# Un schéma modulaire pour l'écriture des ordonnanceurs

Gerson G. H. Cavalheiro[a], Jean-Louis Roch[b]

LMC – IMAG – Projet APACHE
Institut National Polytechnique de Grenoble
100 rue des Mathématiques
B.P. 53 F-38041 Grenoble Cedex 9
[a]Boursier CAPES–COFECUB (Brésil)
[b]Maître de Conférences INPG
CNRS, INRIA, INPG et UJF

**Résumé**

Ce travail présente un schéma modulaire pour l'écriture des algorithmes dynamiques d'équilibrage de charge en environnements de calcul parallèle ou distribué. Le modèle de ce schéma est fondé sur l'identification de quatre modules indispensables pour l'implantation d'un environnement d'équilibrage de charge (EEC) et de l'interface de communications entre ces modules. L'utilisation de tels environnements permet l'implémentation d'algorithmes d'équilibrage de charge (AEC) qui ne soient ni imbriqués dans une application, ni dépendants d'une architecture de machine spécifique. Pour illustrer l'utilisation pratique du schéma proposé, des résultats numériques sont présentés à partir des expériences réalisées sur une grappe homogène de stations de travail et sur une machine parallèle. La conclusion revient sur l'intérêt de l'EEC dans l'environnement de programmation parallèle ATHAPASCAN.

**Mots-clés :** Équilibrage de Charge, Ordonnancement Dynamique, Programmation Modulaire

## 1. Introduction

Les algorithmes d'équilibrage de charge (AECs) sont des outils indispensables pour gérer les ressources des machines parallèles. Leur but est d'exploiter les ressources des machines de façon à optimiser l'exécution d'une application selon un, voire plusieurs, critères de performance. La littérature présente un grand nombre d'AECs, quelques exemples sont donnés en [1] et [6] qui cherchent à optimiser le temps d'exécution. D'autres travaux, comme [2], essaient de les structurer dans une classification. Cependant, la liaison entre les efforts de classification d'AECs et d'implémentation des outils d'équilibrage de charge est très faible : les classifications ne peuvent être utilisées que pour donner les bases du comportement des algorithmes et éventuellement à fin comparative ; ces classifications ne permettent pas de spécifier un algorithme au niveau implémentation.

Dans ce travail, nous présentons un schéma modulaire pour l'implémentation des AECs sur des machines parallèles ou distribuées. Grâce à ce schéma, nous approchons la description d'un algorithme au niveau de son implémentation. Pour cela nous étendrons la discussion sur les environnements d'équilibrage de charge (EEC) pour aborder aussi les interfaces de l'AEC avec *i*) son utilisateur – le producteur de travail, et *ii*) le support d'exécution – la machine cible (dont la structure interne est décrite dans la littérature comme composée d'une unité de contrôle et d'une unité d'information de charge). Nous proposons donc la décomposition de la structure d'un EEC en quatre modules : le module de construction de tâches (MCT), le module de gestion (MG), le module d'information de charge (MIC) et le module d'exécution (ME) en caractérisant l'interaction entre ces modules.

La section suivante (2) présente une synthèse des bases de l'EEC que nous proposons et de son interface et le modèle de tâche manipulé. L'utilisation de l'environnement est illustrée dans la section 3 à travers un extrait de résultats obtenus sur des expérimentations réalisées. Finalement la section 4 présente une discussion sur l'utilisation et l'intérêt d'un tel EEC.

## 2. La structure de l'environnement d'équilibrage de charge

Les EECs peuvent être vus comme des *ressources qui effectuent la gestion de ressources*, où les consommateurs sont les applications et les ressources gérées sont les machines parallèles cibles [2]. Selon cette "philosophie", plusieurs AECs ont été développés et implémentés ; dans la plupart des cas, ces implémentations privilégient soit une machine cible, comme par exemple les algorithmes basés sur le principe du *gradient* [4], soit une classe d'applications comme [5] pour la dynamique moléculaire. D'autres travaux, comme [3], proposent une structuration des EECs de façon à faciliter le développement et la réutilisation des AEC. Inséré dans ce dernier cadre, le schéma proposé permet de séparer l'AEC à la fois de l'application et du support d'exécution, pour pouvoir facilement être réutilisé ou bien adapté à un autre couple [application, machine].

### 2.1. La tâche : unité de service

L'unité de base d'un service spécifié par l'application est dénommée *tâche*. Les descriptions des tâches sont soumises par l'application à l'EEC pour que son exécution soit effectuée. Cette description, en plus de la séquence d'instructions qu'implémente le service de la tâche – parmi lesquels on trouve des instructions de création d'autres tâches – contient aussi une interface de manipulation, pour permettre à l'EEC de la déplacer et de contrôler son exécution ainsi qu'une collection d'attributs et de contraintes. Les attributs peuvent être exploités par l'EEC pour optimiser l'exécution de l'application. Les contraintes doivent forcément être prises en compte par l'EEC afin d'éviter une exécution erronée de l'ensemble de services créés par l'application.

Parmi les attributs qui peuvent être rendus disponibles par la tâche, il est possible d'avoir la représentation de son *état actuel* qui dit si la tâche est prête ou pas à être exécutée, ou encore si elle est bloquée dans l'attente d'une donnée ou d'un autre événement. D'autres attributs peuvent être par exemple son *coût*, sa *priorité* par rapport à d'autres tâches, la *liste de paramètres* nécessaires à son exécution et leur *localisation* s'il s'agit d'une machine à mémoire distribuée. Au cas où l'application gère un graphe d'exécution de tâches, une tâche peut ainsi contenir comme attributs la liste de ses *successeurs* et de ses *prédécesseurs* dans le graphe.

Au contraire des attributs, qui peuvent ou pas être exploités par l'EEC de façon à mieux équilibrer la charge, les contraintes d'exécution explicitées par une tâche *doivent* être respectées par l'EEC. Dans l'ensemble de contraintes valables, une tâche peut définir une *liste de nœuds* où elle peut être exécutée, cas typique dans un environnement hétérogène, ou encore la quantité minimale de *mémoire* nécessaire pour son exécution sur un nœud.

L'interface de la tâche donne accès aux fonctionnalités de *déplacement* entre nœuds et de contrôle d'exécution : *exécution* pour démarrer l'exécution du service de l'application et *préemption* pour être interrompue – redémarrage par un nouveau appel à *exécution* sur le même site ou sur un autre, après un déplacement de la tâche.

### 2.2. Caractérisation des modules

Les quatre modules de l'EEC sont présentés dans la figure 1 avec les flux de communication entre eux. Dans ce schéma le *consommateur* de l'EEC est représenté par le MCT et les *ressources* de calcul par le ME. L'ensemble est présent sur tous les nœuds de la machine; cependant, l'accès aux services d'un module est fait toujours localement et d'une façon synchrone non bloquante. La discussion de l'interface de communication interne des modules est propre à l'implémentation de l'algorithme et n'est pas abordée dans cet article.

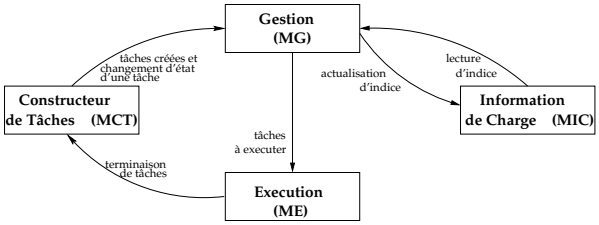


FIG. 1 – *Les flux de communication entre les modules de l'environnement d'équilibrage de charge*

Le MCT produit des descriptions de tâches en liaison avec l'application d'un utilisateur. Toutes les descriptions créées sont envoyées au MG pour être éventuellement exécutées. Dans le cas où le MCT gère un graphe de précédence de tâches, il peut profiter du message provenant du ME pour connaître le moment de terminaison d'une tâche et pouvoir actualiser le graphe de précédence.

Le MG est le cœur de l'EEC où est implémenté les actions de décision de l'AEC. Il a l'accès à toutes les tâches dès qu'elles sont créées et jusqu'à leur terminaison. Cependant, cet accès permet seulement de lire les attributs et les contraintes des tâches et de déclencher les opérations de déplacement, exécution et préemption. L'application (via le MG) doit informer l'ordonnanceur des changements d'états des tâches, qui correspondent à des modifications de ces attributs ; de tels changements peuvent entraîner de nouvelles prises de décision par l'ordonnanceur. L'AEC accède au MIC pour avoir des informations sur l'état de la machine et aux attributs et contraintes des tâches pour attribuer à chaque tâche une data et un site pour son exécution – décision qui pourrait être remise en question à n'importe quel moment d'activité du MG. À son coté est construit le MIC, pour fournir le mécanisme de contrôle des indices de charge nécessaires à l'exécution correcte de l'AEC implémenté. Le MIC peut être programmé pour *réveiller* le MG quand une distribution de charge déséquilibrée est repérée.

Le ME fournit au EEC un groupe de $n \geq 1$ fils d'exécution sur chaque nœud. La quantité de fils d'exécution est définie par le MG selon ses besoins et peut varier en cours d'exécution. Chaque fil exécute séquentiellement une tâche jusqu'à qu'elle soit finie ou préemptée. Des qu'un fil devient inactif, il attend que lui soit attribué une nouvelle tâche par le MG.

### 2.3. Correction de l'EEC

Tout AEC implémenté sur les bases de l'EEC proposé doit être utilisable par n'importe quelle application sur n'importe quelle machine. Les performances peuvent ne pas être satisfaisantes, mais les résultats, eux, doivent être corrects. Pour que l'EEC puisse garantir cette correction d'exécution, tout AEC doit d'une part respecter les contraintes propres aux tâches et d'autre part garantir que toute tâche prête sera exécutée exactement une fois.

De plus, la vivacité de l'EEC doit être assurée : pour cela, tout AEC doit garantir que, s'il existe des tâches *prêtes* et des processeurs inactifs, l'exécution de nouvelles tâches doit être déclenchée au bout d'un temps fini. En conséquence, l'AEC ne doit pas entraîner d'interblocage.

## 3. Utilisation pratique

Les courbes présentées dans la figure 2 ont été obtenues par l'exécution d'une application qui implémente un algorithme de découpe récursif dont chaque tâche génère d'autres tâches (deux) jusqu'à atteindre un seuil. Le seuil choisi détermine le grain de parallélisme désiré. Chaque courbe représente le gain de performance (*speed-up*) obtenu par l'exécution de l'algorithme parallèle sur 4 nœuds par rapport à son exécution sur 1 nœud ; l'axe des $X$ représente l'impact du nombre de fils d'exécution fourni par le ME sur chaque nœud. Le graphe (a) provient de l'expérimentation réalisée sur un IBM-SP2 et le graphe (b) de la répétition de la même expérience sur un réseau homogène de 4 nœuds (stations SUN/Solaris, réseau Mirynet).

Deux grains ont été utilisés pour définir le seuil de génération de tâches de cette application (de seuil maximal 40) : un pour générer un grand nombre de tâches de faible coût (seuil = 15) et un autre pour générer moins de tâches de coût plus élevé (seuil = 25). L'application a été ordonnancée par deux AEC basés sur le principe du vol de travail : dès que un nœud n'a plus de tâches prêtes localement, un message est envoyé à un autre nœud, choisi au hasard, pour demander du travail. Lorsqu'un nœud reçoit une requète de vol, il renvoie au demandeur une de ses tâches prêtes si il en possède; sinon, il transmet la demande de vol à un autre nœud, lui aussi choisi au hasard. Pour le premier AEC, appelé DAB dans la figure 2, le nœud qui reçoit une réquisition de travail envoie comme réponse la derniére tâche qui est devenue prête. Le deuxième AEC, appelé COUT, analyse les coûts d'exécution des tâches et envoie en réponse d'un vol la tâche la plus coûteuse.

D'après les courbes présentées, on constate que, pour cette application, le réseau est plus adapté a l'exécution de tâches de petite taille et que ses nœuds ne supportent pas bien la concurrence pour l'accès au processeur réel. Pour la machine parallèle, on vérifie sa sensibilité à une bonne distribution de la charge et son adaptation à des tâches de plus grand coût.

À titre indicatif, le temps de l'exécution séquentielle de l'application est d'approximativement 85 se-

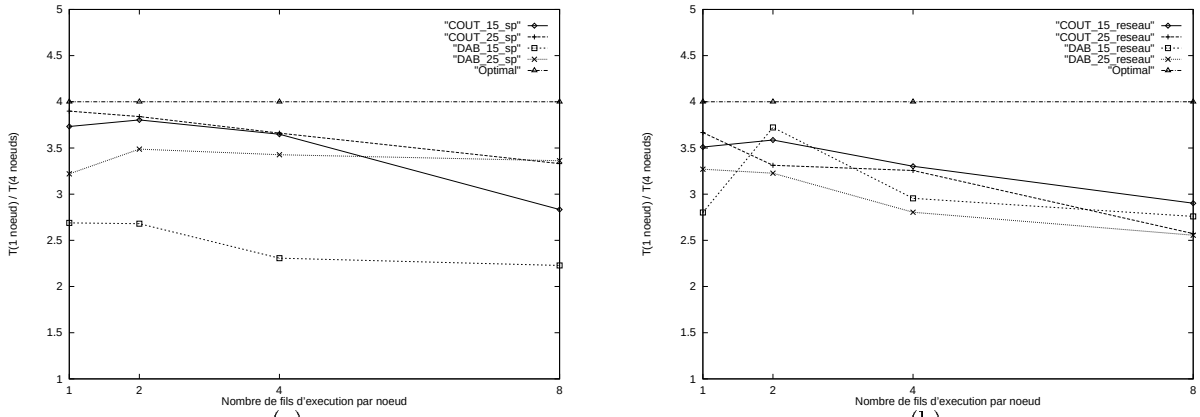


FIG. 2 – *Performance de deux AEC sur (a) une machine parallèle et (b) un réseau homogène*

condes sur un nœud du réseau homogène et de 75 secondes sur un nœud de la machine parallèle. Le nombre de tâches générées est 32000 avec un seuil de 25 et 393000 avec un seuil de 15.

## 4. Conclusion

Dans ce travail, nous avons présenté une interface virtuelle pour l'implémentation d'algorithmes d'équilibrage de charge (AECs) dans le but de les séparer à la fois du niveau application et du niveau architecture. Cette interface permet ainsi de donner une description d'un algorithme d'AEC qui puisse dériver vers une implémentation. Nous avons décrit un environnement composé de quatre modules qui couvrent les services fournis ou requis par les environnements d'équilibrage de charge (EEC) : un niveau *utilisateur de l'EEC* (MCT), responsable de la création de tâches de l'application, une centrale de décision (MG), un moniteur de charge de la machine (MIC) et une unité d'exécution (ME) liée à la machine. Il a été donné une considération particulière à l'interface de communication entre ces modules.

L'EEC proposé a été implémenté pour supporter le noyau d'ordonnancement d'ATHAPASCAN-1, à partir duquel nous avons obtenu les mesures présentées. Cette implémentation, denommée Athapascan-GS, a été réalisée en langage C++, afin d'utiliser les avantages de la programmation orientée objet pour la définition d'interface virtuelle et faciliter l'ajout d'autres AECs. L'utilisation de fils d'exécution multiples, coopérant par communication et mémoire partagée, est rendue possible par ATHAPASCAN-0 qui fournit une couche portable.

La suite naturelle de ce travail est l'implémentation d'un ensemble d'algorithmes d'équilibrage de charge, soit d'algorithmes présentés dans la littérature, soit d'algorithmes obtenus à partir du développement de nouvelles politiques d'ordonnancement. L'ensemble des algorithmes implémentés sera disponible avec ATHAPASCAN-1 sous la forme d'une bibliothèque indépendante.

## Bibliographie


1. Blumofe (R.D.) and Leiserson (C.E.). – Scheduling multithread computation by work stealing. – *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pp. 356-368, Santa Fe, Nov. 1994.
2. Casavant (T.L.) and Kuhl (J.G.). – A taxonomy of scheduling in general-purpose distributed computing systems. – *IEEE Trans. Soft. Eng.*, 14(2):141-154, Feb. 1988.
3. Jacqmot (C.). – *Load management in distributed computing systems*: Towards Adaptative Strategies. – DII - UCL , PhD Thesis, Louvain, Belgique, Jan. 1996.
4. Lin (F.C.H.) and Keller (R.M.). – The gradient model load balancing method. – *IEEE Trans. Softw. Eng.*, 13(1):32-38, Jan. 1987.
5. Bernard (P.-E.). – *Parallélisation et multiprogrammation pour une application irrégulière de dynamique moléculaire opérationnelle.* – LMC - INPG, Thèse de Doctorat, Grenoble, France, Oct. 1997.
6. Willebeek-LeMair (M.H.) and Reeves (A.P.). – Strategies for dynamic load balancing on highly parallel computers. – *IEEE Trans. Parallel and Distributed Systems*, 4(9):979-993, Sept. 1993.