Chapitre 5 - Placement de composants logiciels : Application à l'équilibrage de charge

«La simple existence d'un problème ne prouve pas qu'il y ait une solution» (proverbe yiddish)

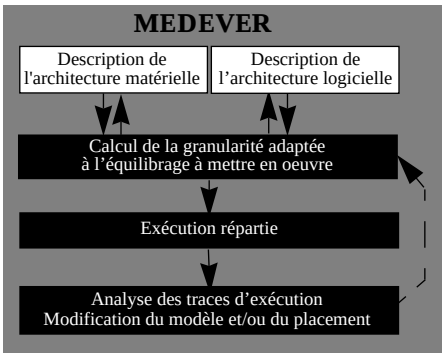
Résumé

L'équilibrage de charge est un point de vue orienté système du placement dynamique d'applications parallèles et réparties. C'est pourquoi, comme préambule à ce chapitre, nous réalisons un état de l'art sur les techniques de placement et notamment a celles adaptées à des architectures hybrides. Nous décrivons ensuite les choix architecturaux d'IDEFIX, une plate-forme expérimentale de partage et d'équilibrage de charge.

Apports scientifiques

Pour démontrer la validité de notre méthode MEDEVER, nous avons cherché à intégrer nos langages de description d'architectures avec une plate-forme de répartition de charge. Le placement est alors effectué à partir de la description d'une architecture logicielle réalisée en VODEL-D et d'une architecture matérielle décrite avec HADEL. Les algorithmes de placement utilisés sont ceux offerts par la plate-forme de répartition de charge. Nous avons constaté que HADEL et VODEL-D offrent bien plus d'informations que n'en utilisent les plate-formes de répartition de charge pour réaliser le placement dynamique. Ceci ouvre alors la voie à la recherche de nouveaux algorithmes travaillant sur un ensemble plus riche de données de base. Il faut néanmoins rester conscient que le calcul du placement dynamique d'un composant logiciel doit être effectué dans un temps court et borné. Un équilibre est donc à respecter entre les calculs à réaliser et la qualité du placement que l'on recherche.

Plan



Equilibrage de charge

Etat de l'art sur les techniques de placement

☞ Conception d'une plate-forme de placement hybride basée sur notre méthode MEDEVER, sur une plate-forme d'équilibrage de charge et sur les langages HADEL, VODEL-D

Mots clefs

Gatostar, IDEFIX, équilibrage de charge, placement statique, placement dynamique, placement hybride, application répartie, application parallèle, IDEFIX, composants logiciels.

Nous avons mis en oeuvre l'équilibrage de charge en axant nos recherches vers des techniques de placement dynamique sur des architectures matérielles hybrides sur réseau local. Les architectures hybrides sur réseau local offrent une puissance de calcul très importante du fait de la puissance conjuguée des machines mono et multiprocesseurs et de la bande passante disponible. Si la puissance théorique de ces plate-formes matérielles est incontestable, leur utilisation optimale fait encore l'objet de recherches poussées [PRC 96]. On cherche alors des algorithmes de placement statique et surtout dynamique, capables d'équilibrer et de partager la charge sur des machines hétérogènes. C'est pourquoi l'utilisation d'un outil d'équilibrage de charge dans ce cadre nous a semblé judicieux. Nous avons alors opté pour Gatostar [Folliot & al. 95], qui est un logiciel développé dans notre équipe et pour lequel disposons du code source.

Néanmoins, les outils d'équilibrage de charge ne gèrent pas, en général, une description très élaborée à la fois de l'architecture matérielle support de l'exécution d'une application et de l'architecture logicielle. Or notre langage HADEL est particulièrement adapté à la description d'architectures matérielles hybrides et VODEL-D est dédié à la description d'application réparties et parallèles. D'où l'idée de coupler nos langages avec les services offerts par une plate-forme d'équilibrage de charge. Le cycle de développement d'une application répartie et parallèle reste donc inchangé et s'appuie sur notre méthode MEDEVER. et sur un environnement d'exécution réparti. Ces travaux ont donné naissance à IDEFIX une plate-forme expérimentale de partage et d'équilibrage de charge.

Dans la section 1, nous présentons un état de l'art sur les techniques de placement statiques et dynamiques et les différentes familles d'algorithmes qui s'y rattachent. Nous concluons alors sur le fait que le placement d'applications parallèles et réparties sur des architectures hybrides modifie considérablement le problème. Nous décrivons ensuite, dans la section 2 et dans le cadre de la méthode MEDEVER, comment nous avons construit une plate-forme expérimentale de placement hybride. Enfin, la section 3 conclue sur la validité et l'intérêt d'une telle plate-forme.

1. Placement sur des systèmes parallèles et répartis

Un système parallèle ou réparti est un ensemble de processeurs reliés par un réseau de communication. Les processeurs communiquent par échange de messages. Un système parallèle est un système réparti fortement couplé utilisant un ensemble de processeurs homogènes reliés entre eux par des liens à haut débit. Le réseau d'interconnexion est statique (arbre, grille, hypercube) ou reconfigurable. Le coût de communications entre ces processeurs n'est pas uniforme car il dépend souvent du chemin à parcourir entre l'émetteur et le récepteur. En général, une machine parallèle est mono-utilisateur et mono-application.

L'exécution de programmes parallèles et/ou répartis sur une architecture matérielle répartie est un problème difficile. Le placement de programmes parallèles est fortement lié aux langages parallèles, aux compilateurs et aux machines spécifiques utilisés. Néanmoins, les modèles de programmation des machines parallèles sont en général de deux types :

- dans le modèle SPMD (*Single Program Multiple Data*) un même programme est réparti sur plusieurs unités de traitement. Certaines différences existent quant à leur mise en oeuvre. Ainsi, sur une machine SIMD, chaque instruction est exécutée en même temps, alors que sur une machine MIMD, chaque unité de traitement est susceptible d'exécuter n'importe quel morceau de programme.
- dans le modèle MPMD (*Multiple Program Multiple Data*) plusieurs programmes distincts sont répartis entre plusieurs unités de traitement.

Le placement de programmes répartis est quant à lui, rendu difficile par l'hétérogénéité des processeurs utilisés et donc par des problèmes de gestion des formats et de l'alignement des données. Il doit aussi prendre en compte les coûts des communications entre sites et le partage dynamique des machines entre plusieurs utilisateurs (ce qui n'est en général pas le cas des machines parallèles). Il est donc clair que la traduction d'un langage de haut niveau en binaire ne suffit plus à la création d'une application répartie ou parallèle. On doit alors prévoir avant l'exécution sur quels processeurs, sur combien de processeurs et suivant quelles stratégies, placer les processus et les ressources d'un programme. Comme les composants actifs de l'application communiquent par échanges de messages, il faut aussi réduire au maximum, les coûts de communications. Les solutions aux contraintes qui régissent le placement d'applications réparties et parallèles sont diverses et passent par des techniques :

- d'équilibrage de la charge ;
- de minimisation du nombre de messages échangés ;
- de minimisation des délais de routage ;
- de respect des contraintes dues à la machine comme la mémoire, la place disque et la puissance du CPU.

Il faut donc trouver des algorithmes d'allocation qui automatisent la tâche de placement. Le but est d'offrir une programmation transparente des machines parallèles et une portabilité des programmes entre les diverses structures de communication des architectures parallèles. Nous présentons dans un premier temps deux modes de représentation des programmes servant aux algorithmes de calcul de placement. Ensuite, nous concentrons notre étude sur les algorithmes de placement statique dans un premier temps et dynamique dans un second temps. Enfin, nous tentons d'unifier les approches de placement sur des systèmes parallèles et répartis et d'introduire la notion de placement hybride.

1.1. Représentation des programmes

Un programme est généralement représenté par un graphe de précédence ou par un graphe de tâches. Ces différentes représentations sont utilisées dans les algorithmes de placement courants. Nous présentons ces deux types de graphes ci-dessous.

1.1.1. Graphes de précédence

Dans le cas de graphes de précédence, un programme est représenté par un graphe orienté. Dans ce graphe, les sommets sont l'ensemble des tâches élémentaires 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.

1.1.2. Graphes de tâches

Si on ignore les relations de précédence entre les tâches, un arc non orienté représentant uniquement un lien de communication, on obtient un graphe de tâches. Cela revient en quelque sorte à ignorer le parallélisme relatif des tâches. Aux noeuds et aux arêtes sont associés des coûts de calcul et de communication.

1.1.3. Synthèse

Les spécificités de chacun des modèles de graphes sont résumées dans le Tableau 38.

Tableau 38: Comparaison des graphes de dépendance et des graphes de tâches

Graphe de dépendance	Graphe de tâches
Graphe orienté	Graphe non orienté
Indique des relations de précédence entre des processus ou des applications	Indique des relations d'échange de données entre des processus ou des applications
Outil d'ordonnancement	Outil de placement

Les algorithmes réguliers tels que ceux de l'algèbre linéaire s'expriment naturellement sous forme de graphe de précédence. De nombreux travaux théoriques existent sur la résolution des problèmes de placement vue du coté de l'ordonnancement avec des graphes de précédence [Gonzales 77], mais peu de réalisations pratiques sont proposées [Bouvry & al. 93]. Le modèle du graphe de tâches sert, quant à lui, de base à la plupart des algorithmes de placement [Billionnet & al. 89]. Enfin, il est aussi possible de mixer les deux [Folliot 92].

Ces algorithmes qu'ils soient statiques ou dynamiques sont l'objet de la section suivante.

1.2. Les algorithmes de placement

Soit une machine parallèle composée d'un ensemble P de n processeurs (p1,..., pn) qui communiquent entre eux par l'intermédiaire d'un réseau d'interconnexion (statique ou reconfigurable).

Soit un programme parallèle composé d'un ensemble de k processus qui communiquent entre eux par messages. L'allocation de l'ensemble des processus vers l'ensemble des processeurs est une fonction qui propose n^k placements possibles. Dans le cas général, le calcul d'un placement optimal parmi les n^k possibles est NP complet. En général, le processus d'optimisation consiste à minimiser une fonction coût. Cette dernière peut représenter soit le temps d'exécution, soit le nombre de processeurs utilisés, soit le nombre de ressources utilisées, etc.

On distingue les méthodes de placement, suivant l'instant où le placement est décidé d'une part et suivant le type d'architecture sur lequel le placement a lieu (cf. Figure 58).

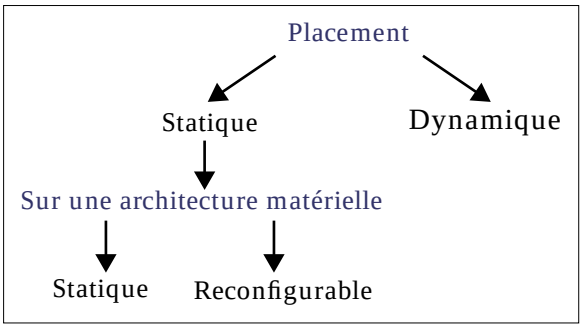


Figure 58 : Classification générale des algorithmes d'allocation

L'allocation statique décide du placement des processus au moment de la compilation ou du chargement du programme.

L'allocation dynamique fait un placement à l'exécution par transfert de processus. Pour le cas SIMD, puisqu'il n'existe qu'une seule unité de contrôle (Single Instruction), le seul problème qui se pose est celui de l'allocation de données dans les mémoires de processeurs. Dans le cas MIMD, on distingue les processeurs à mémoire commune ou non. S'il existe une mémoire commune, la connaissance de l'état global du système et du transfert de processus ne se pose pas. L'ajout de processeurs ou la montée de la charge finit par

créer un goulot d'étranglement au niveau de l'accès exclusif à certaines partie de la mémoire. Pour les architectures MIMD sans mémoire commune, la plupart des algorithmes d'allocation proposés sont généralement statiques et peu d'entre eux ont été implémentés [Norman & al. 93].

Dans la littérature, les modèles proposés pour représenter les programmes et les architectures (modèles probabiliste, théorie des graphes, théorie des files d'attente,...) et les méthodes de résolution (programmation dynamique, procédure par séparation et évaluation) diffèrent. Nous présentons dans la suite de cette section un état de l'art rapide des méthodes utilisées dans le cas de l'allocation statique, puis dynamique.

1.2.1. Algorithmes d'allocation statique

Les algorithmes de placement statique se situent entre l'étape de compilation et l'étape de chargement de l'application sur la machine cible. En général, les modèles utilisés nécessitent une connaissance de l'ensemble des tâches, de leur relations de précédence et de leur coût de communication. On cherche ensuite alors à les comparer en terme de complexité ou de granularité.

On sous entend donc que ces valeurs ne se modifieront pas au cours de l'exécution et leur obtention peut être le résultat de [Hémery 94] :

- une analyse faite durant l'étape de compilation ;
- une modélisation analytique de l'application (par exemple avec des files d'attente ou des réseaux de Petri stochastiques) ;
- une simulation qui évite de disposer d'une machine parallèle dans la phase de développement d'une application ;
- d'une évaluation des exécutions précédentes.

Dans le cas de l'allocation statique, le type d'architecture matérielle utilisée est important et on distingue alors deux cas [Talbi 93] :

- le placement est fait sur une architecture dont le réseau d'interconnexion des processeurs est prédéfini. La plupart des algorithmes traitent alors du cas où le nombre de processeurs et le réseau d'interconnexion sont fixés (par exemple [Lo & al. 89] traite de l'Hypercube). Soit la topologie est fixée, mais le nombre de processeurs, lui, n'est pas fixé et dépend de la disponibilité des processeurs et/ou du meilleur compromis entre le nombre de processeurs le gain obtenu (*speed-up*).
- le placement s'accompagne d'une réorganisation de l'architecture sous-jacente (très souvent le cas dans le domaine des transputer [Inmos 88]).

Dans la suite de ce paragraphe, nous présentons d'abord les algorithmes d'allocation statique sur architecture statique optimale, puis sur architecture non optimale. Nous détaillons ensuite les algorithmes d'allocation statique sur architecture reconfigurable.

1.2.1.1. Allocation statique sur architecture statique optimale

Les architectures statiques optimales sont soit basées sur la théorie des graphes, soit basées sur la programmation mathématique.

Architectures statiques optimales basées sur la programmation mathématique

Les architectures statiques optimales basées sur la programmation mathématique, quant à elles, se ramène à un problème d'optimisation combinatoire. Elles utilisent en général les techniques de la programmation non linéaire en nombres entiers pour sa résolution. On utilise alors des techniques de programmation mathématique comme par exemple la programmation dynamique [Bokhari 81 & Hansen & al. 86] et les procédures par séparation et évaluation (*Branch and Bound*) [Sinclair 87, Chu & al. 90]. En général, il s'agit de minimiser les fonctions coûts, tout en respectant un certain nombre de contraintes. On peut s'intéresser aux contraintes temporelles, de charge, de ressources, de redondance.

Architectures statiques optimales basées sur la théorie des graphes

Les architectures statiques optimales basées sur la théorie des graphes modélisent le problème par des graphes et utilisent des techniques de traitement de graphes pour le résoudre (recherche d'homomorphisme faible entre deux graphes, partition de coupe minimale d'un graphe [Stone 77]).

Les stratégies basées sur la théorie des graphes modélise l'architecture cible par un graphe connexe $G_h = (P_h,L_h)$ où :

- P_h est l'ensemble des sommets et représente les processeurs ;
- V_h est l'ensemble des arêtes du graphe et représente les liens physiques entre processeurs.

G_h est non orienté si les liens de communication entre processeurs sont bi-directionnels. Des poids peuvent être associés aux arcs dans le cas de réseaux hétérogènes, pour pondérer les coûts de communication. De même, des poids peuvent être associés aux places pour pondérer leur capacité de traitement.

Le programme est lui aussi modélisé par un graphe des modules non orienté $G_S = (P_S,L_S)$ où :

- P_S est l'ensemble des sommets et représente les processus ;
- L_S est l'ensemble des arêtes du graphe et représente les communications entre processus.

Des poids peuvent être associés aux arcs, pour pondérer les coûts des communications entre processus. De même, des poids peuvent être associé aux places pour pondérer les coûts d'exécution. Ainsi, pour [Shen & al. 85], la recherche d'un placement consiste en un homomorphisme faible(1) entre le graphe de modules et celui des processeurs.

Synthèse

Les différents algorithmes d'allocation statique sur architecture statique optimale sont résumés sur la Figure 59.

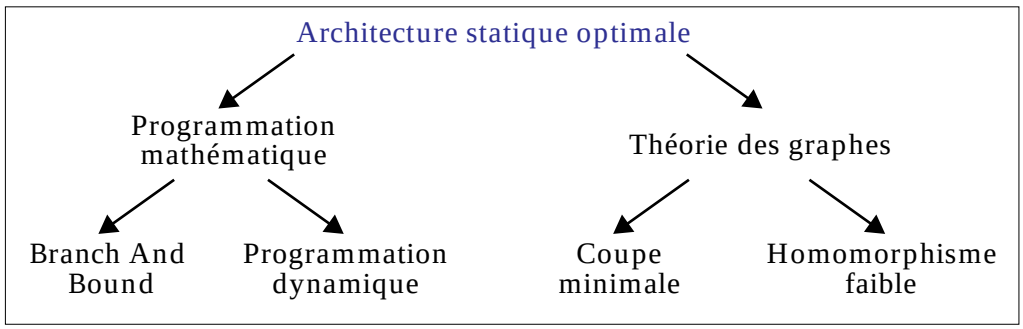


Figure 59 : Algorithmes d'allocation statique sur une architecture statique optimale

1.2.1.2. Allocation statique sur architectures statiques non optimales

Les architectures statiques non optimales utilisent la plupart du temps des stratégies basées sur l'utilisation d'heuristiques. Les heuristiques sont des critères, des principes ou des méthodes permettant de déterminer parmi plusieurs lignes de conduite, celle qui promet d'être la plus efficace pour atteindre un but. Les heuristiques permettent de réduire la dimension du problème (algorithmes polynomiaux) mais ne fournissent que des solutions approchées. Les deux types d'algorithmes les plus utilisés et les plus efficaces sont les algorithmes gloutons et les algorithmes itératifs.

(1) Il existe un homomorphisme faible entre un graphe de modules G1 et de processeurs G2, s'il existe une projection du graphe G1 sur le graphe G2 tels que deux modules voisins soient alloués sur deux processeurs voisins.

Algorithmes gloutons

Les algorithmes gloutons doivent être initialisés par une solution partielle, qu'ils cherchent ensuite à améliorer à chaque étape, jusqu'à l'obtention d'une solution. A chaque étape, le choix fait est définitif et on peut s'interdire de remettre en cause un choix précédemment établi. Cet algorithme est dépendant de l'ordre du choix des processus et des processeurs.

Algorithmes itératifs

Les algorithmes itératifs sont initialisés avec une solution complète qu'ils cherchent à améliorer à chaque itération. Le plus souvent, la solution de départ S_D, n'est autre qu'un placement aléatoire. La fonction d'avancement de l'algorithme, essayant ensuite des échanges de processus qui minimisent la fonction coût donnée. La complexité des algorithmes itératifs est en général plus grande que celle des algorithmes gloutons, mais donne de meilleurs résultats.

Parmi les algorithmes itératifs, on distingue :

- **le recuit simulé** [Bollinger & al. 88] basé sur une analogie avec la physique statistique. Partant d'un placement initial complet, on cherche à l'améliorer par un critère local. On accepte une modification de ce placement :
  - si le placement est amélioré dans le cas général ;
  - si le placement est plus mauvais, mais avec une probabilité qui décroît lors du déroulement de l'algorithme. L'idée est alors de se sortir d'un minimum local.

  Les placements générés sont bons en moyenne, mais les temps de calculs sont longs et les paramètres difficiles à ajuster pour un problème donné [Berman 89].
- **les algorithmes génétiques** [Whiley & al. 90] inspirés d'une analogie avec l'évolution des espèces [Holland 75]. A partir d'une population initiale on effectue des croisements pour engendrer de nouvelles configurations. On garde ensuite les meilleurs des populations résultantes pour effectuer d'autres croisements. Afin d'éviter les minima locaux, de temps à autre, les espèces sous soumises à des mutations. L'intérêt des algorithmes génétiques est qu'ils sont facilement parallélisables et qu'ils donnent de bons résultats dans le cas où le graphe de tâches n'est pas très différent de la topologie physique [Talbi 93].
- **les algorithmes tabou** [Glover & al. 92], de conception plus récente, recherchent à partir d'une configuration initiale, la meilleure solution que l'on puisse atteindre à l'aide d'un ensemble donné de transformations (en agissant sur les relations de voisinage). Afin d'éviter de revenir sur des solutions déjà explorées, on construit une liste tabou des derniers mouvements effectués et on interdit toute tentative de marche arrière. Les recherches tabou sont, en général, difficiles à mettre en oeuvre notamment car les relations de voisinage et la taille de la liste tabou dépendent du problème.

Synthèse

Les algorithmes de type gloutons et itératifs même s'ils donnent des résultats bons en moyenne, peuvent être très mauvais dans des cas particuliers (ils restent dans des espaces de solutions sub-optimaux) [Hémery 94]. Pour profiter des avantages des deux types d'algorithmes, il est intéressant d'utiliser des algorithmes itératifs, ayant comme solution initiale le résultat d'un algorithme glouton.

Les différents algorithmes d'allocation statique sur architecture statique non optimale sont résumés sur la Figure 60.

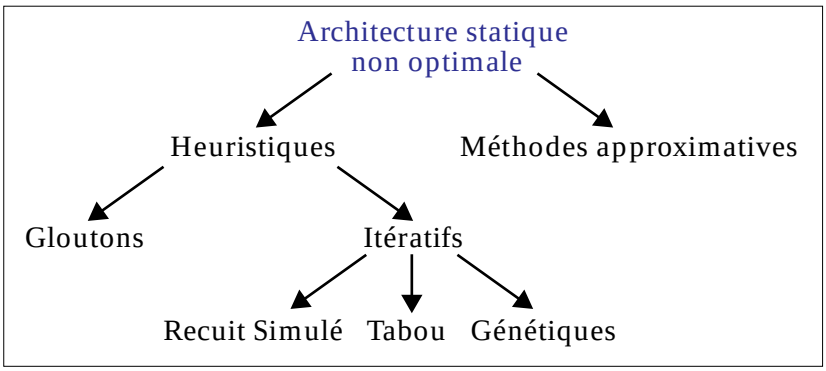


Figure 60 : Algorithmes d'allocation statique sur une architecture statique non optimale

1.2.1.3. Allocation statique sur architecture reconfigurable

Lorsqu'on cherche à réaliser le placement d'une application on se demande toujours comment adapter un programme à la topologie matérielle. On cherche à placer les processus sur les processeurs disponibles. Si on se place de plus dans le cas d'architecture matérielles reconfigurables, on peut se poser la question duale : comment adapter la structure de l'architecture matérielle à une application ?

Le problème étant NP-complet, seules des heuristiques dépendant du domaine sont utilisables. Ainsi, [Talbi 93] propose une heuristique qui à partir d'un graphe de processus composant le programme génère la configuration du réseau de processeurs. Le but étant de minimiser le temps d'exécution total du programme.

1.2.1.4. Conclusion sur les algorithmes d'allocation statique

La modélisation du problème de placement statique aboutit à des algorithmes d'une complexité qui est équivalente à celle de la classe des problèmes NP-complets et ceci dés que le nombre de noeuds (ou de tâches) est supérieur à trois. On recherche alors des heuristiques qui parcourent l'espace des solutions en retenant la meilleure solution de l'échantillon examiné. On privilégie ainsi l'obtention d'une bonne solution dans un temps de calcul raisonnable, à la meilleure solution possible.

L'approche la plus prudente pour obtenir ces bonnes solutions consiste à développer des boîtes à outils de méthodes de placement, réglables par l'utilisateur au cas par cas [Pellegrini 95]. D'autres approches, très prometteuses, visent à combiner des stratégies de recherche mixtes pour obtenir la remise en cause de certains choix pathogènes. L'utilisation de stratégies gloutonnes améliorées localement par des stratégies de séparation et d'évaluation en est un exemple [Bouvry & al. 93].

Les algorithmes d'allocation statiques supposent :

- la charge des machines reste inchangée durant l'exécution d'un programme ;
- l'architecture est fiable et que le taux de panne est quasi nul ;
- que les configurations et les topologies matérielles restent statiques ;
- les données qui servent au calcul des coûts de placement sont exactes et on connaît le nombre exact de processus, leur temps d'exécution et les coûts des communications qu'ils engendrent.

Lorsqu'on désire relâcher une ou plusieurs des contraintes exposées ci-dessus, on a recours à des algorithmes de placement dynamique. On a aussi recours à des algorithmes de placement dynamique, lorsqu'on doit gérer des requêtes multi-utilisateurs et multi-applications, avec des programmes à placer non seulement en fonction des critères précédemment définis, mais aussi en fonction de la charge courante des machines.

1.2.2. Algorithmes d'allocation dynamique

Les stratégies d'allocations dynamiques(2) sont utilisées lorsque l'allocation des tâches est régulée pendant l'exécution de l'application. Leur mise en oeuvre représente un coût qui s'ajoute au coût d'exécution de l'application. Les gains apportés en terme d'utilisation des ressources sont souvent bons, alors que ceux basés sur les temps de réponse peuvent décevoir. C'est pourquoi deux types de stratégies de régulation existent et tendent :

1) à éviter qu'un noeud (ou plus concrètement un processeur) ne soit inactif alors que des tâches restent en attente sur d'autres noeuds ou à lisser les périodes de congestion des machines. On parle alors de stratégies de **partage de charge** (*load sharing*) ;
2) à équilibrer le nombre de tâches par noeud sur l'ensemble du système. On parle alors de stratégies d'**équilibrage de charge** (*load balancing*).

Le processus de distribution dynamique des tâches sur les différents noeuds d'une architecture décentralisée est composée de deux étapes distinctes (cf. Figure 61) : une étape d'évaluation de l'état du système et une étape de prise de décision.

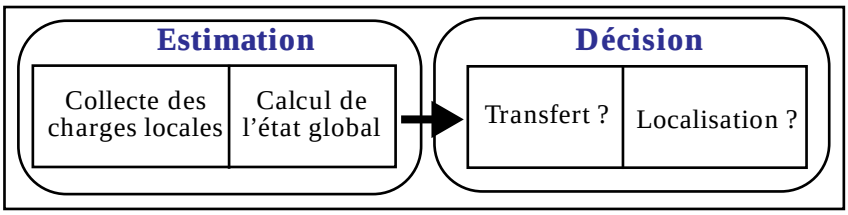


Figure 61 : Régulation dynamique de la charge

L'étape d'évaluation consiste à définir la charge locale d'un noeud et à disséminer cette information pour établir un état global ou partiel du système. La deuxième étape consiste à décider de la fréquence d'utilisation de l'algorithme de placement dynamique, des processus à transférer et le(s) noeud(s) destinataire(s).

Lorsqu'on dispose d'une estimation de l'état du système et que celui-ci laisse apparaître un problème de charge, le processus de décision intervient. Globalement deux décisions doivent être prises :

1) une décision de transfert si on a mis en évidence une mauvaise distribution de charge.
2) une décision de localisation pour choisir où placer les tâches qui doivent être transférées.

L'adaptabilité requiert que les algorithmes construits soient indépendants de la taille du système et de la topologie. Il est aussi souhaitable que l'algorithme soit peu sensible aux caractéristiques physiques et de topologie du réseau.

Dans la suite de cette section nous présentons brièvement les stratégies de calcul des indicateurs de charge, de transfert des informations de charge, des politiques d'information et de localisation des noeuds utilisables.

Quantification de la charge d'un noeud

Les algorithmes de placement dynamique sont basés sur une évaluation périodique et sur la comparaison de la charge des noeuds. La quantification de la charge locale d'un noeud se fait en général par le calcul d'une valeur de charge ou de seuils utilisant des états de charge du noeud. La comparaison de la charge des noeuds est plus facilement utilisable lorsque les noeuds sont hétérogènes, car elle dispose d'une métrique simple indiquant uniquement si un noeud est faiblement, moyennement ou fortement chargé [Kremien &

(2) L'étude présentée dans cette section est basée en partie sur les travaux de [Bernard & al. 96], [Bouvry & al. 93], [Folliot 96b], [Hémery 94] et [Talbi 93].

al. 93]. La charge étant en général calculée en divisant le nombre de processus actifs sur une machine par la puissance de la machine.

Un bon indicateur de charge doit refléter la charge de travail actuelle, estimer la charge dans un futur proche, gommer les fluctuations de courte durée et prendre en compte les spécificités d'un noeud [Bernard & al. 91]. La mesure d'un indicateur de charge est le plus souvent une mesure système de l'ensemble des tâches actives et en attente sur un noeud. Néanmoins, entre le moment où se fait la mesure et le moment de la décision du placement, la valeur mesurée peut fortement varier. C'est pour pallier ces problèmes qu'on utilise des valeurs moyennes dans le calcul de l'indicateur de charge (par exemple la longueur moyenne des files d'attente d'accès au processeur sur un intervalle de temps donné). L'indicateur le plus courant est celui de la somme des longueurs des files d'attentes d'accès CPU et d'entrée-sortie, échantillonnées toutes les quelques secondes et lissée [Ferrari & al. 87].

Parmi les autres informations pouvant concerner le calcul d'un indicateur de charge on trouve entre autres :

- le nombre de messages en attente de traitement sur un noeud : lorsque l'activité d'une tâche est mesurée par le nombre de messages qu'elle doit traiter (cas du serveur dans le sens client-serveur du terme) ;
- l'âge des tâches : plus une tâche est active depuis longtemps et plus elle a de chance de le rester encore longtemps ;
- le nombre de tâches en attente de traitement (dans le cas de gestion multi-application) ;
- le taux d'occupation mémoire ;
- le nombre d'utilisateurs connectés sur la machine (dans le cas de système multi-utilisateur) ;
- le type de la tâche : on dissocie alors les tâches qui font du calcul intensif (accès au processeur prédominant, ce qui induit une forte charge du noeud) de celles qui font des communications (et qui sont souvent en attente de réponse).

Les études dans ce domaine sont peu développées et le dosage idéal entre tous ces indicateurs pour une application donnée est encore du domaine de la recherche.

Politique de transfert de la charge

La politique de transfert détermine l'éligibilité des processus pour le placement ou la migration. Elle dépend des politiques de partage de charge ou d'équilibrage de charge et est basée sur des valeurs de seuil. Ainsi, lorsqu'une charge locale dépasse un certain seuil, la machine est dite surchargée et on cherche à transférer une partie de sa charge d'exécution. Ce transfert se fait soit au démarrage du processus (cas du placement), soit en faisant appel à la politique de localisation durant l'exécution du processus (cas de la migration).

Des études [Eager & al. 86 & Zhou & al. 87] ont montré que la valeur optimale du seuil dépend de la charge moyenne du système. De plus, dans certains cas, la stabilité de l'algorithme à base de seuil est compromise et entraîne un effet domino. L'effet domino se produit lorsqu'une machine surchargée se libère au dépens d'une autre machine disponible, qui devient elle même surchargée [Bernard & al. 91]. Une solution simple consiste à placer un processus sur une machine à la condition que cela n'entraîne pas son passage dans l'état surchargé.

Enfin, les techniques de placement et de migration coûtant cher, il est primordial de filtrer les processus candidats au placement. Ce filtrage se fait soit manuellement (par type d'utilisateur, par type de commandes, par nom ou par âge), soit en utilisant la notion de forces d'attraction et de répulsion qui existent entre les processus [Tokoro 90, Raverdy 96].

Politique d'information et de localisation

Lorsqu'il s'agit de prendre une décision concernant la localisation d'une tâche, il est nécessaire de disposer de l'état de charge des noeuds potentiellement concernés. La collecte d'informations de charge est un processus non négligeable aussi bien en coût de communication (nombre de messages échangés), que dans le degré de confiance que l'on pourra avoir dans l'estimation (les informations collectées sont-elles encore pertinentes au moment de la décision ?). **La politique d'information** qui spécifie la nature et la quantité des informations utilisées pour la prise de décision, ainsi que la manière dont ces informations sont réparties est donc à choisir précautionneusement. Une fois les informations de charge disponibles, il reste à choisir **une politique de localisation**, qui va déterminer un site d'accueil approprié pour chaque processus éligible.

De très nombreuses stratégies ont été proposées pour définir la quantité d'informations utilisées par les algorithmes, pour centraliser ou distribuer la collecte de ces informations et pour centraliser ou non la prise de décision.

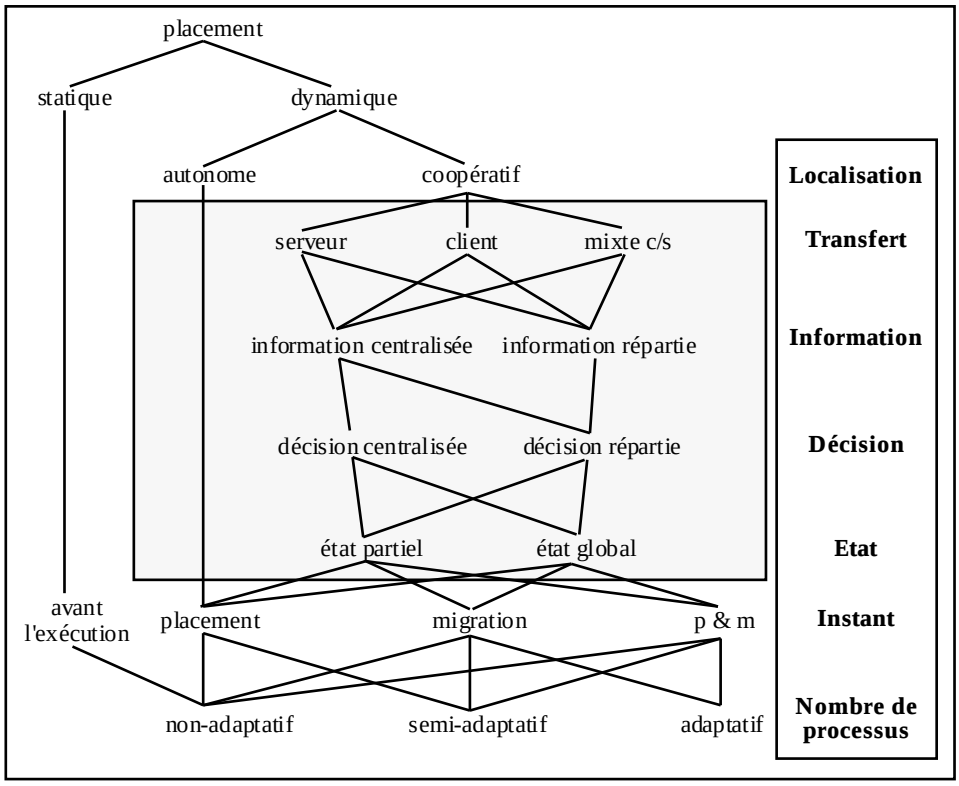


Figure 62 : Classification des algorithmes de placement dynamique

Ces stratégies et les algorithmes de placement de processus sont résumées dans la classification présentée à la Figure 62 [Bernard & al. 96]. La localisation est autonome (ou aveugle) lorsque le choix du site d'exécution est effectué sans aucune information sur l'état de la machine choisie pour le placement. La localisation coopérative est réalisée en suivant une approche serveur (qui lance des appels d'offre lorsqu'il doit faire un placement), une approche client (qui effectue des demandes de ressources) ou une approche mixte tenant compte de la charge globale du système. L'information sur la charge des machines est soit centralisée, soit répartie sur l'ensembles des sites. Dans le même ordre d 'idées, la décision de placement est soit centralisée, soit répartie. Finalement, la décision de placement est prise à partir d'un état partiel ou global du système.

Sur la partie inférieure de la Figure 62 (hors de la zone grisée) est proposée une extension de la classification des algorithmes de placement d'applications parallèles en fonction de leur dynamicité. L'instant de décision est déterminé au lancement de l'application

dans le cas du placement (comme par exemple dans UTOPIA [Zhou & al. 93]), lors de l'exécution dans le cas de la migration (comme par exemple dans MOSIX [Barak & al. 96]) ou les deux à la fois (comme dans Gatostar [Folliot & al. 95]). Le nombre de processus de l'application peut changer en fonction de l'évolution de l'état global du système (algorithme adaptatif), être choisi au démarrage de l'application (algorithme semi-adaptatif) ou bien être fixé avant l'exécution du programme (non adaptatif).

Quelques algorithmes

On définit un algorithme comme un chemin dans le graphe de la Figure 62. Les plus connus sont les suivants [Zhou 88, Theimer & al. 89 et Bernard & al. 96] :

- CENTRAL : centralise les informations de charge et les décisions
  (mixte c/s - information centralisée - décision centralisée - état global)
- DEMANDE : les machines surchargées demandent de l'aide
  (client - information répartie - décision répartie - état global)
- OFFRE : les machines sous-chargées offrent leurs services
  (serveur - information répartie - décision répartie - état global ou état partiel)

En ce qui concerne les algorithmes adaptatifs, il existe de nombreux travaux dont l'étude approfondie est hors du cadre de cette thèse. Néanmoins, nous énumérons dans la suite quelques exemples de grandes classes de solutions.

Ainsi, les algorithmes adaptatifs s'appuient sur :

- une combinaison des algorithmes OFFRE et DEMANDE en fonction de la charge du système (algorithme mis en oeuvre dans Gatostar [Folliot & al. 95]) ;
- des algorithmes utilisant des domaines [Zhou & al. 93 et Raverdy 96]. La notion de domaine, permet de regrouper un ensemble de machines utilisées de façon temporaire et dynamique. Le domaine est particulièrement efficace pour gérer l'adaptabilité de l'algorithme d'équilibrage de charge. La répartition de la charge peut alors être faite domaine par domaine, sans se préoccuper des autres domaines ;
- des algorithmes d'enchères (*bidding*) [Smith 88]. A la création d'un processus, une requête d'enchère pour l'exécution du processus est diffusé à tous les processeurs. A la réception de cette requête, chaque processeur envoie son coût d'exécution. Quand tous les coûts sont reçus, le processeur proposant le moindre coût est choisi. La méthode des enchères peut donner de bons résultats dans le cas de processus où le coût d'exécution et la politique des localisation sont compensés part le coût d'exécution du processus [Ross & al. 90] ;
- des algorithmes basés sur un modèle micro-économique régis par la loi de l'offre et de la demande [Ferguson & al. 88]. Dans cette méthode, les tâches sont considérées comme des consommatrices de processeurs et les noeuds proposent les prix pour la consommation de leurs processeurs. Comme les noeuds auront connaissance des prix des noeuds voisins, la loi de l'offre et de la demande va entraîner une régulation de la consommation des processeurs par les processus ;
- un système expert qui est considéré comme un système spécialisé qui utilise une base de connaissance et des techniques d'inférence et de raisonnement pour prendre des décisions. Dans un système réparti, chaque noeud possède son propre système expert qui prend des décision d'allocation des processus créés sur ce noeud [Jun & al. 90 et Brehm & al. 95].

Conclusion

Un algorithme d'allocation dynamique de processus est composé de deux entités de base : une entité d'information et une entité de contrôle. Ces entités sont utilisées pour calculer à un instant donné et pour disséminer des informations sur la charge des noeuds du système.

La distribution de la charge est basée sur des politiques de transfert, d'information et de localisation qui ne sont pas orthogonales, de sorte que le choix de l'une d'entre elles influe sur les deux autres. Il est alors très difficile de réaliser une comparaison objective des algorithmes de placement dynamique. Néanmoins, il se dégage à la suite de cette étude un certain nombre de points :

- même les algorithmes les moins sophistiqués améliorent les temps de réponse moyens des processus ;
- le gain est d'autant plus important que la charge globale est déséquilibrée et de forte moyenne ;
- les meilleurs résultats sont obtenus avec les algorithmes utilisant une information globale récente ;
- les algorithmes basés sur la diffusion sont peu extensibles ;
- les algorithmes ayant une information et une décision répartie sont plus tolérants aux fautes des machines ;
- les stratégies en aveugle sont intéressantes lorsque les tâches sont courtes et que le système est surchargé.

De plus, l'évaluation des algorithmes proposés se fonde sur des simulations ou des techniques analytiques et rarement sur des systèmes réels. Notons enfin, que la tolérance aux fautes et que l'équilibrage de charge peuvent aussi passer par la modification des couches systèmes comme dans DAWGS [Clark & al. 92] ou par des mécanismes purement matériels (grappes de disques RAID, machines serveurs en miroirs, ferme de processeurs symétriques, etc.).

1.3. Du placement dynamique vers le placement hybride

La Figure 63 est une tentative de représentation de la problématique globale du placement dynamique [Folliot 96a].

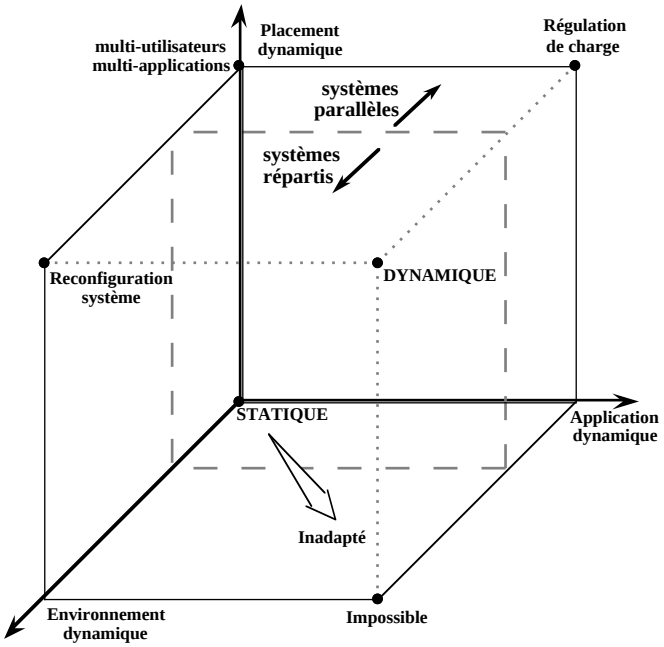


Figure 63 : Tentative de représentation de la problématique globale du placement dynamique.

Les trois axes utilisés pour la construction du cube sont :

- **l'axe application dynamique** représente le degré d'évolution de l'application au

cours de son exécution. Si le nombre de processus est fixe et n'évolue pas, l'application est statique. Elle est d'autant plus dynamique que le nombre de processus et la consommation des ressources est susceptible d'évoluer.

- **l'axe environnement dynamique** représente le degré d'évolution de la plate-forme système support de l'exécution. Les dynamicité des paramètres tels que le nombre et la charge des processeurs, ainsi que la charge des réseaux de communication sont alors pris en compte.
- **l'axe placement dynamique** indique les capacités de gestion des requêtes multi-utilisateur et multi-application.

A l'intersection du repère définit par les trois axes, on trouve le placement statique d'applications statiques en environnement statique. A l'opposé sur le cube, on trouve un environnement totalement dynamique offrant la gestion multi-application et multi-utilisateur, la régulation de charge et la (re)configuration de l'environnement.

On remarque alors les faits suivants [Folliot 96b] :

- le placement statique devient inadapté (voire impossible) dés que la dynamicité de l'application ou de l'environnement deviennent important (cf. le plan créé par les axes application dynamique et environnement dynamique).
- le placement dynamique d'une application statique en environnement statique ne présente aucune intérêt, puisque tous les paramètres nécessaires au placement sont connus à l'avance.
- le placement dynamique d'une application dynamique en environnement statique concerne le domaine de la régulation de charge (placement dynamique de graphes de tâches ou traitement de données irrégulières).
- le placement dynamique d'une application statique en environnement dynamique représente la configuration et la reconfiguration du système. Si les applications y sont connues à l'avance, il n'en est pas de même ni des noeuds d'exécution et ni de la charge globale du système.

Sur la Figure 63, on remarque aussi que le plan qui partage le cube en deux parties (en traits pointillés sur le cube) sépare les systèmes parallèles (qui ont un environnement plutôt statique), des systèmes répartis (qui ont un environnement plutôt dynamique). Cette séparation semble logique si l'on prend en compte que les paramètres de placement dynamique sur ces deux types d'architecture sont foncièrement différents. A la lecture du Tableau 39, qui donne d'ailleurs à ce sujet quelques points de comparaison, on constate à quel point la complémentarité de ces deux types de systèmes est forte.

Tableau 39: Comparaison du placement dynamique sur des systèmes parallèles et répartis.

	Système parallèle	**Système réparti**
Granularité de tâches	Nombreuses tâches courtes	Quelques tâches à grain moyen
Durée de vie des tâches	Très courte	Importante par rapport à une communication ou une opération
Topologie	Influence importante sur les calculs et les distances entre noeuds	Quelconque, pas de distance entre les noeuds et donc pas de voisins.
Placement des données	Prépondérant - La localité des données est un paramètre important (placement dynamique des données)	Utilisation de mémoire virtuelle répartie et de gestionnaire de fichiers réparti
Placement des tâches	Statique	Prépondérant (placement statique et / ou dynamique)

Tableau 39: Comparaison du placement dynamique sur des systèmes parallèles et répartis.

	Système parallèle	**Système réparti**
Communication	Faible distance et haut débit	Moyenne et longue distance et débit variable
Diffusion	Coût généralement élevé (surtout pour la diffusion globale) mais fonction de la topologie	Coût lié au nombre de machines, à la bande passante disponible sur le réseau et au protocole utilisé
Migration	Généralement pas envisageable	Envisageable à faible coût par rapport au temps d'exécution
Gestion	Mono-utilisateur / mono-application pour offrir le maximum de puissance des processeurs homogènes durant un minimum de temps	Multi-utilisateur / multi-application pour offrir un maximum de puissance partagée sur des processeurs pas toujours homogènes

De cette remarque est née notre volonté d'unifier le placement et la gestion de ces deux types d'architecture logicielle dans MEDEVER. Il nous adonc fallu prendre en compte des techniques de placement hybride Le placement hybride étant un placement dynamique réalisé à la fois sur des réseaux de stations de travail et des machines parallèles. Ce placement peut être réalisé sur des architectures locales ou de plus grande échelle. Ainsi, les multi-réseaux sont des architectures matérielles composées de réseaux locaux qui établissent des liens privilégiées et partagent tout ou une partie de leurs ressources.

1.3.1. Prise en compte des coûts de communication

Ces architectures matérielles hybrides ont comme spécificité majeure d'offrir des communications par passage de messages qui varient énormément à la fois en terme de vitesse et de fiabilité des transmissions. C'est pourquoi, elles ont donné naissance à un nouveau modèle d'architecture matérielle, appelé NUMP (*Non Uniform Message Passing*) [Yang 92]. L'étude des différences de vitesse de transmission dans ces architectures (cf. Tableau 40) fait ressortir deux niveaux de granularité des communications.

Le niveau gros grain considère les communications entre processus qui s'exécutent sur des sites distants que ce soit pour des tâches séquentielles ou pour des groupes de tâches parallèles. Le niveau grain fin, par contre, prend en compte les communications entre processus qui s'exécutent sur des machines parallèles. Ceci revient alors à dire que la granularité des communications est fine sur des machines parallèles à mémoire partagée (pour une étude sur les bibliothèques de messages se reporter à [McBryan 94]), moyenne sur des machines à passage de message et forte (voire très forte) pour l'ensemble des sites d'un réseau hybride. **Dans un langage comme VODEL-D, l'entité canal est utilisé pour caractériser de manière dynamique les évolutions des communications entre composants logiciels.** Les architectures NUMP sont donc gérables par notre langages VODEL-D. Les algorithmes de placement sont par contre eux, complexes et difficiles à mettre en oeuvre, comme nous allons le voir dans la section suivante.

Tableau 40: Classement des différents moyens de communication sur une architecture NUMP

Classement (du plus rapide au plus lent)
1 - Les communications par mémoire partagée
2 - Les communications des machines multiprocesseurs à passage de messages
3 - Les communications dans les réseaux locaux
4 - Les communications dans les multi-réseaux.

Pour décrire les liens de communication entre les différents éléments de l'architecture matérielle, on utilise HADEL. Les H-Machines décrivent les stations de travail mono ou multiprocesseurs et les H-Liens représentent les liens physiques entre les H-Machines. Ces objets sont utilisés pour construire un graphe pondéré où les H-Machines sont les noeuds et les H-Liens les arcs. **La pondération des arcs de ce graphe est un problème difficile, car il doit modéliser les liens de communications entre machines**. Or on ne connaît pas aisément et de manière dynamique les différences de bande passante et de vitesse des entrées-sorties entre deux machines. Ce graphe étant par la suite utilisé pour calculer le placement, il est donc de la plus grande importance de ne pas attribuer des pondérations aberrantes.

1.3.2. Mise en oeuvre du placement hybride

Le placement hybride est par nature hiérarchique, car il s'applique d'abord au choix d'une machine parmi N, puis sur cette machine au choix d'un ou d'un groupe de processeurs (s'il en existe plus d'un) parmi M. Cette hiérarchie au niveau des processeurs entraîne au niveau système la redéfinition globale :

- des techniques de contrôle d'accès aux machines ;
- de relation de confiance entre les réseaux (on cherche à savoir qui accède aux ressources du réseau local auquel on appartient et pour quoi faire) ;
- d'adressage et de routage (maintenant sur deux niveaux) ;
- de gestion des communications inter-processus (pour la gestion des algorithmes de placement hybride).

Le choix d'une stratégie de placement sur un réseau hybride est un problème délicat à résoudre car il implique la fusion de techniques de placement dynamique avec migration de processus sur de réseaux de stations de travail et des techniques de placement statique sur des machines multiprocesseurs.

Il existe néanmoins déjà des outils de placement hybride. Ainsi, Stardust [Cabillic & al. 96] est un des premiers exemples d'environnement d'exécution offrant un placement hybride d'applications sur des réseaux de machines hybrides hétérogènes. De plus, il existe également des systèmes opérationnels de gestion des multi-réseaux, tels que Condor (étendu avec les «flock») [Evers & al. 93], Globe [Van Steen & al. 95], Legion [Grimshaw & al. 94] et Utopia multi-cluster [Zhou & al. 93]. Enfin, l'utilisation de bibliothèques de communication portables permet l'intégration de techniques de placement hybride statique sur des outils d'équilibrage de charge déjà existants comme Gatostar [Bretelle & al. 95] ou directement à l'intérieur des applications avec PVM et MPI (cf. Annexe E pour un état de l'art sur les bibliothèques de communications hétérogènes).

En ce qui concerne notre approche, il n'est actuellement pas prévu dans notre langage HADEL de gérer des architectures matérielles ayant plus de deux niveaux de hiérarchie. Ceci étant dû à l'hypothèse que nous avons fait sur les architectures matérielles cibles gérées dans MEDEVER, à savoir les architectures matérielles hybrides locales. Le placement hybride impose enfin, de gérer les caractéristiques des réseaux utilisés de manière plus fine que celle proposé par HADEL.

1.4. Synthèse

Le placement d'une application ne se résume pas à l'utilisation d'un des deux modèles de placement étudiés précédemment (le placement statique ou dynamique). Le placement consiste dans un premier temps à connaître l'entité à placer, puis à connaître l'architecture de la machine cible et à définir quel est l'objectif réel du placement. Répondre à ce dernier point implique la prise en compte des directives de l'utilisateur, du découpage de l'application, de l'analyse du source de l'application, du chargement et du placement dynamique de l'application [Hémery 94].

Il se pose alors le problème de l'intégration des différentes informations obtenues à chaque niveau et la gestion des contradictions éventuelles. Ceci est d'autant plus vrai si l'on se place dans le cadre du placement hybride, où les différents composants de l'application peuvent avoir des logiques applicatives et des contraintes d'exécution totalement différentes.

Dans notre méthode MEDEVER, nous prenons en compte ces problèmes. Nous distinguons d'ailleurs deux types d'équilibrage, l'équilibrage de charge et d'application. Même s'ils s'appuient sur des couches systèmes communes, ils diffèrent dans les contraintes et les buts à atteindre. L'équilibrage de charge a pour but d'offrir les machines qui remplissent au mieux les besoins de l'application, alors que l'équilibrage d'application est centrée sur la validation et l'optimisation de l'architecture logicielle d'une application.

Notre méthode a aussi été conçue pour prendre en compte le placement hybride et ses contraintes supplémentaires, notamment en terme de gestion de hiérarchie de machines (ie. de processeurs) et de gestion de la granularité des composants de l'application devant s'adapter au mieux à cette hiérarchie. C'est pourquoi notre méthode s'appuie sur l'utilisation des langages HADEL, VODEL et VODEL-D (cf. figure 64).

La description d'une architecture hybride est réalisée avec le langage HADEL, qui combinée avec la description statique de l'architecture logicielle, permet le calcul du placement statique d'une application. Les algorithmes de placement statique dépendent du type d'équilibrage choisi. Une fois le placement statique défini, il est possible de suivre l'évolution dynamique à la fois de l'architecture matérielle et logicielle. Les changements dans l'architecture matérielle seront notifiés avec le langage HADEL (ou avec les entités passives physiques de VODEL-D), alors que les modifications sur l'architecture logicielle s'appuient sur les spécificités du langage VODEL-D. Ainsi, dans VODEL-D, chaque évolution est associée à un changement de plan d'exécution. L'utilisation de VODEL-D est nécessaire pour le recalcul à la volée du placement dynamique, qui ne doit pas toujours être réalisé en fonction de critère de charge. L'application conserve alors dans ce dernier cas un droit de regard sur le placement qu'on lui propose

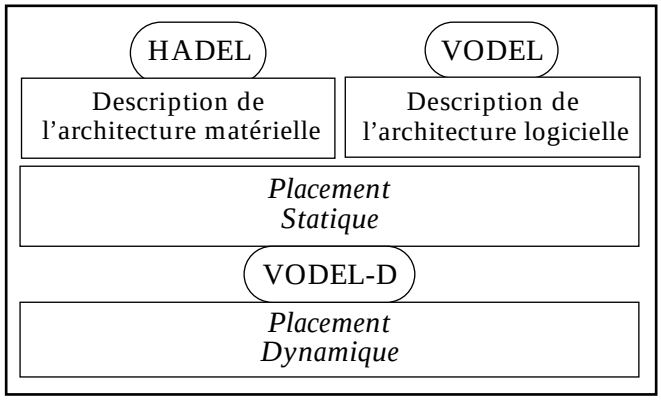


Figure 64 : L'équilibrage de charge et MEDEVER

Le Tableau 41 présente les actions réalisables en fonction des descriptions disponibles. L'intersection de la ligne et de la colonne de même nom indique le type de description réalisé.

Tableau 41: Actions réalisables en fonction des descriptions architecturales

	HADEL	**VODEL**	**VODEL-D**
HADEL	Description, administration et simulation d'architectures matérielles	Equilibrage de charge sur architecture matérielle choisie par l'utilisateur	Equilibrage de charge d'application dynamique

Tableau 41: Actions réalisables en fonction des descriptions architecturales

	HADEL	**VODEL**	**VODEL-D**
VODEL	Equilibrage de charge sur architecture matérielle choisie par l'utilisateur	Description statique de l'architecture logicielle	Comparaison entre l'architecture logicielle de définition et d'exécution
VODEL-D	Equilibrage de charge et d'application dynamique	Comparaison entre l'architecture logicielle de définition et d'exécution	Description dynamique de l'architecture logicielle d'exécution

La section suivante présente IDEFIX la plate-forme de placement hybride que nous avons conçue.

2. Equilibrage de charge et placement hybride dans IDEFIX

L'environnement IDEFIX (*Integrated Development Environment with Flexible dIstributed eXecution*) a été créé dans le but de placer dynamiquement une application répartie ou parallèle sur un environnement matériel hybride. C'est pourquoi, nous avons repris les axes de la Figure 63, pour concevoir et mettre en oeuvre IDEFIX. En effet, nous pensons que l'unification du placement dynamique en environnement parallèle et réparti implique l'utilisation de méthodes et de langages adaptés à la description des architectures matérielles et logicielles sur toute la surface du cube définie par les trois axes de la Figure 63.

Nos choix en ce qui concerne ces langages et environnements pour la mise en oeuvre d'IDEFIX sont les suivants (cf. Figure 65) :

- **l'axe application dynamique** représentant le degré d'évolution de l'application au cours de son exécution, sa représentation est réalisable au sein du langage VODEL dans le cas d'une application statique. La gestion de la dynamicité passe par des extensions du langage VODEL, car la granularité de l'application et la structure de son architecture sont susceptibles d'évoluer. Ces extensions sont disponibles dans VODEL-D ;
- **l'axe environnement dynamique** représente le degré d'évolution de la plate-forme système hybride support de l'exécution. Or HADEL a été créé dans ce but et supporte nativement, de part ses deux niveaux de hiérarchie, la description d'architectures matérielles hybrides.
- **l'axe placement dynamique** indique les capacités de gestion des requêtes multi-utilisateur et multi-application. Par exemple, Gatostar réalise toutes ses actions et offre des services supplémentaires tels que le placement multi-critère et la tolérance aux fautes.

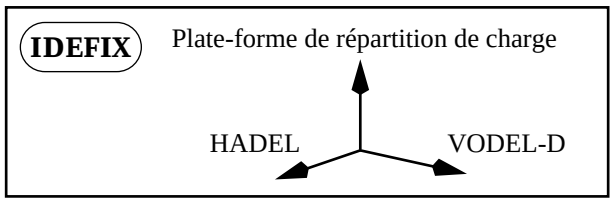


Figure 65 : IDEFIX = HADEL, VODEL-D et une plate-forme d'équilibrage de charge

Ainsi, les langages HADEL et VODEL sont utilisés dans IDEFIX pour décrire respectivement les architectures matérielles hybrides et les architectures logicielles. Ils servent aussi à calculer le placement initial de manière statique. Cette approche est justifiée si

l'on considère que les applications ont une durée de vie importante et qu'une erreur initiale de placement coûterait très cher par la suite (avant d'atteindre un équilibre pour la charge). Ils fournissent ensuite à l'environnement d'équilibrage de charge les informations nécessaires à l'évolution dynamique de l'application (graphe d'exécution dynamique de VODEL-D) sur une architecture matérielle elle aussi dynamique (machine parallèle virtuelle issue de la description d'HADEL). Le calcul de la granularité adaptée pour chaque composant en respectant les spécifications de l'application et les évolutions du placement des entités logicielles, entraînent une caractérisation dynamique de la granularité.

L'exécution répartie peut se faire sur des machines à charge variable ou nulle. En cas de surcharge, la migration de processus peut-être demandée. La décision de migration est en général prise au niveau système par le répartiteur de charge. Nous présentons dans un premier temps l'architecture générale des plate-formes d'équilibrage de charge. Puis, nous décrivons l'intégration du langage HADEL dans IDEFIX. Enfin, nous montrons comment la gestion des applications statiques et dynamiques peut être réalisée à partir du langage VODEL-D.

2.1. Gestion de l'équilibrage de charge

L'architecture d'un système d'équilibrage de charge est composée de deux types de gestionnaire : ceux présents obligatoirement sur toutes les machines et ceux qui sont soit centralisés, soit répartis (cf. Figure 66).

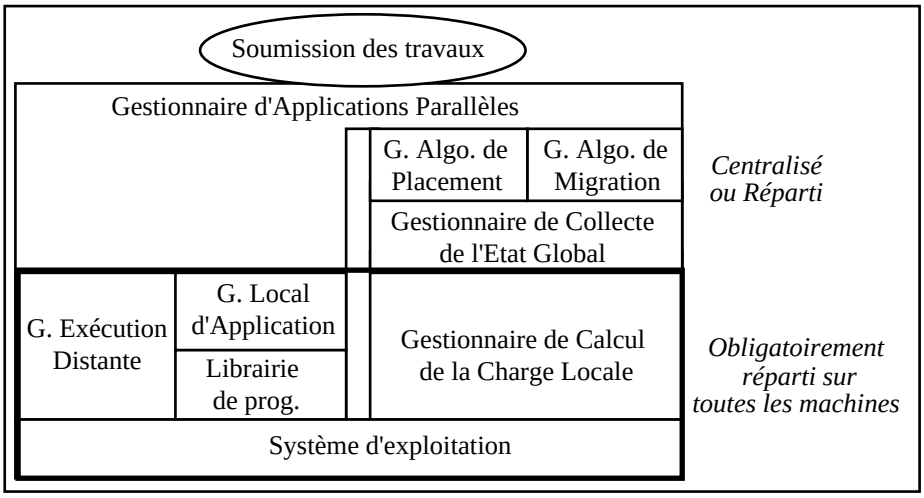


Figure 66 : Architecture de la gestion système du placement dynamique

Les gestionnaires centralisés et présents sur chaque machine sont [Folliot 96b] :

- **le gestionnaire d'exécution à distance** est chargé de recevoir les requêtes d'exécution, de lancer les processus avec les droits appropriés, de gérer la fin des processus et de retourner le résultat de l'exécution au gestionnaire demandeur ;
- **le gestionnaire local d'application** est chargé de faire le lien entre l'exécution locale des processus d'une application (à travers la bibliothèque de programmation fournie par le système de placement) et les gestionnaires d'applications distants concernés ;
- **le gestionnaire de calcul de la charge locale** qui surveille et mémorise l'état local de la machine. Il fournit les informations dont il dispose au gestionnaire de collecte de l'état global.

Les autres gestionnaires, dont la mise en oeuvre est centralisée ou répartie, sont les suivants :

- **le gestionnaire de collecte de l'état global** récolte les informations réparties avec le concours des gestionnaires de calcul de la charge locale. En général, il a aussi la charge de stocker les caractéristiques statiques de l'environnement (nom et adresse des machines et leurs paramètres de puissance et de disponibilité) ;
- **le gestionnaire d'algorithmes de placement** met en oeuvre les politiques de placement disponibles à la demande du gestionnaire d'applications parallèles. Pour cela, il se base sur les informations fournies par le gestionnaire de collecte de l'état global (qui d'ailleurs peut aussi jouer le rôle d'ordonnanceur). Les requêtes de décision de placement sont ensuite envoyées aux gestionnaires d'exécutions distantes concernées ;
- **le gestionnaire d'algorithmes de migration** réalise des politiques adaptatives en collaboration ou non avec le gestionnaire d'algorithmes de placement. En général, il n'est pas appelé explicitement, mais implicitement en fonction des informations fournies par le gestionnaire de collecte de l'état global. Les requêtes de migration sont envoyées aux gestionnaires locaux d'application des machines de départ et de destination ;
- **le gestionnaire d'applications parallèles** reçoit les demandes de soumission de travaux, stocke les informations sur l'état de l'application et fait appel au gestionnaire de placement pour leur donner une affectation. Ce gestionnaire collabore avec les gestionnaires locaux d'application via la bibliothèque de programmation disponible et réalise les fonctions d'une machine parallèle virtuelle.

La mise en place d'une plate-forme d'équilibrage de charge consiste donc dans un premier temps à installer les gestionnaires de base sur chacune des machines participant à l'équilibrage de charge. Puis, dans un second temps, à sélectionner la ou les machines qui exécuteront les gestionnaires de collecte et d'algorithmes (en tenant compte de la charge CPU, la taille du disque, la vitesse des entrées/sorties et la position sur le réseau physique). Enfin, chaque gestionnaire doit s'exécuter de manière permanente (sous la forme d'un processus démon Unix par exemple) en limitant, si possible, le nombre de catégories de gestionnaire à un, quel que soit le nombre d'applications et d'utilisateurs.

2.2. Intégration de l'environnement dynamique

HADEL est notre langage de description hiérarchique d'architectures matérielles hybrides. La hiérarchie n'est que sur deux niveaux qui correspondent à deux niveaux de granularité :

- le niveau gros grain ou macro, est dédié à la représentation des machines de type station de travail et de leurs topologies d'interconnexion.
- le niveau grain fin ou micro, décrit de façon précise les architectures de machine de type multiprocesseurs.

La description d'une architecture hybride est réalisée au moyen des deux types d'objets :

- les H-machines décrivent les stations de travail mono ou multi-processeurs.
- les H-liens décrivent les lignes physiques de connexions.

La gestion des environnements dynamiques avec HADEL nous a conduits, comme dans VODEL, a redéfinir le mode de description et de vérification d'une architecture matérielle que nous présentons dans la section 1. La section 2 est constituée d'un bref état de l'art sur les jeux d'essai, indispensables à la caractérisation selon des critères uniques des puissances des machines présentes sur le réseau hybride. Ces paramètres sont d'une importance cruciale, car ils sont utilisés pour le placement dynamique.

2.2.1. Description dynamique d'une architecture matérielle

La description de l'architecture matérielle est décomposée en trois étapes (cf. Figure 67) :

1) **Description** : une architecture virtuelle (ou idéale) est construite par l'utilisateur qui réutilise (ou non) les modules déjà existants en bibliothèques. Nous qualifions alors cette architecture décrite de «virtuelle». L'architecture virtuelle est une architecture souhaitée ou imaginée et ne correspond pas obligatoirement à la réalité (machine non disponible ou ne respectant pas les contraintes de placement imposées par exemple).
2) **Validation** : cette architecture virtuelle est alors vérifiée. Les erreurs sont de deux types : syntaxique ou structurelles. Les erreurs syntaxiques sont corrigées par l'étude de la conformité de la description textuelle de l'architecture à la grammaire du langage. Les incohérences structurelles de l'architecture sont principalement basées sur des contraintes de topologie et de vérification de la validité des attributs.
3) **Correspondance** : si l'architecture virtuelle est validée, on cherche une correspondance entre les machines décrites et les machines réelles se rapprochant le plus possible de cette description. En cas de non correspondance, on peut éventuellement orienter l'utilisateur vers des techniques d'émulation d'architecture [Tron 94] ou de simulation (comme dans [Prylli 95], dans Proteus [Brewer & al. 91] ou dans QNAP2 [Simulog 88]).

La mise en oeuvre de ces trois étapes est réalisée par l'utilisation de bibliothèques d'architecture. Ces bibliothèques comprennent une base de données de matériels et une base de données de topologies. Elles sont utilisées lors de l'étape de validation de l'architecture, mais aussi lors de l'étape de correspondance en prenant en compte les contraintes de l'utilisateur (cf. Figure 67).

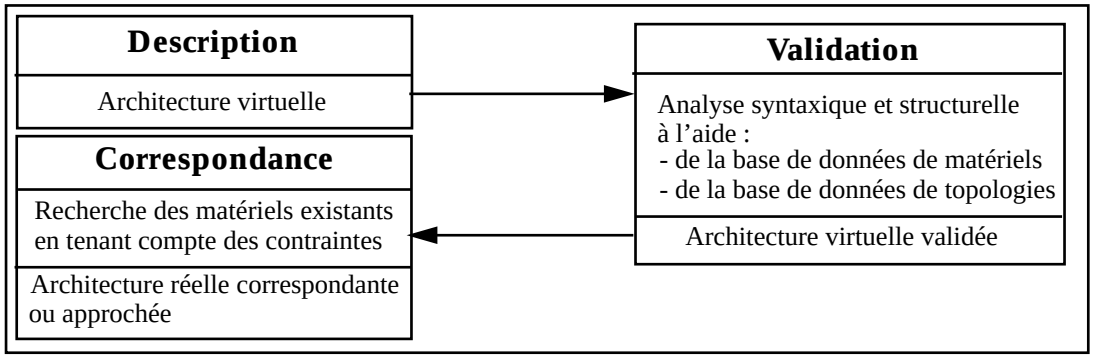


Figure 67 : Description d'une architecture matérielle avec HADEL

La réalisation de ces trois étapes entraîne au niveau technique la mise en place :

- d'un dialogue de type homme/machine entre le concepteur et les outils lors des phases de description, de validation et de correspondance ;
- d'un dialogue entre les outils logiciels qui coopèrent et s'échangent des messages concernant des objets et/ou des attributs des objets de description de l'architecture ;
- de bases de données de matériels et de topologie servant de référentiels de validation ;
- de passerelles vers des environnements tiers tels que des logiciels de simulation ou d'équilibrage de charge.

La qualité des informations disponibles dans la base de données de matériels et de topologies dépend des mécanismes de récolte et de mise à jour de ces informations. Parmi ces informations, la puissance des machines présentes sur le réseau est un point important à

ne pas négliger, surtout pour des machines parallèles spécifiques. Un moyen d'obtenir des mesures selon une métrique unique consiste à utiliser des jeux d'essai (cf. Annexe F pour un bref état de l'art sur les jeux d'essai). Dans HADEL, par défaut, nous considérons les puissances des machines en MIPS.

2.2.2. Intégration de HADEL dans IDEFIX

L'intégration de l'environnement dynamique dans IDEFIX consiste d'abord à étendre la plate-forme d'équilibrage de charge pour qu'elle gère, non plus une configuration de noeuds de calculs statiques, mais plusieurs configurations de manière dynamique. En effet, actuellement dans Gatostar, les caractéristiques statiques de l'environnement sont définies par l'administrateur réseau dans un fichier de configuration. Ce dernier contient les descriptions qualitatives (noms et types des machines…) et quantitatives (vitesses relative des processeurs, disques, mémoire centrale) des ressources de la machine parallèle virtuelle. L'administrateur est ainsi libre d'y inclure les machines qu'il désire comme support de la machine parallèle virtuelle qu'il construit.

Ensuite, la description de la machine virtuelle d'exécution doit pouvoir être spécifiée et modifiée par l'utilisateur, en fonction de ses besoins. Les machines étant sélectionnées soit directement par l'utilisateur, soit par des requêtes sur la base de données de matériels disponibles dans IDEFIX.

2.2.3. Synthèse

L'utilisateur qui désire décrire une architecture matérielle commence par en créer une virtuelle. Si cette architecture correspond à des éléments réels de l'architecture matérielle disponible, alors l'utilisateur récupère une description réelle des machines dont il a besoin. Sinon, le logiciel l'aiguille plutôt vers des techniques de simulation ou d'émulation qui sont intéressantes principalement sur des machines parallèles reconfigurables. Cette architecture est ensuite prise en compte par la plate-forme d'équilibrage de charge pour réaliser l'exécution dynamique de l'application.

2.3. Gestion des applications dynamiques : VODEL-D

En ce qui concerne le placement statique d'applications parallèles et réparties, VODEL intègre les fonctionnalités nécessaires à son intégration avec une plate-forme d'équilibrage de charge. En effet, la description statique d'une application est réalisée via un graphe d'entités logicielles (les DVSC) et de communication (les DVSL et DVGL). A partir de ces descriptions :

- soit on génère un fichier de configuration dans un langage interprété par la plate-forme d'équilibrage de charge ;
- soit on pilote directement la plate-forme d'équilibrage de charge, en utilisant ses API d'accès ou en modifiant son code source.

La deuxième méthode à l'avantage d'offrir la possibilité de modifier les algorithmes de placement en prenant en compte les nombreux attributs disponibles dans HADEL et VODEL-D. Nous détaillons l'utilisation de VODEL-D avec la plate-forme d'équilibrage Gatostar [Folliot 92] dans le chapitre 6.

3. Conclusion

L'équilibrage de charge sur réseaux hybrides passe par l'intégration des spécificités des systèmes répartis et parallèles. Cette intégration est réalisée selon les trois axes présentés sur la Figure 63 :

- l'axe application dynamique qui représente les évolutions dynamiques du contexte et des composants d'une application. Dans IDEFIX, cet axe est associé à un lan-

gage de description dynamique d'une architecture logicielle tel que VODEL-D ;

- l'axe environnement dynamique qui s'appuie désormais sur des architectures hybrides reliées entre elles par des liaisons hauts débits. Dans IDEFIX, cet axe est associé à un langage de description d'une architecture matérielle tel que HADEL ;
- l'axe placement dynamique qui évolue pour prendre en compte de nouvelles contraintes de placement sur des multi-réseaux ou des réseaux hybrides. Dans IDEFIX, cet axe est associé à une plate-forme d'équilibrage de charge tel que Gatostar.

De plus, nous avons remarqué que l'intégration de l'axe langage et de l'axe système facilite la gestion des applications dynamiques.

Notre plate-forme IDEFIX a été créée en prenant en compte notre méthode MEDEVER (cf. Figure 68). Son implémentation est présentée dans le chapitre suivant et permet à partir d'une description de l'architecture matérielle (décrite via le langage HADEL ou un fichier de configuration spécifique) et d'une description de l'architecture logicielle (décrite en VODEL-D), de réaliser de l'équilibrage dynamique de charge. VODEL-D est alors utilisé pour calculer le placement statique de l'application et pour générer des contraintes d'attraction et de répulsion entre les tâches. La granularité des éléments gérés est du niveau programme. Les stratégies d'exécution réparties et de placement dynamique sont soit gérées par la plate-forme d'équilibrage de charge (pour des questions de performance), soit gérées par un programme externe. La plate-forme d'équilibrage de charge gère les traces d'exécution et produit des statistiques sur l'utilisation des ressources et sur les durées d'exécution et de communication. Certaines plate-formes, telles que Gatostar, offrent même un système de mémorisation des exécutions qui permet d'optimiser les exécutions futures en fonction des exécutions passées (ce qui revient à modifier à posteriori la description de l'architecture logicielle ou matérielle).

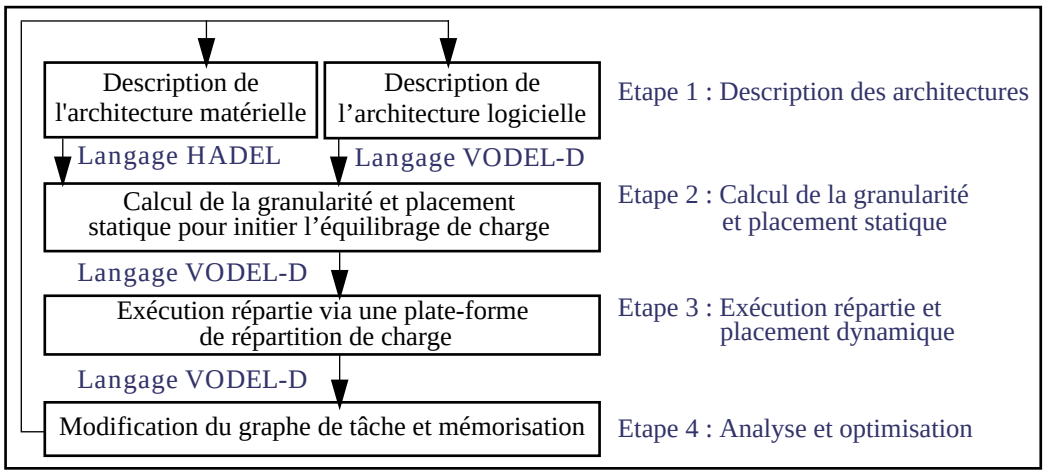


Figure 68 : La méthode MEDEVER appliquée à l'équilibrage de charge

4. Références bibliographiques

[Barak & al. 96] A. Barak, O. Laden & Y. Yarom, «The Now MOSIX and its Premptive Process Migration Scheme», http://www.cs.huji.ac.il/papers/distrib, 1996.

[Berman 89] F. Berman, «Experience with an automatic solution for the mapping problem», Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, 1989.

[Bernard & al. 91] G. Bernard, D. Stève & M. Simatic, «Placement et migration de processus dans les systèmes répartis faiblement couplés», Technique et Science Informatiques, Vol. 10 (5), pp. 375-392, 1991.

[Bernard & al. 96] G. Bernard & B. Folliot, «Caractéristiques générales du placement dynamique : synthèse et problématique», Actes de l'école thématique CNRS placement dynamique et répartition de charge, Presqu'île de Giens, pp. 3-22, Juillet 1996.

[Billionnet & al. 89] A. Billionnet, M.-C. Costa & A. Sutter, «Les problèmes de placement dans les systèmes distribués», Technique et Science Informatiques, Vol. 8 (4), pp. 307-337, 1989.

[Bokhari 81] S.H. Bokhari, «A shortest tree algorithm for optimal assignements across space and time in distributed processor system», IEEE transactions on Software Engineering, Vol. SE-7 (6), pp. 583-589, November 1981.

[Bollinger & al. 88] S.W. Bollinger & S.F. Midkiff, «Processor and link assignement in multicomputers using simulated annealing», Proceedings of the Int. Conf. on Parallel Processing, pp. 1-7, 1988.

[Bouvry & al. 93] P. Bouvry, D. trystram & J.M. Geib, «Répartition de charge», Algorithmes parallèles : analyse et conception, Chapitre 8, Hermès Eds., 1994.

[Brehm & al. 95] E.W. Brehm, R.T. Goettge & F.W. McCaleb, «START/ES: an expert system tool for system performance and reliability analysis», Performance Evaluation, No 22, pp. 43-58, 1995.

[Bretelle & al. 95] B. Bretelle, T. Terracol & B. Folliot, «Placement d'applications parallèles en environnement hybride», Université pierre et Marie Curie, Paris, Rapport de Recherche, No MASI 95-19, Juin 1995.

[Brewer & al. 91] E. A. Brewer, C. N. Dellarocas, A. Colbrook, & W. E. Weihl, «Proteus: A High PerformanceParallel Architecture Simulator», Technical Report MIT/LCS/TR-516, Massachusetts Institute of Technology, Laboratory of Computer Science, September 1991.

[Cabillic & al. 96] G. Cabilic & I. Puaut, «Répartition de charge dans Stardust : un environnement pour l'exécution d'applications parallèles en milieu hétérogène», Actes de l'école thématique CNRS placement dynamique et répartition de charge, Presqu'île de Giens, pp. 167-174, Juillet 1996.

[Chu & al. 90] W.W. Chu & J-S. Yur, «A branch and bound with underestimates algorithm for the task assignement problem with precedence constraint», 10th Int. Conf. on Distributed Computing Systems, Paris, France, pp. 449-501, May 1990.

[Clark & al. 92] H. Clark & B. McMillin, «DAWGS: A Distributed Compute Server Utilizing Idle Workstations», Journal of Parallel and Distributed Computing, Vol. 14, pp. 175-186, Feb. 1992.

[Eager & al. 86] D.L. Eager, E.D. Lazowska & J. Zahorjan, «Adaptive Load Sharing in Homogeneous Distributed Systems», IEEE Transactions on Software Engineering, Vol. 12 (5), pp. 662-675, May 1986.

[Evers & al. 93] X. Evers, J. De Jongh, R. Broontje & al., «Condor Flocking: Load Sharing Between Pools of Workstation», technical report DUT-TWI-93-104, Delft University of Technology, The nederlands, 1993.

[Ferguson & al. 88] D. Ferguson, Y. Yemini & C. Nikolaou, «Microeconomic Algorithms for Load Balancing in Distributed Computer Systems», 8th International Conference on Distributed Computing Systems, San Jose, California, pp. 491-499, June 1988.

[Ferrari & al. 87] D. Ferrari & S. Zhou, «An empirical investigation of load indices for load balancing applications», Proceedings of the Performances'87, Bruxelles, Belgique, pp. 515-528, Dec. 1987.

[Folliot 92] B. Folliot, «Méthodes et outils de partage de Charge pour la conception et la mise en oeuvre d'Applications dans les systèmes répartis hétérogènes», Thèse de l'Université Pierre et Marie Curie, 4 place Jussieu, 75252 Paris cedex 05, Rapport de Recherche 93-27, Décembre 1992.

[Folliot & al. 95] B. Folliot, P. sens & P.-G. Raverdy, «Plate-forme de Répartition de Charges et de Tolérance aux Fautes pour Applications Parallèles en Environnement Réparti», Calculateurs Parallèles, Vol. 7 (4), pp. 345-366, 1995.

[Folliot 96a] B. Folliot, Editorial, Calculateurs Parallèles, Numéro Thématique : Placement Dynamique et Répartition de Charge, Vol 8 (1), 1996.

[Folliot 96b] B. Folliot, «Contribution à une approche système du placement dynamique dans les systèmes répartis hétérogènes», Thèse d'habilitation à diriger les recherches de l'Université Pierre et Marie Curie, Décembre 1996.

[Glover & al. 92] F. Glover & M. Laguna, «Tabu Search, a chapter in Modern Heuristic Techniques for Combinatorial Problems», W.H. Freeman Eds., N-Y, 1992.

[Gonzales 77] M.J. Gonzales, «Deterministic Processor Scheduling», ACM Computing Surveys, Vol. 9 (3), September 1977.

[Gregory 92] K. Gregory, «Programming with Motif», Springer Verlag Eds., 1992.

[Grimshaw & al. 94] A. Grimshaw, W. Wulf, J. French & al., «Legion: The Next Logical Step Toward a Nationwide Virtual Computer», Technical Report CS-94-21, Department of Computer Science, University of Virginia, 1994.

[Hansen & al. 86] J.V. Ansen & W.C. Giauque, «Task Allocation in Distributed processing Systems», Operations Research Letters, Vol 5, N° 3, pp. 137-143, August 1986.

[Hémery 94] F. Hémery, «Etude de la répartition dynamique d'activités sur architectures décentralisées», Thése de doctorat de l'Université des Sciences et Technologie de Lille, Numéro d'ordre 1339, Juin 1994.

[Holland 75] J.H. Holland, "Adaptation in natural and artificial Systems", Ann Arbor, University of Michigan Press, 1975.

[Inmos 88] Inmos, «Transputer Development System», Prentice Hall Eds., 1988.

[Jun & al. 90] C. Jun, X. li & S. Zhong-xiu, «A Model for Intelligent Task Scheduling in a Large Distributed System», Operating System Review, Vol. 24(4), pp. 26-33, Oct. 1990.

[Kremien & al. 93] O. Kremien, J. Kramer & J. Magee, "Scalable Load Sharing for Distributed Systems", Proceedings of the 26th Hawaii International Conference on System Sciences, IEEE Computer Society Press, Los Alamitos, 1993, pp.632-641.

[Lo & al. 89] W-T. Lo, S.K. Tripathi & D. Ghosal, «A scheme for allocating task graphs on hypercubes», High Performance Computing, J-L. Delhage Eds., Elsevier Science Publishers, pp. 167-180, 1989.

[McBryan 94] O. McBryan, «An Overview of Message Passing Environments», Parallel Computing, Vol. 20, pp. 417-443, 1994.

[Norman & al. 93] M.G. Norman & P. Thanisch, «Models of Machines and Computation for Mapping in Multicomputers», ACM Computing Surveys, Vol. 25(3), pp. 263-302, 1993.

[Pellegrini 95] F. pellegrini, «Application de méthodes de partition à la résolution de problèmes de graphes issus du parallèlisme», Thèse de doctorat de l'université de Bordeaux I, No 1224, Janvier 1995.

[PRC 96] «Placement et Répartition de Charge : application aux systèmes parallèles et répartis», G. Bernard, J. Chassin de Kergommeaux, B. Folliot, C. Roucairol Eds., Collection didactique INRIA, Décembre 1996.

[Prylli 95] L. Prylli, «Distributed simulation of parallel computers», Research Report LIP/95-12, Ecole Normale Supérieure de Lyon, Laboratoire de l'Informatique du Parallélisme, May 1995.

[Raverdy 96] P.-G. Ravedy, «Gestion des ressources et répartition de charge dans les systèmes hétérogènes à grande échelle : application aux environnements mobiles et parallèles», Thèse de l'Université Pierre et Marie Curie, 4 place Jussieu, 75252 Paris cedex 05, 1996.

[Ross & al. 90] A. Ross & B. McMillin, «Experimental Comparison of Bidding and Drafting Load Sharing Protocol», The Vth Distributed Memory Computing Conference, Charleston, South Carolina, pp. 968-974, April 1990.

[Shen & al. 85] C.-C. shen & W.-H. Tai, «A graph matching computing systems using a minmax criterion», IEEE Transactions on Computer Systems, Vol. 3 (4), pp. 197-203, March 1985.

[Simulog 88] Simulog (Eds.), «Manuel de référence de QNAP2», 1988.

[Smith 88] R.G. Smith, «The contract net protocol: High level communication and control in a distributed problem solving» Readings in Distributed Artificial Intelligence, Morgan Kauffmann Publishers, California, pp. 357-366, 1988.

[Sinclair 87] J.B. Sinclair, «Efficient computation of optimal assignements for distributed tasks», Journal of Parallel and Distributed Computing, Vol. 4, pp. 342-362, 1987.

[Stone 77] H.S. Stone, «Multiprocessor scheduling with the aid of network flow algorithms», IEEE transactions on Software Engineering, Vol SE-3 (2), pp. 85-93, January 1977.

[Talbi 93] E. Talbi, «Allocation de processus sur les architectures parallèles à mémoire distribuée», thèse de doctorat de l'INPG, Mai 1993.

[Theimer & al. 89] M. Theimer & K. Lantz, «Finding Idle Machines in a Workstation-based Distributed System», IEEE Transaction on Software Engineering, Vol. 15 (11), pp. 1444-1458, Nov. 1989.

[Tokoro 90] M. Tokoro, «Computational Field Model: toward a new computing model/technology for open distributed environment», Proceedings of the 2nd Workshop on Future Trends in Distributed Computing Systems, Le Caire, Egypt, September 1990.

[Tron 94] C. Tron, «Modèles quantitatifs de machines parallèles : les réseaux d'interconnexion», Thèse de doctorat de l'Institut Polytechnique de Grenoble, Décembre 1994.

[Van Steen & al. 95] M. Van Steen, P. Homburg, L. Van Doorn, A. Tanenbaum & W. de Jonge, «Toward Object-based Wide Area Distributed Systems», Proceedings of the International Workshop on Object Orientation in Operating Systems, pp. 224-227, 1995.

[Whiley & al. 90] D. Whiley, T. Starkweather & C. Bogart, «Genetic algorithms and neural networks: optimizing connections and connectivity», Parallel Computing, Vol. 14 (3), pp. 347-361, Août 1990.

[Yang 92] Cui-Quing Yang, «Distributed Computing in a NUMP Environment», Operating Systems Review, Vol. 26(2), Avril 1992, pp. 82-91, April 1992.

[Zhou & al. 87] S. Zhou & D. Ferrari, «A Mesurement Study of Load Balancing Performance», Proceedings of the 7th International Conference on Distributed Computing Systems, pp. 490-497, Sept.ember 1987.

[Zhou 88] S. Zhou, «A Trace-driven Simulation Study of Dynamic Load Balancing», IEEE Transactions on Software Engineering, Vol. 14, pp. 1327-1341, 1988.

[Zhou & al. 93] S. Zhou, X. Zheng, J. Wang & P. Delisle, «Utopia: A Load Sharing Facility For Large, Heterogeneous Distributed Computer Systems», SOftware - Practice and Experience, Vol. 23(12), pp. 1305-1336, December 1993.