps. L'équation du second ordre 4.3 et celle de Chebyshev 4.5 se récrivent selon l'équation 4.10 pour garantir la positivité de la charge et prendre en compte β fonction du temps.

$$W^{(t+1)} = \begin{cases} MW^{(0)} & t = 0, \\ \beta^{(t)} MW^{(t)} + (1 - \beta^{(t)}) W^{(t-1)} & t > 0. \end{cases} \tag{4.10}$$

4.4 Illustration et simulation des algorithmes de second ordre

Commençons par illustrer les différents aspects de la mise en place de ces algorithmes.

4.4.1 Illustration des algorithmes de second ordre

Afin d'illustrer ces deux algorithmes, nous les appliquons sur le réseau quelconque figure 4.1 utilisé en première partie du document. Commençons par déterminer la

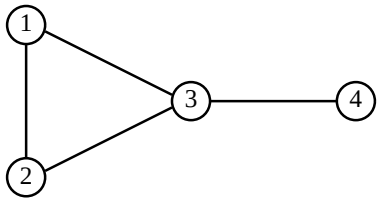


FIG. 4.1 – Graphe quelconque.

matrice de diffusion M optimale. Pour ce faire nous nous référons à la section 1.2.9. Tout d'abord, il nous faut déterminer le laplacien du graphe, à la différence de l'exemple pris dans la section 1.2.9, nous considérons le réseau homogène. Dans ce cas le laplacien L est égal à AA^T, soit

$$L = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}.$$

Ceci nous permet de déterminer la valeur optimale de α : $\alpha_{opt} = \frac{2}{\lambda_2+\lambda_n} = 2/5$, où λ_2 et λ_n sont respectivement la seconde plus petite et la plus grande valeur propre de L. Rappelons que cette valeur optimale de α ne garantit pas la positivité de la charge, α est donc choisi tel que : $\alpha = \min_i(\alpha_{opt}, \frac{1}{L_{ii}}) = \min(2/5, 1/3) = 1/3$. D'où :

$$M = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 \\ 1/3 & 1/3 & 1/3 & 0 \\ 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 1/3 & 2/3 \end{bmatrix}.$$

La matrice M étant définie, nous pouvons déterminer β_{opt} pour l'algorithme de second ordre en fonction de ses valeurs propres :

$$\left[\begin{array}{cccc} -1/3 & 0 & 2/3 & 1 \end{array} \right].$$

Ce qui nous donne :

$$\beta_{opt} = \frac{2}{1 + \sqrt{1 - (2/3)^2}} = 1.1458981.$$

Considérons un état initial le plus défavorable possible, c'est à dire que nous plaçons toute la charge du système sur un seul nœud,

$$W^{(0)} = \left[\begin{array}{c} 4 \\ 0 \\ 0 \\ 0 \end{array} \right].$$

Cet état initial est utilisé pour les deux algorithmes. Le premier pas s'exécute à l'identique pour les deux algorithmes et est un pas de diffusion de premier ordre, nous obtenons donc :

$$W^{(1)} = \left[\begin{array}{r} 1.3333333 \\ 1.3333333 \\ 1.3333333 \\ 0 \end{array} \right].$$

Pour l'itération suivante, nous devons déterminer $\beta_{max}^{(1)}$ selon l'équation 4.7, nous obtenons ainsi : $\beta_{max}^{(1)} = 1.5$. En ce qui concerne l'algorithme de second ordre, on remarque que β_{opt} est inférieur à $\beta_{max}^{(1)}$, la valeur optimale peut donc être utilisée. Pour mettre en évidence l'importance de β_{max}, nous présentons ci-dessous deux cas. Premièrement nous posons $\beta^{(1)} = \beta_{opt}$, puisque pour l'exemple choisi β_{opt} est toujours inférieur à $\beta_{max}^{(t)}$. Puis nous utilisons $\beta^{(t)}$ compris entre 2 et $\beta_{max}^{(1)}$. Le premier cas ne pose pas de problème puisque β_{opt} est inférieur à $\beta_{max}^{(1)}$. Pour le second cas nous avons $\beta^{(t)}$ compris entre 0 et 2, l'algorithme converge donc vers l'équilibre, mais $\beta^{(t)}$ est supérieur à $\beta_{max}^{(1)}$.

t	2	3	...	∞
$\beta^{(t-1)}$	1.1458981	1.1458981	...	1.1458981
W_1	0.9442719	1.1388026	...	1
W_2	1.527864	1.1388026	...	1
W_3	1.018576	0.9442719	...	1
W_3	0.5092880	0.7781228	...	1

t	2	3	...	∞
$\beta^{(t-1)}$	1.6	1.6	...	1.6
W_1	-0.2666667	0.9540741	...	1
W_2	2.1333333	0.9540741	...	1
W_3	1.4222222	0.5748148	...	1
W_3	0.7111111	1.517037	...	1

Avec $\beta^{(t)}$supérieur à $\beta_{max}^{(1)}$ on obtient une valeur négative pour $W_1^{(2)}$ ce qui n'est pas cohérent pour une charge et confirme bien l'importance de β_{max}.

En ce qui concerne l'algorithme de Chebyshev, β_{Che} doit être calculé à chaque itération. Le tableau suivant présente les valeurs de β_{Che}, β_{max} et la répartition de charge calculée selon l'algorithme de Chebyshev.

t	2	3	...	∞
$\beta_{Che}^{(t-1)}$	1	1.2857143	...	2
$\beta_{max}^{(t-1)}$	1.5	4.5	...	∞
W_1	1.3333333	1.1428571	...	1
W_2	1.3333333	1.1428571	...	1
W_3	0.8888889	0.9523809	...	1
W_3	0.4444444	0.7619048	...	1

On peut noter que pour les deux exemples les valeurs optimales de β sont inférieures à β_{max}. Le paramètre β_{max} intervient généralement pour des réseaux de grande taille et selon la topologie du réseaux. Ceci est illustré dans la section suivante.

4.4.2 Simulations sur réseaux statiques

Afin de mettre en évidence le comportement des algorithmes de type second ordre sur réseaux statiques, nous reprenons les différents réseaux étudiés en chapitre 3. Ces réseaux sont : une ligne et un anneau de 64 nœuds, une grille 2D et un tore 8×8, une grille 3D $4 \times 4 \times 4$ et enfin un hypercube de dimension 6. Toutes ces topologies sont constituées de 64 nœuds. Nous rappelons que les résultats obtenus ont été réalisés par simulations avec le logiciel scientifique scilab [22]. Comme pour les précédentes simulations, le système est considéré équilibré lorsque la différence de charge entre le nœud le plus chargé et celui le moins chargé est inférieure à 1 et, à l'itération 0, toute la charge du système est placée sur le processeur 0, tous les autres ayant une charge nulle.

Avant de présenter les résultats obtenus avec ces algorithmes, intéressons nous à l'importance de la valeur maximale de β (β_{max}). Le tableau 4.1 présente pour les

deux algorithmes - diffusion de second ordre (SOS) et Chebishev (Cheb) - et pour les différents types de réseaux, les itérations pour lesquelles β_{opt} est supérieur à β_{max}. On remarque tout d'abord que pour la ligne et l'anneau β_{opt} est toujours utilisé,

		SOS		Cheb	
	itération t	β_{opt}	$\beta_{max}^{(t)}$	$\beta_{Che}^{(t)}$	$\beta_{max}^{(t)}$
ligne	-	-	-	-	-
anneau	-	-	-	-	-
grille 2D	2	-	-	1.861033	1.600000
	3	1.570769	1.455156	-	-
	4	-	-	1.684108	1.666667
	5	1.570769	1.563529	1.638238	1.361200
	6	-	-	1.610246	1.597242
grille 3D	2	-	-	1.686724	1.500000
	3	1.397659	1.355650	-	-
	5	-	-	1.418503	1.280737
tore	2	1.329408	1.285203	1.594326	1.285203
hyper.	2	1.176571	1.166667	1.342466	1.166667

TAB. 4.1 – Toutes itérations pour lesquelles β_{opt} est supérieur à β_{max}, le signe - signifie que β_{opt} est inférieur à β_{max}, dans les cas présentés, β_{max} est utilisé à la place de β_{opt}.

ce qui nous donne une vitesse de convergence maximale pour ces deux topologies. En ce qui concerne les autres topologies, β_{opt} ou $\beta_{Che}^{(t)}$ est parfois supérieur à $\beta_{max}^{(t)}$. Pour ces itérations, $\beta_{max}^{(t)}$ est donc utilisé à la place de β_{opt} ou $\beta_{Che}^{(t)}$ pour garantir une charge positive. Notons que ceci intervient au début de l'exécution de l'algorithme, c'est à dire lorsque le système est le plus fortement déséquilibré. Dans notre contexte d'exécution, au-delà de la sixième itération β_{opt} peut toujours être utilisé. Ceci confirme le fait que si le système n'est pas totalement déséquilibré, β_{max} peut être ignoré.

Intéressons nous à présent au nombre d'itérations nécessaires à ces algorithmes pour équilibrer le système. Le tableau 4.2 donne ces valeurs pour SOS et Cheb selon les différentes topologies et rappelle celles obtenues pour GDE et FOS afin de les comparer. En ce qui concerne SOS et Cheb leur vitesse de convergence est quasi toujours identique. La méthode de Chebyshev n'apporte un gain de temps que pour la ligne, ce qui correspond à une topologie pour laquelle la valeur optimale de β a toujours été utilisée. En comparant SOS ou Cheb à l'algorithme de base FOS avec un

	ligne	anneau	grille 2D	grille 3D	tore	hyper.
SOS	176	81	30	19	16	11
Cheb	159	81	30	19	16	11
$\text{GDE}_{arbi_{opt}}$	182	89	44	32	22	6
FOS_{opt}	4395	1185	154	61	43	20

TAB. 4.2 – Nombre d'itérations nécessaires à la convergence de SOS et Cheb avec l'utilisation de $\beta^{(t)}$ ($\beta^{(t)} = \max(\beta_{max}, (\beta_{opt}$ ou $\beta_{Che}^{(t)}))$), en comparaison avec les algorithmes de premier ordre les plus rapides.

choix optimal de M, on observe que le gain de temps apporté par les algorithmes de second ordre est très important, il va de 50% pour l'hypercube à 96% pour la ligne. En ce qui concerne GDE, l'algorithme de premier ordre le plus rapide, on note que hormis pour l'hypercube, les algorithmes de second ordre sont plus efficaces. Ces diverses simulations montrent bien les avantages qu'apportent les algorithmes de second ordre sur réseaux statiques. Bien que la détermination de β_{max} soit contraire au caractère distribué des algorithmes, nous mettons en évidence le fait que cette borne peut être négligée si le système n'est pas totalement déséquilibré.

4.5 Conclusion

Ce chapitre introduit les algorithmes de type second ordre sur réseaux statiques. Il présente plus particulièrement l'algorithme de diffusion de second ordre et la méthode de Chebyshev. Pour ces deux algorithmes nous donnons les valeurs optimales de β qui peuvent facilement être trouvées dans la littérature. Nous introduisons, relativement à ce paramètre, une contrainte qui est fréquemment négligée dans les différentes études concernant le second ordre. Cette contrainte est la positivité de la charge. Celle ci peut être négligée si le réseaux est de petite taille ou si la répartition de charge initiale est favorable, c'est à dire une répartition initiale n'engendrant pas de charge négative par l'application de l'algorithme avec des paramètres optimaux. Ceci est souvent le cas dans un contexte réel où la charge initiale est répartie de manière homogène et dans lequel l'algorithme d'équilibrage a pour objectif d'adapter la répartition à l'hétérogénéité du réseau. Dans un contexte plus général, cette contrainte ne peut pas être négligée. La dernière section de ce chapitre illustre ce phénomène et montre que, selon le choix de β avec une répartition initiale donnée, la charge peut devenir négative tout en convergeant vers une répartition uniforme.

Suite à cette introduction des algorithmes de second ordre, nous nous intéressons à l'application de ceux-ci aux réseaux dynamiques. Le chapitre suivant présente les différentes modifications apportées à ces algorithmes afin qu'ils convergent vers l'équilibre sur ce type de réseaux. Il présente également une simulation permettant de comparer ces algorithmes à ceux de premier ordre.

Chapitre 5

Second Ordre pour Réseaux Dynamiques

5.1 Introduction

Comme pour les algorithmes de premier ordre, nous cherchons à appliquer ceux de second ordre aux réseaux dynamiques. Ceci n'a, à notre connaissance, jamais été réalisé. Le principe utilisé pour adapter ces algorithmes à ce type de réseaux est le même que pour ceux de premier ordre : l'équilibrage ne s'effectue que sur les liens en états de communication, c'est à dire ni sur les liens cassés ni sur ceux surchargés. Même si le principe est simple et que son application aux algorithmes de premier ordre reste trivial, en ce qui concerne les algorithmes de second ordre il faut gérer le cas de la mémoire. La détermination de β est également un problème majeur, nous avons déterminé dans la section 4.3 sa borne supérieure dans le contexte de réseaux statiques, dans le cas présent il faut prendre en compte, d'une part le dynamisme du réseau et d'autre part l'effet mémoire modifié par notre adaptation.

Ce chapitre présente dans un premier temps, en section 5.2, les modifications apportées aux algorithmes de type second ordre afin qu'ils s'adaptent aux réseaux dynamiques. La détermination de β est également développée dans cette section, elle permet de garantir la positivité de la charge à tout instant. En ce qui concerne la preuve de la convergence de ces algorithmes sur réseaux dynamique, nous ne le montrons qu'expérimentalement. La preuve théhorique est nettement plus complex que dans le cas d'algorithmes de premier ordre et nous ne sommes, à l'heure actuelle, pas parvenu à la finaliser. Nous illustrons en section 5.3 le comportement de ces algorithmes par différentes simulations sur réseaux dynamiques, et nous mettons en évidence l'impacte du dynamisme sur ces algorithmes.

5.2 Adaptation des algorithmes de second ordre aux réseaux dynamiques

Commençons par présenter les adaptations nécessaires à ces algorithmes afin qu'ils puissent évoluer sur des réseaux dynamiques.

5.2.1 Modification de l'algorithme

L'algorithme devant évoluer sur un réseau dynamique, nous rappelons en premier lieu la modélisation de ce type de réseau. De manière générale, un réseau se modélise par un graphe $G = (V, E)$ où V est l'ensemble des nœuds et E est l'ensemble des arcs du réseau (cf. section 1.2.1). Comme présenté en section 2.2.1, le dynamisme est introduit par l'ensemble $E_B^{(t)}$ qui contient les arcs (i, j) inutilisables à l'instant t. $E_B^{(t)}$ est un sous-ensemble de E. Un réseau dynamique est donc représenté par le graphe $G^{(t)} = (V, E, E_B^{(t)})$. Rappelons que si $E_B^{(t)}$ est vide $G^{(t)}$ est équivalent au graphe G.

La modélisation des algorithmes de second ordre sur réseaux dynamiques n'étant pas aussi évidente que celle des algorithmes de premier ordre, nous la présentons pas à pas en partant de ce principe : les échanges de charge ne s'effectuent que sur les liens opérationnels. Dans les algorithmes de second ordre étudiés, la charge échangée est donnée par l'équation 4.1. En appliquant le principe de l'équilibrage de charge sur réseaux dynamiques on obtient naturellement l'équation 5.1. L'équilibrage entre deux nœuds voisins i et j ne s'effectue que si l'arc (i, j) n'est pas dans l'ensemble $E_B^{(t)}$ (ensemble des arcs inutilisables).

$$l_{ij}^{(t)} = \begin{cases} \alpha_{ij}\left(w_j^{(t)} - w_i^{(t)}\right) & t = 0 \wedge (i,j) \notin E_B^{(0)}, \\ (\beta - 1) l_{ij}^{(t-1)} + \beta \alpha_{ij}\left(w_j^{(t)} - w_i^{(t)}\right) & t > 0 \wedge (i,j) \notin E_B^{(t)}, \\ 0 & \text{sinon.} \end{cases} \tag{5.1}$$

Pour un nœud i donné, sa mise à jour reste classique et est donnée par :

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j l_{ij}^{(t)},$$

En ce qui concerne la mise à jour du système, étudions tout d'abord le premier pas qui, étant un pas de premier ordre de l'algorithme de diffusion, ne pose pas de problèmes majeurs. En se référant au chapitre 2.2.1 nous avons directement :

$$W^{(1)} = M^{(1)} W^{(0)},$$

où $M^{(t)}$ est définie par :

$$m_{ij}^{(t)} = \begin{cases} \alpha_{ij} & \text{si } (i,j) \in E \wedge (i,j) \notin E_B^{(t)} \wedge i \neq j, \\ 1 - \sum_k \alpha_{ik} & \forall k | (i,k) \in E \wedge (i,k) \notin E_B^{(t)} \wedge i = j \\ 0 & \text{sinon.} \end{cases}$$

Pour les pas suivants, la mise à jour du système doit gérer la mémoire de l'algorithme. Pour ce faire, reprenons la mise à jour d'un nœud i donné pour t supérieur à 0.

$$\begin{array}{rl} w_i^{(t+1)} = & w_i^{(t)} + \sum_j l_{ij}^{(t)} \\ & w_i^{(t)} + \sum_j \left((\beta - 1) l_{ij}^{(t-1)} + \beta \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) \right) \quad (i,j) \notin E_B^{(t)} \end{array}$$

Notons que :

$$\sum_{j|(i,j) \notin E_B^{(t)}} (\beta - 1) l_{ij}^{(t-1)} = \sum_j (\beta - 1) l_{ij}^{(t-1)} - \sum_{k|(i,k) \in E_B^{(t)}} (\beta - 1) l_{ik}^{(t-1)}$$

et que

$$\sum_{j|(i,j) \notin E_B^{(t)}} \beta \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) = \sum_j \beta m_{ij}^{(t)} \left(w_j^{(t)} - w_i^{(t)} \right).$$

Posons k voisin de i tel que l'arc (i,k) soit cassé à l'instant t ($(i,k) \in E_B^{(t)}$). L'expression de $w_i^{(t+1)}$ peut donc s'écrire de la manière suivante :

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j (\beta - 1) l_{ij}^{(t-1)} - \sum_k (\beta - 1) l_{ik}^{(t-1)} + \sum_j \beta m_{ij}^{(t)} \left(w_j^{(t)} - w_i^{(t)} \right),$$

soit

$$w_i^{(t+1)} = \left[w_i^{(t)} + (\beta - 1) \sum_j l_{ij}^{(t-1)} + \beta \sum_j m_{ij}^{(t)} \left(w_j^{(t)} - w_i^{(t)} \right) \right] - (\beta - 1) \sum_k l_{ik}^{(t-1)}.$$

On remarque dans cette dernière expression une similitude, partie entre accolades, avec l'algorithme du second ordre sur réseaux statiques (cf équation 4.2) où, pour le cas présent, $m_{ij}^{(t)}$ remplace α_{ij}. La principale différence entre la version pour réseaux statiques et celle pour réseaux dynamiques est l'apparition de $-(\beta - 1) \sum_k l_{ik}^{(t-1)}$ dans la version dynamique. Ceci représente la partie de la mémoire à ignorer due aux pertes des liens (i,k). Pour l'ensemble du réseau, notons $\bar{L}^{(t)}$ le vecteur contenant ces valeurs à ignorer à un instant t donné. Le vecteur $\bar{L}^{(t)}$ est définit par :

$$\bar{l}_i^{(t)} = \sum_k l_{ik}^{(t)}$$

Nous pouvons ainsi donner l'expression vectorielle des algorithmes de type second ordre sur réseaux dynamiques.

$$W^{(t+1)} = \begin{cases} M^{(t)}W^{(t)} & t = 0 \\ \beta M^{(t)}W^{(t)} + (1-\beta)W^{(t-1)} - \bar{L}^{(t)} & t > 0 \end{cases} \tag{5.2}$$

5.2.2 Détermination du paramètre de second ordre β

Comme pour les algorithmes de premier ordre sur réseaux dynamiques, la détermination de β optimal n'est pas possible du fait du dynamisme du réseau. Nous considérons donc, pour les réseaux dynamiques, la valeur optimale de β calculée selon le réseau statique correspondant. Dans le cas de l'algorithme de diffusion de second ordre, nous avons donc β calculé selon l'équation 4.6 que nous rappelons ici :

$$\beta_{opt} = \frac{2}{1+\sqrt{1-\mu_2^2}},$$

et dans le cas de Chebyshev nous obtenons β selon l'équation 4.4 :

$$\begin{array}{rcll} \beta_{Che}^{(1)} & = & 1, & \\ \beta_{Che}^{(2)} & = & \frac{2}{2-\mu_2^2}, & \\ \beta_{Che}^{(t)} & = & \frac{4}{4-\mu_2^2\beta_{Che}^{(t-1)}} & t > 2. \end{array}$$

L'expression des algorithmes de type second ordre sur réseaux statiques ayant été modifiée pour s'adapter aux réseaux dynamiques, la valeur maximale de β (β_{max}) doit être redéfinie en conséquence. La valeur de β_{max} est déterminée de telle sorte que l'application de l'algorithme n'entraîne pas de charge négative pour un nœud quelconque du système.

$$\begin{array}{rcl} w_i^{(t+1)} & \geq & 0 \\ w_i^{(t)} + (\beta-1)\sum_j l_{ij}^{(t-1)} + \beta\sum_j m_{ij}^{(t)}\left(w_j^{(t)} - w_i^{(t)}\right) - (\beta-1)\bar{l}_i^{(t-1)} & \geq & 0 \end{array}$$

Par analogie avec la détermination de β_{max} dans le cas de réseaux statiques (cf. section 4.3), on en déduit que :

$$(1-\beta)w_i^{(t-1)} + \beta w_{i(fos)}^{(t+1)} - (\beta-1)\bar{l}_i^{(t-1)} \geq 0$$

où $w_{i(fos)}^{(t+1)}$ est l'expression d'un pas de premier ordre dans le cas de réseaux dynamiques $\left(w_{i(fos)}^{(t+1)} = w_i^{(t)} + \sum_j m_{ij}^{(t)}\left(w_j^{(t)} - w_i^{(t)}\right)\right)$. On obtient ainsi pour β :

$$\beta\left(w_{i(fos)}^{(t+1)} - w_i^{(t-1)} - \bar{l}_i^{(t-1)}\right) \geq -\bar{l}_i^{(t-1)} - w_i^{(t-1)}.$$

Selon la section 4.3, nous savons que seule la borne supérieure de β nous intéresse. Cette borne est définie lorsque $w_{i(fos)}^{(t+1)} - w_i^{(t-1)} - \bar{l}_i^{(t-1)} < 0$, sous cette condition nous avons également $-\bar{l}_i^{(t-1)} - w_i^{(t-1)} < 0$. On en déduit donc :

$$\begin{aligned}
\beta &\leq \frac{-\bar{l}_i^{(t-1)} - w_i^{(t-1)}}{w_{i(fos)}^{(t+1)} - w_i^{(t-1)} - \bar{l}_i^{(t-1)}} \\
\beta &\leq \frac{-\left(w_i^{(t-1)} + \bar{l}_i^{(t-1)}\right)}{w_{i(fos)}^{(t+1)} - \left(w_i^{(t-1)} + \bar{l}_i^{(t-1)}\right)} \\
\beta &\leq \frac{w_i^{(t-1)} + \bar{l}_i^{(t-1)}}{w_i^{(t-1)} + \bar{l}_i^{(t-1)} - w_{i(fos)}^{(t+1)}} \\
\beta &\leq 1 + \frac{w_{i(fos)}^{(t+1)}}{w_i^{(t-1)} + \bar{l}_i^{(t-1)} - w_{i(fos)}^{(t+1)}}
\end{aligned}$$

Notons que cette borne est supérieure ou égale à 1. Cette borne se restreint à un nœud i en particulier. De manière générale, pour tout le système nous avons donc la valeur maximale de β déterminée par l'équation 5.3.

$$\beta_{max}^{(t)} = \min_i \left(1 + \frac{w_{i(fos)}^{(t+1)}}{w_i^{(t-1)} + \bar{l}_i^{(t-1)} - w_{i(fos)}^{(t+1)}}\right) \quad \forall i | w_{i(fos)}^{(t+1)} - w_i^{(t-1)} - \bar{l}_i^{(t-1)} < 0 \tag{5.3}$$

On en déduit ainsi que la valeur optimale de β à chaque instant est déterminée par l'équation 5.4.

$$\beta_{opt}^{(t)} = \min(\beta_{opt}, \beta_{max}^{(t)}) \tag{5.4}$$

Dans l'équation 5.4, β_{opt} est la valeur optimale de β dans le cas de l'algorithme de diffusion de second ordre ou $\beta_{Cheb}^{(t)}$ dans le cas de Chebyshev.

5.3 Simulations

Étudions à présent le comportement de ces deux algorithmes sur différentes topologies. Nous reprenons les six réseaux précédents auxquels nous appliquons un pourcentage de perte de liens. Nous étudions tout d'abord leur comportement sur ces réseaux avec un faible pourcentage de perte. Le tableau 5.1 présente ces résultats pour des pertes de liens comprises entre 2 et 8 %. Dans ce tableau, le signe "-" signifie que l'algorithme n'a pas convergé en moins de 10000 itérations. Les résultats les plus marquants dans ce tableau sont ceux obtenus pour les topologies ligne et anneau. Ils semblent incohérents dans certains cas mais s'expliquent assez simplement. Les deux algorithmes étudiés sont très sensibles aux pertes de liens, ils le sont d'autant plus que la topologie sur laquelle ils sont appliqués est faiblement connectée (ligne, anneau). Cette sensibilité est telle que l'algorithme de second ordre devient plus lent

	pertes	ligne	anneau	grille 2D	grille 3D	tore	hyper.
SOS	2%	302	110	34	20	17	11
Cheb	2%	1154	1263	38	19	16	11
SOS	4%	501	132	37	22	18	11
Cheb	4%	601	1166	34	22	18	12
SOS	6%	2103	174	37	23	19	12
Cheb	6%	1248	1204	38	23	18	12
SOS	8%	-	203	42	24	19	13
Cheb	8%	1268	1308	45	24	20	13

TAB. 5.1 – Nombre d'itérations nécessaires à la convergence de SOS et Cheb avec 2 à 8% de perte de liens

que celui de premier ordre qu'il doit théoriquement accélérer. Les résultats présentés ne sont donc pas des moyennes : les écarts-types obtenus pour la ligne et l'anneau sont trop importants pour qu'une moyenne ait un sens.

Ce problème ne semble apparaître que pour les deux premières topologies étudiées. Comparons donc pour les autres topologies l'impact des pertes de liens sur le second ordre et Chebyshev par rapport à l'impact sur les algorithmes de premier ordre. Pour cette comparaison, nous prendrons en compte uniquement les algorithmes de premier ordre les plus rapides : la diffusion de premier ordre avec un choix optimal de α et GAE avec une stratégie M2LL et un choix optimal de λ (cf. chapitre 3) . Les tableaux 5.2, 5.3 et 5.4 présentent le nombre d'itérations nécessaires à la convergence des algorithmes pour les diverses topologies avec respectivement 10, 30 et 50% de perte de liens. Les résultats obtenus confirment, principalement pour Chebyshev

	anneau	grille 2D	grille 3D	tore	hyper.
SOS	242	45	25	20	13
Cheb	271	45	25	21	13
$GAE_{M2LL_{opt}}$	352	70	40	28	14
FOS_{opt}	1212	170	68	46	22

TAB. 5.2 – Nombre d'itérations nécessaires à la convergence de SOS et Cheb avec 10% de perte de liens

sur anneau, les remarques que nous avons tirées des simulations sur topologies avec faible pourcentage de pertes. En étudiant plus précisément les différents résultats

	anneau	grille 2D	grille 3D	tore	hyper.
SOS	500	70	41	31	19
Cheb	1101	71	39	30	18
$GAE_{M2LL_{opt}}$	653	91	41	33	16
FOS_{opt}	1489	206	83	56	28

TAB. 5.3 – Nombre d'itérations nécessaires à la convergence de SOS et Cheb avec 30% de perte de liens

	anneau	grille 2D	grille 3D	tore	hyper.
SOS	859	101	54	40	26
Cheb	784	103	52	42	26
$GAE_{M2LL_{opt}}$	912	111	54	39	19
FOS_{opt}	1832	253	104	70	35

TAB. 5.4 – Nombre d'itérations nécessaires à la convergence de SOS et Cheb avec 50% de perte de liens

obtenus pour les autres topologies, on remarque tout d'abord que Chebyshev est équivalent à l'algorithme de second ordre et n'apporte aucun avantage dans notre contexte d'exécution. En comparant les résultats obtenus avec les algorithmes de second ordre et ceux obtenus avec les algorithmes de premier ordre, on remarque tout d'abord que la diffusion de premier ordre est toujours moins rapide qu'un algorithme de second ordre. Par contre GAE est équivalent au second ordre pour la grille 3D et pour le tore, et est plus rapide que le second ordre dans le cas de l'hypercube. GAE étant dérivé d'un algorithme dédié aux hypercubes, il est tout à fait normal qu'il soit le plus rapide pour cette topologie. Le second ordre ne présente donc, par rapport à GAE, qu'un avantage dans le cas de la grille 2D.

5.4 Conclusion

Ce chapitre termine notre partie sur les algorithmes de second ordre. Il présente principalement l'application de ces algorithmes aux réseaux dynamiques. Une première section introduit les modifications apportées à l'algorithme classique pour réseaux statiques afin qu'il puisse évoluer sur un réseau dynamique. Nous présentons également comment déterminer la borne supérieure de β de telle sorte que la charge de chaque nœud reste positive. Des simulations terminent ce chapitre et permettent

d'illustrer le comportement de ces algorithmes sur différentes topologies avec divers pourcentages de perte de liens.

De manière générale, les algorithmes de second ordre sont certes les plus efficaces dans le cas de réseaux statiques, mais présentent plusieurs inconvénients. La détermination de la borne supérieure de β à chaque instant nécessite une connaissance globale de la topologie et de la répartition de charge. Cette contrainte est de plus contraire au caractère distribué de l'algorithme. Elle ne peut être négligée que dans le cas de réseaux statiques avec une répartition initiale de la charge qui ne soit pas totalement déséquilibrée. Dans le cas de réseaux dynamiques, les algorithmes de second ordre sont très sensibles aux pertes de liens et sont imprévisibles voir inutilisables si la topologie est trop faiblement connectée, ce qui est le cas pour une ligne ou un anneau. En ce qui concerne l'anneau, ces algorithmes fonctionnent relativement bien mais leur vitesse d'équilibrage est imprévisible. Les avantages qu'apportent les algorithmes de second ordre dans le cas de réseaux dynamiques sont donc relativement faibles par rapport aux algorithmes de premier ordre et ils entraînent des contraintes non négligeables.

Troisième partie

De la Théorie à la Pratique

Chapitre 6

L'Application et l'Équilibrage de Charge

6.1 Introduction

Nous nous sommes jusqu'à présent consacrés à la théorie des algorithmes d'équilibrage de charge et à des simulations permettant de comparer les différents algorithmes sans interaction avec une application. Dans ce chapitre, nous nous intéressons à la mise en place concrète d'un algorithme d'équilibrage de charge dans une application réelle. L'application choisie est la résolution d'une équation différentielle partielle (EDP) modélisant le problème d'advection-diffusion qui correspond à un problème de cinétique chimique. Dans la littérature, nous trouvons différentes mise en application des algorithmes de diffusion sur des problèmes réels [51, 38]. Nous présentons plus particulièrement dans ce chapitre les différents problèmes pouvant intervenir lors de la mise en place de l'équilibrage de charge. L'application d'advection-diffusion nous est fournie et notre objectif est d'inclure notre code d'équilibrage à l'intérieur, la bibliothèque de communication utilisée est MPI. Dans ce chapitre, nous nous restreignons à un contexte synchrone.

Une première section 6.2 présente l'application d'advection-diffusion et sa modélisation pour l'implantation distribuée. Une seconde section 6.3 introduit le choix de la méthode d'équilibrage et la mise en place de cet algorithme d'équilibrage de charge dans l'application avec les différents problèmes rencontrés. Cette section donne également des comparatifs de temps d'exécution entre la version avec et sans équilibrage.

6.2 L'application : advection-diffusion

Commençons par décrire l'application d'advection-diffusion et son implantation afin de comprendre la mise en place de l'équilibrage et les différents problèmes pouvant intervenir.

6.2.1 Description séquentielle de l'application

Le problème correspond à un système d'équations différentielles partielles (EDPs). Il modélise un mécanisme de cinétique chimique d'advection-diffusion. Cette application permet de déterminer les évolutions en fonction du temps des concentrations de deux espèces chimiques c^1 et c^2 dans l'espace. Pour notre étude, nous considérons un espace à deux dimensions. On effectue une discrétisation cartésienne de cet espace afin de créer une grille. L'évolution des concentrations des espèces chimiques est donc calculée en chacun des points $x - z$ de la grille résultante. Le système différentiel permettant de déterminer ces concentrations, est donné par l'équation 6.1 où i est utilisé pour distinguer les deux espèces chimiques considérées ($i = 1, 2$),

$$\frac{\partial c^i}{\partial t} = K_h \frac{\partial^2 c^i}{\partial x^2} + V \frac{\partial c^i}{\partial x} + \frac{\partial}{\partial z} K_v(z) \frac{\partial c^i}{\partial z} + R^i(c^1, c^2, t), \tag{6.1}$$

et où les termes de la réaction $R^i(c^1, c^2, t)$ sont donnés par :

$$\begin{aligned} R^1(c^1, c^2, t) &= -q_1 c^1 c^3 - q_2 c^1 c^2 + 2q_3(t)c^3 + q_4(t)c^2 \\ R^2(c^1, c^2, t) &= q_1 c^1 c^3 - q_2 c^1 c^2 - q_4(t)c^2 \end{aligned}$$

Pour le problème, nous utilisons les valeurs suivantes pour les constantes et paramètres :

$$\begin{aligned} &K_h = 4.0 \times 10^{-6}, \\ &V = 10^{-3}, \\ &K_v(z) = 10^{-8} exp(z/5), \\ &q_1 = 1.63 \times 10^{-16}, \\ &q_2 = 4.66 \times 10^{-16}, \\ &c^3 = 3.7 \times 10^{16}, \end{aligned}$$

et les $q_j(t)$ sont définis par :

$$\begin{aligned} q_j(t) &= exp[-a_j/sin(\omega t)] \quad &pour\ sin(\omega t) > 0 \\ q_j(t) &= 0 \quad &pour\ sin(\omega t) \leqslant 0 \end{aligned}$$

où $j = 3,\ 4,\ \omega = \pi/43200,\ a_3 = 22.62,\ a_4 = 7.601$. L'intervalle de temps d'intégration est [0, 7200s]. Les conditions initiales homogènes de Neumann sont

imposées et sont les suivantes :

$$c^1(x,z,0) = 10^6\alpha(x)\beta(z),$$
$$c^2(x,z,0) = 10^{12}\alpha(x)\beta(z)$$

avec

$$\alpha(x) = 1 - (0.1x-1)^2 + (0.1x-1)^4/2,$$
$$\beta(z) = 1 - (0.1z-1)^2 + (0.1z-4)^4/2$$

La discrétisation de l'espace considéré permet donc de transformer le système d'EDPs pour obtenir un système d'équations différentielles ordinaires (EDOs) de type

$$\frac{dy(t)}{dt} = f(y(t),\ t) \tag{6.2}$$

modélisant l'équation 6.1. Le problème revient ainsi à la résolution de l'ODE donnée par l'équation 6.2. En utilisant la méthode de Euler implicite pour discrétiser l'équation 6.2, on obtient l'équation 6.3.

$$\frac{y(t+h) - y(t)}{h} = f(y(t+h), t+h) \tag{6.3}$$

Si la fonction f est non linéaire, on ne peut pas trouver directement la solution à partir de cette formule. Dans ce cas, la solution classique consiste à utiliser la méthode itérative de Newton (ou méthode des tangentes) qui permet de trouver les solutions de l'équation $g(x) = 0$ grâce à la formule suivante :

$$x^{k+1} = x^k - \frac{g(x^k)}{g'(x^k)} \tag{6.4}$$

avec x^0 donné. L'équation 6.3 est équivalente à :

$$y(t) + h * f(y(t+h), t+h) - y(t+h) = 0$$

On pose :

$$F(y(t+h),\ y(t), t+h) = y(t) + h * f(y(t+h), t+h) - y(t+h)$$

Résoudre l'équation 6.3 revient donc à trouver $y(t+h)$ tel que :

$$F(y(t+h),\ y(t), t+h) = 0 \tag{6.5}$$

Avec la méthode de Newton, équation 6.4, on trouve les solutions de 6.5 grâce à la formule :

$$y(t+h)^{k+1} = y(t+h)^k - \frac{F(y(t+h)^k,\ y(t), t+h)}{F'(y(t+h)^k,\ y(t), t+h)} \tag{6.6}$$

avec $y(t+h)^0$ donné. En général, une bonne approximation consiste à utiliser l'égalité 6.7 comme condition initiale.

$$y(t+h)^0 = y(t) \tag{6.7}$$

Dans la pratique, pour faciliter le calcul, il est possible d'utiliser $F'(y(t+h)^0,\ y(t), t+h)$ au lieu de $F'(y(t+h)^k,\ y(t), t+h)$ dans l'équation 6.6 (ce qui évite de calculer F' à chaque nouvelle itération). Rappelons que si F est une fonction, alors $F(y)$ est un vecteur

$$F(y) = \begin{cases} f_1(y_1, \ldots, y_n) \\ \ldots \\ f_n(y_1, \ldots, y_n) \end{cases}$$

et $F'(y)$, la Jacobienne de $F(y)$, s'écrit sous la forme :

$$F'(y) = Jacobienne(F(y)) = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial y_1} & \cdots & \frac{\partial f_n}{\partial y_n} \end{bmatrix}$$

Donc dans notre cas, la Jacobienne de $F(y)$ est :

$$F'(y) = \begin{bmatrix} \frac{f_1(y_1+hh,\ldots,y_n)-f_1(y_1,\ldots,y_n)}{hh} & \cdots & \frac{f_1(y_1,\ldots,y_n+hh)-f_1(y_1,\ldots,y_n)}{hh} \\ \vdots & \ddots & \vdots \\ \frac{f_n(y_1+hh,\ldots,y_n)-f_n(y_1,\ldots,y_n)}{hh} & \cdots & \frac{f_n(y_1,\ldots,y_n+hh)-f_n(y_1,\ldots,y_n)}{hh} \end{bmatrix}$$

En posant :

$$dy^{k+1} = y(t+h)^{k+1} - y(t+h)^k,$$

l'équation 6.6 peut s'écrire sous la forme :

$$F'(y(t+h)^0,\ y(t), t+h) * dy^{k+1} = -F(y(t+h)^k,\ y(t), t+h) \tag{6.8}$$

Enfin, résoudre l'équation 6.8 revient à trouver la solution x d'un système linéaire donné par l'équation 6.9.

$$A * x = b \tag{6.9}$$

avec A telle que :

$$A = F'(y(t+h)^0,\ y(t), t+h),$$

x tel que :

$$x = dy^{k+1}$$

et b tel que :

$$b = -F(y(t+h)^k,\ y(t), t+h),$$

Le modèle ainsi défini, il est facilement soluble.

6.2.2 Modélisation de l'application

Pour ce problème, la grille de discrétisation des deux espèces chimiques considérées est stockée dans un vecteur y. Le vecteur y est défini comme suit :

$$y = (c^1_{11}, c^2_{11}, \ldots, c^1_{Mx1}, c^2_{Mx1}, c^1_{12}, c^2_{12}, \ldots, c^1_{1Mz}, c^2_{1Mz}, \ldots, c^1_{MxMz}, c^2_{MxMz}),$$

où Mx et Mz sont respectivement le nombre d'éléments sur l'axe x et l'axe z. Ainsi, une concentration c^i_{jk}, concentration de l'espèce chimique i aux coordonnées jk, est située dans le vecteur y à la position $(j-1+(k-1)*Mx)*2+i$. Les fonctions y_j (avec $j \in \{1, \ldots, 2 * Mx * Mz\}$) permettant de déterminer l'évolution en chaque point de chaque espèce chimique sont donc aussi assimilées à des composantes spatiales par la suite. Afin de mettre en évidence cette modélisation, prenons l'exemple de la

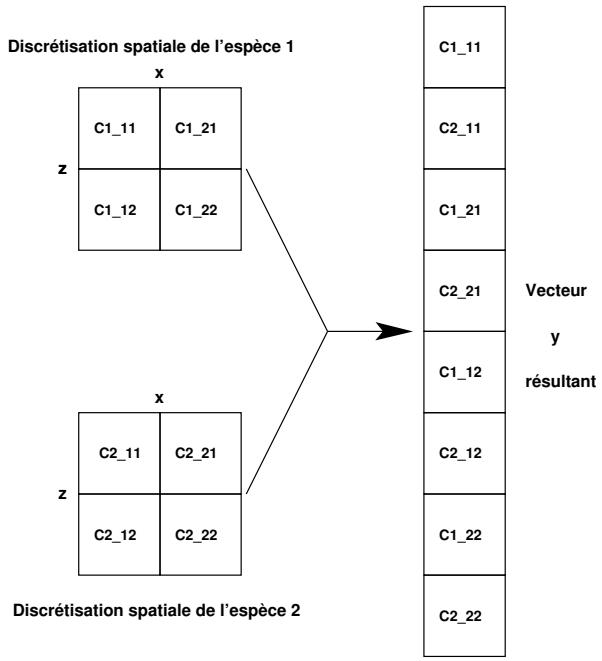


FIG. 6.1 – Discrétisation spatiale de la grille.

figure 6.1 où l'on discrétise l'espace considéré en 2 éléments sur l'axe x ($Mx = 2$) et en 2 éléments sur l'axe z ($Mz = 2$). Le vecteur résultant est donc constitué de 8 éléments (concentrations) notées Ci_jk (ou i représente l'espèce chimique considérée et jk les coordonnées de l'élément).

Une fois le vecteur y construit, la démarche pour résoudre le système d'ODEs est la mise en pratique des différentes étapes présentées en section 6.2.1. L'algorithme s'exécute en plusieurs pas de temps, et chacun de ces pas se découpe en trois

étapes : la première consiste à discrétiser l'ODE avec la méthode implicite d'Euler, la seconde est la déduction d'un système linéaire avec la méthode de Newton, et enfin la résolution du système linéaire résultant. Pour résoudre le système linéaire, nous utilisons la méthode $GMRES$ (Generalized Minimum RESidual) disponible au sein de la bibliothèque de calcul matriciel $sparseLib++$ [28]. Les constantes du problème (pas de temps, intervalle de résolution ...) sont données dans le section 6.2.1, on fixe également un seuil ε sous lequel nous considérons que le système linéaire déduit par la méthode de Newton a convergé. Notons que la résolution de notre problème est réalisée de manière doublement itérative : l'une pour la résolution de Newton et l'autre pour le système linéaire résultant.

La résolution distribuée de ce problème nécessite une répartition des données. Plusieurs méthodes sont possibles, les plus classiques étant les répartitions selon les axes. Pour notre problème, nous optons pour un découpage selon l'axe z, ce dernier est le plus simple au niveau répartition de données et présente également un avantage au niveau de l'équilibrage. Les autres méthodes classiques consistent à découper le problème selon l'axe x ou selon les deux axes en même temps. La découpe choisie engendre une topologie de communication de type ligne : un processeur a toujours deux voisins, un droite et un gauche, sauf pour les extrémités. L'équilibrage de charge s'effectue donc sur une seule dimension et est simple à mettre en place, dans le cas d'une découpe selon les deux axes, l'équilibrage s'effectue en deux dimensions et sous ces conditions, sa mise en place est nettement moins triviale. Ce problème d'équilibrage de charge en deux dimensions est présenté dans [55]. La figure 6.2 illustre cette découpe selon z pour deux processeurs. Elle présente le point de vue physique, découpe de la grille de discrétisation, ainsi que celui de l'implantation, découpe du vecteur y. Pour notre application, nous répartissons les données de manière uniforme sur les différentes machines coopératives. Notons pour finir que la résolution distribuée de l'application nécessite quelques modification par rapport a celle séquentielle présentée en section 6.2.1, le détail de ces modification est présenté dans [15].

6.3 L'équilibrage de charge et l'application

Suite à cette présentation de l'application, intéressons nous à la détermination de l'algorithme d'équilibrage de charge à utiliser et aux différents problèmes soulevés au cours de son implantation.

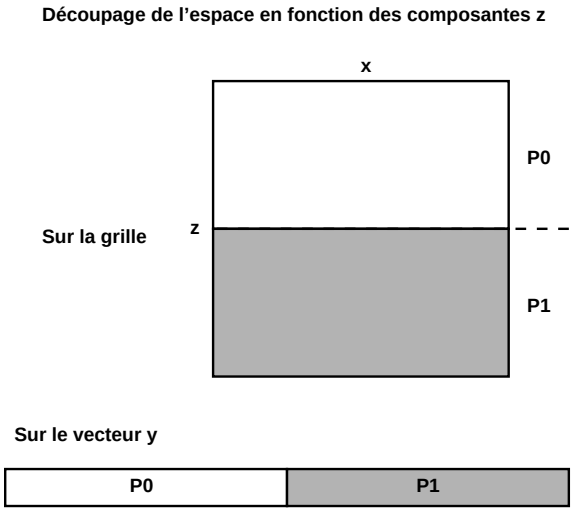


FIG. 6.2 – Répartition des éléments de y sur les différents processeurs.

6.3.1 Mise en place de l'équilibrage

L'application d'advection-diffusion présentée en section 6.2 nous est fournie avec une allocation statique et uniforme des données. Notre objectif est de mettre en place un système d'équilibrage de charge dans cette application afin d'optimiser le temps de calcul. Le rôle de l'équilibrage de charge est de combler les différences de puissance de calcul disponible sur chaque processeur. Ces différences peuvent être naturelles (un processeur plus puissant qu'un autre) ou dues à des charges extérieures à notre application. La première étape afin d'introduire un algorithme d'équilibrage de charge dans cette application est la détermination de la charge. Depuis le début de ce mémoire, nous parlons de charge sans jamais réellement définir cette notion. Nous savons que la notion de charge processeur n'est pas définie de la même manière selon le système d'exploitation considéré. De plus, dans le contexte SPMD (Single Program Multi Data), contexte de l'application, la charge processeur déterminée par le système d'exploitation n'est pas des plus adaptée. Cette notion de charge processeur est plus appropriée aux systèmes multithreadés et à l'équilibrage (ou répartition) de charge par migration de thread (processus léger). Dans notre cas nous avons donc choisi de déterminer la charge induite par l'application sur chaque processeur. Cette charge devant être définie de la manière la plus simple possible, nous posons que pour un processeur donné, la charge induite par l'application est le nombre de données (concentrations) traitées par ce processeur. Notons que pour l'application, à la diférence des études précédentes, la charge d'un processeur est

entière.

La charge étant ainsi définie, l'algorithme d'équilibrage doit répartir la charge, donc les données, de telle sorte que le temps global d'exécution soit réduit. Cette répartition s'effectue en fonction de la puissance disponible de chaque processeur. La puissance de chaque processeur n'est pas toujours connue et n'inclue pas les charges extérieures pouvant consommer de la puissance de calcul, nous avons donc redéfini la puissance de calcul disponible de chaque processeur. Cette définition consiste simplement à prendre le temps moyen nécessaire à l'application pour effectuer une itération de Newton. Cette moyenne est initialisée à chaque pas de temps. Connaissant, pour un processeur donné, la charge de notre application (le nombre de données à traiter) et le temps nécessaire pour effectuer une itération de Newton sur ces données, nous connaissons donc la puissance disponible sur ce processeur pour l'application. L'équilibrage s'effectue à chaque pas de temps comme le montre la figure 6.3

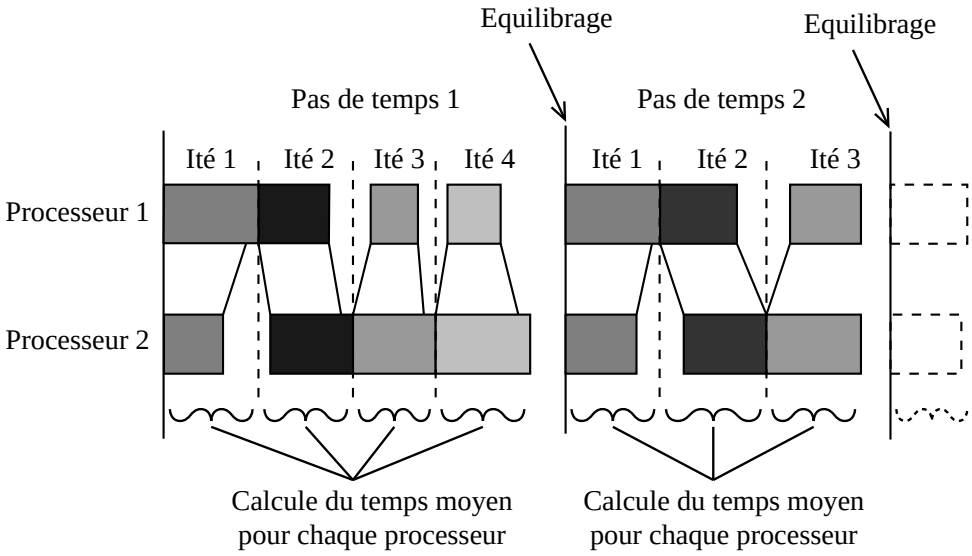


FIG. 6.3 – Équilibrage dans un contexte synchrone.

Ces deux définitions présentent différents avantages. Tout d'abord, l'algorithme d'équilibrage de charge nous donne la quantité de charge à déplacer entre deux processeurs donnés. Notre définition de la charge nous permet donc de savoir quelle quantité de données est à déplacer entre ces processeurs. Ceci n'est pas aussi trivial si la charge considérée est celle donnée par le système d'exploitation. Notre définition de la puissance des différentes machines s'apparente plus particulièrement à la puissance disponible pour notre application. Cette puissance disponible prend naturel-

lement en compte, de manière indirecte, la puissance de la machine, les charges extérieures à notre application et la mémoire disponible. Si une charge extérieur vient perturber notre calcul ou si la mémoire disponible n'est pas suffisante, l'application est ralentie et la puissance disponible est amoindrie. Ce système présente également un autre avantage : la puissance de calcul disponible est calculée à chaque pas de temps, nous obtenons donc une puissance de calcul dynamique.

Ayant défini les différents paramètres nécessaires à l'équilibrage de charge, il nous faut à présent déterminer quel algorithme est à utiliser. Selon le découpage que nous avons choisi pour l'application, la topologie de communication résultante est une ligne. L'application est totalement distribuée et toutes les communications qui lui sont nécessaires respectent la topologie. De plus la taille de la topologie n'est pas imposée et nos expérimentations nous amènerons à tester différentes tailles. Nous optons ainsi pour une méthode d'équilibrage ne nécessitant pas de connaissance globale du système pour conserver le caractère distribué de l'application : ni méthode optimale nécessitant un calcul avant l'équilibrage, ni méthode relaxée ou de second ordre. Nous avons donc le choix entre la diffusion de premier ordre et GDE. En nous reportant au chapitre 3 nous trouvons que pour une ligne la valeur optimale de α, paramètre de la diffusion, est connue et est $1/2$; nous remarquons également que, dans le cas d'une ligne, la diffusion optimale est équivalente à GDE avec un choix inné de λ. Il en résulte que le choix entre ces deux méthodes est indifférent, nous optons donc pour une méthode de diffusion de premier ordre avec un choix optimal de α.

6.3.2 Gain apporté par l'équilibrage

Suite à l'implantation de l'algorithme d'équilibrage dans l'application d'advection-diffusion, nous avons réalisé différentes expériences. Elles ont été déployées sur un réseau local à 100Mb, avec des stations de travail de puissances différentes (2.6GHz, 2.4GHz et 1.7GHz). Tout d'abord nous avons mis en évidence l'impact de l'équilibrage de charge sur le temps d'exécution de l'algorithme. Pour ce faire, nous avons réalisé sur quatre machines de puissances différentes des simulations avec des tailles de problèmes différents. Les résultats obtenus pour ces exécutions sont présentés dans le tableau 6.1. Ce tableau donne les temps moyens en secondes pour l'exécution de l'algorithme sans équilibrage, avec équilibrage et avec équilibrage une fois sur trois, c'est à dire qu'au lieu d'équilibrer à tous les pas de temps, nous lançons le système d'équilibrage tous les trois pas de temps. Pour les deux expérimentations avec équilibrage, nous donnons également le gain de temps relatif à la version sans équilibrage. Plusieurs observations peuvent être déduites de ces résultats. En premier lieu, on remarque que de manière générale plus la taille du problème est importante,

taille : x x x	100	200	300	400	500	600	700
sans équilibrage	11.4	75.3	266	628	1391	2656	4541
équilibrage (1)	11	72.7	253	603	1243	2285	3774
gain (1)(%)	3	3	5	4	10	14	17
équilibrage 1/3 (2)	10.8	72.8	255	608	1261	2326	3766
gain (2)(%)	3	3	5	3	9	12	17

TAB. 6.1 – Temps d'exécution du problème d'advection-diffusion selon différentes tailles de problèmes et sans charge extérieure.

plus le gain apporté par l'équilibrage l'est aussi. Ce qui est tout à fait normal : l'équilibrage étant itératif, il lui faut un certain temps afin d'équilibrer correctement le système. En outre, plus la taille du problème est grand, plus la répartition uniforme initiale donne un système déséquilibré relativement à la puissance des machines. Dans le cas de problèmes de petit taille, l'algorithme d'équilibrage n'a pas le temps d'équilibrer correctement le système avant que l'application ne se termine, mais il ne ralentit toutefois pas l'exécution du problème. Dans un second temps on notera que le gain apporté par l'équilibrage devient intéressant à partir d'une taille de problème de 500x500. Pour une taille de problème de 500x500 ou 600x600, le fait d'équilibrer une fois sur trois est très légèrement moins rapide que d'équilibrer à chaque pas de temps et pour une taille de 700x700 les deux méthodes sont équivalentes. Ceci s'explique de la même manière que précédemment : le fait d'équilibrer une fois sur trois prend légèrement plus de temps pour atteindre l'équilibre, mais lorsque l'équilibre est atteint le surcoût de l'équilibrage est moins important. Le dernier point que nous abordons à propos de ces résultats est la légère baisse de gain pour une taille de 400x400 relativement à une taille de 300x300. Cette perte de performance est liée aux caractéristiques de l'application. Ce point est plus particulièrement détaillé dans le chapitre 7 où son influence est plus importante que dans le contexte synchrone.

Nous avons réalisé les expériences précédentes dans un cadre quasi idéal, les machines utilisées nous sont quasiment dédiées et aucune charge extérieure ne vient perturber l'application, hormis quelques applications inactives qui génèrent une charge négligeable. Étudions donc le comportement de l'équilibrage de charge lorsqu'une charge extérieure vient perturber l'application. Pour ces simulations, lançons sur une machine participant au calcul un programme générant de la charge. La consommation de puissance du processeur (cpu) engendrée par ce programme est représentée par la figure 6.4. Il consomme 50% du cpu pendant 10s toutes les 20s. Les expérimentations précédentes ont été relancées avec cette charge extérieure. Les résultats obtenus sont présentés dans le tableau 6.2. La première remarque que

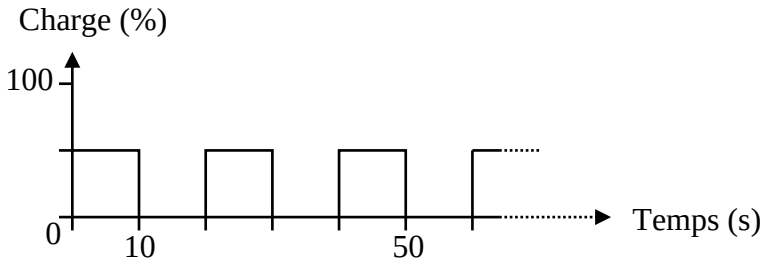


FIG. 6.4 – Évolution de la charge extérieure en créneaux.

taille : x x x	100	200	300	400	500	600	700
sans équilibrage	15.1	111	391	856	1748	3696	5520
équilibrage (1)	12.8	94	364	908	1859	3222	5430
gain (1)(%)	15	16	7	-6	-6	13	2
équilibrage 1/3 (2)	13.4	92	365	908	1740	3057	5383
gain (2)(%)	11	17	7	-6	0	17	2

TAB. 6.2 – Temps d'exécution du problème d'advection-diffusion selon différentes tailles de problèmes et avec une charge extérieur en créneaux.

nous pouvons faire concerne les deux problèmes de petite taille : même si le temps d'exécution est court, le gain apporté par l'équilibrage est relativement important par rapport à celui apporté dans le contexte d'exécution précédent. Ceci s'explique par la taille du problème et la nature de la charge extérieure : pendant une période de charge extérieure, soit $10s$, l'exécution du problème réalise plusieurs pas de temps du fait de sa petite taille, le système d'équilibrage réussit donc à gagner du temps sur cette période, de même pour la période de $10s$ sans charge extérieure. Par contre, pour les tailles de problèmes supérieures à 200x200, on remarque que le gain apporté par l'équilibrage diminue et devient négatif. L'explication de ce phénomène est la même que précédemment, la différence étant que la taille du problème est plus importante, le temps d'un pas de temps est donc plus grand. On remarque en particulier pour une taille de problème de 400x400 que le temps nécessaire pour réaliser un pas de temps est d'environ $20s$, soit la période de notre charge extérieure. Sous ces conditions, le système d'équilibrage voit la puissance cpu disponible du nœud surchargé comme 75% de sa puissance cpu totale : 50% consommés pendant $10s$ et 0% pendant $10s$ soit 25% consommés pendant $20s$. Le système d'équilibrage équilibre donc les charges en conséquence et se stabilise : il converge vers un équilibre appa-

rent. En fait le nœud ayant la charge extérieure se retrouve surchargé pendant $10s$ et sous chargé les $10s$ suivantes. Le problème de l'oscillation de la charge extérieure se retrouve également dans les expériences avec une taille de problème supérieure, ainsi que dans le cas où l'on n'équilibre qu'une fois sur trois ; dans ce cas, le phénomène se décale légèrement. Notons toutefois que la charge extérieure utilisée est intéressante dans un cas d'étude mais n'est pas représentative d'une charge extérieure créée de manière naturelle.

6.4 Conclusion

Ce chapitre introduit concrètement la mise en place d'un algorithme d'équilibrage de charge dans une application. L'application que nous avons choisie est le problème d'advection-diffusion, une application de cinétique chimique modélisant l'évolution de la concentration de deux espèces chimiques dans le temps et l'espace. Nous présentons ce chapitre en deux parties : tout d'abord, nous présentons l'application et son implantation parallèle. Puis nous expliquons le cheminement afin d'inclure l'équilibrage de charge dans ce problème et présentons les différents résultats d'exécution de cette application avec et sans équilibrage. Au regard des résultats obtenus, nous pouvons dire que l'équilibrage de charge apporte quasi toujours un avantage dans notre contexte d'exécution. Hormis dans le cas de l'introduction d'une charge extérieure périodique où certaines configurations sont ralenties par l'équilibrage de charge, toutes les autres configurations sont accélérées par l'équilibrage. De plus, ces quelques configurations pénalisées par l'équilibrage le sont du fait de la périodicité de la charge extérieure, périodicité improbable dans un contexte réel.

Les expérimentations présentées dans ce chapitre ne sont pas réalisées avec des pertes de liens, ceci est dû au caractère synchrone de l'exécution ; caractéristique peu compatible avec un réseau dynamique. Dans le chapitre suivant, nous considérerons une implantation asynchrone de l'application sur un réseau dynamique pour mettre en évidence l'apport de nos algorithmes sur cette application.

Chapitre 7

Application sur Réseaux Dynamiques

7.1 Introduction

Suite à l'étude du comportement de l'application d'advection-diffusion sur un réseau statique, nous nous penchons sur son comportement sur réseaux dynamiques. La version synchrone qui nous a été fournie n'est pas compatible avec le dynamisme du réseau : si un seul lien de communication est temporairement coupé ou surchargé, toute l'application est bloquée ou ralentie. Pour être compatible avec le dynamisme du réseau, nous utilisons donc une version asynchrone qui nous a également été fournie. Cette version de l'application fonctionne avec une version MPI dédiée aux applications asynchrones : Heterogeneous MPI. Cette dernière est détaillée dans l'article [5]. Cette version asynchrone de l'application diffère en trois points de celle synchrone. Premièrement au niveau des communications, elles sont, dans la version présente, non bloquantes et dites asynchrones. Deuxièmement au niveau des itérations de Newton, elles sont réalisées sans synchronisation avec les processeurs voisins. Pour finir, un nouvel algorithme de détection de convergence est utilisé. La structure des deux programmes reste toutefois très proche, l'équilibrage de charge est ainsi implanté de la même manière dans les deux versions. La question de l'intérêt de l'équilibrage de charge dans les algorithmes asynchrones peut se poser, ce problème a été étudié dans [8, 6]. En ce qui concerne la détection de convergence, nous avons mis en place un système qui conserve le caractère distribué et asynchrone de l'application.

Ce chapitre se compose de deux parties, dans un premier temps la section 7.2 présente les modifications apportées à l'algorithme et détaille plus précisément le principe de la détection de convergence utilisée. Une seconde section 7.3 s'attache

plus précisément à l'application de l'équilibrage de charge sur un réseau dynamique et aux différents problèmes rencontrés.

7.2 Résolution asynchrone du problème d'advection-diffusion

Intéressons nous dans un premier temps aux adaptations nécessaires à la mise en place de l'algorithme dans un contexte asynchrone.

7.2.1 Modifications apportées à l'application

La résolution d'un problème itératif de manière asynchrone présente différents avantages. La résolution asynchrone d'un problème itératif distribué consiste à effectuer sur chaque processeur les successions d'itérations indépendamment de ses voisins. En d'autres termes, lorsqu'un processeur a fini une itération, il commence la suivante sans attendre les résultats calculés par ses voisins. Ces résultats sont pris en compte de manière dynamique à l'intérieur d'une itération. Cette méthode permet d'éliminer tous les temps morts induits par les envois et les attentes de messages de la méthode synchrone. Mais le point le plus important est la tolérance aux pertes ou retards de messages : en asynchrone, si un message se perd ou est fortement retardé du fait d'une coupure ou d'une surcharge du réseau, l'algorithme converge toutefois vers la solution. Cette tolérance aux pertes de messages est cependant régie par des conditions qui correspondent à la réalité. Différentes études mettent en évidence les avantages de cette méthode [7, 14].

Notons toutefois que tout problème itératif ne peut pas forcément être résolu de manière asynchrone, cette méthode de résolution est présentée dans [20]. Dans le cas d'advection-diffusion, une étude préalable que nous ne détaillerons pas, montre que notre problème peut être résolu en partie de manière asynchrone.

Comme nous l'avons vu dans la section 6.2, notre problème est doublement itératif. Dans notre cas, seules les itérations de Newton peuvent être effectuées de manière asynchrone, le problème reste cependant synchrone au niveau des pas de temps. La figure 7.1 illustre la résolution asynchrone d'advection-diffusion. D'un point de vue implantation, seul le système de communication change, afin de permettre l'envoi et la réception de données au cours du calcul.

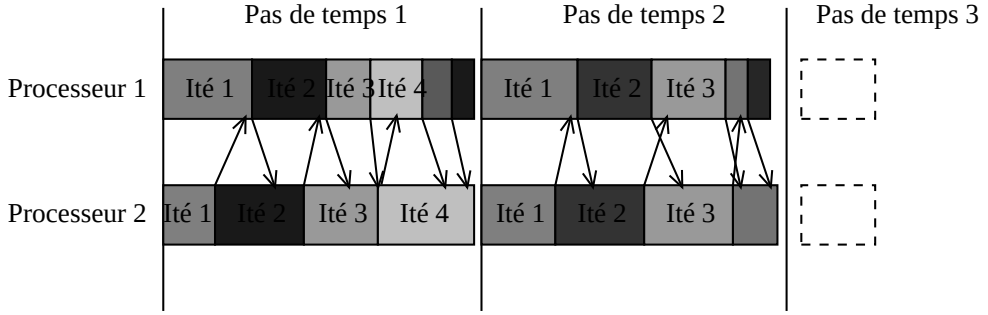


FIG. 7.1 – Résolution asynchrone d'advection diffusion.

7.2.2 Nouvel algorithme de détection de convergence

La détection de convergence dans l'exécution asynchrone de problèmes pose quelques problèmes non négligeables. Dans un contexte synchrone, cette phase de la résolution est connue et différentes méthodes sont utilisées [37, 27, 61, 71]. Les plus communes sont centralisées ou en "all to all". Dans le cas centralisé, au cours d'une phase de synchronisation, les différents processeurs coopératifs communiquent leur état à un seul processeur, généralement le premier, qui vérifie si tous les processeurs ont convergé : si tous ont convergé, il indique aux différents processeurs l'arrêt du calcul, sinon il les laisse recommencer une nouvelle itération. Dans le second cas, chaque processeur vérifie l'état du système après réception de l'état de chaque processeur, et stoppe le calcule si tous ont convergé. Dans un contexte asynchrone, il n'y a généralement pas de phase de synchronisation ; ces systèmes ne peuvent donc pas être appliqués simplement. Différentes études [19, 63, 23] proposent des algorithmes de détection de convergence pour les contextes asynchrones, mais ils sont soit centralisés, soit fortement contraignant. C'est pourquoi nous avons mis en place un système distribué de détection de convergence, que nous présentons dans [9], qui ne nécessite pas de synchronisation. De plus, cette algorithme reste cohérent avec le caractère distribué des application qu'il doit contrôler, ceci n'est généralement pas le cas dans la plupart des algorithmes de détection de convergences que nous pouvons trouver dans la littérature.

Le principe de base que nous utilisons est celui utilisé dans l'élection de leader du protocole IEEE1394 (Firewire) [64] ou dans certains systèmes distribués [3, 29]. Cet algorithme d'élection de leader est dédié à la détermination d'un maître dans un réseau de type arbre. Le fonctionnement de cet algorithme est assez simple : tous les nœuds de l'arbre envoient un message à leur unique voisin qui ne les a pas contactés. L'algorithme commence donc au niveau des feuilles de l'arbre car seuls ces nœuds

ont à la base un seul voisin. Les flèches numérotées (1) de la figure 7.2 présentent cette première étape. Pour l'exemple considéré, l'étape suivante - flèches numérotées

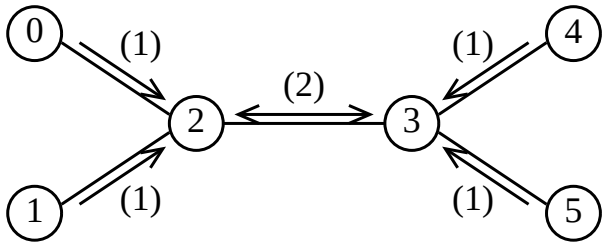


FIG. 7.2 – Pas de l'élection de leader dans le protocole IEEE1394.

(2) - présente un conflit : l'élection de leader se termine normalement lorsque un unique nœud a été contacté par tous ses voisins et n'a, en conséquence, plus personne à contacter. Cet unique nœud est déclaré leader (ou racine) de l'arbre. Dans notre cas, deux nœuds sont dans cette configuration (nœuds 2 et 3). L'algorithme définit alors arbitrairement l'un des deux comme leader. Il résulte de cet algorithme une arborescence où la racine est le leader. La figure 7.3 présente l'arborescence résultante, où le nœud 2 a été arbitrairement pris comme leader.

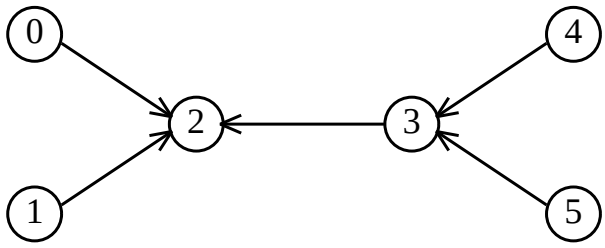


FIG. 7.3 – Arborescence résultant de l'élection de leader dans le protocole IEEE1394.

Afin de détecter la convergence grâce au principe de l'élection de leader que nous venons de présenter, nous avons modifié l'algorithme. D'une part, un nœud i n'envoie un message à son unique voisin qui ne l'a pas contacté que s'il est passé en dessous du seuil de convergence. Nous nommons le nœud destinataire de ce message nœud père de i. Notons toutefois qu'un nœud qui est passé sous le seuil de convergence peut repasser au dessus de ce seuil suite à la prise en compte de nouvelles données émises par ses voisins. D'autre part, nous avons également mis en place un système d'annulation d'information de convergence : si un nœud passe au dessus du

seuil de convergence, il informe son nœud père qu'il est repassé au dessus du seuil, le nœud père considère donc que ce fils ne l'a pas contacté. Si nécessaire, le nœud père contacte son propre père afin de faire remonter l'information. Pour éviter un phénomène d'oscillations autour du seuil de convergence et l'envoi successif d'information de convergence et d'annulation, un nœud n'envoie à son père un message comme quoi il est passé sous le seuil de convergence que s'il est resté sous ce seuil depuis plusieurs itérations malgré la réception de mises à jour de données de ses voisins. Ainsi, lorsqu'un nœud i passe sous le seuil de convergence et que tous ses voisins l'ont déjà contacté, ceci signifie que tous les nœuds du calcul ont convergé. Le nœud i informe donc ses fils de stopper le calcul, ceux-ci font de même avec leurs propres fils etc... S'il y a conflit comme nous l'avons vu dans l'élection de leader, ceci ne pose pas de problème dans notre cas : les deux nœuds informent leurs fils, ils s'informent mutuellement de la convergence du système mais ne répercutent l'information qu'une seule fois à leurs fils. Pour être efficace, le système de détection de convergence tourne en parallèle du calcul proprement dit : il est géré par un processus léger. Ce processus tient à jour les informations de convergence des voisins et, à la fin de chaque itération, le calcul vérifie son état par rapport au seuil de convergence et informe son futur père si nécessaire.

Notons que, d'une part, si le réseau n'est pas un arbre, un arbre de recouvrement du graphe doit être utilisé pour la détection de convergence ; d'autre part, bien que cet algorithme ait été développé pour la détection de convergence dans les algorithmes asynchrones, il peut tout à fait s'appliquer à un contexte synchrone.

7.3 Implantation de l'équilibrage de charge

Étudions à présent l'effet de l'équilibrage de charge sur cette application, ainsi que les différents problèmes rencontrés. Bien que l'application soit asynchrone, l'algorithme d'équilibrage utilisé est synchrone ; il existe différentes applications asynchrones des algorithmes d'équilibrage de charge [73, 20], mais nous ne les étudions pas ici.

7.3.1 Équilibrage classique

La première implantation de l'équilibrage de charge dans l'application d'advection-diffusion asynchrone est effectuée de la même manière que dans la version synchrone. La notion de charge induite par l'application est le nombre d'éléments que doit traiter un processeur, et la puissance disponible sur un processeur donné consiste à relever le temps moyen nécessaire à ce processeur pour effectuer une itération. Les

premières simulations que nous avons mis en place sont les mêmes que pour la version synchrone, c'est à dire sur le même réseaux, avec les mêmes machine et pour les mêmes tailles du problème. Nous lançons sur quatre machines de puissances différentes l'exécution du problème avec une taille allant de 100x100 à 700x700. Les résultats obtenus sont présentés dans le tableau 7.1. Notons que ces résultats ne sont pas comparables avec ceux obtenus avec la version synchrone : les paramètres initiaux utilisés ne sont pas les mêmes. Notre objectif est de comparer les versions avec et sans équilibrage. Une comparaison entre les versions synchrone et asynchrone est présentée dans l'article [14]. Les résultats obtenus nous montrent que

taille : x x x	100	200	300	400	500	600	700
sans équilibrage	22.8	117	396	936	1757	3283	5066
équilibrage	22.5	119	413	933	1862	3232	5000
gain (%)	1	-1	-4	0	-6	2	1

TAB. 7.1 – Temps d'exécution d'advection-diffusion selon différentes tailles de problème et sans charge extérieure.

l'équilibrage de charge n'a rien apporté au niveau gain de temps à notre application, au contraire, dans la plupart des cas il ralentit l'exécution de l'algorithme. La question qu'on se pose alors est : pourquoi lorsqu'on équilibre la répartition de données de manière à prendre en compte les différences de puissance disponible sur chaque machine, l'exécution du problème est elle ralentie ? En observant le comportement de l'équilibrage de charge et le temps moyen d'un pas de Newton, on remarque que la duré moyenne d'un de ces pas tend à s'uniformiser sur toutes les machines. L'algorithme d'équilibrage de charge effectue donc bien son travail, mais ne nous fait pas gagner de temps, et peut ralentir l'exécution de l'application. Afin de comprendre la raison de ce phénomène, nous devons étudier le comportement du problème.

Nous savons que dans un pas de temps, les itérations de Newton se terminent lorsque toutes les machines participant au calcul ont convergé, c'est à dire lorsque la différence entre deux pas de Newton est inférieure à un seuil fixé. En regardant le comportement de l'algorithme, on remarque que tous les processeurs ne passent pas sous ce seuil en même temps. Bien au contraire, l'erreur, c'est à dire la différence de précision entre deux pas de Newton, est nettement plus importante sur le dernier processeur que sur le premier. Ceci signifie que le premier processeur passe très rapidement sous le seuil de convergence alors que le dernier met beaucoup plus de temps pour converger. On en déduit donc que le dernier processeur impose le temps de convergence de Newton pour toute l'application. Revenons à notre système d'équilibrage : le résultat de notre équilibrage est un temps uniforme de pas de

Newton sur chaque machine. Ceci ne change pas le fait que le temps de convergence de Newton est imposé par le dernier processeur. Il en résulte donc que, passé un certain nombre d'itérations, tous les processeurs sont quasiment synchronisés mais que seul le dernier réalise des calculs utiles, puis qu'il est le dernier à ne pas avoir convergé. Tous les autres sont sous le seuil de convergence et calculent uniquement pour attendre le dernier. Le gain apporté par l'équilibrage est donc infime et son coût est généralement plus important. Suite à ces observations, nous abandonnons cette méthode d'équilibrage pour nous focaliser sur un système basé sur l'erreur, afin que tous les processeurs convergent en même temps.

7.3.2 Équilibrage selon l'erreur

Les observations réalisées en section 7.3.1 nous ont donc amené à réaliser l'équilibrage non pas selon le temps moyen d'un pas de Newton, mais sur celui nécessaire à un processeur pour passer et rester sous le seuil de convergence de Newton. Afin de mettre en place ce système nous avons redéfini la puissance de calcul disponible sur chaque processeur. Dans la version précédente, la puissance disponible est définie comme étant le temps nécessaire pour réaliser un pas de Newton sur toutes les données du processeur. Dans cette nouvelle version, la puissance disponible est le temps nécessaire pour passer et rester sous le seuil de convergence pour le nombre de données traitées. Ceci entraîne une unique modification dans notre système d'équilibrage de charge : au lieu de calculer le temps moyen d'une itération de Newton, nous prenons le temps nécessaire pour qu'un processeur passe définitivement sous le seuil de convergence. Les expériences réalisées précédemment ont donc été relancées avec cette modification. Les résultats obtenus sont présentés dans le tableau 7.2. Ces résultats mettent bien en évidence l'apport de l'équilibrage

taille : x x x	100	200	300	400	500	600	700
sans équilibrage	22.8	117	396	936	1757	3283	5066
équilibrage	20	94	307	692	1418	2930	4525
gain (%)	12	20	22	26	19	11	11

TAB. 7.2 – Temps d'exécution d'advection-diffusion selon différentes tailles de problèmes et sans charge extérieure.

de charge lorsque ce dernier est réalisé selon l'erreur. Le gain varie entre 10% et plus de 25% pour les expérimentations réalisées. Cependant, on observe une baisse du gain pour les tailles supérieures à 400x400. Pour expliquer ce phénomène, une étude

approfondie du comportement de l'application avec équilibrage est nécessaire. Rappelons que l'erreur n'est pas proportionnelle à la quantité de données traitées : les derniers processeurs effectuent plus d'itérations que les premiers afin que le processus de Newton converge. Dans notre cas, avec quatre processeurs, la différence entre les deux premiers et les deux derniers est très importante. Au cours de l'exécution du problème, l'avant dernier processeur se décharge rapidement sur le second. La différence de puissance disponible entre ces deux processeurs est telle que l'avant dernier se retrouve très vite avec peu de données à traiter. La conséquence de cette configuration est la suivante : l'avant dernier processeur a peu de données, ça convergence est donc fortement liée aux mises à jour qu'il reçoit de ses voisins, et tout particulièrement du dernier processeur. Nous savons que ce dernier processeur est le plus lent à converger. Il envoie donc, jusqu'à sa convergence, des mises à jour à l'avant dernier. L'avant dernier processeur ayant peu de données à traiter, il passe rapidement sous le seuil de convergence, mais à chaque réception de mises à jour il repasse au dessus de ce seuil. Par conséquent, l'avant dernier processeur oscille autour du seuil de convergence et ne converge que lorsque le dernier lui envoie sa dernière mise à jour. Les deux derniers processeurs convergent donc en même temps. Selon nos définitions de la charge et de la puissance de calcul disponible, si deux processeurs convergent en même temps c'est qu'ils sont équilibrés. Or, dans notre configuration, le temps de convergence de l'avant dernier processeur est imposé par le dernier. Lorsque le système d'équilibrage se stabilise, les deux premiers processeurs sont équilibrés, ainsi que les deux derniers, mais le système complet ne l'est pas totalement. L'algorithme d'équilibrage ne parvient pas à équilibrer tout le système puisque, d'une part l'avant dernier n'a plus assez de données pour en transmettre au second et d'autre part, le dernier et l'avant dernier convergent toujours en même temps malgré la différence de quantité de données, l'algorithme d'équilibrage les considère donc comme équilibré. Nos définitions de la charge et de la puissance de calcul disponible ne sont donc pas idéales et ne permettent pas un équilibrage parfait de l'application. Cependant, elles donnent un gain de temps d'exécution non négligeable. Pour en revenir à la perte de gain au delà d'une taille de 400x400, nous notons que plus la taille du problème est importante plus le déséquilibre final l'est aussi. Le gain apporté par l'équilibrage est donc relativement moins important lorsque la taille du problème est grande. En ce qui concerne les petites tailles du problème, le phénomène de déséquilibre n'a pas le temps d'apparaître entièrement et d'influer sur le gain, le gain croît donc avec la taille.

7.3.3 Équilibrage avec perte de liens

Notre objectif étant l'équilibrage de charge sur réseaux dynamiques, étudions le comportement de notre application lorsque le réseau n'est pas fiable à 100%. Puisque les bibliothèques de communication ne supportent pas, à l'heure actuelle, la perte de messages, nous simulons le dynamisme du réseau. Différentes contraintes sont à respecter : les algorithmes asynchrones supportent la perte de messages puisque ces messages sont des mises à jour de données, mais ils ne supportent pas la perte définitive d'un lien de communication. Si un ou plusieurs messages sont perdus l'algorithme utilisera une mise à jour ultérieure, mais il ne peut pas fonctionner sans mise à jour de données de ses voisins. L'équilibrage de charge quant à lui ne supporte pas la perte de messages, puisqu'il est synchrone et que les données sont transférées pour être traitées sur un autre processeur. La perte de message dans ce cas signifie la perte de données et non la perte de mises à jour. L'algorithme d'équilibrage de charge doit connaître à priori les liens inutilisables afin de ne pas s'exécuter sur ceux-ci. Pour respecter ces contraintes, notre simulateur de réseaux dynamiques ne coupe pas les liens de communication mais ralentit fortement les envois de messages par des temporisations. Ainsi, pour la partie asynchrone, un message est perdu si une nouvelle mise à jour est envoyée alors que l'ancienne n'est pas encore partie du fait d'un ralentissement et, du point de vue équilibrage de charge, un lien ralenti est considéré comme inutilisable.

En ce qui concerne la mise en pratique, nous posons que tous les liens ne sont pas fiables. D'un point de vue pratique, ceci revient à considérer que toutes les machines sont distantes. L'équilibrage mis en place est naturellement selon l'erreur, la première méthode étudiée n'étant pas efficace. Afin de reproduire facilement le dynamisme du réseau, nous posons que les liens sont ralentis tous les deux pas de temps. Dans un premier temps nous réalisons les expérimentations avec différentes tailles de problèmes sur quatre machines de puissances différentes comme précédemment. Les temps moyens nécessaires à l'exécution sont présentés dans le tableau 7.3. On re-

taille : x x x	100	200	300	400	500	600	700
sans équilibrage	45.2	727	1777	2487	4090	7203	9630
équilibrage	43.9	719	1413	2148	3605	6314	8210
gain (%)	3	1	20	14	12	12	14

TAB. 7.3 – Temps d'exécution d'advection-diffusion selon différentes tailles de problèmes, sans charge extérieure et avec pertes de liens.

marque tout d'abord que, hormis pour les deux plus petits problèmes, le dynamisme

du réseau perturbe peu le gain apporté par l'algorithme d'équilibrage. Concernant les deux premiers problèmes, leur temps d'exécution est court et l'équilibrage ralenti par le dynamisme du réseau ne permet pas de gagner de temps. Pour les autres tailles de problèmes, le gain apporté par l'algorithme d'équilibrage est quasi équivalent à ceux obtenus dans la version sur réseaux statiques. En comparant ces résultats avec ceux obtenus sur réseaux statiques, on observe que l'influence du dynamisme du réseau influe de manière importante sur les temps d'exécution du programme. Ceci est dû à un double impacte du dynamisme. Premièrement, puisque les communications sont ralenties, le programme l'est aussi. Deuxièmement, le ralentissement des communications implique moins de mises à jour envoyées et donc un temps de convergence de Newton plus important.

Pour terminer et confirmer nos résultats, nous déployons l'application sur un réseau de dix machines hétérogènes. Le réseau initial est à 100Mb et est libre de charge extérieure. Toutefois, pour nos expérimentations, nous posons que les cinq dernières machines sont distantes et que, comme précédemment, le réseau est surchargé tous les deux pas de temps pour ces machines. Nous disposons dans ce réseau de trois types de machines (800, 1700 et 2400 Mhz), qui sont organisées de manière entrelacée. Les expérimentations sont réalisées pour des tailles de problèmes entre 200x200 et 600x600. La taille 100x100 est trop petite pour le nombre de machines et celle de 700x700 prend trop de temps à être exécutée sur le réseau dynamique. Une première exécution de l'application nous a montré que le problème engendré par notre calcul de la puissance disponible selon l'erreur devient contraignant. Dans les expérimentations précédentes nous n'utilisons que quatre machines et l'impact de ce problème de définition de la puissance gène peu l'équilibrage. Dans le cas présent, nous travaillons sur dix machines et ce problème de définition a nettement plus d'impact. Nous avons donc légèrement modifié la définition de la puissance disponible sur chaque machine. Dans la version précédente, la puissance disponible est le temps nécessaire pour passer et rester sous le seuil de convergence selon le nombre de données traitées. Dans cette nouvelle version, la puissance disponible est le temps nécessaire pour passer pour la première fois sous le seuil de convergence selon le nombre de données traitées. On considère ainsi que si un processeur sous le seuil de convergence repasse au-dessus de ce seuil, cela est dû à la réception de données d'un de ses voisins qui n'est pas encore passé sous le seuil. Si ce voisin n'est pas encore passé sous le seuil, c'est qu'il est plus lent que le processeur courant et, par conséquent, qu'ils doivent s'équilibrer entre eux. Les résultats obtenus sont présentés dans le tableau 7.4. Ce tableau donne, à des fins de comparaisons, les exécutions avec et sans équilibrage, et sur un réseau statique et un dynamique. Dans un premier temps, les résultats obtenus sur le réseau statique montrent que notre nouvelle définition de la puissance disponible sur chaque machine est plus efficace

taille : x x x		200	300	400	500	600
Réseau statique	sans équilibrage	205	580	1542	3199	5920
	équilibrage	171	467	1172	2382	4158
	gain (%)	17	19	24	26	30
Réseau dynamique	sans équilibrage	1281	2580	3892	6456	10085
	équilibrage	970	2306	3868	6047	8935
	gain (%)	24	11	1	6	11

TAB. 7.4 – Temps d'exécution d'advection-diffusion selon différentes tailles de problèmes, avec et sans perte de liens sur 10 processeurs.

que la précédente. Le gain obtenu sur ce réseau est plus important et plus régulier qu'auparavant. Dans un second temps, on remarque que l'application est très sensible au dynamisme du réseau : le temps d'exécution est entre 2 et 6 fois supérieur sur le réseau dynamique. Ceci confirme nos résultats précédents. Rappelons toutefois que le dynamisme est simulé et que le ralentissement qu'il engendre au niveau des communications est assez fort. Le dernier point qui apparaît dans ces résultats est celui concernant le gain apporté par l'équilibrage de charge sur le réseau dynamique. Le gain décroissant pour les problèmes de petite taille s'explique de la même manière que précédemment en tenant compte denotre nouvelle définition de la puissance disponible. Pour les autres tailles du problème, la différence de gain provient de la taille du réseau : le nombre de machines étant plus important, l'équilibrage du système est plus long et le gain apporté par l'équilibrage est donc relativement plus faible. Cependant, au-delà d'une taille 400x400 le gain est régulier et croît en fonction de la taille du problème, ceci est dû à la définition de la puissance disponible.

7.4 Conclusion

Ce chapitre présente la mise en pratique d'un algorithme d'équilibrage de charge dans une application concrète et sur un réseau dynamique. L'application utilisée est celle d'advection-diffusion que nous avons présentée dans le chapitre 6. Nous présentons les modifications apportées à cette application afin qu'elle se déroule de manière asynchrone. Nous nous plaçons dans un contexte asynchrone pour que la résolution du problème soit compatible avec le dynamisme du réseau. Nous donnons également dans ce chapitre un nouvel algorithme de détection de convergence qui respecte le caractère distribué et asynchrone de l'application. Bien que cet algorithme ait été développé dans un contexte asynchrone, il s'applique facilement dans celui

synchrone. Dans ce contexte d'exécution et pour l'application choisie, nous sommes obligés d'adapter l'équilibrage et de le réaliser selon l'erreur, c'est à dire selon le temps nécessaire à un processeur pour converger, et non pas selon le temps d'un pas de Newton. L'objectif est, dans la mesure du possible, de permettre à tous les processeurs de converger simultanément. Cette modification est réalisée uniquement dans le contexte asynchrone mais peut très bien être mise en place dans celui synchrone. Ceci n'a pas été réalisé puisque nous nous intéressons plus aux problèmes de l'équilibrage de charge sur réseaux dynamiques qu'à celui de la mise en place d'un algorithme d'équilibrage dans une application. En ce qui concerne l'exécution de l'application sur réseaux dynamiques, nous montrons que l'introduction d'un algorithme d'équilibrage de charge permet quasi toujours un gain de temps équivalent à celui obtenu sur un réseau statique. Bien que le gain de temps ne soit pas négligeable, il n'est pas exceptionnel. Ceci est dû à plusieurs points. L'algorithme d'équilibrage est générique, une méthode d'équilibrage étudiée spécifiquement pour l'application aurait donné de meilleurs résultats. Nous avons vu que nos définitions de la charge et de la puissance de calcul ne sont pas des plus adéquates, elles génèrent un blocage qui empêche l'équilibrage complet du système. Afin d'éviter se phénomène, différentes pistes peuvent être étudiées : par exemple, prendre le temps lorsque un processeur passe pour la première fois sous le seuil de convergence ou encore considérer l'erreur proprement dite après quelques itérations de Newton.

Conclusion

Au cours de cette thèse, nous nous sommes intéressés aux algorithmes itératifs d'équilibrage de charge sur réseaux dynamiques. Nous avons défini un réseau dynamique comme étant un réseau sur lequel certains liens de communication peuvent être perdus ou surchargés. Ces algorithmes itératifs ont été choisis en raison de leur caractère totalement distribué et de leur généricité. Ceci permet de les appliquer sur n'importe quel système nécessitant de l'équilibrage de charge. Ce mémoire de thèse apporte de nouvelles contributions dans différents domaines, nous les rappelons ci-dessous.

Dans une première partie, nous nous sommes consacrés aux algorithmes de premier ordre. Ensuite, nous en avons introduit un nouveau - la diffusion relaxée - qui accélère la convergence de l'algorithme de diffusion classique. Nous avons également présenté dans cette partie une méthode permettant d'adapter ces algorithmes aux réseaux hétérogènes. Après une présentation de ces algorithmes sur réseaux statiques, nous nous sommes intéressés à leur mise en place sur réseaux dynamiques. Pour chacun d'eux, nous donnons leur adaptation à ce type de réseaux. Le point le plus important de cette partie est la preuve de la convergence de ces algorithmes sur réseaux dynamiques. Cette convergence s'effectue sous certaines conditions cohérentes avec la réalité. Différentes simulations ont permis de mettre en évidence le comportement de ces algorithmes et les différentes étapes de leur exécution.

Dans un second temps nous avons étudié les algorithmes de second ordre. Comme pour ceux de premier ordre, nous avons commencé par les présenter sur réseaux statiques. Ces algorithmes ont l'avantage de converger vers l'équilibre nettement plus rapidement que ceux de premier ordre, mais ils ne tiennent pas compte de la positivité de la charge. A notre connaissance, il n'existe pas de travaux dans ce sens. Nous avons donc introduit une nouvelle contrainte sur la détermination du paramètre de second ordre afin que l'algorithme garantisse la positivité de la charge à tout instant. Pour les deux algorithmes de second ordre que nous avons étudiés, nous avons proposé leur adaptation aux réseaux dynamiques, et nous avons montré expérimentalement qu'ils convergent malgré le dynamisme du réseau. La

preuve théorique est nettement plus complexe que dans le cas d'algorithmes de premier ordre si bien qu'elle n'a, à l'heure actuelle, pas encore été finalisée. Suite à différentes simulations, nous avons montré que ces algorithmes sont très sensibles au dynamisme du réseau et que, dans le cas de topologies en lignes ou en anneaux, le comportement des algorithmes de second ordre est imprévisible dès que certains liens sont perdus. Dans certains cas, ils peuvent même converger moins rapidement que les algorithmes de premier ordre. Ceci s'explique par le fait que l'effet mémoire des algorithmes de second ordre est perturbé par le dynamisme.

Pour finir, nous avons présenté une mise en pratique d'un algorithme d'équilibrage de charge sur un problème concret. Ceci nous a permis de détailler les différentes étapes nécessaires au déploiement d'un algorithme d'équilibrage de charge. Le problème choisi consiste à résoudre une EDP (équation différentielle partielle) modélisant le problème de cinétique chimique d'advection diffusion. Dans cette partie, nous avons commencé par présenter l'application d'advection-diffusion ; ce qui nous a permis de détailler le choix et la mise en place de l'algorithme d'équilibrage de charge que nous avons utilisé. Nous avons expliqué, dans cette mise en place, la détermination de la charge induite par l'application et la puissance disponible sur chaque processeur coopératif. Des expérimentations ont été réalisées sur réseaux statiques et dynamiques. Dans le cas de réseaux dynamiques, nous avons justifié qu'une exécution synchrone de l'application n'est pas compatible avec le dynamisme du réseau. Une version asynchrone de l'application a donc été mise en place pour pallier ce problème. Le fait d'utiliser une version asynchrone du problème nous a obligé à redéfinir la notion de puissance disponible de chaque machine afin de prendre en compte ce nouveau mode d'exécution. Les résultats de ces expérimentations montrent que, pour l'application choisie, l'apport de l'équilibrage de charge est plus ou moins important selon la taille du problème, les caractéristiques du réseau et la définition de la puissance disponible sur chaque machine. De manière plus générale, les simulations et les expérimentations que nous avons réalisées montrent les avantages de tel ou tel type d'algorithme d'équilibrage selon la topologie du réseau et la difficulté à les mettre en place.

Suite à ce mémoire, différentes études pourraient élargir le travail de thèse déjà réalisé. En ce qui concerne les algorithmes de premier ordre, seule la détermination du paramètre de relaxation de la diffusion relaxée peut être légèrement améliorée. Dans notre présentation de cet algorithme, la détermination de ce paramètre nécessite une connaissance globale du réseau et le calcul des valeurs propres de la matrice de diffusion. Pour les topologies les plus classiques, une étude permettant de déterminer ce paramètre uniquement en fonction de la topologie et surtout sans calcul de valeurs propres, faciliterait la mise en place de cet algorithme dans un contexte réel d'exécution. Pour les algorithmes de second ordre, leur convergence sur réseaux dy-

namiques n'as pas été prouvée et ce point reste donc un problème ouvert. En ce qui concerne la mise en application des algorithmes d'équilibrage de charge que nous avons présentés, la notion de charge et celle de puissance disponible n'ont été définies que pour l'application d'advection diffusion. Il semble intéressant d'étudier ces deux notions afin qu'elles soient déterminées de manière optimale en fonction de la méthode de calcul ou du système distribué à équilibrer. D'un point de vue plus général, une estimation du coût de l'équilibrage de charge et du gain qu'il apporte en fonction du problème à équilibrer, permettrait d'évaluer l'apport de la mise en place de l'équilibrage dans cette application. Dans ce mémoire, nous avons étudié le comportement des algorithmes d'équilibrage de charge sur des réseaux classiques ; le même type d'études pourraient être mené sur des structures de réseaux plus compliquées, par exemple des réseaux ad-hoc ou "peer to peer". Sur ce type de réseaux, le dynamisme n'est pas seulement au niveau des liens de communication mais également un niveau des machines : certaines d'entre elles peuvent spontanément apparaître dans le réseaux ou définitivement disparaître. Ce type d'étude élargirait le champs d'application des algorithmes d'équilibrage de charge que nous avons présentés.

Bibliographie

[1] S. Agrawal, J. Dongarra, K. Seymour, and S. Vadhiyar. Netsolve : Past, present, and future - a look at a grid enabled server. In *Grid Computing : Making the Global Infrastructure a Reality*. Wiley Publishing, April 2003.

[2] W. Aiello, B. Awerbuch, B. Maggs, and S. Rao. Approximate load balancing on dynamic and asynchronous networks. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 125–136, San Diego, California, 1993.

[3] G. Antonoiu and P.K. Srimani. A self-stabilizing leader election algorithm for tree graphs. *Journal of Parallel and Distributed Computing*, 34(2) :227–232, 1 May 1996.

[4] H. Arndt. On finite dimension exchange algorithms. *Linear Algebra and its Applications*, 380 :73–93, 2004.

[5] O. Aumage and G. Mercier. MPICH/MadIII : a Cluster of Clusters Enabled MPI Implementation. In *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pages 26–35, Tokyo, May 2003. IEEE.

[6] J.M. Bahi, S. Contassot, and R. Couturier. On the interest of load-balancing asynchronous parallel iterative algorithms. Research Report LIFC, 2004.

[7] J.M. Bahi, S. Contassot-Vivier, and R. Couturier. Asynchronism for iterative algorithms in a global computing environment. In *int. conf. on High Performance Computing Systems and Applications HPCS'02*, pages 90–97, Moncton Canada, June 2002. IEEE computer society press.

[8] J.M. Bahi, S. Contassot-Vivier, R. Couturier, and F. Vernier. Asynchronisme et équilibrage de charge dans la grille de calcul. In *Ecole d'hiver GRID 2002, Aussois, France*, pages 365–373, December 2002.

[9] J.M. Bahi, S. Contassot-Vivier, R. Couturier, and F. Vernier. A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms. *TPDS, IEEE Computer Society*, 16(1) :4–13, January 2005.

[10] J.M. Bahi, R. Couturier, and F. Vernier. Load balancing on dynamic network. Technical Report RR-2002-1, LIFC, Université de Franche-Comté, september 2002.

[11] J.M. Bahi, R. Couturier, and F. Vernier. Accelerated diffusion algorithms on general dynamic networks. In *Proceedings of 5th International Conference, PPAM Czestochowa, Poland*, volume 3019 of *LNCS*, pages 77–82. PPAM, Springer-Verlag Heidelberg, 2003.

[12] J.M. Bahi, R. Couturier, and F. Vernier. Broken edges and dimension exchange algorithm on hypercube topology. In *Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 140–145. Euromicro, IEEE Computer Society Press, February 5–7, 2003.

[13] J.M. Bahi, R. Couturier, and F. Vernier. Synchronous distributed load balancing on dynamic networks. *JPDC*, acceptée.

[14] J.M. Bahi, R. Couturier, and P. Vuillemin. Asynchronous iterative algorithms for computational science on the grid : three case studies. In *Vecpar 04 To appear*, pages 0–0, Valencia Spain, June 2004. Springer Verlag LNCS XXXX.

[15] J.M. Bahi, R. Couturier, and P. Vuillemin. Solving nonlinear wave equations in the grid computing environment : an experimental study. *to appear Journal of Computational Acoustics (JCA)*, 2004.

[16] J.M. Bahi and J. Gaber. Load balancing on networks with dynamically changing topology. In *Europar 2001 conference, Lecture Notes on Computer Science*, pages 175–182, Manchester, UK, 2001.

[17] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Scheduling strategies for mixed data and task parallelism on heterogeneous clusters. *Parallel Processing Letters*, 13(2), 2003.

[18] A. Berman and R.J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Academic Press, SIAM, Philadelphia, third edition, 1979 edition, 1994.

[19] D.P. Bertsekas and J.N. Tsitsiklis. Convergence rate and termination of asynchronous iterative algorithms. In *Conference Proceedings, 1989 International Conference on Supercomputing*, pages 461–470, Crete, Greece, June 5–9, 1989. ACM SIGARCH.

[20] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation : Numerical Methods*. Englewood Cliffs NJ, Prentice-Hall, 1989.

[21] J.E. Boillat. Load balancing and poisson equation in a graph. *Concurrency : Practice and Experience.*, 2(4) :289–313, 1990.

[22] C. Bunks, J.P. Chancelier, F. Delebecque, C. Gomez, M. Goursat, R. Nikoukhah, and S. Steer. *Engineering and scientificcomputing with Scilab*. Birkhauser, 1999.

[23] A.S. Charão. *Multiprogrammation parallèle générique des méthodes de décomposition de domaine*. PhD thesis, Institut National Polytechnique de Grenoble, 2001.

[24] S. Contassot-Vivier, F. Lombard, J.M. Nicod, and L. Philippe. Evaluation of the diet hierarchical metacomputing architecture. *Parallel and Distributed Computing Practice : Special Issue on Algorithms, Systems and Tools for High Performance Computing on Heterogeneous Networks*, 2003.

[25] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7 :279–301, 1989.

[26] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25 :289–313, 1998.

[27] E.W. Dijkstra, W.H.J. Feijen, and A.J.M. vanGasteren. Derivation of a termination detection algorithm for distributed computation. *Information Processing Letters*, 16(5) :217–219, 1983.

[28] J. Dongarra, A. Lumsdaine, R. Pozo, and K. Remington. A sparse matrix library in c++ for high performance architectures. In *Proceedings of the Second Object Oriented Numerics Conference*, pages 214–218, 1994.

[29] M. El-Ruby, J. Kenevan, R. Carison, and K. Khalil. Leader election in distributed computing systems. In Naveed A. Sherwani, Elise de Doncker, and John A. Kapenga, editors, *Proceedings of Computing in the 90's*, volume 507, pages 350–356, Berlin, Germany, 1991. lncs, Springer.

[30] R. Elsässer, B. Monien, and R. Preis. Diffusive load balancing schemes on heterogeneous networks. In *Proceedings of the 12th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 30–38, Bar Harbor, Maine, July 9–12, 2000. SIGACT/SIGARCH and EATCS.

[31] R. Elsässer, B. Monien, and R. Preis. Diffusion schemes for load balancing on heterogeneous networks. *Theory of Computing Systems*, 35(3) :305–320, 2002.

[32] R. Elsässer, B. Monien, G. Rote, and S. Schamberger. Toward optimal diffusion matrices. In *16th International Parallel and Distributed Processing Symposium. IPDPS 2002, Proceedings.* IEEE Computer Society Press, May 2002.

[33] R. Elsässer, B. Monien, and S. Schamberger. Load balancing in dynamic networks. In *7th International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN 2004)*, pages 193–200. IEEE Computer Society, May 2004.

[34] S. Fiorini and R.J. Wilson. Edge-coloring of graphs. In *L.W. Beineke and R.J. Wilson, editors, Selected topics in graph theory.* Academic Press, 1978.

[35] C. Flament. Théorie des graphes et structure sociale. Paris, La Haye, Mouton-Gauthier-Villars, 1965.

[36] I. Foster and C. Kesselman. Globus : A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2) :115–128, Summer 1997.

[37] N. Francez. Distributed termination. *ACM Transactions on Programming Languages and Systems*, 2(1) :42–55, January 1980.

[38] M.A. Franklin and V. Govindan. A general matrix iterative model for dynamic load balancing. *Parallel Computing*, 22(7) :969–989, April 1996.

[39] J. Gaber. *Plongements et manipulations d'arbres dans les architectures distribuées.* PhD thesis, Université des sciences et technologies de Lille, 1998.

[40] J. Gaber and B. Toursel. Randomized load distribution of arbitrary trees in distributed networks. In *the 1998 ACM symposium on Applied Computing table of contents*, pages 564 – 568, Atlanta, Georgia, United States, 1998. ACM.

[41] J.E. Gehrke, C.G. Plaxton, and R. Rajaraman. Rapid convergence of a local load balancing algorithm for asynchronous rings. *Theoretical Computer Science*, 220(1) :247–265, 1999.

[42] B. Ghosh, F.T. Leighton, B.M. Maggs, S. Muthukrishnan, C.G. Plaxton, R. Rajaraman, A.W. Richa, R.E. Tarjan, and D.I. Zuckerman. Tight analyses of two local load balancing algorithms. *SIAM Journal on Computing*, 29(1) :29–64, 1999.

[43] B. Ghosh and S. Muthukrishnan. Dynamic load balancing in parallel and distributed networks by random matchings. In *Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, pages 226 – 235. ACM Press, 1994.

[44] B. Ghosh, S. Muthukrishnan, and M.H. Schultz. First and second order diffusive methods for rapid, coarse, distributed load balancing. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 72–81, Padua, Italy, June 24–26, 1996. SIGACT/SIGARCH. Extended abstract.

[45] G.H. Golub and Varga R.S. Chebyshev semi-iterative methods, successive over-relaxation iterative methods, and second order richardson iterative methods. I. *j-NUM-MATH*, 3 :147–156, 1961.

[46] H. Guyennet, B. Herrmann, L. Philippe, and F. Spies. A performance study of dynamic load balancing algorithms for multicomputers. In *Conference on Massively Parallel Computing Systems.* IEEE, 1994. May.

[47] H. Guyennet, B. Herrmann, L. Philippe, and F. Spies. Simulation of dynamic load balancing algorithms for multicomputers. In *High Performance Computing Conference (HPC'94)*, Singapore, September 1994.

[48] H. Guyennet, F. Spies, and M. Trehel. Modelling and simulation of dynamic load balancing using queuing theory. *Journal of Parallel Algorithms and Applications*, 5(0 1+2), 1994.

[49] A. Heirich. A scalable diffusion algorithm for dynamic mapping and load balancing on networks of arbitrary topology. *Int. J. Found. Comput. Sci.*, 8(3) :329–, 1997.

[50] S.H. Hosseini, B. Litow, M. Malkawi, J. McPherson, and K. Vairavan. Analysis of a graph coloring based distributed load balancing algorithm. *Journal of Parallel and Distributed Computing*, 10 :160–166, 1990.

[51] Y.F. Hu and R.J. Blake. An improved diffusion algorithm for dynamic load balancing. *Parallel Computing*, 25(4) :417 – 444, april 1999.

[52] G. Karagiorgos and N.M. Missirlis. Accelerated diffusion algorithms for dynamic load balancing. *Inf. Process. Lett*, 84(2) :61–67, 2002.

[53] O. Krone and M. Raaband B. Hirsbrunner. Load balancing for network based multi-threaded applications. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1497 of *LNCS*, pages 206–214. 5th European PVM/MPI Users' Group Meeting, September 7-9 1988.

[54] S.Y. Lee and C.H. Cho. Load balancing for minimizing execution time of a target job on a network of heterogeneous workstations. In L. Rudolph (Eds.) D.G. Feitelson, editor, *Job Scheduling Strategies for Parallel Processing*, volume 1911, page 174. Springer-Verlag Heidelberg, 2000.

[55] A. Legrand and Y. Robert. *Algorithmique Parallèle – Cours et exercices corrigés*. Dunod, 2003.

[56] B. Litow, S.H. Hosseini, K. Vairavan, and G.S. Wolffe. Performance characteristics of a load balancing algorithm. *Journal of Parallel and Distributed Computing*, 31 :159–165, 1995.

[57] R. Lling, B. Monien, and F. Ramme. A study of dynamic load balancing algorithms. In *Proceedings of the 3rd IEEE SPDP*, pages 686–689, 1991.

[58] U. Nagashima and H. Takagi. Ninf : a network-based information library for global world-wide computing infrastructure. In *High-Performance Computing and Networking, International Conference and Exhibition*, pages 491–502, april 1997.

[59] R. Namyst and J.F. Mehaut. PM^2 : Parallel multithreaded machine. A computing environment for distributed architectures. In *Parallel Computing : State-of-the-Art and Perspectives*, volume 11, pages 279–285, 1996.

[60] X.S. Qian and Q. Yang. Load balancing on generalized hypercube and mesh multiprocessors with lal. In *Proceedings of the 11th International Conference on Distributed Computing Systems*, pages 402–409, 1991.

[61] S.P. Rana. A distributed solution to the distributed termination problem. *Information Processing Letters*, 17 :43–46, July 1983.

[62] T. Rotaru and H.H. Nägeli. Dynamic load balancing by diffusion in heterogeneous systems. *Journal of Parallel and Distributed Computing*, 0 :In Press, 2004.

[63] S.A. Savari and D.P. Bertsekas. Finite termination of asynchronous iterative algorithms. *Parallel Computing*, 22 :39–56, 1996.

[64] IEEE Computer Society. Ieee standard for a high performance serial bus. Technical Report Std 1394-1995, IEEE, August 1996.

[65] M. Tréhel, C. Balayer, and A. Alloui. Modeling load balancing inside groups using queueing theory. In *PDCS'97 (10th International Conference on Parallel and Distributed Computing System)*, New Orleans, Louisiana, oct 1997.

[66] R.S. Varga. Matrix iterative analysis. *Prentice Hall Series in Automatic Computations, Englewood Cliffs : Prentice-Hall*, 1962.

[67] C.Z. Xu and F.C.M. Lau. Analysis of the generalized dimension exchange method for dynamic load balancing. *Journal of Parallel and Distributed Computing*, 16(4) :385–393, 1992.

[68] C.Z. Xu and F.C.M. Lau. Iterative dynamic load balancing in multicomputers. *Journal of the Operational Research Society"*, 45(7) :786–796, 1994.

[69] C.Z. Xu and F.C.M. Lau. Optimal parameters for load balancing with the diffusion method in mesh networks. *Parallel Processing Letters*, 4(1-2) :139–147, 1994.

[70] C.Z. Xu and F.C.M. Lau. The generalized dimension exchange method for load balancing in k-ary n cubes and variants. *J. Parallel Distrib. Comput.*, 1(24) :72–85, 1995.

[71] C.Z. Xu and F.C.M. Lau. Efficient termination detection for loosely synchronous applications in multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 7(5) :537–544, May 1996.

[72] C.Z. Xu, F.C.M. Lau, and R. Diekmann. Decentralized remapping of data parallel applications in distributed memory multiprocessors. *Concurrency : Practice and Experience*, 9(12) :1351–1376, 1997.

[73] C.Z. Xu, B. Monien, R. Lueling, and F. Lau. Nearest neighbor algorithms for load balancing in parallel computers. *Concurrency : Practice and Experience*, 7(7) :706–739, 1995.

[74] C.Z. Xu, B. Monien, R. Lüling, and F.C.M. Lau. An analytical comparison of nearest neighbor algorithms for load balancing in parallel computers. In *Proc. of the 9th International Parallel Processing Symposium*, pages 472–479. IEEE Computer Society Press, 1995.

Bibliographie Personnelle

[10] J.M. Bahi, R. Couturier, and F. Vernier. Load Balancing on Dynamic Network, Technical Report RR-2002-1, LIFC, Université de Franche-Comté september, 2002.

[8] J.M. Bahi, S. Contassot-Vivier, R. Couturier, and F. Vernier. Asynchronisme et équilibrage de charge dans la grille de calcul In *Ecole d'hiver GRID 2002, Aussois, France* pages 365–373. December 2002.

[12] J.M. Bahi, R. Couturier, and F. Vernier. Broken Edges and Dimension Exchange Algorithm on Hypercube Topology. In *Euromicro Conference on Parallel, Distributed and Network-based Processing,* pages 140-145. Euromicro, IEEE Computer Society, February 5–7, 2003.

[11] J.M. Bahi, R. Couturier, and F. Vernier. Accelerated Diffusion Algorithms on General Dynamic Networks. In *Proceedings of 5th International Conference, PPAM Czestochowa, Poland.* volume 3019 of *LNCS*, pages 77-82, Springer-Verlag Heidelberg, 2003.

[9] J.M. Bahi, S. Contassot-Vivier, R. Couturier, and F. Vernier. A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms, *TPDS, IEEE Computer Society*, (16)1 :4–13, January 2005.

[13] J.M. Bahi, R. Couturier, and F. Vernier. Synchronous Distributed Load Balancing on Dynamic Networks, *JPDC*, acceptée.

Résumé

Ce mémoire de thèse s'attache aux problèmes d'équilibrage de charge sur les réseaux dynamiques. Un réseau dynamique est un réseau sur lequel certains liens de communication peuvent être perdus ou surchargés. Dans un premier temps, nous introduisons un nouvel algorithme d'équilibrage de charge, puis nous présentons, l'adaptation aux réseaux dynamiques des algorithmes de premier ordre. Dans un second temps nous nous intéressons aux algorithmes de second ordre. Pour ces derniers, nous définissons une nouvelle contrainte qui garantit la positivité de la charge car la charge d'un système ne peut pas être négative. Suite à cette définition, nous donnons l'adaptation de ces algorithmes aux réseaux dynamiques. Nous illustrons le comportement des deux types d'algorithmes présentés - premier et second ordre - par différentes simulations qui mettent en évidence l'impact du dynamisme du réseau sur leur évolution. Ces simulations nous montrent que les algorithmes de second ordre sont nettement plus sensibles que ceux de premier ordre. Pour finir ce mémoire, une mise en pratique d'un algorithme d'équilibrage de charge sur un problème concret - la résolution d'une équation différentielle partielle - détaille les différentes étapes nécessaires au déploiement d'un algorithme d'équilibrage de charge.

Mots clefs : équilibrage de charge, réseaux dynamiques, diffusion, algorithmes itératifs, équation différentielle partielle.

Abstract

This thesis studies the problem of load balancing on dynamic networks. A dynamic network can be viewed as a network in which some edges may fail during the execution of an algorithm. The first part introduces a new load balancing algorithm, and it describes the adaptation of first order algorithms to dynamic networks. The second part deals with second order algorithms. We introduce a new constrain for these algorithms, so as to ensure that the load stay positive for the load of a system cannot be negative. Then we give the adaptation of these algorithms to dynamic networks. We illustrate the behavior of the first and second order algorithms with some simulations that bring the impact of the dynamism of networks on load balancing algorithms to the fore. These simulations show that the second order algorithms are more reactive to the dynamism than the first order algorithms. Finally, we have implemented a load balancing algorithm on a real application that solves a partial differential equation. With this experimentation, we describe the different steps of the deployment of a load balancing algorithm.

Key words : load balancing, dynamic networks, diffusion, iterative algorithms, partial differential equation.