Régulation de charge dans les systèmes distribués : architecture et algorithmes

Frédéric Wautelet

Faculté Notre-Dame de La Paix, Namur, Belgique,
fwautele@info.fundp.ac.be

Résumé. L'importance des algorithmes de répartition de charge dans les systèmes distribués n'est plus à démontrer. En effet, le choix d'un type d'algorithme peut influencer fortement les performances du système. De plus, un algorithme de répartition de charge peut être adapté pour un type particulier de problème mais peut ne pas l'être pour un autre. Cet article propose une vue d'ensemble de l'architecture générale des algorithmes de régulation de charge et les plus courants d'entre eux.

1 Définitions

1.1 Système distribué

Selon Tanenbaum [13], une définition d'un système distribué au sens large serait la suivante :

> "*Un système distribué* est un ensemble d'ordinateurs indépendants qui apparaît pour ses utilisateurs comme un seul système cohérent."

Plus précisement, on peut voir un système distribué comme un ensemble d'*unités de calcul* (appelé "nœud") possédant les caractéristiques suivantes [14] :

- Chaque nœud possède son propre espace d'adressage ;
- Le nombre de nœuds dans le système distribué est arbitraire ;
- Le système est capable de traiter un nombre quelconque de processus ;
- La communication entre les différents nœuds se fait au moyen de messages. Il convient, dès lors, de tenir compte du delai de communication entre les différents nœuds ;
- Les interactions entre les différents processus se font de manière coopérative, et non sur base du modèle maître/esclave ;
- En cas de panne d'une (ou plusieurs) unité(s) de calcul, le système doit pouvoir se reconfigurer (tolérance au panne).

Remarquons que tous les auteurs n'adhèrent pas à l'ensemble de ces caractéristiques et plus particulièrement aux deux dernières.

1.2 Régulation de charge

On peut définir la régulation de charge comme [1] :

"un système qui permet d'améliorer les performances d'un système distribué en répartissant la charge parmi un ensemble de nœuds qui coopèrent entre eux."

Dans cet article, on parlera indifféremment de *régulation* ou de *distribution de charge*. La régulation de charge comprend deux types d'algorithme selon que cette régulation est réalisée lors de la compilation ou à l'exécution du processus. Dans le premier cas, on parlera d'*ordonnancement déterministe* (*scheduling*) tandis que dans le deuxième cas, on parlera de *répartition de charge* (*load balancing*).

2 Classification des algorithmes de régulation de charge

Il n'est ni aisé ni souhaitable d'établir une classification rigide des algorithmes de régulation de charge. Néanmoins, Jie Wu [14] donne cinq critères permettant de différentier ces algorithmes :

- La structure des processus
- La méthode de communication entre les processus
- La connaissance des délais de propagation
- La tolérance aux pannes
- Le type d'application

Voyons plus en détails ces critères :

2.1 Structure des processus

Un algorithme distribué est composé d'un ensemble de processus. Une classification des processus peut être réalisée sur base du degré de symétrie :

- *Asymétrique* : chaque processus exécute un code différent (par exemple, un algorithme client/serveur) ;
- *Symétrie textuelle* et *symétrie forte* : le code de chaque processus est identique mais les processus peuvent se comporter de façons différentes en fonction des messages reçus ;
- *Symétrie totale* : le code de chaque processus est strictement identique et chaque processus se comporte de la même façon.

2.2 Méthode de communication entre les processus

La coopération des processus entre les différents nœuds implique deux types d'interaction : la *communication* et la *synchronisation*. Il y a deux moyens complémentaires d'interaction dans les systèmes distribués :

- L'*échange de messages* entre processus ;
- Les *données partagées* par les différents processus lorsque le système ne dispose pas de mémoire partagée.

L'*échange de messages* peut se faire de plusieurs façons. Les modèles les plus souvents utilisés sont les suivants :

- Le *point-à-point synchrone* dans lequel l'émetteur est bloqué tant que le destinataire n'a pas acquitté le message ;
- Le *point-à-point asynchrone* dans lequel l'émetteur n'attend pas l'acquis du destinataire ;
- Le *rendezvous* qui est basé sur le principe du *publish/subscribe*. Un processus qui désire envoyer un message ne spécifie pas le destinataire réel mais donne au système de communication un message avec un sujet donné. C'est ce système de communication qui transmettra ce message sur le réseau. Les receveurs, quant à eux, donnent la liste des sujets qui les intéressent au système de communication. Celui-ci s'arrangera pour ne fournir aux récepteurs que les messages dont le sujet intéresse les récepteurs ;
- L'*appel de procédure à distance* (RPC) ou l'*invocation d'objet distant* (Java RMI) ;
- Les *messages one-to-many* utilisant la transmission broadcast ou multicast.

Le *partage des données* peut être réalisé :

- avec des structures de données distribuées ;
- avec des variables logiques distribuées.

Cet aspect ne sera pas développé dans cet article.

2.3 La connaissance des délais de propagation

Dans un système distribué, les délais de propagation sont non négligeables et la connaissance de ces délais est très importante. On peut établir un classement des systèmes distribués sur base de ce critère :

- *Type I* : le délai de communication entre chaque paire de processus est connu exactement. Tutefois ce délai peut différer entre chaque paire ;
- *Type II* : le délai de communication est variable et n'est pas connu avec précision ;
- *Type III* : le délai de communication est variable mais connu *a posteriori* avec grande précision ;
- *Type IV* : le délai de communication est variable mais on connaît une borne supérieure de ce délai.

2.4 La tolérance aux pannes

On peut distinguer quatre types de pannes :
- La panne matérielle d'un nœud ;
- L'erreur dans un logiciel ;
- Les problèmes de communication ;
- Les problèmes de timing.

La tolérance aux pannes est basée sur la redondance, qui peut prendre les formes suivantes :
- Redondance matérielle ;
- Redondance logicielle ;
- Redondance d'information ;
- Redondance temporelle[1].

2.5 Le type d'application

Il est relativement évident que, pour un problème donné, les performances peuvent dépendre fortement de l'algorithme de régulation de charge choisi.

3 Algorithmes de distribution de charge statique

La distribution de charge peut se faire de deux façons :
- Soit de manière statique, c'est-à-dire que l'allocation des processus aux nœuds est déterminé à la compilation sur base d'une connaissance *a priori* du système. On parlera alors d'*ordonnancement déterministe* (*scheduling*) ;
- Soit de façon dynamique, c'est-à-dire que la répartition de charge se fait uniquement lors de l'exécution. On parlera alors de *régulation de charge* (*load balancing*). Dans cette section, on étudira les algorithmes d'*ordonnancement déterministe* tandis que les algorithmes de *régulation de charge* seront développés dans la section suivante.

Dans les algorithmes statiques, on peut distinguer deux classes : l'*ordonnancement optimal* et l'*ordonnancement sub-optimal*. L'ordonnancement sera dit optimal si une allocation optimale des processus sur les nœuds peut être faite sur base de critères tel que le temps d'exécution. En général, ces problèmes d'ordonnancement sont NP-complets et on devra se contenter d'une solution sub-optimale dans certains cas.

[1] La redondance temporelle consiste à masquer une panne transitoire par un nombre fixé de tentatives successives.

3.1 Stratégie d'ordonnancement

L'élaboration d'une stratégie d'ordonnancement requiert la détermination de trois paramètres :

1. L'interconnection des nœuds ;
2. Le partitionnement des tâches ;
3. L'allocation des tâches.

Voyons plus en détails ces trois paramètres :

L'interconnection des nœuds : les différents nœuds peuvent être interconnectés de diverses manières. La topologie obtenue peut influencer grandement les performances du système distribué. Il convient, dès lors, de choisir avec soin la topologie du système distribué en fonction de l'application que l'on souhaite faire exécuter par le système. Les topologies d'interconnexion possibles sont, par exemple : bus, anneau, arbre binaire, étoile,...

Le partitionnement du problème consiste à décomposer le problème en sous-problèmes qui pourront être alloués aux différents nœuds. La granularité de cette découpe peut influencer les performances du système. Une granularité trop grande réduirait le parallélisme, tandis qu'une granularité trop fine engendrerait un surcoût en terme de délai de communication.

L'allocation des sous-problèmes traite la manière dont les différents sous-problèmes devront être distribués entre les différents nœuds du système. Cette allocation est notamment fonction de la topologie choisie.

3.2 Modèles d'ordonnancements

Il existe différentes méthodes permettant de calculer l'ordonnancement optimal, localement optimal ou sub-optimal.

La recherche de l'ordonnancement optimal étant, la plupart du temps un problème NP-complet, il est réservé pour des problèmes particuliers ayant, par exemple, un certain type de contraintes. Parmi les méthodes de calcul d'ordonnancement, on peut citer les suivantes :

- La *recherche exhaustive* : on recherche une solution optimale dans l'espace des solutions, par exemple, grâce à la méthode de "*branch-and-bound*" ;

- La *recherche opérationnelle linéaire ou non-linéaire* : on convertit le problème en un problème linéaire ou non linéaire, ensuite, on établit la fonction objectif à minimiser (en général le temps d'exécution) ainsi que l'ensemble des contraintes (dérivées des relations de précédence, des délais de communication, etc.). Enfin, on résoud le problème en utilisant des techniques de programmation dynamique ;
- Le *recuit simulé* ;
- Les *algorithmes génétiques* ;
- Les *méthodes heuristiques.*

4 Algorithme de répartition de charge dynamique

[12] et [1] donnent un aperçu d'une architecture générale d'un système de régulation de charge dynamique. Cette architecture permettra d'établir une classification plus fine des différents algorithmes possibles. Une taxonomie fonctionnelle des algorithmes de répartition de charge décompose le système en six composants distincts (cf. fig 1) :

- La *politique de participation* détermine les nœuds qui peuvent participer à la distribution de charge ;
- La *politique de sélection des candidats* détermine les jobs, processus ou objets qui peuvent être distribués ;
- La *politique de placement* détermine la paire de nœuds qui devront échanger des jobs, processus ou objets ;
- Le *mécanisme de mesure de charge* est la méthode utilisée pour mesurer la charge d'un nœud ;
- Le *mécanisme de communication* est la méthode utilisée pour échanger l'information (de charge, par exemple) entre les nœuds ;
- Le *mécanisme de transfert* est le protocole utilisé pour transférer les jobs, processus ou objets entre les nœuds.

Il est à remarquer que les choix pour les différents composants ne doivent pas être nécessairement indépendants. Il est tout à fait possible qu'une *politique de sélection des candidats*, de par l'information qu'elle requiert, détermine le *mécanisme de mesure de charge.*

De même, tous les composants ne sont pas requis. Si on prend l'exemple d'une *politique de sélection de localisation* aléatoire, un méchanisme de mesure de charge n'est pas nécessaire.

Voyons plus en détails ces différents composants :

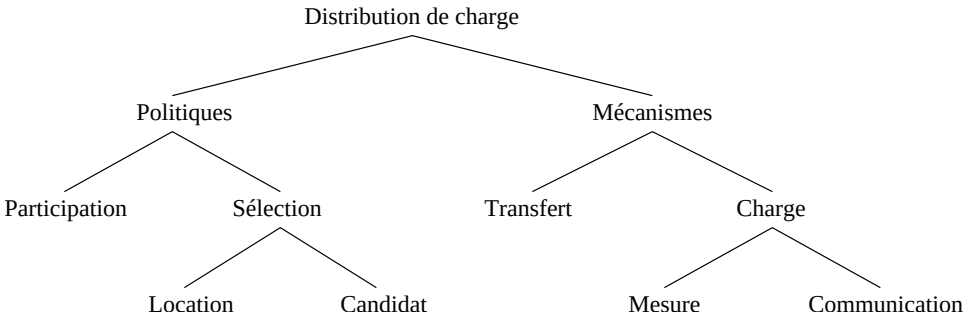

Fig. 1 – Taxonomie fonctionnelle des méthodes de répartition de charge

4.1 Les politiques de distribution de charge

La politique de participation permet de décider si un nœud particulier peut participer à la distribution de charge.

La solution la plus simple est de fixer une fois pour toute la liste des hôtes qui font partie du système.

La participation peut être conditionnée par l'absence d'activité aux périphériques d'entrée (clavier, souris,...) sur l'hôte considéré. Cette politique peut être très utile dans le cas où l'on désire utiliser les périodes d'inactivité de stations de travail dans un système distribué. Si une activité est détectée sur une station participant au système distribué, les processus étrangers qui pourraient subsister sur cet hôte se termineront normalement ou devront être migrés. Ce choix pourrait se baser sur certains critères comme la charge induite par ces processus étrangers ou le temps estimé de leur exécution. Ensuite, la station refusera tout processus étranger.

Le critère le plus utilisé est la charge de l'hôte. Dans ce cas, on peut distinguer deux types de participation d'un hôte : l'une pour exporter une partie de sa charge vers les autres hôtes, l'autre pour accepter de la charge d'autres hôtes. Pour cela, on associe un seuil à chacune de ces participations, ce qui permet de définir des politiques différentes quant à l'acceptation et l'exportation de charge vers d'autres hôtes.

On peut utiliser d'autres critères sur base, par exemple, de la sécurité.

La politique de sélection des candidats permet de déterminer le (ou les) processus qui sera (seront) migré(s)[2] vers un autre

[2] Les algorithmes ou stratégies qui utilisent la migration de processus sont dit *préemptifs*

nœud ou pour le placement initial d'un nouveau processus. On distingue quatre stratégies possibles qui diffèrent par leur coût et leur qualité de service :

- Sélection d'un candidat aléatoire ;
- Sélection d'un candidat raisonable ;
- Sélection d'un bon candidat ;
- Auto-sélection.

La *sélection d'un candidat aléatoire* est la solution la plus simple et la moins coûteuse en terme de temps de calcul pour déterminer le processus qui devra être exécuté sur un autre nœud. Cependant, il est évident que cette stratégie risque de choisir des processus qui possèdent des caractéristiques incompatibles avec la distribution. Parmi ces processus inadaptés, on peut citer les processus dont le temps d'exécution n'est pas suffisant pour justifier le coût de transfert du processus.

La *sélection d'un candidat raisonable* élimine d'emblée les processus qui ne sont pas adaptés à l'exécution distante. Le critère le plus souvent utilisé est le temps d'exécution qui peut être déterminé soit de façon statique, soit de façon dynamique.

La *sélection d'un bon candidat* est la stratégie la plus coûteuse mais la plus efficace. Dans certains cas, la politique de sélection peut coopérer avec la politique de placement afin de trouver le processus le mieux adapté au nœud sur lequel il devra être exécuté en fonction, par exemple, des ressources encore disponibles sur ce nœud. Dans le cas du placement initial, il faut connaître *a priori* les caractéristiques du processus avant son affectation au nœud destination. Par contre, si on utilise la migration de processus, certaines caractéristiques des processus telles que la consommation CPU ou mémoire peuvent être monitorées ce qui permet d'ensuite sélectionner le meilleur processus à migrer.

L'*auto-sélection* part du constat qu'un processus est le plus apte à connaître ses besoins et est donc le plus à même de faire le meilleur choix.

La politique de placement décide la paire de nœuds entre lesquels la charge devra être transferée. Cette décision peut se faire de manière centralisée ou bien de manière distribuée.

Lorsque la décision est prise de *manière centralisée*, un agent central décide sur base des informations sur la charge de chaque nœud participant au système distribué le nœud source et le nœud destination. Cet agent central est responsable du partage de ressources du système entre les différents nœuds. Les nœuds, quant à eux, décident de leur participation au système distribué et fournissent les informations de charge. Par exemple, lorsque l'agent

central découvre un nœud A peu chargé, il sélectionne un nœud surchargé B et ensuite autorise A à exécuter un processus de B. L'avantage de cette approche est sa simplicité. Par contre, il provoque un goulot d'étranglement au niveau de l'agent central, surtout lorsque le nombre de nœuds devient important. Aussi, ce nœud constitue un point central d'échec.

Une stratégie centralisée intermédiaire consiste à faire de la régulation hiérarchique, où les processus de régulation sont organisés suivant un contrôle hiérarchique [12]. C'est cette approche qui est utilisée dans LSF (cf. 5.4).

La politique de placement sera dite *distribuée* si tous les nœuds exécutent le même processus de régulation et jouent ainsi le même rôle. On distingue, à ce niveau, deux types de stratégies : les stratégies actives et les stratégies passives.

Dans les *stratégies actives*, la régulation se fait à l'initiative du nœud créateur de charge. Quand un nœud se rend compte que sa charge est trop importante, il tente de trouver un candidat susceptible d'accepter son excès de charge. Dans ce cas, comme le nœud source est fixé, la politique consiste à trouver le nœud destination, soit en sélectionnant le nœud le moins chargé, soit en choisissant un nœud au hasard dans l'ensemble des nœuds participant à la distribution de charge. Les principaux désavantages de cette technique sont que l'on introduit une charge supplémentaire sur un nœud qui est déja surchargé ; qu'un nœud peu chargé doit attendre jusqu'à ce qu'un autre nœud lui donne du travail.

Par contre, dans les *stratégies passives*, la régulation se fait à l'initiative de nœuds autres que le nœud créateur de charge. C'est un nœud en faible charge qui proposera l'exécution de processus d'un nœud en forte charge. Dans ce cas, c'est le nœud destination qui est fixé et la politique consiste donc à rechercher un nœud source susceptible de céder une partie de sa charge. Le choix du nœud source peut se porter soit sur le nœud le plus chargé ou un nœud au hasard parmi les nœuds chargés. Cette technique est moins utilisée.

Quand on compare les performances des stratégies actives et passives, on constate que les stratégies actives sont plus efficaces à faible et moyenne charge tandis que les stratégies passives le sont à forte charge.

Il existe aussi d'autres politiques intermédiaires. Par exemple, on pourrait combiner les stratégies actives et passives pour profiter au mieux de leurs caractéristiques respectives à faible et forte charge. Par exemple, si un nœud est surchargé, le mode actif est mis en œuvre tandis que si un nœud est peu chargé, c'est le mode passif qui sera appliqué.

L'avantage des politiques distribuées sont leur résistance aux pannes et aux reconfigurations du système.

4.2 Les mécanismes de distribution de charge

Les mécanismes de distribution de charge jouent trois grands rôles : le transfert de charge, la mesure de la charge et la communication de ses mesures.

Le transfert de charge Ce mécanisme est responsable du transfert physique de charge du nœud source vers le nœud destination. Ce transfert peut s'effectuer à deux moments : soit avant le début de l'exécution du processus sur le nœud source, on parlera alors de *placement initial* ; soit pendant l'exécution du processus sur le nœud source, on parlera alors de *migration de processus*.

La mesure de la charge Le but de la mesure de charge est de pouvoir comparer la charge des nœuds et ce, en fonction de la politique choisie. La mesure de la charge consiste généralement en une combinaison d'indicateurs comme :

- L'utilisation mémoire ;
- La taille de la file d'attente du CPU ;
- La taille de la file d'attente des périphériques E/S ;
- Nombre de processus exécutés ;
- Le temps depuis la dernière activité sur les périphériques d'entrée (clavier, souris, etc.) ;
- Le temps de réponse après une sollicitation multicast.

On peut aussi utiliser des modèles statististiques ou stochastiques afin d'établir des prédictions de charge.

La communication Une fois les mesures de charge collectées localement, il faut la transmettre à l'agent central pour une architecture centralisée, soit à tout ou à une partie des nœuds dans le cas d'une architecture distribuée. La difficulté principale vient du coût induit par la récolte et la distribution des informations de charge. Èn outre s'ajoute des problèmes de fiabilité et de consistance. La plupart des solutions existantes sont des variations à partir de quatres approches de base :

- Le *polling* est un message transmit à un nœud pour lui demander sa charge. Cette méthode fournit les informations les plus à jour mais possède un coût important ;

- L'échange de message par *broadcast* sur le réseau. Bien que moins coûteux que le polling, cette méthode produit une charge réseau assez conséquente. De plus, tous les hôtes du sous-réseaux vont recevoir ces données, qu'ils participent au système distribué ou non. Les avantages principaux de cette méthode est la possibilité de donner à tous les nœuds du système une information à jour et également la possibilité de synchroniser les différents nœuds ;
- L'utilisation du *multicast* possède les mêmes avantages que le broadcast mais permet en outre une meilleure sélectivité en ne mettant en communication que les hôtes participant à la distribution de charge. Un avantage supplémentaire est que le multicast permet de simplifier l'identification des hôtes participants, facilitant ainsi la sélection des nœuds source et destination pour la politique de placement ;
- Le "*collection agent*" utilise un système qui centralise les informations en un point. On peut distinguer trois approches qui varient selon la manière dont les informations sont collectées et redistribuées :
  - L'approche *globale* où chaque hôte transmet son information à l'agent central uniquement lorsque l'état de la charge locale a changé. L'information est périodiquement transmise par l'agent central à tous les nœuds du système ;
  - L'approche *centrex* est similaire à l'approche globale sauf que l'information n'est pas diffusée périodiquement à tous les nœuds du système mais seulement lorsqu'un nœud en fait la demande ;
  - L'approche *centrale* est, quant à elle, similaire à l'approche centrex mais l'agent central répond à un nœud qui désire exporter d'une partie de sa charge en suggérant un nœud destination.

  Les deux dernières approches réduisent la surcharge du réseau.

4.3 Algorithmes

On peut subdiviser les algorithmes de répartition de charge dynamique en deux classes [14] :

- Les méthodes *itératives* ;
- Les méthodes *directes*.

Les méthodes itératives tente d'uniformiser la charge sur les différents nœuds du système en échangeant de la charge entre nœuds voisins. A l'inverse, les méthodes directes déterminent d'abord les paires source/destination entre lesquelles la charge

devra être échangée. On présentera la méthode directe ainsi que deux méthodes itératives : la *diffusion* et le *gradient*.

Les algorithmes directs. Dans ce type d'algorithme, on estime tout d'abord la charge moyenne du système simplement en sommant les charges de chaque nœud et en divisant le tout par le nombre total de nœuds. Ensuite, cette moyenne est transmise vers chaque nœud qui peut dès lors déterminer si sa charge est supérieure ou inférieure à cette moyenne. Puis, en utilisant des outils issus de la théorie des graphes, les mouvements de charges à effectuer sont déterminés de telle façon que le nombre de ces mouvements soit minimal. Ces mouvements dépendent évidement de la topologie choisie pour le réseau d'interconnexion.

La diffusion. Le principe général de cet algorithme est que chaque nœud échange de la charge avec un ou plusieurs de ses voisins, si ceux-ci sont moins chargés que lui. Il s'en suivra une "diffusion" de la charge qui tendra à équilibrer la charge entre chaque nœud. Cette algorithme s'apparente au phénomène naturel de diffusion.

Une version accélérée de cet algorithme est présentée dans [5].

Le gradient. Cette méthode consiste à calculer des gradients sur base des différences de charge dans le système. A chaque nœud, on associe une "altitude". Un nœud surchargé correspondra à une haute altitude, tandis qu'un nœud peu chargé correspondra à une basse altitude. La charge va alors migrer en suivant les pentes les plus abruptes, donnés par les gradients les plus élevés.

5 Implémentations disponibles

Cette section enumère quelques implémentations disponibles des logiciels ou bibliothèques permettant de transformer un ensemble de machines en un système distribué.

5.1 Parallel Virtual Machine (PVM)

PVM [11] est une API permettant à un ensemble hétérogène de machines (Unix et Windows) connectées ensemble par un réseau d'être utilisé comme une seule machine parallèle.

5.2 Message Passing Interface (MPI)

MPI [10] est un standard spécifiant une interface permettant l'écriture de programmes qui échangent des messages.

5.3 Local Area Multicomputer (LAM)

LAM [7] est une implémentation portable de MPI.

5.4 Load Sharing Facility (LSF)

LSF [8] est un logiciel permettant de contrôler et coordonner un ensemble d'ordinateurs (appelé "*ferme*"). Ce logiciel a deux fonctions principales :

1. donner une vision unique d'un ensemble d'ordinateurs, sous la forme d'une ferme ;
2. permettre la répartition des travaux (commandes, programmes utilisateurs, applications) sur les différents éléments de la ferme, afin d'équilibrer au mieux la charge.

5.5 Multicomputer Operating System for UnIX (Mosix)

Mosix [9] est un logiciel spécialement conçu pour donner des capacités de clustering à Unix.

6 Conclusion

Il n'y a pas de méthodes ou algorithmes de régulation de charge qui puisse être appliqué universellement et de façon optimale. Les performances dépendent fortement des caractéristiques des applications. Il est donc conseillé d'étudier au cas par cas les caractéristiques de l'application à distribuer afin de choisir la méthode de régulation la plus appropriée. Dans cet article, il a été proposé une classification de ces méthodes de régulation, la régulation ayant été étudiée selon deux approches : statique et dynamique.

Pour la régulation statique, il existe différents algorithmes permettant de déterminer la distribution des processus sur les différents nœuds du système. Cette méthode permet souvent d'obtenir de bonnes performances. Mais elle requiert une connaissance *a priori* du comportement des différents processus que

comporte l'application que l'on désire distribuer et plus particulièrement l'influence de l'échange de messages sur les performances du système. De plus, elle possède les inconvénients majeurs de nécessiter une recompilation de l'application en cas d'ajout ou suppression d'un nœud et d'être peu résistante aux pannes ou reconfigurations du système.

En ce qui concerne la régulation dynamique, une classification fonctionnelle a été proposée, qui décompose un système de répartition de charge dynamique en trois politiques, à savoir : participation, sélection et placement ; et en trois mécanismes, à savoir : mesure de charge, communication et transfert. Cette classification permet de comparer les caractéristiques d'un grand nombre d'algorithmes de répartition de charge dynamique.

Références

1. K. Budbendorfer and J. H. Hine. A compositional classification for load-balancing algorithms. Technical report, Victoria University of Wellington - Department of Computer Sciences, July 1998.
2. Fabrice Gadaud and Valéry Touodop. Mosix. Technical report, École Polytechnique de l'Université de Nantes - Département Systèmes Informatiques, Logiciels &réseaux.
3. Bruce Hendrickson. Load balancing fictions, falsehoods and fallacies. *Applied Mathematical Modelling*, 2000.
4. John Hine and Teijo Holzer. Task allocation in a distributed system. Technical report, Victoria University of Wellington - Department of Computer Sciences.
5. Gregory Karagiorgios and Nikolaos Missirlis. Accelerated diffusion algorithm for dynamic load balancing. *Information Proceding Letters*, 2002.
6. Herbert Kuchen and Andreas Wagener. Comparison of dynamic load balancing strategies. Technical report, Lehrstuhl für Informatik II - RWTH Aachen.
7. LAM / MPI Parallel Computing. `http://www.lam-mpi.org/`.
8. LSF. `http://accl.grc.nasa.gov/lsf/`.
9. MOSIX. `http://www.mosix.cs.huji.ac.il/`.
10. The Message Passing Interface (MPI) standard. `http://www-unix.mcs.anl.gov/mpi/`.
11. PVM (Parallel Virtual Machine). `http://www.csm.ornl.gov/pvm/`.
12. E. G. Talbi. Régulation de charge dans les systèmes distribués et parallèles. Technical report, Laboratoire d'Informatique Fondamentale de Lille - Université de Lille I.
13. Andrew S. Tanenbaum and Maarten Van Steen. *Distributed System : Principles and Paradigms.* Prentice Hall, 2002.
14. Jie Wu. *Distributed System Design.* CRC Press, 1998.