Equilibrage de charge pour les grilles de calcul : classe des tâches dépendantes et indépendantes.

Meriem Meddeber[1] et Belabbas Yagoubi[2]

[1]Université de Mascara, Faculté des sciences, Département des sciences et technologies, 29000 Mascara, Algérie
m.meddeber@yahoo.fr
[2]Université d'Oran, Faculté des sciences, Département d'informatique, 31000 Oran, Algérie
byagoubi@yahoo.fr

Résumé. L'équilibrage de charge dans les grilles de calcul représente un défi pour les chercheurs et les développeurs de ces types de systèmes. Ce défi est en rapport avec les particularités de ces infrastructures, à savoir, l'*hétérogénéité*, la *dynamicité* et le *passage à l'échelle*. Ce challenge devient beaucoup plus complexe quand il s'agit d'équilibrer la charge d'un système traitant des tâches avec précédences. Dans ce papier, nous proposons un modèle hybride sur lequel nous développons une stratégie d'équilibrage à deux niveaux : **Local**, dans le but d'éviter le recours au réseau de communication à large échelle, et **Global**, pour une régulation de la charge de toute la grille. La stratégie proposée vise à réduire le temps de réponse et le coût de transfert des tâches soumises au système. Nous nous intéressons dans un premier temps à la classe des tâches indépendantes (sans relations de précédences), ensuite nous améliorerons la stratégie pour pendre en compte la classe des tâches dépendantes.

Mots-clés: Grilles de calcul, Équilibrage de charge, Tâches indépendantes, Tâches dépendantes, Stratégie distribuée.

1 Introduction

Les grilles de calcul [1] sont des architectures en plein développement. Elles consistent en un réseau d'ordinateurs faiblement couplés et ont pour but d'offrir une très grande puissance de calcul à leurs utilisateurs de la façon la plus transparente possible. Ces ordinateurs peuvent être des supercalculateurs, des clusters ou des stations de travail ordinaires. Ils sont reliés par un réseau à très grande échelle, le plus souvent Internet. De ce fait, une grille est un ensemble de logiciels permettant de répartir et d'exécuter des applications extrêmement coûteuses en calcul sur un parc de machines hétérogènes et placées sous des autorités administratives différentes.

Ainsi, un problème crucial à résoudre est la gestion et le partage des ressources multiples dans le but d'augmenter les performances du système. Des tâches sont générées par un ensemble d'utilisateurs, et attendent que des ressources soient libres pour pouvoir être traitées. Il semble alors naturel d'avoir les situations où certaines

ressources soient surchargés tandis que d'autres soient sous chargées ou complètement libres. Grâce au réseau de communication, on peut envisager de pallier de telles situations en transférant une partie de la charge d'une ressource surchargée vers une ressource sous-chargée. C'est à ce problème que l'équilibrage de charge apporte une solution à travers un ensemble de technique [2] permettant une distribution équitable de la charge de travail sur les ressources disponibles d'un système. L'équilibrage de la charge de travail sur les diverses ressources disponibles d'une grille de calcul s'avère être un véritable défi dont les objectifs peuvent être de trois types: (i) Minimisation du temps moyen de réponse des applications, (ii)Maximisation du degré d'occupation des ressources, et (iii) réduction des coûts de communication.

Dans une grille de calcul, une application est toujours décomposée en plusieurs ensembles de tâches parmi lesquelles il existe des relations de précédences. De telles relations deviennent un nouveau défi pour le problème d'équilibrage de charge.
Actuellement, la plupart des travaux de recherche se sont concentrés sur l'équilibrage de charge des tâches indépendantes (sans relations de précédence) [3] [4]. Bien que de tels efforts de recherches puissent résoudre le problème de l'hétérogénéité et de la dynamicité des ressources de la grille, ils ne sont pas adaptables au problème d'équilibrage des tâches dépendantes.
Dans le domaine des tâches avec contraintes de précédences, plusieurs travaux ont été dédiés au problème d'ordonnancement des tâches [5] [6]. Bien qu'ils visent à minimiser le temps de réponses des tâches soumises à la grille, ces travaux de recherches se révèlent utiles uniquement si on peut injecter suffisamment de connaissances dans le processus d'ordonnancement et d'application. En d'autres termes, si les caractéristiques de la plate-forme cible (vitesses des processeurs et capacités des liens) et de l'application cible (coût des calculs et des communications associés à chaque tâche) sont connues avec suffisamment de précision, alors un excellent niveau de performance peut être atteint par le biais de stratégies statiques[7]. Toutefois, nous nous intéressons à des systèmes fortement dynamique et très hétérogènes du point de vue plate-forme et applications, ces caractéristiques compliquent toute estimation préalable de cette connaissance, ce qui rend ces travaux de recherche inapplicable à ce type d'infrastructure.

Dans cet article nous proposons un modèle hybride à deux niveaux (sous forme d'une forêt d'arbres) pour résoudre le problème d'équilibrage de charge dans les grilles de calcul. Sur la base de ce modèle, nous définissons dans la troisième section une stratégie d'équilibrage dédiée aux tâches indépendantes. La quatrième section de ce papier vise à améliorer la stratégie proposée de façon a l'adapter à la classe des tâches dépendantes. Par ce travail, nous nous intéressons à : (i) La réduction du temps de réponse global des tâches soumises à la grille, (ii) La réduction des coûts de communication induits par le transfert des tâches.

2 Model proposé

2.1 Présentation

D'un point de vue topologique, une grille de calcul (Fig. 1) est composée d'un ensemble de G clusters connectés à travers un réseau global WAN. Chaque cluster est composé d'un ensemble d'éléments de calcul (EC) qui communiquent à travers un réseau local LAN. L'ensemble des ressources de calcul et moyens de communications peuvent être hétérogènes sur le plan des architectures, des systèmes d'exploitation et des réseaux de communications.

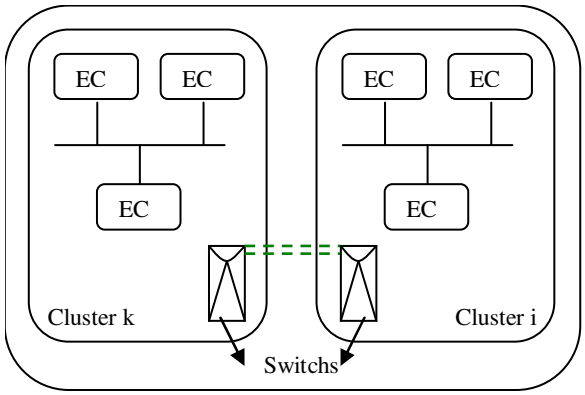

Fig. 1. Exemple de topologie d'une grille.

A chaque cluster nous associons un arbre à deux niveaux (Fig. 2) :

- Le niveau 0 représente le gestionnaire du cluster dont le rôle est de gérer la charge local de ce dernier.
- Le second niveau, correspond aux éléments de calcul qui constituent le cluster.

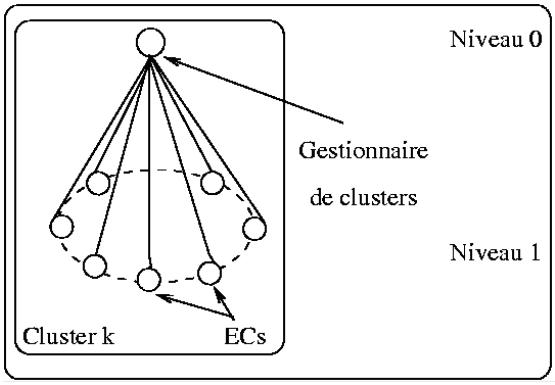

Fig. 2. Modèle générique de représentation d'un cluster.

Les arbres à deux niveaux associés aux divers clusters sont agrégés pour former une forêt d'arbres, représentant la grille toute entière (Fig.3) [8].

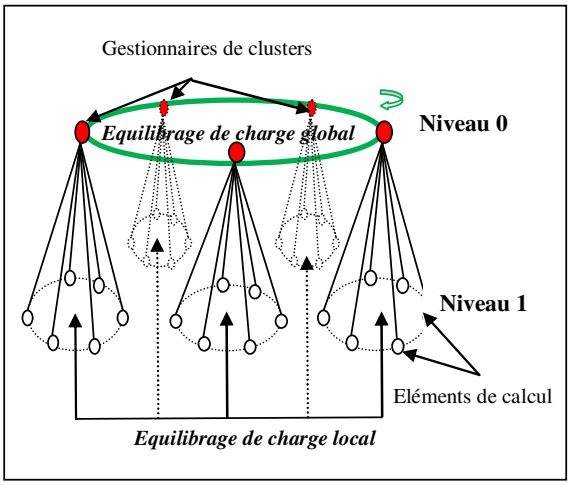

Fig. 3. Modèle générique de représentation d'une grille.

A chaque niveau nous associons une fonction :

- Niveau 0 : Ce niveau, contient G gestionnaires de clusters, chaque nœud à deux fonctions principales :
  (i) gère la charge locale de son cluster associé ;
  (ii) Participe à l'équilibrage de la charge globale de la grille.

- Niveau 1 : À ce niveau, nous trouvons les éléments de calcul de la grille connectés à leurs clusters respectifs. Chaque élément de calcul de ce niveau a pour rôle de :
  (i) Collecter son information de charge ;
  (ii) Envoyer cette information au gestionnaire de cluster associé ;
  (iii) Exécuter les décisions d'équilibrage prises par le gestionnaire.

2.2 Caractéristiques

Le modèle d'équilibrage de charge proposé est caractérisé par:

- La structure du modèle facilite les flux d'informations à travers les nœuds de l'arbre. En Effet nous distinguons trois types flux:
  (i) ***Flux montant*** : Ce flux concerne la circulation des informations de charge du niveau 1 vers le niveau 0. Grâce â ce flux, le gestionnaire du cluster pourra avoir une vue globale de la charge du cluster.
  (ii) ***Flux horizontal*** : Il concerne les informations nécessaires à l'exécution des opérations d'équilibrage de charge. Ce flux se trouve aux deux niveaux, il permet d'équilibrer la charge interne du cluster au niveau 1 et d'équilibrer la

charge globale de la grille au niveau 0 du modèle. En plus, ce flux véhicule les informations de charge entre les gestionnaires des clusters du niveau 0.
(iii) ***Flux descendant*** : Ce flux permet de véhiculer les décisions d'équilibrage prises par les gestionnaires de cluster aux niveaux 0 vers les nœuds du niveau 1.

- La modélisation d'une grille en arbre s'effectue par une transformation univoque. A chaque grille correspond un et un seul arbre de représentation et ce quelque soit la complexité topologique de la grille,
- Le modèle proposé supporte en termes de ressources:
  (i) ***La dynamicité*** : puisque les connexions/déconnexion des utilisateurs sont de simple opération d'ajout/suppression des feuilles de l'arbre,
  (ii) ***L'hétérogénéité*** : car nous n'avons imposé aucune contrainte sur les nœuds de l'arbre,
- Le modèle proposé est composé uniquement de deux niveaux, quelque soit la complexité topologique d'une grille.

3 Stratégie d'équilibrage de charge des tâches indépendantes

3.1 Principe

La stratégie d'équilibrage de charge proposée est dédiée à la classe des tâches indépendantes et composée des trois étapes suivantes [9]:

Étape 1 : Estimation de la charge courante du cluster (resp. de la grille)
Sachant que N représente le nombre d'éléments de calcul du cluster (resp. le nombre de cluster de la grille), Chaque gestionnaire du cluster exécute les étapes suivantes :

- Estime la charge courante (*LOD*) du cluster (resp. de la grille) en se basant sur les informations de charge envoyées par ses éléments de calcul (resp. par les gestionnaires de clusters de la grille).

$$LOD = \sum_{i=1}^{N_k} LOD_i$$

- Estime la capacité (*SAT*) et la vitesse *(SPD*) du cluster (resp. de la grille).

$$SPD = \sum_{i=1}^{N_k} SPD_i \qquad SAT = \sum_{i=1}^{N_k} SAT_i$$

,

- Estime le temps d'exécution moyen du cluster (resp. de la grille).

$$TEX = \frac{LOD}{SPD}$$

- Calcul l'écart type (σ) sur les charges de travail de ses éléments de calcul (resp. de l'ensemble des clusters de la grille) dans le but de mesurer l'étendue des variations de charge entre le cluster et ses nœuds (resp. entre les différents clusters).

$$\sigma = \frac{1}{N} \sqrt{\sum_{i=1}^{N_k} (TEX_I - TEX)^2}$$

- Envoie son information de charge de travail à l'ensemble des clusters de la grille.

Etape 2: prise de décision
Dans cette étape, le gestionnaire décide s'il est nécessaire d'effectuer une opération d'équilibrage de charge ou non. Pour cela, il exécute les deux actions suivantes:

a- Définition de l'état de déséquilibre/saturation/surcharge du cluster

Si nous considérons que l'écart type mesure la variation moyenne entre le temps d'exécution des éléments de calcul et celui de leur gestionnaire associé, nous pouvons dire qu'un cluster est en état d'équilibre lorsque cet écart est relativement faible. En effet, cela implique que les temps d'exécution de chaque élément de calcul convergent vers le temps d'exécution de son cluster. Ensuite, nous définissons les états d'équilibre et da saturation.

État d'équilibre: En pratique, il s'agit de définir un seuil d'équilibre, noté **ε**, à partir duquel nous pouvons dire que l'écart type tend vers zéro et donc le cluster est en état d'équilibre. Ainsi nous pouvons écrire :
Si ($\sigma \leq \varepsilon$) **Alors** le cluster est équilibré **Sinon** il est déséquilibré.

État de saturation: Un cluster peut être déséquilibré tout en étant saturé. Dans ce cas particulier, il n'est pas utile d'entamer un équilibrage local, puisque le cluster restera surchargé. Pour mesurer la saturation d'un cluster, nous définissons un autre seuil, noté **δ**, que nous appellerons seuil de saturation. Lorsque la charge courante d'un cluster avoisine sa capacité maximale, il est évident qu'il ne sert à rien de l'équilibrer puisque tous ses éléments de calcul sont saturés.

État de surcharge: Si le cluster est en état de saturation, il déterminera son état par rapport aux clusters de la grille. Soit $\boldsymbol{TEX_G}$ le temps d'exécution de la grille $\boldsymbol{TEX_i}$ le temps d'exécution du cluster. Nous définissons un intervalle de confiance basé sur l'écart type: $[TEX_G - \sigma; TEX_G + \sigma]$. Ainsi nous pouvons écrire :
Si ($TEX_i > (TEX_G + \sigma)$) **Alors** le cluster est surchargé.

b- Partitionnement

- Pour un cas de déséquilibre, nous déterminons les éléments de calcul surchargés (sources) ainsi que les éléments de calcul sous-chargés (receveurs), Selon l'écart entre le temps d'exécution de chaque élément de calcul et celui de son cluster $\boldsymbol{TEX_c}$.
Ainsi, tout élément est considéré comme:

- Sous-chargé si son temps d'exécution est inférieur à $\boldsymbol{TEXc - \sigma}$;
- Surchargé si son temps d'exécution est supérieur à $\boldsymbol{TEXc + \sigma}$;

- Pour un cas de surcharge du cluster, nous déterminons les clusters sous chargés, selon l'écart entre le temps d'exécution de chaque cluster et celui de la grille estimer par le cluster surchargé. Ainsi, Un cluster est sous-chargé si son temps d'exécution est inférieur à $\boldsymbol{TEX_G - \sigma}$.

Etape 3: Transfert de tâches.
Afin de transférer des tâches à partir des éléments de calculs (resp. clusters) surchargés vers les éléments de calculs (resp. clusters) sous-chargés, nous proposons l'heuristique suivant :

a- Calculer la disponibilité, en termes de capacité de calcul, qui correspond à la charge totale offerte par les éléments de calcul (resp. clusters) receveurs.

$$\text{Offre} = \sum_{\text{Nik} \in \text{GER}} \frac{LOD_k \cdot \text{SPD}_{ik}}{\text{SPD}_k} - \text{LOD}_{IK}$$

b- Calculer la demande, i.e., la charge totale requise par l'ensemble des éléments de calcul (resp. cluster) sources.

$$\text{Demande} = \sum_{\text{Nik} \in \text{GES}} \text{LOD}_{IK} - \frac{LOD_k \cdot \text{SPD}_{ik}}{\text{SPD}_k}$$

c- Si l'offre est très inférieure à la demande (l'offre n'est pas en mesure de satisfaire suffisamment la demande), il n'est pas recommandé d'entamer un équilibrage local (resp équilibrage global).
Nous introduisons un troisième seuil, appelé seuil d'espérance et dénoté **ρ**, afin de mesurer l'écart relatif entre l'offre et la demande. Nous pouvons écrire l'expression suivante:
Si (Offre /Demande >ρ) **Alors** Effectuez un équilibrage local (resp. global) **Sinon** Effectuer un équilibrage global (resp. ne rien faire).

d- Effectuer un transfert de charge en tenant compte des coûts de communication dans le cas particulier d'un équilibrage global.

3.2 Résultats expérimentaux

Pour expérimenter la stratégie d'équilibrage proposée, nous avons utilisé un PC Pentium IV de 1.7GHz, doté d'une mémoire de 512 Mo et fonctionnant sous Linux.
Les expérimentations ont étaient faites a l'aide du simulateur GridSim [10].
Pour obtenir des résultats fiables, nous avons réitéré les mêmes expériences plus de cinq fois.
Les meilleurs résultats ont été obtenus pour $\varepsilon = 1500$,$\rho = 0.75$, et $\delta = 0.8$. A l'aide de ces expériences faites avant (sans exécution de l'algorithme d'équilibrage) et après (en exécutant l'algorithme d'équilibrage).
Nous considérons différents nombre de clusters comportant 30 éléments de calcul chacun et nous varions le nombre de tâches de 4000 à 1000.
Fig.4 illustre la variation du temps de réponse, obtenue par notre stratégie d'équilibrage de charge.

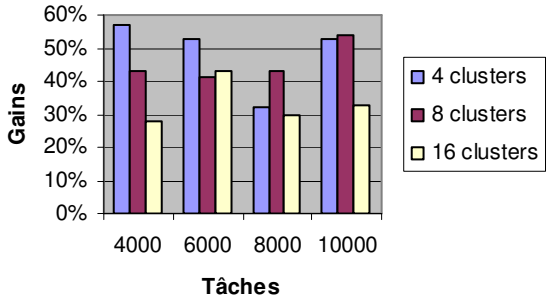

Fig. 4. Résultats Expérimentaux.

A travers ces résultats, nous constatons que :

• Tous les gains sont supérieurs à 22%. Etant donnée la nature des applications traitées par les grilles, il nous semble que ce gain est important et pourra, éventuellement, être amélioré.

• Les meilleurs gains ont été obtenus pour un nombre de clusters compris entre 2 et 8 et pour un nombre de tâches supérieur à 6000 (grille est dans un état stable).

• Les gains les plus faibles ont été obtenus lorsque le nombre de clusters a été fixé à 16. On peut justifier cela par l'instabilité de l'état de la grille (la plupart des nœuds sont sous chargés ou complètement libres).

4 Stratégie d'équilibrage de charge des tâches dépendantes

4.1 Relation entre tâches: contraintes de précédence

Les dépendances de tâches permettent de déterminer le moment où une tâche peut débuter. En fait, une tâche peut être associée à un ou plusieurs prédécesseurs qui doivent être complétés avant de pouvoir débuter la tâche en question. De plus, un temps d'attente peut être défini entre les prédécesseurs et les successeurs correspondant au temps de transfert des paramètres requis par les successeurs.

4.1.1 Définition

On appelle une contrainte de précédence entre la tâche t_i et la tâche t_j ou t_i précède t_j si t_j doit attendre la fin d'exécution de t_i pour commencer sa propre exécution [11].

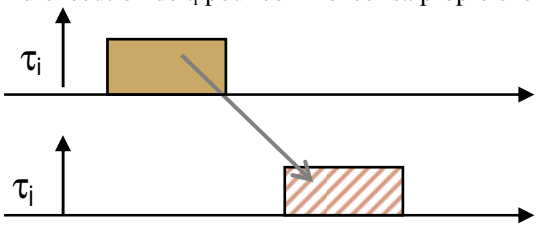

Fig. 5. Deux tâches liées par une contrainte de précédence

4.1.2 Graphes de précédence

Une application est généralement représentée par un graphe de précédence. Dans ce graphe, les sommets sont l'ensemble des tâches soumises au système et les arcs, les contraintes de précédence. Chaque tâche est caractérisée au minimum par son temps d'exécution et chaque arc par un coût de communication représentant la quantité d'informations à échanger entre les tâches. La largeur du graphe représente le nombre maximum de processus en parallèle et la hauteur de l'arbre représente le temps maximum d'exécution.

Fig. 6, présente un exemple de deux applications comportant chacune trois tâches, les relations de précédences qui existent entre les tâches des deux applications sont de type émission/ réception de message.

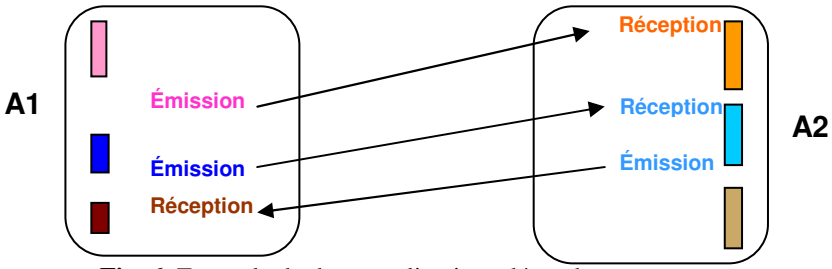

Fig. 6. Exemple de deux applications dépendantes.

Le graphe de précédence associé est schématisé sur la Fig. 7. La tâche T1 s'exécutera en premier puisqu'elle n'a pas de contraintes à satisfaire, et les tâches T5 et T6 s'exécuteront en derniers.

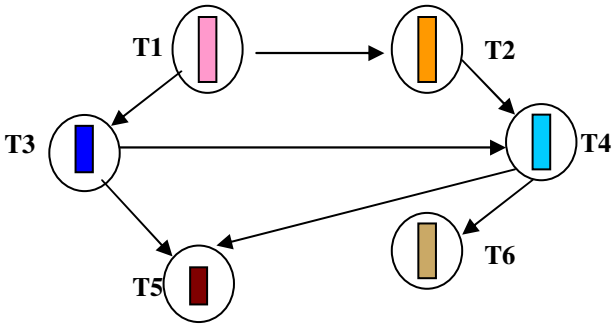

Fig. 7. Graphe de précédence associé.

4.1.3 Méta tâches

Un graphe de précédence peut être divisé en plusieurs sous graphes, comportant chacun un certain nombre de tâches dépendantes appelés méta-tâches (Fig. 8). Notons qu'il n'existe pas de relations de précédence entre les méta-tâches.

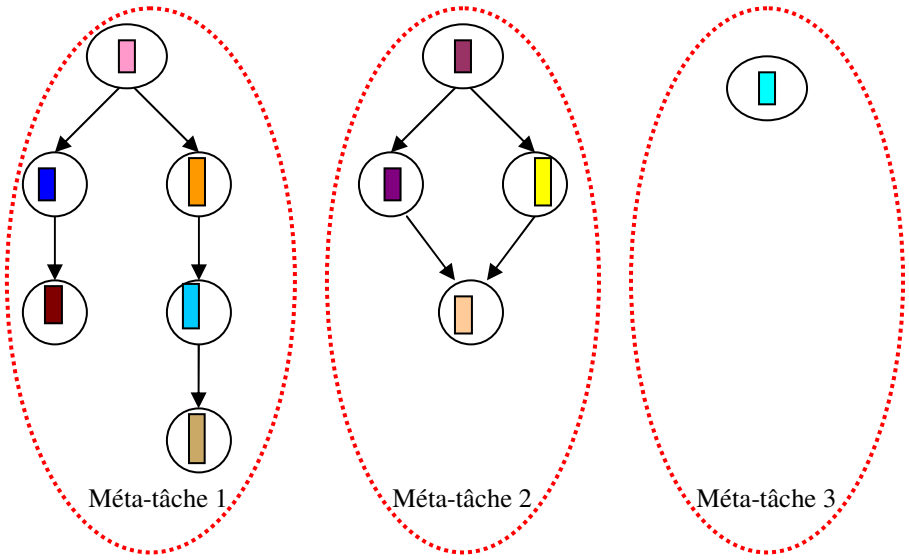

Fig. 8. Exemple de trois méta-tâches.

4.2 Principe

Cette partie de l'article vise à améliorer la stratégie proposée dans la section trois, dans le but de l'adapter à la classe des tâches dépendantes.
Pour cela, nous apercevons chaque méta-tâche comme une seule tâche indépendante des autres.
Ainsi nous modifions les étapes 1 et 3 comme suit :

Étape 1 : Estimation de la charge courante du cluster (resp. de la grille)

1- Chaque élément de calcul (resp. gestionnaire de cluster) calcul sa charge courante en prenant en considération uniquement les tâches prête à être exécutée (avec 0 contraintes à satisfaire).

2- Chaque gestionnaire du cluster estime :

- La charge *LOD*, la vitesse *SPD* et la capacité *SAT* du cluster (resp. grille).
- Le temps d'exécution du cluster :

$$Tex = \frac{LOD}{SPD} + \sum_{i=1}^{P} Tcom_i$$

Avec :
Tcom : temps de communication des informations transmises par les tâches prête à être exécutées et celle qui dépendent d'elles.
P : Nombre de tâches prête à être exécutées.

- Envoie son information de charge de travail à l'ensemble des clusters de la grille.

Etape 3: Transfert de tâches.

3- Calculer l'offre

4- Calculer la demande

5- **Si** (Offre /Demande >ρ) **Alors** Effectuez un équilibrage local (resp. global) **Sinon** Effectuer un équilibrage global (resp. ne rien faire).

6- Effectuer un transfert de charge en tenant compte des coûts de communication dans le cas particulier d'un équilibrage global.
Le transfert ce fait par méta-tâches, pour ne pas augmenter le temps de communication **Tcom**.

5 Conclusion

Cet article propose un modèle et une stratégie d'équilibrage de charge pour résoudre le problème d'équilibrage de charge dans les grilles de calcul. Nous avons, tout d'abord conçu un modèle représentant une grille de calcul, qui prend en compte les caractéristique d'une grille de calcul, à savoir l'hétérogénéité et la dynamicité des ressources et des réseaux de communications, ainsi que le passage à l'échelle.

Nous avons ensuite développé sur ce modèle une stratégie d'équilibrage de charge qui vise dans un premier temps les tâches indépendante ou sans contraintes de précédences. Les résultats obtenus par l'implémentation de la stratégie sous le simulateur GridSim sont pour le moment très satisfaisant.

Nous avons ensuite apporté quelques améliorations à la stratégie présentée afin de supporter les tâches avec contraintes de précédences qui présentent un réel chalenge pour l'équilibrage de charge dans les grilles de calcul.

Comme perspectives, nous visons à expérimenter notre stratégie sous le simulateur GridSim et sur un simulateur développé selon nos propores besoin. Nous nous intéressons a distribué le modèle pour éviter le recours a des procédures de tolérance aux pannes et de goulot d'étranglement causés par les gestionnaires de clusters. Enfin et une fois la stratégie maitrisée nous voulons l'intégrer au middleware GLOBUS[12].

Références

1. Smith, I. Foster and C Kesselman: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kauffman, San Francisco (1999)
2. F. Dong and G. Akl : Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Technical Report No. 2006-504, (2006)
3. H. Johansson, J. Steensland: A performance characterization of load balancing algorithms for parallel SAMR applications. Technical Report 2006-047, Uppsala University (2006)
4. H. Shan, L. Oliker, R. Biswas, W. Smith: Scheduling in heterogeneous grid environments: The effects of data migration, In Proc. of ADCOM2004, India, December (2004)
5. T. Chen1, B. Zhang2, and X. Hao2: A Dependent Tasks Scheduling Model in Grid. 10th Asia-PacificWeb Conference, Shenyang, China, April 26-28 (2008)
6. G. Malewicz: Scheduling Dags under Uncertainty. ACM Symposium on Parallel Algorithms and Architectures, Las Vegas Nevada USA, pp 66--75 (2005)
7. A. Legrand, H. Renard, Y. Robert, F. Vivien: Placement et équilibrage de charge pour calculs itératifs sur grappes hétérogènes. SympAAA'2003, France, (2003)
8. B. Yagoubi, M. Meddeber: A load balancing model for grid environment. 22nd IEEE International Symposium on Computer and Information Sciences, Ankara, Turkey, (2007)
9. B. Yagoubi: Modèle d'équilibrage de charge pour les grilles de calcul, Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées (ARIMA), Volume 7, pp: 1--19, ISSN: 1638-5713 (2007)
10. R. Buyya and M. Murshed: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and scheduling for Grid Computing, Journal of Concurrency and Computation: Practice and Experience, pp 1175--1220 (2002)
11. F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri; Ordonnancement temps réel; Edition kermes Science, Janvier (2000)
12. I. Foster. Globus toolkit version 4: Software for service oriented systems. In IFIP: International Conference on Network and Parallel Computing, pages 2-13, China, (2005)