CONFIIT, un intergiciel pour le calcul pair-à-pair

Michaël Krajecki, Olivier Flauzac, Pierre-Paul Mérel

CReSTIC, Université de Reims Champagne-Ardenne,

BP 1039, F-51687 Reims Cedex 2, France

{michael.krajecki, olivier.flauzac, pp.merel}@univ-reims.fr

Résumé

Nous présentons dans ce papier, l'intergiciel CONFIIT (*Computation Over a Network with Finite number of Irregular Independant Tasks*), développé pour le *GRID computing*. Celui-ci est entièrement écrit en Java, et permet d'aggréger la puissance de calcul disponible sur un réseau en un anneau virtuel. L'efficacité de cette solution est montrée à travers des expérimentations de résolution du problème de Langford.

1 Introduction

Le besoin en puissance de calcul et en capacité de stockage augmente continuellement. Dans le même temps, les capacités de calcul augmentent elles-aussi, mais la demande est toujours supérieure à l'offre. Pendant des années, le calcul et le stockage étaient réalisés de manière centralisée. Depuis quelques années, une nouvelle approche a émergé : la distribution du calcul et des stockages d'informations sur un réseau. Cette nouvelle approche tend vers une globalisation de toutes les ressources disponibles sur un réseau local, et à plus grande échelle, sur Internet. On appelle *GRID computing* cette nouvelle manière de concevoir les applications de calcul. Dans [7], les auteurs présentent l'étendue de ce nouveau concept, et expliquent comment concevoir des solutions efficaces basées sur le GRID.

CONFIIT, est un intergiciel pair-à-pair, totalement distribué, conçu pour partager les moyens de calcul disponibles sur un réseau local ou sur Internet.

Le reste de ce papier est organisé comme suit : tout d'abord, nous présentons succinctement les solutions existantes de *GRID computing*. Nous introduisons ensuite la notion d'application FIIT, puis nous décrivons CONFIIT. Nous terminons enfin par une série d'expérimentations validant le logiciel CONFIIT sur le plan pratique.

2 Travaux connexes

Les applications GRID sont en fait conçues dans deux buts : le partage de données et le calcul distribué.

Les applications GRID de partage de données ont pour objectif de permettre à l'utilisateur de partager et d'obtenir des données à travers le réseau. Ces applications sont basées sur plusieurs composantes : recherche de fichiers, publication de fichiers, obtention de statistiques, et bien sûr, téléchargement de fichiers. Napster, FreeNet ou Gnutella sont des exemples connus de ce type d'applications.

Les applications GRID de calcul distribué visent à partager un calcul sur le réseau, pour construire une solution globale. Dans le reste de ce papier, nous nous concentrerons exclusivement sur ce type d'applications.

Dans le concept de GRID, on trouve différentes applications ou environnements. On distinguera trois familles principales. La première catégorie regroupe les applications conçues pour la résolution d'un problème spécifique. La plus connue est le projet SETI[1], développé pour la recherche d'intelligence extraterrestre. La deuxième famille concerne les protocoles et bibliothèques conçues pour aider au développement d'applications GRID dédiées. Le projet JXTA[2] en est un exemple. La dernière famille concerne les intergiciels, offrant divers services (construction et administration de grilles, partage de tâches et collecte de résultats). XtremWeb[4], Globus[6] ou DIET [2] en font partie.

Quelque soit l'application, et quelques soient les moyens utilisés, un outil basé sur le GRID doit vérifier certaines propriétés pour être efficace[8] : partager des capacités matérielles et logicielles, et ordonnancer leur usage sur le réseau sous-jacent.

Dans [1], F. Cappelo nous éclaire sur l'utilisation, dans la plupart des cas, des architectures centralisées ou semi-centralisées. On peut noter que Freenet est conçu comme un système de stockage d'informations distribué

[1] `http://setiathome.ssl.berkeley.edu/`

[2] `http://www.jxta.org/`

décentralisé [3].

3 Applications FIIT

Nous avons défini la notion d'applicatins FIIT dans [9]. Elles sont constituées d'un nombre fini de tâches irrégulières et indépendantes. Nous supposons que chaque tâche vérifie trois propriétés : tout d'abord, une tâche ne peut faire aucune hypothèse sur l'exécution d'une autre tâche. Ainsi, il n'y a aucunes communications entre les tâches. Ensuite, le temps d'exécution d'une tâche est imprévisible. En d'autres termes, on ne peut avoir une bonne estimation du temps d'exécution d'une tâche avant sa fin. En troisième lieu, le même algorithme est utilisé pour calculer chaque tâche de l'application. Ainsi, deux tâches ne seront distinguées que par l'ensemble des données qu'elles traitent.

A cause de l'irrégularité des tâches, nous devons proposer des statégies d'équilibrage de charge pour répartir efficacement les applications FIIT. Pour sa part, l'utilisateur devra répondre à deux questions pour développer une application FIIT : comment redéfinir le problème avec un nombre fini de tâches indépendantes, et comment définir une fonction qui résoud ces tâches. Les choix de l'utilisateur à ce niveau sont prépondérants. Il doit en effet définir la granularité (le nombre de tâches) adaptée à la machine ou au réseau visé. En effet, la granularité a un impact sur la qualité de l'équilibrage de charge et sur l'utilisation du réseau d'interconnexion.

4 L'intergiciel CONFIIT

CONFIIT (*Computation Over Network for FIIT*) est un intergiciel pour le calcul pair-à-pair. Il a pour but de distribuer sur le réseau, toutes les tâches obtenues par décomposition d'un problème FIIT, de résoudre chaque tâche, et de diffuser les résultats de ces différentes tâches. CONFIIT est totalement distribué, et est conçu pour fonctionner sur un réseau hétérogène : différents systèmes (Unix, Windows...) , matériels (Intel, Sparc...) ou architectures (mono, multi-processeurs).

4.1 Architecture générale

Chaque machine du réseau participant à un calcul CONFIIT est appelée un *nœud*. Un nœud est constitué de trois *threads* principaux : un gestionnaire de topologie et de communication, un gestionnaire de tâches, et un ou plusieurs solveurs, dédiés à un problème.

Tous les nœuds sont connectés selon un anneau logique. Ainsi, chaque nœud connaît sont prédecesseur et son successeur. Les différents nœuds communiquent par l'intermédiare d'un jeton qui diffuse l'état du calcul[3] le long de l'anneau.

Un gestionnaire de tâche peut gèrer plusieurs solveurs. Typiquement, un solveur doit être lancé sur chaque processeur disponible sur le nœud. En plus des trois *threads* principaux, chaque nœud dispose de ses propres données sur le problème et sur l'état du calcul.

4.2 Gestion des tâches

Une fois inséré dans l'anneau, un nœud dispose des différents paramètres du calcul en cours (une liste des tâches et les résultats partiels connus). Il est alors capable de lancer localement les calculs.

Au début d'un calcul, toutes les tâches sont notées *non calculées*. Quand une tâche est terminée (localement), son résultat est stocké et sera propagé le long de l'anneau par le jeton de collaboration. Pour éviter une perte de temps, le gestionnaire de tâches lance une nouvelle tâche sans attendre le jeton. Mais la tâche choisie peut avoir été lancée sur un autre nœud, et sera alors *répliquée* dans l'anneau. La liste des tâches à calculer est réarrangée localement par chaque nœud. Ainsi, quand le gestionnaire de tâches choisit la $n^{ième}$ tâche, elle sera généralement différente de la $n^{ième}$ tâche sur un autre nœud [5].

4.3 Implémentation Java

Le prototype de CONFIIT, développé en Java, est constitué de six classes principales. La classe `Confiit` constitue le programme principal. Nous avons choisi le problème de Langford pour expérimenter CONFIIT, car il constitue un défi de calcul ouvert. On peut résumer ce problème comme suit : pour un nombre donné n de couleurs ordonnées, on dispose de $2n$ cubes (2 cubes de chaque couleur). Chaque cube doit être placé de sorte qu'il y ait exactement k cubes entre les deux cubes de couleur k. Le but est de dénombrer tous les arrangements possibles.

La figure 1 schématise l'architecture du prototype CONFIIT. Les communications ne sont pas présentées ici. Elles sont réalisées par des appels XML-RPC[4]. Notons que c'est la distribution Apache qui est utilisée. Quand un nœud doit envoyer des informations, il utilise un objet `XmlRpcClient` pour *appeler* le nœud distant. Le message est transmis dans les paramètres RPC. Les appels CONFIIT sont implémentés par un serveur XML-RPC Apache. Chaque procédure RPC correspond à un appel CONFIIT.

[3] L'état du calcul correspond à la liste des états individuels des tâches (non commencée, en cours ou terminée.

[4] Car XML-RPC permet d'encapsuler les données à échanger dans un protocole normalisé reconnu.

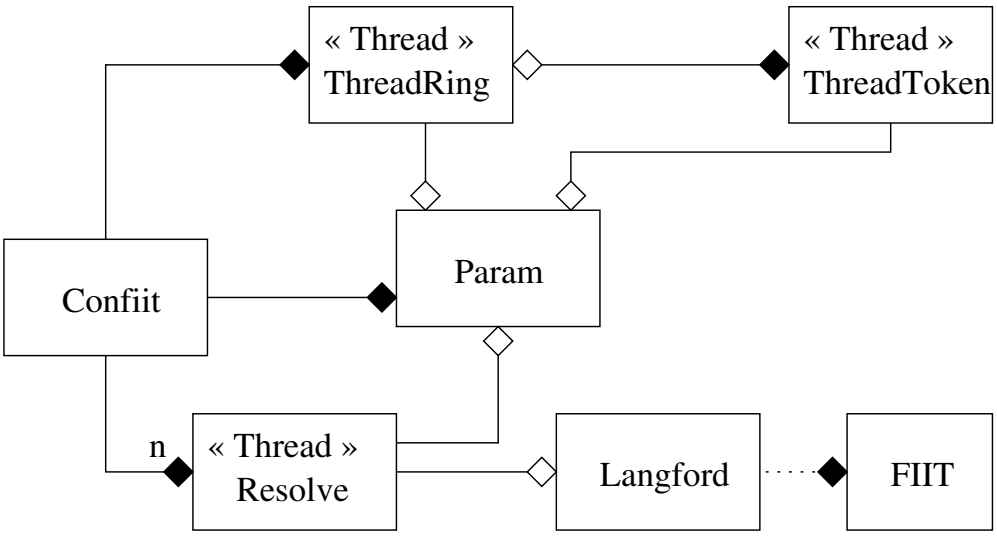

FIG. 1 – Structure de CONFIIT.

Quand un nœud initie un calcul, il rassemble d'abord les informations nécessaires à l'application (nombre de couleurs et granularité pour le problème de Langford). Quand un nœud rejoint un anneau existant, il récupère les paramètres d'exécution depuis un nœud déjà inséré.

Après l'initialisation, le programme principal crée un objet de classe `Param` qui rassemble les données partagées par tous les *threads* du nœud. Les *threads* de service sont alors instanciés.

4.3.1 Objet `ThreadRing`

`ThreadRing` est chargé de la gestion de l'anneau. Il mémorise les informations relatives à la topologie et offre plusieurs services :

- `InsertNode` pour permettre à un nouveau nœud d'entrer dans l'anneau. Celà peut être vu comme un appel *public* (implémenté par un appel RPC), car chaque machine à l'extérieur de l'anneau peut ainsi devenir un nœud CONFIIT.
- `AddNode` pour permettre la propagation des informations sur l'entrée d'un nouveau nœud.
- `DeleteNode` pour permettre la propagation des informations sur la suppression d'un nœud. Ces deux services sont considérés comme *privés* car seuls des nœuds CONFIIT peuvent les utiliser.
- `SendToken` pour permettre l'envoi d'un jeton de collaboration. C'est un appel *interne* (non implémenté par XML-RPC).

4.3.2 Objet `ThreadToken`

`ThreadToken` est chargé de la gestion du jeton de collaboration. Il met à jour la vue locale du calcul (dans `Param`), en fonction des informations reçues du jeton, et propage globalement les informations sur l'anneau. Il implémente seulement un appel RPC : `TokenReceive`, qui constitue le point d'échange entre les nœuds. C'est un appel privé selon la taxinomie CONFIIT.

4.3.3 Objet `Resolve`

`Resolve` est chargé de la résolution des tâches. Aucune méthode appellable n'est proposée ici, mais le *thread* calcule les différentes tâches en appelant l'objet `Langford`. On notera que `Langford` est spécifique à notre exemple et pourra être remplacé par un objet quelconque, implémentant l'interface `FIIT`.

4.4 Gestion de la topologie

La gestion de la topologie est réalisée dans `ThreadRing`. Pour assurer la robustesse du système vis-à-vis des pannes système et des entrées dans l'anneau, il maintient une structure centrale représentant l'anneau virtuel. Chaque nœud participant à un calcul est connu par les autres. Les informations sont stockées dans une structure nommée `ring`.

Le mécanisme implémente une k-tolérance pour reconstruire un anneau fonctionnel quand k nœuds successifs s'arrêtent dans un anneau de taille n ($k \leq n$). Dans le cas d'anneaux de grande taille, la k-tolérance permet

de maintenir un anneau fonctionnel dans la plupart des cas, sans imposer d'efforts importants pour mettre les structures locales à jour.

Comme `ring` est une donnée locale, un algorithme distribué est mis en place pour maintenir une vue globale du réseau sur chaque nœud. Chaque changement de topologie est diffusé par un jeton spécial circulant à contre-sens par rapport au jeton de collaboration. Ainsi, quand un nœud entre ou sort de l'anneau, les nœuds qui sont le plus concernés par le changement sont averti en premier.

4.4.1 Entrée dans l'anneau

Une machine désirant entrer dans un calcul existant appelle un nœud CONFIIT sur la procédure `InsertNode`. En réponse, il reçoit par un appel `DefineTask`, les paramètres d'exécution ainsi que la topologie de l'anneau. Il peut ainsi commencer un calcul, sans connaitre l'ensemble des tâches déjà calculées.

Du point de vue du nœud appelé, il est nécessaire de mettre les autres nœuds au courant de la modification (par un appel à `AddNode`), et d'envoyer toutes les informations relatives aux tâches déjà calculées vers l'appelant. Cette étape est réalisée par un *thread* spécialisé afin de ne pas surcharger le service d'insertion principal. En effet, l'envoi des tâches, et surtout de leurs résultats, peut s'avérer très coûteux dans la cas de grosses applications.

On notera que le jeton d'ajout d'un nœud (circulant à contre-sens) sera lancé sans attente particulière, et ne fera qu'un tour d'anneau avant d'être détruit, limitant ainsi la latence de la mise à jour, et la charge du réseau.

4.4.2 Sortie de l'anneau

Un nœud peut quitter un calcul CONFIIT pour plusieurs raisons :

1. l'utilisateur décide d'arrêter sa collaboration au calcul ;
2. un problème réseau rend le nœud injoignable ;
3. le système est tombé en panne.

Dans le premier cas (sortie *volontaire*), la procédure est initiée par le nœud sortant, alors que pour les deux autres, elle est initiée par son prédécesseur dans l'anneau. Seuls ces deux derniers cas nous intéresseront.

Un problème imposant une modification de topologie est détecté au moment d'une tentative d'envoi du jeton de collaboration. Il induit alors la suppression du nœud injoignable et l'envoi d'un jeton de notification. Comme pour l'entrée dans l'anneau, ce jeton circule à contre-sens par rapport au jeton de collaboration, et ne fera qu'un seul tour avant d'être détruit.

4.5 Circulation des jetons

Le principal problème que peut rencontrer CONFIIT se produit quand un nœud tombe en panne alors qu'il était en possession du jeton. Ce cas induit l'arrêt de la collaboration entre les nœuds de calcul. Le même problème se produit en cas de perte d'un jeton de mise à jour de la topologie, et sera résolu de la même manière : à chaque envoi d'un jeton, un délai de garde est lancé, et le jeton sera ré-émis si le délai arrive à échéance avant que le jeton n'ait été reçu à nouveau.

Toutefois, ce mécanisme présente un inconvénient caché : plusieurs nœuds peuvent détecter une perte du jeton dans le même tour, induisant une duplication du jeton[5]. Il devient alors nécessaire d'éliminer les jetons redondants. Nous définissons dans CONFIIT, un mécanisme d'élection qui s'appuie sur un marquage du jeton avec un numéro de séquence (*Seq*) et un nom de nœud (*Name* :). Chaque nœud garde une trace du dernier jeton reçu (*maxSeq* et *maxName*).

A réception d'un jeton, le nœud analyse la situation selon le mécanisme suivant :

1. $maxSeq > Seq$: le token est ignoré.
2. $maxSeq < Seq$: le token est traité, et *maxSeq* est mémorisé.
3. $maxSeq = Seq$: la discussion doit alors porter sur *Name* :. Si celui-ci est égal à *maxName*, le jeton est traité (cas normal). En cas de différence, le jeton ne sera maintenu que si $maxName > Name$ (élection du jeton portant le nom unique le plus grand).

5 Expérimentations

Un certain nombre d'expérimentations ont été effectuées pour valider l'intergiciel CONFIIT. Nous présentons ici trois grandes phases de test. Les deux premières visant à valider les algorithmes mis en œuvre dans CONFIITsur des problèmes de taille modeste, la troisième visant à tester le système sur un problème de taille importante, et dans un réseau local étendu.

5.1 Machine parallèle

Les premiers tests ont été réalisés sur de petites instances du problème de Langford (15 couleurs = 39 809 64 solutions). Ce problème est résolu depuis longtemps, mais il permet de tester le système en ayant des références extérieures. Les tests sur machines parallèles permettent quant à eux, de vérifier les algorithmes de base, sans risque de disparition d'un ou plusieurs nœuds.

[5] Expérimentalement, plus de 50% des nœuds détectent la perte du jeton dans le même tour.

La machine parallèle utilisée est une SUNFire 6800 disposant de 24 processeurs UltraSparc III 900 MHz et 24 Go de mémoire. Les différents résultats sont présentés dans le tableau 1.

L'efficacité de CONFIIT est prouvée quant à l'équilibrage de charge puisque les expérimentations font apparaître une accélération presque linéaire selon le nombre de processeurs utilisés. On notera que pour ces tests, la topologie est réduite à un seul nœud, disposant d'un nombre variable de *threads* de calcul.

	Temps	Accél.	Efficacité
1 proc.	4933	1.00	100%
2 proc.	2471	1.99	99.5%
4 proc.	1239	3.98	99.5%
8 proc.	626	7.88	98.5%
10 proc.	499	9.88	98.9%
12 proc.	415	11.89	99.0%
16 proc.	331	14.90	93.1%

TAB. 1 – CONFIIT sur SUNFire 6800.

Les 24 processeurs de la machine n'ont pu être utilisés car les tests ont été effectués en concurrence avec les autres utilisateurs.

5.2 Réseau de PCs

La deuxième série d'expérimentations a été effectuée sur un réseau de PCs : des Célérons 433MHz à 667MHz sous Windows NT, afin de tester le prototype sur un réseau local, et vérifier notamment l'échange d'informations entre plusieurs nœuds de calcul. Les mesures observées portaient sur le temps d'exécution, le nombre de tâches répliquées, ainsi que l'accélération.

Le temps de résolution séquentiel du problème de Langford pour 15 couleurs est de 10582 secondes sur un Céléron 433 MHz, et de 7733 secondes sur un Céléron 633 MHz.

La table 2 présente les résultats obtenus sur le réseau de Céléron 433MHz, en variant le nombre de machines impliquées dans la calcul.

	Temps	Accél.	Efficacité
1 proc.	10582	1.00	100%
2 proc.	5372	1.97	98.5%
4 proc.	2931	3.61	90.2%
8 proc.	1365	7.75	96.9%
16 proc.	745	14.20	88.75%

TAB. 2 – CONFIIT sur Céléron.

Dans cette série de tests, CONFIIT a là aussi montré son efficacité, puisqu'il permet d'obtenir une accélération supérieure à 14 pour 16 processeurs. Comparativement aux résultats obtenus sur machine parallèle, le réseau de PCs est moins performant. Cependant, il faut tenir compte de deux facteurs qui viennent perturber les performances :

1. l'utilisation de 16 processeurs impose le lancement de 16 nœuds de calcul, et donc une perte au lancement : les insertions se font les unes après les autres et consomment une partie de la puissance de calcul pour mettre les données topologiques en commun.
2. La collaboration de plusieurs nœuds induit des tâches répliquées, augmentant artificiellement la charge totale de travail.

La résolution du problème de Langford avec 15 couleurs génère 2186 tâches avec une profondeur 3 (3 cubes placés arbitrairement en haut de l'arbre d'exploration). On a observé expérimentalement que le nombre de tâches répliquées atteint presque 9% avec 16 processeurs. Ce chiffre est toutefois révisable à la baisse car ces tests ont été réalisés avec une procédure de choix des tâches qui a été sensiblement améliorées depuis.

5.3 Défi de calcul

La troisième phase de test avait pour but de valider l'intergiciel CONFIIT dans une expérimentation à plus grande échelle. Nous avons pour cela mobilisé un nombre important de machines de systèmes et architectures différentes :

- des PCs Céléron, Pentium et AMD de générations différentes, fonctionnant sous Windows NT et XP.
- des stations de travail Sun Sparc sous Solaris 8 et 9.
- des Macintochs sous Mac Os X.
- des PCs et serveurs bi-processeurs sous Linux.
- une machine parallèle SUNFire 6800 avec 24 processeurs.

Les codes de CONFIIT utilisés étaient un peu différents de ceux utilisés dans les expérimentations précédentes. Ils mettaient notamment en œuvre des techniques d'échange et d'ordonnancement des tâches améliorées pour limiter la charge du réseau, et éviter aux machines les moins rapides de pénaliser globalement le calcul. De plus, la version de l'algorithme de résolution du problème de Langford a été elle aussi améliorée[6]. Une seule résolution a été lancée, avec un calcul du problème pour 23 couleurs (à une profondeur de division de 16, soit 65536 tâches à répartir). On notera que ce problème faisait partie d'une série de défis de calcul puisque personne n'était encore parvenu à le résoudre.

[6]Les améliorations de sont pas présentées ici car elles ne font pas réellement partie de CONFIIT, et relèvent du niveau applicatif.

Le nombre de machines impliquées dans le calcul a atteint un maximum de 63, pour un total de 82 processeurs. Le minimum n'est toutefois pas descendu en dessous de 58 pour 77 processeurs, si on exclut la phase de démarrage du calcul. On notera que toutes ces machines étaient réparties sur le réseau local de l'Université de Reims, mais séparées sur des *switchs* différents, traversant deux *firewalls*. Le réseau d'interconnexion proposait des liens à 10, 100 et 1000 Mbits, en fonction des nœuds.

Il est impossible de parler ici d'accélération ou d'efficacité, tant les machines participant au calcul étaient différentes en terme de performances. Le calcul a duré plus de 100 heures au total, cumulant plus de 300 jours de calcul séquentiel sur l'ensemble des participants.

Le résultat final constitue toutefois un nouveau record du monde sur la résolution du problème de Langford avec 23 couleurs. On peut ainsi affirmer que celui-ci comporte 3 799 455 942 515 488 arrangements possibles. On notera que le calcul a permis de les dénombrer, mais pas de les construire.

6 Conclusion

La version actuelle de CONFIIT a permis de valider un certain nombre de solutions techniques pour la réalisation d'un produit final stable et efficace. On peut notamment considérer comme sûrs, les protocoles d'échange d'informations de collaboration, d'insertion, ou de reconstruction de topologie en cas de pannes locales.

Des expérimentations sur des topologies de taille importante (60 machines), d'architecture ou de systèmes d'exploitation différents, ont été effectuées et ont permis la réalisation de calculs très long. Elles ont toutefois mis en évidence un certains nombre de problèmes qui seront à résoudre dans un futur proche :

- les très longs calculs se sont avéré gourmands en mémoire à cause des communications, et les machines ne disposant pas de suffisament de mémoire RAM ont ralenti de manière significative la circulation du jeton de collaboration ;
- la structure en anneau reste fragile en cas de pannes simultanées d'un nombre important de machines.

Les prochaines versions de CONFIIT viseront naturellement à résoudre ces problèmes. Il sera probable de passer par l'écriture d'une bibliothèque de communication en remplacement de XML-RPC qui est à l'origine de la plupart des difficultés rencontrées avec la mémoire. La topologie en anneau sera aussi remise en cause, car elle ne semble pas raisonnable pour des utilisations à très grande échelle.

Enfin, il sera nécessaire d'évoluer vers un veritable intergiciel, c'est-à-dire ne plus lancer directement un calcul par une application CONFIIT, mais disposer de classes de problèmes « FIIT*ables* » lancés sur des communautées de calcul (anneau ou autre) en cours de fonctionnement.

Références

[1] F. Cappello. Calcul global pair à pair : extension des systèmes pair à pair au calcul. *La Lettre de l'IDRIS*, 4 :14–25, April 2002.

[2] Eddy Caron, Frédéric Desprez, Frédéric Lombard, Jean-Marc Nicod, Martin Quinson, and Frédéric Suter. A scalable approach to network enabled servers. In B. Monien and R. Feldmann, editors, *Proceedings of the 8th International EuroPar Conference*, volume 2400 of *Lecture Notes in Computer Science*, pages 907–910, Paderborn, Germany, August 2002. Springer-Verlag.

[3] I. Clarke. *A Distributes Decentralised Information Storage and Retrieval System*. University of Edinburgh, 1999.

[4] Gilles Fedak, C. Germain, V. Néri, and F. Cappello. XtremWeb : A generic global computing system. In *CCGRID2001, workshop on Global Computing on Personal Devices*. IEEE Press, 2001.

[5] Olivier Flauzac, Michaël Krajecki, and Jean Fugère. Confiit : a middleware for peer to peer computing. In M.L. Gravilova, C.J.K. Tan, and P. L'Ecuyer, editors, *The 2003 International Conference on Computational Science and its Applications (ICCSA 2003)*, volume 2669 (III) of *Lecture Notes in Computer Science*, pages 69–78. Springer-Verlag, Montréal, Québec, June 2003.

[6] I. Foster and C. Kesselman. Globus : A metacomputing infrastructure toolkit. *Supercomputer Applications*, 11(2) :115–128, 1997.

[7] I. Foster and C. Kesselman. *Computational Grids*, chapter 2. Morgan-Kaufman, 1998.

[8] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid : Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[9] Michaël Krajecki. An object oriented environment to manage the parallelism of the FIIT applications. In V. Malyshkin, editor, *Parallel Computing Technologies, 5th International Conference, PaCT-99*, volume 1662 of *Lecture Notes in Computer Science*, pages 229–234. Springer-Verlag, St. Petersburg, Russia, September 1999.

[10] Michaël Krajecki, Olivier Flauzac, and Pierre-Paul Mérel. Focus on the communication scheme in the middleware confiit using xml-rpc. In *International Workshop on Java for Parallel Distrubuted Computing (IWJPDC'04)*, Santa-Fe, New-Mexico, April 2004.