I-Cluster : Agrégation des ressources inexploitées d'un intranet et exploitation pour l'instanciation de services de calcul intensif

Bruno Richard

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

THESE

Pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : « Informatique : Systèmes et Communications »

Préparée au laboratoire **IMAG Informatique et Distribution**,
dans le cadre de **l'Ecole Doctorale « *Mathématiques, sciences et technologie de l'information, Informatique* »**

Présentée et soutenue publiquement par

Bruno Richard

Le 12 décembre 2003

Titre :

I-Cluster : Agrégation des ressources inexploitées d'un intranet et exploitation pour l'instanciation de services de calcul intensif

Directeur de thèse : Brigitte Plateau

Co-directeur de thèse : Jacques Briat

JURY :

Mr Jacques Mossière	, Président
Mr Luc Bougé	, Rapporteur
Mr Frédéric Desprez	, Rapporteur
Mme Brigitte Plateau	, Directeur de thèse
Mr Jacques Briat	, Co-Directeur de thèse
Mr Sylvain Sadier	, Examinateur

« Droit devant soi on ne peut pas aller bien loin »

Le petit Prince, Antoine de Saint-Exupéry.

Remerciements

Cette thèse s'est déroulée selon un cadre inhabituel : Etant ingénieur chez Hewlett-Packard Grenoble depuis 8 ans, Sylvain Sadier, alors Directeur de mon laboratoire, m'a proposé d'effectuer une thèse afin d'affirmer la position de l'équipe au sein du tissu académique Français et, plus spécifiquement, Grenoblois.

Je cherchais alors depuis plusieurs années à pouvoir trouver un financement pour un tel projet personnel, aussi j'ai immédiatement saisi cette opportunité.

J'ai alors effectué mon travail de thèse au sein du laboratoire Informatique et Distribution (ID-IMAG) où j'ai passé trois années très instructives, sur une thématique technique très intéressante, et où j'ai pu apprécier l'équipe.

Après ces trois années, j'ai de réels remerciements à exprimer à de nombreuses personnes :

Tout d'abord à **Sylvain Sadier**, qui a été l'initiateur de ma thèse et qui m'a permis de la mener à bien tout en restant à mon poste de *Research Program Manager* dans l'équipe HP Labs Grenoble,

à **Dominique Vicard**, superviseur motivant s'il en est, pour avoir toujours su garder un œil positif sur mon travail,

à **Brigitte Plateau**, qui m'a accueilli au sein de son laboratoire pendant ma thèse, qui en a été la Directrice, et dont le soutien pendant les périodes difficiles m'a été précieux,

à **Christophe Grosthor**, qui, malgré les milliers de kilomètres qui nous séparent, est resté un de mes principaux inspirateurs,

à **Luc Bougé** et **Frédéric Desprez** qui ont accepté de rapporter sur ma thèse, pour l'intérêt qu'il ont porté à mon travail et pour les améliorations qu'ils m'ont permis de faire sur mon manuscrit,

à **Jacques Briat**, co-directeur de ma thèse, pour l'étendue de ses connaissances qu'il m'a faites partager,

à **Jean-François Méhaut**, pour son œil averti et ses nombreuses relectures, qui m'ont permis de correctement structurer ma thèse,

à **Jacques Mossière**, Président de mon jury de thèse,

à **Philippe Augerat**, pour ces nombreuses discussions qui m'ont beaucoup apporté ainsi que pour sa volonté à faire avancer les choses,

à **Jean-François Larvoire**, **Denis Chalon**, et **Denis Flaven**, noyau de compétences de HP Grenoble, que j'ai maintes fois mis à contribution,

à **Pierre Lombard** pour sa patience et son expertise dans de nombreux domaines, et pour ces trois années où nous avons partagé le même bureau,

à **Nicolas Maillard**, pour ses efforts sur le benchmark Linpack avec la grappe I-Cluster, et sa coopération très active dans la mise en œuvre du projet vCluster au laboratoire CPAD au Brésil, ainsi que pour les nombreux échanges productifs qu'il m'a permis,

à **Stéphane Martin**, pour les nombreux moments agréables que nous avons partagé, et sa contribution technique lors des expérimentations sur les grappes,

aux personnes ayant permis la mise en œuvre des expériences discutées dans ce manuscrit : **Jacques Misselis** ; **Heinz Schmidt** de HP Allemagne ; **Cirano Silveira**, **Joao Jornada**, **Reynaldo Novaes** et **Indiara Palavro**.de HP Brésil ; **César De Rose** du laboratoire CPAD de la PUCS ; **Simon Derr**, **Céline Robert** du laboratoire ID-IMAG pour leurs effort lors de la mise en œuvre de la grappe I-Cluster ; **Bruce Greer** et **Greg Henry** de Intel ; **Antoine Petitet** de Sun Labs,

à mes collègues de **HP Labs Grenoble**,

à **Cécile Billon** pour son support documentaire sans faille,

aux membres du **laboratoire ID-IMAG**, avec qui j'ai eu du plaisir à travailler : Les permanents, les thésards, les stagiaires, et les très précieuses assistantes,

à tous ceux qui ont supporté et encouragé mon effort, mes amis, ma famille,

et à **Nathalie**, mon épouse, pour beaucoup de choses qui ne regardent que nous.

Table des matières

Table des figures

1. Introduction

Le calcul scientifique et technique demande de grandes puissances de traitement informatique. Par le passé de nombreuses architectures ont été proposées par les constructeurs afin de remplir les besoins des utilisateurs. Plus récemment, les grappes de calcul ont connu un essor important grâce aux rapports performance/prix qu'elles permettent d'atteindre. Ces grappes ont typiquement une architecture en grappe de multiprocesseurs (CLUMPs) interconnectés par des réseaux à haut débit et à faible latence. Ces dernières années, nous assistons à la transformation de ces architectures en grappes de monoprocesseurs grand public, interconnectés par des réseaux Ethernet standard.

Les grappes de calcul basées sur des machines et des réseaux communs fonctionnant dans de bonnes conditions de performances, on peut désormais s'intéresser aux machines disponibles sur un intranet d'entreprise afin de les agréger en *grappes virtuelles*. Notre objectif ici est de développer une infrastructure logicielle distribuée permettant d'exploiter la puissance disponible sur les machines des utilisateurs de l'intranet lors de leurs périodes d'inactivité.

L'étude faite ici essaiera de montrer la faisabilité d'une architecture basée sur l'exécution de logiciel réparti sur les machines d'un intranet, permettant de tirer parti de manière transparente des ressources inexploitées d'un réseau d'entreprise, telles que les PC des utilisateurs, afin de les utiliser pour exécuter des services de calcul intensif. Nous détaillerons en particulier les mécanismes permettant l'identification des ressources de calcul disponibles, la détection de leurs périodes d'inexploitation à l'aide de leur profil d'utilisation et d'un système de prédiction, leur agrégation en grappes virtuelles ainsi que l'isolation de l'exécution de code afin de protéger les parties utilisateurs des parties calcul.

1.1. Propriétés visées

Notre étude porte sur un système possédant les propriétés suivantes :

- Nous voulons pouvoir exécuter des tâches parallèles avec *grain de calcul fin*. Nous ne nous étendrons pas ici sur la terminologie afférente à la programmation parallèle et distribuée, et nous nous limiterons à dénommer *tâches à grain fin* les tâches parallèles distribuées concourantes ou interdépendantes par opposition aux tâches indépendantes. Il est à noter toutefois que le calcul à grain fin

est une classe de calcul exigeante pour le système, et le cas le plus délicat à supporter. Notre infrastructure a la responsabilité d'identifier les machines disponibles sur le réseau permettant de résoudre un problème dans les meilleures conditions possibles, en tenant compte des caractéristiques du travail demandé par l'utilisateur, des caractéristiques des machines disponibles sur le réseau, de leur charge, ainsi que de leur topologie d'interconnexion.

- *Résilience*. Dans le cas de déconnexions des machines (volatilité), de déplacements de machines dans la topologie du réseau ou de partitionnements du réseau, notre système devra détecter et prendre en compte ces évènements, et y réagir de manière à conserver l'équilibre global du système. Notons que ces problèmes font partie intégrante des réseaux basés sur des technologies sans fil, et l'on peut s'attendre à trouver de plus en plus de déploiements d'intranets sans fil, d'où l'intérêt de supporter les contraintes dès l'architecture de notre solution.
- *Mode pair-à-pair*. La thématique de recherche du laboratoire HP Labs Grenoble est autour des « *écosystèmes d'ustensiles Internet* ». Il apparaît important de pouvoir se détacher du classique serveur central au profit d'une intelligence communautaire décentralisée. Pour les traitements et algorithmes de groupe nous nous sommes donc concentrés sur des mécanismes fonctionnant en mode pair-à-pair.
- *Auto-organisation*. Les mécanismes mis en œuvre « vivent » de manière répartie et communautaire. La complexité de gestion des systèmes distribués, qui plus est sans serveur central, nous impose d'imaginer un système qui s'organise automatiquement en fonction des caractéristiques changeantes de l'environnement.

1.2. Contraintes

Un objectif de notre étude étant de pouvoir utiliser le système développé de manière productive voire industrielle, il est important qu'il puisse permettre une utilisation efficace, productive et agréable tant pour les utilisateurs que pour les administrateurs du système. Les contraintes que nous nous imposons donc pour cette étude sont les suivantes :

- *Réutilisation des codes existants*. Les travaux exécutés font partie du domaine du métier des utilisateurs. Il ne s'agit donc pas de demander des modifications de ces logiciels pour pouvoir exécuter ces travaux. Des applications parallèles fonctionnant sur une grappe classique Linux devront pouvoir s'exécuter sans modification ni recompilation. I-Cluster est destiné à permettre l'exploitation des ressources matérielles de l'intranet pour l'exécution des tâches de supercalcul. Nous ne proposons pas une nouvelle manière de distribuer des codes de manière répartie. Au contraire, nous développons une plate-forme d'exécution permettant d'utiliser les systèmes actuels de gestion de tâches parallèles (PBS [PBS], Java Runtime Environment), ainsi que les langages de programmation (C, C++, Java, FORTRAN), les bibliothèques métier (par exemple BLAS [LHK+79] pour l'algèbre

linéaire) et les paradigmes de distribution, de parallélisme et d'échange de messages disponibles (MPI [Mpi93], PVM [Sun90], RMI [BDV+98][Sun96], CORBA [Omg92]). Nous nous limitons toutefois au support des environnements et outils basés sur le système d'exploitation Linux. Ce choix n'est pas trop limitatif, car la plupart des outils de calcul haute performance, de programmation et d'exploitation parallèle disposent de portages efficaces fonctionnant sous Linux. De cette manière, les codes parallèles hérités du passé peuvent tourner sans modification sur I-Cluster, sous réserve qu'ils soient prévus pour tourner sur une grappe Linux. Cette contrainte est très acceptable pratiquement, car Linux est un système d'exploitation de grappe très répandu, aussi les applications de supercalcul sont fréquemment portées sous cet environnement.

- *Transparence utilisateur*. Nous proposons un système capable d'utiliser les machines inexploitées de l'intranet. Pour autant, il ne s'agit pas de demander leur complaisance aux utilisateurs de ces machines ni de leur imposer de dédier explicitement leur machine lorsqu'ils ne l'utilisent pas. Nous devrons donc installer nos composants logiciels sans perturber le fonctionnement normal de la machine (en particulier pas de repartitionnement du disque dur), détecter automatiquement les périodes de disponibilité de chaque machine et exécuter les tâches de calcul sans perturbation pour les tâches de l'utilisateur.
- *Protection*. La sécurité par isolation de l'environnement d'exécution, des travaux et des sessions entre les modes de calcul d'une machine et les modes d'utilisation normale par son possesseur. Il ne sera donc jamais possible à l'utilisateur d'interagir ou de surveiller des travaux de calcul effectués à distance sur sa machine, ni au système de calcul distribué d'accéder aux données de l'utilisateur.
- *Passage à l'échelle.* Afin de pouvoir offrir un réel intérêt dans le cadre d'un intranet, notre système doit pouvoir gérer l'ensemble des machines disponibles sur l'intranet, afin de pouvoir identifier les grappes virtuelles (constituées de machines utilisateurs en jachère) correspondant au mieux aux travaux soumis de manière distribuée. Le nombre de machines prises en compte sera donc de l'ordre de 10000, ce qui correspond à un intranet moyen.

Nos travaux ont été effectués dans le cadre de ***I-Cluster***, [RAu02a] qui est un projet de recherche mené en partenariat par HP Labs Grenoble et ID-IMAG (projet INRIA Apache).

1.3. Structure du document

Notre étude aurait demandé un effort très important pour que toutes les propriétés recherchées soient pleinement analysées et implémentées. Nous avons donc restreint l'étude de manière à nous concentrer sur quelques points dont le développement est particulièrement intéressant.

L'objectif de notre effort, ainsi que les contraintes que nous nous posons, sont exposés dans le présent chapitre 1.

Un état de l'art permettant de présenter les éléments historiques qui serviront de base au projet est disponible dans le chapitre 2.

Dans le chapitre 3, nous présenterons le système I-Cluster, les composants qui le constituent et la manière dont il se met en place lors de l'exécution. L'architecture générale du système sera exposée dans le chapitre 3.1. Les principaux composants seront alors détaillés :

- Le bac à sable d'exécution de tâches (chapitre 3.2), à savoir d'une part les mécanismes de détection du profil d'utilisation de chaque machine et de prédiction de ses fenêtres de tir ; et d'autre part des outils permettant l'isolation de code et de données, qui permet de garantir l'isolation et l'intégrité des données et programmes utilisateur et calcul.
- le *nuage I-Cluster*(chapitre 3.3). Cette base de données distribuée en mode pair-à-pair utilisant des algorithmes bavards permet de connaître l'état de disponibilité des ressources de calcul disponibles sur l'intranet. Cette architecture permet de respecter des contraintes de volatilité des machines et de passage à l'échelle massif du nombre de participants.
- Nous verrons ensuite (chapitre 3.4) comment les tâches de calcul sont modélisées et décomposées afin de pouvoir, lors d'une requête utilisateur, utiliser les informations disponibles dans le nuage I-Cluster pour trouver les machines permettant de satisfaire la requête.

Des résultats expérimentaux permettront ensuite de valider l'approche prise dans notre étude : Le chapitre 3.5 présentera une manipulation effectuée sur 225 machines de bureau standard interconnectées en Ethernet, et qui montre qu'une haute performance en supercalcul est possible même sur une telle infrastructure. On pourra donc affirmer dans la suite de notre étude que la recherche de machines inexploitées sur un intranet peut, sous certaines conditions, permettre l'agrégation des ressources afin d'effectuer des calculs à haute performances sur les grappes virtuelles ainsi constituées.

Le chapitre 3.6 présentera une analyse du parc de machines disponibles sur l'intranet de Hewlett-Packard. Cette analyse révèle qu'un pourcentage important des machines peuvent être utilisées comme composants d'un système de calcul intensif distribué. Ceci justifie l'approche de notre étude de recherche de grappes virtuelles parmi les machines en jachère d'utilisation sur un intranet.

Le chapitre 4 nous permettra enfin de conclure sur cette étude.

1.4. Contributions

Les principales contributions de ce travail portent sur les aspects suivants :

- Un travail d'analyse des systèmes d'isolation de code actuels et de reformulation du besoin, aboutissant à la réalisation d'un prototype basé sur le suivi de profil d'utilisation des machines, la prédiction

des fenêtres de tir et le basculement automatique entre modes utilisateur et calcul d'une machine.

- Un module logiciel d'instrumentation des capacités matérielles d'une machine, basées sur la technologie du *System Management BIOS*.
- Une étude comparative et taxonomique des systèmes pair-à-pair, menant à la conception et au développement du système de *nuage I-Cluster*, permettant la gestion d'une base de donnée distribuée de ressources dans le contexte d'un intranet équipé d'un grand nombre de machines très volatiles.

Ces contributions ont fait l'objet de quelques publications :

- Richard B., Augerat P., Maillard N., Derr S., Martin S., Robert C., "I-Cluster: Reaching TOP500 performance using mainstream hardware", HP Labs technical report HPL-2001-206, http://www.hpl.hp.com/techreports/2001/HPL-2001-206.html, 2001.
- Bruno Richard, "I-Cluster : Un environnement d'exploitation des jachères de calcul en intranet", ACM/SIGOPS JTE Cluster Computing, Besançon, 2001.
- Dejan S. Milojicic, Vana Kalogeraki, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, John Sontag, "Peer-To-Peer (P2P) Technology", HP Labs technical report HPL-2002-57, http://www.hpl.hp.com/techreports/2002/HPL-2002-57.html, 2002. Soumis à ACM Computing Surveys.
- Bruno Richard, Philippe Augerat, "I-Cluster: Intense computing with untapped resources", Proceedings of the 4th International Conference on Massively Parallel Computing Systems, 2002.
- Bruno Richard, "I-Cluster: the Execution Sandbox", Proceedings of the IEEE International Conference on Cluster Computing 2002.
- C. De Rose, F. Blanco, N. Maillard, K. Saikoski, R. Novaes, O. Richard, B. Richard, "The Virtual Cluster: a Dynamic Environment for Exploitation of Idle Network Resources", Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2002), Vitória, Brazil, 2002.
- Bruno Richard, Donal Mac Nioclais, Denis Chalon, "Clique: A transparent, Peer-to-Peer collaborative file sharing system", HP Labs Technical Report HPL-2002-307, 2002.
- Bruno Richard, Philippe Augerat, "Grappe virtuelle I-Cluster : Gestion distribuée des ressources", Actes de l'Ecole d'hiver GRID 2002, Aussois, France.
- Bruno Richard, Denis Chalon, Donal Mac Nioclais, "Clique: A transparent, peer-to-peer replicated file system", Proceedings of the 4th IEEE International Conference on Mobile Data Management, Melbourne, Australia, January, 2003.

Nos éléments de recherche essentiels ont été brevetés :

- Jean-Francois Larvoire, Bruno Richard, "Backward Compatible Boot System for a Computer", European Patent Office application number 01410082.0, 2001.
- Bruno Richard, "Peer-to-Peer network", European Patent Office application number 01410124.0, 2002.
- Bruno Richard, Denis Chalon, "Distributed File System", European Patent Office application number 2354108.9, 2002.
- Bruno Richard, "Process for data distribution through a network", European Patent Office application number 02354123.8, 2002.

2. Contexte technologique et scientifique : Exploitation des ressources de calcul distribuées

De nombreux projets ont été réalisés sur le thème de l'exploitation de machines standard pour exécuter des applications de calcul intensif. Nous verrons que de nombreux projets ont vu le jour dès la fin des années 1980, ayant pour base des stations de travail inexploitées fonctionnant sous Unix. Ces architectures permettaient d'exploiter au mieux des machines coûteuses en déléguant les tâches de calcul des machines les plus chargées vers des machines moins actives, allant jusqu'à exploiter des périodes de micro-inactivité de l'ordre de la milliseconde.

L'objectif de ces projets était de pouvoir mieux exploiter les possibilités matérielles de machines haut de gamme, afin de pouvoir obtenir des taux d'utilisation allant de 95 à 100% et donc d'amortir les coûts matériels.

Vers le début des années 1990, les ordinateurs de bureau de type IBM PC et compatibles se sont démocratisés, et l'effet d'échelle a provoqué une chute importante de leur coût d'achat. Le rapport performance/prix de ces machines était bien supérieur a ceux de stations de travail ou de calculateurs spécialisés [BGr01]. Cependant, leurs capacités modestes en limitaient l'accès pour des applications nécessitant des performances importantes. L'algorithmique parallèle a contribué à l'affranchissement vis-à-vis des limites imposées par les capacités des ordinateurs en termes de puissance de processeur ou de taille de mémoire. De nombreux projets ont alors essayé d'agréger les capacités d'un grand nombre de machines afin de pouvoir exécuter des applications nécessitant des ressources importantes, tout en conservant un rapport performance/prix assez haut. Les technologies des grappes de calcul sont nées de ces projets et restent aujourd'hui un moyen d'accéder à de hauts niveaux de performance sans nécessiter des investissements trop importants. En particulier, les systèmes à image unique (*Single system Image*) forment une classe de systèmes permettant la transparence du matériel pour l'utilisateur final, au prix d'un système et d'un noyau plus lourds.

Vers le milieu des années 1990, quelques projets ont essayé de tirer parti des ressources inexploitées disponibles sur les machines connectées à Internet ; Ces projets, basés sur un serveur central distribuant du travail en mode maître-esclave aux machines disponibles, ont donné naissance au *Metacomputing* ou *calcul global*. Certains de ces projets, tels SETI@home [And+99][And+02],

mettent en œuvre plusieurs millions de machines clientes et permettent d'obtenir des puissances de calcul énormes – 20 Tflop/s pour SETI@home, le même ordre de grandeur que les plus puissants superordinateurs. Cependant, de telles infrastructures ne permettent une grande performance que si les travaux à exécuter ne sont pas interdépendants, c'est-à-dire qu'ils ne nécessitent pas de communications entre les différents sites de calcul. Ceci est efficace pour des applications multiparamétriques ou a gros grain de calcul, mais ne permet pas d'effectuer du calcul parallèle avec interdépendances entre les calculateurs.

Le calcul en grille (*grid Computing*) est aussi apparu au milieu des années 1990. Il permet le partage, la sélection et l'agrégation de ressources de calculs variées (superordinateurs, grappes de calcul, systèmes de stockage) et géographiquement distribuées. Ces ressources sont fédérées et présentées comme un système unifié permettant de résoudre des calculs à grande échelle ainsi que des applications manipulant intensivement des données (modélisation moléculaire, physique des hautes énergies). Le principe du calcul en grille repose sur l'analogie avec le réseau électrique, où les générateurs sont distribués, et sur lequel on peut brancher un appareil utilisateur afin d'accéder à la puissance électrique sans avoir à se soucier de sa provenance. Le calcul en grille est apparu après que la *National Science Foundation* (NSF - agence du gouvernement Américain responsable du soutien de la science et de la recherche) ait réalisé que le coût de maintenance d'un centre de calcul était tel que la performance de la machine doit être maximale pour compenser ces coûts fixes. En conséquence, pour être compétitif, un centre de calcul doit être l'un des plus gros ordinateurs du monde. Fort de cette conclusion, le nombre de centres de calcul du NSF a été réduit à deux en 1999, et des investissements importants ont été faits afin de développer les technologies de calcul en grille afin de tirer parti au mieux de nombre réduits de centres de calcul, mais disposant de capacités extrêmement importantes.

Depuis 2000, nous assistons à une arrivée en masse de *systèmes pair-à-pair*. Le pair-à-pair est une classe de systèmes auto-organisants qui permettent de réaliser une fonction globale de manière décentralisée, en tirant parti du partage des ressources disponibles sur un grand nombre d'extrémités de l'Internet. Ces systèmes ont pour particularité de fonctionner sur une notion de contrat communautaire global, ou chacun va coopérer pour le partage de ressources telles que fichiers, puissance de calcul, stockage ou bande passante.

Quelques systèmes de gestion de ressources de calcul distribuées sont présentés ci-dessous. Nous avons partagé ces systèmes en deux classes : Ceux permettant l'exploitation de machines pendant leur périodes d'inexploitation (jachères) et ceux permettant l'exploitation de ressources dédiées telles que des grappes de calcul.

2.1. Exploitation des machines en jachère

L'exécution de travaux parallèles sur un groupe de machines est parfois gérée en travail d'arrière-plan. Chacune des machines est affectée prioritairement à l'exécution des travaux de son utilisateur, et ne partage que les cycles de calcul inexploités. L'utilisateur autorise l'accès à ces cycles de calcul si ce choix n'influe pas significativement sur sa propre productivité.

La meilleure façon de s'assurer que l'utilisateur n'est pas impacté par des travaux d'arrière plan est de n'utiliser que les machines inactives[1], plutôt que d'essayer d'exploiter les périodes de micro-inactivité des machines. Cette approche, bien que conservatrice, permet de maintenir un bon équilibre entre les diverses machines participant à un calcul parallèle. En effet, nous verrons plus loin que cet équilibre est indispensable à une bonne efficacité de la grappe lors de calculs en grain fin.

D'autre part, la disponibilité de nombreuses machines sur le réseau permet d'en trouver facilement qui sont *inutilisées*. Il est difficile de définir exactement ce que signifie *inutilisée*, mais des études [Nic87][JXT93][KRR95] ont montré que 60 à 90 % des machines d'un réseau typique peuvent être inutilisées à n'importe quel instant, même pendant les périodes les plus actives de la journée.

L'utilisation des machines inexploitées requiert plusieurs éléments :

- La possibilité d'identifier les périodes d'inactivité. La détection des entrées clavier ou souris permet d'en faire une première analyse, ainsi que la charge du processeur et les sessions réseau ouvertes.
- La possibilité pour l'utilisateur de reprendre la main sur sa machine à son retour. Si les travaux sont structurés en tâches indépendantes il devient possible de migrer le travail en cours sur une autre machine, comme c'est le cas avec Condor, de tuer le travail et de le redémarrer sur une autre machine, comme avec Javelin, ou encore de *geler* le travail en cours afin de le redémarrer dans l'état où il était lors de la prochaine période d'inactivité comme pour SETI@home.
- L'isolation des codes et données utilisateur par rapport à ceux nécessaires aux calculs communautaires. Un mécanisme de bac à sable permettra cette isolation.

La mesure de l'activité des machines est souvent réactive et ne se base que sur l'analyse de son état courant. Par exemple, Sprite considère un nœud comme inactif si le taux d'utilisation du processeur est de moins de 1% et si le clavier a été inutilisé pendant 30 secondes [Dou90]. Avec Piranha, c'est après 5 minutes d'inactivité clavier qu'une machine sera considérée comme inactive, si la charge du processeur au bout de 1, 5 et 10 minutes doit être inférieure à 4, 0.3 et 0.1 % [GKa92].

Dans notre cas, nous utilisons un mécanisme qui permet de changer d'état la machine entre son contexte utilisateur normal et un contexte d'exécution de calculs. Ce changement est assez lourd et prend plusieurs dizaines de secondes, il n'est donc pas question de provoquer un tel changement de manière invasive pour l'utilisateur. C'est pourquoi nous avons mis au point un système de capture et d'analyse du profil d'utilisation de la machine, qui nous permet de prédire les fenêtres de tir possibles (la nuit, le week-end, lors d'absences régulières) pendant lesquelles on pourra provoquer le changement de contexte pour effectuer des calculs, puis retourner dans le mode normal d'exécution avant le retour de l'utilisateur.

---

[1] Nous utilisons dans notre étude et donc dans ce document le terme « jachère » pour référencer les machines temporairement inexploitées d'un intranet. La jachère est un terme technique issu de l'agriculture ancienne, et se pratiquait notamment dans le cadre de l'assolement triennal. La sole réservée chaque année à la jachère servait de pâture aux troupeaux.

Cet outil d'analyse de profil et de prédiction des fenêtres de tir possibles n'a pas été, à notre connaissance, été envisagé par de précédents projets.

L'exploitation des jachères de calcul a été analysée par les projets NOW, Condor, Butler, Stealth, Process servers, DAWGS. Condor et DAWGS font de la migration de travaux lorsque les machines supportant l'exécution deviennent trop chargées. Butler tue les travaux et les redémarre sur un autre nœud. Process server et Stealth changent la priorité des travaux de manière dynamique.

Pour l'exécution de codes de calcul sur la machine d'un utilisateur, plusieurs solutions sont possibles. Beaucoup de projets se sont basés sur des architectures où les travaux se font en tâche de fond, avec une basse priorité. C'est le cas pour NOW, Condor (qui tue le processus local lors du retour de l'utilisateur et le relance sur une autre machine depuis son dernier point de gel/reprise), Piranha (qui tue purement et simplement le processus lors du retour de l'utilisateur, comme c'est le cas pour I-Cluster), Sprite ou Locus (qui migrent le processus lors du retour de l'utilisateur).

Les approches basées sur des tâches de fond ont l'inconvénient de mobiliser des ressources du système. En effet, même si les travaux de calcul se font en tâche de fond, des structures de données sont utilisées par le système d'exploitation, de l'espace mémoire est consommé, de l'espace de pagination, etc.

Un utilisateur averti ne sera donc jamais heureux qu'une entité logicielle se permettre d'utiliser sa machine dans son dos. De plus, ce type d'évènement peut se produire à un moment où l'utilisateur a un besoin critique de sa machine (exemple classique : dix minutes avant une présentation lorsqu'il a besoin de ses transparents).

Nous apportons deux solutions novatrices dans le cadre du présent travail :

- La détection/anticipation du profil d'utilisation de la machine, qui nous permettra de prédire sans perturbation pour l'utilisateur les périodes pendant lesquelles la machine est régulièrement disponible (la nuit, le week-end), et qui nous permettent d'établir des fenêtres de tir pendant lesquelles la machine sera disponible pour faire des calculs,
- Le changement de mode, qui nous permettra de basculer un ordinateur donné entre son mode utilisateur où le fonctionnement reste classique, et son mode calcul pendant lequel il se verra donner une personnalité de machine de calcul à distance basée sous Linux.

Nous aurions pu nous limiter à une gestion du profil associée à un logiciel de sauvegarde d'écran, afin d'anticiper les fenêtres de tir en nous basant uniquement sur les périodes pendant lesquelles la sauvegarde d'écran est active. Ceci aurait simplifié notre gestion de la détection des fenêtres de tir d'une machine. Cependant cette information ne nous fournit des informations que sur la présence de l'utilisateur, et non pas sur l'état des ressources en cours d'exploitation et la charge des tâches utilisateur, et ne constitue donc pas un indicateur fiable de disponibilité de la machine.

Une approche intéressante (pour sa simplicité de mise en oeuvre) dans *Bayanihan* [Sar98] permet l'utilisation de Java pour effectuer des calculs sur des machines volontaires, en utilisant la machine virtuelle Java comme protection.

2.1.1. NOW

Le projet NOW a été développé par l'université de Berkeley au milieu des années 1990. NOW visait à permettre l'utilisation d'un réseau de stations de travail (*Network Of Workstations - NOW*) comme un superordinateur distribué à l'échelle d'un bâtiment. Cette vision, introduite par Thomas Anderson [ACP95], a été menée à bien et de nombreux systèmes NOW ont été déployés. NOW offre un système de contrôle de ressources bâti au-dessus d'une collection d'ordinateurs individuels du réseau. L'interconnexion entre les machines est un réseau à grande bande passante, et chacune des machines est une station de travail puissante. Ici on trouve le besoin de ne pas « perdre » de la puissance de calcul sur ces machines coûteuses.

Les points suivants ont été particulièrement développés dans le cadre du projet :

- Communication à faible latence (Active Messages). Les transmissions et réceptions réseau se font sans passer par le noyau système, ce qui permet un grand gain de performances sur les calculs en grain fin.
- Gestion globale des ressources du réseau (GLUnix). C'est une couche logicielle au-dessus de systèmes Unix non modifiés, qui donne l'illusion d'un système d'exploitation global.
- Gestion globale de la ressource disque (xFS). C'est un système de gestion de fichiers parallèle sans serveur, avec traitement des défaillances (RAID5 distribué).

Le projet NOW a été le précurseur des systèmes de gestion de parcs de machines avec un système à image unique. Cependant dans notre cas nous ne trouvons pas face au même problème, qui est d'utiliser la puissance disponible dans les ordinateurs de bureau standard et disponibles *a priori*, que l'on trouve aujourd'hui sur tous les bureaux et dont le coût est déjà amorti par ailleurs pour des besoins bureautiques plutôt que pour des nécessités de calcul.

2.1.2. AppLeS

AppLeS est développé à l'université de Californie à San Diego [BWC+03]. L'objectif du projet réside dans le fait que l'ordonnancement dans un métaordinateur peut être fait efficacement au niveau de l'application, et non pas au niveau des ressources de calcul. En effet, des ressources de calcul très variées peuvent être disponibles, et les politiques de contrôle peuvent être très différentes les unes des autres en fonction des sites de calcul.

L'approche centrée sur l'application offerte par AppLeS amène une méthodologie basée sur des agents. Chaque application dispose de son propre agent qui va :

- collecter des informations sur le système,
- déterminer un ordonnancement propre à l'application,
- implanter cet ordonnancement sur le métaordinateur disponible.

Chaque agent possède un coordinateur qui va se reposer sur les quatre services ci-dessous :

- un *sélecteur de ressources*, qui va émettre des choix en termes de combinaisons de ressources disponibles pour l'application,
- un *planificateur*, qui va créer un ordonnancement conforme à une combinaison donnée de ressources,
- un *évaluateur de performances*, qui utilise une métrique de performance utilisateur pour estimer quel est le meilleur ordonnancement,
- un *actionneur*, qui va placer le meilleur ordonnancement sur les machines appropriées.

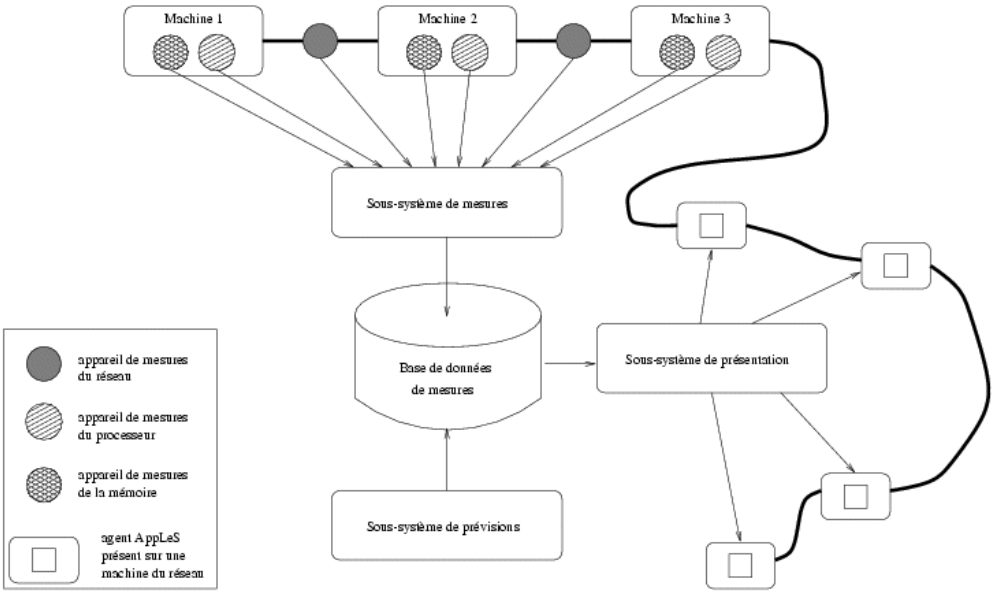

Figure 1 : Structure du système AppLeS

Des capteurs (*Network Weather Service*, [WSH99]) vont permettre de renseigner le système sur l'état du réseau et prédire les pics de charge.

D'un autre côté, le développeur ou l'utilisateur va fournir des informations sur les caractéristiques et les contraintes de l'application.

Un système de capteurs actifs permet de prendre en compte les mesures issues des expériences passées, et donc d'ajuster le modèle d'exécution.

2.1.3. Condor

Condor [LLM88] est un système développé à l'université du Wisconsin à Madison, permettant de faire du calcul très intensif (*High Throughput Computing*). C'est un environnement capable de distribuer des travaux de calculs sur des stations de travail distantes, en faisant une utilisation efficace des ressources réseau disponibles. Condor utilise un système de publication dans lequel chaque machine participante va exposer ses capacités de calcul et ses ressources disponibles à un serveur. Lors du démarrage d'un calcul par un

utilisateur, Condor va trouver la machine disponible la plus appropriée, lui allouer le calcul et le lui déléguer.

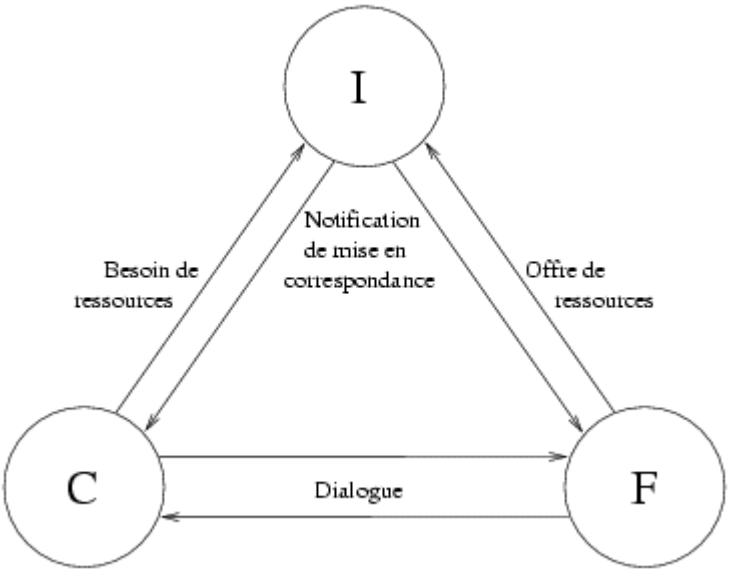

Figure 2 : Modèle structurel de Condor

Dans la figure ci-dessus, nous pouvons voir le modèle structurel de Condor, où les fournisseurs F de ressources (*resource owner agents*) sont mis en contact avec des clients C de puissance de calcul (*customer agents*) par le courtier intermédiaire I (*Matchmaker*).

Condor est capable de faire un point de sauvegarde (*checkpoint*) du travail en cours afin de permettre une reprise à un autre moment, et éventuellement sur une autre machine, de ce travail. Lorsqu'un utilisateur revient à sa machine et la demande implicitement, le travail Condor est stoppé et il est relancé depuis son dernier point de sauvegarde sur une autre machine.

Condor est conçu pour les environnements où les utilisateurs disposent de machines puissantes (stations de travail), et n'alloue des travaux que sur des machines individuelles : Il n'est pas possible pour Condor d'agréger les ressources de plusieurs machines distinctes et de les allouer à un travail parallèle. Des extensions permettant l'exécution de travaux PVM sur Condor ont été expérimentées par Jim Pruyne [PLi94], mais l'allocation ne permet pas de reconnaître les paramètres essentiels à la performance en grain fin tels que l'homogénéité des machines allouées où leur proximité en termes de topologie réseau (donc leur faible latence de communication).

D'autres contraintes inhérentes à l'utilisation de Condor sont que les travaux à exécuter doivent être recompilés d'une part, ce qui interdit l'utilisation de logiciels « sur étagère », et d'autre part que les travaux de calcul sont exécutés dans l'espace utilisateur, sans utiliser de technologie d'isolation de code. Les données et applications de l'utilisateur sont donc exposées en cas de malveillance et d'intrusion par le système Condor.

2.1.4. Piranha

Le système Piranha [GKa92] de l'université de Yale est basé sur le paradigme de programmation *Linda* [CGe89]. Piranha réalise un *parallélisme adaptatif*, construit sur les *Tuples associatifs*. Des calculs parallèles peuvent être effectués par Piranha en injectant des Tuples non évalués dans l'espace des Tuples. L'espace des Tuples peut donc être vu comme un ensemble de Tuples indépendants en attente d'évaluation.

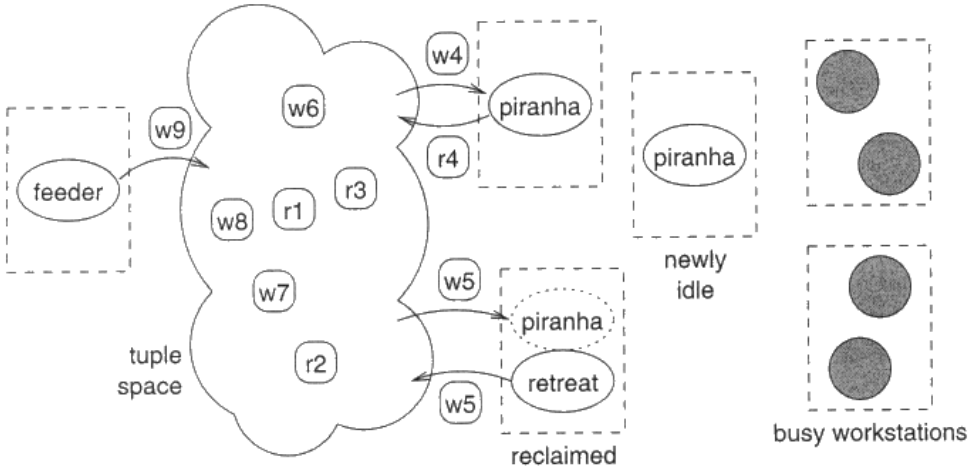

Figure 3 : Génération et exécution de Tuples par Piranha

Les programmes exécutés sous Piranha disposent de trois fonctions :

Feeder, qui est exécutée continuellement sur le premier nœud alloué au programme. Cette fonction va générer les Tuples nécessaires au programme et les injecter dans l'espace des Tuples.

Piranha, qui est appelée automatiquement sur chaque machine qui devient disponible pour l'exécution. Cette fonction va récupérer des Tuples à évaluer dans l'espace des Tuples, exécuter les calculs nécessaires à leur évaluation, créer des Tuples résultat puis les injecter dans l'espace des Tuples, de manière à pouvoir générer un résultat d'exécution du programme.

Retreat, qui sera appelée lorsque la fonction *Piranha* devra être interrompue, typiquement lors du retour de l'utilisateur. Tout programme de calcul est alors immédiatement arrêté et purgé. Il est de la responsabilité du développeur de l'application Linda de gérer ces arrêts et de prendre les actions qui s'imposent, tels que la relance sur un autre site, la sauvegarde de l'état de l'application ou sa migration, après la réception de cette demande de sortie imprévue du programme.

Enfin, Piranha va se baser sur l'inactivité clavier/souris de la machine, ainsi que sur la charge du processeur, afin de pouvoir déterminer si la machine peut être exploitée pour effectuer des calculs collaboratifs.

Piranha a constitué l'un des premiers systèmes de calcul sur machines en jachère, et a eu un certain succès commercial suite à sa distribution par IBM. Toutefois, Piranha reste fortement limité en ce qui concerne le passage à

l'échelle, principalement par les exigences de la mémoire virtuelle distribuée nécessitée par Linda. D'autre part, si Piranha supporte bien la dynamicité « amicale » des machines qui apparaissent et disparaissent du système, il ne peut tolérer la volatilité « violente » d'une machine qui se déconnecte brutalement, car la fonction *Retreat* doit être exécutée pour garder la cohérence de l'espace des Tuples.

2.1.5. Registrar

Andrew Tanenbaum décrit dans [Tan95] le système *registrar*. Ce projet a vu le jour après que des mesures sur les stations de travail de différentes universités montrent qu'à tout instant du jour ou de la nuit, au moins 30% des machines étaient inactives. Et ce chiffre monte à près de 100% la nuit.

L'objectif de registrar est de donner accès aux machines inutilisées par le biais de commandes *rsh*, en proposant un système d'annuaire de machines inexploitées qui permette de sélectionner une machine sans avoir à la connaître explicitement.

Comme le montre la figure ci-dessous, le fonctionnement de registrar est très simple :

Lorsque aucune activité clavier ni souris n'a été enregistrée pendant plusieurs minutes, et qu'aucun processus utilisateur n'est actif, la machine est identifiée comme inactive,

La machine inactive va se déclarer auprès d'un serveur, qui va ajouter le nom de la machine à une liste de machines disponibles,

Lorsqu'un utilisateur a besoin d'une machine distante, il va faire une requête auprès du serveur afin d'obtenir les références d'une machine disponible, qui lui sont retournées par le serveur,

L'utilisateur fait une requête d'accès à la machine inexploitée obtenue auprès du serveur,

L'environnement est copié sur la machine cible, et le processus y est démarré.

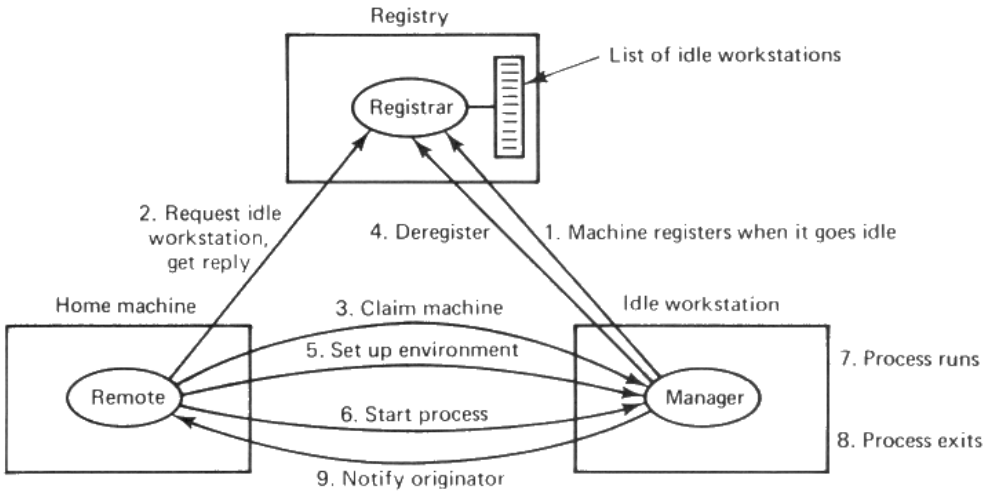

Figure 4 : L'algorithme de registrar pour trouver et exploiter les stations de travail en jachère

Registrar est un mécanisme très simple, qui permet une allocation assez brutale des machines et permet leur exploitation de manière assez efficace. Cependant, il reste à la charge de l'utilisateur de faire en sorte que ses applications fonctionnent bien lorsqu'elles sont exécutées à distance, ce qui inclut une gestion des horloges et des signaux du système indépendante de la machine hôte (une commande *make* par exemple ne pourra pas fonctionner proprement car les horloges des machines hôte et cible ne sont pas synchrones).

Un autre problème de registrar est que rien n'est prévu lors du retour de l'utilisateur de la machine cible. Il devra alors accepter l'exécution sur sa machine d'applications qui ont été démarrées depuis d'autres stations de travail, et ce jusqu'à la terminaison de ces applications.

2.1.6. Pair-à-pair

Les systèmes pair-à-pair tels que Freenet [CSW+01], Chord [SMK+01], CAN [RFH+01] ou Pastry [RDr01] permettent de construire un routage applicatif permettant de récupérer des informations distribuées dans le système. De tels systèmes pourraient être utilisés pour faire circuler des informations sur l'état dynamique des ressources de calcul disponibles. Cependant les coûts d'utilisation réseau sont importants pour faire une recherche. Lors de l'instanciation d'une requête de calcul par un utilisateur, il serait nécessaire de lancer un grand nombre de recherches simultanées afin de trouver les ressources disponibles sur le réseau, engendrant un probable écroulement des performances du système. A noter de plus que Freenet, CAN et Pastry ne proposent pas de mutabilité des données (les données ne sont pas modifiables une fois placées dans le système), et il n'est donc pas possible de rafraîchir directement les valeurs d'une donnée stockée par le système.

D'autres systèmes pair-à-pair offrent un multicast applicatif basé sur des réseaux sémaphores (*overlay networks*). Par exemple, le système HyperCast [LBe99] est un système de distribution d'informations qui construit un ensemble d'arbres de recouvrement permettant de garder une bonne efficacité dans le cas ou le nombre de nœuds est très grand. Ce système calcule des triangulations de Delaunay entre les différents nœuds en partant d'un graphe de connexion complet et va rapidement converger vers un réseau sémaphore optimal. Ce système résiste très bien aux partitionnements réseau, aux délocalisations de nœuds, et permet une diffusion économique des informations en termes de consommation réseau. Cependant, encore une fois la connaissance exhaustive de l'ensemble de ses pairs par chacun des nœuds limite le passage à l'échelle. D'autre part, la triangulation de Delaunay nécessite de disposer d'une métrique de localisation géographico-topologique des nœuds sur le réseau, ce qui n'est pas facilement disponible.

Un système tel que Gnutella [Gnu01] aurait pu nous permettre la publication des informations de ressources de chacune des machines du nuage, et une requête d'interrogation vers les pairs voisins aurait pu alors être utilisée pour trouver un jeu de machines approprié à un travail parallèle donné. Toutefois le mode réactif utilisé par Gnutella a été mis en cause à de nombreuses reprises [Rit01] au niveau de la capacité à gérer un passage à l'échelle du système sans provoquer une explosion du nombre de messages

nécessaires au fonctionnement. Comme nous allons le voir, le mécanisme pair-à-pair utilisé dans I-Cluster fonctionne de manière proactive et permet de garantir que le nombre de messages nécessaires au fonctionnement du système reste limité même lors d'un passage à l'échelle massif de la communauté de calcul.

2.1.7. Métacalcul

Le terme Métacalcul (*Metacomputing*) a été inventé par Larry Smarr [SCa92] à la suite du projet CASA de la NCSA, qu'il dirigeait. Il peut parfois être dénommé calcul global (*Global Computing* ou encore *Desktop Computing*). Le but est ici le passage à l'échelle massif de l'agrégation des ressources d'un grand nombre de machines individuellement connectées à Internet. Les applications les plus courantes du métacalcul sont des programmes propriétaires contrôlés par un serveur central en suivant un paradigme maître-esclave. Ces programmes sont généralement des systèmes multiparamétriques, qui exécutent un grand nombre de fois la même application avec des jeux de paramètres différents pour chaque exécution. La somme des besoins de calcul est très importante, de l'ordre de plusieurs années de calcul séquentiel.

Un des premiers systèmes notoires de calcul global vit le jour en 1994 lorsque Atkins *et al* [AGL+94] utilisèrent 1600 machines appartenant à 600 participants différents pour « casser » le *challenge RSA* entre août 1993 et le 2 avril 1994. Un serveur central envoyait par e-mail à chaque participant les jeux d'exécution qu'il devait exécuter, et le résultat était renvoyé automatiquement par e-mail au serveur. Une fois le challenge RSA réussi, il a été imaginé de nombreuses manières d'exploiter la puissance dormante disponible dans les ordinateurs connectés à Internet.

Le calcul en grille (*Grid Computing*) est un autre concept qui a été exploré à partir de 1995 lors de l'expérience I-WAY [Fos99] par Ian Foster et son équipe de recherche. Cette expérience a mis en œuvre 17 centres de calcul répartis en Amérique du nord, interconnectés par des réseaux à haute vitesse. Le but étant de pouvoir exploiter un site de calcul distant de la même manière qu'en local, en disposant d'un frontal d'accès transparent.

De cette activité sont nés de nombreux projets de recherche, aidés par un fort support financier de la part de l'agence scientifique du gouvernement américain (*National Science Foundation*), dans le but de réduire les coûts d'achat de gros calculateurs en permettant une meilleure répartition des ressources qu'ils offrent. Les efforts sont maintenant concentrés autour du projet Globus [FKe96], qui permet aux utilisateurs d'accéder à des ressources de calcul et de stockage réparties mondialement sans avoir à se soucier de leur localisation.

Une description des systèmes de métacalcul plus complète peut être trouvée par ailleurs [BFo98], et nous nous limiterons ici à la revue des systèmes les plus intéressants relativement à notre étude.

2.1.8. FAFNER

FAFNER est un projet démarré en 1995 visant à factoriser des clefs cryptographiques, réalisé par un consortium comprenant *Bellcore Labs*, l'université de Syracuse et *Co-Operating Systems*.

Les systèmes de cryptographie à clef publique reposent sur l'utilisation de deux clefs : L'une est publique et peut être communiquée, tandis que la clef privée y est mathématiquement liée de telle manière que seul un message crypté avec la clef privée puisse être décrypté avec la clef publique. La cryptographie à clef publique se repose sur le fait que le décryptage d'un message se fait en temps polynomial (d'ordre assez bas) lorsque la clef privée est connue, alors que ce décryptage ne peut se faire qu'en temps super-polynomial lorsque seule la clef publique est connue, rendant ce décryptage impossible dans la pratique, aucune entité n'ayant la puissance de résoudre un problème super-polynomial utilisant des tailles de clefs suffisantes. L'algorithme RSA [RSA78] est un exemple d'algorithme à clef publique.

Les clefs RSA sont générées par combinaison de grands nombres premiers. La sécurité de RSA repose en partie sur le fait qu'il est très difficile de factoriser les très grands nombres (les clefs RSA comportent 154 ou 512 chiffres).

RSA Data Security Inc. a lancé un challenge de factorisation en mars 1991. Ce challenge demandait une grande puissance de calcul. Pour cette raison, des algorithmes de factorisation parallèles ont été développés de manière à ne plus nécessiter de communications après le démarrage du calcul, qui devient donc indépendant.

Le projet FAFNER a été mis en place pour factoriser RSA130 en utilisant une nouvelle technique nommée le *Number Field Sieve* qui utilise des serveurs de calcul. Une interface Web d'accès aux serveurs a été développée, permettant à des volontaires d'accéder à un logiciel de factorisation local, à la dissémination des tâches de criblage, et à la collecte des résultats. Chaque contributeur devait télécharger un programme, le compiler et le lancer en tâche de fond sur la machine participante, qui prenait alors part au calcul global.

Le projet FAFNER a été réussi, grâce à trois facteurs clefs :

- l'algorithme de *Number Field Sieve* qui avait été développé ne demandait que peu de ressources locales, ce qui permettait son utilisation sur la plupart des machines disponibles,
- le projet ne demandait qu'un enregistrement anonyme, sans devoir révéler leur identité pour offrir leurs ressources de calcul au projet (ceci n'était pas le cas dans les projets de factorisation précédents, qui se basaient sur des communications par e-mail),
- les sites serveurs formaient une hiérarchie qui réduisait les goulots d'étranglement à ce niveau, en ne nécessitant qu'une administration réduite ne nécessitant pas l'intervention d'administrateurs.

FAFNER a été le précurseur de nombreux autres projets de calcul global, et a permis de prouver le concept, ainsi que de valider les trois clefs du succès décrites ci-dessus.

2.1.9. SETI@home

SETI@home [And+99][And+02] est un projet scientifique faisant partie du programme SETI de recherche d'intelligence extraterrestre. SETI@home est basé sur l'agrégation de la puissance de calcul disponible sur des millions de machines connectées à Internet, atteignant la puissance colossale de 20 Tflop/s.

Le projet consiste en une seule et unique application de traitement de données émises par le radio télescope du Mont Arecibo : chaque parcelle de l'espace est scrutée dans la bande des 2 MHz pendant 50 secondes, ce qui fournit un échantillon de mesure de 250 Ko. Ces échantillons sont transmis à des machines de l'Internet participant volontairement au système. Chacune de ces machines est équipée de l'économiseur d'écran spécifique (le logiciel est cependant disponible de manière autonome) [SWB+97], qui va, en plus de son rôle premier, effectuer la récupération d'échantillons auprès du serveur, les traiter localement puis retourner un rapport d'analyse au serveur.

Chaque traitement d'un échantillon dure quelques heures (en fonction de la puissance de la machine cliente), et l'économiseur d'écran répète le processus tant que la machine reste en mode de sauvegarde d'écran.

Afin d'éviter que le calcul en cours ne soit perdu si l'utilisateur provoque la sortie du mode de sauvegarde d'écran de la machine, l'application SETI@home dispose d'un mécanisme de gel/reprise (*checkpointing*), qui va sauvegarder un point de reprise de l'application toutes les 10 minutes. De cette manière, chaque tranche de 10 minutes passée à calculer va produire un résultat intermédiaire qui pourra être utilisé pour continuer par la suite et permettre d'arriver au bout du traitement de l'échantillon en cours.

SETI@home constitue un succès énorme et la première réussite de déploiement d'un système de calcul global agrégeant les ressources des machines de millions d'utilisateurs non spécialistes. L'approche de cet ordinateur virtuel mondialement distribué est très encourageante, bien que la limitation mono-application et l'impossibilité de communication inter-processeurs ne permettent pas d'étendre facilement son champ d'application.

2.1.10. XtremWeb

XtremWeb [GNF+00][FGN+01] est un système développé au Laboratoire de Recherche en Informatique (LRI) de Paris XI, permettant la mise en place de services de calcul global en mode pair-à-pair (*Systèmes Distribués à Large Echelle - SDLE*). XtremWeb fournit un environnement simplifiant le développement d'applications de calcul haut débit. Il est conçu pour être utilisé à l'échelle mondiale, sur des machines inexploitées de l'Internet, appartenant à des utilisateurs volontaires pour offrir leur puissance de calcul au système.

Les communications entre les nœuds n'étant pas prises en charge[2], le système reste limité aux calculs décomposables en tâches indépendantes et de petite taille.

XtremWeb se repose sur les composants distribués suivants :

- un *serveur*, qui gère un sous-ensemble des machines clientes du système, et qui ordonnance leurs travaux,

[2] *L'équipe XtremWeb travaille aussi sur MPICH-V [BBC+02], qui offre un mécanisme permettant de conserver le paradigme de communication par passages de messages MPI tout en permettant le gel/reprise d'applications interdépendantes. A noter que d'autres travaux tels que MPICH-CM [SGB+02] abordent ce problème d'une manière similaire.*

- un *méta serveur*, sur lequel tourne le *répartiteur*. Ce répartiteur reçoit les requêtes de calcul et les répartit sur les différents serveurs disponibles,
- les machines des volontaires, qui peuvent fonctionner en mode *utilisateur*, où l'on peut soumettre des applications au système ; ou bien en mode *collaborateur*, où elle est alors utilisée pendant ses périodes d'inactivité pour exécuter les applications soumises.

XtremWeb offre donc une réponse aux deux problèmes suivants :

- la *versatilité* : Contrairement à un système tel que SETI@home, XtremWeb n'est pas lié à une application particulière mais permet au contraire d'exécuter toutes sortes d'applications, et offre le support nécessaire (sous forme de bibliothèques) pour en développer de nouvelles ;
- la *répartition de la charge du serveur central* : Contrairement à un système tel que Condor qui n'a qu'un serveur central et peut donc difficilement passer à l'échelle, XtremWeb propose une structure de serveurs hiérarchique à deux niveaux (répartiteur et ordonnanceur), ce qui permet de rajouter des serveurs lorsque le nombre de machines clientes croît et ainsi passer à l'échelle de manière contrôlée.

XtremWeb est un système de calcul global flexible et reconfigurable. Sa mise en place a toutefois montré que la partie cliente du système nécessite une excellente intégration au système d'exploitation, ainsi qu'une stabilité à toute épreuve, sous peine de rejet psychologique par les volontaires. Ceci met l'accent sur la difficulté de mise en œuvre de systèmes qui par ailleurs ne présentent pas forcément d'avancées scientifiques importantes.

2.1.11. Bac à sable et confinement d'exécution

Les machines de bureau posent des contraintes implicites au niveau de leur environnement de travail. Par exemple, le système, le système d'exploitation est souvent imposé. La proportion de systèmes clients basés sur Microsoft Windows™ est estimée par Gillen et al. [GKS+01] entre 91 et 96% des machines déployées.

Par ailleurs, pour des raisons historiques, les environnements de calcul sont majoritairement basés sur Linux ou Unix.

Les systèmes actuels posent des contraintes implicites sur l'environnement d'exécution. Par exemple, le système d'exploitation de l'utilisateur est bien souvent imposé par le système de calcul distribué. Pour des raisons de facilité de développement, ou pour des motivations plus académiques que réelles, ces systèmes imposent souvent l'utilisation d'Unix comme système d'exploitation. Or nous avons vu que la proportion de systèmes clients basés sur un système d'exploitation Microsoft Windows est énorme, en particulier sur les réseaux d'entreprise. Il est donc important de pouvoir supporter Windows, et ne pas limiter notre environnement de calcul à Linux.

Dans le cadre de I-Cluster, notre approche est de s'adapter à l'environnement et à l'infrastructure, plutôt que d'imposer notre propre mécanique, de manière à pouvoir bénéficier d'un important taux d'acceptation par les utilisateurs. Entre autres, nous cherchons à protéger l'environnement d'exploitation de l'utilisateur lors de son exécution ainsi que dans son mode dormant (en l'absence de l'utilisateur ou lors des déconnections de la machine). Nous n'avons pas de processus en tâche de fond lors des modes utilisateurs, excepté un module très léger de surveillance de la charge. Et nous avons un cloisonnement des données utilisateur par rapport aux données d'exécution de tâches I-Cluster, ce qui permet de limiter les risques liés à la sécurité des données.

2.1.12. Modèles d'isolation de code

Les problèmes de sécurité liés à l'introduction et l'exécution de code externe sur une machine de l'intranet nécessitent un mécanisme de protection. En particulier, il nous faut isoler les données, processus et sessions réseau de l'utilisateur de ceux nécessaires à l'exécution des tâches de calcul distribué.

Nous considérons le problème de la sécurité seulement selon l'aspect de la prévention des accès non autorisés. Nous avons retenu le choix d'architecturer notre système de manière à définir deux modes d'exécution (mode utilisateur et mode cluster) pour chaque machine, bien distincts et exclusifs de manière temporelle. Nous reviendrons sur le détail de ces deux modes, qui permettent une isolation solide des objets d'exécution des tâches utilisateur ou des tâches de calcul.

Hawblitzel et al [Haw98] ont suggéré une classification des mécanismes de protection de code en trois classes définies comme suit :

2.1.12.1. Mémoire physique virtuelle

La première classe concerne les mécanismes basés sur de la mémoire physique virtuelle, qui rend possible une isolation des codes à l'aide de ressources matérielles. Notre but étant d'offrir une architecture basée sur les PC standard disponibles sur un intranet, il ne nous était pas possible de modifier leur architecture sans avoir à justifier des problèmes et des coûts de déploiement importants.

2.1.12.2. Les systèmes à capacité

La seconde classe de solutions, les systèmes à capacité, a été utilisée dans les systèmes d'exploitation à micronoyau tels que Amoeba [TKV+91] ou Chorus [ARG89]. Ces systèmes, basés sur l'examen des types de données lors de l'exécution, nécessitent l'instrumentation des codes source, et laissent ainsi les traces nécessaires au mécanisme contrôlant les listes de capacités. Le Java Runtime Environment décrit par Arnold et al. [AGH94] dispose d'un bac à sable qui est automatiquement activé lorsque les applets appartenant a cette classe sont exécutées. *Internet C++* [FDF03] en est encore un autre exemple, permettant l'exécution de code POSIX associé à une machine virtuelle donnant accès à une interface réseau (modèle basé sur les sockets Berkeley) ainsi qu'à une interface graphique (OpenGL). Un bac à sable de la classe des systèmes à

capacité nous aurait probablement permis l'exécution native d'applications Windows, et donc de partager dynamiquement les ressources de chaque machine en lançant ces applications en tâche de fond tout en disposant de sécurité logicielle contre les fautes éventuelles de ces applications. Toutefois, cette classe de mécanismes ne permet pas d'utiliser les applications de calcul intensif qui sont principalement développées autour du système Unix ou Linux. Il n'était pas souhaitable pour notre système d'avoir à imposer un type spécifique d'applications, aussi nous n'avons pas retenu de solution basée sur les systèmes à capacité.

2.1.12.3. L'isolation de fautes par logiciel

L'isolation de fautes par logiciel (*Software-based Fault Isolation* ou encore SFI) constitue la dernière classe de mécanismes de protection. Ce type de mécanismes ne nécessite pas de points d'instrumentation dans le code mais se base sur une surveillance qui se fait en parallèle lors de l'exécution. VMWare [Vmw00] offre une machine virtuelle permettant l'exécution d'applications Linux au-dessus d'une machine équipée du système d'exploitation Windows (l'inverse est aussi possible). Nous avons de la même manière choisi d'opter pour un système d'isolation de fautes par logiciel. Dans notre cas, les ressources de chaque machine vont être gérées par notre système logiciel qui va les allouer pleinement soit à l'utilisateur, soit à des tâches de calcul. Nous aurions pu nous limiter à un partage moins strict des ressources et développer une couche d'émulation du système Linux au-dessus de Windows, telle que celle développée par *User-Mode Linux* (UML-Win32) [Dik00]. De cette manière, les tâches de calcul auraient pu être lancées en arrière-plan de la même manière que dans le système NOW [ACP95]. Cependant il aurait alors été beaucoup plus difficile de garantir l'intégrité de la machine et d'atteindre un bon niveau de sécurité. D'autre part, nous avons vu lors de nos expérimentations sur Linpack qu'un facteur prépondérant pour l'efficacité d'un calcul parallèle en grain fin était de pouvoir garder une homogénéité de charge entre les machines. Dans le cas d'un système s'exécutant en arrière-plan il est difficile de pouvoir garantir que l'utilisateur ne va pas lancer des travaux en avant-plan sur sa machine, qui auront pour effet de charger celle-ci et donc de dégrader fortement l'efficacité du calcul en cours.

Bosilca et al. [BFH+02] ont fait une évaluation de performance des techniques de confinement d'exécution. Ils partagent les techniques de confinement d'exécution en deux classes :

- l'exécution de code intermédiaire par une machine virtuelle, ou
- l'exécution de code natif dans un environnement protégé.

L'étude porte sur l'exécution de code dans des machines virtuelles Java d'une part [AGH94] et sur d'autres systèmes d'interception des appels système (Janus [GWT+96], Consh [AKS98] et MAPbox [ARa00]) tous basés sur le mécanisme *ptrace* du noyau Linux. Ces systèmes permettent une isolation d'exécutables Linux par-dessus Linux lui-même.

Dans [SSC+93], Swanson et al. décrivent un système permettant l'exploitation des instants d'inactivité de machines, à l'aide d'un système

« schizophrène » permettant l'encapsulation de code dans une machine virtuelle, basée sur des extensions du noyau Mach.

Dans notre cas, nous avons défini une architecture d'isolation de code de niveau beaucoup plus bas, car l'ordinateur va changer de mode d'exécution et prendre soit une personnalité utilisateur, soit une personnalité calcul. Concernant l'isolation du code, Tanenbaum [Tan95] met l'accent sur 3 axes principaux de recherche dans le cadre de l'utilisation des machines en jachère :

1) Comment trouve-t-on une machine inactive ?
2) Comment peut-on exécuter un processus à distance de manière transparente ?
3) Que se passe-t-il si l'utilisateur de la machine revient ?

Dans I-Cluster, nous nous basons sur les outils existants de gestion de grappes pour répondre à la question 2. Pour les questions 1 et 3, nous adoptons une approche originale : plutôt que de réagir aux états des machines passant en mode inactif ou retournant dans un mode d'utilisation standard, nous analysons le profil d'utilisation normale de la machine par son [ses] utilisateur[s], et nous allons réaliser une **prédiction des fenêtres de tir** possibles de la machine.

Cette approche devient possible aujourd'hui seulement. Par le passé, les stations de travail exploitées par les systèmes de calcul sur grappes virtuelles se basaient sur de petits nombres de machines disponibles par exemple dans un laboratoire. Ces machines étaient puissantes et coûteuses, il s'agissait donc d'en tirer le meilleur parti possible. De nos jours, la plupart des employés de bureau ont leur ordinateur dédié, qui offre une puissance de calcul et une communication tout à fait respectable, même dans le cas de l'utilisation en mode grappe de calcul. Nous avons donc tiré parti du fait de la présence d'un grand nombre de machines sur un intranet, et nous avons donc privilégié la transparence utilisateur de notre système. En conséquence, nous n'utilisons pas pleinement la puissance inexploitée disponible depuis le parc de machines, mais nous répondons facilement et efficacement aux questions 1 et 3 ci-dessus de Tanenbaum.

2.2. Exploitation de ressources dédiées

Un gestionnaire de grappe ou de grille de calcul est un intermédiaire entre une entité (processus, application, utilisateur) soumettant des tâches à exécuter et un ensemble de ressources qu'il a la responsabilité de gérer [Buy99].

Des systèmes tels que AMOEBA [TKV+91] ou Glunix [ADV+94] offrent une vision unique d'un ordinateur constitué de multiples entités réparties.

Différentes politiques d'utilisation des ressources ont été définies : politiques basées sur le partage, politiques fonctionnelles (lorsque les utilisateurs ont des niveaux d'importance différents), par calendrier (lorsque les travaux doivent être exécutés en un temps donné), ou encore manuelles.

Dans certains cas tels que le notre, la politique sera du type premier arrivé, premier servi (FIFO) : Une requête de demande d'allocation de machines pour un travail donné peut être autorisable ou non, selon que le demandeur a le droit

d'accès ou non aux machines et que les machines sont déjà attribuées ou non, et qu'elles ont disponibles pour la durée de la fenêtre de tir demandée.

2.2.1. NetSolve

Le projet NetSolve [CDo95] a pour objectif de prendre en charge des ressources de calcul disparates disponibles sur le réseau et de les agréger en une machine de calcul scientifique. Un serveur NetSolve est responsable du dialogue avec les agents NetSolve installés sur chacune des machines de calcul participant à la communauté. Les utilisateurs peuvent soumettre des travaux au serveur, qui seront distribués sur les machines disponibles en fonction de leurs capacités, de leur charge et de la nature du travail demandé.

Le système est capable de s'adapter aux nécessités des travaux, cependant il n'est pas possible pour NetSolve d'agréger plusieurs machines pour résoudre un unique travail de manière parallèle. En effet, NetSolve ne permet que l'allocation d'une seule machine à un travail donné, et ne permet pas d'allouer un lot de machines destinées à effectuer ce travail en parallèle.

2.2.2. DIET

La plate-forme DIET (*Distributed Interactive Engineering Toolbox*) [Cde03][Lom02], est un travail conjoint entre plusieurs équipes INRIA démarré en 2000. DIET est une plate-forme de métacalcul multi-agents. Les agents interconnectés entre eux ont chacun la responsabilité de la gestion d'un sous-ensemble d'un parc de serveurs de calcul.

Par rapport à d'autres systèmes de métacalcul tels que NetSolve [AAB+01] ou Ninf [NSS99], DIET ne se repose pas sur un serveur central pour ordonnancer les travaux, mais se base sur une hiérarchie d'agents. Ceci évite le goulot d'étranglement dû à la présence d'un unique serveur central et répartit la charge sur les agents. DIET repose en partie sur le composant FAST (*Fast Agent's System Timer*) afin de détecter et d'instrumenter les ressources disponibles sur chaque machine de calcul, de manière dynamique. FAST permet aussi de construire une métrique de capacité de chaque machine basée sur un jeu de tests (*benchmark*) typique, qui permet par la suite d'évaluer le temps nécessaire à l'exécution d'une application donnée sur une machine, et donc de prévoir l'utilisation future de la machine.

DIET repose sur le paradigme de communication CORBA, qui permet une excellente portabilité et une bonne performance. Le courtier (*Trader*) va jouer un rôle d'allocateur de ressources et va offrir l'infrastructure nécessaire au développement d'applications de calcul numérique sous forme de bibliothèques qui encapsulent les détails de l'intercommunication de données entre les sites d'exécution.

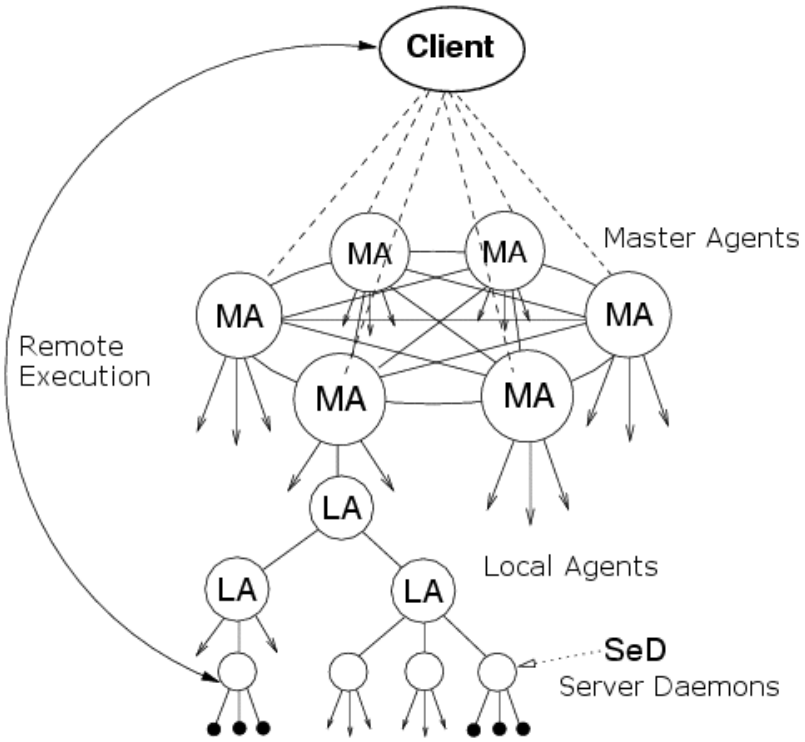

Figure 5 : Hiérarchie d'agents DIET

De nombreux outils de résolution de problèmes numériques en grille (*Problem Solving Environments*) ont été développés, tels NetSolve [AAB+01], Ninf [NSS99], NEOS [FMM00] et RCS [AGO97]. Ils sont généralement référencés comme "*Network Enabled Server (NES) environments*" [MNS+00]. Ces environnements sont généralement composés de cinq composants différents : les *clients* soumettent les problèmes à résoudre aux *serveurs* ; une *base de données* contient les informations à propos des ressources logicielles et matérielles disponibles ; un *ordonnanceur* sélectionne un site d'exécution approprié en fonction du problème soumis et des informations de la base de données ; et enfin un *moniteur* acquiert de l'information relative au statut d'exécution des ressources de calcul.

Un serveur DIET est construit autour de démons gérant les ressources de calcul et d'un démon serveur. La hiérarchie d'agents utilise un redirecteur qui permet de sélectionner un agent maître le plus proche du client possible. Une telle hiérarchie permet d'absorber la charge et de faciliter le passage à l'échelle.

SLiM (*Scientific Libraries Metaserver*) est un composant permettant d'ajuster les problèmes soumis par les clients aux implémentations disponibles sur les serveurs. Les informations sont stockées dans un annuaire LDAP hiérarchique.

FAST (*Fast Agent System Timers*) est un composant permettant la prédiction dynamique de performance en environnement grille. FAST se repose sur des capteurs logiciels permettant d'instrumenter des paramètres de ressources tels que la charge CPU, ainsi que la charge réseau à l'aide de NWS (*Network Weather Service*) [WSH99][WSP97]. Basé sur un modèle de consommation associé à chaque triplet {problème, machine, paramètres}, un jeu de valeurs calculées à l'aide d'exécutions de jeux de tests lors de l'initialisation

du système, associé à une extrapolation polynomiale, permet de prédire les besoins d'une exécution donnée de manière semi statique.

L'architecture offerte par DIET permet donc d'obtenir des propriétés proches de celles que nous désirons au niveau du passage à l'échelle par exemple, de l'adaptation dynamique à la charge des processeurs et du réseau.

2.2.3. Beowulf

Le calcul en grappe est un type d'infrastructure qui a été popularisé par le projet *Beowulf* de la NASA [BSS+95.]. Le but principal de ce projet fondateur était de construire une machine de calcul scientifique sur une base de matériel de consommation courante i.e., basé sur des PC standard interconnectés par Ethernet.

Pour la petite histoire, Beowulf, héros mythique des sagas Scandinaves, était le neveu de Hygelac, Roi de la tribu des Geats. Il a défait le monstre cannibale Grendel, qui menaçait d'anéantir cette tribu. Les grappes de type Beowulf pourront peut-être aussi nous défaire des superordinateurs propriétaires qui cannibalisent les ressources des services de recherche.

Le projet Beowulf repose sur des concepts simples :

- ordinateurs à grande diffusion (*Components Off The Shelf – COTS*),
- composants réseau à faible coût,
- système d'exploitation libre (*open source*),
- matériel non propriétaire,
- logiciel libre (*open source*).

La « preuve du concept » de Beowulf a été réalisée en 1994 par Thomas Sterling et Donald Becker. Leur grappe, baptisée Wiglaf, avait 16 nœuds basés sur des processeurs 80486 à 100MHz, de 16 Mo de RAM et d'un disque dur de 540 Mo à 1 Go chacun. Chaque noeud était un PC de bureau standard, et l'interconnexion se faisait à l'aide de cartes Ethernet 10 Mbps, et leur système d'exploitation était Linux (RedHat 5.0).

Le point d'engorgement du système était constitué par le réseau. Des cartes Ethernet ont donc été rajoutées à chaque machine pour faire de l'agrégation de bande passante (*Channel Bonding*) afin de contourner cette limitation et ainsi augmenter les performances de la grappe.

De nouvelles techniques sont mises au point par des équipes de recherche pour construire des algorithmes efficaces de parallélisation sur des grappes de type Beowulf. Ce domaine de recherche en pleine évolution reste ouvert à des problèmes et à des disciplines diverses.

Ce projet a eu un succès immédiat et a ouvert la voie à de nombreux autres projets académiques ou commerciaux. Toutefois, la structure d'un système Beowulf reste fortement dépendante d'une architecture centralisée et administrée à la main. Dans notre cas, où le nombre de machines à gérer peut être très grand, il ne devient plus possible de se reposer sur de tels mécanismes.

2.2.4. MOSIX

MOSIX est un *Multicomputer Operating System for unIX* [BWh89]. C'est un projet qui vit depuis plusieurs décennies, à l'Université hébraïque de Jérusalem. L'objectif de MOSIX est de fournir un ensemble de mécanismes adaptatifs de partage de ressources permettant de manipuler une « ferme de serveurs » (ensemble de machines de calcul) comme un système unique, en maximisant la performance par l'utilisation efficace des ressources réseau. Ces fermes de serveurs se composent de machines standard, pouvant être hétérogènes, reliées par un réseau standard.

MOSIX est basé sur un noyau Unix modifié de manière à permettre la migration des processus entre les machines appartenant à la ferme de calcul. MOSIX dispose aussi d'algorithmes de distribution d'informations de charge des machines, permettant l'instrumentation de la charge des différentes machines d'une ferme de serveurs, de détecter les différences de charge entre machines et de provoquer les migrations de tâches depuis les machines les plus chargées vers les machines faiblement chargées (*PPM-Preventive Process Migration*). Le système est symétrique et distribué, sans contrôle centralisé.

Chaque processus possède un nœud initial (*UHN-Unique Home Node*) qui est généralement celui sur lequel l'utilisateur est connecté. Le corps (*body*) du processus contient son contexte au niveau utilisateur ainsi que certaines informations, indépendantes du site d'exécution, du niveau noyau. Le représentant (*deputy*) contient les informations dépendantes du site d'exécution, et réside sur le nœud initial. Un lien de communication est établi entre le représentant et le corps de manière à ce que le processus puisse accéder à son environnement local par le représentant, et que les autres processus puissent accéder à lui. Les processus migrés sur une machine distante utilisent autant que possible les ressources locales à cette machine, mais les ressources système telles que connexions réseau ou fichiers ouverts vont rester sur le nœud initial et vont être gérés par le représentant. La figure ci-dessous illustre ce mécanisme.

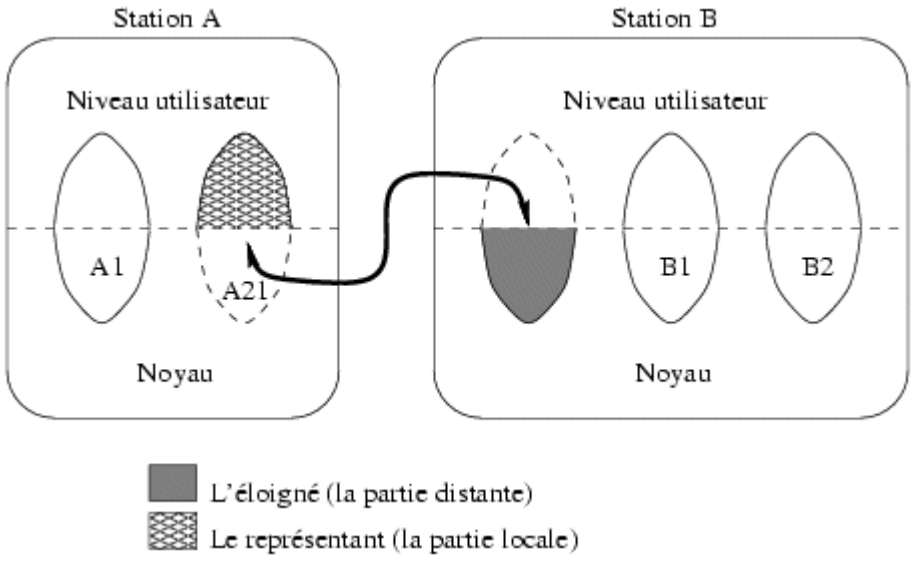

Figure 6 : La migration de processus avec MOSIX

Les informations de charge sont distribuées à l'aide d'un algorithme de bavardage aléatoire [BGW93]. Chaque station de travail maintient un vecteur d'information à propos de la charge de chacune des autres machines de la ferme de calcul. A intervalles de temps réguliers, elle va envoyer ce vecteur d'informations à une autre station de la ferme, choisie aléatoirement. Lors de la réception d'un tel vecteur d'informations, le contenu du vecteur est combiné avec le contenu du vecteur local de manière à ne garder que les informations les plus récentes. Si la charge entre deux machines est très différente, une opération de migration est lancée. Il est à noter que la définition pour MOSIX de la charge de chaque machine est une fonction directe de la quantité de mémoire disponible, et non pas de la charge processeur, qui varie beaucoup et de manière difficile à caractériser. Ceci permet d'éviter de gérer des variations trop rapides de la charge des serveurs, ce qui provoquerait une charge importante de l'algorithme de bavardage.

Il n'y a pas de contrôle central ni de relation maître-esclave entre les nœuds d'une ferme MOSIX : Chaque nœud opère de manière autonome, et il gère ses décisions de contrôle de manière indépendante. Cette architecture permet une configuration dynamique, et une excellente adaptation aux conditions changeantes telles que des nœuds tombant en panne ou rejoignant la ferme.

MOSIX ne gère pas le contrôle d'accès, et ne gère pas non plus l'activité des utilisateurs des serveurs (ou stations de travail) constituant la ferme de calcul. De fait, la notion d'utilisateur ou encore de possesseur de la machine est assez floue, car les travaux de l'utilisateur peuvent aussi migrer vers d'autres machines moins chargées.

Les systèmes d'exploitation des machines de la ferme doivent être homogènes.

L'architecture de migration choisie impose une surcharge due à l'utilisation du représentant : Chaque appel système devant être transféré sur le UHN. Un système de sockets de communication migratrices a été développé afin de migrer les structures de communication en même temps que leur processus.

2.2.5. DPM

DPM (*Distributed Process Manager*) [Dro98] est un système permettant de gérer un ensemble de machines fédérées telles qu'une grappe d'ordinateurs. Développé par César De Rose lors de sa thèse à l'université de Karlsruhe, DPM construit un graphe sémaphore (*overlay graph*) constituant une hiérarchie virtuelle des machines, qui permet d'allouer des partitions ou sous-partitions de la grappe à un travail donné. DPM a des caractéristiques de gestion de la dynamicité qui lui permettent de gérer la volatilité des machines ou les partitionnements du réseau.

Bien qu'il ne soit pas auto-organisant ni particulièrement efficace pour gérer de grands nombres de machines, nous avons travaillé sur des extensions de DPM liées au présent travail de thèse [DBM+02]. Ce travail, dirigé par César De Rose du centre de recherches en calcul haute performance (CPAD) de l'université catholique de Rio Grande Sud (PUCRS), a été nommé *vCluster*. Le but de vCluster est d'exploiter les ressources de calcul disponibles sur un grand nombre de machines d'un réseau, en utilisant les mécanismes de gestion distribuée des ressources de I-Cluster (le nuage, décrit ci-après au chapitre 3.3) à

un haut niveau, de manière à bénéficier des possibilités en termes de passage à l'échelle, tout en utilisant les mécanismes de DPM au niveau des ressources locales (par exemple les machines disponibles dans une salle de classe). Ce travail permet d'exploiter les bénéfices de nos deux efforts de recherche. Cependant, la complexité accrue de mise en œuvre de vCluster réserve son utilisation à des besoins de démonstration ou d'expérimentation.

Le bac à sable I-Cluster (décrit au chapitre 3.2 du présent document) est utilisé au laboratoire CPAD, où une cinquantaine de machines sont équipées des composants logiciels qui leur permettent d'exploiter les périodes d'inactivité des ordinateurs d'une salle de classe et de pouvoir en agréger la performance à l'aide de DPM.

2.2.6. Globus

Globus [FKe96] est un projet initialement développé à l'université de Chicago et depuis repris par *Argonne National Laboratory*. Ce projet consiste en un ensemble de services et d'outils pour la construction de grilles de calcul. Globus est un ensemble d'outils autonomes les uns par rapport aux autres. Son usage principal est la construction et l'exploitation de grilles parallèles virtuelles, mais son champ d'application effectif est assez large, car de nombreux composants de Globus sont utilisés par diverses équipes. Globus fournit des outils d'accès transparent aux ressources distantes, la connexion d'outils de mesure aux applications ou encore des services de sécurité, d'authentification et de connectivité utilisateur. Le *Global Grid Forum* est une communauté internationale qui a pour objectif de standardiser les technologies de calcul en grille et leurs interfaces et qui fournit la spécification technique *Open Grid Services Infrastructure (OGSI)* ; le *Globus Toolkit* est une implémentation en logiciel libre (actuellement à sa version 3) de la plupart de ces protocoles, API et services, disponibles en paquets logiciels dénommés *modules*.

Les modules disponibles dans le *Globus Toolkit* offrent les possibilités suivantes :

- l'infrastructure de sécurité (*Grid Security Infrastructure - GSI*), qui permet d'étendre les solutions de sécurité locales telles que PKI ou Kerberos pour fournir un accès uniforme (*Single sign-on*) à une grille, ainsi que des mécanismes de délégation et de politiques de sécurité,
- la gestion de données, qui permet la gestion de jeux de données importants. GridFTP est une extension de FTP qui fournit un service de transfert de fichier à hautes performances, incluant un support pour le parallélisme, la tolérance aux pannes, le transfert partiel de fichiers et le contrôle de tampons TCP,
- l'infrastructure d'information permet, à l'aide de *Monitoring and Discovery Service (MDS)*, de fournir un accès uniforme à l'information entre les composants tels que les ordinateurs, systèmes de stockage et réseaux.

- la gestion de ressources permet l'accès uniforme aux ressources des systèmes hétérogènes à l'aide de *Resource Specification Laguage (RSL)*, qui, combiné avec *Globus Resource Allocation Manager (GRAM)*, permet d'éviter aux utilisateurs d'avoir à effectuer des choix explicites parmi des ressources parfois très variées et qui sont ici découvertes et gérées de manière transparente.

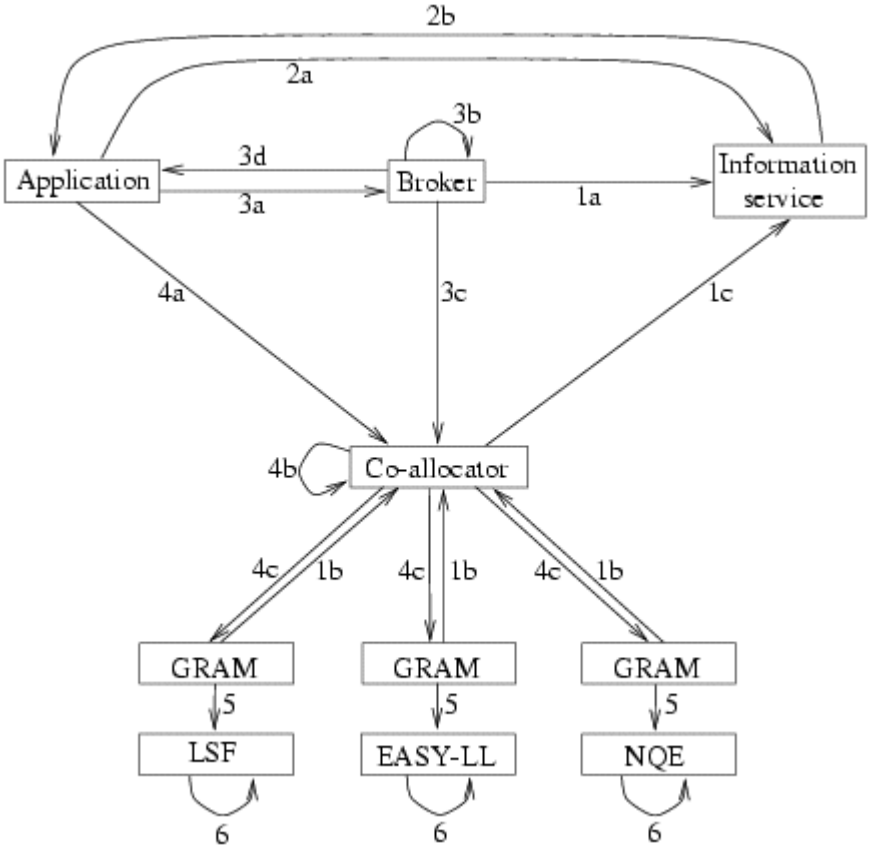

Figure 7 : Le système de gestion de ressources de Globus

La figure ci-dessus nous montre comment le système de gestion des ressources procède :

1. collecte d'informations sur les GRAM et les ressources disponibles.
2. sélection du *Broker* par l'application.
3. construction d'un ordonnancement et sélection des co-allocateurs.
4. raffinement de l'ordonnancement par les co-allocateurs.
5. traduction des ordres RSL (*Resource Specification Language*) par les outils de gestion locale des ressources.
6. gestion spécifique des tâches par l'outil local.

Le service GRAM (*Globus resource Allocation Manager*) est le pivot de Globus, qui assure le dialogue entre les méta-ordonnanceurs d'une part et les modules de répartition de charge tels que Condor-G, LSF, Maui ou PBS d'autre

part, qui sont disponibles à un niveau local, par exemple sur une grappe faisant partie d'une grille.

Globus est une réponse académique visant à fédérer le champ de recherche des multiples équipes travaillant sur les métaordinateurs, et à concentrer les efforts sur des mécanismes d'accès standard. Cependant, la vision du monde Grille telle qu'elle est vue par Globus ne résout pas les problèmes de gestion inter-organisations (qui sera l'administrateur d'une architecture globale ?). La gestion autonome du système n'est pas non plus un objectif de Globus, qui préfèrera une administration manuelle des outils plutôt que des systèmes auto-maintenants et auto-organisants.

3. Le système I-Cluster

Le système I-Cluster est un ensemble de mécanismes distribués, permettant la mise en œuvre de services de calcul intensif par agrégation des ressources inexploitées du réseau.

3.1. Architecture du système I-Cluster

3.1.1. Composants du système

La finalité de notre étude est d'obtenir un système distribué permettant de faire tourner des applications parallèles sur un intranet d'entreprise, composé de machines de travail génériques et non dédiées, en prévoyant et en exploitant les périodes d'inactivité des machines.

Plusieurs composants principaux sont nécessaires au système I-Cluster, certains résidant localement sur chaque machine du système, d'autres étant des assemblages virtuels de composants logiciels distribués fournissant un service global :

- Le **bac à sable d'exécution de tâches** est un composant disponible sur chacune des machines du système qui va proposer ses ressources pour effectuer des calculs. Il a plusieurs rôles ; d'une part l'analyse du profil d'utilisation de la machine par son possesseur afin de déterminer les périodes d'utilisation et de disponibilité de la machine ; en conséquence la prédiction des *fenêtres de tir* (nuit, week-end) pendant lesquelles la machine pourra être utilisée à des fins de calcul communautaire ; et enfin le mécanisme d'isolation qui compartimente le système d'exploitation de l'utilisateur par rapport à l'environnement de calcul distribué, en respectant des contraintes de disponibilité de la machine pour son (ses) utilisateur(s) et de sécurité par protection des programmes et données.
- Le **nuage ou *Cloud***, qui est une base de données distribuée en mode pair-à-pair. Le nuage est une entité virtuelle basée sur la coordination d'agents présents sur chacune des machines du système à l'aide d'un système bavard. Le nuage permet le passage à l'échelle sur plusieurs dizaines de milliers de machines ainsi que la résistance aux fautes fréquentes des machines (volatilité). La finalité de cette base de données dans le système I-Cluster est de pouvoir connaître les caractéristiques et la disponibilité des machines présentes sur le réseau, afin de les identifier pour leur soumettre éventuellement des

travaux de calcul par allocation à des tâches en mode de grappes virtuelles.

- L'**allocateur de machines ou *Match Finder***, qui est un composant local à chaque machine de soumission de travaux (toute machine pouvant être utilisée à cet effet). Le match Finder va se charger de recevoir la description d'une tâche demandée par un utilisateur, puis va se baser sur l'état courant des machines du réseau –disponible à l'aide des informations du nuage– pour identifier une *grappe virtuelle* composée de machines en jachère qui permettra de répondre au mieux à la requête. Le Match Finder assure aussi l'allocation des machines à la requête et effectue le placement final des tâches nécessaires sur les machines de la grappe virtuelle identifiée.

La figure suivante met en situation ces différents composants sur des machines d'un intranet. A noter que dans cet exemple, une machine est utilisée comme machine de soumission des travaux mais ne dispose pas du bac à sable, et ne peut donc pas être exploitée pour exécuter des tâches de calcul lors de ses périodes d'inactivité.

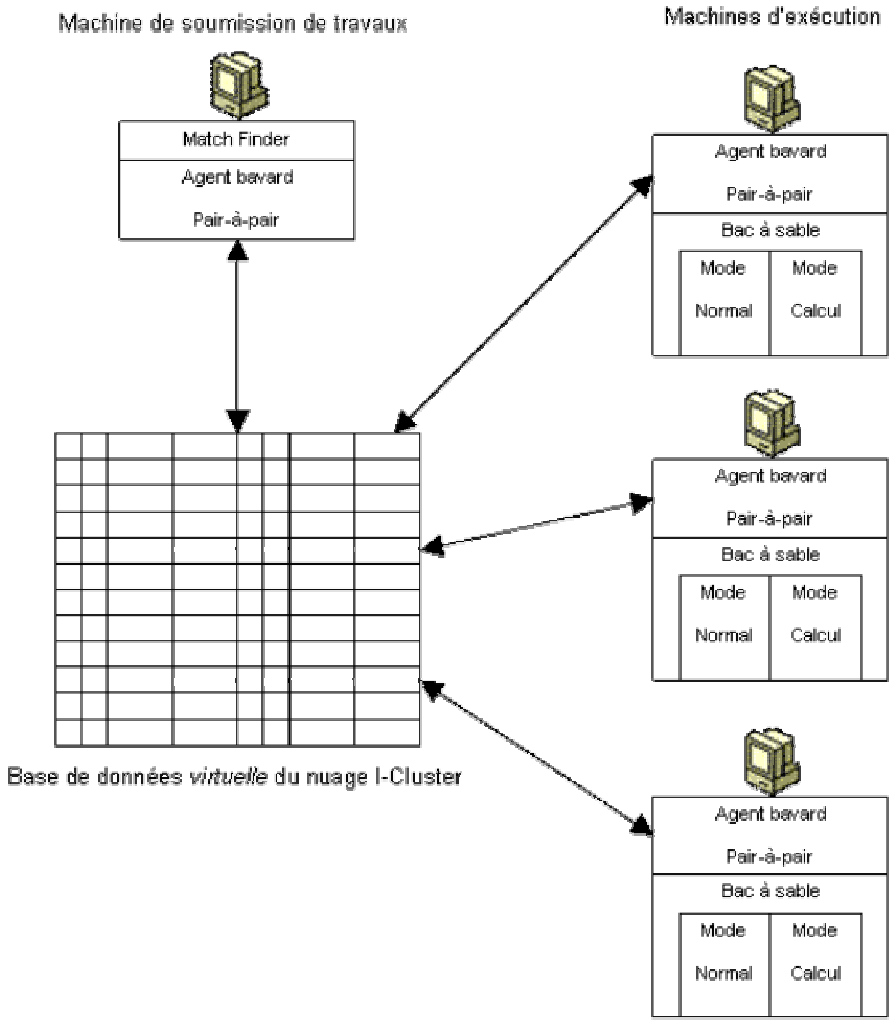

Figure 8 : Architecture du système I-Cluster, montrant les principaux composants

3.1.2. Mode opératoire du système

Le système I-Cluster a plusieurs aspects, qui se révèlent si l'on reprend la chronologie des évènements.

D'une part, les agents du nuage I-Cluster sont continuellement actifs, et dialoguent les uns avec les autres afin de faire vivre la base de données distribuée et de refléter au mieux l'état des machines du système.

Le bac à sable de chaque machine va, lors de l'arrivée dans une de ses fenêtres de tir, passer de mode normal en mode calcul. Cette information sera aussitôt diffusée dans le nuage afin que la machine ainsi rendue disponible puisse être mise à disposition afin d'effectuer des calculs.

De la même manière, à la fin d'une fenêtre de tir, le bac à sable va repasser la machine en mode normal afin que l'utilisateur trouve sa machine dans le même état que lorsqu'il l'avait quittée (la veille au soir par exemple).

Dans le cas où l'utilisateur revient à son bureau pendant une fenêtre de tir, par exemple s'il revient travailler à son bureau la nuit de manière inhabituelle, l'activité utilisateur va être détectée par le bac à sable et remettre la machine en état normal, en abandonnant les éventuels travaux en cours.

Lors d'une requête au Match Finder par un utilisateur, un certain nombre de machines disponibles pour le calcul (dont le bac à sable est en mode calcul) seront identifiées à l'aide de la base de données de ressources du nuage et seront allouées à ce travail.

Lors de la fin d'un travail, les machines qui avaient été allouées se remettent en état disponible pour d'autres calculs.

3.2. Le bac à sable d'exécution de tâches

Nous avons donc décidé d'exploiter la puissance de calcul disponible sur les machines inexploitées d'un intranet, et de les agréger en grappes virtuelles permettant d'exécuter des travaux à grain fin. Toutefois, ces machines ne sont pas exploitables telles quelles et certains facteurs nous limitent dans cette optique :

- La vocation première d'une machine de l'intranet est généralement d'être utilisée comme machine de bureau. Son utilisateur est psychologiquement "possesseur" de la machine, même si la machine a été achetée par son entreprise. Dans notre contexte, l'aspect positif est que l'utilisateur est responsable de la mise à jour de sa machine (achat de nouveau matériel), ainsi que de la préservation de l'état de marche de la machine : si la machine tombe en panne, c'est à son utilisateur de déclencher les actions nécessaires à la réparation. C'est en particulier lui qui redémarrera la machine en cas de faute logicielle. Ce mode est très différent du mode classique de fonctionnement d'une grappe, où un administrateur sera responsable de la globalité de la grappe. En

termes d'économie de personnes, c'est un gain notable.
Cependant, l'utilisateur de la machine étant son possesseur, il n'est pas acceptable pour lui de ne pas pouvoir disposer pleinement de sa machine. En d'autres termes, nous ne pouvons pas réserver la machine à certaines périodes pour l'exécution exclusive de travaux de calcul. Nous devons donc prévoir un moyen de pouvoir détecter et prédire les périodes d'inactivité de la machine de manière assez fiable pour ne pas gêner l'utilisateur. De plus, en cas de requête implicite de la machine par l'utilisateur (travail le week-end par exemple), il faut pouvoir rendre le contrôle rapidement à l'utilisateur.

- pour des raisons similaires, il est important de limiter les causes de pannes de la machine due à l'infrastructure que nous ajoutons. Dans tous les cas, il faudra faire en sorte que les composants se configurent automatiquement, et se réparent automatiquement. Les états puits (états récurrents de la machine dont on ne peut sortir sans l'intervention d'un administrateur) doivent être évités à tout prix. En cas de faute du système, au pire un redémarrage de la machine doit conduire à un état connu du système, tel qu'il aurait été sans la présence de I-Cluster.

- les travaux que nous désirons lancer en mode cluster seront des travaux à grain fin. Nous avons vu (chapitre 3.4) qu'un élément majeur de la performance en grain fin d'une grappe était l'équilibre entre les machines de la grappe. Pour obtenir cet équilibre, il est important de contrôler la charge de la machine causées par les applications utilisateur et les ressources systèmes qu'elles impliquent (nous verrons que l'architecture que nous avons choisie permet de ne pas avoir à subir cette charge).

- un dernier facteur important se situe au niveau de la sécurité. Il est essentiel de protéger les données de l'utilisateur présentes sur son disque dur, ses applications, sessions, contre toute faute ou malveillance du système de calcul que nous ajoutons. De la même manière, il faut protéger les applications et les données du système de calcul des intrusions de l'utilisateur. Nous avons donc besoin d'un système d'isolation de tâches et de données. Pour ce qui est de la protection des applications de calcul nous avons vu que l'utilisateur peut réclamer implicitement sa machine lors d'un retour non anticipé. Dans ce cas il nous faudra un système de gel/reprise (*checkpointing*) d'application distribuée. Je n'ai pas abordé ce problème particulier dans le cadre de ma thèse, laissant à Jean-Michel N'Long2 le soin de composer la sienne autour de ce thème au sein du laboratoire ID-IMAG.

3.2.1. Les composants du bac à sable

Le modèle du bac à sable (*sandbox model*) permet de décrire un environnement d'exécution de code ainsi qu'un ensemble de règles permettant la programmation d'un code s'exécutant dans le bac à sable. Un bac à sable permet d'exécuter du code à bas niveau de fiabilité (*untrusted code*) sur une machine, en disposant de fonctionnalités données (par exemple un accès réseau), tout en protégeant la machine cible ainsi que les données qui y sont stockées contre toute attaque de sécurité qui pourrait provenir de code malicieux ou déficient chargé par le bac à sable.

Nos objectifs sont d'obtenir un tel bac à sable, en favorisant la transparence pour l'utilisateur de notre système, la stabilité afin de ne pas provoquer des

fautes sur les machines utilisées, la sécurité des données et des applications et aussi l'efficacité lors de calculs intensifs.

Notre bac à sable est structuré autour de trois composants principaux, qui seront décrits en détail par la suite :

- Le premier composant va avoir la responsabilité d'établir le profil d'utilisation de la machine, afin de détecter les périodes d'inactivité récurrentes (nuits, week-ends, absences ou réunions hebdomadaires, etc.) et de déterminer les fenêtres de tir disponibles pour chaque machine. Par exemple, sur une machine donnée ce composant pourra déterminer que chaque nuit de 20h00 à 8h00 du matin est une fenêtre de tir valable, ainsi que les samedi et dimanche, et les mercredis après-midi en fonction de la présence de l'utilisateur et de l'utilisation faite de la machine (*backups* nocturnes). Il détectera aussi les périodes de vacances de l'utilisateur afin que sa machine soit alors mise à disposition pour effectuer des calculs.

- Un second composant va déposer le logiciel nécessaire au mode de calcul sur le disque dur de chaque machine. Une manipulation spécifique du disque lors de l'installation de notre système sur la machine va permettre l'allocation de l'espace de stockage nécessaire, et va installer la mécanique nécessaire à son contrôle tout en cachant sa présence sur le disque dur à l'utilisateur. Une fois installé ce composant devient transparent, voire invisible (aucune modification n'est apparente dans le système de partitionnement du disque).

- Un dernier composant va se charger du changement de mode de la machine. Basé sur la détection des fenêtres de tir, ce composant va provoquer l'hibernation de la machine et la passer dans le mode de calcul et offrir les ressources de la machine à la communauté. De la même manière, lorsque la fin de la fenêtre de calcul approche il va annuler les éventuels travaux en cours et remettre la machine dans son mode utilisateur, en restaurant l'état de la machine tels qu'il a été hiberné. Ce composant a aussi la responsabilité de réveiller la machine si elle est éteinte lors de l'arrivée d'une fenêtre de tir et de la passer en mode calcul. Ceci est fait à distance, suivant le protocole *Remote Wake Up* utilisant une trame Ethernet de format spécial. Il peut donc aussi gérer un réveil par une autre machine du réseau.

En plus de ces composants, nous disposons d'un service de boîte aux lettres qui nous permet de communiquer entre nos composants s'exécutant en mode utilisateur et en mode cluster. Ce service nous permet en pratique de communiquer les paramètres réseau de la machine depuis le mode utilisateur vers le mode calcul.

3.2.1.1. L'analyse du profil d'utilisation de la machine

Il nous faut déterminer le profil d'utilisation de chacune des machines pour en déduire ses fenêtres de tir. Contrairement à de nombreux projets tels NOW [ACP95] qui font une analyse de la charge des machine en pourcentage de charge du processeur, nous devons analyser les périodes d'inactivité de la machine, c'est-à-dire les périodes pendant lesquelles l'utilisateur n'utilise pas sa machine, et que celle-ci n'est pas en train de travailler. Les périodes d'inactivité de plus de 1 heure qui se reproduisent de semaine en semaine seront identifiées comme des fenêtres de tir, pendant lesquelles nous pourrons pleinement disposer de la machine pour effectuer des calculs.

Points de capture

Afin de construire le profil d'utilisation de la machine, un processus moniteur (*monitoring thread*) s'exécute en tâche de fond (daemon système) et se base sur les points de capture suivants :

- Economiseur d'écran. Notre tâche analyse les messages Windows émis par le système d'économie d'écran. Lorsqu'un économiseur d'écran démarre ou s'arrête, nous en sommes donc notifiés. D'autre part, nous analysons aussi le fait qu'un utilisateur est logué ou non, car le module gestionnaire des économiseurs d'écran n'est activé qu'après la connexion d'un utilisateur.
- Activité utilisateur. Notre moniteur est enregistré pour recevoir les notifications du système lors de l'appui sur les touches du clavier ainsi que sur l'activité de la souris.
- Activité système. Notre moniteur accède aux compteurs d'activité CPU, et ne considère la machine comme inactive qu'au dessous d'un certain seuil de charge. Ce seuil est dans la pratique réglé à 10% d'activité CPU sur une période de 1 minute. Nous avons rencontré quelques problèmes avec ce point de mesure dans le cas où l'utilisateur avait configuré un économiseur d'écran qui consomme une grosse puissance CPU (flowerbox openGL par exemple). Dans ce cas le taux de CPU reste constamment au-dessus de 10%. Nous n'avons pas trouvé de bonne manière de résoudre ce cas si ce n'est que de déconseiller l'utilisation de tels économiseurs, qui n'épargnent d'ailleurs ni l'écran ni le CPU.
- Activité réseau. Le moniteur est capable de déterminer si des sessions réseau sont ouvertes en tant que client, ou encore si des processus serveur sont en attente de connexion de clients. Ceci permet d'éviter de considérer que la machine ne soit considérée comme inactive alors qu'un gros téléchargement est en cours par exemple. De la même manière si un serveur HTTP est disponible, en attente de réception sur un port réseau, la machine sera alors considérée comme en activité. Il est possible de configurer le moniteur de manière à ce qu'il ignore l'activité de certains numéros de ports ou l'activité réseau de certains processus. Un serveur HTTP ou FTP installé sur la machine ne sera alors pas considéré comme une activité de la machine.
  Il est à noter que, faute de trouver des spécifications assez précises sur le système Windows, nous ne sommes pas arrivés à implémenter cette fonctionnalité de manière fiable dans notre prototype, et nous avons donc dévalidé la scrutation de l'activité réseau.
- Changement de connectivité. Lorsque la machine est débranchée du réseau (cas du portable amené à la maison) le moniteur en prendra note et ne considèrera pas la machine comme disponible pour effectuer des calculs.
- Mode de sauvegarde de l'énergie. Notre moniteur observe les messages émis par le gestionnaire de sauvegarde de l'énergie (power management) de Windows. Il sera donc notifié lors des changements

d'état de la machine dans les modes *On*, *Standby*, *Suspend*, *Hibernate*, et *Off*.

Il est à noter que notre processus moniteur ne s'exécute que dans le mode utilisateur de la machine, et pas dans le mode calcul. Or l'objectif de notre moniteur est de prédire les périodes pendant lesquelles la machine peut être mise à disposition pour effectuer des calculs, autrement dit les périodes d'inactivité de la machine. On remarquera que, si la machine est en mode calcul, on peut forcément la considérer comme disponible pour des calculs. Aussi les évènements de passage en mode calcul et de retour en mode utilisateur peuvent-ils être ignorés, car les autres évènements tels que l'arrêt de la machine ou son activité en mode utilisateur sont suffisants pour déduire l'inactivité. En d'autres termes, les périodes passées en mode calcul sont considérées comme des moments où la machine est éteinte du point de vue de sa disponibilité.

De la même manière, si un utilisateur revient de manière inhabituelle à sa machine, par exemple de nuit, il peut la trouver en mode de calcul. Un simple appui sur le clavier provoquera le retour immédiat de celle-ci en mode utilisateur et sa sortie normale d'hibernation (ce mécanisme sera décrit en détails plus loin). L'évènement de sortie provoquée par l'utilisateur du mode de calcul ne sera pas observé par notre moniteur, cependant l'évènement immédiatement suivant de sortie d'hibernation suffira pour pouvoir considérer la machine comme active et indisponible pour le calcul dans notre profil.

A l'aide de nos points de mesure, notre moniteur va pouvoir construire le profil d'activité de la machine et le stocker. A cette fin, nous avons une base de données qui permet de stocker le mode de la machine quart d'heure par quart d'heure depuis les N dernières semaines (N est typiquement égal à 4). Cette base de données sera ensuite utilisée pour reconnaître les autosimilarités de l'état de la machine de semaine en semaine et d'en déduire les fenêtres de tir disponibles.

Le profil d'activité de la machine pourrait être construit en stockant toutes les informations relatives aux points de mesure observés, ainsi qu'à l'heure système à laquelle ils ont été observés. Toutefois, nous avons préféré gérer l'état de la machine à l'aide d'un automate à états finis réagissant aux évènements décrits ci-dessus, et qui ne comporte que trois modes possibles :

- **Libre** (*Free*). Dans cet état, la machine n'est pas utilisée par l'utilisateur mais est probablement utilisable pour du calcul. A noter que dans le cas d'un ordinateur portable ramené chez soi le soir, la machine ne sera pas connectée et ne sera donc pas disponible pour des calculs.
- **Actif** (*Busy*). Dans cet état, la machine est considérée comme étant utile à l'utilisateur. Ceci n'induit pas forcément la présence de l'utilisateur, par exemple dans le cas d'une machine en train de faire une grosse opération de nuit, une sauvegarde ou un service par réseau.
- **Inconnu** (*Unknown*). Cet état consolide tous les états possibles de la machine où nous ne pouvons pas fiablement considérer que la machine est soit libre soit active. Ce mode est le mode d'initialisation

du profil, c'est-à-dire que lors de l'installation de notre système sur une nouvelle machine le profil de celle-ci la considèrera en état inconnu pendant les N dernières semaines.

En outre, nous avons ajouté un délai virtuel de 30 minutes lors du passage du mode libre au mode actif. En effet, rappelons qu'un objectif du système est de rester transparent aux yeux de l'utilisateur. Afin d'éviter un passage trop agressif de la machine en mode calcul, nous voulons éviter qu'un utilisateur qui utilise régulièrement sa machine jusqu'à 18h00 ne voie celle-ci passer en mode de calcul dès 18h00. A l'aide de notre délai virtuel, le profil de la machine considérera celle-ci comme active pendant 30 minutes après que l'utilisateur ait cessé de l'utiliser.

Automates de gestion

Les évènements sont gérés par plusieurs automates à états finis. Chacun de ces automates va permettre de déterminer un sous-état relatif à l'automate en cours. En d'autres termes, la gestion de l'activité clavier/souris par exemple sera faite par un automate, alors qu'un autre automate fera la gestion de la connectivité, un autre encore la gestion de l'activité réseau, etc. Les évènements observés ci-dessus par notre processus moniteur correspondent à des évènements réels. De manière à simplifier la conception des automates nous avons réduit ces évènements à des évènements virtuels. Voyons maintenant le détail de ces automates :

Gestion de l'activité utilisateur

Cet automate très simple[3] va gérer le login/logout des utilisateurs.

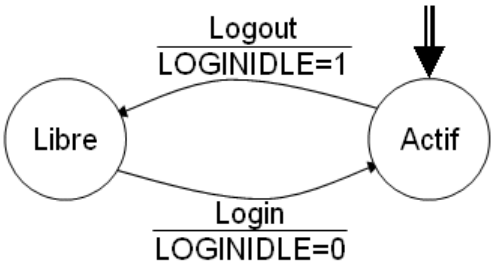

Figure 9 : Gestion de l'activité utilisateur

Gestion de l'activité processeur

Cet automate observe l'activité du processeur pendant une période fixe (1 minute) et détecte les moments de non-activité. Notons que CPUHIGH et CPULOW sont des transitions et non pas des états.

[3] *Note : Les états des automates seront représentés par les bulles, qui disposent d'une étiquette. Les transitions seront représentées par des flèches unidirectionnelles. Attaché à chaque flèche de transition on trouvera au-dessus de la barre horizontale le nom de l'évènement provoquant la transition, et au-dessous de cette barre l'action à entreprendre lors de cette transition. Dans le cas où un évènement ne provoque pas de changement ni d'action il ne sera pas représenté par souci de clarté des schémas. La double flèche représente une transition virtuelle vers l'état initial de l'automate.*

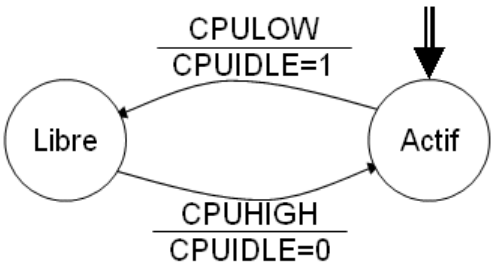

Figure 10 : Gestion de l'activité processeur

Gestion de l'activité réseau

Cet automate permet de détecter les périodes pendant lesquelles une session réseau est ouverte, et où la machine doit donc être considérée comme active.

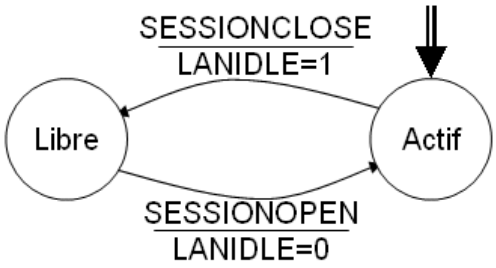

Figure 11 : Gestion de l'activité réseau

Gestion de l'activité clavier/souris

Cet automate permet de détecter les périodes pendant lesquelles l'utilisateur n'a pas utilisé ses clavier et souris depuis plus de 10 minutes. Un temporisateur est redémarré à chaque occurrence d'un évènement. Lorsque ce temporisateur atteint sa limite de 10 minutes un évènement TIMEOUT est généré.

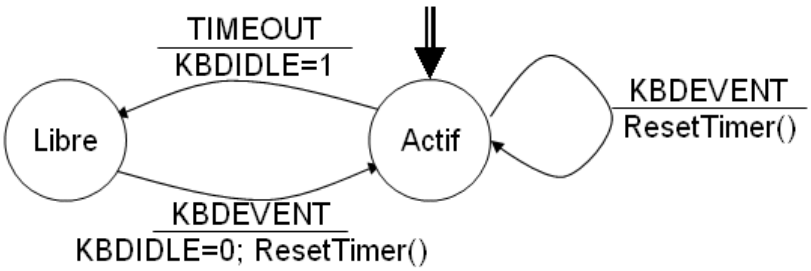

Figure 12 : Gestion de l'activité clavier/souris

Gestion de l'activité du gestionnaire d'énergie

Nous avons ici fortement réduit l'automate réel de gestion de l'énergie. Quel que soit l'état de sauvegarde d'énergie dans lequel part la machine, nous la considérons comme libre. En effet, les modes *Standby*, *Suspend*, *Hibernate* et *Off* sont très différents au niveau Windows, mais expriment tous un état d'inactivité de la machine.

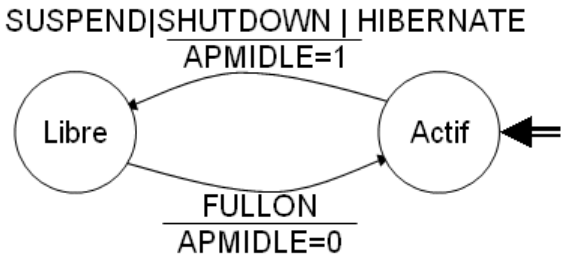

Figure 13 : Gestion de l'activité du gestionnaire d'énergie

Le macro-état Inconnu ne correspond pas à un état possible des automates. Il pourra être toutefois stocké dans la base de données afin de permettre une politique de changement d'état plus ou moins active en fonction des besoins de l'utilisateur. Ce mécanisme est repris en détail au chapitre suivant.

Afin de permettre un discours simple ici nous avons simplifié la représentation de la gestion des états dans la base de données, ainsi que nous pouvons le voir dans la figure ci-dessous. La base de données d'historique de l'activité de la machine est considérée comme étant un tableau infini, indexé par date d'évènement. Chaque enregistrement contient l'état de la machine pendant le quart d'heure considéré, un 1 indiquant une machine disponible pour les calculs (*Idle*), un 0 indiquant une machine utilisée au moins une fois par l'utilisateur pendant la période. Initialement l'état de la machine est inconnu et représenté par un point d'interrogation.

Bien sûr dans la pratique la base de données n'est pas infinie. L'implémentation courante est basée sur un système de fenêtres roulantes sur une période de 2 à 32 semaines (typiquement 4 semaines). La gestion de ces fenêtres roulantes n'est pas explicitée ici, et le lecteur pourra se reporter à [Knu73] pour une description plus approfondie.

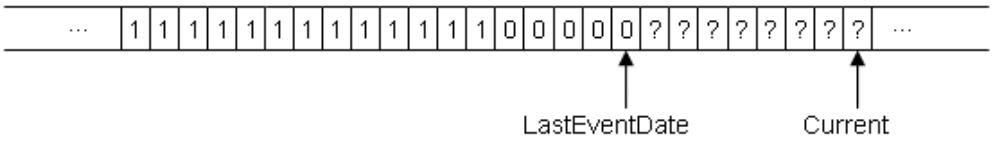

Figure 14 : Exemple de représentation de la base de données d'historique d'une machine

Dans cet exemple on voit tout d'abord une période où la machine à été détectée comme inactive (suite de « 1 »), puis une période où de l'activité utilisateur à été vue (suite de « 0 »). Le pointeur « LastEventDate » indique l'instant de dernière écriture dans la base de données. Le pointeur « Current » indique l'instant présent.

Le petit morceau de code suivant (pseudo langage C) va permettre de consolider l'état de nos différents automates en reconnaissant les modes d'inactivité complète de la machine (lorsque LOGINIDLE, CPUIDLE,

LANIDLE, KBDIDLE, APMIDLE sont tous à 1). Notre délai virtuel de 30 minutes, décrit plus haut, n'est pas intégré à ce pseudo code pour faciliter la compréhension du lecteur. Cette routine est globale à tous les automates et est appelée lors d'un changement d'état de l'un d'eux.

```
void calledOnEvent()
{
  // oldState va garder l'état entre les appels de cette routine
  static state oldState = UNKNOWN;

  // lastEventDate va garder le quart d'heure de dernière modification
  static quartDheure LastEventDate = quartDheure(gettimeofday());

  // CurrentStateIdle va contenir l'état courant de la machine
  state CurrentStateIdle = LOGINIDLE && CPUIDLE && LANIDLE
                         && KBDIDLE && APMIDLE;

  // Current contient le quart d'heure courant
  quartDheure Current = quartDheure(gettimeofday());

  if(CurrentStateIdle == 0)   // Si le quart d'heure courant est actif
  {
    for(i = oldState + 1; i <= Current; i++)
      Activity[i] = 0;        // Mettre à jour les N dernières entrées
    oldState = 0;             // On prépare l'appel suivant
    LastEventDate = Current;
  }
  else
  {
    for(i = oldState + 1; i < Current; i++)
      Activity[i] = 1;        // Mettre à jour les N-1 dernières entrées
    oldState = 1;             // On prépare l'appel suivant
    LastEventDate = Current;
  }
}
```

On remarquera que le comportement est légèrement différent selon que l'état est actif ou non. En effet, si l'état courant est actif nous marquerons le quart d'heure courant comme tel dans la base de données. Par contre si l'état courant n'est pas actif, nous mettrons à jour la base de données mais pas le dernier quart d'heure. Ceci parce que l'état actif est prévalent sur l'état inactif : Pendant le quart d'heure courant, la machine peut voir des instants d'activité mais aussi d'inactivité. Dans ce cas, la base de données sera toujours mise à jour en indiquant un état actif pour la totalité du quart d'heure considéré.

Réglage initial de la base de données

La base de données, rappelons-le, contient l'état de la machine quart d'heure par quart d'heure sur les N dernières semaines (typiquement N=4 semaines). Initialement la machine est considérée en état inconnu pendant toute cette période, et la base de données est remplie de cette manière.

Toutefois, comme nous le verrons ci-dessous, la prédiction des fenêtres de tir de la machine est basée sur l'information disponible depuis la base de données de la machine, et va se baser sur les autosimilarités entre les états observés de la machine dans les N dernières semaines de scrutation. Or le déploiement de notre système peut être fait dans différents contextes : dans le cas des machines d'un campus universitaire par exemple, on voudra probablement privilégier la disponibilité de nombreuses machines à des fins de calculs distribués. Dans ce cas, la détection des fenêtres de tir devra être

agressive. Par contre, dans le cas d'un déploiement sur les machines des utilisateurs d'une entreprise, on voudra probablement privilégier l'aspect « invisible » de notre système, et la détection des fenêtres de tir devra alors être plus passive. C'est long comme lacune.

Afin de pouvoir sélectionner l'agressivité de notre détecteur de fenêtres de tir, il nous faut changer l'état initial de la base de données. Si cet état reflète une machine pleinement disponible dans les N dernières semaines, toute période d'inactivité de la machine sera suivie, après notre délai d'activité virtuel de 30 minutes, d'une fenêtre de tir de durée d'une semaine (voir Figure 15). Autrement dit, la machine pourra être entraînée en mode de calcul 30 minutes après la dernière activité détectée de l'utilisateur. Les étudiants de machines du campus retrouveront donc leur machine en mode calcul après leur pause déjeuner ou le matin en arrivant sur le campus. Il est à noter toutefois qu'après avoir construit une connaissance du profil réel d'utilisation de la machine sur N-1 semaines, notre système aura ensuite fiablement détecté les fenêtres de tir possibles.

Si, par contre, l'état initial de la base de données reflète une machine pleinement indisponible dans les N dernières semaines, il faudra N-1 semaines, soit typiquement 3 semaines, avant qu'une fenêtre de tir devienne possible. L'intérêt de ce réglage est que notre système restera très transparent à l'utilisateur et attendra d'avoir construit une connaissance du profil réel de la machine sur N-1 semaines pour commencer à détecter des fenêtres de tir. Par contre le gros inconvénient est que le système, une fois installé sur une machine, va mettre très longtemps avant de pouvoir offrir les ressources disponibles de la machine.

Afin d'éviter ces deux extrêmes et de pouvoir d'une part rester invisible aux yeux de l'utilisateur courant, et d'autre part offrir les ressources inexploitées de la machine dès le premier jour de l'installation de notre système, nous avons choisi d'initialiser la base de données de telle manière que la machine est en état disponible sauf pour les périodes entre 7h00 et 20h00, qui sont vues en état actif. De cette manière, avant d'avoir construit une connaissance du profil réel d'utilisation de la machine sur N-1 semaines, notre système ne trouvera des fenêtres de tir que sur des périodes nocturnes.

Un modèle de réglage initial de notre base de données plus élaboré aurait pu être choisi afin de pouvoir satisfaire des utilisateurs de profils différents, mais nous avons préféré en rester à un mécanisme simple et satisfaisant dans la majorité des cas d'utilisation.

Détection des fenêtres de tir disponibles

Nous avons donc construit une base de données qui nous permet de connaître l'état d'activité de la machine quart d'heure par quart d'heure, sur les N dernières semaines. Chaque quart d'heure est référencé comme libre (machine disponible pour les calculs), actif (l'utilisateur utilise la machine) ou encore inconnu. A partir de ces informations, nous allons devoir déterminer les périodes pendant lesquelles la machine est habituellement libre (par exemple tous les lundi soir de 20h00 au mardi matin 8h00). Il nous faut donc auto-

corréler ces informations afin de déterminer les plages récurrentes de disponibilité de la machine.

Nous avons expérimenté plusieurs politiques différentes pour l'auto-corrélation de nos données. Dans une première implémentation, l'analyse se basait sur le taux d'utilisation de la machine pendant chaque quart d'heure, moyenné sur les N semaines d'analyse stockées dans la base de données. Si cette moyenne était au-dessous d'un certain seuil τ, la période considérée était identifiée comme une fenêtre de tir. Toutefois ce mécanisme comportait un inconvénient au niveau de l'utilisateur : Si celui-ci laissait sa machine inactive pendant plus d'une semaine (lors d'un déplacement ou pendant ses vacances par exemple), tout instant devenait éligible comme fenêtre de tir. Aussi, la machine passait en mode de calcul dès qu'elle était inactive pendant plus de 30 minutes (durée de notre délai virtuel vu au chapitre 0). La transparence de notre système n'était alors plus préservée. De plus, pour « nettoyer » la base de données il fallait alors attendre N-1 semaines (typiquement 3 semaines) avant de retrouver une base de données contenant des informations d'utilisation régulières.

Une deuxième politique de détection a alors été implémentée. Celle-ci identifie un quart d'heure comme une fenêtre de tir si la machine n'a jamais été vue comme active dans les N dernières semaines à la même heure. Dans ce cas, le problème a été la gestion des périodes exceptionnelles d'utilisation de la machine. Par exemple, si un utilisateur revient travailler un dimanche, cette période sera identifiée comme inactive, et la machine se rendra toujours indisponible pour des calculs le dimanche, et ce pendant N-1 semaines. Ceci ne nuit pas à la transparence du système, mais entraîne un abaissement malheureux de son nombre de fenêtres de tir potentielles.

Une dernière politique de détection a donc été implémentée. Celle-ci identifie un quart d'heure comme une fenêtre de tir si la machine n'a été vue qu'au plus une fois comme active dans les N dernières semaines à la même heure. Cette politique nous permet d'identifier efficacement les fenêtres de tir même en présence d'exceptions, tout en restant assez transparente à l'utilisateur. Le seul problème subsistant étant que lors du changement de profil d'utilisation de la machine (par exemple quelqu'un qui ne travaillait jamais le mercredi qui se met a venir ce jour-la de manière régulière) devra potentiellement provoquer la sortie de sa machine un mercredi deux fois avant que le profil ne soit correctement interprété.

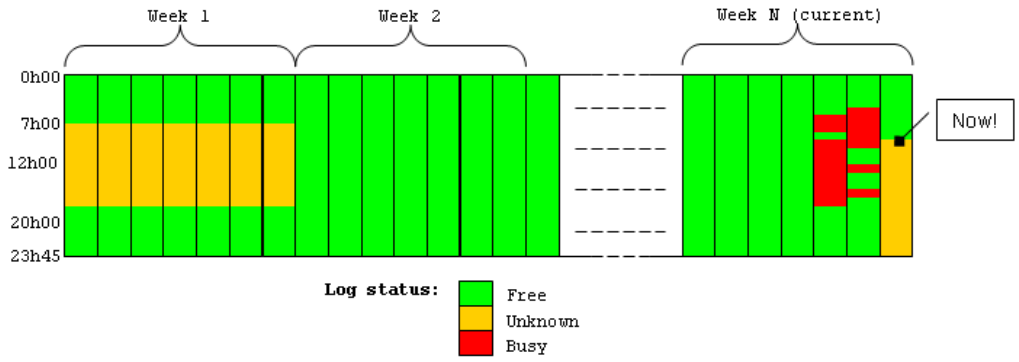

Figure 15 : Un exemple d'activité contenu dans la base de données

La figure ci-dessus nous montre un exemple de base de données. L'instant présent (NOW) correspond à un début de fenêtre de tir, et la machine va donc provoquer un passage en mode de calcul afin d'exploiter les ressources de calcul de la machine.

Périodes de garde

Dans notre implémentation, nous avons ajouté la notion de périodes de garde (*blackout*). Ces périodes correspondent à des états qui sont manuellement considérés comme actifs, et qui interdisent donc la détection de fenêtres de tir et donc le passage en mode de calcul de la machine. On peut donc, dans notre implémentation, configurer les jours de semaine de 8h00 à 20h00 comme des périodes de garde. Dans ces plages horaires, nous avons donc la garantie que la machine sera toujours disponible en mode utilisateur.

3.2.1.2. Partition I-Cluster

Nous appelons ici la partition I-Cluster l'ensemble de mécanismes permettant de déposer sur le disque dur d'une machine les composants nécessaires à l'opération de notre système. Voyons tout d'abord comment la décomposition entre les modes est réalisée.

Chaque PC de l'intranet dispose de deux personnalités différentes : Il peut soit être en mode utilisateur, c'est-à-dire dans un mode proche de celui d'un PC en fonctionnement. Dans ce mode, le système d'exploitation est quelconque, au choix de l'utilisateur ou de l'administrateur de la machine. Dans la pratique, plus de 96 % des postes clients sont équipés d'une version du système d'exploitation Microsoft Windows (Windows 3.1, 9x, NT, 2000 ou XP), d'après [GKS+01]. Une deuxième personnalité des PC est le mode I-Cluster, ou le PC est dédié au calcul, et est partagé de manière communautaire de même que les autres PC du réseau en mode I-Cluster.

Afin d'obtenir un bon cloisonnement entre les modes utilisateur et I-Cluster, nous avons développé une architecture permettant de « cacher » notre bac à sable I-Cluster sur le disque dur, de manière invisible à l'utilisateur. Pour ce faire, nous avons basé le système sur une extension des spécifications GPT [Int01] qui décrivent l'organisation d'un disque dur au niveau de ses partitions, de leur organisation sur le disque, ainsi que du choix des partitions de démarrage.

La figure ci-dessous présente les composants nécessaires dans chacun des modes d'opération de la machine. Dans le mode utilisateur, on trouve le système d'exploitation de la machine (Windows en l'occurrence) ainsi que les applications standard de l'utilisateur. Nous ne rajoutons qu'un agent I-Cluster qui permettra de gérer le profil d'utilisation de la machine, les passages en mode cluster ainsi que la gestion des ressources distribuée (le nuage I-Cluster) que nous détaillerons dans un chapitre ultérieur. Dans le mode cluster, caché à l'utilisateur, on trouvera un système Linux équipé avec les bibliothèques et composants nécessaires aux calculs distribués tels que MPI, PVM, ainsi que les outils de lancement et de copie de fichiers en parallèle. La boîte aux lettres est un espace de stockage accessible par notre agent I-Cluster ainsi que par le mode cluster, et qui permet de transférer des informations entre les deux modes. Cette

boite aux lettres est typiquement utilisée pour transférer les paramètres réseau de la machine depuis le mode utilisateur vers le mode cluster.

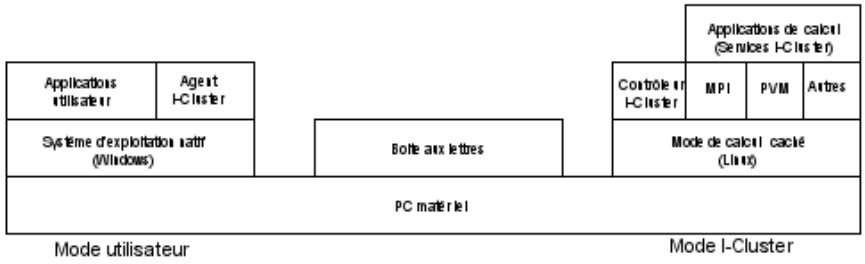

Figure 16 : Les composants de I-Cluster

Le bac à sable I-Cluster peut être vu comme l'image d'une partition de taille fixe (1 Go), contenant l'ensemble des outils permettant au PC de se comporter comme un élément d'une grappe lorsqu'il est en mode I-Cluster. Entre autres, cette partition contient les middlewares nécessaires tels que MPI, ainsi que les outils de transfert/réplication de fichiers et de lancement de tâches parallèles. Le laboratoire ID-IMAG de l'INRIA Rhône-Alpes a développé les outils adaptés à ces fonctions dans notre contexte, tels que Ka-tools ou la distribution CLIC adaptée aux grappes de calcul.

La "partition" I-Cluster est créée de manière très transparente lors du déploiement de l'environnement I-Cluster : nous avons créé un outil qui permet la création d'un fichier contigu et inamovible au sein du système de gestion de fichiers utilisateur (Microsoft Windows). Nous copions ensuite de manière automatique l'image brute de la partition I-Cluster. Nous disposons donc maintenant d'une partition I-Cluster incluse dans la partition native de l'utilisateur.

De manière à éviter que la partition cachée n'apparaisse aux yeux de l'utilisateur dans sa table de partitions, et par là risquer des conflits avec les éventuels gestionnaires de boot, il nous a fallu trouver une manière de ne plus rendre celle-ci visible. Nous avons basé notre système sur une extension de « Extensible Firmware Interface » (EFI) [Int01]. EFI est un standard qui a été créé par Intel afin de permettre une évolution du système de partitionnement des disques, qui a été lui-même créé en 1981 sur le IBM PC XT et n'a pas évolué depuis. Les systèmes actuels récents (postérieurs à 1999) supportent tous les extensions EFI, et en particulier le nouveau système de partitionnement, ce qui nous permet de baser nos outils sur cette norme sans crainte de possibles incompatibilités sur certains systèmes.

Le partitionnement classique d'un disque dur est extrêmement simple : le premier bloc (cylindre 0, tête 0, secteur 0) du disque dur contient une structure de 512 octets définie ainsi :

Offset	Taille	Contenu
0	446	Code de démarrage du disque. Après l'initialisation du BIOS ce code est charge à l'adresse 7c00:0000, puis le BIOS lui passe la main. Ce code est ensuite responsable de l'identification et du chargement du bootstrap du système d'exploitation.
446	16	Descripteur de la partition 1
462	16	Descripteur de la partition 2
478	16	Descripteur de la partition 3
494	16	Descripteur de la partition 4
510	4	Signature AA55h indiquant la conformité du formatage du MBR

Table 1 : Table de partitionnement des disques

Comme nous le voyons, ce partitionnement contient un code de démarrage, qui est du code Intel x86 16 bits mode réel. Dans le cas de nouvelles architectures telles que le processeur Intel® Itanium® (IA64), il n'est plus possible d'exécuter de telles instructions. De plus, la structure de description des partitions limite celles-ci à une taille maximale de 2 téraoctets, ce qui deviendra bientôt une limite critique. D'autre part, la limitation a 4 partitions ne peut être aujourd'hui évitée qu'en créant des sous partitions logiques, amenant une complexité supplémentaire dans la gestion des disques.

Pour ces raisons, une nouvelle structure de partitionnement a été définie par EFI, la « *GUID Partition Table* » (GPT). GPT autorise le positionnement d'informations de partitionnement sur le disque de manière non figée ; ces informations sont représentées sous la forme d'un arbre constitué de descripteurs de zones physiques du disque, qui elles-mêmes contiennent éventuellement d'autres zones. La gestion de cet arbre de partitionnement du disque n'appartient plus à un aléatoire code de boot placé sur le MBR, mais le BIOS de la machine est responsable de la découverte de l'arbre, de son interprétation et du lancement des actions appropriées. Par exemple, on peut créer un arbre de partitionnement contenant, depuis la racine, une feuille « Windows » et une feuille « Linux ». La feuille Windows aura un sous-arbre avec deux feuilles décrivant les zones du disque « utilisateur » et « programmes ». Sous la feuille « Linux » on pourra trouver plusieurs feuilles pointant sur des zones de partitionnement « swap », « / », « /usr », « /var », « /home », « /tmp ». Un gestionnaire de démarrage, dépendant du BIOS de la machine, permettra la configuration des modes de boot de la machine, permettra

la sélection des partitions de boot et des options à passer au système d'exploitation qui va démarrer.

La détection de l'arborescence des partitions par le gestionnaire de démarrage du boot de la machine se base non pas sur une adresse absolue sur le disque comme c'était le cas pour l'ancien formatage MBR qui devait trouver une structure particulière à l'adresse (cylindre 0, tête 0, secteur 0), mais sur une détection de signature (GUID) par scrutation du disque. De cette manière, il est possible de construire des partitions GPT sur un disque tout en préservant une structure MBR. Il suffit pour cela de mettre les informations GPT et leur UUID dans une zone inexploitée du disque dur, en l'occurrence sur la piste 0 (cylindre 0, tête 0) après le MBR (qui se trouve dans le secteur 0 de cette piste).

C'est le mécanisme que nous utilisons pour placer des informations de formatage GPT sur le disque sans modifier la structure de partitionnement MBR.

La figure suivante reprend l'architecture de notre système permettant de cacher notre partition du mode de calcul sur le disque dur de l'utilisateur.

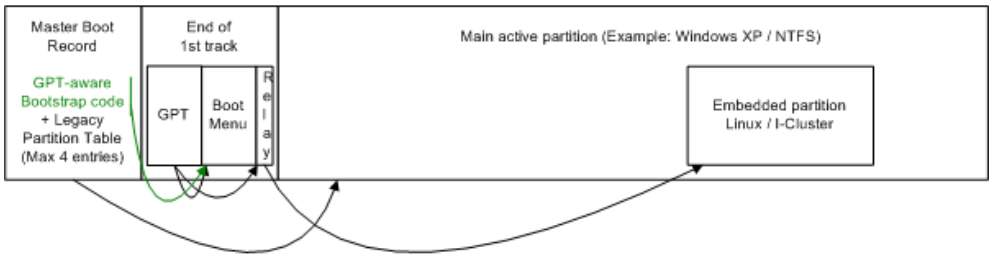

Figure 17 : Partitionnement du disque dur permettant l''invisibilité'' de notre partition

Sur cette figure on voit que le master boot record reste en place, et pointe sur la partition standard de l'utilisateur. Cette partition est inchangée par notre système. Sur la fin de la piste 0 (first track) nous avons la description de notre arbre de partitionnement GPT, qui pointe sur la partition créée dans un fichier caché dans le système de fichiers de l'utilisateur.

Afin de permettre le support des machines non équipées de EFI/GPT, nous avons ajouté un morceau de code dans le MBR original de l'utilisateur. Cette opération n'est pas indispensable si la machine est compatible GPT, car le BIOS sera alors capable de détecter notre arbre de partitionnement et de commander le démarrage sur la partition cluster ou sur la partition standard de l'utilisateur. Le morceau de code ajouté va effectuer la scrutation du disque pour trouver un GUID GPT, qui lui décrira l'adresse disque d'une zone de relais. Cette zone de relais contiendra la suite du code, car seuls 446 octets sont disponibles dans le MBR, et nous ne pouvions pas stocker toute la logique de notre mécanisme dans si peu de place.

Le noyau Linux présent dans la partition cluster est construit avec les modules d'origine IA64 gérant les GPT. Nous les avons modifiés pour supporter

nos extensions à la spécification GPT, c'est-à-dire la présence simultanée de partitions GPT et de partitions classiques, ainsi que le cas de partitions incluses les unes dans les autres. Ces modifications sont en fait assez simples, le plus dur étant le décodage des tables GPT, ce que nous n'avons pas eu à refaire.

Ainsi, Linux est capable de monter sa propre partition, et y accéder de manière native.

Un problème plus délicat se pose pour LILO, qui est le lanceur de Linux : LILO n'est pas utilisé sur les machines IA64. A la place, elles disposent d'un programme plus sophistiqué, faisant appel au BIOS EFI IA64. En l'absence d'un BIOS EFI IA32, nous ne pouvions pas le reprendre.

Plusieurs solutions ont été envisagées :

- fusionner LILO et notre menu de lancement. C'est un travail très important,

- enchaîner notre menu de lancement puis un LILO standard, dont les tables internes contiendraient l'emplacement physique de la partition de I-Cluster. Le travail est plus simple mais moins solide,

- créer un disque virtuel à partir de la partition I-Cluster, et mettre dedans un LILO standard.

C'est cette dernière solution, bien qu'apparemment plus complexe, qui a été retenue. En effet, le code d'émulation de disque virtuel au niveau BIOS IA32 était par ailleurs déjà disponible. Cela évitait une dépendance sur l'emplacement physique de la partition I-Cluster. Le LILO à l'intérieur du disque virtuel n'a qu'à connaître l'emplacement du noyau de Linux RELATIVEMENT au début du disque virtuel, et non au début du disque physique. Cela permet de distribuer un image fixe, utilisable sur tout PC, quel que soit l'emplacement de la partition I-Cluster.

Notre mécanisme permettant de cacher notre partition cluster sur le disque dur de l'utilisateur est d'une grande simplicité. Toutefois, il faut remarquer que la simplicité de description a été contrecarrée par une grande difficulté de mise au point. En effet, la spécification EFI étant récente, nous avons dû développer l'ensemble des outils nécessaires à notre mécanisme, avec tous les soucis et bugs que l'on peut imaginer.

3.2.1.3. Changement de mode

Le changement de mode est provoqué automatiquement, lors de la détection d'une fenêtre de tir par le système. L'heure de réveil est programmée afin de provoquer le retour de la machine en mode normal 30 minutes avant que l'utilisateur n'en ait besoin. Ensuite une hibernation est provoquée avec ordre de redémarrage immédiat. Le système va ensuite redémarrer sur la partition I-Cluster et permettre d'être alloué à des calculs et à les exécuter.

Dans notre prototype actuel il est aussi possible pour l'utilisateur de forcer le passage en mode calcul, en cliquant sur une icône.

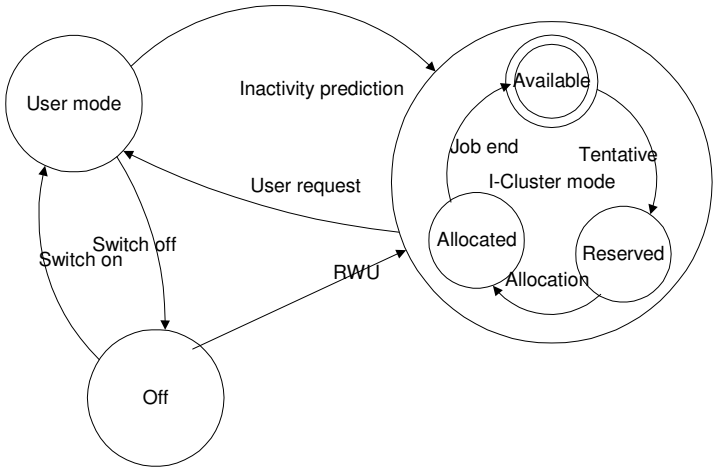

Figure 18 : Machine à états montrant les évènements système et leurs conséquences

Comme le montre la figure ci-dessus, certains évènements peuvent provoquer un changement d'état du système. Par exemple, I-Cluster peut détecter l'inactivité d'un PC à l'aide d'un module d'examen de l'utilisation du CPU, associé à une scrutation des évènements de la gestion d'énergie de Windows. Ce module va analyser les évènements d'endormissement/réveil générés par Windows, basés sur les réglages de l'utilisateur. A l'aide de ces informations, il peut décider d'un changement de personnalité. Un élément psychologique majeur de notre solution est que l'utilisateur reste le maître privilégié de la machine. Ceci sous-entend que la priorité des travaux I-Cluster sera toujours moindre que celle des travaux de l'utilisateur. Par exemple, si un PC a été alloué pour un travail I-Cluster mettant en œuvre 4 machines à partir de 20 heures, l'utilisateur de ce PC peut revenir a son bureau a 21 heures et se remettre au travail sur son PC, provoquant l'abandon du travail en cours. Une extension de I-Cluster permettra la migration de la tâche du PC qui se réveille suite à une demande de son utilisateur, mais cette extension reste pour l'instant un prospect du projet, et sa description est en dehors du cadre de ce présent document.

Réveil par le réseau

Les PC modernes se conformant aux spécifications *PXE* et *OnNow* permettent le réveil de la machine lorsqu'une trame spécifique circule sur le réseau. De cette manière, une machine participant à un I-Cluster peut réveiller une autre machine de son intranet pour la faire participer à une grappe virtuelle.

3.2.2. Déploiement automatique

Le déploiement de I-Cluster sur un intranet se fait de manière simple : Il suffit d'installer la partition I-Cluster sur chacune des machines participant au système I-Cluster. Nous avons eu comme objectif de garder un système hautement autonome, et cette installation a été réduite à sa plus simple expression : L'exécution d'un programme *setup.exe*, qui procède a la création de la partition, qui est cachée dans le système de fichiers de l'utilisateur, et ne nécessite donc pas de repartitionnement du disque. Quelques réglages sont

disponibles pour l'utilisateur de la machine au travers des réglages de son outil de sauvegarde d'écran : Il peut spécifier des plages horaires pendant lesquelles le mode I-Cluster ne sera jamais activé (par exemple entre 8h00 et 20h00 les jours de semaine), il peut spécifier si le réveil de la machine à distance par le réseau est activé.

3.2.3. Automatisation de la maintenance

I-Cluster est prévu pour se passer de l'administrateur partout où cela est possible. En particulier, nos mécanismes se basent sur le fait que la machine dans son mode utilisateur est fonctionnelle. Ceci sous-entend que la configuration réseau est bonne, et que les périphériques fonctionnent correctement. Dans ces conditions, le mode I-Cluster permet de récupérer les paramètres réseau du PC, ainsi que les paramètres d'activation des périphériques indispensables au mode I-Cluster. Ceci est réalisé à l'aide d'une pile logicielle basée sur une distribution Linux qui auto-détecte les périphériques indispensables tels que carte réseau, écran, clavier et disque dur. A ceux-ci s'ajoutent évidemment les composants logiciels indispensables au fonctionnement du matériel de base (mémoire, processeur, contrôleur d'interruptions et de bus). Une fois que la partition I-Cluster est disponible sur le PC, le bac à sable est opérationnel et le PC peut faire la transition entre les modes utilisateur et I-Cluster à l'aide de l'agent I-Cluster, en fonction de certains évènements système (sauvegarde d'écran par exemple). Les composants logiciels constituant la partition I-Cluster sont considérés comme en écriture seule : Ceci signifie que leur état ne peut évoluer que par l'installation d'une nouvelle partition par-dessus l'ancienne. Ceci est limitatif, mais nous permet de pouvoir garantir la sécurité de l'utilisateur de la machine en ne lui fournissant que des composants qualifiés, lui interdisant les manipulations d'administration qu'il serait tenté de faire. En particulier, le mode administrateur (root) de la partition I-Cluster est rendu inaccessible.

3.2.4. Conclusion sur le bac à sable

Nous proposons un système de bac à sable nouveau, effectuant un changement complet d'état de la machine ainsi qu'une analyse de sa disponibilité au cours du temps.

Notre architecture se base donc sur l'existence d'un bac à sable caché sur le disque dur de chaque PC de l'intranet. Chaque PC peut ainsi d'une part analyser l'utilisation de la machine par son possesseur, construire le profil correspondant puis s'en servir pour prédire les fenêtres de tir, périodes pendant lesquelles la machine ne sera pas utilisée par son possesseur.

Ces fenêtres de tir pourront être exploitées pour faire du calcul intensif communautaire, en basculant la machine de son mode normal en un mode de calcul, en utilisant une partition cachée sur son disque dur qui contient le logiciel nécessaire pour faire de cette machine un élément d'une grappe.

Les mécanismes qui ont été développés au cours de notre étude sont lourds, car ils nécessitent un redémarrage (transparent) de la machine, mais ont l'avantage de permettre une bonne isolation entre le mode utilisateur et le mode grappe. D'autre part nous disposons d'une réelle partition de disque pour y mettre le logiciel nécessaire au mode grappe, incluant le logiciel de base. De

cette manière, nous pourrons utiliser toute distribution logicielle prévue pour faire fonctionner une grappe et l'installer par-dessus notre système. Les administrateurs de grappes ne seront donc pas dépaysés, si ce n'est que les grappes seront virtuelles, avec une durée de vie limitée à la durée d'un travail.

Le retour d'un PC en mode normal se fait de manière très transparente pour l'utilisateur. Notre système utilise les fenêtres de tir prédites afin de rebasculer la machine en mode utilisateur 30 minutes avant le retour de celui-ci. L'exploitation des fonctionnalités d'hibernation de Windows permet de remettre la machine dans l'état précis où elle était avant de basculer en mode grappe. De cette manière une machine peut passer chaque nuit ou chaque week-end en mode grappe et effectuer des calculs de manière communautaire sans que son utilisateur n'en ait connaissance.

La sécurité offerte par notre bac à sable repose sur la sécurité de Linux. En effet, il faudra les droits d'administration *root* pour pouvoir prétendre à une attaque du système. Nous sommes conscients qu'il est difficile mais pas impossible pour un utilisateur malicieux de s'introduire dans le système du possesseur d'une machine. Cependant le fait de limiter notre étude à des machines d'un intranet limite grandement l'intérêt que peuvent avoir de telles attaques. Remarquons cependant que l'utilisation de versions sécurisées de Linux [EBC01] permettrait de venir à bout de ce problème au prix d'un système plus complexe et peut-être moins performant.

Nos extensions à la norme EFI ont été soumises au comité [Int01] pour inclusion éventuelle dans les futures versions du standard.

3.3. Le nuage I-Cluster : Un système de gestion distribuée des ressources

Les machines d'un intranet sont donc maintenant capables d'instrumenter leurs capacités matérielles, et donc indirectement la puissance de calcul dont elles sont capables. Elles peuvent aussi déterminer leur profil d'utilisation, c'est-à-dire les périodes auxquelles un utilisateur les utilise, les périodes auxquelles elles sont connectées au réseau, inactives ou encore débranchées. Elles peuvent donc en déduire les fenêtres de tir pendant lesquelles elles peuvent être exploitées pour des calculs distribués, typiquement la nuit, pendant les week-ends ou en cas d'absence régulière de leur utilisateur (qui travaille à temps partiel par exemple). Il est nécessaire de pouvoir mettre ces informations à disposition des utilisateurs désirant utiliser les machines inactives pour lancer des travaux.

Nous allons décrire ici une structure de base de donnée distribuée en mode pair-à-pair, qui nous permettra de maintenir une connaissance de l'état des machines appartenant au système et donc de prendre des décisions concernant leur allocation à des tâches en mode de grappe virtuelle.

3.3.1. Description

Le *nuage I-Cluster*, que nous appellerons par la suite nuage, est un annuaire de ressources distribuées à grande échelle étudié pour répondre aux besoins spécifiques du projet I-Cluster. En particulier, il permet le partage d'informations à grande échelle, en utilisant des mécanismes de bavardage pair-

à-pair (*gossiping*) dérivés des algorithmes distribués utilisés dans MOSIX [BGW93]. Il s'organise et s'adapte automatiquement à la disponibilité et à la volatilité des machines, à la connectivité réseau variable (cas des zones d'ombre dans les réseaux sans fil), à un grand nombre de machines participantes (de l'ordre de 10 000), tout cela sans utiliser de serveur central. Et le nuage supporte l'arrivée dynamique de nouveaux participants, tout en conservant une vitesse de convergence des informations rapide.

3.3.1.1. Limitations des implémentations actuelles

De nombreux efforts de recherche ainsi que des environnements d'exécution permettant d'exploiter les ressources de calcul disponibles sur un intranet d'entreprise sont déjà en place. Des architectures distribuées d'exécution telles que Legion [GFK+99], Condor [LLM88], LSF [Zho92], Globus [FKe96], NetSolve [CDo95], MOSIX [BGW93], permettent de gérer la puissance disponible sur une grappe et de distribuer des tâches de calcul sur des PC disponibles. Certaines de ces architectures permettent l'exécution sur un ensemble non homogène de machines tels que les machines d'un intranet. D'autres systèmes tels que SETI@home [And+99][And02] ou XtremWeb [FGN+01] permettent une telle distribution de tâches à une échelle mondiale, en combinant la puissance inutilisée de PC connectés à Internet. L'équipe HP Labs Grenoble a (avait !) pour objectif stratégique la recherche sur les communautés d'ustensiles Internet (*Internet appliance ecosystems*). Il était donc naturel au fil du projet I-Cluster de s'intéresser à la manière dont les annuaires de ressources sont implémentés dans les solutions de calcul distribué classiques. Des projets tels que Legion, Condor ou PBS [HTw96] disposent d'annuaires de ressources centralisés, Parfois de manière hiérarchique [FKe96]. Ces annuaires sont basés sur une base de données de ressources centralisée, qui permettra l'accès aux informations de manière simple. Toutefois, l'écriture dans de telles bases de données est très coûteuse. Des projets tels que Condor ou Globus sont actuellement basés sur un annuaire LDAP [HSm95], ce qui nuit au passage à l'échelle massif ainsi qu'au support de la volatilité et de la dynamicité des informations. Dans le cas d'une grappe de calcul les informations sont assez statiques : les machines sont dédiées au calcul et sont généralement disponibles à tout instant, ce qui réduit le besoin d'informations dynamiques et donc d'écritures dans la base de données de gestion des ressources distribuées. Pour ces raisons, les projets basés sur les technologies Condor et Globus sont limitées à des sites de quelques milliers de machines au maximum, et sont à la recherche d'alternatives à LDAP qui permettraient de supporter plus de dynamicité et un plus grand nombre de machines.

Lors de l'étude I-Cluster, les technologies pair-à-pair (*Peer-to-Peer*) [MKN+02] ont connu parallèlement un essor important, tant au niveau de la recherche qu'à celui des outils disponibles. Des projets comme Freenet [CSW+01] ou Gnutella [Gnu01] mettent en perspective les propriétés de passage à l'échelle et de dynamicité des algorithmes pair-à-pair. L'orientation de notre étude vers des algorithmes pair-à-pair s'est donc faite très naturellement, car nous pouvions rester dans un domaine d'étude de l'écosystème d'ustensiles Internet en bénéficiant des propriétés nécessaires au projet I-Cluster.

L'état actuel des systèmes de gestion distribuée des ressources reflète dans la majorité des cas une gestion centralisée des ressources héritée d'architectures limitées à un nombre de ressources de calcul de l'ordre de quelques dizaines ou centaines de nœuds. Dans le cas d'un intranet d'entreprise la gestion de l'ensemble des machines des employés peut atteindre des dizaines de milliers de machines, et dans ce cas le serveur gérant la base de données de ressources devient rapidement un goulot d'étranglement. En effet, nous avons vu que les informations nécessaires pour la prise en compte efficace de chaque machine nécessitent de connaître à chaque instant la disponibilité des machines -qui évolue en fonction du profil de leur utilisation et de l'activité en temps réel de leurs utilisateurs-, leur connectivité variable au réseau telle que les ordinateurs portables en mode sans fil qui peuvent se voir topologiquement connectés à plusieurs endroits différents du réseau tels que du bureau de l'utilisateur puis d'une salle de réunion. Un autre cas pathologique est la congestion ou la rupture de ligne du réseau reliant une partie des machines (*branch office*) à l'ensemble de l'intranet. Le graphe du réseau physique dans un tel cas devient partitionné et fait apparaître des îlots isolés de machines. Bien qu'il soit techniquement possible d'utiliser des machines d'une même partition pour instancier une tâche de calcul, la rupture du lien avec le serveur central constitue un cas de faute pour les systèmes classiques, alors que la gestion de ressources utilisée dans I-Cluster autorise le fonctionnement dans de telles conditions avec une dégradation limitée. Certains systèmes centralisés tels que Condor ou SETI@home peuvent passer à l'échelle et s'accommoder d'un grand nombre de machines au détriment d'une réactivité et d'une souplesse limitées.

Une autre propriété souhaitable d'un système de partage de ressources distribué est l'administration réduite. En particulier, le nuage I-Cluster s'adapte automatiquement aux changements d'état, apparition ou disparition des éléments du système sans nécessiter d'intervention humaine : chaque machine de l'intranet est supposée maintenue dans des conditions normales de fonctionnement. En particulier, la connectivité réseau et le fonctionnement matériel (redémarrages après les fautes matérielles) doivent être assurés par l'utilisateur pour son propre besoin. I-Cluster se base sur cet environnement minimal et ne demande donc pas d'administration particulière pour son opération.

3.3.1.2. Bavardage avec MOSIX

MOSIX [BWh89] offre une solution élégante aux problèmes de passage à l'échelle et de réactivité. Il consiste en une extension au système d'exploitation Unix permettant la migration de tâches entre les différentes machines d'une ferme de serveurs. La migration se fait automatiquement, lorsque la charge d'une machine devient trop importante, alors qu'une autre machine de la ferme est moins chargée. La migration prend alors en charge de manière transparente -mais peu efficace- la migration des sessions réseau et des fichiers ouverts par l'application, et effectue un cliché de la mémoire utilisée par l'application à migrer, puis transfère cette image mémoire sur le serveur cible. Les informations de charge sont mises à jour dynamiquement par MOSIX à l'aide d'un service bavard (*gossip service*).

Chaque serveur dispose de capteurs permettant d'instrumenter sa charge locale (usage de la mémoire et du swap). Ce capteur est activé régulièrement

toutes les 3 secondes sur chaque serveur et met à jour les informations de charge dans la base de données locale, associée à une estampille temporelle. La base de données locale dispose en outre des informations de charge de chacun des serveurs de la ferme. Le service bavard connaît la liste exhaustive de tous les serveurs de la ferme, et va se connecter à intervalles réguliers et rapides (de l'ordre de quelques dizaines de ms) sur un des autres serveurs de la ferme, choisi à chaque fois aléatoirement. Lors de cette connexion, l'information reçue par le serveur récepteur est intégrée dans sa base de données locale, et chacune des informations reçues est comparée à sa valeur précédente à l'aide des estampilles temporelles qui y sont associées. De cette manière, il va pouvoir mettre à jour les informations obsolètes et les remplacer par les valeurs plus fraîches envoyées par le serveur émetteur. Ce mécanisme permet la diffusion dynamique des informations de charge très rapidement. L'équipe MOSIX a montré [BGW93] que le temps de demi-diffusion moyenne, c'est-à-dire le temps moyen de diffusion d'une information depuis un serveur vers la moitié des machines de la ferme, était de l'ordre de $\log_2(N)$, N étant le nombre de serveurs de la ferme.

Ce résultat nous permet d'espérer des algorithmes de MOSIX les qualités que nous attendons pour le nuage I-Cluster, en particulier l'auto-administration du système, la répartition automatique de la charge de mise à jour des informations et le passage à l'échelle.

Cependant, concernant le passage à l'échelle, il est nécessaire pour chaque serveur de connaître exhaustivement la liste de tous les serveurs de la ferme et de maintenir le vecteur d'informations correspondant. Or dans notre cas nous voulons pouvoir passer à large échelle, c'est-à-dire à plusieurs milliers de machines. Dans ce cas, le coût de stockage du vecteur est de l'ordre de N sur chacun des serveurs, de même que les besoins en communication associés. Il nous faudra donc un espace de stockage total ainsi qu'un flot de communications de l'ordre de N^2. Ceci nous interdit un passage à l'échelle massif et restreint l'utilisation de MOSIX à des fermes de taille assez réduite (dizaines de machines).

D'autre part l'algorithme demande une connaissance exhaustive des machines de la ferme, et il n'est pas possible d'ajouter de nouveaux serveurs sans stopper le système.

Le nuage I-Cluster hérite de beaucoup de principes et d'algorithmes de MOSIX, et propose une version modifiée de l'algorithme de gestion distribuée des ressources qui permettra le passage à l'échelle massif, de l'ordre de plusieurs milliers de machines. Nous allons voir ci-dessous les détails de notre algorithme.

3.3.1.3. Architecture générale du nuage

L'objectif du nuage I-Cluster est donc de permettre la diffusion des informations de capacité, de disponibilité et de connectivité des nœuds de l'intranet.

Le nuage est construit à l'aide d'une communauté d'agents pair-à-pair. Un agent est disponible sur chaque machine participant au nuage, sous forme d'un programme Java s'exécutant en tâche de fond. Chaque agent peut se connecter à

un autre agent, ou encore jouer le rôle de serveur lorsqu'une connexion entrante est ouverte depuis un autre pair. L'agent pair-à-pair se compose des éléments suivants :

- une base de données d'objets,
- un récepteur capable de traiter les messages entrants et d'y répondre,
- un émetteur qui va contacter les pairs du système à intervalles réguliers afin de partager des informations avec eux.

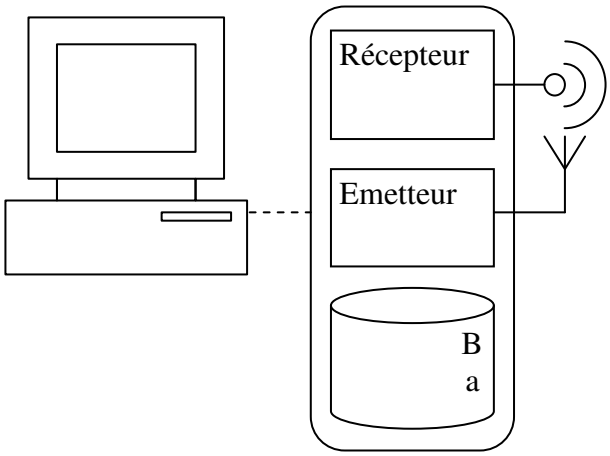

Figure 19 : Les différents composants nécessaires sur un pair

La figure ci-dessous reprend cette architecture. Nous y trouvons quelques machines dans un nuage, et des connexions émetteur-récepteur ont été représentées pour matérialiser les échanges d'informations entre pairs.

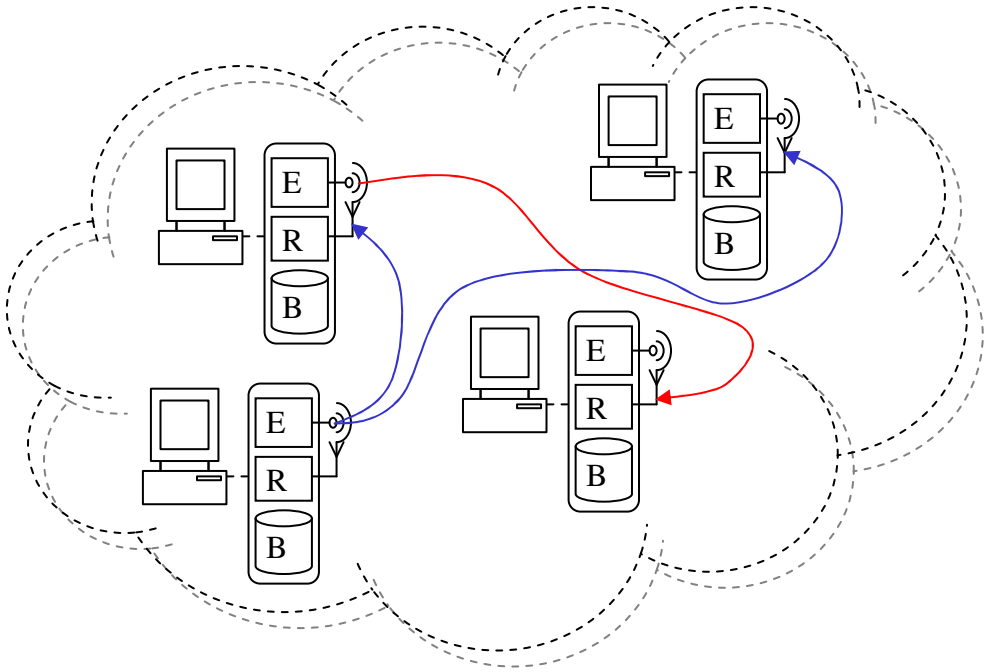

Figure 20 : Architecture pair-à-pair du nuage I-Cluster

3.3.2. Base de données d'objets

Les informations relatives à chaque machine sont stockées dans une base de données distribuée en mode pair-à-pair. Chacun des participants du nuage dispose de sa propre image de la base de données, ne contenant qu'une partie des informations disponibles dans le système global. En particulier, chaque participant ne va stocker que les informations relatives à un certain nombre d'autres participants, ce nombre dépendant des capacités de la machine. Par exemple, un serveur puissant pourra stocker les informations relatives à plusieurs milliers de participants; Un PC de bureau en stockera un millier et un PDA quelques centaines. En effet, le fait de stocker des informations sur un plus grand nombre de participants génère une charge de calcul locale ainsi qu'une charge réseau, et il nous faut donc limiter la taille de la base de données locale de chaque machine pour ne pas dégrader leur fonctionnement normal.

Les objets de la base de données seront dans la plupart des cas des cartes de ressources décrivant les capacités de calcul d'une machine donnée, ainsi que des informations d'adressage permettant l'identification et la communication avec d'autres participants, et des informations de disponibilité et de connectivité qui nous permettront de mettre au point les politiques de groupe ayant pour but de détecter les machines fautives. Une dernière information, mise à jour dans le cas où la machine est disponible, est une évaluation de la prochaine fenêtre de tir de la machine, plage horaire pendant laquelle la machine peut être exploitée par l'environnement I-Cluster sans gêne pour son utilisateur.

Un identifiant représentant l'état courant (disponible, en cours d'utilisation, en cours de calcul, éteint, déconnecté) de cette machine est aussi attaché à ses informations. Une particularité de cet identifiant est qu'il est possible à d'autres machines du nuage de le modifier afin de le passer dans l'état déconnecté, ce qui se produit lorsque plusieurs machines du nuage n'arrivent plus à se connecter a cette machine, et qui décident alors par consensus de déclarer la machine déconnectée.

Les informations stockées dans la base de données de chacune des machines sont associées à une estampille temporelle représentant leur instant de dernière mise à jour, ainsi qu'un identifiant unique (*UUID : Universally Unique IDentifier*) à la machine en cause. Notons que les cartes de ressources de la base de données sont à écrivain unique, à savoir la machine qui les a émises. Seule cette machine pourra donc mettre à jour sa carte de ressources, qui ne sera donc changée que lors de reconfigurations ou mises à jour matérielles du système.

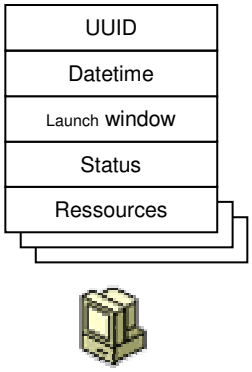

Figure 21 : Objet de la base de données locale avec sa carte de ressources, son identifiant, estampille temporelle, fenêtre de tir, des informations de connectivité et de disponibilité.

3.3.2.1. Informations partagées

Les informations partagées par le nuage sont dynamiques et sont constituées des cartes de ressources individuelles de chaque machine, comme nous l'avons vu plus haut. Chaque machine va donc instrumenter ses propres capacités matérielles ainsi que la prédiction des fenêtres de tir pendant lesquelles elle pourra être exploitée pour des calculs distribués. Ces informations seront mises à disposition de la communauté du nuage en mode lecture seulement.

3.3.2.2. Informations dynamiques

Les informations de disponibilité et de connectivité sont aussi publiées, et elles peuvent être modifiées par les pairs et sont donc en mode lecture/écriture pour toutes les machines. Nous verrons ci-dessous qu'il nous sera alors nécessaire de gérer la cohérence de ces informations, à l'aide d'un système de versions et de réplication optimiste, qui lui-même demandera des protocoles de réconciliation et de résolution de conflits.

Il nous faudra aussi disposer d'un protocole de consensus permettant la détection des machines fautives ou déconnectées. En effet, nous verrons que lors d'une faute de communication entre deux machines, il est difficile de distinguer si la première machine est fautive, si la seconde est fautive ou si le lien de communication est fautif. Il nous faudra un algorithme de groupe pour pouvoir résoudre ce problème.

Notre système de consensus permettra de gérer de manière automatique les mouvements des machines sur le réseau, les déconnexions et les partitionnements du réseau, qui sont des caractéristiques inhérentes au support des réseaux sans fil tels Bluetooth ou Ethernet 802.11.

Une caractéristique importante du nuage est la convergence des informations : Si les machines sont connectées et qu'il n'y a plus aucune mise à jour du nuage (pas de modification des informations disponibles), toute information disponible dans le nuage convergera vers le même état après un temps borné, de l'ordre de *log(N)*. Dans la pratique cette convergence

s'effectuera après quelques dizaines de secondes, ce qui est suffisant pour nos besoins.

3.3.2.3. Caractérisation de la base de données

Chaque objet placé dans la base de données est donc caractérisé par un quadruplet *(t, uuid, k, v)* ou t est une estampille temporelle correspondant au moment de la dernière écriture de l'objet, id est un identifiant unique de la machine ayant fait la dernière écriture, k est la clef, qui permet d'indexer la valeur v.

La clef k est l'une des suivantes :

k = ResourceCard. Cette clef permet de stocker et de récupérer la carte de ressources associées à une machine donnée, identifiée par son uuid. A noter que la clef ResourceCard est en lecture seulement, sauf pour le possesseur de la carte de ressources en question.

k = State. Avec cette valeur de clef, il sera possible de stocker et de récupérer les informations relatives à l'état de la machine identifiée par son uuid.

La valeur v peut être d'un type quelconque. Dans la pratique nous stockons les informations de cartes de ressources sous la forme des Condor ClassAds [Pli96], qui sont constitués d'une simple chaîne de caractères comme suit :

```
Attribute = "Value" Attribute = "Value" …
```

Par exemple on pourra trouver une carte de ressources exprimée par une chaîne de caractères de la forme suivante :

```
Name = "icluster136.imag.fr" \
Arch = "INTEL" \
Cpus = "1" \
Memory = "256" \
Mips = "733" \
MyType = "Machine" \
OpSys = "I-Cluster" \
StartdIpAddr = "129.88.96.136:2387"
```

Ces attributs sont des attributs standard de Condor. Nous avons cependant quelques additions et modifications par rapport au *MatchMaker* (le courtier effectuant le placement des applications sur les machines) de Condor :

Un nouveau type OpSys a été défini : "I-Cluster".

Un nouvel attribut NetRange a été défini. Cet attribut permet d'identifier un groupe de machines (range) comme ayant une bonne proximité topologique et une basse latence de communication réseau. Dans le cas ci-dessus, on trouvera par exemple l'attribut NetRange = "129.88.96.255/24" qui identifie le sous-réseau Ethernet utilisé par cette machine. A l'aide de cet attribut, il sera possible, lors d'une requête de calcul en grain fin sur N machines, de trouver des machines qui partagent le même commutateur réseau et qui pourront constituer une grappe virtuelle efficace.

L'attribut NetType a aussi été défini afin d'instrumenter le type de connectivité disponible. Par exemple, on aura NetType = "ETHERNET100" dans notre cas ci-dessus.

L'attribut Uuid a été défini afin de stocker l'identifiant unique de chaque pair du nuage. Par exemple on va trouver Uuid = "2583FEB34D3B40E9A16A8319C9AB309F05" comme identifiant pour notre exemple ci-dessus.

Certains attributs ne sont pas présents dans le ClassAd exposé par la carte de ressources (k=ResourceCard), mais dans l'objet d'état (k=State) de la machine considérée. Ainsi dans l'objet d'état nous retrouverons l'attribut Uuid, mais aussi les attributs standard de Condor suivants :

State, qui prendra l'une des valeurs standard "*Owner*", "*Unclaimed*", "*Matched*", ou "*Claimed*". Nous y avons ajouté les valeurs "*ShutDown*" et "*Suspect*", qui nous permettent de gérer les états de déconnexion et de faute de la machine.

LaunchWindowFinish, qui donnera la date et l'heure prévue de la prochaine fenêtre de tir de la machine. On aura par exemple LaunchWindowFinish = "14 Jan 2002 08:45:00 2002 GMT". A noter que nous n'avons pas prévu de LaunchWindowStart dans la version actuelle, pour la simple raison que nous n'avons pas de composant permettant de poser une requête pour un traitement dans le futur (pas de Queueing System) et donc les requêtes soumises sont traitées immédiatement. Or la disponibilité des machines peut être testée de manière immédiate à l'aide de l'attribut State, qui sera égal à "Unclaimed" si la machine est entrée automatiquement dans une fenêtre de tir.

Enfin, l'attribut SubmitUuid sera utilisé dans le cas où un pair modifie une information dont il n'est pas le possesseur, et contiendra dans ce cas le Uuid du « modifiant ». Ceci nous sera utile pour notre gestion de consensus pour déterminer si une machine est fautive (déconnectée) ou non.

3.3.2.4. Opérations sur la base de données

Nous avons pour méthodes d'accès à la base de données Store et Load.

Store prend en arguments (t, uuid, k, v) et va mettre à jour les informations de la carte de ressources (Cas où k = ResourceCard) ou de l'état (Cas où k = State) de la machine identifiée par uuid, en leur donnant la nouvelle valeur.

L'estampille temporelle t peut être nulle. Dans ce cas, une estampille correspondant à l'instant présent est générée automatiquement lors de l'appel à Store. Ce n'est pas une horloge système, qui poserait des problèmes de cohérence d'horloges dues aux dérives et erreurs des horloges matérielles des machines du nuage. C'est une horloge logique distribuée basée sur le modèle de Lamport [Lam78] et qui nous garantit que les estampilles temporelles de deux évènements asynchrones respecteront toujours une éventuelle causalité entre ces évènements (relation happens-before : si un évènement *a* intervient causalement avant un évènement *b*, on garantit que l'estampille temporelle de *a* reflète une heure logique inférieure à celle de *b*).

Store retourne une erreur lorsqu'on demande la modification d'une carte de ressources dont le possesseur n'est pas le processus local. Ceci garantit que les cartes de ressources sont en mode simple écrivain.

Lors de la mise à jour d'une valeur, c'est-à-dire lors de l'invocation de Store sur un objet déjà existant, la méthode va vérifier si la nouvelle valeur est identique à la valeur à mettre à jour. Dans ce cas, aucune action ne sera entreprise, et l'estampille temporelle t restera à la valeur qu'elle avait avant l'écriture.

Load prend en argument (uuid, k) et retourne la valeur correspondante v, ou reporte une exception lorsque la valeur demandée n'est pas disponible dans la base de données.

3.3.3. Diffusion d'informations et mise en cohérence de la base de données

La base de données locale est partagée à l'aide d'un agent local réalisant le service bavard (*gossip service*). Ce mécanisme est proche de celui utilisé par MOSIX et vu précédemment. A intervalles réguliers -typiquement toutes les secondes-, l'agent va sélectionner aléatoirement un autre pair de sa base de données locale. Il va se connecter à ce pair, et ils vont échanger le contenu de leurs bases de données respectives. Les éventuelles informations obsolètes de chacun seront mises à jour si l'autre pair dispose des mêmes informations (même uuid) avec une estampille temporelle plus récente. La communication entre ces deux pairs est alors rompue, dans l'attente d'une prochaine itération du processus. Ce protocole permet la diffusion épidémique des informations de proche en proche parmi les pairs du nuage.

3.3.3.1. Protocole de communication

Définissons maintenant le protocole de bavardage utilisé ainsi que sa sémantique : Le protocole utilisé est basé sur des sessions TCP. Chaque participant peut se connecter en point à point à un autre pair du nuage. La session ouverte pourra permettre d'une part de maintenir la base de données distribuée du nuage, et d'autre part l'allocation des machines à des travaux ainsi que l'invocation des travaux sur les machines allouées.

Les types de messages disponibles pour notre protocole sont les suivants :

CONNECT : Demande d'ouverture de connexion pour échange de données du nuage.

RESPONSE : Ce message sert de réponse positive à une demande d'ouverture de connexion. Le message RESPONSE contient des informations de mise à jour permettant d'exposer le contenu de la base de données locale de l'appelé vers à l'appelant. D'autre part, le message RESPONSE peut être réémis de l'appelant vers l'appelé pour corriger ou agrémenter les informations obtenues dans la première phase. La figure ci-dessous explicite ce flot.

CLAIM : Ce message permet à l'appelant de réserver l'appelé pour une allocation éventuelle.

ACK : Ce message est une réponse positive à une réservation par un CLAIM.

NAK : Ce message est une réponse négative à une réservation par un CLAIM, émis si la machine est déjà réservée ou allouée. NAK peut aussi être émis en réponse à un CONNECT, ce qui permet de refuser proprement une demande de connexion, par exemple dans le cas où la machine appelée est trop chargée pour gérer une nouvelle connexion.

RELEASE : Ce message permet d'annuler la réservation faite précédemment par un CLAIM.

INVOKE : Ce message est envoyé sur une machine déjà réservée par un CLAIM, et qui a confirmé la réservation par un ACK.

DISCONNECT : Ce message permet de clore la session.

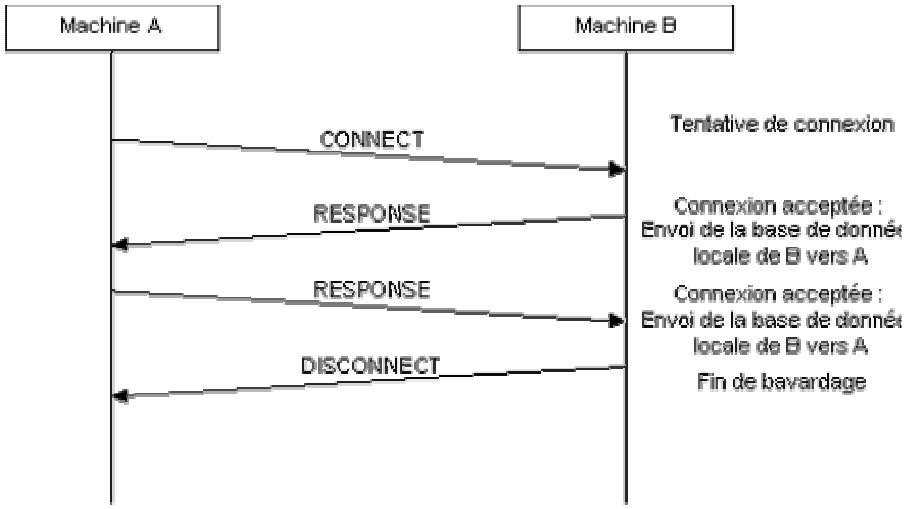

Figure 22 : Flot de messages envoyés pendant une session de bavardage entre deux machines

Lors de l'invocation d'une requête d'instanciation de service, N machines vont être réservées par la machine initiatrice. Dans la figure ci-dessous, nous pouvons voir le flot de messages, montrant les deux phases du protocole : Lors d'une première phase les machines sont réclamées (*claimed*) par la machine initiatrice. Si les réclamations sont acceptées par les N machines, la machine initiatrice va alors invoquer une commande d'exécution sur chacune des machines.

Notons que les machines allouées sont responsables de leur propre déallocation. Ceci est fait lors de la détection de la fin de la fenêtre de tir, ou bien à la fin du travail à exécuter. Dans ces cas la machine repasse en état disponible et les éventuels travaux non terminés sont abandonnés.

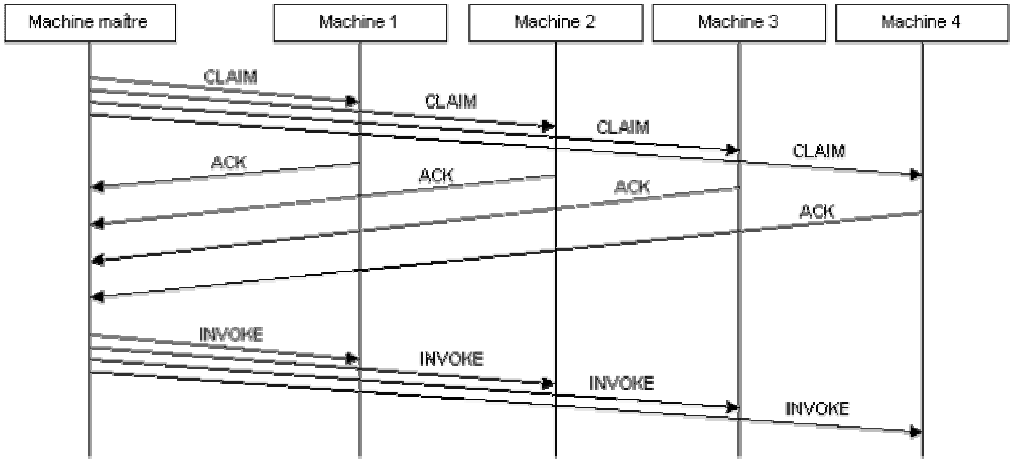

Figure 23 : Flot de messages lors de l'allocation de quatre machines par une machine initiatrice.

Lorsque l'une des machines ne répond pas positivement (*ACK*) à une réclamation, la deuxième phase du protocole, l'invocation, n'a pas lieu, et un message RELEASE permet de remettre les machines en état d'attente de travail, en abandonnant la réservation qui a été faite.

3.3.3.2. Aspects techniques de la mise en œuvre

Modes *push*, *pull* et anti-entropie

La communication entre deux pairs du nuage a pour but la diffusion des informations d'état et les cartes de ressources des machines disponibles. Cette diffusion peut prendre place par différents mécanismes :

Dans le mode ***push***, les informations sont envoyées par l'appelant vers l'appelé. L'appelé va alors faire l'union de ses propres informations et des nouvelles informations envoyées par l'appelant, en mettant à jour les informations obsolètes par une simple vérification des estampilles temporelles de chaque objet de la base de données : si deux versions différentes d'un même objet de la base de données sont disponibles (l'un provenant de la base locale, l'autre ayant été envoyé par l'appelant), seule la version possédant l'estampille temporelle la plus grande est préservée. Ce mode de transfert offre une bonne facilité de mise au point et un temps d'exécution des requêtes assez court.

Le mode ***pull*** est relativement similaire au push, si ce n'est que les informations sont envoyées à l'appelant par l'appelé, qui mettra à jour sa base de données locales en faisant l'union de ses propres informations et des nouvelles informations reçues. Le mode pull, cependant, offre de sérieuses limitations : Il n'est pas possible pour une machine d'accélérer la propagation d'une information qu'il a publiée, il doit se limiter à attendre que les autres machines ne provoquent cette propagation. Le mode push peut permettre d'augmenter le taux de diffusion d'une information importante en faisant varier le taux de connexion du pair émetteur. D'autre part, il est plus difficile pour un pair de joindre un groupe fonctionnant en mode pull. En effet, il faudra que l'un des membres du groupe connaisse le nouveau pair par un moyen externe afin de pouvoir le faire connaître à ses voisins et éventuellement afin de pouvoir s'y connecter. Dans le mode push il suffit à un pair de se connecter à un quelconque

pair du groupe, puis de lui envoyer les informations relatives au nouveau pair, pour que celui-ci soit ajouté à la base de données locale.

Dans le mode **anti-entropie**, lors de la connexion d'un appelant vers un appelé est faite, le contenu des deux bases de données locales est mis en commun, les éventuelles informations obsolètes de l'un sont rafraîchies par les informations plus récentes de l'autre. A la fin de la mise en relation, le contenu des deux bases de données locales est identique entre l'appelant et l'appelé. Le mode anti-entropie est très efficace, mais nécessite une certaine complexité algorithmique. En effet, les opérations de mise en relation des bases de données de deux pairs se font de manière atomique au sens des écritures dans les bases de données. Or il est important pour un pair, nous le verrons plus loin, de pouvoir accepter plusieurs requêtes de mise à jour concurrentes. Il nous faut alors ajouter une grande complexité de gestion de la cohérence des écritures concurrentes dans la base de données, ou encore gérer des fenêtres d'accès en écriture exclusives, ce qui mettrait à mal la performance du système.

De nombreux systèmes bavards, tels MOSIX, fonctionnent en mode symétrique et sont basés sur un mode push. Dans le cas de l'agent I-Cluster, notre protocole fonctionne en mode anti-entropie : L'appelant envoie un message CONNECT, auquel l'appelé répond par un RESPONSE an y attachant les informations de sa base de données locale. Il est alors possible pour l'appelant de répondre à ce RESPONSE par un autre RESPONSE, auquel il attache les informations de sa propre base de données locale. Toutefois, un essai de mise au point de l'agent dans ce mode anti-entropie a demandé de gros efforts au niveau des synchronisations lors de communications concurrentes, et nous avons donc réduit le protocole à un mode push seulement : L'appelé n'attache donc aucune information à son (premier) message RESPONSE, et seul l'appelant va envoyer des informations.

Cas particulier des informations d'état de la machine locale

Lors de la mise en relation de deux machines et de la comparaison des informations de leurs bases de données locales respectives, un cas particulier se présente lorsqu'une machine reçoit une information d'état se rapportant à elle-même. Une machine dispose toujours de la priorité en écriture sur ses propres informations.

Par exemple : Une machine A est déconnectée. Une autre machine C se connecte à A à l'instant T_1, la détecte comme telle et marque donc l'objet d'état de A comme déconnecté dans sa propre base de données locale. A l'instant T_2, C envoie cet objet à B lors d'une communication normale. A l'instant T_3, la machine A se reconnecte. A l'instant T_4, B se connecte à A et lui envoie donc l'objet d'état de A montrant celle-ci comme déconnectée à l'instant T_1. Lors de la réception, la machine A détecte que son propre objet d'état et celui reçu n'ont pas la même estampille temporelle, et va donc générer une nouvelle version de cet objet d'état, qui aura donc la date T_4. Ce nouvel objet va par la suite être propagé vers les autres machines, mettant à jour celles-ci avec les informations correctes, à savoir que l'état de A est connecté.

Afin de toujours donner la priorité au propriétaire d'une information lors des échanges de données, dans un tel cas nous avons donc un traitement d'exception de la forme suivante :

```
Compare(ObjetLocal, ObjetReçu)
  {
  if(ObjetLocal.uuid == uuidLocal)
    {
      if(ObjetLocal.date < ObjetReçu.date)  // Objet reçu plus récent
        MiseAJour(ObjetLocal)               // Créer une nouvelle version
                                            //   qui écrasera les précédentes
    }
  }
```

Simulation des vitesses de diffusion

Les mécanismes du nuage I-Cluster décrits jusqu'ici suivent de près ceux utilisés par MOSIX, excepté le fait que nous pouvons supporter des écrivains multiples sur certains types d'objets de notre base de données, alors que MOSIX ne gère que des objets à écrivain unique.

Afin de motiver le choix pour notre architecture d'une variante de MOSIX, nous avons écrit un simulateur permettant l'évaluation de la vitesse de diffusion de messages. Nous considérons un cas de simulation avec N machines. Les N machines sont pleinement fonctionnelles, et supportent un nombre illimité de connexions simultanées. L'étude se fait par découpage du temps en étapes synchrones. A l'étape 0, une des machines va mettre une information à jour. Aucune autre information ne changera pendant la durée de la simulation. Pendant chacune des étapes de temps, chacune des N machines va se connecter à une autre machine, choisie aléatoirement parmi les N-1 restantes, et diffuser l'information si elle la possède déjà.

Nous étudions alors, en fonction du nombre N de machines, le nombre de machines ayant reçu l'information en fonction du nombre d'étapes de temps écoulées.

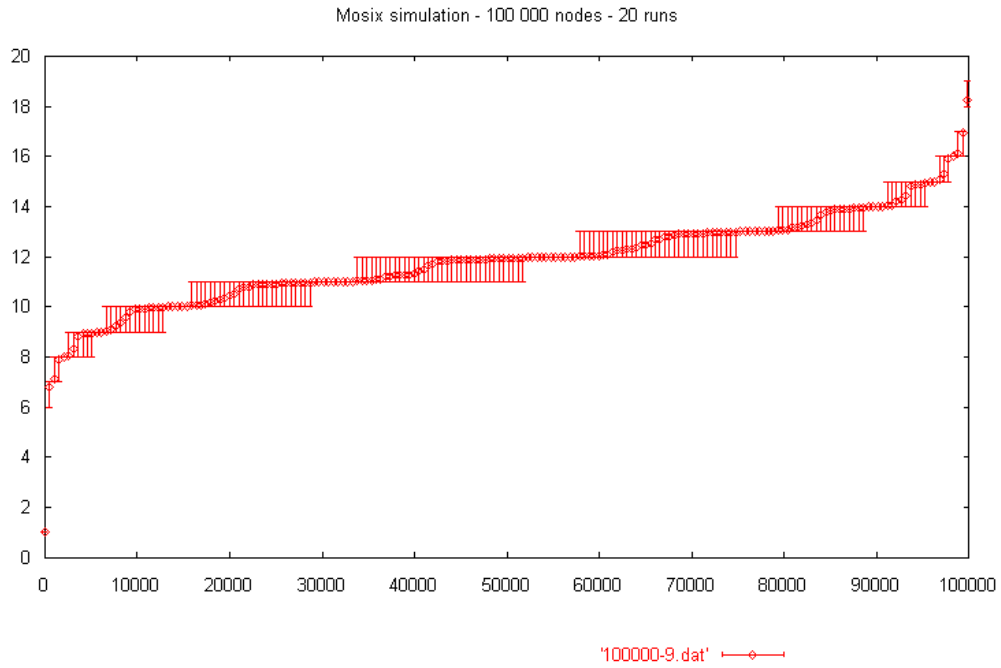

Figure 24 : Simulation de la vitesse de convergence avec MOSIX

La figure ci-dessous nous montre les résultats obtenus avec N=100 000 machines. La simulation a été lancée 20 fois afin de lisser les résultats obtenus.

Chaque point de la courbe est une moyenne des nombres d'étapes nécessaires à la diffusion, et les valeurs maximum et minimum ont été représentées.

On voit que la diffusion de l'information est très rapide, avec 19 étapes dans le plus mauvais cas. La diffusion sur la moitié des machines se fait en 12 étapes.

Cette étude nous montre que, dans un cas pleinement connecté, et avec un algorithme fonctionnant sur des plages de temps de l'ordre de 1 seconde, ce qui est très acceptable en termes de charge pour chaque machine, une information est diffusée vers la moitié du parc de machines en 12 secondes. Cette vitesse de réaction du système est très acceptable dans notre cas, où les changements d'états seront sporadiques et peu fréquents.

Mécanisme d'oubli et effet *small world*

La mise en commun complète entre les pairs de leurs informations, telle que nous l'avons décrite ci-dessus, demande pour chaque machine de stocker les informations relatives à l'ensemble des machines du nuage. Ceci pose un problème de passage à l'échelle, car l'espace nécessaire croit sur chaque machine en O(n) et donc l'espace total consommé sur l'ensemble des machines croit en $O(n^2)$.

D’autre part, la communication nécessaire sur l’ensemble du réseau d’interconnexion des machines du nuage est importante : Chaque pair générant une connexion vers un autre pair du réseau à des intervalles T_{gossip}, et la consommation moyenne de l’ensemble des messages d’une connexion donnée étant de S octets, chaque pair génère donc une consommation de S/T_{gossip} octets par seconde sur le réseau, soit un besoin total de $N.S/T_{gossip}$ octets par seconde pour l’ensemble des machines de la communauté.

Ces deux facteurs limitent clairement le passage à l’échelle de ce mécanisme, où le décuplement du nombre de machines multiplie par 100 la charge sur le réseau d’où une explosion combinatoire suivie d’un écroulement du système.

Afin d'éviter cette explosion, nous avons utilisé un mécanisme d'oubli permettant de limiter le nombre de références stockées par chacun des pairs du réseau. Ce mécanisme doit cependant permettre de garder l'effet Small World [Wat99][Kle00]. L'effet Small World demande deux caractéristiques essentielles : un chemin moyen entre pairs (*average path length*) le plus court possible, et une agrégation (*clustering*) maximale dans le graphe des nœuds du réseau.

Nous avons donc choisi de limiter la base de données locale de chacun des pairs du nuage en nombre d’objets stockables. La limite pratique BN_{max} est variable en fonction des capacités de chaque machine, de l’ordre de quelques centaines d’objets au maximum sur un PC standard, jusqu’à un millier sur un serveur puissant et disponible. BN_{max} est fixée sur un pair donné lors de l’installation de notre agent, puis ne change plus lors du cycle de vie de l’agent.

Métrique composite d'évaluation

Lors de la mise en commun des bases de données de deux pairs, il est possible que de nouveaux objets soient offerts par le pair distant. En ce cas, il faut ajouter ces objets à la base de données locale. Dans le cas où la taille de la base est déjà égale à BN_{max}, nous n’avons pas l’espace disponible pour à la fois conserver les objets déjà disponibles et en plus y ajouter les nouveaux objets.

Il nous faut donc choisir quels objets de la base de données vont être préservés. Une métrique composite va permettre d'évaluer l'importance locale des objets de la base de données. Cette métrique est composée en fonction de paramètres tels que la puissance, la disponibilité, la proximité de chaque objet de la base de données locale. Une description détaillée de cette métrique est disponible au paragraphe suivant. Les pairs les moins importants (possédant la métrique la plus mauvaise) sont éliminés de la base de données locale afin de ne pas dépasser son seuil de disponibilité et donc sa taille en nombre de pairs référencés.

La métrique est donc calculée en fonction des paramètres suivants :

- Puissance (nombre de processeurs et fréquence),
- Intérêt des fenêtres de tir disponibles,

- Vitesse de communication,
- Proximité sur le réseau (la machine considérée est sur le même sous-réseau que le pair local),
- Regroupement de la machine par rapport à d'autres pairs de la base de données locale. S'il existe un certain nombre de machines partageant le même sous-réseau, il sera intéressant de les conserver dans la base de données locale afin de pouvoir exécuter des travaux en grain fin, par allocation de grappes virtuelles composées de machines topologiquement proches.

D'autres paramètres pourraient intervenir dans notre métrique, tels que :

- Proximité réseau entre la machine considérée et les sources de données nécessaires aux calculs, afin de limiter les charges de transferts de grosses données,
- Historique de l'utilisation d'une machine, par exemple le succès des travaux lancés et leur vitesse d'exécution,
- Ancienneté de la machine sur le système (*uptime* I-Cluster). Ceci permettrait d'éviter les mouvements importants dans le système lors de la connexion d'une nouvelle machine très puissante par exemple.

A l'aide de cette métrique, chaque pair peut évaluer chacun des membres de sa base de données et ne garder que les éléments les plus importants pour lui. De cette manière, il n'est pas nécessaire de conserver localement une liste importante de pairs pour que le système soit efficace. Dans la pratique on peut se limiter à quelques centaines d'entrées dans la base de données locale.

La taille de la base de données est spécifique à chaque machine, par exemple un PC de bureau pourra avoir une base de 1000 entrées, et un PDA 200 entrées.

3.3.4. Contrôle de la taille des bases de données locales

Nous avons donc vu comment nous limitions la taille maximum de la base de données locale de chaque pair, par l'utilisation d'une métrique de qualité des pairs et oubli des pairs de plus mauvaise qualité.

Cette politique égoïste pose un risque de morcellement de la communauté et de saturation des machines de plus forte métrique.

Illustrons cet effet à l'aide de l'exemple suivant : Si l'on considère que chaque machine ait un BN_{max} identique, et supposons l'arrivée d'un nouveau pair, de métrique plus forte que toutes les autres machines. Cette machine sera toujours préservée lors d'un bavardage. Lorsque la carte de ressources de cette machine se sera diffusée dans le réseau, toutes les machines auront alors réservé une entrée de leur base de données locale pour ce nouveau pair.

Si l'on répète ce raisonnement, on montre par récurrence qu'après l'arrivée de BN_{max} nouveaux pairs à forte métrique, les bases de données locales de chacune des machines de la communauté ne contiendront plus que les nouveaux

pairs, les pairs ayant les métriques les moins fortes ayant disparu de la base de données globale.

Pour éviter cet effet pervers appelé « *riches get richer effect* » [Wat99][Axt00], une faible proportion des pairs à mauvaise métrique, donc jugés comme moins intéressants, vont être préservés dans la base de données locale et ne seront jamais oubliés lors des phases d'échange entre pairs. Ceci permet de conserver de bonnes propriétés Small World en préservant des chemins d'accès direct dans la communauté de pairs.

3.3.5. Gestion des déconnexions

Les machines de la communauté ont un cycle de vie propre, et fortement dépendant de leur utilisateur. En effet, de nombreux utilisateurs éteignent leur machine la nuit et le week-end, lorsqu'ils n'en ont pas besoin. Par ailleurs, les problèmes réseau sont fréquents, et encore augmentés lors de l'utilisation de technologies sans fil, qui posent des problèmes de visibilité inter-machines, de zones d'ombre etc. Ces conditions courantes d'utilisation des machines augmentent leur volatilité, et il nous est donc nécessaire de supporter efficacement un tel environnement dans le nuage I-Cluster.

Nous nous trouvons donc dans un environnement disposant d'un grand nombre de pairs, donc avec un grand nombre de processus indépendants. Pour agrémenter encore cette situation, nous avons vu que le taux de connexion et déconnexion était élevé, nécessitant une gestion de cette volatilité afin d'offrir un service d'annuaire de ressources qui reste efficace dans ce cas.

3.3.5.1. Catégories de fautes supportées

Les fautes que nous désirons gérer sont des fautes de type crash non définitif d'une machine (extinction par exemple) ou rupture de communication réseau.

Plusieurs scénarios peuvent être établis.

Dans le cas de l'extinction d'une machine (brutale ou non), le nuage devra pouvoir fiablement établir que la machine en cause est dans l'état déconnecté.

Le cas de la déconnexion d'une machine est identique au précédent.

Lors du rallumage ou de la reconnexion d'une machine, le nuage doit pouvoir fiablement établir que la machine est retournée en état connecté.

Dans le cas de la connectivité partielle d'une machine : seul un sous-ensemble des machines du nuage voient la machine en cause. Un tel cas peut se produire par exemple dans un réseau sans fil sans routeur d'infrastructure. Dans un tel cas une machine A peut être connectée à une machine B, B est connectée à une machine C mais A et C ne sont pas connectables (cf. figure ci-dessous). Dans ce cas il est difficile d'établir si une telle machine doit être considérée comme connectée ou non. Nous avons simplifié la contrainte de la manière suivante : Si deux tentatives de connexion successives (au sens de l'horloge virtuelle globale au nuage) à une machine M_0 par deux machines différentes M_i et M_j, $(i \neq j)$, alors la machine considérée sera considérée comme déconnectée.

Si une quelconque machine parvient à se connecter à M_0, celle-ci sera alors considérée comme connectée.

Dans le cas d'un partitionnement du réseau en plusieurs sous-réseaux, chaque machine d'un des sous-réseaux devra voir toutes les machines de ce sous-réseau comme connectées, et toutes les autres comme déconnectées. Ceci est le cas du *branch office*, où un bâtiment est temporairement coupé du réseau, par exemple lors d'une panne de son routeur, mais qui continue à fonctionner de manière autonome.

Lors du recouvrement d'une partition du réseau, c'est-à-dire si plusieurs partitions du réseau se rejoignent, lors de la remise en marche du routeur de l'exemple ci-dessus, toutes les machines doivent pouvoir fiablement détecter que toutes les autres machines sont connectées.

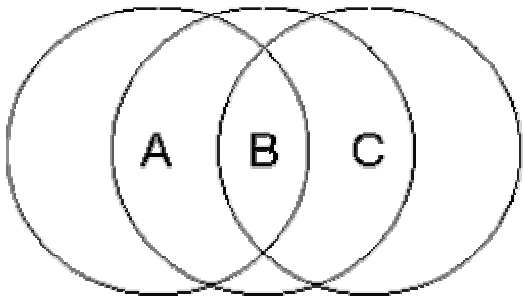

Figure 25 : Cas particulier des limites de visibilité dans les réseaux sans fil : A voit B, B voit A et C mais A ne voit pas C

La figure ci-dessus illustre un cas particulier des réseaux sans fil. Il est possible pour une machine de ne pas en « voir » une autre, qui est pourtant disponible et visible. Dans ce cas (A voit B qui voit C, mais A ne voit pas C), si A tente de communiquer avec C, ce sera sans succès, et A va donc marquer C comme *suspect* dans sa base de données. Puis B communique avec A, et note donc que C a été marqué comme suspect. Puis C se connecte à B. Dans ce cas il va le retirer de la liste de suspects (logique, puisqu'il arrive à communiquer).

Notre mécanisme de suspicion assure donc que les machines non directement joignables en mode sans fil ne seront pas évincées de la base de données globale.

3.3.5.2. Algorithme distribué de détection de fautes et recouvrement

Selon la classification de Christian et al [CAS86], les fautes que nous étudions sont de type crash, c'est-à-dire que les PC cessent de répondre.

Ces crash offrent parfois un recouvrement (*recovery*), par exemple lorsqu'un PC se reconnecte au réseau après une période de déconnexion.

Nous n'avons pas de faute de type omission. En effet, le protocole de bavardage est considéré comme atomique, c'est-à-dire que lors d'une procédure de bavardage entre deux pairs les opérations sont effectuées dans leur ensemble ou bien sont globalement annulées. Nous ne considérons pas non plus les fautes de timing ou les fautes byzantines.

Dans nos conditions, c'est-à-dire avec un mécanisme asynchrone, Fischer, Lynch et Paterson ont démontré [FLP85] qu'il était impossible d'obtenir un consensus en cas de panne d'un des membres de la communauté. Il nous faut donc examiner les différentes possibilités pour répondre aux cas de machines déconnectées et obtenir une réponse du système permettant la tolérance de ces fautes.

Chandra et Toueg [CTo96] proposent un modèle de détecteurs de fautes permettant de résoudre un problème d'accord insoluble tel que le consensus : Un oracle appelé détecteur de défaillances renseigne les processus sur les autres processus supposés fautifs. L'intérêt d'un telle démarche est qu'elle encapsule la synchronie nécessaire à la conception de protocoles d'accord dans des propriétés abstraites et ne faisant donc pas directement appel aux conditions temporelles. De ce fait, les protocoles obtenus ne font aucune référence au temps physique. Cette démarche a été la première fois formalisée par Chandra et Toueg. Contrairement à quasiment tous les travaux existants, qui implémentent les détecteurs de défaillances en utilisant le temps physique (temporisateurs et délais de garde), des implémentations de détecteurs de défaillances ne reposant que sur un mécanisme simple d'appel/réponse et sur le nombre de défaillances maximal dans le système ont pu être construits. Une implémentation correcte suppose que l'ordre de réception des messages respecte une certaine propriété qui varie selon la classe de détecteurs visée.

D'autres travaux tels que les algorithmes auto-stabilisants [Sch93] offrent une réponse à l'impossibilité du consensus de Fischer, Lynch et Paterson. Dans cette approche de la tolérance aux pannes, on considère que les pannes peuvent mettre les processus et les canaux dans n'importe quel état, et l'on souhaite que, lorsque les pannes cessent, l'algorithme rejoigne un état spécifié après une certaine période.

Un algorithme efficace dans le cas de grands nombres de machines dans la communauté a été proposé par Van Renesse et al. [VMH98]. Cet algorithme fonctionnant par bavardage va permettre une détection rapide des machines fautives. Cependant, leur algorithme ne supporte pas le cas de la déconnexion d'une machine, qui est dans ce cas détectée comme fautive et qui ne peut pas revenir dans un état stable. Dans le cas du nuage I-Cluster, nous voulons être capables de déterminer, lors d'une faute de connexion entre deux machines, si la faute est sur l'une des deux machines (qui est débranchée ou déconnectée du réseau) ou si la connexion est fautive (cas du *branch office*, où une passerelle en panne provoque un partitionnement du réseau).

La politique de résolution de fautes que nous avons choisie est donc une extension du protocole de Van Renesse qui se base sur le principe suivant : Si la communication vers une machine M_1 ne peut être effectuée depuis une seconde machine M_2, celle-ci va inscrire la machine en cause M_1 en état *Suspect* et publier cette information dans la base de données distribuée du nuage. Si une quelconque troisième machine M_3 (distincte de M_1 et M_2) possède l'information comme quoi M_1 est en état *Suspect* et qu'une communication ne peut y parvenir,

M_3 passe la machine M_1 de l'état *Suspect* à l'état *Déconnecté* et publie cette information dans la base de données.

Dans tout cas où une communication peut être établie depuis une machine quelconque vers une machine en état *Suspect*, le fait de pouvoir établir la communication retire la supposition de suspicion. Dans ce cas, l'état de la machine est changé de *Suspect* en son état réel (*Owner*, *Unclaimed*, *Matched*, ou *Claimed* selon le cas, chaque machine ayant la connaissance de son statut et pouvant effectuer elle-même cette mise à jour).

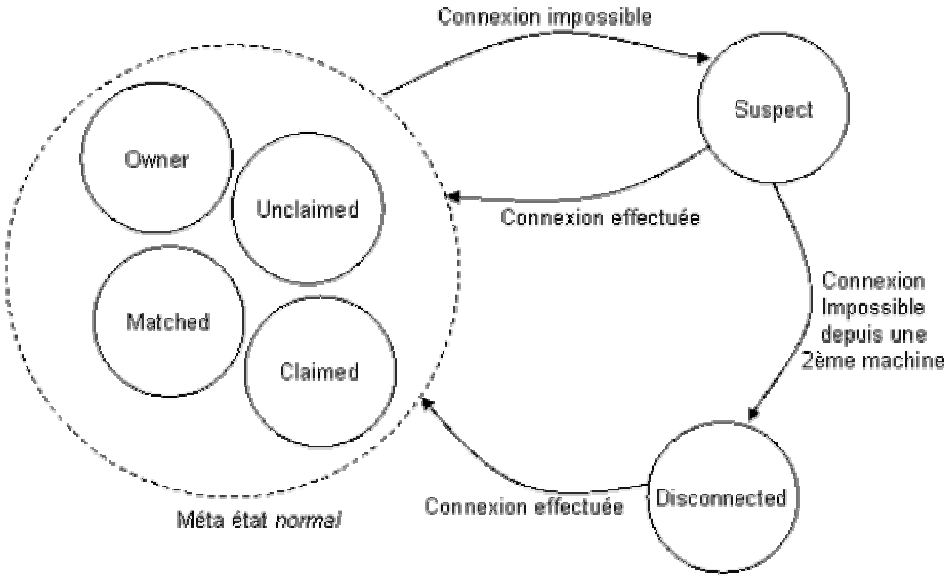

Figure 26 : Automate d'états montrant le passage en état suspect puis déconnecté

L'estampille temporelle associée à ces changements d'état permet de garder une relation d'ordre entre les évènements se produisant sur les diverses machines du nuage.

Précisons aussi que cette mécanique utilise l'attribut *SubmitUuid* de l'objet *State*, décrit ci-dessus (chapitre 3.3.2.1) afin de permettre le stockage de l'identifiant de la machine ayant publié cette information. De cette manière, si une même machine M_2 identifie une machine M_1 comme déconnectée (connexion impossible), elle publie cette information en y attachant son propre identifiant. De cette manière, si la même machine M_2 renouvelle la tentative de connexion vers M_1 et échoue encore, cet échec sera ignoré au sens de l'algorithme et ne provoquera pas de modification dans la base de données.

Ce mécanisme évite un fonctionnement pervers de l'algorithme : Lorsqu'une machine est déconnectée du réseau, elle va identifier toutes les autres comme « suspectes », car elle ne pourra plus se connecter à aucune autre machine. Chacune des autres machines va donc passer en état *Suspect*. Cependant, elle ne pourra pas les considérer comme déconnectées : Elle ne pourra pas changer l'état des autres machines, car c'est elle-même qui a déjà provoqué leur premier changement d'état vers *Suspect*. Seule une autre machine (d'identifiant différent) pourra provoquer le changement de toutes ces machines de l'état *Suspect* vers l'état *Shutdown*. Donc, dans le cas d'une machine

déconnectée du réseau, toutes les autres machines passeront (après un certain temps) en état *Suspect*, mais aucune ne sera en état *Shutdown*. Lors de l'éventuelle reconnexion de la machine au réseau, aucune perturbation du nuage ne sera provoquée et les machines repasseront toutes rapidement dans leur état réel (*Owner*, *Unclaimed*, *Matched*, ou *Claimed*).

Lors de la reconnexion d'une machine qui a été mise en état *Suspect* ou *Shutdown* par les autres machines du nuage, cette machine, lors du bavardage avec les pairs, va s'apercevoir que son état a été modifié par une machine tierce. Dans ce cas, elle va mettre à jour ses informations « State » avec l'estampille temporelle du moment, puis publier cette information dans le nuage, qui va la diffuser et donc écraser les anciennes valeurs de l'état.

Dans la méthode Load de la base de données, cela donne lieu à un traitement spécial : Lors de la lecture de l'objet d'état (cas où k = State) du processus local, si la valeur v retournée indique un état suspect (c'est-à-dire que l'attribut State est à la valeur *Suspect*) nous mettrons à jour notre état à la valeur courante, par exemple *Unclaimed*, avant de retourner cette nouvelle valeur. Ceci a pour effet d'annuler la mise de la machine locale dans la liste de suspects.

```
if (k = State) then
  if (uuid == MyUuid) then
    if SubmitUuid <> NULL    // Quelqu'un d'AUTRE a publié des infos sur moi !
      then
       UpdateMystate()
```

Le protocole utilisé pour le nuage est résilient aux fautes au niveau d'une connexion entre deux machines. C'est-à-dire qu'en cas de faute d'une des deux machines connectées, la session sera annulée soit proprement (NAK), ce qui ne provoquera pas de modification de l'état de chacune des machines, soit par une rupture de la session. Dans ce cas, la connexion sera réputée comme impossible (cf. la gestion de ces cas ci-dessous) et la session sera annulée.

3.3.5.3. Impact des erreurs de détection

Dans certains cas, il se peut que la détection des machines déconnectées soit incorrecte (*false positives*). Soit que la détection prenne du temps, soit que la connectivité de la machine soit incomplète et ne permette pas de la juger comme parfaitement connectée ni déconnectée. Nous allons voir ici que l'impact de ces incorrections n'est pas trop grave.

Si on a incorrectement détecté une machine comme déconnectée alors qu'elle ne l'était pas, l'impact reste limité. En cas de demande d'allocation de machines à un travail, elle ne pourra pas être utilisée. Par ailleurs la rapidité de convergence des informations dans le nuage assure que l'erreur sera rapidement réparée, et que la machine ne restera en état déconnecté que pendant une période courte (quelques secondes) avant de se retrouver en état connecté.

Si on a incorrectement détecté une machine comme connectée alors qu'elle ne l'était pas, l'impact est là aussi limité. En cas de demande d'allocation de machines à un travail, cette machine ne pourra pas répondre au protocole d'allocation (Match Finder, que nous décrirons dans le chapitre suivant) et elle passera aussitôt en état déconnecté.

3.3.5.4. Détection des fautes – Connexions impossibles

Le nuage est une base de données distribuée. Les données sont répliquées entre les pairs participant au nuage.

Une particularité du nuage est que la vision locale de chaque pair du système peut ne pas être complète. Personne n'a de vision globale du système. Une vision globale d'un système distribué ne permettrait de toutes façons pas de résister aux fautes permanentes des machines [Tan95].

3.3.6. Adjonction de nouveaux pairs au nuage

Plusieurs mécanismes sont envisageables pour ajouter de nouveaux participants à un système pair-à-pair et procéder à l'amorçage de leur bases de données de pairs :

Un mécanisme, utilisé par de nombreux systèmes tels que KazaA, Napster, Freenet, est d'utiliser un point d'accès central. Ce point central est connu de l'application, et le nouveau pair va effectuer une requête au serveur afin d'obtenir l'adresse d'autres participants de la communauté. Par la suite chaque pair va gérer sa propre liste d'adresses en fonction des relations qu'il a pu avoir avec les autres pairs, et va aussi partager sa propre liste de manière à ce que les listes locales d'adresses soient dynamiquement mises à jour sans avoir besoin de faire intervenir le serveur central. Cependant ce mécanisme demande une gestion coûteuse au niveau du serveur central, qui est alors un point faible du système. En particulier, le passage à l'échelle peut être difficile comme le montrent des projets comme Condor ou l'annuaire LDAP de Globus (aujourd'hui abandonné pour des raisons de saturation des serveurs).

Un autre mécanisme de découverte qui est utilisé dans JXTA, Jini ou Groove est de disposer d'un canal d'annonce (Broadcast ou Multicast). Chaque nouveau participant annonce sa présence sur ce canal et attend la réponse d'un quelconque autre participant. Ce mécanisme souffre cependant de deux problèmes principaux : il est nécessaire de procéder à une gestion fine du canal de communication pour éviter les avalanches de réponses (*Broadcast Storms*) ; et d'autre part un tel canal ne permet pas une annonce facile hors d'un intranet, car les messages Broadcast ou Multicast ne sont pas routés sur les réseaux Internet, sauf dans le cas exceptionnel du M-Bone Internet. Cette solution a été envisagée pour I-Cluster car les composants fournis par JXTA nous auraient permis de développer ce mécanisme très facilement à l'aide du système de canaux (*pipes*) à diffusion de groupe proposé. Cependant, le modèle JXTA se repose sur un mécanisme nécessitant la connaissance de la hiérarchie de pairs par chaque participant, et ne peut donc pas passer à l'échelle sans mettre en place une hiérarchie de super-pairs. Le modèle du nuage, où la liste des pairs est indépendante pour chaque participant et très dynamique, ne permet pas une utilisation facile d'un tel mécanisme.

Le dernier mécanisme, qui a été retenu pour notre architecture, est basé sur une graine externe (*external seed*). Pour participer à la communauté d'un nuage il sera nécessaire de connaître l'un des participants de ce nuage. Son adresse sera entrée comme celle d'un unique membre de la base de données locale. Lors de la première mise en relation avec cette machine, le nouveau participant recevra alors par bavardage la liste des membres connus par la dite machine, et sa base de données locale sera alors amorcée.

3.3.7. Conclusion sur le nuage I-Cluster

Le nuage I-Cluster est une contribution intéressante de notre étude. Il permet de remplir la fonction de bases de données distribuée sur un grand nombre de machines, ne nécessitant pas de serveur central, permettant la mise à jour des informations contenues de manière à permettre une convergence globale rapide des différentes répliques de ces informations.

Basé sur les technologies pair-à-pair, ainsi que sur un mécanisme de bavardage hérité du projet MOSIX amélioré de manière à permettre un passage à l'échelle sur plusieurs dizaines de milliers de machines, le nuage I-Cluster a pu être prototypé sans toutefois passer le cap de la validation sur un grand nombre de machines, le code de l'agent n'étant pas assez stable pour pouvoir fonctionner de manière productive.

3.4. Description et lancement de travaux

Nous avons vu jusqu'ici comment nous pouvions découvrir les machines d'un intranet, et distribuer les informations relatives à leurs ressources individuelles, leurs fenêtres de tir respectives ainsi que leur connectivité et leur situation sur la topologie du réseau. Le *nuage*, décrit plus haut, nous permet d'accéder à une base de données d'information sur les machines susceptibles d'être utiles à des travaux de calcul.

Afin de pouvoir identifier des grappes virtuelles parmi les machines disponibles, et de faire correspondre de telles grappes à des travaux soumis par les utilisateurs, il nous faut examiner d'une part comment faire correspondre une soumission d'un travail au meilleur ensemble de machines disponibles dans le nuage, c'est-à-dire les machines de puissance et de capacités adaptées au travail, ainsi que le nombre de machines nécessaire et leur éventuelle proximité sur le réseau qui permettront de pouvoir exécuter des travaux à grain fin. D'autre part, il nous faudra pouvoir lancer les travaux demandés, c'est-à-dire allouer les machines pour la durée de ce travail. Ces fonctions sont réalisées par le Match Finder, qui se rapproche fonctionnellement du *MatchMaker* de Condor mais en conservant l'aspect décentralisé du nuage I-Cluster et en prenant en compte la connaissance de l'interconnexion entre les machines à allouer.

Des travaux typiques soumis au Match Finder peuvent être, par exemple, les suivants.

Le lancement d'un travail parallèle nécessitant une machine biprocesseur avec au moins 2 Go de mémoire centrale et des processeurs cadencés à une vitesse supérieure à 2,5 GHz. Dans ce cas, l'identification de la grappe virtuelle se limite à l'allocation d'une unique machine pour le travail soumis.

La décomposition d'un travail de rendu graphique en lancer de rayon (POVray) sur plusieurs machines. Il est par exemple possible dans un tel cas de lancer le travail sur une dizaine de machines. On observe que la puissance de calcul totale obtenue est proche de la somme des puissances individuelles de chaque machine participant au travail. Ceci s'explique par le fait que le grain de calcul est assez gros, donc les communications ne sont pas pénalisantes, et aussi

car le travail à exécuter consiste essentiellement en travail du processeur, les entrées (lecture du script à exécuter) et sorties (rendu dans un fichier image ou sur un terminal X-Window) étant très légères en rapport.

L'exécution d'un programme parallèle d'algèbre linéaire en grain fin. Comme nous l'avons montré ci-dessous, l'exemple de Linpack s'applique bien à une grappe virtuelle identifiée parmi les machines en jachère sur l'intranet. Toutefois, rappelons qu'il nous faudra veiller à obtenir des machines très homogènes en termes de performances individuelles, et à ce que la distance maximale entre eux (en termes de topologie réseau) soit petite, typiquement il faudra identifier des machines appartenant à un même sous-réseau Ethernet.

Le lancement d'un programme massivement parallèle, par exemple par exécution multiparamétrique d'un même travail avec un grand nombre de données d'entrée différentes, avec un nombre de machines important (supérieur au nombre de machines disponibles). Dans ce cas il faudra pouvoir se reposer sur un ordonnanceur qui prendra en compte la distribution des travaux individuels sur chaque machine, en se chargeant de la répartition de la charge.

L'exécution d'une tâche malléable (qui pourra s'adapter à un nombre variable de machines d'exécution) dans un temps maximum donné. Il faudra donc pouvoir déterminer, en fonction des machines disponibles, de leurs performances individuelles et des caractéristiques du travail à lancer, quelles machines allouer à ce travail afin de pouvoir garantir sa terminaison avant la date limite.

Ces exemples nous mettent en lumière les contraintes que nous devons observer au niveau de notre Match Finder : Le respect des fenêtres de tir de chaque machine, l'adaptation du choix des machines au travail demandé en fonction de leurs performances, de leur nombre et de leur homogénéité, ainsi que leur proximité sur le réseau.

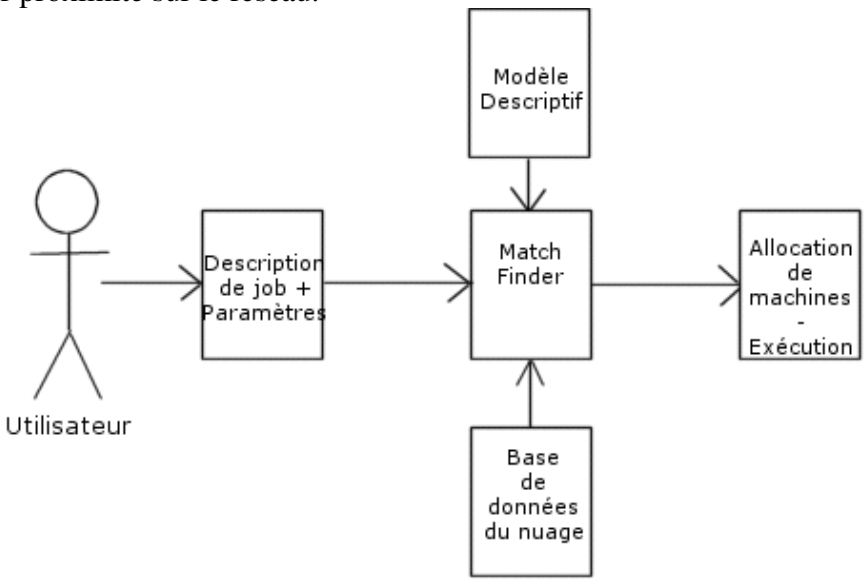

Figure 27 : Interactions du Match Finder

La figure ci-dessus nous montre les interactions du Match Finder avec les autres composants. L'utilisateur va soumettre un travail, dans lequel il va donner le type d'application à exécuter ainsi que les paramètres nécessaires à l'application, les fichiers d'entrée éventuels, et où doivent être placés les fichiers de sortie. L'utilisateur va de même soumettre une politique de sélection, permettant d'imposer un temps d'exécution maximal (ce qui nécessitera donc des machines puissantes et/ou nombreuses), un travail prioritaire ou non, des contraintes économiques qui pourraient être utiles si certaines des machines disponibles ont un coût d'utilisation élevé.

Le Match Finder connaît une bibliothèque de modèles descriptifs d'applications, qui pour chaque application vont lui donner les caractéristiques des machines nécessaires (taille de mémoire, vitesse et modèle des processeur). Le modèle, associé aux paramètres soumis par l'utilisateur, pourra permettre au Match Finder d'évaluer la charge du travail demandé par l'utilisateur.

Basé sur cette charge ainsi que sur la connaissance de la disponibilité des machines du nuage, le Match Finder va ensuite essayer de « reconnaître » la grappe virtuelle la plus adaptée parmi les machines disponibles sur le nuage et correspondant aux besoins du travail demandé.

Un allocateur de machines va ensuite procéder à la réservation des machines appropriées à l'aide d'un algorithme *double commit*, qui va tout d'abord envoyer une requête de réservation à chacune des machines, puis si toutes les machines ont confirmé la réservation, une requête d'allocation leur est envoyée.

Les machines sont alors entièrement disponibles pour l'application jusqu'à la fin de leurs fenêtres de tir respectives et le travail demandé peut être démarré. Si le travail demandé se termine l'application peut alors procéder à la dé allocation des machines, qui retournent alors dans un état d'attente de travail.

Le Match Finder développé n'a pas pu être finalisé, ni au niveau de son étude, ni au niveau de sa réalisation. Nous avons cependant pu étudier les problèmes posés et développer quelques pistes de recherche :

De nombreuses descriptions de travaux sont disponibles dans divers projets. Nous avons basé notre effort sur Condor, dont le *Match Maker* décrit ci-après permet la description de travaux et de machines, ainsi que le mécanisme permettant d'identifier les machines disponibles pour l'exécution d'un travail donné. Notre effort a principalement consisté en l'extension de la grammaire de Condor pour permettre la description de travaux parallèles, demandant une certaine homogénéité entre les machines et éventuellement un latence réseau basse.

Il est désirable pour l'utilisateur de faire une soumission de ses travaux sans se soucier du support d'exécution. En effet, certains travaux pourront être résolus par une seule machine biprocesseur ou bien par une grappe virtuelle de 4 machines, avec la même efficacité. L'utilisateur ne doit pas avoir à se soucier des machines allouées, mais uniquement du résultat après exécution. Il apparaît donc important de pouvoir créer un modèle de description de travaux qui permettra le meilleur ajustement possible entre le travail soumis par l'utilisateur et les machines disponibles sur le nuage. Dans certains cas tels que Linpack, il

est important d'allouer une matrice carrée (NxN) de machines afin que l'implémentation HPL puisse être efficace. On préfèrera donc allouer 16 machines homogènes à un travail Linpack que 20 machines, ce qui va déséquilibrer les flots de messages entre les machines et dégrader fortement la performance d'exécution globale. A notre connaissance, ce type de description de travaux, qui permettrait la sélection de la grappe virtuelle la plus appropriée à un travail donné, n'a pas encore fait l'objet de publications, bien que certains projets tels que AppLeS [BWC+03] suivent une méthodologie proche de la nôtre. L'étude de RSL (*Resource Specification Language*), GRAM (*Grid Resource Allocation Management*) et GRIS (*Grid Resource Information Service*) du projet Globus peuvent servir de base, ainsi que les éléments de DIET [Cde03][Lom02].

Dans le cadre d'applications manipulant de grosses quantités de données, il est important de pouvoir décrire l'utilisation qui sera faite de ces données par l'application afin de pouvoir tenir compte non seulement des caractéristiques individuelles de chaque machine, mais aussi de la topologie du réseau d'interconnexion. De cette manière, il sera possible de distribuer les données à chaque machine en faisant intervenir cette topologie, et de sélectionner les machines qui permettent un arbre de distribution le plus efficace possible.

3.4.1. Description des travaux

Les travaux devant être exécutés doivent être décrits en termes de types de machines et environnement nécessaire. Le *Match Maker* de Condor dispose déjà des fonctionnalités nécessaires, à l'aide des descriptions par ClassAds des travaux et des machines disponibles dans le nuage.

Notre objectif étant d'exécuter des travaux parallèles, nous avons étendu la grammaire de Condor afin de pouvoir faire des requêtes sur des travaux demandant plus d'une machine, ainsi que des travaux en grain fin, nécessitant donc des machines appartenant à une même localisation du réseau, par exemple au même sous-réseau Ethernet.

Les ClassAds (*litt. « Petites annonces »*) de Condor permettent pour chaque machine d'annoncer le type de ressources qu'elle peut offrir. Le Match Maker, lorsqu'un travail lui est soumis, va ensuite se baser sur les contraintes et les préférences associées à ce travail pour sélectionner la machine la plus appropriée et lui soumettre son exécution.

3.4.2. Match Finder

Le prototype actuel de Match Finder ne remplit pas réellement sa fonction de sélection d'une grappe virtuelle la plus adaptée possible au travail soumis. Il se résume à une recherche d'une machine répondant à un jeu de critères donnés à l'aide d'une recherche d'expressions régulières parmi les caractéristiques exposées par les cartes de ressources des machines référencées dans le nuage :

Les cartes de ressources sont constituées d'une chaîne de caractères de la forme « attribute = value attribute = value… ». On peut donc, par exemple, soumettre à notre prototype une requête du type :

```
Arch = "INTEL" Memory = "256" OpSys = "I-Cluster"
```

qui sélectionnera toutes les machines du nuage comportant au moins 256 Mo de mémoire et basées sur une architecture Intel. Notre mécanisme de sélection se résume à une recherche de sous-chaîne dans une chaîne et n'a pas d'autre intérêt que de pouvoir démontrer l'invocation de requêtes d'allocation au nuage.

Une fois identifiées les machines du nuage correspondant à la requête, une machine aléatoire est sélectionnée parmi celle-ci. L'état de la machine est alors passé en mode *Matched*, et elle peut ensuite être utilisée à l'aide d'une commande *rsh* d'exécution à distance. L'utilisation optionnelle de *ssh* permet de sécuriser cette communication.

3.4.3. Conclusion sur le Match Finder

Dans notre étude, le Match Finder constitue la partie la moins achevée. Nous avons pu évaluer divers mécanismes utilisés aujourd'hui par divers projets pour l'allocation de ressources de calcul communautaires. Cependant, il n'existe pas (encore) de projet qui permette la reconnaissance d'une grappe virtuelle optimale parmi un ensemble de machines, avec un problème donné.

Nous avons fait un travail d'analyse léger, ainsi que la réalisation d'un prototype fonctionnel minimal permettant au système I-Cluster de fonctionner, mais le Match Finder reste une des parties de notre architecture qui demande encore le plus d'effort avant de pouvoir remplir son objectif d'allocation transparente et efficace de grappes virtuelles appropriées à un travail donné.

3.5. Analyse sur une infrastructure expérimentale

Notre objectif étant d'utiliser les machines en jachère sur un intranet standard afin de les agréger pour faire du calcul intensif, il apparaît important de pouvoir évaluer la capacité d'une telle infrastructure pour faire du calcul. En effet, les PC de l'intranet considéré disposent individuellement d'une puissance de calcul importante, chacun d'eux étant équipé d'un processeur de classe Pentium à une fréquence élevée (typiquement 1 GHz), et d'une mémoire conséquente. Cependant, un calcul mettant en commun ces ressources ne disposera pas du cumul des puissances individuelles des machines. On peut ici évoquer l'importance du grain de calcul [SGT96] : Les problèmes à gros grain de calcul se décomposent en tâches indépendantes, qui vont pouvoir être effectuées simultanément sur différentes machines de l'intranet, et la puissance globale disponible sera alors de l'ordre de la somme des puissances de calcul individuelles de chacune des machines utilisées.

Toutefois, il ne s'agit pas ici de se limiter à des calculs à gros grain. Un calcul à grain fin nécessite une distribution des tâches sur les différentes machines, mais aussi une communication importante entre les machines. La synchronisation nécessaire entre les tâches s'exécutant sur les différentes machines provoque d'une part des temps d'inactivité pendant l'attente de messages, d'autre part des ralentissements dus aux temps de transferts des messages ainsi qu'aux latences du réseau, et enfin des risques de saturation du réseau qui provoqueraient un écroulement global du calcul. Il est donc difficile d'obtenir une bonne puissance de calcul en grain fin à cause de ces pertes.

Pourtant, des grappes basées sur du matériel standard sont couramment utilisées à des fins de calcul scientifique haute performance. Afin de permettre une bonne souplesse d'utilisation et en particulier pour pouvoir exécuter efficacement des calculs à grain fin, la plupart des architectures de calcul sur grappes de PC actuelles disposent de certaines caractéristiques :

- homogénéité. Les machines d'une grappe sont généralement toutes identiques en termes de capacités matérielles.

- réseau haut de gamme. Les communications se font au travers de réseaux à faible latence et à haut débit. Myrinet, SCI ou Quadrics sont des exemples de telles infrastructures.

- machines puissantes. Le réseau étant un facteur limitant des grappes en calcul fin, le nombre de machines d'une grappes est minimisé, tout en augmentant la puissance individuelles de chaque machine. Il est fréquent de trouver des machines biprocesseurs, ce qui donne [DKo98] un meilleur rapport performance/coût pour la grappe.

Il est évident que dans notre cas, aucune de ces trois caractéristiques ne peut être assurée. Il n'est pas question de modifier la configuration matérielle des machines disponibles sur l'intranet. Nous ne pouvons donc pas disposer d'un réseau à basse latence entre les machines et nous devrons nous reposer sur une interconnexion Ethernet standard.

Par contre, nous disposons de beaucoup de machines sur l'intranet, et pour effectuer un calcul donné il devient possible de sélectionner un groupe de machines de l'intranet ayant des caractéristiques telles que homogénéité de configuration matérielle, ou bien proximité des unes avec les autres selon la topologie du réseau. En effet, si à un instant donné on dispose de 1000 machines en jachère sur un intranet, il ne devrait pas être trop difficile d’en identifier 8 par exemple, qui soient puissantes, homogènes et proches, et que l’on pourra utiliser comme une grappe virtuelle efficace.

Afin de mesurer la capacité réelle d'une grappe composée d'un groupe de machines d'un intranet, nous avons effectué une série d'expériences entre novembre 2000 et mars 2001 sur une plate-forme d'expérimentation servant de modèle à une grappe de machines d'un intranet standard. Ces expériences basées sur Linpack (factorisation d'une matrice de grande taille) ont permis de montrer la faisabilité d'un calcul en grain fin sur un grand nombre de machines dans des conditions d'exploitation standard telles que celles qu'on peut trouver sur un intranet d'entreprise.

En octobre 2000, une première partie de la plate-forme d'expérimentation I-Cluster était construite, composée de 100 machines interconnectées par 3 commutateurs Ethernet 100. Après avoir affiné la configuration logicielle, nous avons mesuré une performance de 36 Gflop/s sur ces 100 machines dans les conditions fixées par le TOP500 [MSD+01]. Cette performance était plus élevée

que ce que nous avions anticipé, et nous avons décidé d'étendre la plate-forme à 225 machines et de déterminer comment tirer une performance maximale de cette grappe, afin d'entrer dans le classement TOP500.

3.5.1. Echelle de performances

Il est tentant de développer un superordinateur à partir de composants de grande consommation. Ces types de composants étant produits en grands volumes, leur prix est bas alors que leur qualité est élevée. Nous pourrions donc imaginer acheter des milliers de PC standard et les interconnecter en utilisant des technologies réseau communes. Un bon exemple d'un tel ordinateur à grande échelle (*métaordinateur*) est utilisé par le projet SETI@home [And+99][And02], qui réunit des millions de machines connectées à Internet, et qui atteint une performance cumulée de plusieurs Tflop/s.

En ce qui concerne les *superordinateurs*, une manière classique de mesurer leur performance est d'utiliser le benchmark Linpack tel qu'il est réglementé par le consortium TOP500. Linpack évalue le système par des calculs d'algèbre linéaire à grande échelle sur des matrices pleines. Ce type de programme nécessite de lourds calculs en double précision sur chacun des nœuds de calcul, mais il requiert aussi beaucoup de communications entre les nœuds (principalement des communications de groupe synchrones). S'il était exécuté sur un métaordinateur tel que celui construit par le projet SETI@home, le benchmark Linpack donnerait des résultats très mauvais, car -entre autres- la communication entre les nœuds souffre de la latence inhérente à Internet. D'un autre côté, les superordinateurs (ou les grappes de calcul haute performance) composées de matériel sophistiqué et de réseau haute performance passent bien à l'échelle avec Linpack, mais leur prix est très élevé. Un objectif de notre expérience était de déterminer, pour un ensemble de machines de bureau standard interconnectées par un réseau d'entreprise, les limites du passage à l'échelle relativement au coût d'un superordinateur.

3.5.2. Configuration de la plate-forme d'expérimentation I-Cluster

3.5.2.1. Les nœuds de calcul

Notre grappe d'expérimentation est composée de 225 HP e-PC. Chacun de ces PC est équipé d'un Pentium III ® à 733 MHz, de 256 Mo de RAM, et d'un disque de 15 Go. Le e-PC représente l'évolution du PC classique vers une machine plus petite, plus simple et plus fiable, conçue comme un PC de bureau d'entreprise typique. Dans notre contexte, ceci nous apporte la stabilité ainsi qu'un modèle de maintenance simplifié. L'espace nécessaire est bien plus faible que si nous avions utilisé des machines de bureau classiques, et les besoins en puissance électrique ne sont que d'environ 50 W par machine, ce qui ne constitue que 30% de la puissance requise par un PC classique. La ventilation et la climatisation sont plus faciles à établir avec une grappe de e-PC qu'avec une grappe de machines standard. Nous avons pu observer entre décembre 2002 et juillet 2003, lors de la recette d'une grappe de 104 Itanium II par l'INRIA, de nombreux soucis tels que la consommation trop élevée pour le réseau électrique, le manque de climatisation, le surpoids à supporter par les parquets de la salle des machines.

D'un autre côté, le fait d'avoir choisi les e-PC, avec leur modèle de maintenance, nous interdit de modifier leur configuration matérielle (boîtier scellé). Il nous était donc impossible d'ajouter de la mémoire, une deuxième carte réseau ou de changer le type de réseau pour choisir un réseau basse latence, haute performance tel que Myrinet [BCF+95], SCI [Gus92] or VIA [Via97][Dun+98], qui sont le lot commun des grappes de calcul.

3.5.2.2. La connectivité

La technologie de réseau étant contrainte à Ethernet 100 par les HP e-PC, les nœuds sont connectés à cinq commutateurs Ethernet 100. Ces commutateurs sont des HP ProCurve 4000 interconnectés entre eux par gigabit Ethernet. Notons que cette configuration correspond à la configuration standard que l'on peut trouver dans un intranet d'entreprise. Ethernet étant choisi comme mode d'interconnexion, les commutateurs doivent utiliser des algorithmes *store-and-forward* pour manipuler les trames Ethernet et calculer leurs routes. Ceci provoque un délai lors du traitement de chaque trame, augmentant la latence réseau et diminuant la performance globale de la grappe. Notre configuration nous permet d'interconnecter les commutateurs en un pentagramme complet, un anneau ou un anneau double (cf. Figure 28 : Topologie de la plate-forme d'expérimentation I-Cluster). Dans la configuration en pentagramme complet, chaque commutateur est connecté aux quatre autres à l'aide d'un lien Ethernet 1000TX. Plusieurs alternatives ont été évaluées. Une arborescence de commutateurs, par exemple, donnerait une bonne performance réseau car le routage est simple, mais les communications inter-commutateurs auraient nécessité de traverser 3 commutateurs, ce qui aurait donné une mauvaise latence réseau.

Les réseaux à voisinage plat (*Flat Neighborhood Networks*) tels que décrits par [DMa00] présentent une alternative pour la topologie réseau intéressante, mais nécessitent plusieurs cartes réseau dans chaque nœud de la grappe. Dans notre cas, le choix du modèle HP e-PC, qui n'est pas extensible, nous interdit le support de plus d'une carte réseau.

Avec notre interconnexion en pentagramme complet, les trames Ethernet naviguant d'un PC à un autre PC attaché à un commutateur différent doivent traverser chacun des deux commutateurs, doublant donc la latence due aux commutateurs. Comme nous le verrons plus loin, la connexion sur le même commutateur de tous les nœuds d'une rangée Linpack améliore sensiblement la performance du benchmark (typiquement 14%).

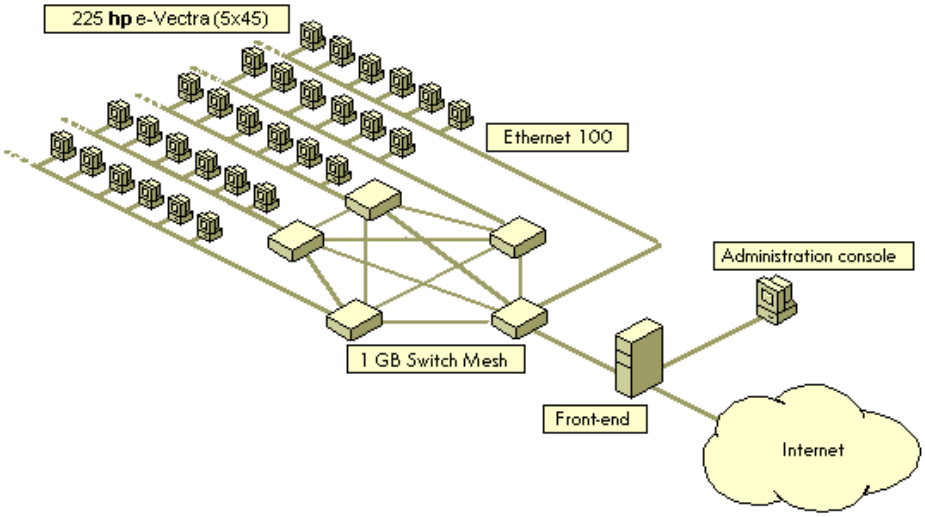

Figure 28 : Topologie de la plate-forme d'expérimentation I-Cluster

3.5.2.3. Le logiciel

La plate-forme d'expérimentation I-Cluster était basée sur un système d'exploitation Linux Mandrake 7.0 (différentes versions de noyau ont été testées sans montrer de bénéfice important).

3.5.2.4. Le benchmark parallèle Linpack

Nous avons utilisé le paquetage HPL[4] (*High-Performance Linpack*) [DBM+79] comme base de nos expériences. Il consiste en une factorisation LU par décomposition de Cholesky distribuée d'une matrice dense de *n x n* nombres double précision. La complexité de l'algorithme est de l'ordre de :

$$ops = \frac{2.n^3}{3} + 2.n^2$$

La parallélisation est faite par une décomposition en blocs de la matrice, ce qui permet d'une part d'appliquer un algorithme récursif, et une granularité séquentielle qui est compatible avec l'utilisation calculs matriciels séquentiels, ce qui permet l'appel à des bibliothèques spécialisées à haute performance telles que BLAS [LHK+79].

L'algorithme de bloc nécessite une distribution en grille virtuelle des processeurs disponibles, de manière à projeter la matrice bidimensionnelle sur une topologie à deux dimensions. L'analyse numérique montre que la plupart des communications se font par diffusion entre les processeurs d'une même ligne de blocs. Une grille extrêmement plate provoquerait donc des goulots d'étranglements dus aux diffusions intensives.

[4] *Dans ce document, "HPL" ou "test Linpack" fera référence au programme HPL sauf contre-indication.*

D'autre part, la phase de l'algorithme correspondant à la propagation du bloc pivot -qui intervient pour chaque ligne- est un évènement de synchronisation : tous les processeurs doivent attendre le bloc pivot avant de pouvoir exécuter les calculs ultérieurs. Il n'est donc pas possible d'obtenir une bonne efficacité si la grille est trop profonde. L'équilibre entre le nombre de lignes et de colonnes de la grille est donc l'une des questions clefs pour l'efficacité de l'algorithme parallèle. [DBM+79] fournit une analyse complète des coûts de communication de l'algorithme. Leur conclusion est que, sur un modèle de machine simplifié, l'efficacité parallèle (rapport entre le temps séquentiel et le temps parallèle réduit à un seul processeur) est une fonction décroissante de la quantité $lp.\left(\frac{pq}{n^2}\right)$, avec p le nombre de lignes de la grille, q le nombre de colonnes, n la taille de la matrice, et l la latence du réseau. Chaque nœud requiert donc une taille de mémoire de $\frac{n^2}{pq}$, et si on la garde constante (i.e. en augmentant la taille de la matrice lorsqu'on augmente le nombre de nœuds participant au calcul), il ne nous reste qu'un facteur limitant dû au produit *latence x temps de diffusion*, ce qui est cohérent : Plus la grille est plate, plus la diffusion est longue, et plus l'exécution parallèle prend du temps. La latence joue un rôle très important dans cette inefficacité. Ce dernier résultat nous a poussés à optimiser le réseau et en particulier la pile TCP-IP du système d'exploitation, les gestionnaires de périphériques, et la couche de diffusion logicielle implémentée par la bibliothèque de passage de messages.

3.5.2.5. L'optimisation des paramètres

Les expériences ont démarré lors de la réception du premier lot de machines en octobre 2000. Pour des raisons de limitation de budget nous avions commencé avec 100 machines. Ce nombre a été augmenté à 216 machines en mars 2001, puis à 225 en avril. Ce nombre de 225 machines nous permet de constituer une grille virtuelle de 15x15 machines. Nous verrons plus loin l'importance d'avoir une grille carrée.

Nous avons effectué de nombreuses expériences afin d'évaluer l'importance de chaque composant et paramètre dans la performance globale de la grappe. L'idée était de réaliser des plans d'expérience permettant d'optimiser chacun des paramètres de manière la plus indépendante possible, d'identifier les optimisations prometteuses en termes de gain de performance, de synchroniser les optimisations puis de répéter le processus jusqu'à ce que plus aucune modification de paramètre ne donne de gain. Nous nous sommes concentrés sur les paramètres suivants :

- choix de la bibliothèque BLAS (MKL, Atlas…), compilateur utilisé et options de compilation,
- paramètres de Linpack (taille de matrice, taille de bloc, algorithme de diffusion),
- système d'exploitation et configuration des gestionnaires de périphériques,
- topologie réseau, y compris l'adaptation aux paramètres de Linpack.

Nous avions donc de nombreux niveaux d'optimisation à gérer, depuis le matériel et la topologie jusqu'aux niveaux frontaux tels que les options de Linpack.

La bibliothèque BLAS

La bibliothèque BLAS est responsable des opérations vectorielles et matricielles de base. Ces routines sont la base des opérations faites par le logiciel Linpack.

Les premières mesures ont été faites à l'aide d'une bibliothèque BLAS optimisée avec le logiciel ATLAS [PWD01], qui optimise la bibliothèque BLAS en fonction du matériel disponible sur chacun des nœuds de calcul. Le compilateur *gcc* a été utilisé pour les premières expériences. Nous avions prévu d'utiliser le compilateur Intel ou celui de Portland Group afin d'exploiter les possibilités *Streaming SIMD Extensions* (SSE) du processeur Intel Pentium ® III, qui permettent d'effectuer deux instructions double précision en une seule étape. Malheureusement une partie de ces extensions ne sont pas activées sur le Pentium III, et nous n'avons donc jamais pu mesurer d'avantage majeur à utiliser l'un ou l'autre de ces compilateurs.

Nous avons comparé plusieurs bibliothèques BLAS : Atlas, La bibliothèque *Intel Math Kernel Library* ainsi qu'une bibliothèque du projet ASCII (qui est en fait une fusion entre la bibliothèque Atlas et un *dgemm* optimisé pour le Pentium III par Greg Henry de Intel).

20 essais ont été faits pour chaque bibliothèque BLAS :

	Moyenne	**Ecart type**
Henry	499.38	6.52
Atlas 3.2.0	493.96	1.8
Atlas 3.2.1	478.3	1.7
MKL	439.9	2.1

Table 2 : Performance des bibliothèques BLAS en Gflop/s

La conclusion de ces essais a été de choisir une bibliothèque basée sur Atlas, avec les optimisations de *dgemm* de Greg Henry.

Les paramètres Linpack

Le test HPL comporte lui-même un certain nombre de paramètres, sans compter ceux disponibles au moment de la compilation. Entre autres, il est possible d'ajuster la taille *N* de la matrice; la taille *N_b* des sous blocs; le nombre de rangées (*p*) et de colonnes (*q*) de la grille virtuelle de processeurs; l'algorithme utilisé pour la factorisation (*pfact*); la taille minimale d'un bloc pour faire un appel récursif à l'algorithme de factorisation; le type de diffusion requise (plusieurs arbres de recouvrement sont disponibles); le nombre de pivots à calculer avant de les envoyer aux autres processeurs; et l'algorithme à utiliser pour échanger les pivots. Certains de ces paramètres dépendent les uns des autres. Une partie importante de notre travail a été d'ajuster chacun d'eux en

fonction de l'architecture de notre grappe. Le guide de configuration de HPL suggère l'utilisation d'une grille virtuelle carrée, avec un nombre de colonnes légèrement plus grand que le nombre de lignes. Evidemment, en fonction de la topologie, la diffusion doit être modifiée. Apparemment, la possibilité de décomposer *N* en deux facteurs premiers proches est prédominante pour le passage à l'échelle de la performance (une décomposition en N = n.(n+1) se révèle optimale expérimentalement).

L'algorithme de diffusion

Linpack est un programme synchrone basé sur MPI, utilisant donc un mécanisme de *rendez-vous*. Si un des nœuds de calcul est plus lent que les autres, la performance globale en souffre, car les nœuds plus rapides vont s'aligner sur le plus lent pendant les rendez-vous. HPL dispose de 6 algorithmes de diffusion. Chacun d'eux a été testé exhaustivement, dans presque toutes les configurations. La diffusion optimale a dû être modifiée entre les tests sur 98 machines et ceux sur 210 et 225 machines. Néanmoins, il est facile de sélectionner l'algorithme de diffusion convenant le mieux à une forme de grille donnée, car ce paramètre est indépendant des autres. Nous avons aussi testé des algorithmes de notre conception, qui étaient des variations autour des algorithmes prédéfinis. Nous avons aussi testé l'algorithme *MPI_Bcast* de MPICH, qui utilise un arbre binomial et qui est donc différent de tous les algorithmes de HPL. Aucune de ces expériences n'a donné de résultats meilleurs que ceux des algorithmes natifs de HPL. Une expérience aurait pu être menée sur un algorithme de diffusion reposant sur du IP multicast, mais nous n'avons pas disposé du temps nécessaire à sa mise en oeuvre. Cet algorithme présente l'énorme avantage de ne pas nécessiter d'arbre de recouvrement et permet d'envoyer un message de groupe en temps constant, quel que soit le nombre de récepteurs. D'autres équipes de recherche ont mis en œuvre cet algorithme [YDF+02] avec succès. A noter que dans notre cas, où le réseau est équilibré et bien dimensionné, on peut raisonnablement espérer que les trames réseau ne seront jamais désynchronisées, perdues ou dupliquées, aussi la gestion d'erreur - qui est un problème inhérent au multicast IP- peut être supprimée au profit d'un gain de performances réseau.

La taille de la matrice et de ses blocs

Nous avons géré ces paramètres de manière empirique. Notre meilleure performance a été obtenue avec une matrice dont la taille mémoire était de l'ordre de 200 à 220 Mo par nœud :

$$M(n) = 8.\frac{n^2}{pq}$$

Il semblait que pour les matrices les plus grandes, certaines machines passaient parfois en mode *swap*. Ceci peut être dû à la gestion des tampons par MPI ou par le système d'exploitation. Une taille de 212 Mo a constitué le meilleur compromis.

La taille N de la matrice a finalement été choisie à 80370. L'inversion d'une telle matrice représente un effort de calcul important. Aucun homme n'est jamais assez fort pour ce calcul.

Concernant la taille de bloc, nous devrions avoir théoriquement :

- une approximation de la taille *M x N* de la matrice traitée par chaque nœud;
- la meilleure valeur pour le paramètre *K* de l'opération *dgemm* de BLAS, déterminée de manière séquentielle pour chacune des matrices *M x K* et *K x N*;
- les valeurs *N_b = K* et *N* fixées de manière appropriée. Ces valeurs devant respecter les règles *pN_b divise N* et *qN_b divise N*; soit encore *gpcd(p,q) x N_b divise N*.

3.5.2.6. Gestionnaires de réseau et noyau Linux

La plate-forme d'expérimentation I-Cluster est conçue pour une utilisation journalière, et nous avons donc choisi de garder la pile TCP/IP du noyau Linux 2.2.4. Ce choix dégrade la performance de Linpack à cause de la latence de la pile TCP. Cette latence est due en partie à des duplications de données entre l'espace utilisateur et la mémoire de la carte réseau. Ce phénomène pourrait être évité en utilisant du matériel spécifique (Myrinet [BCF+95], SCI [Gus92]) ou des architectures (VIA [Dun+98]) ou protocoles (GAMMA [CCi97]) spécifiques à basse latence. Jusqu'à maintenant, aucune de ces solutions n'offrent la même stabilité, coût et compatibilité que TCP. La latence réseau est fortement mise en exergue par HPL, par la transmission d'un grand nombre de message de taille moyenne.

Plusieurs paramètres ont été ajustés pour la performance Linpack : Par exemple la taille maximale des tampons TCP a été modifiée et plusieurs gestionnaires de carte réseau ont été testés. Chacun des gestionnaires a été testé sur des noyaux Linux 2.2.17 et 2.4.2. Ce dernier noyau montrait un peu plus de performance et a été utilisé pour la suite des tests.

3.5.2.7. L'architecture du réseau

Nos commutateurs permettent de connecter 45 machines sur chacun d'entre eux. Avec plus de 45 machines il faut donc utiliser une configuration avec plusieurs commutateurs et l'allocation des nœuds à la grille virtuelle de Linpack devient complexe. La première étape est d'équilibrer les nœuds considérés entre les commutateurs. Il nous faut ensuite spécifier un ordre pour les nœuds utilisés par MPI, tel qu'il construise la grille virtuelle avec les bons nœuds à la bonne place et minimise les communications inter-commutateurs. Pour un calcul avec 210 nœuds et 5 commutateurs, une bonne configuration nous a permis de gagner 5 Gflop/s et ainsi d'atteindre 76.4 Gflop/s au lieu de 71.8 Gflop/s.

Nous n'avons pu expérimenter que 3 topologies d'interconnexion des commutateurs : L'anneau, l'anneau double et le graphe complet (pentagramme). De manière surprenante, le passage de l'anneau en anneau double ne fait gagner que 1,5% sur Linpack. Ceci est dû au fait que les commutateurs doivent mettre

en œuvre un algorithme d'équilibrage de charge entre les liens doubles, et cet algorithme provoque une latence de traitement des trames réseau. La topologie en pentagramme ne donne pas de meilleurs résultats que l'anneau double.

3.5.2.8. Les résultats

Nous avons tout d'abord fait nos expériences sur une grappe de 100 nœuds, pour laquelle nous avons vu de très bons résultats en termes de passage à l'échelle. L'utilisation de 1 à 45 nœuds est facilitée par le fait que tous les nœuds sont sur le même commutateur. L'utilisation d'une grille virtuelle presque carrée produit les meilleurs résultats, comme prévu plus haut. Avec plus de nœuds et de commutateurs, le passage à l'échelle s'est révélé bon, mais le réglage des paramètres Linpack n'est pas facile. Des exécutions aveugles de Linpack fournissent des résultats non déterministes.

Grille	**Performance max. (Gflop/s)**
15 x 14	67.9
14 x 15	76.4
10 x 21	74.4

Table 3 : Performance par forme de grille

D'autre part, l'allocation optimale des nœuds sur la grille virtuelle n'est pas possible pour toutes les tailles de grappes. Nos meilleurs résultats avec 210 nœuds ont donné 76.4 Gflop/s alors qu'avec 215 nœuds nous n'avons pas pu dépasser 55 Gflop/s. Ceci s'explique par le fait que les seules grilles constructibles avec 215 nœuds ont une forme 5x43 ou 43x5, donc très loin d'être carrées.

Une grille 15x15 donne théoriquement des résultats optimaux. Pour obtenir les meilleurs résultats expérimentaux nous avons étendu la taille de matrice au maximum en évitant de provoquer du swap de mémoire. Le meilleur résultat a été de **81.6 Gflop/s**.

Ces expériences nous ont montré que l'augmentation du nombre de nœuds de calcul montrait une augmentation linéaire de la performance Linpack. La figure ci-dessous présente les mesures faites sur des machines d'un seul commutateur. Les formes de grille plates (nombres premiers), telles que 1x41, ne sont pas incluses dans ces résultats. Avec 45 nœuds sur un commutateur, la performance atteint 350 Mflop/s par nœud. Les mauvais résultats correspondent aux formes de grille très peu carrées (2x17, 2x19).

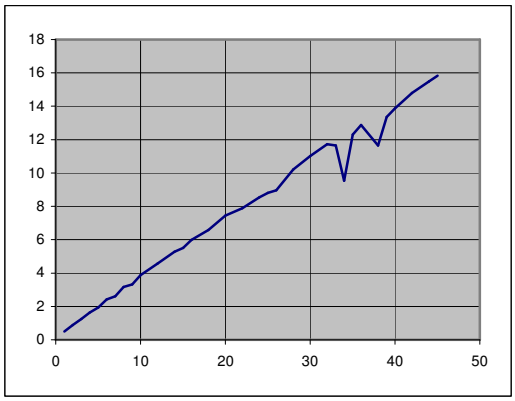

Figure 29 : Performance par taille de grappe

La figure suivante montre que Linpack passe à l'échelle avec plus de machines réparties sur différents commutateurs.

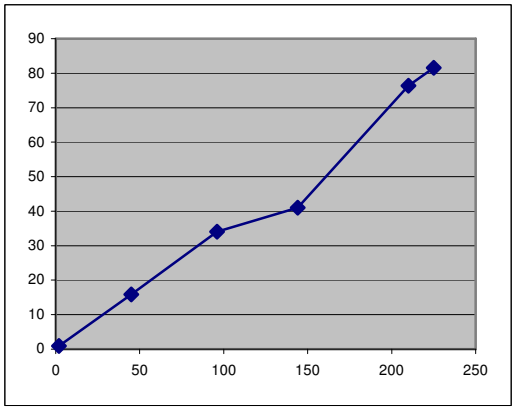

Figure 30 : Performance avec plusieurs commutateurs

3.5.2.9. Etude du rapport performance/prix

La table ci-dessous montre les coûts matériels de la plate-forme I-Cluster. A noter que le logiciel utilisé est *Open Source*, et nous n'avons donc eu aucune dépense pour le logiciel.

Composants	**Coût ($)**
225 *HP* e-Vectra @$950	213750
5 *HP* ProCurve switch @$3300	16500
25 *HP* ProCurve expansion boards @$700	17500
Total	**247750**

Table 4 : Coûts matériel de la plate-forme I-Cluster (Avril. 2001)

Le ratio performance/prix de la plate-forme d'expérimentation I-Cluster est donc environ de *$3000* par Gflop/s. Ce ratio est variable en fonction du nombre

de nœuds par commutateur, mais reste bon quelle que soit la taille de la grappe, comme le montre la Figure 31 ci-dessous.

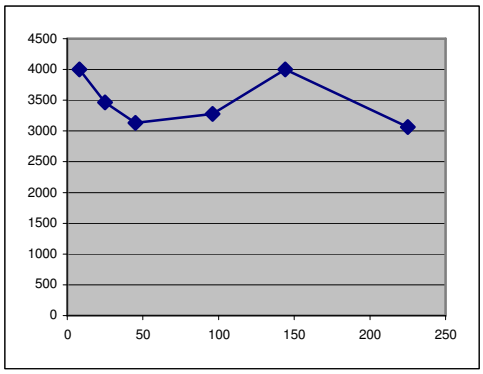

Figure 31: Prix/Gflop/s en fonction du nombre de nœuds

Cette série d'expériences nous a permis de soumettre nos résultats au TOP500 [MSD+01]. En mai 2001, la plate-forme d'expérimentation I-Cluster est donc apparue à 81.6 Gflop/s comme le 385ème superordinateur au monde, et le 15ème en France.

Plus remarquable est le fait que nous ayons été les premiers à entrer au TOP500 avec du matériel de grande consommation (PC monoprocesseur standard, non modifiés, réseau Ethernet).

Ceci montre et valide donc le fait que l'utilisation de machines inutilisées d'un intranet pour faire du calcul intensif est possible, et permet de très hauts niveaux de performance. Nos expériences montrent aussi l'importance pour une grappe de calcul d'avoir une bonne homogénéité en termes de capacité matérielle entre les machines; l'importance de la latence du réseau; et l'importance de l'adéquation entre le travail à exécuter et le nombre de machines à utiliser, ce nombre devant rester le plus petit possible.

3.6. Mesure de l'hétérogénéité et de la disponibilité sur un cas réel

Notre objectif étant d'exploiter les machines inutilisées d'un intranet et de les agréger en grappes virtuelles, il apparaît important d'évaluer un environnement réel en termes de parc de machines et de mesurer quelle disponibilité ont celles-ci. L'environnement étudié ici consiste en la totalité des machines de bureau de **l'intranet mondial de Hewlett-Packard**. L'étude a porté sur un volume de plusieurs centaines de milliers de machines. Le service de maintenance des PC de bureau Hewlett-Packard dispose d'outils de traçage permettant la gestion logicielle et matérielle de la plupart des PC du parc. L'inventaire exhaustif de chaque machine est fait de manière régulière. De nombreux paramètres sont instrumentés, tels que la capacité mémoire, la vitesse processeur, la taille de disque dur, le type de connectivité réseau, le sous-réseau

de la machine. D'autre part, une instrumentation des logiciels installés est faite, mais nous ne la considérerons pas dans cette étude.

L'expérience sa été réalisée entre septembre et décembre 2002. Environ 260 000 machines ont été analysées, cependant seules 111 043 machines ont pu être complètement instrumentées, les autres machines ne disposant pas d'un agent d'instrumentation SNMP, DMI ni WMI.

Dans l'objectif d'évaluer la capacité de chacune des machines d'être allouée à une grappe virtuelle, l'analyse des capacités matérielle des machines peut être caractérisée de la manière suivante :

- processeur disponible, nombre et fréquence d'horloge,
- taille de disque dur,
- capacité mémoire,
- type de machine,
- système d'exploitation,
- capacités de communication,
- sous-réseau.

Nous nous sommes limités pour cette étude aux machines de bureau disposant d'une architecture compatible IBM PC, en ignorant donc les serveurs et les calculateurs, qui ne sont généralement pas utilisés par une personne unique et dont les périodes de jachère ne sont donc pas exploitables.

La Table 5 ci-dessous montre la répartition des machines par type de processeur. Sur notre échantillon de machines 55 % d'entre elles sont de type Pentium® III. Nous pouvons donc raisonnablement nous attendre à pouvoir disposer facilement de quelques machines de type pour pouvoir les agréger en grappes virtuelles homogènes.

Processeur	**Nombre de machines**	**%**
AMD® Athlon™	140	0.13
Cyrix®	3	0.0
Intel® Celeron™	661	0.6
Intel® Pentium®	8094	7.29
Intel® Pentium® II	31072	27.98
Intel® Pentium® III	62029	55.86
Intel® Pentium® 4	4530	4.08
Inconnu	4514	4.07
Total	**111043**	**100**

Table 5 : Types de processeurs

La Table 6 montre que les fréquences des processeurs disponibles sont assez élevées, avec une moyenne de 690 MHz. La distribution est assez uniforme autour de cette moyenne, et nous n'aurons donc pas de problème à trouver des machines rapides lors de la recherche de grappes virtuelles.

Fréquence (MHz)	**Nombre de machines**	**%**
< 265	5142	4.63
266-532	44086	39.7
533-797	24119	21.72
798-1063	22269	20.05
1064-1329	3457	3.11
1330-1595	423	0.38
> 1596	8986	8.09
Inconnue	2561	2.31
Total	**111043**	**100**

Table 6 : Fréquences d'horloge

La Table 7 nous montre que nous pouvons aussi compter sur un espace de stockage de 1 MO sur 99% des machines. Dans l'implémentation actuelle de notre plate-forme, ceci nous suffit pour installer le système I-Cluster et disposer d'une espace de stockage de 512 Mo pour les travaux de calcul en mode cluster. Nous n'aurons donc pas de limitation en termes de place disque sur un parc de machines tel que celui de Hewlett-Packard.

Stockage (Mo)	**Nombre de machines**	**%**
<1	41	0.04
1-2	539	0.49
2-4	3923	3.53
4-8	22077	19.88
8-16	33241	29.94
16-32	35873	32.31
32-64	6982	6.29
>64	310	0.28
Inconnue	8057	7.26
Total	**111043**	**100**

Table 7 : Espace de stockage

L'analyse de la Table 8 nous apporte une mauvaise nouvelle : Sur les 111 043 machines de notre échantillon, 110 sont des serveurs, que l'on ne pourra donc pas utiliser en mode cluster pour ne pas perturber leur exploitation, et

55 809 machines, soit environ 50% du parc, sont des portables. L'utilisation des jachères d'utilisation des portables n'est pas aussi prometteuse que de celles des machines de bureau, car les utilisateurs vont débrancher leur machine la nuit et les emporter chez eux dans la plupart des cas.

Type	**Nombre de machines**	**%**
Machine de bureau	47151	42.46
Serveur	110	0.1
Portable	55809	50.26
Inconnu	7973	7.18
Total	**111043**	**100**

Table 8 : Type de machine

Le système d'exploitation de la totalité des machines analysées est Microsoft Windows®, avec plus de 92 % des machines sur une souche Windows ® NT™ (NT4, 2000 ou XP) et un peu moins de 8 % des machines sur une souche Windows 9x (95, 98 ou ME).

Il est à noter que l'échantillon analysé chez Hewlett-Packard reflète une politique d'entreprise où Windows® est imposé sur les machines de bureau. L'échantillon est cependant valable car des travaux récents [GKS+01] font état de 96 à 99 % des machines de bureau équipées avec Windows.

La Figure 32 montre que la topologie du réseau HP est telle que de nombreuses machines sont regroupées dans chaque sous-réseau. Ceci nous indique qu'il sera facile de trouver une grappe virtuelle de machines qui pourront exécuter des travaux impliquant des communications intensives.

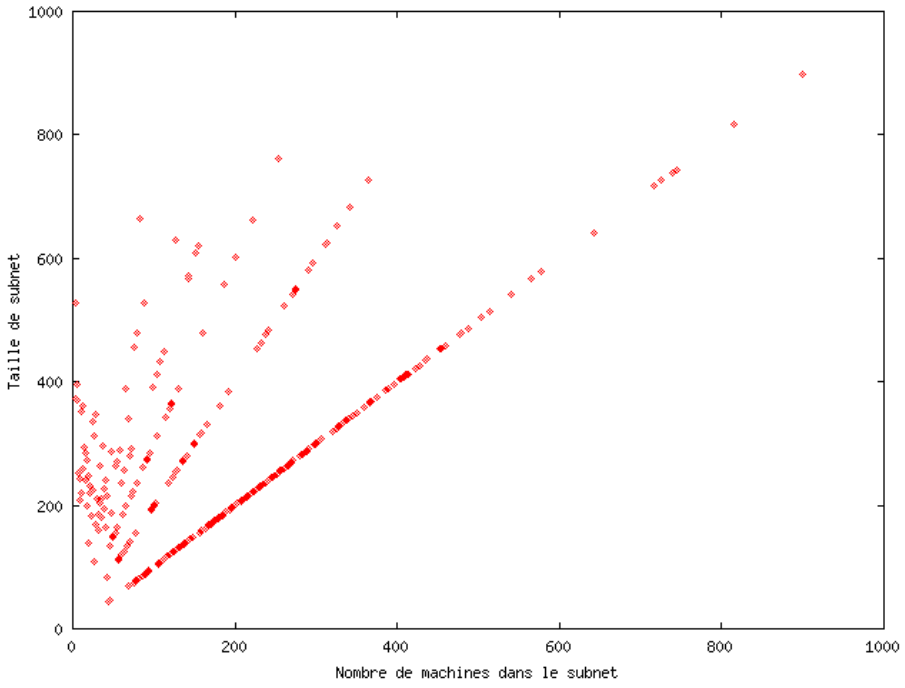

Figure 32 : Nombre de machines par taille de sous-réseau

Il est à noter que HP étant une entreprise mondiale, il est possible de trouver des machines disponibles à toute heure, même pendant la journée. Par exemple, depuis la France on peut exploiter les machines situées en Amérique jusqu'au début de l'après-midi car leurs utilisateurs sont au lit. Et on peut ensuite se rabattre sur les machines situées en Asie-Pacifique.

Profil d'utilisation. Il aurait fallu pouvoir étudier la disponibilité des machines et définir leur profil afin de déterminer les plages horaires auxquelles une machine peut être requise pour faire partie d'une grappe virtuelle. Cette étude n'a pas pu être faite, sauf sur quelques machines de tests. Les données intéressantes auraient été la détermination des moments auxquels chaque machine appartenant à un utilisateur était dans un mode inactif. Cependant, une telle mesure étant considérée comme une violation de la vie privée de l'utilisateur par la politique de HP, nous n'avons pas pu collecter d'informations réelles sur ce sujet. On peut toutefois aisément faire une projection de l'utilisation de machines : Un utilisateur va généralement utiliser sa machine exclusivement pendant ses heures de travail, soit grossièrement du lundi au vendredi, de 8h00 à 20h00. Ce qui nous donne un taux d'utilisation des machines de moins de 40 %. De plus, de nombreuses personnes travaillent à temps partiel, et chacun dispose de jours d'absence pour congés, déplacement professionnel ou maladie, ce qui étend encore la plage d'utilisation possible des machines.

Cette étude nous permet de conclure qu'un intranet typique tel que celui de Hewlett-Packard, les 47000 machines de bureau (nous ne comptons pas les machines portables) sont utilisables dans une proportion évaluée à 50 % -- car nous n'exploitons pas les périodes d'inactivité inférieures à quelques heures-- et donc environ 25000 machines sont disponibles pour être allouées à des calculs intensifs. Nous avons vu que les capacités moyennes des machines sont bonnes, et que leur bon regroupement par sous-réseau est favorable à l'exécution de travaux de calcul intensif sur grappes virtuelles.

Ceci justifie donc l’intérêt d’exploiter les machines en jachère sur un intranet.

4. Conclusion

L'étude réalisée ici pour le projet I-Cluster visait à l'étude de la possibilité d'exploiter les machines d'un intranet d'entreprise de plusieurs milliers de machines, et de coordonner leurs ressources partagées afin d'utiliser leurs instants d'inactivité pour pouvoir faire émerger les *grappes virtuelles* disponibles sur l'intranet.

Cette étude comportait aussi quelques contraintes telles que la nécessité de pouvoir s'adapter –de manière transparente et invisible à l'utilisateur-- à l'environnement Windows des possesseurs des machines, alors que l'aspect calcul haute performance devait être basé sur les outils traditionnels de gestion et d'exploitation de grappes de calcul, et en particulier sous Linux. De même l'automatisation et la transparence des outils est importante afin de pouvoir facilement gagner l'acceptation des utilisateurs et ainsi optimiser l'exploitation des machines disponibles sur l'intranet.

Une autre contrainte était de pouvoir fonctionner en mode pair-à-pair, offrant un passage à l'échelle massif de l'infrastructure ainsi que des capacités d'auto-organisation et d'adaptation aux variations de l'environnement, sans toutefois imposer une charge importante sur les machines utilisées ni sur le réseau d'interconnexion.

L'architecture développée est assez nouvelle, se basant sur des outils standard tels que Condor, Globus, MOSIX ou JXTA et offrant deux composants principaux :

- d'une part sur un changement de personnalité des machines entre un *mode utilisateur* sous Windows et un *mode grappe* sous Linux, permettant une bonne isolation du code entre les deux modes, ainsi qu'une pleine disponibilité des machines lorsqu'elles sont en mode grappe (performance et simplicité de gestion), ce composant ayant atteint une excellente qualité logicielle, conforme aux critères commerciaux de Hewlett-Packard,
- d'autre part un méta-annuaire, basé sur une base de données distribuée en mode pair-à-pair, qui permet à tout instant de connaître l'état de la communauté des machines et de trouver les machines ayant les caractéristiques nécessaires à l'exécution d'un calcul

parallèle donné. Les caractéristiques matérielles étant elles-mêmes instrumentées à l'aide d'un composant que nous avons développé pour les besoins du projet.

Un composant permettant de faire une sélection efficace des machines à allouer à un travail demandé a été développé (le *Match Finder*). Ce composant est cependant trop primaire pour fonctionner dans des conditions réelles d'exploitation, impliquant des travaux parallèles nécessitant une bonne homogénéité des machines allouées ainsi qu'une faible latence d'interconnexion. D'autre part, il sera nécessaire d'étudier la faisabilité d'un système de soumission disposant de files d'attente, de manière à pouvoir placer des travaux dans la journée qui ne seront exécutés que la nuit, lorsqu'un grand nombre de machines sont disponibles.

Après une phase de mesure des capacités réelles d'une grappe basée sur des machines communes d'entreprise interconnectées par un réseau standard, nous avons pu établir que la puissance disponible était potentiellement élevée

L'analyse des machines de l'intranet de Hewlett-Packard a aussi permis de montrer que de nombreuses machines en jachère permettraient à tout instant de la journée de trouver des machines disponibles et de les exploiter pour faire un gros calcul scientifique. Par extension, on peut s'attendre à trouver des machines utiles sur un intranet commun de quelques centaines de machines, et utiliser un grand nombre des machines la nuit et le week-end.

Une étude en cours par Jean-Michel N'Long2 dans le cadre de sa thèse au laboratoire ID-IMAG permettra de mettre en place au-dessus du système I-Cluster des mécanismes de gel/reprise (*Checkpointing*) de manière à ce que les travaux abandonnés suite a l'éventuel retour de l'utilisateur sur sa machine puissent continuer sur d'autres machines sans devoir être abandonnés.

Des études complémentaires au système sont nécessaires, telles qu'une meilleure infrastructure de sécurité (dans l'état actuel on se limite à un droit d'accès au système global, permettant de lancer des travaux), un mécanisme de suivi des travaux pendant l'exécution, une comptabilité des exécutions.

La limitation à un intranet nous permet d'éviter en partie les contraintes de sécurité et garantit certaines propriétés du réseau telles que la routabilité des messages et une relative homogénéité des capacités de communication à l'intérieur du réseau.

Le système I-Cluster est novateur par ses mécanismes mais aussi par son approche, qui permet de le déployer sur un parc de machines existantes sans perturber les possesseurs de machines, en pouvant utiliser les outils classiques

d'exploitation de grappes de calcul, et sans nécessiter de déploiement toujours difficile de serveurs ni d'infrastructure matérielle ou logicielle spécifique ailleurs que sur les machines à exploiter.

Hélas, la fermeture des laboratoires HP Labs Grenoble survenue fin 2002 ne permettra pas de terminer le projet I-Cluster dans son intégralité.

Lors d'une présentation du projet à Myron Livny en octobre 2002, le créateur de Condor m'a expliqué être très intéressé par le nuage I-Cluster et ses possibilités de passage à grande échelle. En effet, lorsqu'un système doit passer à l'échelle avec plus de quelques centaines de participants, la volatilité des machines nécessite de fréquentes écritures dans la base de données centrale de Condor, qui était basée –à cette époque- sur LDAP. LDAP fournit une capacité de montée en charge excellente en lecture, mais ces écritures fréquentes provoquent un écroulement rapide du serveur Condor.

Lors de discussions avec des responsables informatiques de TotalFinaElf en mars 2003, j'ai découvert que, s'il était coûteux pour eux de déployer un nouveau serveur à des fins de contrôler des calculs distribués sur les machines des employés pendant leurs heures creuses, ils craignent encore plus de déployer un agent même très transparent, tel que celui que nous avons développé pour I-Cluster, sur de nombreuses machines de leur intranet. Une telle indication aurait été importante au moment de la conception et de l'architecture du projet, et une solution basée sur un logiciel de calcul distribué par réseau sur chacune des machines participantes et ne modifiant pas du tout le disque dur des machines aurait été préférable. Le projet *Oucapo* de Philippe Augerat et al. a tiré parti de cette indication et permet un calcul distribué sur les machines inexploitées de l'intranet à la manière de I-Cluster, mais le logiciel système est récupéré à l'aide d'un démarrage par réseau plutôt que d'être placé sur le disque dur de manière cachée.

Ces réactions par rapport à des projets réels d'exploitation de machines en jachère sur l'intranet nous ont permis de mieux ressentir le besoin réel d'un système tel que I-Cluster. Aussi, si cette étude devait être démarrée aujourd'hui, la démarche suivie par le projet aurait certainement beaucoup moins mis l'accent sur la nécessité de pouvoir exécuter des travaux parallèles et aurait par ailleurs demandé une meilleure intégration aux mécanismes de grille standard aujourd'hui, et notamment aux interfaces Globus.

Enfin, une évolution importante par rapport au monde d'aujourd'hui : La loi de Moore, observée dès 1965, prédit que la capacité de calcul moyenne des ordinateurs double tous les 18 mois. Même si cette loi est optimiste, elle reflète assez bien l'évolution rapide de la puissance de calcul des machines. On peut donc dire que pour un travail complexe mais borné, une unique machine standard sera suffisante pour résoudre le travail. Par exemple, Météo France est considéré comme un organisme ayant de gros besoins de calculs. Leurs contraintes d'exécution de ce calcul sont qu'il doit se terminer en moins de deux heures, ainsi que de garantir la qualité du service. Or le maillage utilisé dans leur application (à base de formules Navier-Stockes) est limité en grande partie

par le nombre de points de mesure disponibles sur le terrain (les stations météo). A l'aide de la loi de Moore, on peut projeter qu'en 2008 environ, un PC standard sera capable de résoudre leur problème dans le temps imparti. Ceci nous montre encore que, dans le cadre de l'exploitation des machines en jachère disponibles sur les réseaux d'entreprise, il apparaît important de se concentrer non pas sur l'efficacité individuelle de chaque machine, mais plutôt sur la capacité à gérer toutes ces machines. En effet, on aura souvent intérêt (financièrement s'entend) à utiliser un grand nombre de machines simples (disponibles sur l'intranet) plutôt que de concentrer l'exécution sur une seule machine de puissance importante.

Annexe A : Capteur de ressources locales

Afin de pouvoir exploiter efficacement une machine d'un intranet au sein d'une grappe virtuelle, il est nécessaire de pouvoir évaluer son potentiel ainsi que sa disponibilité. Le chapitre 3.2 détaille comment la disponibilité des machines peut être analysée de manière à déterminer les fenêtres de tir, pendant lesquelles les machines seront disponibles pour exécuter des travaux.

Concernant le potentiel de chaque machine, il nous faut déterminer les capacités matérielles de chaque machine, en particulier le nombre et la fréquence des processeurs, l'espace mémoire disponible, l'espace de stockage disponible, la connectivité au réseau et la situation topologique sur l'intranet.

Toutefois, aucun outil n'est disponible pour réaliser cette fonction de manière standard et fiable.

Outil d'instrumentation

Nous avons donc réalisé un outil d'analyse qui, exécuté localement, fournit une carte de ressources (*resource card*) de la machine et qui permettra d'offrir une description de la machine au système global.

L'outil développé est spécifique à Windows, et s'exécute sur tout PC, indifféremment de sa marque ou de son modèle.

La description offerte est conforme au format CIMXML [DWZ01]. Elle se compose donc d'un fichier XML dont l'arbre décrit chaque élément nécessaire. Ce format permet de manipuler un fichier de manière très indépendante de l'outil, et en particulier il est possible d'utiliser une variété de logiciels courants afin de manipuler le fichier, ou bien encore afin de manipuler les fichiers produits par un grand nombre de machines de l'intranet. Les logiciels *HP Toptools* et *HP Asset* m'ont ainsi permis de manipuler les fichiers CIMXML produits par mon outil d'analyse sur un certain nombre de machines (plusieurs centaines) et d'analyser la capacité de l'intranet analysé à offrir des grappes virtuelles homogènes. Les résultats de ces analyses sont disponibles dans le chapitre 3.6.

Analyse du BIOS

L'outil d'analyse des capacités matérielles se base sur une description du matériel fournie par le BIOS des PC. Il s'agit donc d'identifier un arbre de description des ressources en ROM, conforme au format standardisé par

SMBIOS [Dmt95], puis de récupérer les informations nécessaires dans cet arbre. Cette opération est simple mais demande une certaine finesse dans le codage afin de ne pas perturber le système d'exploitation lors de l'accès direct à la mémoire.

Il est donc aisé de reconnaître les informations sur les éléments dépendant directement de la carte mère de la machine, à savoir :

- nombre, type et fréquence des processeurs,
- nombre de niveaux de mémoire cache ainsi que leur taille,
- taille de mémoire de la machine. Il est possible aussi de reporter le nombre de barrettes mémoire et la technologie de mémoire employée, mais je ne me suis pas intéressé à ces possibilités qui vont au-delà du besoin de détection de grappes virtuelles homogènes.

La détection des informations se fait par scrutation de la zone de mémoire morte de e000:0 à ffff:0 (adresse physique non virtualisée), qui contient une ancre spécifique « _SM_ ». Cette ancre identifie le point d'entrée permettant d'accéder aux tables SMBIOS. Le morceau de code ci-dessous explicite la structure réelle que nous allons trouver en mémoire :

```
typedef struct
    {
    unsigned long Anchor;       /* must be _SM_ */
    unsigned char Checksum;     /* structure checksum */
    unsigned char Length;       /* structure length (1Eh) */
    unsigned char MajorVersion; /* 02 */
    unsigned char MinorVersion; /* 01 */
    unsigned short MaxSize;     /* largest structure size in bytes */
    unsigned char Revision;     /* currently 0 */
    unsigned char Reserved[5];  /* Should be set to all 00s */
    /* Now are entries for legacy compatibility */
    unsigned char IntAnch[5];   /* Intermediate Anchor string _DMI_ */
    unsigned char IntChecksum;  /* Intermediate checksum bytes 10h-1Fh */
    unsigned short StrucLength; /* SMBIOS structure table length */
    unsigned long StrucAddress; /* 32 bits physical address of the table */
    unsigned short StrucNum;    /* number of SMBIOS structures */
    unsigned char BCDRevision;  /* 21h for SMBIOS 2.1 */
    } SMBIOS_HEAD;
```

Une fois le point d'entrée SMBIOS trouvé nous allons trouver en mémoire une liste chaînée dont chaque maillon contiendra un élément d'information spécifique sur un composant de la machine.

```
Typedef struct
    {
    unsigned char Type;         /* Should be 4 for processor */
    unsigned char Size;         /* Structure size */
    unsigned short Handle;      /* Holds next entry */
    SMSTRING Designation;       /* Socket designation */
    unsigned char Type;         /* Processor type */
    unsigned char Family;       /* Processor family */
    SMSTRING Manufacturer;      /* Processor manufacturer */
    unsigned long ID;           /* Processor ID */
    SMSTRING Version;           /* Processor version */
    unsigned char Voltage;      /* Try to guess… */
    unsigned short Clock;       /* External Frequency */
    unsigned short MaxSpeed;    /* Maximum speed */
    unsigned short CurSpeed;    /* Current speed */
    unsigned char Status;       /* Processor present? */
    unsigned short Upgrade;     /* Possible upgrades */
    } PROCESSOR_ENTRY;
```

```
GetManufacturerProcessor();
    char *GetDescriptionProcessor();
    char *GetFamilyProcessor();
    char *GetCurrentClockSpeedProcessor();
    char *GetMaxClockSpeedProcessor();
```

Cet exemple nous montre la finesse des informations qu'il est possible de récupérer. Dans la pratique, nous sous limitons au nombre de processeurs, leur type et leur fréquence, ce qui donne une information satisfaisante sur la puissance de calcul de la machine. Toutefois, nous ne gérons pas l'information sur les caches internes des processeurs, qui permettraient d'affiner encore notre description.

De la même manière, nous pouvons récupérer l'information nécessaire sur les barrettes de mémoire centrale et de mémoire cache externes.

A titre d'exemple, voici un extrait du fichier CIMXML résultant de l'analyse d'une machine par notre outil d'instrumentation, qui fournit l'information disponible sur le processeur. Nous avons ici une machine équipée d'un Pentium III à 700 MHz.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CIM VERSION="2.0" DTDVERSION="2.0">
  <MESSAGE ID="20021202155525" PROTOCOLVERSION="1.0">
    <MULTIRSP>
        <IMETHODRESPONSE NAME="EnumerateInstances">
          <IRETURNVALUE>
            <VALUE.NAMEDOBJECT>
              <INSTANCENAME CLASSNAME="CIM_Processor">
                <KEYBINDING NAME="CIM_Processor">
                  <KEYVALUE>20021202155525-060</KEYVALUE>
                </KEYBINDING>
              </INSTANCENAME>
              <INSTANCE CLASSNAME="CIM_Processor">
                <QUALIFIER NAME="ModTime" TYPE="datetime">
                  <VALUE>20021202155525.267000-060</VALUE>
                </QUALIFIER>
                <PROPERTY NAME="Description" TYPE="string">
                  <VALUE>Pentium(R) III</VALUE>
                </PROPERTY>
                <PROPERTY NAME="CurrentClockSpeed" TYPE="uint32">
                  <VALUE>700</VALUE>
                </PROPERTY>
```

Par contre, l'analyse du SMBIOS ne fournit pas d'informations sur les capacités réseau ni sur les périphériques de stockage de la machine. Il nous faut donc récupérer ces informations à partir du système d'exploitation.

Analyse du système d'exploitation

Cette partie de l'outil d'analyse de configuration a été développée par une équipe de HP Brésil, en collaboration avec le laboratoire CPAD dirigé par César de Rose. L'instrumentation des disques se base pour une part sur les disques physiques de la machine et d'autre part sur ses disques logiques (partitions). Cet outil a été très difficile à réaliser car nous voulions pouvoir supporter une

machine quelconque d'un intranet, donc équipée de disques IDE ou SCSI, avec partitionnements multi-disques, les disques externes et aussi les clefs de stockage USB. Ce besoin de fiabilité s'est ressenti au niveau de la difficulté de mise au point de l'outil qui a été faite par deux ingénieurs au Brésil, assistés par un ingénieur de qualité et de test.

Stockage de masse

Notre instrumentation des disques durs est assez peu élégante, mais a le mérite de fonctionner dans toutes les configurations de disques durs, même si ceux-ci se composent de disques RAID, de disques USB, SCSI ou encore de disques comportant une interface propriétaire.

Le fonctionnement de l'outil est simple, et consiste en l'énumération exhaustive de tous les disques physiques présents sur la machine (17 disques peuvent être supportés). Pour chaque disque trouvé, nous ouvrons une connexion vers son gestionnaire de périphérique, qui va ensuite nous donner les caractéristiques physiques du média, c'est-à-dire son type (disquette, disque dur, CD-ROM, disque RAM ou disque éjectable) et sa géométrie en termes de nombres de cylindres, têtes, et secteurs.

Le résultat de cette analyse est donc un tableau détaillant les caractéristiques de l'ensemble des disques physiques disponibles sur la machine.

Nous construisons ensuite une table de disques logiques Windows (A:, C:, D: etc…) que nous lions à la première table. La liaison entre les deux tables n'est pas bijective. En effet, un disque dur partitionné en deux n'aura qu'une entrée dans la table des disques physiques, mais deux entrées dans la table des disques logiques. Au contraire, dans le cas de deux disques durs en RAID-0, deux disques physiques ne seront représentés que comme un seul disque logique.

Topologie réseau

Pour ce qui est de l'instrumentation du réseau, le problème est différent : il est relativement facile de détecter et d'instrumenter une carte Ethernet sur une machine, mais comment instrumenter la situation topologique d'une machine sur l'intranet ? Comme nous en verrons l'analyse plus loin, il nous suffit de connaître le sous-réseau de chaque machine ainsi que sa technologie d'interconnexion (répéteurs ou commutateurs) pour pouvoir identifier des grappes virtuelles efficaces.

Nous avons donc défini de nouvelles classes étendant le standard CIMXML, qui nous ont permis de décrire cette technologie d'interconnexion, le sous-réseau de chaque machine et les paramètres réseau de base. L'instrumentation de ces classes se fait grâce à l'utilisation des services du système d'exploitation.

La récupération des informations relatives à la connectivité réseau se fait de la même manière que pour les unités de stockage de masse : Une énumération des points de connectivité TCP-IP est faite à l'aide de la base de registres de Windows. Pour chaque point de connectivité nous allons ensuite récupérer les informations disponibles auprès de Windows concernant le type de connectivité (vitesse, filaire, infra rouge, modem ou hertzien), puis nous allons ignorer les

connections en accès indirect par PPP (*Point to Point Protocol*), qui ne nous permettrons pas de toutes façons d'exécuter des travaux parallèles.

Sur chaque point de connectivité nous aurons donc les paramètres de connectivité TCP-IP, la bande passante maximale disponible ainsi que le type de lien.

Cas de PocketWindows, de Chaï et de Linux

Notre objectif a longtemps été de cibler les divers appareils connectés au réseau (*Internet appliances*) dans le cadre de l'étude I-Cluster, et non pas de se limiter à la seule exploitation des PC inexploités de l'intranet. En particulier, l'utilisation des imprimantes pour effectuer des calculs parallèles pendant leurs périodes d'inactivité a longtemps été considérée. Nous avons donc étudié comment il serait possible d'instrumenter les ressources physiques disponibles depuis des imprimantes, des assistants de poche (PDA) et autres terminaux de visualisation (*X terminals*).

Les PDA sont équipés de PocketWindows, un système d'exploitation assez primitif développé par Microsoft. Nous avons porté notre outil d'instrumentation sur PocketWindows afin de pouvoir faire participer les PDA inexploités à des calculs parallèles (on peut rêver). Ce logiciel, qui n'a été développé que pour les versions de PocketWindows tournant sur les processeurs ARM, ne fournit qu'un sous-ensemble des informations disponibles sur PC ; à savoir, le type et la vitesse du processeur, la taille de mémoire centrale (partagée entre mémoire vive et mémoire de stockage) et la connectivité disponible (généralement en réseau sans fil Ethernet 802.11b). Le composant développé pour PocketWindows avec ces fonctionnalités réduites fonctionne bien.

De la même manière, certains PDA sont équipés de Linux, et nous avons aussi porté notre outil d'instrumentation sous Linux. Cependant, des choix non techniques nous ont empêchés de terminer l'outil, qui n'instrumente que la mémoire central et le processeur à ce jour.

Dans le cas des imprimantes, les séries HP LaserJet 4500 et suivantes sont équipées en standard de Chaï, une machine virtuelle Java embarquée. Cette machine virtuelle nous permet de lancer des applet Java et donc de contrôler l'exécution d'un programme parallèle s'exécutant sur l'imprimante à ses heures perdues. Chaï nous offre une API d'instrumentation qui permet de récupérer les informations de description des ressources matérielles disponibles sans avoir à écrire de code d'instrumentation, et donc sans avoir à se lier à une architecture spécifique. Malheureusement là encore, des impératifs de projet ne nous ont pas permis de continuer à travailler sur Chaï et l'outil correspondant n'a pas été développé.

Références

[AAB+01] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Sagi, Z. Shi, S. Vadhiyar, "User's guide to NetSolve v1.4", Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, USA, 2001.

[ABD+02] Augerat P., Billot W., Derr S., Martin C., "A scalable file distribution and operating system installation toolkit for clusters", Proceedings of CCgrid 2002.

[ACP95] T. E. Anderson, D. Culler, D. Patterson, "A case for NOW (Networks of Workstations)", IEEE Micro, Feb 1995.

[ADa00] Abis M., Dayley B., "An Open Alternative to Java and C-Sharp", http://ivm.sourceforge.net/, 2000.

[ADV+94] R.H. Arpaci, A. Dusseau, A.M. Vahdat, L.T. Liu, T.E. Anderson, D.A. Patterson, "The Interaction of Parallel and Sequential Workloads on a Network of Workstations", UC Berkeley Technical Report CS-94-838, November, Also Submitted to Sigmetrics '95, 1994.

[AGH94] Arnold K., Gosling J., Holmes D., "The Java™ Programming Language", The Java™ Series, Addison Wesley, 1994.

[AGL+94] D. Atkins, M. Graff, A. K. Lenstra and P. C. Leyland., "The Magic Words are Squeamish Ossifrage", Advances in Cryptology - ASIACRYPT '94, J. Pieprzyk and R. Safavi-Naini, eds., Lecture Notes in Computer Science 917, (1995), 263-277, 1994.

[AGO97] P. Arbenz, W. Gander, M. Oettli, "The Remote Computational System", Parallel Computing, 23(10):1421-1428, 1997

[AIS+97] A.D. Alexandrov, M. Ibel, K. E. Schauser and C. J. Scheiman, "SuperWeb: Research Issues in Java-Based Global Computing", Concurrency: Practice and Experience, June 1997.

[AIS+97b] A.D. Alexandrov, M. Ibel, K. E. Schauser and C. J. Scheiman, "SuperWeb: Towards a Global Web-Based Parallel Computing Infrastructure", 11th International Parallel Processing Symposium (IPPS'97), Geneva, April 1997.

[AKS98] A. Alexandrov, P. Kmiec, K. Schauser, "Consh: A confined execution environment for internet computations", Proceedings of the USENIX Annual Technical Conference, 1998.

[AMS02] Augerat P., Martin C., Stein B., "Scalable monitoring and configuration tools for grids and clusters", Proceedings of the Tenth Euromicro Workshop on Parallel, Distributed and Network-based Processing, 2002.

[And+99] Anderson D., et al., "SETI@home: The Search for Extraterrestrial Intelligence", Technical report, Space Sciences Laboratory, University of California at Berkeley. http://setiathome.ssl.berkeley.edu/, 1999.

[And02] D.P. Anderson et al., "SETI@home: An Experiment in Public-Resource Computing", Communications of the ACM, pp. 56-61, Nov. 2002.

[ARa00] A. Acharya, M. Raje, "Mapbox: Using parameterized behavior classes to confine applications", In Proceedings of the USENIX Security Symposium, pages 1--17, Denver, CO, August 2000.

[Axt00] R. Axtell, " Effects of Interaction Topology and Activation Regime in Several Multi-Agent Systems", Proceedings of Multi-Agent-Based Simulation, Second International Workshop, MABS 2000, Boston, MA, USA, 2000.

[ARG89] Abrossimov V., Rozier M., Gien M., "Virtual Memory Management in Chorus", Proceedings of Progress in Distributed Operating Systems and Distributed Systems anagement, 1989.

[BBC+02] G. Bosilca, A. Bouteillier, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, A. Selhikov, "MPICH-V: Parallel Computing on P2P Systems", International Conference on SuperComputing SC'2002, Baltimore USA, 2002.

[BCF+95] N. Boden, R. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, Wen-King Su, "Myrinet: A Gigabit-per-Second Local-Area Network", IEEE Micro, vol. 15 (1) pp 29-36, 1995.

[BDV+98] F. Breg, S. Diwan, J. Villacis, J. Balasubramanian, E. Akman, D. Gannon, " Java RMI Performance and Object Model Interoperability: Experiments with Java/HPC++", Proceedings of the ACM Workshop on Java for High-Performance Network Computing, Concurrency: Practice and Experience (vol 10/no 11-13), John Wiley & Sons. September 1998.

[BFo98] M. Baker, G. Fox, "Metacomputing: Harnessing Informal Supercomputers", Portsmouth University preprint, 1998.

[BGr01] Gordon Bell, Jim Gray, "High Performance Computing: Crays, Clusters, and Centers", Microsoft Research Report MSR-TR-2001-76, 2001.

[BGW93] Barak A., Guday S., Wheeler R.G., "The MOSIX Distributed Operating System, Load Balancing for UNIX", Lecture Notes in Computer Science, Vol. 672, Springer-Verlag, 1993.

[BFH+02] G. Bosilca, G. Fedak, T. Herault et F. Magniette, "Evaluation de performance de différentes techniques de confinement d'exécution

pour le calcul Pair-à-Pair" Renpar'13, Hammamet, Tunisie, Avril 2002.

[BKK+96] Arash Baratloo, Mehmet Karaul, Zvi Kedem and Peter Wyckoff, "Charlotte: Metacomputing on the Web", In Proc. of the 9th International Conference on Parallel and Distributed Computing Systems, September 1996.

[BSS+95] Becker D.J., Sterling T., Savarese D., Dorband J.E., Ranawak U.A., Packer C.V., "Beowulf: A Parallel Workstation for Scientific Computation", Proceedings of the International Conference on Parallel Processing, 1995.

[Buy99] Rajkumar Buyya, "High Performance Cluster Computing, Volume 1: Architectures and Systems", Prentice Hall, 1999.

[BWC+03] F. Berman, R. Wolski, H. Casanova, et al., "Adaptive Computing on the Grid using AppLeS", IEEE transactions on Parallel and Distributed Systems, Vol. 14, No. 4, 2003.

[BWh89] Barak A., Wheeler R., "MOSIX: An Integrated Multiprocessor UNIX", Proceedings of Winter 89 USENIX Conference, pp.101-112, San Diego, CA., 1989

[CAS86] F. Cristian, H. Aghili, R. Strong, "Clock Synchronization in the Presence of Omissions and Performance Faults, and Processor Joins", 16th International Symposium on Fault Tolerant Computing Systems, 1986.

[CCi97] G. Chiola, G. Ciaccio, "GAMMA: a Low-cost Network of Workstations Based on Active Messages", in proceedings of PDP'97 (5th EUROMICRO workshop on Parallel and Distributed Processing), London, UK, January 1997.

[CCI+97] Peter Cappello, Bernd Christiannsen, Mihai F. Ionescu, Michael O,Neary, Klaus E. Schauser and Daniel Wu, "Javelin: Internet-Based Parallel Computing Using Java", June 1997.

[Cde03] Eddy Caron, Frédéric Desprez, Franck Petit, Vincent Villain, "A Hierarchical Resource Reservation Algorithm for Network Enabled Servers",Proceedings of the 17th International Parallel and Distributed Processing Symposium, Nice - France, April 2003.

[CDo95] Casanova H., Dongarra J.J., "NetSolve: A network server for solving computational science problems", Tech. Report CS-95-313, University of Tennessee, 1995.

[CFF+01] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, "Grid Information Services for Distributed Resource Sharing", Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[CGe89] Nicholas Carriero, David Gelernter, "Linda in Context", Communications of the ACM 32(4): 444-458, 1989

[CSW+01] I. Clarke, O. Sandberg, B. Wiley, T.W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System

in Designing Privacy Enhancing Technologies". International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, Springer: New York, 2001.

[CTo96] T. D. Chandra, S. Toueg, "Unreliable failure detectors for reliable distributed systems", Journal of the ACM 43(2):225--267, March 1996.

[DBM+79] J. Dongarra, J. Bunch, C. Moler, G. W. Stewart, "LINPACK Users Guide", SIAM, Philadelphia, PA, 1979.

[DBM+02] C. De Rose, F. Blanco, N. Maillard, K. Saikoski, R. Novaes, O. Richard, B. Richard, "The Virtual Cluster: a Dynamic Environment for Exploitation of Idle Network Resources", Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2002), Vitória, Brazil, 2002.

[Dik00] J. Dike, "A user-mode port of the Linux kernel", In Proceedings of the USENIX annual Linux Showcase and Conference, Atlanta, 2000.

[DKo98] Y. Duan, P. Kollman, "1 microsecond molecular dynamics simulation of the villin headpiece" Science 282, 740-744, 1998.

[DMa00] H. G. Dietz, T. I. Mattox, "KLAT2's flat neighborhood network", In Proceedings of the Extreme Linux track in the 4th Annual Linux Showcase, Atlanta, GA, October 2000.

[Dmt95] DMTF Consortium, "System Management BIOS (SMBIOS) Specification", http://www.dmtf.org/, 1995.

[DOK+91] F. Douglis, J.K. Ousterhout, M.F. Kaashoek, A.S. Tanenbaum, "A Comparison of Two Distributed Systems: Amoeba and Sprite", Computing Systems, pp 353-384, 1991.

[Dou90] F. Douglis, "Transparent process migration in the Sprite operating system", Ph.D. Thesis, University of California, Berkeley, CA 94720, 1990.

[Dro98] C. De Rose, "Distributed Processor Management in Multicomputers", PhD Thesis, University of Karlsruhe, Germany, 1998.

[DRo01] Druschel, Rowstron, "Past" IEEE Workshop on hot Topics in Operating Systems, 2001.

[Dun+98] D. dunning et al., "The Virtual Interface Architecture", IEEE Micro, vol. 18 (2), pp. 66-75, 1998.

[DWZ01] A. DeVos, S.E. Widergren, J. Zhu, "XML for CIM Model Exchange", IEEE conference for Power Industry Computer Applications (PICA 2001), Sydney, Australia, 2001.

[EBC01] Edwards N., Berger J., Choo T.H., "A Secure Linux Platform", Proceedings of the 5th Annual Linux Showcase and conference, 2001.

[FDF03] R. Figueiredo, P. Dinda, J. Fortes, "A Case for Grid Computing on Virtual Machines" Proceedings of IEEE ICDCS, 2003.

[Fed02] Gilles Fedak, "XtremWeb: A Peer-to-Peer Global Computing experimental platform", FOSDEM Free Software and Open Source Developers Meeting, Brussels, 2002.

[FGN+01] Gilles Fedak, Cecile Germain, Vincent Neri, Franck Cappello, "XtremWeb: an experimental platform for Global and Peer-to-Peer Computing", HEPiX 2001, UNIX users in the High Energy Physics, Orsay, France, 2001.

[FKe96] Foster I., Kesselman C., "Globus: A Metacomputing Infrastructure Toolkit", Proceedings of the Workshop on Environments and Tools for Parallel Scientific Computing, SIAM, Lyon, France, 1996

[FLP85] M. Fischer, N. Lynch, M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process", Journal of the ACM, 32:374-382, 1985.

[FMM00] M.C. Ferris, M.P. Mesnier, J.J. Mori, "NEOS and Condor: Solving Optimization Problems over the Internet", ACM Transactions on Mathematical Software, 26(1):1-18, 2000.

[Fos99] I. Foster, "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers Inc., San Francisco, California, 1999.

[Fos02] I. Foster et al., "Grid Services for Distributed Systems Integration", Computer, pp. 37-46, June 2002.

[GFK+99] Grimshaw A.S., Ferrari A., Knabe F., Humphrey M., "Legion: An operating system for wide-area computing", 1999

[GKa92] Gelertner D., Kaminsky D., "Supercomputing out of recycled garbage: preliminary experience with Piranha", proceedings of the ACM Conference on Supercomputing, 1992.

[GKS+01] Gillen A., Kusnetzky D., Sayama A., Dayal H., Hingley M., "Windows Operating Environment Forecast and Analysis", IDC publisher, document 24827, 2001.

[GNF+00] Germain C., Néri V., Fedak G., Cappello F., "XtremWeb: building an experimental platform for Global Computing", Proceedings of Grid2000, 2000.

[Gnu01] "The Gnutella Protocol Specification". Online. http://dss.clip2.com/GnutellaProtocol04.pdf, 2001

[Gon01] L. Gong, "JXTA: A Network Programming Environment", IEEE Internet Computing, pp. 88-95, May 2001.

[Gus92] D. B. Gustavson, "The Scalable Coherent Interface and Related Standards Projects", IEEE Micro, 12(1):10-22, 1992.

[GWT+96] I. Goldberg, D. Wagner, R. Thomas, E.A. Brewer, "A secure environment for untrusted helper applications: confining the wily hacker", In Sixth USENIX Security Symposium, Focusing on Applications of Cryptography, 1996.

[GWu+97] Grimshaw A.S., Wulf W.A., et al., "The Legion Vision of a Worldwide Virtual Computer", Communications of the ACM, Volume 40, Number 1, Pages 39--45, 1997.

[Haw98] Hawblitzel C., Von Eicken T., "A case for language-based protection", Tech. Rep. 98-1670, Department of Computer Science, Cornell University, 1998.

[HSm95] T.A. Howes, M. Smith, "A Scalable, Deployable Directory Service Framework for the Internet", Technical report, Center for Information Technology Integration, University of Michigan, 1995.

[HTw96] Henderson R., Tweten D. "Portable Batch System: External reference specification", Technical report, NASA, Ames Research Center, 1996.

[HVo98] Hawblitzel C., Von Eicken T., "A case for language-based protection", Tech. Rep. 98-1670, Department of Computer Science, Cornell University, 1998.

[Int01] Intel et al., "Extensible Firmware Interface", Version 1.02, http://developer.intel.com/technology/efi/, 2001.

[JXT93] J. Ju, G. Xu, J. Tao, "Parallel Computing using Idle Workstations", Operating Systems Review, pp. 87-96, 1993.

[Kle00] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective", in Proceedings of the 32nd ACM Symposium on Theory of Computing, 2000.

[Knu73] D.E. Knuth, "Fundamental Algorithms", Addison-Wesley, 1973.

[KRR95] Kim Keeton, Steve Rodrigues, Drew Roselli, "Previous Work in Distributed Operating Systems", Technical Report, The Berkeley Network of Workstations (NOW) Project, 1995.

[Lam78] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", Communications of the ACM 21(7): 558-565, 1978.

[LBe99] J. Liebeherr, Tyler K. Beam, "HyperCast: A Protocol for Maintaining Multicast Group Members in a Logical Hypercube Topology", July 1999. Proc. First International Workshop on Networked Group Communication (NGC '99), in: Lecture Notes in Computer Science, Vol. 1736, pp. 72-89, 1999.

[LHK+79] Lawson C.L., Hanson R.J., Kincaid D., Krogh F.T., "Basic Linear Algebra Subprograms for FORTRAN usage", ACM Transactions on Mathematical Software, vol. 5, pp. 308--323, 1979.

[LLM88] Litzkow M.J., Livny M., Mutka M.W., "Condor: A hunter of idle workstations", Proceedings of the 8th International Conference on Distributed Computer Systems, 1988.

[Lom02] F. Lombard, "Spécification et mise en œuvre d'une plateforme de métacomputing multi-agents", Thèse de Doctorat, université de Franche-Comté, 2002.

[Mic97] Microsoft Corporation, "OnNow: the evolution of the PC platform", http://www.microsoft.com/hwdev/onnow.htm, 1997.

[MKN+02] Dejan S. Milojicic, Vana Kalogeraki, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, John Sontag, "Peer-To-Peer (P2P)

Technology", HP Labs Technical Report HPL-2002-57 (Submitted to ACM Computing Surveys), 2002.

[MNS+00] S. Matsuoka, H. Nakada, M. Sato, S. Sekiguchi, "Design Issues for the Network Enabled Server Systems for the Grid", Grid Forum, Advanced Programming Models Working Group Whitepaper, 2000.

[Moo65] Moore G.E., "Electronics", Volume 38, num.8, pp.114-117, 1965.

[Mpi93] MPI Forum, "MPI: A message passing interface", Proceedings of 1993 Supercomputing Conference, Portland, Washington, 1993.

[MRi01] Martin C., Richard O.,"Parallel launcher for clusters of PCs", Proceedings of Parco 2001.

[MSD+01] Meuer H.W., Strohmaier E., Dongarra J.J., Simon H.D., "TOP500 Supercomputer Sites", Proceedings of the 16th International Supercomputer Conference, June 2001.

[NHG01] A. Natrajan, M. Humphrey, A. Grimshaw, "Capacity and Capability Computing using Legion", Lecture Notes in computer Science, 2073, 2001.

[Nic87] Nichols D., "Using Idle Workstations in a Shared Computing Environment", Proceedings of the 11th ACM Symposium on Operating System Principles, pp. 5-12, 1987.

[NSS99] H. Nakada, M. Sato, S. Sekiguchi, "Design and Implementations of Ninf: Towards a Global Computing Infrastructure", Future Generation Computer Systems, Metacomputing Issue, 15(5-6):649-658, 1999.

[OCD+88] J. K. Ousterhout, A.R. Cherenson, F. Douglis, M.N. Nelson, B.B. Welch, "The Sprite Network Operating System", IEEE Computer Magazine 21, 2,1988.

[Omg92] Object Management Group, "The Common Object Request Broker (CORBA): Architecture and Specification", document number 91.12.1, Revision 1.1, 1992.

[Ora01] A. Oram et al., "Peer-to-Peer: Harnessing the Power of Disruptive Technologies", O'Reilly and Associates, 2001.

[PLi94] Pruyne J., Livny M., "Providing resource management services to parallel applications", Proceedings of the Second Workshop on Environments and Tools for Parallel Scientific Computing, 1994.

[PLi96] Pruyne J., Livny M., "Interfacing Condor and PVM to harness the cycles of workstation clusters", Journal on Future Gerenations of Computer Systems, 12:53-65, 1996.

[PWa85] G.J. Popek, B.J. Walker, "The LOCUS Distributed System Architecture", Computer Systems Series. MIT Press, Cambridge, Massachusetts, 1985.

[PWD01] A. Petitet, R.C. Whaley, J.J. Dongarra, "Automated Empirical Optimization of Software and the ATLAS Project" in Parallel Computing, Volume 27, Numbers 1-2, pp 3-25, 2001.

[RAM+01] Richard B., Augerat P., Maillard N., Derr S., Martin S., Robert C, "I-Cluster: Reaching TOP500 performance using mainstream hardware", HP Labs technical report HPL-2001-206, http://www.hpl.hp.com/techreports/2001/HPL-2001-206.html,2001.

[RAu02a] Richard B., Augerat P., "I-Cluster: Intense computing with untapped resources", Proceeding of 4th International Conference on Massively Parallel Computing Systems, April 2002, Ischia, Italy.

[RAu02b] Bruno Richard, Philippe Augerat, "Grappe virtuelle I-Cluster : Gestion distribuée des ressources", Actes de l'Ecole d'hiver GRID 2002, Aussois, France, Dec. 2002.

[RCh02] Bruno Richard, Denis Chalon, "Distributed File System", European Patent Office application number 2354108.9, 2002.

[RCM03] Bruno Richard, Denis Chalon, Donal Mac Nioclais, "Clique: A transparent, peer-to-peer replicated file system", Proceedings of the 4th International Conference on Mobile Data Management, Melbourne, Australia, Jan. 2003

[RDr01] A. Rowstron, P. Druschel, "Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems", Proceedings of IFIP/ACM Middleware 2001, Heidelberg, Germany, 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, "A Scalable Content-Addressable Network", Proceedings of ACM SIGCOMM'01, San Diego, 2001.

[Ric01] Richard B., "I-Cluster Framework", HP Labs Grenoble (Unpublished, HP private document), May 2001.

[Ric02] Bruno Richard, "I-Cluster: the Execution Sandbox", Proceedings of the IEEE International Conference on Cluster Computing, Sep. 2002.

[Rit01] J. Ritter, "Why Gnutella Can't Scale. No, Really.", http://www.darkridge.com/jpr5/doc/gnutella.html, 2001.

[RMC02] Bruno Richard, Donal Mac Nioclais, Denis Chalon, "Clique: A transparent, Peer-to-Peer collaborative file sharing system", HP Labs Technical Report HPL-2002-307, 2002.

[RSA78] R. Rivest, A. Shamir, L. Adleman, "A method for obtaining Digital Signatures and Public Key Cryptosystems", Communications of the ACM 21(2) pp 120-126, 1978

[Sar98] L. Sarmenta, "Bayanihan: Web-Based Volunteer Computing Using Java", 2nd International Conference on World-Wide Computing and its Applications (WWCA'98), Tsukuba, Japan, March 3-4, 1998.

[Sar01] L. Sarmenta, "Volunteer Computing", PhD. Thesis, Massachussets Institute of Technology, Department of Electrical Engineering and Computer Science, 2001.

[SCa92] L. Smarr, C.E. Catlett, "Metacomputing", Communications of the ACM, 35(6):45-52, June 1992.

[Sch93] M. Schneider, "Self-stabilization", ACM Computing Surveys, 25(1), 1993.

[SGB+02] A. Selhikov, C. Germain, G. Bosilca, F. Cappello, G. Fedak, "MPICH-CM: a P2P MPI implementation", 9th conference EuroPVM/MPI, Johannes Kepler University Linz, Austria September, October 2002.

[SGT96] D.J. Scales, K. Gharachorloo, C.A. Thekkath, "Shasta: A Low-Overhead Software-Only Approach to Fine-Grain Shared Memory", In Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII), 174-185, 1996.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications", Proceedings of ACM SIGCOMM'01, San Diego, 2001.

[SSC+93] Mark Swanson, Leigh Stoller, Terence Critchlow, Robert Kessler, "The Design of the Schizophrenic Workstation System", Proceedings of the Third Usenix Mach Symposium, 1993.

[Sun90] Sunderam V.S., "PVM: A framework for parallel distributed computing", Concurrency: Practice and Experience, vol. 2(4), pp. 315--339, 1990.

[Sun96] Sun Microsystems, "Remote Method Invocation", 1996.

[SWB+97] W. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, D. Anderson, "A New Major SETI Project Based on Project SERENDIP Data and 100,000 Personal Computers", in Astronomical and Biochemical Origins and the Search for the Life in the Universe, 1997.

[Tan95] A.S. Tanenbaum, "Distributed Operating Systems", Prentice-Hall, 1995.

[TKV+91] A.S. Tanenbaum, M.F. Kaashoek, R. van Renesse, H. Bal, "The Amoeba Distributed Operating System - A Status Report", Computer communications, Vol. 14 , pp. 324-335, Jul 1991.

[TVV+90] Tanenbaum A.S., Van Renesse R., Van Staveren H., Sharp G.J., Mullender S.J., Jansen J., Van Rossum G., "Experiences with the Amoeba distributed operating system", CACM, 33(12):46--63, 1990.

[Via97] "Virtual Interface Architecture Specification v. 1.0", Compaq, Intel and Microsoft Corporations, http://www.viarch.org/, 1997.

[VMH98] R. Van Renesse, Y. Minsky, M. Hayden, "A Gossip-Style Failure Detection Service", in Middleware '98: IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, pp. 55--70, 1998.

[Vmw00] VMware, Inc. "Virtual Platform" http://www.vmware.com/products/virtualplatform.html, 2000.

[Wap02] Wapiti, "Super Ordinateur", Wapiti, le mensuel pour les 7-13 ans, Oct 2002.

[Wat99] D.J. Watts, "Small worlds. The Dynamics of Networks between Order and Randomness", Princeton University Press, Princeton, New Jersey, 1999.

[WSH99] Wolski R., Spring N., Hayes J., "The Network Weather Service: A distributed resource performance forecasting service for metacomputing", Future Generations of Computer Systems, 15:757-768, 1999.

[WSH99b] Wolski R., Spring N., Hayes J., "Predicting the CPU Availability of Time-shared Unix", Proceedings of 8th IEEE High Performance Distributed Computing Conference, August 1999.

[WSP97] Rich Wolski, Neil Spring, Chris Peterson, "Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service", in Proceedings of SC97, November, 1997.

[YDF+02] Xin Yuan, Scott Daniels, Ahmad Faraj, Amit Karwande, "Group Management Schemes for Implementing MPI Collective Communication over IP-Multicast", The 6th International Conference on Computer Science and Informatics, pages 76-80, Durham, NC, March 8-14, 2002.

[Zho92] Zhou S., "LSF: load sharing in large-scale heterogeneous distributed systems", Workshop on Cluster Computing, Tallahassee, Florida, 1992.

I-Cluster : Agrégation des ressources inexploitées d'un intranet et exploitation pour l'instanciation de services de calcul intensif

Résumé :

Notre étude s'intéresse aux machines en jachère disponibles sur un intranet afin de les agréger en grappes virtuelles de calcul scientifique. Dans le cadre du projet I-Cluster, nous avons étudié et réalisé l'infrastructure permettant de tirer parti de manière transparente des PC inexploités d'un réseau d'entreprise.

En particulier, nous présentons des mécanismes novateurs permettant de faire passer un PC entre deux modes de travail "utilisateur" et "calcul", exclusifs l'un par rapport à l'autre. Ces mécanismes sont basés sur l'identification des ressources de calcul disponibles, la détection de leurs périodes d'inexploitation à l'aide de leur profil d'utilisation observé, d'un système de prédiction de fenêtres de tir et d'un bac à sable d'isolation de code.

Par ailleurs, nous proposons un annuaire de gestion distribuée des ressources disponibles, le « nuage I-Cluster », fonctionnant en mode pair-à-pair sans serveur, auto-organisant et passant à l'échelle sur plusieurs dizaines de milliers de machines.

Mots-clés :

Calcul parallèle, métacalcul, grappe virtuelle, pair-à-pair (peer-to-peer), isolation d'exécution, bac à sable, bavardage, réplication optimiste.

I-Cluster: Gathering idle intranet resources for instanciation into compute-intensive services

Summary:

Our study aims at gathering idle intranet resources in order to aggregate them into compute intensive services. In the context of the I-Cluster project, we have been studying and developing an infrastructure that enables transparent use of unused PCs of a corporate network.

In particular, we offer novel mechanisms that enable a switch between two modes "user" and "cluster", exclusive with one another. These mechanisms are based on the identification of available computing resources, the detection of their idle periods using their usage profile, on a launch window prediction system and on an execution sandbox.

Moreover, we propose a distributed resource directory, the "cloud", which is a serverless Peer-to-Peer system, self organizing and scaleable to tens of thousands of machines.

Keywords:

Parallel computing, Metacomputing, virtual cluster, Peer-to-Peer, execution isolation, sandbox, gossiping, opimistic replication.