Le metacomputing au service du calcul haute performance

Pierre Kuonen* — Nabil Abdennadher* — Gilbert Babin**

** Département d'informatique, Ecole Polytechnique Fédérale*
CH-1015 Lausanne, Suisse
{Pierre.Kuonen,Nabil.Abdennadher}@epfl.ch

*** Département d'informatique, Université Laval*
Québec, Canada G1K 7P4
Gilbert.Babin@ift.ulaval.ca

RESUME. *Dans cet article nous nous intéressons aux nouveaux outils qui apparaissent depuis quelques années dans les laboratoires de recherche informatique et qui permettent l'exécution transparente d'applications distribuées sur les grands réseaux informatiques. Cette nouvelle approche porte maintenant un nom : le* metacomputing. *Ces environnements vont très certainement prendre de plus en plus d'importance dans le domaine du calcul de haute performance et plus particulièrement dans celui des applications parallèles. Au travers de l'analyse de trois de ces outils, nous illustrons les différentes approches utilisées aujourd'hui et nous identifions certaines de leurs lacunes en ce qui concerne le développement et l'exécution d'applications de haute performance parallèles. En nous basant sur cette analyse, nous proposons une nouvelle approche pour la réalisation de tels environnements, qui permet de contourner les problèmes rencontrés. Enfin nous proposons d'implémenter une telle approche dans le contexte du projet WOS™.*

ABSTRACT. *This article focusses on new tools available in computer science laboratories which support the transparent execution of distributed applications. This new approach is called* metacomputing. *Metacomputing environments will certainly become more and more important in high performance computing, in particular for parallel applications. We present an analysis of three of these tools. This analysis will serve to identify shortcomings of these approaches for the development and execution of high performance computing applications, and will give a basis from which a new approach may be developed, avoiding the problems identified. Finally, we propose the WOS™ as an implementation framework for this new approach.*

MOTS-CLÉS : *Metacomputing, Programmation parallèle et distribuée, Calcul haute performance*

KEY WORDS : *Metacomuting, Parallel and Distributed Programming, High Performance Computing*

1. Introduction

Plusieurs phénomènes sont en train de changer radicalement le paysage du calcul de haute performance[1]. Ce sont principalement les suivants :

– Le *commodity-based supercomputing* : l'objectif de cette approche est de faire chuter le rapport prix/performance des ordinateurs de haute performance en les réalisant à partir d'éléments matériels et logiciels provenant du marché grand public (essentiellement celui des PC). Cette approche est souvent utilisée pour réaliser des *cluster computers* [BUY 99].

– Le *metacomputing* est né de la constatation que les ordinateurs, autrefois individuels, sont maintenant, pour la plupart, connectés en réseau. L'ensemble de la puissance de calcul non utilisée sur chaque ordinateur individuel représente une capacité de calcul énorme. Les réseaux qui relient les ordinateurs entre eux sont de plus en plus performants. Il devient ainsi tout à fait envisageable d'utiliser l'ensemble de cette puissance de calcul pour résoudre des problèmes difficiles.

– Le concept de *Grid* introduit dans [FOS 99] est une approche similaire au *metacomputing*. Elle a pour but de relier entre eux différents ordinateurs relativement puissants, éventuellement localisés sur des sites éloignés, afin de les utiliser, d'une part comme un seul ordinateur super puissant et d'autre part comme moyen de communication sophistiqué entre les différents experts intervenant dans la résolution de problèmes complexes.

Bien que n'étant pas parfaitement identiques, ces différentes approches visent toutes à réduire le coût de l'accès au calcul de haute performance. Cette diminution sera obtenue grâce, d'une part, à la diminution du coût du matériel et d'autre part grâce à l'apparition de nouveaux environnements de programmation et d'exécution permettant la réalisation et l'exploitation d'applications HPC à des coûts raisonnables. De plus, les quantités de calculs, et donc de résultats produits par les applications HPC devenant de plus en plus importantes, la visualisation de ces résultats, voire même une interaction avec les calculs, devient de plus en plus nécessaire. Jusqu'à récemment, le principal paradigme d'exécution utilisé par les applications HPC était l'exécution par lots (*batch mode*). Ce paradigme doit nécessairement être abandonné au profit d'une approche interactive comme c'est le cas pour la plupart des outils informatiques modernes.

Le défi posé aux informaticiens n'est donc plus d'extraire, à n'importe quel prix à l'aide d'applications exécutées par lots, le maximum de puissance d'une machine coûteuse donnée, mais plutôt d'extraire, à faible coût à l'aide d'applications interactives, une puissance de calcul raisonnable d'un environnement pouvant, potentiellement, fournir de grandes puissances de calcul.

[1] En anglais HPC pour *High Performance Computing*

2. De la technologie de l'information à la technologie du calcul

L'extraordinaire développement des grands réseaux informatiques qui s'est fait durant les dix dernières années, a considérablement changé notre façon d'utiliser les ordinateurs. En effet, la technologie du Web, avec tout le cortège d'outils qui lui est associé, a soudainement donné accès à toute une gamme d'informations et de données qui étaient jusque-là inaccessibles. Aujourd'hui, aucun utilisateur d'informatique ne peut se passer de l'incroyable quantité et diversité d'informations disponibles sur le Web. Le terme « technologie de l'information » tend d'ailleurs à se substituer à celui « d'informatique » et le marché qui lui est associé est l'un des plus porteurs du moment.

Mais la deuxième phase de la révolution provoquée par le développement des grands réseaux informatiques est déjà en route. Non content de fournir l'accès à des informations distantes, les technologies associées au Web commencent à permettre l'accès à de la puissance de calcul distante. On peut s'attendre à ce que cette deuxième révolution affecte principalement les entreprises de tailles moyennes. En effet, ces entreprises disposent généralement de bons réseaux informatiques internes ainsi que d'accès performants au réseau mondial. La puissance de calcul potentielle à leur disposition est donc importante mais reste jusqu'à présent inexploitable. D'un autre coté, leur besoin en HPC, quoique souvent réel, n'est pas assez critique pour qu'elles investissent dans des superordinateurs coûteux et spécifiques. La *technologie du calcul* pour utiliser un néologisme, ou le *global computing* pour utiliser le terme anglophone qui commence à apparaître, devrait donc fournir à ces entreprises les environnements dont elles ont besoin.

Pour réaliser ces nouvelles applications, différents outils sont à la disposition de l'informaticien. Il est possible de les classer en trois catégories :

– Les langages de programmation : Il s'agit de tous les langages permettant d'écrire des applications sur le Web. JAVA en est l'exemple type. Ces langages ont été associés à la première phase de la pénétration du Web. Ils ont été principalement conçus dans le but d'accéder à de l'information distante. Lorsqu'ils permettent de lancer des calculs à distance, la performance n'est en général pas leur soucis principal. Ils sont donc mal adaptés à l'écriture d'applications parallèles performantes.

– Les outils de développement d'applications réparties : Le représentant type de cette classe est CORBA. Développés pour permettre la réalisation d'applications réparties sûres, généralement de type clients-serveurs, ces outils mettent principalement l'accent sur le portabilité, la fiabilité et l'hétérogénéité de l'environnement plutôt que sur les performances. N'ayant pas été conçus dans ce but, ils sont donc également mal adaptés à l'écriture d'applications parallèles performantes.

– Les outils créés spécifiquement pour le *metacomputing* : Relativement récents, ces outils ont pour objectif de permettre l'exploitation de grands réseaux informatiques à des fins calculatoires. C'est cette classe que nous allons étudier.

Aujourd'hui, il existe de nombreux projets ayant pour objectif de développer des outils qui permettent l'implémentation du concept de *metacomputing* [GRI 94, ALE 96, BAR 96, CAS 97, VAH 98, ALM 98, FOS 99, LAS 00]. Il n'est pas envisageable dans le cadre de cet article de les présenter tous. On peut, par contre, au travers de quelques exemples illustrer les principales approches utilisées. Dans la suite de cet article nous présenterons et comparerons trois outils : NetSolve, Globus et Cobra. NetSolve est un outil développé par une équipe de chercheurs actifs dans le domaine du HPC. Il est donc représentatif d'une approche très pragmatique adoptées par des personnes pour qui la performance et la simplicité restent les préoccupations principales. Globus est très certainement le principal et le plus connu des projets dans le domaine du *metacomputing*. Il a pour but d'implémenter le concept de *Grid* présenté dans [FOS 99]. Il constitue une référence incontournable dans le domaine du *metacomputing*. Enfin Cobra a pour principal but d'adapter des outils existants dans le domaine des applications réparties (dans ce cas CORBA) à l'écriture d'applications HPC distribuées. Nous discuterons de l'adéquation de ces outils par rapport au but fixé, à savoir la réalisation et l'exploitation, à moindre coût, d'applications HPC interactives permettant d'utiliser la puissance de calcul potentielle offerte par les grands réseaux informatiques.

3. Critères d'évaluation

Afin de comparer différents projets ou produits, parfois assez différents, et de déterminer leur adéquation à nos besoins, il est nécessaire de définir un ensemble de critères de comparaison.

Le choix de critères est souvent subjectif et donc discutable. Nous n'avons donc pas la prétention de dire que ceux proposés dans cet article soient complets et qu' ils permettront d'analyser un produit sans aucune ambiguïté. Toutefois ils serviront de fil conducteur à notre analyse et seront, le cas échéant, complétés par des commentaires. Voici la liste des critères choisis ainsi que leur interprétation.

– Hétérogénéité : L'outil doit permettre le développement et l'exploitation d'applications multi-matériels et multi-systèmes d'exploitation. Nous distinguerons ces deux sous-cas dans nos analyses.

– Flexibilité : Par flexibilté nous entendons la capacité de l'environnement à gérer efficacement les outils de programmation parallèle existants (ou qui existeront dans l'avenir). Cette caractéristique laisse la plus grande liberté possible à l'utilisateur quand aux modèles et outils de programmation parallèle qu'il peut utiliser lors du développement de son application.

– Performance : Un environnement de *metacomputing* est performant s'il impose un minimum de surcharge de calcul et/ou de communication lors de l'exécution d'une application. Par exemple, une solution « naturelle » au problème de flexibilité est d'implémenter les librairies de communication standards au

dessus d'une couche de communication fournie par l'environnement de *metacomputing*. Cette solution est souvent génératrice de pertes de performance liées au surplus de travail nécessaire lors de chaque communication.

– Dynamique : Ce critère peut avoir plusieurs interprétations. Tout d'abord l'outil est-il dynamique au sens de l'application, c'est-à-dire permet-il d'exprimer une application contenant des processus qui se créent et se détruisent en cours d'exécution ? Cette caractéristique peut aussi être supportée par l'outil de programmation parallèle (cas de PVM). Ensuite, l'exécution est-elle dynamique, c'est-à-dire peut-elle, d'une part se reconfigurer entre deux exécutions pour se répartir au mieux dans l'environnement parallèle (répartition de la charge) et d'autre part se reconfigurer en cours d'exécution pour s'adapter à un changement de la charge de l'environnement (migration de processus et/ou de données).

Les critères choisis restent délibérément à un niveau d'abstraction élevé. Ils ne s'intéressent pas, par exemple, au modèle exact de communication ou à la façon précise de calculer la charge. Notre objectif est, au travers de l'analyse de différents outils et projets, de présenter et de comparer différentes approches possibles du *metacomputing* et non de juger d'une implémentation particulière d'une de ces approches.

4. NetSolve

Le projet NetSolve a été développé à l'Université du Tennessee et au Laboratoire National d'Oak Ridge (*Oak Ridge National Laboratory*). La première version publique de NetSolve a été publiée en mars 1997 [CAS 97]. Il a été initialement conçu dans le but de permettre une utilisation partagée de programmes de calcul scientifique s'exécutant sur des ordinateurs dédiés, NetSolve est basé sur le modèle Client/Serveur. Le client ne connaît pas la localisation de la machine sensée exécuter son programme.

4.1. *Structure de NetSolve*

Un système NetSolve est composé d'un ensemble de services pouvant être appelés par des machines clientes. Chaque service est hébergé et exécuté au niveau d'un ou de plusieurs nœuds serveurs du réseau. Ces services sont répertoriés dans des « bases de données » réparties appelées *Agent NetSolve*. Un service est représenté par son nom, par son emplacement (adresses des machines qui l'exécutent) ainsi que par d'autres informations détaillées dans le paragraphe suivant. Un même service peut être inscrit auprès de plusieurs Agents. Par ailleurs, un service peut exister sous plusieurs versions sur plusieurs machines et être répertorié par le même Agent. Dans ce cas, plusieurs programmes représentent le même service. Pour pouvoir exécuter un service NetSolve, une machine cliente doit être inscrite auprès d'un Agent. Ceci signifie que le client connaît l'adresse de

l'Agent. Le service demandé doit être catalogué auprès du même Agent. A la réception d'une requête client (demande d'exécution d'un service), l'Agent NetSolve consulte sa base de données pour déterminer les machines susceptibles de répondre à la requête, calcule leur charge (*workload)* et retient la moins chargée comme serveur « élu ». Le calcul de la charge d'un serveur est détaillé dans le paragraphe qui suit. Notons enfin qu'un Agent NetSolve ne dispose que d'une vue limitée et statique de la totalité du système, sa base de données ne contient, en général, qu'un sous ensemble des services disponibles. La figure 1 illustre l'architecture de NetSolve [CAS 98a].

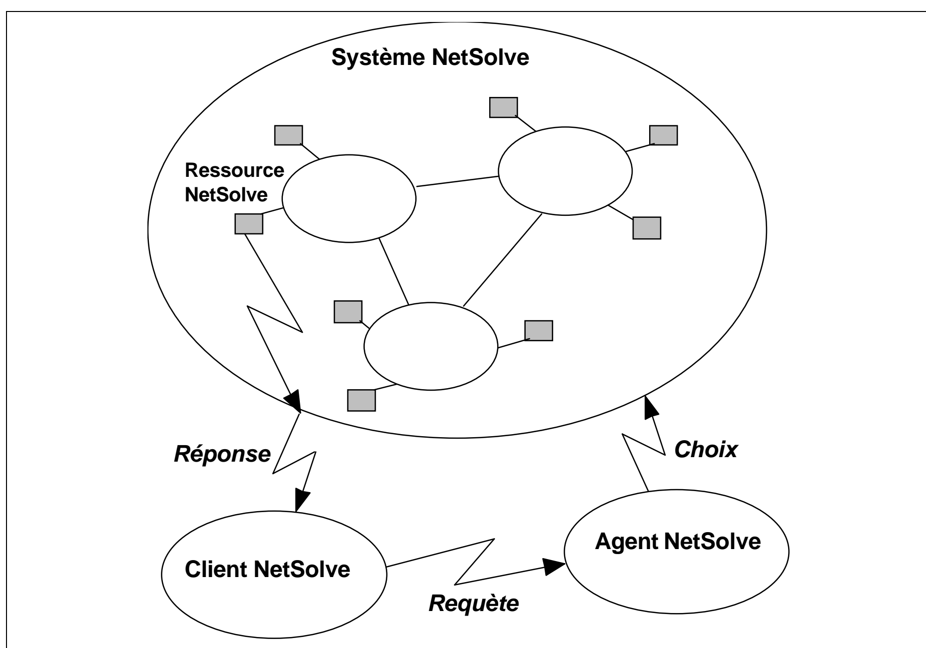

Figure 1. *Architecture de NetSolve*

L'ajout et/ou la suppression d'un service au niveau d'un Agent se fait manuellement. L'appel d'un service NetSolve peut se faire interactivement ou à partir d'un programme. Des librairies de routines sont disponibles à cet effet pour plusieurs langages tels que C, Fortran, JAVA, MATLAB, MATHEMATICA. Ces appels peuvent être bloquants (attente de retour du résultat) ou non bloquants. Le même service peut être exécuté en parallèle sur des données différentes : c'est le *Process Farming* [CAS 98a, CAS 98b]. Ceci revient à exécuter en parallèle les (ou une partie des) programmes représentant le service concerné. Chaque programme s'exécute sur son propre serveur. La gestion de la charge des serveurs permet un déploiement optimisé du modèle SPMD (*Single Program Multiple Data*).

4.2. *Description d'un service et calcul et de la charge d'un serveur*

Un serveur s'inscrit au système NetSolve en fournissant un ensemble d'informations dont les plus importantes sont :

– l'adresse de l'Agent NetSolve où sera inscrit le serveur ;

– le nombre de processeurs pouvant être utilisés en parallèle par le serveur pour exécuter les services NetSolve ;

– la performance brute du serveur ;

– la liste des services offerts par le serveur.

Pour chaque service offert, le serveur doit indiquer :

– son nom (ce nom sera utilisé par le client NetSolve au moment de l'appel) ;

– la description de ses variables d'entrée et de sortie (type de données, type d'objets, etc.) ;

– la complexité algorithmique du programme correspondant. Cette complexité est modélisée par un couple (x,y) et est de la forme :

$$x \cdot S^{y}$$

avec : S = taille des données d'entrée déduite à partir du type des variables d'entrée.

A la réception d'une demande de service, l'Agent consulte sa base de données pour identifier les serveurs pouvant exécuter ce service. Il estime ensuite le temps d'exécution T que prendrait l'exécution du service sur chaque machine candidate. Cette estimation se fait sur la base des données relatives au client (taille des données d'entrée et de sortie), au serveur (caractéristiques du réseau reliant le client au serveur, charge du serveur, etc.) et au service lui-même (complexité algorithmique du programme à exécuter).

La valeur de T est estimée en fonction du temps d'exécution du programme appelé (Te) et du temps de transmission des données d'entrée et de sortie du client vers le serveur et vice versa (Tc). Ce dernier dépend de trois paramètres : taille des données transmises, bande passante et temps de latence du réseau. La valeur de Te est calculée en fonction de la complexité algorithmique du programme appelé et de la performance p du serveur au moment de l'exécution. Cette performance est estimée en fonction des paramètres suivants : nombre de processeurs n, performance brute P, et charge w de la machine. Les deux premiers paramètres sont statiques. Le troisième étant dynamique, il est transmis vers l'agent chaque fois que sa valeur change de manière significative. Une modélisation de la performance p en fonction de P, n et w est proposée dans [CAS 97].

4.3. *Évaluation*

L'objectif de ce paragraphe est d'évaluer NetSolve par rapport aux critères définis dans la section précédente.

– Hétérogénéité : NetSolve n'impose aucune restriction quant à l'architecture matérielle ou le système d'exploitation de la machine cible. De plus, les librairies d'appel au système NetSolve (API) assurent une certaine autonomie vis-à-vis des langages de programmation.

– Flexibilité : Netsolve ne supporte pas les outils de programmation parallèle standards. Le seul modèle proposé par NetSolve est celui du *Remote Program Call*. Une variante de ce modèle est proposé pour le cas du *Process Farming* ; elle permet d'exprimer d'une manière élégante le parallélisme SPMD. L'expression de ce modèle reste toutefois propriétaire.

– Performance : Étant un environnement de *Remote Program Call,* NetSolve ne reconnaît pas une application parallèle comme telle. Celle-ci est considérée comme une boite noire et est exécutée sur une machine donnée. La performance obtenue est donc uniquement liée à la configuration faite par le programmeur et à la performance de la machine cible. Rappelons, en effet que NetSolve ne peut configurer une application parallèle que dans le cas où le parallélisme est exprimé grâce au modèle *Process Farming* qu'il propose.

– Dynamique : L'appel d'un service NetSolve à partir d'un programme et le calcul de la charge d'un serveur potentiel permettent, respectivement, la création dynamique de processus et l'équilibrage de charge des sites. La migration des processus, en d'exécution n'est pas possible.

L'objectif essentiel de NetSolve est de trouver la meilleure machine pour l'exécution à distance d'un programme. Dans ce contexte certains ingrédients dub HPC sont pris en considération : équilibrage de charge, expression du *Process Farming* et du *Remote Program Call* bloquant ou non. Ces deux modèles qui sont largement utilisés par les concepteurs d'applications parallèles sont implémentés de manière propriétaire dans NetSolve. Les outils de programmation standards (MPI, PVM, CC++[2], Fortran M, etc.) ne sont pas supportés par NetSolve.

Notons enfin, que NetSolve est utilisé avec plusieurs langages de programmation (Fortran, C, JAVA, MATHLAB, MATHEMATICA). La possibilité de développer des routines d'appel à NetSolve rend envisageable l'intégration d'autres langages de programmation.

5. Globus

Initialement mené, depuis le début des années 90 par le Laboratoire National d'Argonne (*Argonne National Laboratory*) et l'Université de Californie du sud

[2] *Compositional C++*

(*University of South California*), le projet Globus regroupe maintenant plusieurs autres universités et centres de recherche. Ces universités et centres de recherche constituent actuellement une large plate-forme expérimentale de Globus (projet GUSTO).

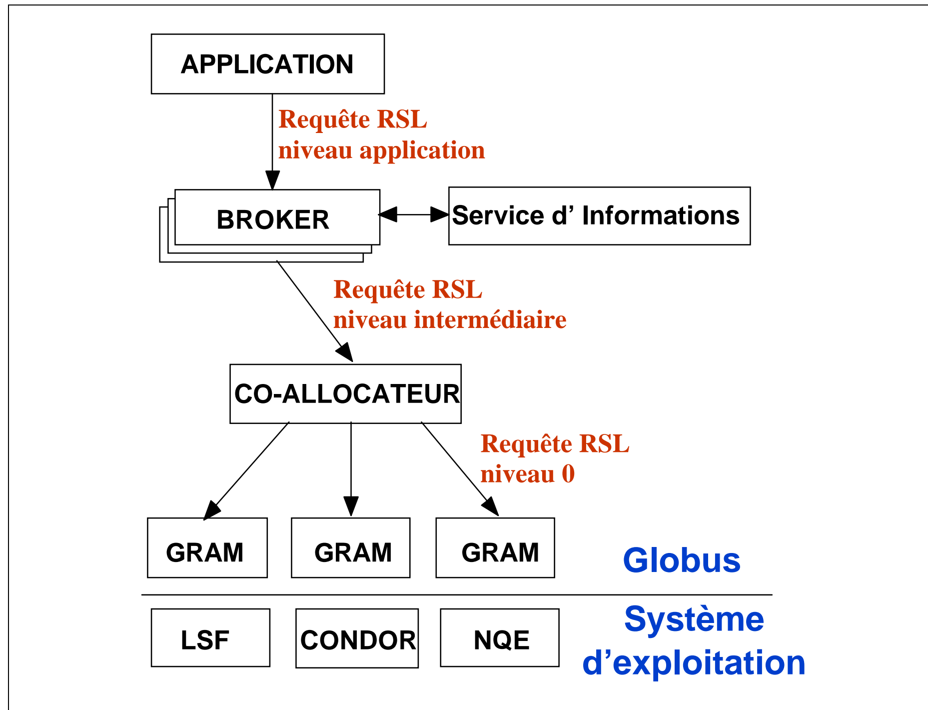

Figure 2. *Architecture du système Globus*

L'architecture de Globus est détaillée dans la figure 2. L'idée est de permettre à l'utilisateur de Globus d'exprimer sa requête en un langage générique qui sera traduit par le système en une demande de ressources nécessaires pour l'exécution du service demandé. L'emplacement de ces ressources est transparent à l'utilisateur. Le langage de requête proposé par Globus (*Request Specification Langage : RSL*) peut être spécialisé pour une famille particulière de services (applications). Une requête RSL est traitée par des *brokers* pour être traduite en terme de ressources physiques reconnaissables par un système d'exploitation local. Une requête RSL peut faire appel à plusieurs ressources géographiquement réparties. Un co-allocateur permet alors de les localiser et de décomposer la requête RSL « multi-sites » en plusieurs requêtes « mono-site ».

Globus est essentiellement composé des modules et outils suivants [FOS 97b, CZA 98] :

– le langage RSL (*Request Specification Language*);

– les coursiers (*brokers*)

– les gestionnaires et allocateurs de ressources (*Globus Resource Allocation Managers* ; *GRAM*);

– les co-allocateurs;

– les services d'information (*Metacomputing Directory Service* ; *MDS)*;

– la couche de communication Nexus.

5.1. *Le langage RSL et ses requêtes*

Les requêtes utilisateurs sont exprimées dans un langage spécifique appelé RSL (*Request Specification Langage*). Ce langage RSL permet à l'utilisateur (application cliente) d'exprimer ses requêtes en terme de ressources. L'expression des besoins peut se faire explicitement (nombre de processeurs requis, quantité de mémoire demandée, type de réseau utilisé, etc.) ou de manière abstraite (exécution d'une application en moins de x unités temps, etc.). La grammaire BNF de ce langage est simple et utilise une notation préfixe [CZA 98] ; sa syntaxe repose sur les structures de données de la norme LDAP (*Lightweight Directory Access Protocol*) [CZA 98, FOS 98a, FIT 97]. Cette norme est utilisée par le système d'informations de Globus (voir ci-dessous). Un exemple simple d'une requête RSL est le suivant :

```
& (executable = mon_programme)
  (| (& (count = 5) (memory >= 64))
     (&(count = 10)(memory>= 32))
  )
```

où `&` exprime la conjonction de deux conditions (ET logique) et `|` exprime leur disjonction (OU logique).

Cette requête peut être traduite en langue naturelle de la façon suivante :

Exécuter le programme parallèle « mon_programme » sur 5 processeurs ayant une mémoire centrale d'au moins 64 Moctets chacun ou 10 processeurs ayant une mémoire centrale d'au moins 32 Moctets.

Dans ce cas, la requête est simple et les ressources demandées sont quantitativement et qualitativement (mémoire centrale et processeurs) définies. Ce sont des requêtes dites de niveau 0.

Une requête RSL peut être générique et abstraite. Dans ce dernier cas, on parle alors d'une requête RSL de haut niveau. Considérons, par exemple, la requête RSL suivante :

Exécuter une application de simulation interactive de 100 000 entités.

La formulation syntaxique de cette requête importe peu dans notre contexte. Les mots clés à retenir sont « simulation interactive » et « 100 000 entités ». Les informations relatives à ces mots clés se trouvent dans le service d'informations. Un compilateur spécialisé (appelé *broker* en anglais) se charge de traduire cette requête « abstraite » en une requête RSL de niveau 0 exprimée en terme de ressources physiques : nombre de processeurs, quantité mémoire, vitesse de réseau, etc. Dans cet exemple, le *broker* en question est spécialisé dans le traitement des requêtes RSL faisant intervenir des applications de simulation interactive. C'est le processus de *spécialisation*.

En général, une requête RSL de haut niveau (niveau application) doit être traitée par un ou plusieurs *brokers* pour aboutir à une requête RSL de niveau 0 (*ground RSL request*) exprimée en terme de ressources physiques. Il existe donc trois types de requête RSL :

– requête RSL de haut niveau (niveau application) : Ces requêtes sont données par l'utilisateur. Elles feront l'objet d'une ou de plusieurs traductions effectuées par un ou plusieurs *brokers* spécialisés afin de la traduire en terme de ressources physiques parfaitement définies, c'est-à-dire en requête RSL de niveau 0 ;

– requête RSL de niveau intermédiaire : Ce sont les requêtes résultant du traitement d'un *broker*. L'expression des besoins, au niveau de ces requêtes, est plus fine que celle d'une requête de haut niveau mais pas suffisamment raffinée pour être reconnue localement au niveau d'un seul site ;

– requête RSL de niveau 0 (*ground RSL request*) : Elle est généralement le résultat de la compilation d'un ou de plusieurs *brokers*. Les ressources appelées par cette requête sont définies et contrôlées localement au niveau du même site.

5.2. *Coursiers (Brokers)*

Les *brokers* ont pour but de transformer une requête RSL de haut niveau, exprimée par l'utilisateur, en une ou plusieurs requêtes RSL de niveau 0 pouvant être traitées par un ou plusieurs GRAMs (voir sect. 5.3). Les *brokers* sont donc pour les requêtes RSL de haut niveau ce que sont les compilateurs pour les

langages de programmation. Les *brokers* peuvent être spécialisés : *brokers* orientés applications de simulation, de traitement de signal, de calcul matriciel, etc. Ils peuvent être construits par l'utilisateur lui-même selon ses besoins.

5.3. *Gestionnaires et Allocateurs de Ressources (GRAM)*

Le GRAM joue le rôle d'interface entre le système Globus et les environnements locaux des différents sites. Il permet donc de gérer l'hétérogénéité du matériel et du système d'exploitation. Le GRAM dépend étroitement des ordonnanceurs et allocateurs de ressources locaux. Actuellement, six interfaces GRAM sont disponibles avec les ordonnanceurs Condor, EASY, Fork, LoadLeveler, LSF et NQE. Les fonctions essentielles d'un GRAM sont les suivantes :

– allocation des ressources et traitement des requêtes RSL de niveau 0. Ces requêtes sont supposées être suffisamment explicites. Elles font référence à des ressources locales identifiées et contrôlées par l'ordonnanceur local ;

– mise-à-jour du système d'informations local afin de pouvoir refléter en « temps réel » l'état et la disponibilité des ressources locales.

5.4. *Co-Allocateurs*

Dans certains cas, une requête RSL peut faire référence à plusieurs ressources réparties sur des sites différents. Cette requête doit être prise en charge par plusieurs GRAMs. Le rôle du co-allocateur est de décomposer cette requête en plusieurs sous-requêtes, chacune de ces sous-requêtes faisant intervenir un site différent et ne faisant référence qu'à des ressources localisées sur ce site. Chaque sous-requête sera traitée par le GRAM qui contrôle les ressources locales qu'elle appelle. Il va donc sans dire que les ressources d'une même application peuvent être géographiquement réparties sur plusieurs sites. Cette caractéristique est utile dans le cas d'un programme parallèle où chaque tâche reçoit ses données d'entrée (ressources) en provenance d'autres tâches, effectue un certain traitement et renvoie le résultat à d'autres tâches.

5.5. *Service d'Informations (MDS)*

Ce service permet un accès efficace et réparti aux informations relatives aux types et à la disponibilité des ressources contrôlées par tous les sites du système. Ces informations sont utilisées pour diverses raisons : localisation des ressources ayant une certaine caractéristique (performance, disponibilité, etc.), identification

du GRAM associé à une ressource et enfin transformation d'une requête de haut niveau (resp. intermédiaire) en une requête intermédiaire (resp. de niveau 0). MDS utilise la structure de données et la librairie de routines (API) définies par le *Lightweight Directory Access Protocol* (LDAP) pour le cataloguage et la désignation des ressources. Une description détaillée de MDS et LDAP se trouve dans [FIT 97].

5.6. *Couche de communication Nexus*

Outre les modules présentés ci-dessus, Globus dispose d'une couche de communication, appelée Nexus, qui permet l'échange de données non seulement entre processus, mais aussi entre *threads* [FOS 96]. Ces *threads* peuvent être locaux (appartenant au même processus) ou distants (appartenant à des processus différents). Nexus est aussi perçu comme machine virtuelle constituée d'un ensemble d'instructions utilisées lors de la compilation de l'application parallèle ; il constitue à ce titre la machine parallèle vue par le compilateur. Cette idée romp avec l'approche classique qui consiste à fournir une bibliothèque de routines appelées à partir du programme source. Actuellement, deux compilateurs de langages parallèles supportent Nexus : CC++ et Fortran M [NEX 94].

Nexus donne aussi la possibilité aux concepteurs d'applications parallèles et/ou distribuées de cacher l'interdépendance qui existe souvent entre l'outil de programmation parallèle (MPI, PVM, CC++, etc.) et le protocole de communication (TCP/IP, mémoire partagée, etc.) [FOS 97a]. Il devient alors possible, par exemple, d'écrire une application parallèle en utilisant la librairie MPI sans se préoccuper de l'architecture de la machine cible (mémoire partagée ou locale) ou du protocole de communication utilisé.

5.7. *Évaluation*

Ce paragraphe évalue Globus en fonction des critères de comparaison de la section 3.

– Hétérogénéité : Grâce aux GRAMs, Globus permet de gérer l'hétérogénéité des sites tant au niveau matériel que logiciel. La couche Nexus permet l'implémentation de plusieurs outils de programmation parallèle (MPI, PVM, CC++, Fortran M, etc.) sur plusieurs protocoles de communication (TCP/IP, Intel NX *message-passing*, *Message Passing Library* (MPL) d'IBM, ATM, Myricom, UDP). Une seconde forme d'hétérogénéité, autre que celle des sites, est prise en charge par Globus : celle de la communication, tant sur le plan modèle que sur le plan protocole.

– Flexibilité : La couche Nexus de Globus permet l'implémentation de tout outil de programmation parallèle. Certains de ces outils sont d'ores et déjà

implémentées : MPI, CC++ et Fortran M. A ce titre, Globus est un outil de *metacomputing* flexible, moyennant un effort d'implémentation de l'outil de programmation parallèle désiré.

– Performances : Les premières mesures effectuées montrent que les implémentations des outils de programmation parallèle effectuées au dessus de Globus sont moins performants que les implémantations « natives ». La figure 3 montre la différence de performance entre MPI-G [FOS 98b] (implémentation de MPICH au-dessus de Globus) et MPICH [MPICH].

– Dynamique : Globus permet la gestion dynamique de l'application et des processus, et ce grâce à l'appel des requêtes RSL. La gestion dynamique de l'exécution est assurée grâce aux mises-à-jour *on line* du MDS. Toute modification d'environnement (ajout et/ou suppression de ressources) est automatiquement prise en compte. Cette fonctionnalité est garantie par la structure même du MDS, dérivée du LDAP.

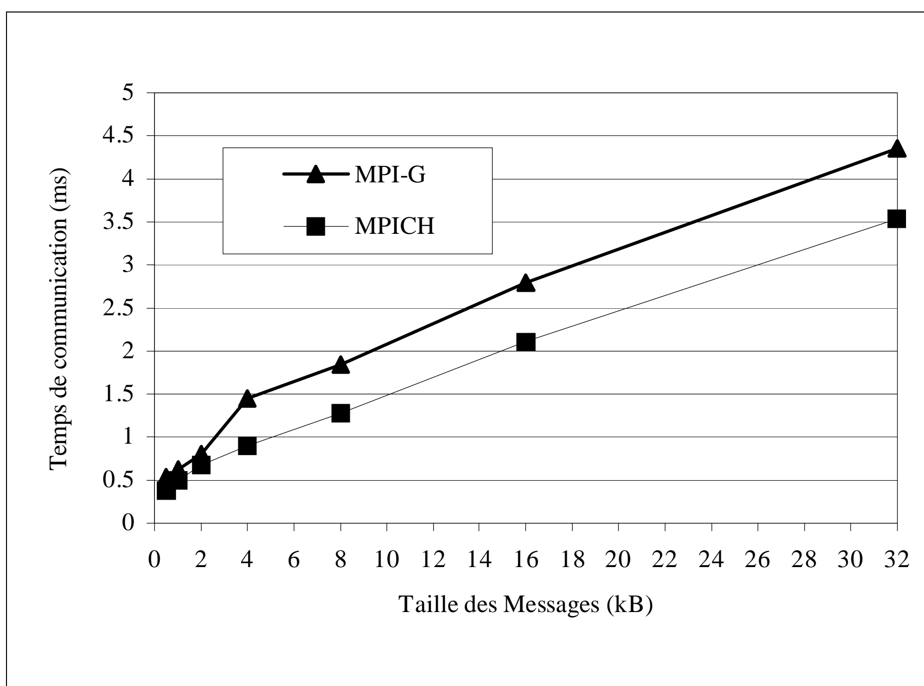

Figure 3. *Performances de MPI-G et de MPICH*

Contrairement aux autres outils de *metacomputing* qui proposent un modèle de programmation et d'exploitation uniforme cachant l'hétérogénéité, tant au niveau matériel que logiciel, Globus fournit une infrastructure pour les développeurs

d'applications parallèles et distribuées permettant une gestion raffinée de l'hétérogénéité. Il ne s'agit plus de cacher la différence (l'hétérogénéité) mais de la gérer efficacement en proposant une boite à outils basés sur des librairies et des composants standards. Un attention particulière a été portée aux aspects « diversité des ressources » et « haute performance » : choix des ressources les plus appropriées pour l'optimisation du temps de réponse.

Dans le contexte du calcul à haute performance, l'originalité de Globus, par rapport à NetSolve est la possibilité de faire appel, grâce aux co-allocateurs, à des ressources non nécessairement localisées sur le même site. Cette caractéristique est intéressante dans le cas des applications parallèles où les données d'entrée d'une tâche peuvent provenir de plusieurs autres tâches placées sur des processeurs différents.

6. Cobra

Cobra est un projet de l'institut de recherche en informatique et systèmes aléatoires (IRISA) de Rennes [PRI 98]. Son but principal est de permettre l'écriture élégante d'applications HPC parallèles intégrées à l'environnement CORBA (*Common Object Request Broker Architecture*) [GEI 97].

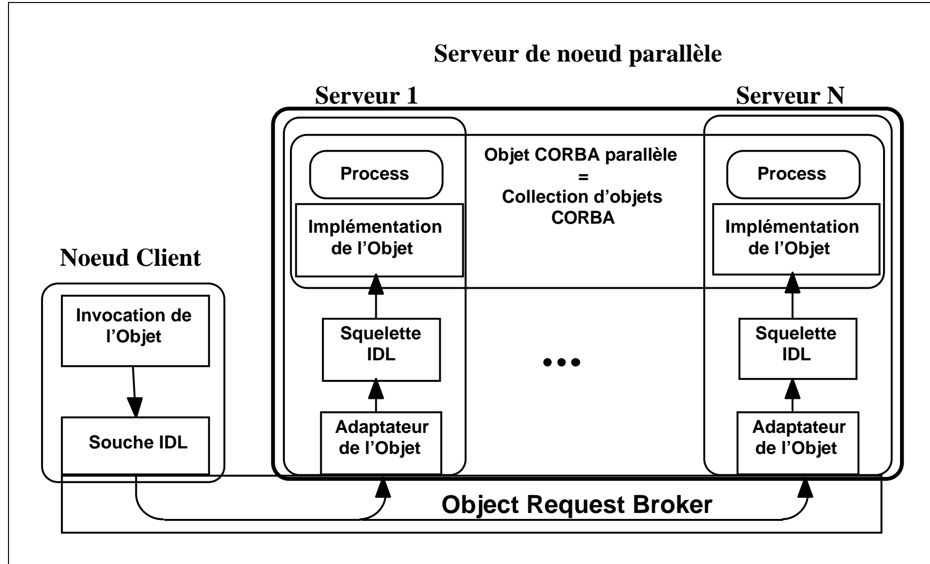

Figure 4. *Modèle d'exécution de Cobra (d'après [PRI 98])*

Les auteurs de ce travail sont partis de la constatation que le standard COBRA est un modèle adéquat pour l'écriture et l'exécution efficace d'applications orientées objets réparties à couplage faible mais reste inadapté à l'exécution d'applications parallèles à couplage fort. Ils ont donc proposé un ensemble minimal d'extensions permettant l'expression et l'exécution efficace d'objets parallèles dans l'environnement CORBA. Un objet parallèle est un objet CORBA formé de la duplication d'un même objet. Le nombre de duplications de l'objet peut être défini statiquement à la compilation ou peut être calculé dynamiquement lors de l'exécution. Du point de vue du client, il n'y a pas de différence entre un objet CORBA classique et un objet parallèle. Le parallélisme est caché à l'intérieur de l'objet parallèle. La figure 4 illustre ce principe. Le nœud client invoque un service selon le mécanisme classique de CORBA à savoir au travers d'une souche (*stub*) IDL (*Interface Definition Language*). La requête du client est ensuite transportée par le bus CORBA (*Object Request Broker*). Lorsque le nœud serveur est un objet parallèle, son interface est spécifiée à l'aide d'une extension du langage IDL de CORBA. Cette extension du standard IDL permet d'une part de spécifier l'interface de l'objet parallèle comme s'il s'agissait d'un objet classique et d'autre part de distribuer la requête sur la collection d'objets qui le constitue. En effet, ainsi qu'il est illustré sur la figure 4, l'interface de l'objet parallèle, spécifiée à l'aide du langage IDL étendu, génère pour chaque objet de la collection une spécification IDL tout à fait standard.

Voici un exemple de déclaration d'interface d'objet parallèle à l'aide du langage IDL étendu. (tiré de [PRI 98]) :

```
interface [*] MatrixComponent
{
  void matrix_vector_mult(in double mat[100][100],
                          in double v[100],
                          out double u[100]);
  void matrix_transpose(in double A[100][100],
                        out double B[100][100]);
}
```

La syntaxe `[*]` indique que l'objet `MatrixComponent` est un objet parallèle constitué d'un certain nombre (indéterminé à la compilation dans le cas du signe `*`) de duplications de lui-même. A la place du signe `*` on peut mettre un chiffre ou une expression arithmétique. L'exécution d'une méthode quelconque d'un objet parallèle (par exemple la méthode `matrix_transpose`) consiste à l'exécuter en parallèle dans chacune des instances de l'objet constituant l'objet parallèle. Lors de l'exécution d'une méthode d'un objet parallèle, les paramètres de la méthode (les données) sont distribuées dans les différents instances constituant l'objet parallèle. Le programmeur peut imposer la façon de distribuer les données selon un modèle équivalent aux modes de distribution des données

définies dans HPF (*High Performance Fortran*). Des extensions adéquates du langage IDL ont été définies à cet effet. Voici un exemple tiré de [PRI 98] :

```
interface [*] MatrixComponent
{ void matrix_vector_mult(in dist[BLOCK][*] double mat[100][100],
                          in double v[100],
                          out dist[CYCLIC] double u[100];}
```

Nous n'entrerons pas dans tous les détails du modèle Cobra. Notons simplement qu'une interface d'objet parallèle ne peut pas hériter d'une interface objet non-parallèle et que l'héritage entre interfaces d'objets parallèles est possible moyennant certaines restrictions.

6.1 *Évaluation*

Mise à part ce qui concerne les objets parallèles, les caractéristiques du modèle Cobra sont essentiellement héritées de CORBA. Ce sont les suivantes :

– Hétérogénéité : Elle est obtenue par l'utilisation du modèle CORBA. En effet l'extension des objets parallèles n'est pas hétérogène ;

– Flexibilité : En utilisant Cobra, le développeur d'applications parallèles est contraint d'utiliser les approches de parallélisation propriétaires supportées par cet outil : les objets répartis offerts par CORBA et les objets parallèles proposés par l'extension Cobra. A ce titre Cobra est peu flexible ;

– Performances : Nous n'avons pas effectué de mesures de performance avec Cobra. Toutefois, le concept d'objets parallèles a été spécifiquement développé pour permettre une implémentation efficace du parallélisme de données. On peut donc raisonnablement admettre qu'il est tout a fait possible de réaliser des implémentations performantes de Cobra pour autant que l'on se limite au modèle de parallélisme imposé par la notion d'objets parallèles ;

– Dynamique : Le modèle client/serveur de CORBA ainsi que le fait que le nombre d'instances dupliquées dans un objet parallèle peut être déterminé lors de l'exécution, permettent l'écriture d'applications parallèles/distribuées dynamiques. Par contre aucun dynamisme n'existe en cours d'exécution.

Le but des auteurs de Cobra était d'être le plus compatible possible avec CORBA ; les extensions proposées sont donc minimales. C'est une excellente approche pour intégrer rapidement des applications parallèles dans un environnement CORBA. En ce sens, Cobra est un succès. Cobra présente toutefois deux inconvénients majeurs :

– il ne permet pas de gérer explicitement et efficacement les outils de programmation parallèle disponibles sur le marché (PVM, MPI, etc.) ;

– il oblige le programmeur à utiliser deux paradigmes de programmation parallèle. Cela contraint ce dernier à décider, lors de l'écriture de son programme,

des parties qui seront exécutées dans un environnement parallèle efficace (les objets parallèles) et celles qui seront exécutées dans un environnement réparti. Ceci rend impossible une adaptation utltérieure du parallélisme en fonction du matériel cible.

Enfin, les auteurs visaient principalement des applications de simulations de phénomènes physiques qui utilisent des structures de données régulières comme les tableaux. Le modèle proposé pour les objets parallèles (SPMD avec distribution des données) est donc bien adapté à l'écriture de telles applications ; il l'est beaucoup moins pour des applications utilisant des structures de données irrégulières voire dynamiques comme les listes, les arbres ou les graphes.

7. Synthèse

L'étude des trois outils présentés dans les sections précédentes et leur évaluation par rapport aux critères préalablement définis à la section 3 permet de tirer les conclusions suivantes :

– L'hétérogénéité est relativement bien traitée. Chaque outil permet d'exécuter des applications parallèles et distribuées dans un environnement hétérogène;

– Les performances sont en général bonnes. Dans la cas contraire, leur dégradation est liée essentiellement au déploiement d'une version privée des outils de programmation parallèles. Par exemple, des mesures comparant MPICH avec la version de MPI implémentée sur Globus ont montré des pertes de performances variant entre 20 % et 30 % pour des messages de moins de 32 Koctets [ABD 00];

– L'aspect dynamique est supporté par tous les outils étudiés au niveau de l'application et de l'exécution. Dans les trois cas, aucune reconfiguration n'est effectuée en cours d'exécution;

– La flexibilité est l'aspect le moins bien traité. Ceci est lié au fait que les trois environnements analysés, soit « ignorent » les outils de programmation parallèle existants (cas de NetSolve et de Cobra), soit les réimplémentent (Globus). Il convient ici de noter que, dans le cas de NetSolve, il est possible d'exécuter un programme parallèle écrit avec un outil de programmation parallèle donné. Toutefois, NetSolve n'aura pas connaissance de cette caractéristique et ne prendra donc aucune mesure afin de rendre cette exécution la plus efficace possible. Pour ce faire, le parallélisme doit être exprimé à l'aide du concept *Process Farming* proposé par les concepteurs de NetSolve. Il en est de même avec Cobra qui impose l'utilisation des objets parallèles pour reconnaitre et exécuter efficacement toute forme de parallélisme. Globus, quant à lui, propose une « boite à outils » permettant l'implémentation privée des outils de programmation parallèles susceptibles d'être utilisés par les utilisateurs de Globus : il s'agit de Nexus. A ce titre, la flexibilité de Globus n'est que partielle.

Le manque de flexibilité des outils présentés peut avoir des répercussions importantes sur l'efficacité de l'exécution. En effet, le fait que ces environnements ne gèrent efficacement que les outils de programmation parallèle qu'ils intègrent

empêche l'utilisation efficace de versions natives ou spécifiquement développées pour un hardware particulier. De même, de nouveaux outils apparaissant sur le marché ne pourront pas être utilisés efficacement et directement par ces environnements.

Afin de résoudre ces problèmes, nous proposons une solution essentiellement basée sur la configuration et l'exécution. En effet, partant de la constatation qu'il existe et existera plusieurs outils de programmation pour des applications HPC distribuées il n'est pas pertinent de lier un environnement de *metacomputing* à un (ou quelques) outil(s) particulier(s). Nous préférons concentrer nos efforts sur la mise au point de méthodes de configuration automatique de l'environnement pour l'exécution efficace de programmes HPC distribués. Pour atteindre cet objective, il est nécessaire que l'outil proposé ait une vision abstraite de ce qu'est une application HPC distribuée. Cette vision abstraite doit être suffisamment générale pour qu'elle puisse permettre, d'une part, de parler de toutes les applications HPC distribuées existantes ou à venir, et d'autre part être suffisamment spécifique pour configurer efficacement l'environnement en fonction de l'outil de programmation parallèle utilisé. C'est l'objectif que nous espérons atteindre grâce au projet WOS™ (*Web Operating System*).

8. Le projet WOS™

Le projet WOS™, est un projet en cours de développement aux Universités Laval (Canada), de Rostock (Allemagne) et *New South Wales* (Australie) [KRO 99]. Il a pour but général de développer un cadre universel pour l'utilisation des ressources du Web. Il est basé sur le concept d'évaluation dirigée par la demande (*demand-driven computation*). Les ressources (programme exécutable ou arguments d'un programme) sont gérées grâce à des structures de données appelée *entrepôts*. Le contenu d'un entrepôt peut être modifié dynamiquement en fonction des requêtes utilisateurs dans le but d'éviter, par exemple, la recherche de ressources dans des sites lointains lorsque cela n'est pas adéquat. Chaque site WOS™ inclut un entrepôt et un moteur de recherche capable de répondre à des requêtes en provenance des utilisateurs ou d'autres moteurs de recherche.

Les protocoles de communication sont l'élément central du WOS™. En effet, la communication entre les différents nœuds est réalisée à l'aide d'un protocole générique de services (WOSP) et d'un protocole simple de découverte et localisation des nœuds (WOSRP). Le protocole de services peut être « versionné » afin de supporter différents services spécialisés. Le WOS™ est donc un système totalement ouvert. Grâce à la notion de « version de WOSP », il est possible d'implémenter tous les services que l'on désire. Le protocole de découverte et localisation se base sur cette notion pour identifier les nœuds candidats à un service donné. En effet, un nœud est candidat s'il possède la bonne version du WOSP. Cela ne signifie pas forcément que ce nœud pourra fournir ce service mais

seulement qu'il est possible de le lui demander car il comprendra « de quoi on parle ». Un des avantages de cette approche est qu'un service (donc une version de WOSP) peut être une abstraction de services réels et donc représenter toute une classe de services réels existants ou à venir. Des travaux sont en cours afin d'implémenter des versions de WOSP pour différents services [KRO 99].

Nous nous proposons d'implémenter un service pour l'exécution d'applications HPC distribuées, supporté par le WOS™ : HP-WOSP. L'approche proposée consiste à intervenir durant la phase de configuration de l'environnement d'exécution mais à ne pas intervenir durant l'exécution du programme. Cette approche a deux avantages : d'une part elle permet d'utiliser n'importe quel outil existant pour écrire l'application et d'autre part, elle ne rajoute un surplus de travail, donc une perte de performance, que durant la phase d'initialisation de l'exécution. La réalisation de cette approche passe par la définition d'un service, donc d'une version spécialisée du WOSP, permettant au système WOS™ de demander l'exécution d'une application HPC distribuée quels que soit les outils utilisés pour écrire cette application. Un tel service fonctionnerait de la façon suivante : dans un premier temps à l'aide d'un protocole adéquat, une requête d'exécution d'un programme HPC distribué est envoyée au WOS™. Ce dernier, grâce à son protocole de découverte et localisation des nœuds WOSRP, recherche l'ensemble des nœuds candidats disponibles pour le traitement demandé. Cela revient à identifier les nœuds disponibles qui possèdent la bonne version du WOSP. Il s'agit dans notre cas du HP-WOSP. Une fois les nœuds trouvés, les WOSP localisés sur les différents nœuds se chargeront de configurer et de lancer l'exécution des différents éléments constitutifs de l'application à exécuter. Après le lancement de l'exécution, il n'y a plus aucune interaction entre le WOS™ et l'application, excepté dans le cas de situations particulières comme, par exemple, une exécution erronée ou la terminaison.

Le projet de réalisation d'un service HPC distribué dans le WOS™ en est à ses débuts et peu des notions présentées ci-dessus ont été réalisées. Toutefois une première version de WOSP permettant d'implémenter les concepts présentés ci-dessus est présentée dans [ABD 00].

9. Conclusion

Les besoins qui ont conduit au développement du calcul parallèle et distribué et ceux qui ont contribué à l'apparition du concept du *metacomputing* présentent plusieurs éléments de ressemblance. Dans le premier cas, le but était de répondre au besoin de plus en plus croissant en puissance de calcul. Dans le second, il s'agissait d'exploiter la puissance de calcul disponible sur les machines d'un grand réseau. Les problèmes abordés dans les deux cas sont assez similaires : partage de ressources, équilibrage de charges, distribution des données et des traitements, etc. Plusieurs solutions retenues dans le domaine du calcul distribué ont été reprises et

adaptées dans le contexte du *metacomputing*. Celui-ci présente, compte tenu des problèmes qu'il est sensé résoudre, une plus grande ouverture sur son environnement (hétérogénéité matérielle, logicielle et d'interconnexion) et sur les ressources qu'il est supposé gérer : ressources de traitement, de stockage, logiciels, etc. Ces constatations nous ont encouragé à prendre le chemin inverse en étudiant et adaptant les outils de *metacomputing* aux besoins du calcul parallèle de haute performance. Dans ce contexte, les réponses apportées par les outils de *metacomputing* sont souvent lourdes ou ignorent le « patrimoine » existant des outils de programmation parallèle.

Actuellement, différentes possibilités d'expression du parallélisme sont offertes par les outils de *metacomputing* : MPI, CC++, Fortran M, etc., dans le cas de Globus ; *Process Farming* et *Remote Program Call* bloquant ou non dans le cas de NetSolve ; enfin les objets parallèles dans le cas de Cobra. Ses approches restent cependant propriétaires. Ceci est essentiellement lié au fait que ces outils interviennent dans tout le cycle de vie du programme parallèle, soit depuis la phase de développement jusqu'à la phase d'exécution. Pour remédier à ce problème, l'idée est de limiter les interventions de l'environnement à ce qui est strictement nécessaire. L'environnement de *metacomputing* ne devrait idéalement prendre en charge que la phase de configuration. D'autre part, il devrait être suffisamment universel pour pouvoir supporter l'exécution de toute application parallèle quel que soit l'outil de programmation utilisé. Au même titre qu'un programmeur écrit un programme sans se préoccuper de l'architecture matérielle et logicielle de la machine cible, il devrait aussi écrire son application parallèle sans se préoccuper de l'environnement de *metacomputing* dans lequel s'exécutera son programme. Le projet WOS™ devrait permettre une plus grande autonomie de l'outil de programmation parallèle par rapport à l'environnement d'exécution et devrait garantir la performance livrée par l'application.

10. Références bibliographiques

[ABD 00] ABDENNADHER N., BABIN G., KROPF P., KUONEN P., « A Dynamically Configurable Environment For High Performance Computing », *High Performance Computing 2000*, Washinton DC, USA, April 2000, à paraître.

[ALE 96] ALEXANDROV, A., IBEL M., SCHAUSE K., SCHEIMANN C., « Superweb: Research issues in java-based global computing », *Actes du Workshop on java for computational science and engineering*, Syracuse University, décembre 1996.

[ALM 98] ALMOND, J., ROMBERG, M., « The UNICORE Project: Uniform Access Supercomputing over the Web », *Proceedings of Cray User Group Meeting*, Stuttgart, juin 1998.

[BAR 96] BARATLOO A., KARAUL M., KEDEM Z., WYKOFF P., « Charlotte: Metacomputing on the web », *9th conference on parallel and distributed systems*, 1996.

[BUY 99] BUYYA R., *High Performance Cluster Computing*, Programming and Applications, vol. 1 et 2, Prentice Hall, 1999.

[CAG 99] CAGNARD P.-J., Etat de l'art des langages parallèles, rapport interne n° 120, mai 1999, EPFL-DI-LITH.

[CAS 97] CASANOVA H., DONGARRA J., « NetSolve : A Network Server for Solving, Computational Science Problems », *Proceedings of Supercomputing 96*, Pittsburgh. Aussi publié dans *International Journal of Supercomputer, Applications and High Performance Computing*, vol. 11, n° 3, 1997, p. 212-223.

[CAS 98a] CASANOVA, H., DONGARRA, J., KARAINOV, A. WANIEWSKI, J., Users' Guide to NetSolve, version 1.2.beta, octobre 1998, http://www.cs.utk.edu/netsolve.

[CAS 98b] CASANOVA, H., DONGARRA. J., « Applying NetSolve's Network-Enabled Server », *IEEE Computational Science & Engineering*, juillet-septembre 1998, p. 57-67.

[CZA 98] CZAJKOWSKI, K., FOSTER, I., KARONIS, N., KESSELMAN, C., MARTIN, S., SMITH, W., TUECKE, S., « A Resource Management Architecture for Metacomputing Systems », *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[FIT 97] FITZGERALD, S., FOSTER, I., KESSELMAN, C., VON LASZEWSKI, G., SMITH, W., TUECKE S., « A Directory Service for Configuring High-Performance Distributed Computations », *Proc. 6th IEEE Symp. on High-Performance Distributed Computing*, 1997, p. 365-375.

[FOS 96] FOSTER, I., KESSELMAN, C., TUECKE, S., « The Nexus Approach to Integrating Multithreading and Communication ». *Journal of Parallel and Distributed Computing*, vol. 37, 1996, p. 70-82.

[FOS 97a] FOSTER, I., GEISLER, J., KESSELMAN, C., TUECKE S., « Managing Multiple Communication Methods in High-Performance Networked Computing Systems ». *Journal of Parallel and Distributed Computing*, vol. 40, 1997, p. 35-48.

[FOS 97b] FOSTER, I., KESSELMAN, C., « Globus, A Metacomputing Infrastructure, Toolkit ». *International Journal of Supercomputer Applications*, vol. 11, n° 2, 1997, p. 115-123.

[FOS 98a] FOSTER, I., VON LASZEWSKI, G., Usage of LDAP in Globus, http://www-fp.globus.org/documentation/papers.html

[FOS 98b] FOSTER, I., GEISLER, J., GROPP, W., KARONIS, N., LUSK, E., THIRUVATHUKAL, G., TUECKE, S., « Wide-Area Implementation of the Message Passing Interface ». *Parallel Computing*, vol. 24, n° 12, 1998, p. 1735-1749.

[FOS 99] FOSTER, I., KESSELMAN, C., *The Grid : Blueprint for a new computing infrastructure*, San Francisco, CA, USA, Morgan Kaufmann, 1999.

[GEI 97] GEIB, J.M., GRANSART, C., MERLE, P., *CORBA : Des concepts à la pratique*, InterEditions, Col. Informatique, 1997, ISBN 2-225-83046-0.

[GRI 94] Grimshaw, A., Wulf W, French J., Weaver A. et Reynolds P., A synopsis of the Legion project, Technical Report CS-94-20, juin 1994, University of Virginia.

[KRO 99] P. Kropf. « Overview of the WOS project », Advanced Simulation Technologis Conferences (ASTC 1999), San Diago, CA, USA, Avril 1999

[LAS 00] Computing Portals projects, project survey, maintained by VON LASZEWSKI G., http://www.computingportals.org/

[MPICH] MPICH-A Portable Implementation of MPI, http://www-unix.mcs.anl.gov/mpi/mpich/index.html

[NEX 94] Nexus : Runtime Support for Task-Parallel Programming Languages. Technical Report, 1994, Mathematics and Computer Science Division, Argonne National Laboratory.

[PRI 98] PRIOL, T., RENE, C., « Cobra : a CORBA-compliant Programming Environnement for High-Performance Computing », *Proc. of Europar'98*, Shouthampton, UK, septembre 1998, p. 114-1122.

[VAH 98] VAHDAT A. *ET AL.*, « WebOS : Operating System For Wide Area Applications », *The 7th IEEE Symposium on High Performance Applications*, juillet 1998.

Pierre Kuonen *a obtenu le diplôme d' ingénieur électricien del' École Polytechnique Fédérale de Lausanne (EPFL) en 1982. De 1982 à 1988, il a travaillé dans l' industrie, tout d' abord en Afrique comme ingénieur de terrain dans l' industrie pétrolière avec la compagnie Schlumberger Oversea S.A., puis comme ingénieur de développement (CAO) à Genève avec la compagnie Charmilles Technologies S.A. En 1988 il a rejoint le laboratoire d' informatique théorique dd' EPFL où il a obtenu, en 1993, le titre de docteur ès science en informatique. Depuis cette date il travaille en tant que collaborateur scientifique dans ce même laboratoire et dirige le Groupe de Recherche en Informatique Parallèle (GRIP). Il est chargé du cours « programmation parallèle » et il dirige plusieurs projets de recherche nationaux et européens dans le domaine du calcul de haute performance parallèle.*

Nabil Abdennadher *a recu son diplome d' Ingénieur en Informatique dd' Ecole Nationale des Sciences de l' Informatique (ENSI) de Tunis et son doctorat de l' Université de Valenciennces (France). De 1992 à 1998, il était Enseignant Universitaire à Université Tunis II. Depuis 1999, il est collaborateur scientifique à l' Ecole Polytechnique Fédérale de Lausanne (EPFL). Ses travaux de recherche portent sur les outils de Metacomputing, de programmation parallèle et du Web-Supercomputing. Il a été aussi consultant à l' Institution de Recherches en Sciences Informatiques etTelecommuication (IRSIT. Tunis) pour la parallélisation d' applications spécifiques (reconnaisaance de formes et de caractères). Il est fondateur d' unestart-up qui opère dans le domaine du calcul parallèle et distribué.*

Gilbert Babin *est diplômé de l' Université de Montréal (Canada), où il a obtenu sonB.Sc. et son M.Sc. en 1986 et 1989, respectivement. En 1993, il complétait un Ph.D. en Decision Sciences and Engineering Systems au Rensselaer Polytechnic Institute (Troy, New York, États-Unis). Sa thèse lui a valu le Del and Ruth Karger Award, prix remis à la meilleure thèse de ce département. Aujourd' hui, il est professeur agrégé au département d' Informatique de l' Université Laval (Sainte-Foy, Canada). Ses travaux de recherche ont*

été publiés dans les transactions de l'Institute for Electrical and Electronic Engineers (IEEE). Ses intérêts de recherche incluent l'intégration de systèmes d'information hétérogènes, l'utilisation efficace des ressources de l'Internet et le Web Operating System (WOS™).

Titre abrégé :

Le metacomputing au service du HPC

Titre anglais :

Metacomputing for High Performance Computing