RenPar'17 / SympA'2006 / CFSE'5 / JC'2006
Canet en Roussillon, 4 au 6 octobre 2006

Administration autonome d'applications réparties sur grilles

Stéphane Fontaine[1], Christophe Taton[2], Sara Bouchenak[1], Thierry Gautier[3]

[1] Université Grenoble I – LSR (équipe SARDES), Grenoble, France
[2] INPG – LSR (équipe SARDES), Grenoble, France
[3] INRIA – ID (équipe MOAIS), Grenoble, France
{Stephane.Fontaine, Christophe.Taton, Sara.Bouchenak, Thierry.Gautier}@inria.fr

---

Résumé
L'exploitation des ressources mises à disposition dans le cadre de Grid'5000 ou dans d'autres grilles requière au moins 3 étapes : (i) effectuer une réservation en précisant des contraintes, (ii) attendre que toutes les contraintes soient satisfaites, (iii) installer puis instancier l'application sur les processeurs alloués. Cet article a pour objectif de présenter l'utilisation de techniques et solutions du domaine de l'«autonomic computing» pour, d'une part, *automatiser* ces étapes lors du *déploiement* d'applications parallèles, et d'autre part, le rendre *adaptatif* à la disponibilité des ressources.

Mots-clés : grilles, déploiement d'applications, administration autonome, déploiement adaptatif

---

1. Introduction

1.1. Contexte et problématique

Les grappes et grilles de calcul à hautes performances se généralisent et deviennent l'environnement de base pour l'exécution de serveurs répartis ou d'applications parallèles de calcul scientifique [4, 11, 1, 3, 12]. Elles impliquent des centaines voire des milliers de processeurs reliés par un réseau de communication. Ces infrastructures fournissent une grande capacité de calcul et une haute scalabilité, permettant ainsi aux applications de répondre à une charge croissante (par exemple, le nombre de processus parallèles dans une application de calcul scientifique). Cependant, de par leur complexité, leur hétérogénéité, leur dispersion géographique et leur partage par des applications concurrentes, l'exploitation de ces infrastructures présente de nombreux problèmes qui restent ouverts, parmi lesquels le déploiement des applications. Par déploiement d'une application, nous entendons la réservation des ressources nécessaires à l'exécution de l'application (i.e. processeurs), l'installation si nécessaire du code et des logiciels utilisés par l'application sur les processeurs réservés, puis l'instanciation des composants exécutables de l'application sur ces processeurs (i.e. lancement des processus parallèles).
Le schéma classique de déploiement d'une application sur une grille, par exemple sur l'infrastructure Grid'5000 [12], consiste à : (i) effectuer une réservation de processeurs de la grille en spécifiant, entre autres critères, le nombre de processeurs requis, (ii) attendre que tous les critères de réservation soient remplis avant d'avoir accès aux processeurs demandés, et enfin (iii) installer puis instancier l'application à déployer sur les processeurs alloués. Ce schéma de déploiement présente deux inconvénients majeurs : un déploiement *strict* et *statique*. En effet, ce déploiement est strict car il dépend de contraintes fortes telles que la disponibilité d'un nombre exact de processeurs. Autrement dit, tant que le nombre de processeurs disponibles dans la grille n'est pas supérieur ou égal au nombre de processeurs demandés par le déploiement, le déploiement de l'application ne peut pas s'effectuer. De plus, ce schéma de déploiement est statique dans le sens où il n'est effectué qu'une seule fois au cours de la vie de l'application (i.e. au lancement de l'application), et ne peut être appliqué dynamiquement pour ré-adapter l'application.

1.2. Contributions scientifiques

En considérant toujours l'exemple du nombre de processeurs à allouer à une application à déployer, une approche plus flexible consisterait ici à allouer les processeurs disponibles au moment de la demande de

déploiement, même si leur nombre est inférieur à celui demandé par le déploiement. Ceci permettrait à l'application de commencer son calcul au plus tôt, dans le but de le terminer au plus tôt, et donc d'avoir une meilleure utilisation globale des ressources de la grille. Ultérieurement au déploiement initial d'une application, si de nouvelles ressources se libèrent, l'application peut être re-déployée dynamiquement pour lui permettre de bénéficier des nouvelles ressources disponibles.
L'administration autonome (ou *Autonomic Computing*) a pour objectif de construire des systèmes informatiques autonomes qui s'auto-régulent, autrement dit des systèmes reconfigurables, ré-adaptables et re-déployables [17]. L'objectif de cet article est justement de traiter le problème du déploiement statique et strict des applications réparties sur les grilles, via un *déploiement adaptatif* tel qu'illustré dans l'exemple précédent. Cette approche présente deux avantages principaux : (i) un lancement du calcul de l'application au plus tôt dans le but d'une terminaison au plus tôt, et (ii) une meilleure utilisation globale des ressources de la grille en évitant d'avoir des ressources disponibles non utilisées alors que des déploiements d'applications sont en attente pour raison de contraintes fortes de déploiement.
Dans cet article, nous décrivons la conception du déploiement adaptatif d'applications réparties sur grilles dans la plate-forme d'administration autonome Jade [5]. Nous illustrons l'approche proposée via le déploiement adaptatif de l'application parallèle DOC-G [1] basée sur l'environnement parallèle Kaapi/Athapascan [15]. Par ailleurs, une validation expérimentale sur l'infrastructure de grille Grid'5000 est en cours [12].

1.3. Plan de l'article

Le reste de cet article est organisé comme suit. La section 2 rappelle la problématique de déploiement d'applications sur grilles. La section 3 présente l'approche proposée de déploiement adaptatif sur grilles. La section 4 décrit le plan de validation expérimentale de l'approche proposée et la section 5 présente nos conclusions et perspectives.

2. Contexte et état de l'art : Déploiement d'applications

Dans cette section, nous définissons tout d'abord les phases de déploiement d'applications sur des infrastructures réparties (ex. grilles, grappes), en illustrant notre discours par des exemples de solutions existantes dans le domaine. Nous décrivons ensuite les caractéristiques et une synthèse du déploiement d'applications réparties.

2.1. Phases du déploiement

Le déploiement d'une application répartie sur un ensemble de ressources matérielles (i.e. processeurs) est constituée de plusieurs étapes successives : (i) la *réservation*, (ii) l'*installation* et (iii) l'*instanciation*. A cela s'ajoutent des opérations de configuration qui peuvent survenir à différentes étapes du déploiement. Dans la suite, nous détaillons chacune de ces phases.

Réservation

Cette phase consiste à requérir les ressources nécessaires à l'exécution de l'application à déployer. Pour cela, des critères de réservation sont spécifiés, tels que l'architecture matérielle requise pour l'exécution de l'application, le nombre de processeurs nécessaires ou la durée de la réservation. Par ailleurs, une réservation peut être faite par extension en spécifiant exactement les ressources requise (ex. les adresses IP des machines à réserver), ou par intention en spécifiant des critères généraux (ex. machines d'une architecture donnée). Les infrastructures réparties de grappes ou de grilles proposent généralement un service de réservation de ressources, tel que LSF, PBS et OAR pour les grappes ou OARGrid pour les grilles [20, 21, 19, 23].

Installation

Cette phase du déploiement a pour objectif l'installation de l'ensemble des logiciels nécessaires à l'exécution de l'application à déployer. Ceci inclut le code de l'application elle-même mais aussi les librairies dont dépend l'application et les ressources binaires utilisées par les logiciels (ex. fichiers de configuration, fichiers d'images, bases de données, etc.). La phase d'installation est nécessaire avant toute phase

[1] Résolution du problème de l'affectation quadratique en optimisation combinatoire [9, 18].

d'instanciation d'une application à déployer. Cependant, si une installation est faite de manière permanente (au boot d'un système par exemple), il n'est pas nécessaire de ré-installer l'application avant de l'instancier. Des solutions telles que NIX, Kadeploy ou le modèle de déploiement de l'OMG gèrent l'installation des logiciels [10, 16, 22].

Instanciation

L'instanciation d'une application est la phase de lancement de l'application ; elle consiste à créer les composants exécutables de l'application et à les démarrer, comme par exemple les processus d'une application parallèle. Parmi les solutions existantes d'instanciation d'applications, nous pouvons citer le systèmes SmartFrog [13], le service de launch dans le modèle de déploiement de l'OMG [22] ou l'infrastructure à composants logiciels Fractal/Julia et ses usines à composants [6].

Configuration

Des opérations de configuration peuvent survenir à différentes étapes du déploiement d'une application comme, par exemple, la configuration d'un logiciel installé (par exemple, sa version) ou la configuration des paramètres d'instanciation d'une application (par exemple, le nombre de processus dans une application parallèle). Des outils tels que le langage de description de SmartFrog [13] ou le langage de description d'architecture (ADL) de Fractal [6] permettent de spécifier des configurations de déploiement.

2.2. Caractéristiques du déploiement

Nous distinguons trois caractéristiques du déploiement d'applications : un déploiement *strict*, un déploiement *statique* ou au contraire, un déploiement *adaptatif* :

– *Déploiement strict*. Un déploiement strict est un déploiement dont les paramètres de configuration représentent des contraintes fortes. Parmi les exemples de déploiement strict, nous pouvons citer un déploiement qui dépend d'une architecture matérielle donnée, ou un déploiement exigeant un nombre exact de processeurs pour l'application. Ainsi, dans le premier exemple, tant qu'aucune ressource disponible dans une grille ne correspond à l'architecture requise, le déploiement de l'application est mis en attente. Quant au second exemple, tant que le nombre de processeurs disponibles dans la grille n'est pas supérieur ou égal au nombre de processeurs demandés par le déploiement, le déploiement de l'application ne peut pas s'effectuer.
– *Déploiement statique*. Un déploiement est dit statique s'il ne peut être effectué qu'une seule fois au cours de la vie d'une application. Ainsi, une fois l'application déployée, celle-ci ne peut être re-déployée. Autrement dit, aucune de ses phases de déploiement ne peut être refaite. L'applications ne peut donc, par exemple, pas être re-configurée ni ré-instanciée.
– *Déploiement adaptatif*. Un déploiement adaptatif est un déploiement qui n'est ni strict ni statique. Un déploiement adaptatif est, en effet, un déploiement dynamique qui permet de re-déployer une application au cours de son exécution pour adapter ses paramètres de configuration. Par exemple, grâce à un déploiement adaptatif, il est possible de re-déployer une application parallèle sur un nouveau type d'architecture matérielle, ou sur un nombre et un ensemble différents de processeurs.

Plusieurs systèmes fournissent des outils pour le déploiement d'applications sur grilles, tels que le système Adage de déploiement d'applications MPI ou autres basées sur le modèle à composants Corba CCM [24], GoDIET [7], GridSolve [25], ou Kadeploy [16] et Globus [14]. Malheureusement, la plupart des solutions existantes de déploiement d'applications sur grilles ne proposent qu'un déploiement strict et statique. Dans la section suivante, nous proposons une approche de déploiement adaptatif d'applications sur grilles.

3. Conception d'un système de déploiement adaptatif sur grilles

Nous présentons tout d'abord dans cette section la plate-forme d'administration autonome Jade [5], qui fournit une infrastructure générique de déploiement d'applications réparties sur grappe. Nous proposons ensuite d'utiliser Jade comme base pour mettre en œuvre un gestionnaire autonome de déploiement adaptatif pour applications réparties sur grille.

3.1. Plate-forme d'administration autonome pour grappe

L'administration autonome d'un système repose sur la régulation d'éléments administrables (*Managed Element* ou ME) orchestrée par des gestionnaires autonomes (*Autonomic Manager* ou AM).
Pour fournir une vue homogène de différents systèmes administrés, tout ME expose une interface d'administration uniforme. Cette interface fournit diverses opérations comme, par exemple, positionner ou lire les paramètres de configuration d'un ME (ex. positionner ou lire le numéro de port d'un serveur de base de données). D'autres opérations de cette interface permettent de gérer le cycle de vie d'un ME (ex. démarrer ou arrêter un serveur web), ou gérer les liaisons d'un ME avec un autre ME (ex. connecter un serveur d'entreprise à un serveur de bases de données). Pour que l'administration des systèmes patrimoniaux se fasse de manière générique, tout élément administré est encapsulé dans un composant logiciel particulier fournissant l'interface d'administration uniforme. Un tel composant est appelé *wrapper*. Il a pour rôle de traduire les opérations génériques de l'interface en actions spécifiques à l'élément administré. Les *wrappers* sont basés sur le modèle à composants Fractal (*http ://fractal.objectweb.org*). Les composants Fractal sont des entités d'exécution réparties. Un composant peut être connecté à un autre composant ou constitué d'autres composants. Fractal autorise également l'introspection et la reconfiguration dynamique de ses composants, ce qui les rend particulièrement adaptés pour nos besoins.
En s'appuyant sur la vue homogène des systèmes administrés qu'offre l'interface d'administration à base de composants, nous construisons des gestionnaires autonomes génériques pour réguler et optimiser des aspects de l'administration de ces systèmes. Un gestionnaire autonome repose sur une boucle de contrôle composée : (i) de sondes permettant la détection de changements dans l'environnement du système administré, (ii) d'un régulateur chargé d'interpréter et de réagir aux diverses informations collectées par les sondes et enfin (iii) d'actionneurs commandés par le régulateur et dont la tâche est d'agir concrètement sur le système administré. La Figure 1 présente une vue générale de l'architec-

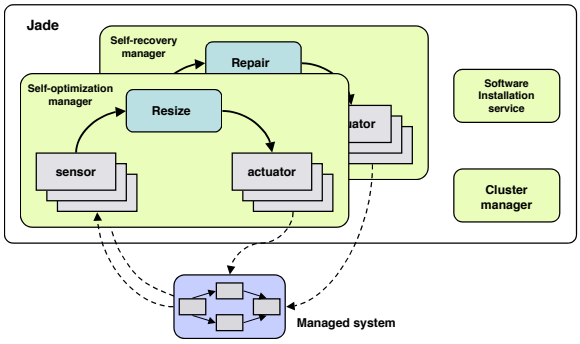

FIG. 1 – Architecture de la plate-forme d'administration autonome Jade

ture de la plate-forme d'administration autonome Jade [5]. La plate-forme Jade intègre divers gestionnaires ciblant des aspects d'administration autonome tels que l'auto-adaptation face à des défaillances (auto-réparation) ou à des variations de charge (auto-optimisation). Le gestionnaire d'auto-réparation s'appuie, par exemple, sur des sondes permettant de détecter les pannes de composants logiciels ou matériels du système administré ; et sur des actionneurs permettant de redéployer et de remplacer les composants défectueux du système pour le rendre à nouveau opérationnel. Le gestionnaire d'auto-optimisation repose, par exemple, sur des sondes permettant de détecter des variations de la charge appliquée au système administré ; et des actionneurs capables de contrôler le taux de duplication des serveurs en grappes afin d'adapter la capacité de traitement du système.
La plate-forme d'administration autonome Jade fournit de plus divers services généraux, parmi lesquels un service de réservations de ressources et un service d'installation de logiciels (voir Figure 1).

3.2. Extension de Jade pour le déploiement adaptatif sur grilles

La plate-forme d'administration autonome Jade fournit une infrastructure générique pour l'administration d'applications réparties sur grappe et propose une interface de déploiement strict et statique. Nous proposons ici d'intégrer à Jade des mécanismes de déploiement adaptatif pour application répartie, au moyen d'un gestionnaire autonome de déploiement adaptatif.

Le gestionnaire proposé est constitué d'une boucle de contrôle composée : (i) de sondes permettant de connaître l'état de la réservation des nœuds de la grille, (ii) d'un régulateur spécialisé pour adapter le déploiement d'une application à mesure que des nœuds sont libérés ou alloués sur la grille, (iii) et enfin, d'actionneurs pour déployer dynamiquement des applications sur la grille.

La Figure 2 présente la structure du gestionnaire de déploiement adaptatif sur la grille. Des sondes observent l'état des réservations des ressources sur chacune des grappes constituant la grille. Une sonde agrège ces observations pour fournir des informations synthétiques au gestionnaire de déploiement. Le gestionnaire dispose alors d'actionneurs à l'échelle de la grille pour allouer ou libérer dynamiquement des nœuds sur la grille, puis pour déployer et instancier des applications sur ces nœuds. Ces actionneurs reposent sur des actionneurs disposés sur chacune des grappes pour effectuer des opérations d'allocation ou de libération de nœuds et de déploiement ou d'instanciation des applications sur les grappes.

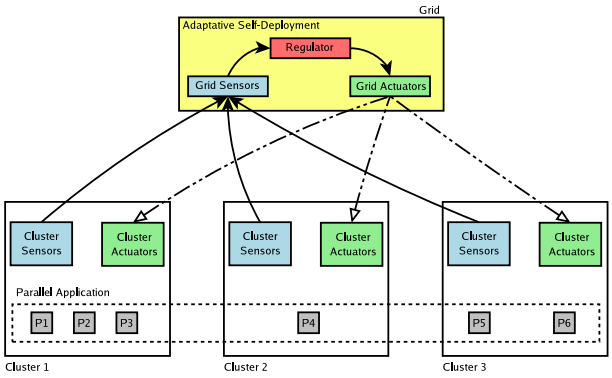

FIG. 2 – Gestionnaire de déploiement adaptatif sur grille

Lorsqu'un nœud se libère sur la grille, une sonde déployée sur la grappe contenant ce nœud émet un événement qui est propagé vers le régulateur du gestionnaire de déploiement adaptatif. Le régulateur décide alors d'attribuer ce nœud disponible à l'application répartie dont il a la charge. À l'aide des actionneurs, le nœud est réservé, puis l'application est installée, instanciée et finalement intégrée à l'application répartie. Symétriquement, lorsqu'une allocation de nœuds survient sur la grille alors que celle-ci est, par exemple, totalement saturée, le régulateur du gestionnaire reçoit une notification et prend alors la décision de libérer certains des nœuds utilisés par l'application dont il a la charge.

3.3. Contraintes sur les applications déployées

Toutes les applications parallèles ne peuvent pas profiter d'un déploiement adaptatif. Une application parallèle est un ensemble de processus communicants participant à la résolution d'un même problème. L'exploitation efficace du parallélisme physique dépend de l'ordonnancement des différentes tâches de calcul du problème sur les processeurs afin d'équilibrer la charge totale sur les différents processeurs tout en minimisant les communications entre processeurs. L'ajout ou le retrait de processeurs à une application en cours d'exécution implique que l'application puisse re-distribuer la charge de calcul en tenant compte de ce changement. Il s'agit de la contrainte la plus importante imposée aux applications pour qu'elles puissent profiter d'un déploiement adaptatif.

La plupart des applications parallèles sont conçues en utilisant l'interface d'échange de messages MPI. Cette interface permet aux concepteurs de décrire les envois et les réceptions de messages effectués par

les différents processus de l'application. On parle de modèle de programmation à *parallélisme explicite*. Le calcul de l'ordonnancement des tâches de calcul est à la charge du programmeur. Généralement, il est intégré à l'application parallèle et est calculé, une fois pour toute, au début de l'exécution en tenant compte du nombre de processeurs précisé à l'étape de lancement. L'ajout et le retrait de processus n'est généralement jamais possible. De plus, la redistribution de charge n'est généralement jamais prise en compte.
Dans le cadre de ce travail, nous utilisons des applications conçues au dessus de l'environnement Kaapi [15] qui permet la description d'un parallélisme potentiel indépendamment du parallélisme physique disponible. On parle de modèle de programmation à *parallélisme implicite*. Le moteur exécutif intègre un algorithme d'ordonnancement qui se charge de replier efficacement le parallélisme potentiel, généralement très important, sur un nombre fixé de ressources. Dans ce cadre, l'ajout de processus de calcul est pris en compte au niveau du moteur exécutif et n'implique pas de contraintes au niveau applicatif. Dans le cas d'un retrait, il est nécessaire de disposer d'un mécanisme de sauvegarde d'état du calcul.

4. Validation expérimentale

4.1. Présentation de l'application DOC-G

L'application servant aux expérimentations est une application en optimisation combinatoire pour la résolution du problème de l'affectation quadratique [18]. Cette application a été développée conjointement avec le PRiSM dans le cadre de l'ACI GRID DOC-G [9]. La méthode de calcul utilisée est une *méthode exacte* de calcul des solutions optimales du problème. L'algorithme est de type Branch&Bound qui permet de développer un arbre de recherche de manière efficace en permettant de couper des branches de l'arbre grâce à des estimations de bornes sur les solutions des sous-arbres à explorer.
Une instance typique du problème a été résolue en 2000 [2] en utilisant en moyenne environ 650 processeurs sur une semaine de calcul. Ce calcul aurait nécessité environ $12,5$ années de calcul sur un seul processeur équivalent : la parallélisation des méthodes de résolution pour ces problèmes est inévitable.
L'environnement de programmation parallèle utilisé est Athapascan pour l'API, associé à Kaapi comme moteur exécutif [15]. Une tâche de l'application consiste à évaluer un sous-arbre. Le parallélisme de l'application est récursif : chaque exécution d'une tâche amène à la création de nouvelles tâches. Le moteur exécutif utilise un algorithme de vol de travail basé sur le vol d'une tâche par un processus n'ayant plus de tâche à exécuter.
L'ajout de processeur est simplement géré comme l'ajout d'un processus qui commence à voler du travail. Cet algorithme équilibre bien la charge de calcul, qui est *a priori* irrégulière. Le retrait de ressources se base sur les techniques de sauvegarde / reprise d'état des processus comme décrit dans [15].

4.2. Déploiement adaptatif de l'application DOC-G

L'application DOC-G offre les deux propriétés requises pour un déploiement adaptatif : un degré de parallélisme dynamique et un mécanisme de sauvegarde de l'état du calcul. La première propriété est nécessaire pour pouvoir adapter dynamiquement le degré de parallélisme (i.e. le nombre de processus) ; la seconde est nécessaire pour le retrait de processus.
Pour mettre en œuvre le déploiement adaptatif de l'application DOC-G via Jade, celle-ci doit fournir l'interface d'administration uniforme d'un Managed Element (voir la Section 3). Pour ce faire, DOC-G est encapsulée dans un composant wrapper. Celui-ci traduit les opérations d'administration (ex. démarrer un processus de calcul) en opérations spécifiques à DOC-G (ex. appeler un script shell). Le wrapper modélise l'application DOC-G suivant deux niveaux : d'une part, les unités de base de l'application (i.e. processus de calcul parallèle) constituent le premier niveau ; et d'autre part, l'agrégation de ces unités en un ensemble représentant l'application répartie forme le second niveau. Cette modélisation fournit une interface de haut niveau pour l'administration de l'application DOC-G (ex. ajouter ou retirer dynamiquement des processus de calcul).
Un scénario de déploiement adaptatif de l'application DOC-G débute par son déploiement initial sur le sous-ensemble de tous les nœuds libres de la grille, celle-ci étant partagée avec d'autres applications. À mesure que les autres applications libèrent des nœuds, le gestionnaire de déploiement adaptatif en est informé par les sondes déployées sur la grille. L'application DOC-G étant paramétrée pour utiliser

un maximum de nœuds, le gestionnaire déclenche l'allocation dynamique des nœuds alors disponibles, déploie des processus de calcul sur chacun d'entres eux et les intègre à l'application DOC-G répartie. Symétriquement, à la réception de demandes de nœuds par d'autres applications, le gestionnaire de déploiement, qui en est également informé, choisit et libère autant de nœuds que nécessaire parmi ceux alloués à DOC-G. Cela est possible car DOC-G tolère la perte d'un ou plusieurs de ses nœuds.

4.3. Expérimentations futures

Le prototype de gestionnaire de déploiement adaptatif est actuellement en cours de validation. Les expériences futures sont prévues tout d'abord sur une architecture de grappe, puis ensuite sur une architecture de grille. Les premières expériences doivent permettre de mesurer la capacité du prototype à exploiter les ressources disponibles en permettant d'automatiser les différentes étapes du déploiement d'application comme décrit dans la section 2. Le critère mesuré lors de cette évaluation sera l'efficacité de l'utilisation des ressources sur la grappe, en tenant compte du début et de la fin des réservations dans le système. La seconde série d'expériences a pour objectif de mesurer la capacité du système à gérer le retrait des ressources.

5. Conclusion et perspectives

Dans cet article, nous avons exposé une approche basée sur le déploiement adaptatif d'une application sur une architecture parallèle de type grappe ou grille. À la différence d'autres outils, cette approche traite de bout en bout les différentes étapes nécessaires au déploiement d'une application. De plus, le déploiement est adaptatif en vu de réagir aux changements de disponibilité des ressources de calcul.
Ce travail est une étape nécessaire pour la construction d'un système autonome de déploiement d'application sur grappe et grille de calcul, l'objectif étant d'exploiter plus efficacement les ressources disponibles. Le retour des expériences permettra, d'une part, de mesurer le comportement du système Jade, pour le déploiement adaptatif sur une grappe et sur une grille, et d'autre part, d'estimer la capacité du système à passer à l'échelle.
Bien qu'imposant certaines contraintes sur les applications parallèles déployées, celles reposant sur un modèle de programmation à parallélisme implicite sont candidates à un déploiement adaptatif. La transformation d'une application parallèle à déploiement strict ou statique vers une application à déploiement adaptatif nécessite, d'une part, une description du parallélisme indépendante du nombre ou des caractéristiques des ressources, c'est-à-dire la conception d'algorithme adaptatif [8], et d'autre part, la possibilité de migrer des processus applicatifs et donc un certain niveau de virtualisation de l'architecture ou du système.

Bibliographie

1. Aaron (M.), Sanders (D.), Druschel (P.) et Zwaenepoel (W.). – Scalable Content-Aware Request Distribution in Cluster-Based Network Servers. *In : USENIX Annual Technical Conference*. – San Diego, CA, juin 2000.
2. Anstreicher (K. M.), Brixius (N. W.), Goux (J.-P.) et Linderoth (J.). – *Solving large quadratic assignment problems on computational grids*. – Rapport technique, Iowa City, Iowa 52242, 2000.
3. Bernholdt (D.), Bharathi (S.), Brown (D.), Chancio (K.), Chen (M.), Chervenak (A.), Cinquini (L.), Drach (B.), Foster (I.), Fox (P.), Garcia (J.), Kesselman (C.), Markel (R.), Middleton (D.), Pouchard (V. Nefedova L.), Shoshani (A.), Sim (A.), Strand (G.) et Williams (D.). – The earth system grid : Supporting the next generation of climate modeling research. *In : IEEE proceedings*, pp. 485—495. – march 2005.
4. Bianchini (R.) et Carrera (E. V.). – Analytical and Experimental Evaluation of Cluster-Based WWW Servers. *World Wide Web Journal*, vol. 3, n4, décembre 2000.
5. Bouchenak (S.), Boyer (F.), Hagimont (D.), Krakowiak (S.), Mos (A.), de Palma (N.), Quéma (V.) et Stefani (J. B.). – Architecture-Based Autonomous Repair Management : An Application to J2EE Clusters. *In : 24th IEEE Symposium on Reliable Distributed Systems (SRDS-2005)*. – Orlando, FL, octobre 2005.
6. Bruneton (E.), Coupaye (T.) et Stefani (J. B.). – Recursive and Dynamic Software Composition with

Sharing. *In : International Workshop on Component-Oriented Programming (WCOP-02)*. – Malaga, Spain, juin 2002. http ://fractal.objectweb.org.
7. Caron (E.) et Dail (H.). – *GoDIET : A Tool for Managing Distributed Hierarchies of DIET Agents and Servers*. – Rapport technique nRR-5520, INRIA, mars 2005.
8. Daoudi (M.), Gautier (T.), Kerfali (A.), Revire (R.) et Roch (J.-L.). – Algorithmes parallèles à grain adaptatif et applications. *Technique et Science Informatiques*, vol. 24, 2005, pp. 1—20.
9. Djerrah (A.), Cung (V.-D.), Jafar (S.), Gautier (T.) et Roch (J.-L.). – *Scalable and Fault-Tolerant Branch-and-bound for QAP with Bob++/Athapascan*. – Rapport technique n2005/85, PRiSM, 2005.
10. Dolstra, E. and de Jonge, M. and Visser, E. – Nix : A safe and policy-free system for software deployment. *In : 18th USENIX Large Installation System Administration Conference (LISA'04)*. – Atlanta, GA, novembre 2004.
11. Fox (A.), Gribble (S.), Chawathe (Y.) et Brewer (E. A.). – Cluster-Based Scalable Network Services. *In : 16th ACM Symposium on Operating Systems Principles (SOSP-97)*. – St. Malo, France, octobre 1997.
12. Grid'5000 Project. – Grid'5000. http ://www.grid5000.org/.
13. HP LabsD. – SmartFrog : Smart Framework for Object Groups. http ://www.hpl.hp.com/research/smartfrog/.
14. I. Foster and C. Kesselman. – The Globus Project : A Status Report. *In : Heterogeneous Computing Workshop*. – Orlando, FL, mars 1998.
15. Jafar, S. and Krings, A. W. and Gautier, T. and Roch, J.-L. – Theft-Induced Checkpointing for Reconfigurable Dataflow Applications. *In : IEEE Electro/Information Technology Conference (EIT 2005)*, éd. par IEEE. – Lincoln, Nebraska, May 2005. Best Paper Award.
16. Kadeploy Project. – Kadeploy. http ://www-id.imag.fr/Logiciels/kadeploy/.
17. Kephart (J. O.) et Chess (D. M.). – The Vision of Autonomic Computing. *IEEE Computer Magazine*, vol. 36, n1, 2003.
18. Koopmans (T.C.) et Beckmann (M.J.). – Assignment problems and the location of economic activities. *Econometrica*, vol. 25, 1957.
19. Load Sharing Facility. – Web Page, Platform Computing, Inc.
20. N. Capit and G. Da Costa and Y. Georgiou and G. Huard and C. Martin and G. Mounié and P. Neyron and O. Richard. – A Batch Scheduler with High Level Components. *In : Cluster computing and Grid 2005 (CCGrid05)*. – Cardiff, UK, mai 2005.
21. OARGrid Project. – OARGrid. http ://gforge.inria.fr/projects/oargrid/.
22. OMG C&D. – Deployment and Configuration of Component-based Distributed Applications Specification. Object Management Group Specification. http ://www.omg.org/docs/ptc/04-08-02.pdf.
23. Portable Batch System. – Web Page, Veridian Systems.
24. S. Lacour and C. Perez and T. Priol. – Deploying CORBA Components on a Computational Grid. *In : 2nd International Working Conference on Component Deployment (CD 2004)*. – Edinburgh, Scotland, UK, mai 2004.
25. YarKhan (A.), Seymour (K.), Sagi (K.), Shi (Z.) et Dongarra (J.). – Recent Developments in GridSolve. *International Journal of High Performance Computing Applications (Special Issue : Scheduling for Large-Scale Heterogeneous Platforms)*, vol. 20, n1, 2006.