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

Administration autonome de services Internet : Expérience avec l'auto-optimisation

Christophe Taton, Sara Bouchenak, Noël de Palma, Daniel Hagimont, Sacha Krakowiak, Jean Arnaud

INRIA Rhône-Alpes – Projet SARDES,
655 avenue de l'Europe
Montbonnot 38334 St Ismier Cedex
{Christophe.Taton,Sara.Bouchenak,Noel.Depalma,Sacha.Krakowiak,Jean.Arnaud}@inria.fr
Daniel.Hagimont@enseeiht.fr

Résumé
L'administration des systèmes répartis est une tâche de plus en plus complexe. Dans ce contexte, l'informatique autonome apporte une réponse intéressante. Elle vise, en effet, la construction de systèmes auto-configurables, auto-réparables, auto-optimisables et auto-adaptables.
Cependant, de nombreux problèmes restent ouverts, parmi lesquels (i) l'administration de systèmes patrimoniaux et (ii) l'administration de systèmes répartis présentant des architectures complexes. Dans cet article, nous traitons précisément ces deux points en présentant Jade, une plate-forme d'administration autonome de systèmes répartis patrimoniaux et complexes.
Nous nous intéressons, plus particulièrement, à l'auto-optimisation des systèmes qui consiste à maintenir les performances d'un système stables face à des variations de la charge du système. La principale contribution de cet article est la définition d'un modèle architectural générique qui facilite l'administration de divers systèmes patrimoniaux et de diverses architectures de systèmes répartis. Nous illustrons l'application de ce modèle à travers l'auto-optimisation de services Internet J2EE répartis complexes.

Mots-clés : Gestion autonome, Systèmes patrimoniaux, Auto-optimisation, Services Internet, Architecture J2EE multi-tiers

1. Introduction

Un des modèles standards de construction de services Internet est le modèle multi-niveaux (i.e. *multi-tiers*) où chaque niveau est reponsable d'une fonction particulière de l'application, par exemple le niveau présentation géré par un serveur web, la logique métier prise en charge par un serveur d'entreprise et la persistance gérée par un serveur de base de données. De plus, pour des besoins de disponibilité et de scalabilité, les différents niveaux de ces applications peuvent être dupliqués, rendant ainsi l'architecture logicielle encore plus complexe. Dans ce contexte, l'administration de tels systèmes devient une tâche ardue et est particulièrement sensible aux erreurs humaines [9]. L'administration autonome (*Autonomic Computing*) est une réponse intéressante à ce problème [8]. Elle a pour objectif de fournir des outils de haut niveau qui permettent d'automatiser le déploiement et la (re)configuration des systèmes informatiques, permettant ainsi aux systèmes de s'auto-adapter face aux défaillances, aux variations de charge, etc. Cependant, de nombreux défis persistent dans le domaine de l'administration autonome, parmi lesquels : (i) l'administration de logiciels patrimoniaux et (ii) l'administration de systèmes répartis présentant des architectures complexes.
Logiciels patrimoniaux. Le terme «logiciel patrimonial» désigne ici une application vue comme une boîte noire, uniquement accessible via son interface applicative et sur laquelle aucune hypothèse n'est faite. Les solutions d'administration de logiciels patrimoniaux sont souvent des solutions ad-hoc dépendant d'un logiciel patrimonial particulier, par exemple un serveur de mail [13] ou un serveur web [6]. Avec une telle approche, il est nécessaire de réimplanter la solution d'administration autonome pour chaque nouveau logiciel patrimonial administré, même si les techniques sous-jacentes à l'administration autonome peuvent être similaires pour différents systèmes administrés.

Architectures complexes. L'architecture des systèmes répartis est de plus en plus complexe. Par exemple, les services Internet actuels sont généralement mis en œuvre au moyen d'une architecture multi-niveaux, où chaque niveau peut lui-même être dupliqué sur plusieurs serveurs. Les solutions d'administration des systèmes sont généralement spécialisées pour des architectures simples (par exemple, un serveur web dupliqué), ou présentent une approche ad-hoc qui est spécifique et dépendante du système administré [18].

Dans cet article, nous apportons une réponse aux deux problèmes précédents à travers la définition d'une plate-forme d'administration qui met en œuvre :

– Des éléments administrables (*Managed Elements* ou ME). De tels éléments exposent une interface d'administration uniforme quel que soit le système patrimonial sous-jacent. Ainsi, différents systèmes patrimoniaux administrés offrent une vue homogène et permettent de mettre en place des politiques d'administration de manière générique. Ceci est rendu possible grâce à l'encapsulation des systèmes patrimoniaux dans des composants logiciels particuliers appelés *wrappers*.
– Des vues architecturales évoluées du système administré. Le système administré est représenté par un assemblage de ME, via des liaisons ou des compositions de plusieurs ME. Ceci permet de refléter l'architecture globale du système administré, aussi complexe soit-elle, et ainsi de proposer une approche générique pour l'administration de systèmes à architectures complexes.

Dans ce papier, nous décrivons la conception et la mise en œuvre de la plate-forme d'auto-administration Jade. Cette plate-forme inclut des gestionnaires autonomes (*Autonomic Managers* ou AM) dont le rôle est de prendre en charge un aspect particulier de l'administration, tel que l'optimisation de performances, la tolérance aux pannes ou la protection contre les attaques. Nous décrivons plus précisément l'auto-optimisation de systèmes répartis dupliqués pour l'adaptation aux variations de la charge des systèmes [1]. Nous utilisons ici des techniques de redimensionnement dynamique du système administré. Nous avons évalué la plate-forme Jade avec un service Internet de vente aux enchères (à la eBay) ; ce service est basé sur une architecture J2EE répartie complexe (multi-niveaux et dupliquée), utilisant divers systèmes patrimoniaux (serveur web, serveur d'entreprise, serveur de base de données). Les résultats de l'évaluation expérimentale montrent que lorsque la charge du service Internet augmente, le temps de réponse moyen des requêtes des clients du service se dégradent significativement en l'absence d'auto-optimisation (10,42 s en moyenne), alors que dans le cas où Jade est utilisé, les performances du service restent stables (temps de réponse client moyen de 950 ms). Par conséquent, lorsqu'il est auto-optimisé, le service Internet peut traiter deux fois plus de requêtes clientes que le même service non administré.

La suite de cet article est organisée comme ceci. La Section 2 rappelle le contexte de ce projet. Les Sections 3 et 4 décrivent respectivement la conception et la mise en œuvre de l'auto-optimisation dans Jade. Les résultats de l'évaluation expérimentale sont ensuite présentés en Section 5. La Section 6 décrit l'état de l'art et la Section 7 présente nos conclusions.

2. Contexte expérimental : Services Internet J2EE multi-niveaux dupliqués

L'environnement expérimental que nous avons choisi repose sur la technologie J2EE [15] qui définit un modèle de développement d'applications réparties. Ce modèle s'appuie sur une architecture multi-niveaux généralement constituée d'un serveur web gérant la présentation de l'application, d'un serveur d'entreprise réalisant la logique métier de l'application et d'un serveur de bases de données pour assurer la persistance des données. Par ailleurs, pour fournir une plus haute disponibilité et scalabilité du service, les différents niveaux de ce modèle peuvent être dupliqués. Parmi les solutions de duplication J2EE existantes, il y a C-jdbc pour la duplication de serveurs de bases de données [7], JBoss clustering pour les serveurs d'entreprise à base d'EJB JBoss [5], mod_jk pour les serveurs d'entreprise à base de Servlets Tomcat ou le switch L4 pour les serveurs web Apache (voir la Figure 1).

Ainsi, les systèmes J2EE multi-niveaux dupliqués constituent un environnement expérimental intéressant du point de vue de l'administration autonome et en particulier, pour l'auto-optimisation. En effet, ces systèmes regroupent l'ensemble des défis énoncés précédemment, à savoir : l'administration de logiciels patrimoniaux, chaque niveau d'un système J2EE étant mis en œuvre au moyen de logiciels spécialisés, et l'administration d'un système présentant une architecture répartie complexe multi-niveaux

[1] Les détails concernant l'auto-réparation dans Jade sont décrits dans [4].

et dupliquée.

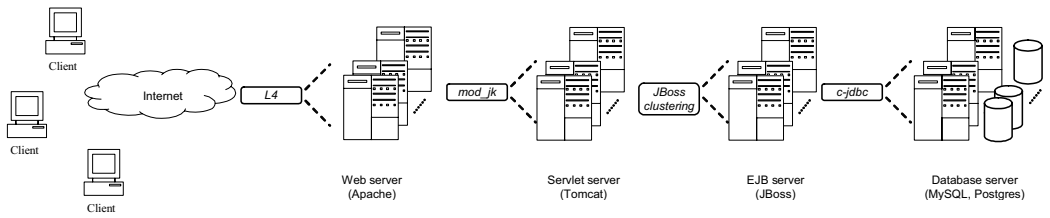


FIG. 1 – Architecture de Services Internet J2EE multi-niveaux et dupliqués

3. Principes de conception de l'administration autonome

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). La solution d'administration autonome proposée est basée sur deux principes de conception : une approche générique pour l'administration de systèmes patrimoniaux et une approche générique pour l'administration d'architectures complexes.

3.1. Approche générique pour systèmes patrimoniaux

Pour fournir une vue homogène de différents systèmes patrimoniaux 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.

3.2. Approche générique pour architectures complexes

Pour proposer une approche générique d'administration de systèmes formant des architectures complexes, nous définissons des éléments administrables architecturaux (*Architectural Managed Element* ou AME). Nous identifions plus précisément divers types d'AME modélisant des patrons de conception courants dans les systèmes répartis. Par exemple, un CoME (*Container Managed Element*) représente un élément *conteneur* qui contient d'autres ME ; un CoME peut, par exemple, servir à modéliser une pile logicielle constituée d'une machine contenant un intergiciel (*middleware*) qui lui, héberge une application (voir la Figure 2, axe vertical). Un CluME (*Cluster ME*) représente une grappe de ME dupliqués (voir la Figure 2, axe diagonal). Un CSME (*Client-Server ME* modélise deux ME où l'un est client et l'autre serveur. Et enfin le MuTiME (*Multi-Tier ME*) est une combinaison de plusieurs CSME pour constituer une architecture multi-niveaux (voir la Figure 2, axe horizontal). Par ailleurs, il est possible de définir des AME plus complexes par assemblage de plusieurs AME. Ainsi, nous pouvons de manière simple et générique représenter des systèmes administrés avec des architectures réparties complexes.

Ainsi, ces différents modèles nous permettent de représenter des systèmes à architecture répartie complexe comme un assemblage hiérarchique de sous-systèmes. Ceci facilite les opérations d'administration de tels systèmes.

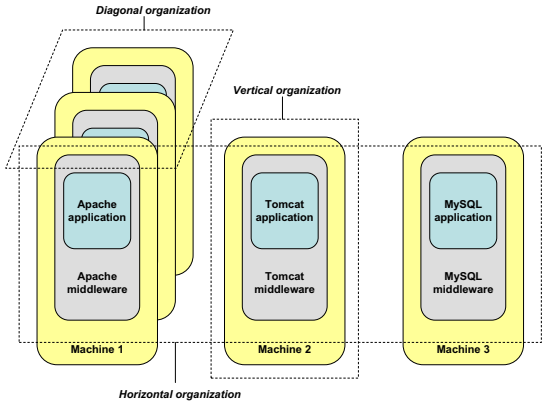


FIG. 2 – Différents modèles d'architectures

4. Mise en œuvre de l'auto-optimisation

Dans cette section, nous décrivons la réalisation d'un gestionnaire d'auto-optimisation dans Jade. Ce gestionnaire est chargé de l'administration d'un ensemble d'éléments dupliqués responsables de la fourniture d'un service, par exemple, une grappe de serveurs web dupliqués. Pour ce type de service, nous constatons que plus le nombre de machines constituant la grappe est élévé, plus les performances du service sont bonnes (ex. latence réduite des requêtes des clients web ou débit élevé des serveurs web). Mais ceci est fait au détriment d'une forte consommation de ressources matérielles. Inversement, plus la consommation de ressources est économe, plus les performances du service sont faibles. Il y a donc un compromis à trouver entre, d'une part, la quantité de ressources nécessaires à un service et, d'autre part, les performances de ce service. Classiquement, l'administrateur humain d'un tel système teste manuellement, pour une charge particulière du système, plusieurs configurations de ce système avant de trouver la plus appropriée. Ceci reste une tâche fastidieuse qui doit être répétée à chaque modification de la charge du système.

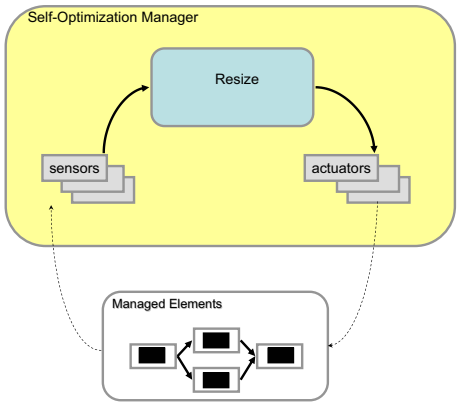


FIG. 3 – Auto-optimisation dans Jade

Le gestionnaire d'auto-optimisation proposé a pour objectif de reconfigurer automatiquement un système face aux variations dynamiques de charge, en lui ré-allouant le nombre nécessaire et suffisant de ressources qui lui garantit des performances optimales. Ce gestionnaire est basé sur une boucle de contrôle constituée : (i) de sondes responsables de la détection de changement de charge, (ii) d'un régulateur responsable de l'analyse des informations produites par les sondes et de la prise de décision concernant la reconfiguration éventuelle à faire et (iii) d'actionneurs commandés par le régulateur pour effectivement réaliser les opérations de reconfiguration (voir la Figure 3). Les actions entreprises par ce gestionnaire reposent sur divers services généraux fournis par Jade, parmi lesquels un service de réservation de ressources ou encore un service de déploiement et d'installation.
Ici, le système administré est vu comme un ensemble d'éléments dupliqués redimensionnable, dans le sens où le nombre d'éléments constituant cet ensemble peut dynamiquement varier. Pour cela, nous avons défini une variation du CluME pour laquelle il est possible d'ajouter/supprimer des éléments dynamiquement. Ainsi, dans la Figure 3, le système admnistré par le gestionnaire d'auto-optimisation est un CluME redimensionnable. Ce composant est associé à des sondes qui mesurent les performances du système administré en termes, par exemple, de latence des requêtes clientes ou de taux d'utilisation des processeurs. Quant aux actionneurs du composant redimensionnable, ce sont les opérations qui permettent d'ajouter ou de supprimer des éléments dupliqués du composant. Enfin, le régulateur de la boucle de contrôle du gestionnaire d'auto-optimisation met en oeuvre un algorithme donné d'auto-optimisation. Dans la suite, nous décrivons différents algorithmes d'auto-optimisation.

Algorithme d'auto-optimisation pour variations graduelles de charge.

Ce premier algorithme permet de répondre à des *variations graduelles* de la charge du système. Il est basé sur un seuil minimum et un seuil maximum entre lesquels il maintient la charge du système administré. Lorsque les sondes de charge indiquent une charge sortant de cet intervalle, une reconfiguration est déclenchée pour ajouter ou retirer un élément dupliqué. Par ailleurs, pour limiter les oscillations possibles après une reconfiguration, toute reconfiguration supplémentaire est inhibée pendant une courte durée. Cet algorithme a, par exemple, été utilisé pour administrer une grappe de serveurs dupliqués (serveurs de bases de données, serveurs d'entreprise).

Algorithme d'auto-optimisation pour variations brutales de charge.

Avec la politique d'auto-optimisation précédente, lorsqu'une augmentation ou baisse de charge est détectée, une reconfiguration unitaire est opérée (ajout ou retrait d'un seul nœud à la fois). Cette politique convient aux augmentations graduelles de charge, mais dans le cas d'une augmentation brutale de charge (par exemple, un pic de charge dû à une période particulière de l'année), la politique précédente déclencherait une succession de reconfigurations unitaires, ce qui correspondrait, par exemple, à un démarrage en série de plusieurs serveurs supplémentaires.

Il serait, dans ce cas, plus judicieux de démarrer les serveurs supplémentaires en parallèle, sous la forme d'une reconfiguration globale unique et donc plus rapide. Nous proposons ici une politique d'auto-optimisation qui suit cette approche. Toute la difficulté ici est de pouvoir calculer, lors d'une évolution de charge, la quantité de ressources (i.e. nœuds) nécessaires à ajouter ou retirer. Pour cela, nous avons constaté qu'un système dupliqué a des besoins en ressources proportionnels à la quantité de travail qu'il doit traiter. Nous introduisons ainsi un indicateur de la quantité de travail (ex. le nombre de requêtes en cours de traitement dans un serveur). Lorsque les sondes de charge détectent une variation de la charge, connaissant le nombre de requêtes à traiter, nous en déduisons la quantité absolue de ressources nécessaires, et donc le nombre de nœuds impliqués dans une reconfiguration globale.

Algorithme d'auto-optimisation à grain plus fin.

Les deux techniques d'auto-optimisation précédentes s'intéressent à l'administration des ressources à la granularité du nœud et considèrent la ressource serveur au sein d'une grappe de serveurs. Nous cherchons ici à raffiner la granularité des ressources administrées, et nous nous intéressons à la ressource processus au sein d'un serveur unique multi-processus.
Nous souhaitons plus particulièrement contrôler le taux de parallélisme d'un serveur, c'est-à-dire le nombre maximum de processus qui participent simultanément au traitement des travaux d'un serveur. Cela revient, par exemple, pour un serveur de bases de données, à contrôler le nombre maximum de ses processus actifs. Nous agissons ainsi sur le nombre maximum de requêtes traitées simultanément dans le serveur de bases de données, ce qui correspond à son taux de parallélisme.
Le taux de parallélisme d'un serveur modifie ses performances. En effet, pour une charge particulière du serveur, plus le taux de parallélisme est élevé, plus le débit en requêtes du serveur sera important, et ce, au détriment de la latence des requêtes. À l'inverse, plus le taux de parallélisme est faible, plus la latence des requêtes sera faible, au détriment du débit en requêtes du serveur. Il y a donc un compromis à trouver sur le taux de parallélisme pour minimiser la latence tout en maximisant le débit de requêtes. De plus, lorsque le niveau de charge du serveur évolue, ce compromis doit être réévalué pour prendre en compte les nouvelles valeurs de latence et de débit imposées par cette charge. Nous proposons ici l'automatisation de ces calculs via une boucle de contrôle reposant sur des sondes mesurant les valeurs de débit et de latences des requêtes du serveur. Le régulateur, à partir de ces informations, décide d'augmenter, de réduire ou de conserver le taux de parallélisme du serveur. Par exemple, si les latences des requêtes d'un serveur de bases de données sont trop importantes, il convient de réduire le taux de parallélisme afin de diminuer les latences des requêtes.

5. Évaluation

Cette section décrit tout d'abord la plate-forme d'évaluation avant de présenter les résultats d'évaluation qualitative et quantitative de Jade.

5.1. Plate-forme d'évaluation

Pour valider l'auto-optimisation proposée par Jade, nous avons utilisé l'application Rubis qui modélise un site web de ventes aux enchères à la eBay [1]. Cette application suit le modèle J2EE à trois niveaux et est ainsi constituée de trois parties d'application : une «application web» sous la forme de pages HTML du site, une «application d'entreprise» constituée de l'ensemble des Servlets Java responsables de l'exécution de la logique métier du site et une application de base de données représentant l'ensemble des tables de la base de données stockant les informations de ce site.
L'application Rubis est déployée sur les intergiciels J2EE suivants : un serveur web Apache, un serveur d'entreprise Jakarta Tomcat et un serveur de bases de données Mysql. De plus, chaque étage est dupliqué pour former, respectivement, une grappe de serveurs web dupliqués à l'aide de PLB, une grappe de serveurs d'entreprise dupliqués via la solution de clustering Tomcat et une grappe de serveurs de bases de données dupliqués avec l'outil C-jdbc [11].

5.2. Évaluation qualitative

La plate-forme d'administration autonome Jade a pour objectif une gestion générique de différents logiciels patrimoniaux administrés et de systèmes dotés d'architectures réparties complexes. Dans cette

section, nous évaluons la généricité fournie par Jade.
Le site Rubis déployé sur un système J2EE multi-niveaux dupliqué est modélisé, dans Jade, par les *wrappers* suivants : un ME pour chaque machine hébergeant ce site, un ME pour chaque intergiciel s'exécutant sur une machine (i.e. Apache, Tomcat, Mysql), un ME pour chaque application hébergée par un intergiciel (i.e. applications web, d'entreprise et de base de données), un CoME pour chaque organisation en pile du type machine/intergiciel/application, un CluME pour chaque grappe d'éléments CoME dupliqués et un MuTiME pour une organisation multi-niveaux de trois éléments CluME.
La Table 1 compare les proportions de code générique et de code spécifique dans Jade. En particulier, Jade est constitué de mécanismes génériques applicables à divers logiciels patrimoniaux tels que, par exemple, le gestionnaire d'auto-optimisation, les AME ou les services de réservation de noeuds, d'installation de logiciels ou de déploiement. Quant au code spécifique impliqué dans nos expérimentations, il est constitué de *wrappers* spécialisant les patrons de conception généraux définis dans Jade comme, par exemple, la spécialisation du CluME pour représenter une grappe d'éléments web dupliqués avec Plb. En résumé, l'intégration d'un nouveau système patrimonial dans Jade nécessite l'écriture d'un *wrapper* constitué seulement de 477 lignes de code Java et 16 lignes d'ADL en moyenne [2].

		# Java classes	**Java code size**	**# ADL files**	**ADL file size**
Code générique	Self-Optimization Manager	14	2340 lines	12	150 lines
	Architectural MEs	4	840 lines	4	200 lines
	Reservation Service	2	475 lines	1	30 lines
	Software Installation Service	3	430 lines	9	830 lines
	Deployment Service	21	2600 lines	9	830 lines
	Total	***44***	***6685 lines***	***35***	***2040 lines***
Code spécifique	Rubis web application	1	150 lines	1	11 lines
	Rubis enterprise application	2	150 lines	1	11 lines
	Rubis database application	1	150 lines	1	11 lines
	Apache web middleware	3	800 lines	1	16 lines
	Tomcat enterprise middleware	3	550 lines	1	12 lines
	Mysql database middleware	4	760 lines	3	40 lines
	Plb web clustring	2	460 lines	1	14 lines
	Tomcat enterprise clustring	2	460 lines	1	14 lines
	C-jdbc database clustering	1	810 lines	1	14 lines
	Total	***19***	***4290 lines***	***11***	***143 lines***
	Moyenne	***2***	***477 lines***	***1***	***16 lines***

TAB. 1 – Code générique vs. code spécifique dans Jade

Alors que si ce même système n'est pas géré de manière générique tel que fait par Jade, la réalisation de son administration nécessiterait la réécriture de versions spécifiques du gestionnaire d'auto-optimisation pour prendre en compte les différents logiciels patrimoniaux, ce qui correspond à environ 2340 lignes de code Java.

5.3. Évaluation quantitative

5.3.1. Environnement expérimental

L'application Rubis utilisée dans nos expérimentations est accompagnée d'un émulateur de clients web qui permet de générer une certaine charge. L'environnement d'expérimentation considéré est constitué des logiciels suivants : le serveur web Apache 1.3.29, le serveur d'entreprise Jakarta Tomcat 3.3.2, le serveur de bases de données Mysql 4.0.17, l'outil de duplication web Plb 0.3, le mécanisme de duplication Tomcat 3.3.2 et l'outil de duplication C-jdbc 2.0.2. L'environnement matériel est constitué de machines Linux x86, 1.8 GHz/1 Go, interconnectées par un réseau Ethernet 100 Mbits.

5.3.2. Résultats expérimentaux

Dans cette section, nous présentons les résultats de l'évaluation des différentes politiques d'auto-optimisation mises en œuvre dans Jade.

[2] ADL - *Architecture Description Language* ou fichier de configuration de composants Fractal.

Auto-optimisation pour évolution graduelle de charge

Dans cette première expérimentation, nous avons considéré le scénario dans lequel le système est initialement constitué d'un serveur d'entreprise et d'un serveur de bases de données. Le système est successivement soumis à une augmentation puis un réduction graduelle de la charge (de 80 à 500 clients web). À mesure que la charge augmente, le niveau bases de données puis le niveau entreprise du système saturent, occasionnant les ajouts successifs de deux nouveaux nœuds pour des serveurs de bases de données dupliqués, puis d'un nœud pour un serveur d'entreprise dupliqué. Ensuite, la charge baisse graduellement. Les serveurs d'entreprise puis les serveurs de bases de données sont alors progressivement déchargés, ce qui entraîne le retrait d'éléments dupliqués du niveau entreprise puis du niveau bases de données. La Figure 4 résume ce scénario en décrivant l'impact de la variation de la charge sur le nombre d'éléments dupliqués.

Dans cette expérimentation, le processeur est l'unique ressource limitante [3]. L'auto-optimisation de ce système se base donc sur des sondes qui mesurent périodiquement l'utilisation des processeurs par les éléments dupliqués (i.e. serveurs d'entreprise, serveurs de bases de données). Une sonde spécialisée agrège l'ensemble des mesures relatives à un niveau (i.e. entreprise ou bases de données) pour calculer une moyenne spatiale et temporelle [4].

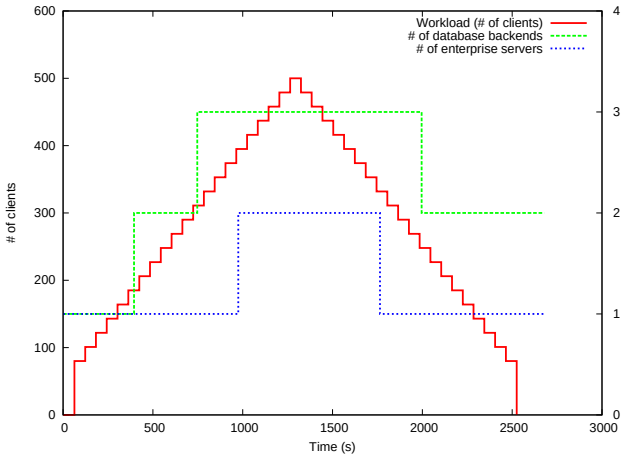


FIG. 4 – Charge vs. degrés de duplication des serveurs

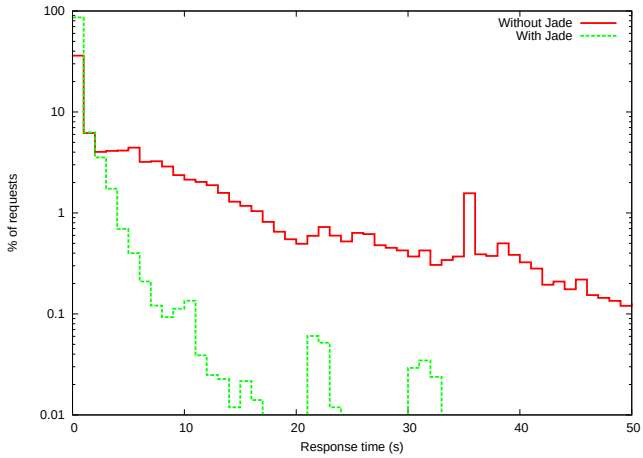


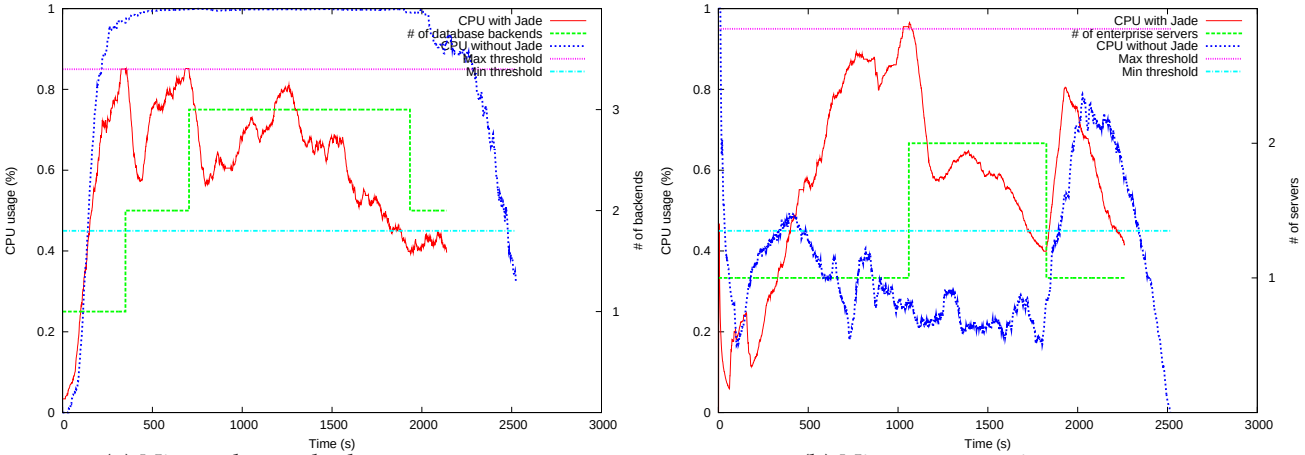
FIG. 5 – Répartition des latences des requêtes clientes avec et sans Jade

Dans la suite, nous comparons le système décrit précédemment dans deux cas de figure : lorsqu'il est auto-optimisé avec Jade et sans l'utilisation de Jade. Sur la Figure 6-(a), lorsque le CPU atteint le seuil maximum autorisé pour le niveau bases de données, Jade alloue dynamiquement un nouveau nœud, régulant ainsi la charge. En l'absence de Jade, la ressource CPU finit par saturer et le niveau bases de données s'écroule tant que la charge reste élevée. La Figure 6-(b) présente des résultats similaires relatifs au niveau entreprise. Lorsque Jade est utilisé, la régulationde la charge s'effectue comme convenu. Notons qu'en l'absence de Jade, l'indice d'utilisation CPU reste faible malgré la charge élevée. Cela s'explique par l'écroulement du serveur de bases de données qui induit des temps de traitement très longs sur le serveur de bases de données et place ainsi le serveur d'entreprise en attente. En effet, vers la fin de l'expérience, l'utilisation CPU du serveur d'entreprise remonte lorsque la saturation du serveur de bases de données cesse.

[3] Si le processeur n'est pas l'unique ressource critique, on peut construire des indicateurs plus sophistiqués par agrégation de différents types de sondes.

[4] Sur l'ensemble des éléments dupliqués du CluME, et sur une fenêtre temporelle glissante de 60 secondes.

La Figure 5 montre un histogramme représentant la répartition des latences des requêtes clients parmi l'ensemble des requêtes traitées lors des exécutions avec et sans Jade. La proportion des requêtes traitées en moins d'une seconde est de 86% avec Jade contre 36% sans Jade, de celles traitées dans un délai compris entre 1 et 20 secondes atteint 13% avec Jade contre 36% en son absence, et enfin de celles traitées en plus de 40 secondes est nulle avec Jade alors qu'elle atteint encore 5.5% sans Jade. Ainsi, le système administré par Jade a été capable de traiter 92512 requêtes contre 41585 en l'absence de Jade, soit moins de la moitié.



(a) Niveau bases de données (b) Niveau entreprise

FIG. 6 – Auto-optimisation du système en présence d'une variation graduelle de la charge

Auto-optimisation pour évolution brutale de charge

Nous considérons ici le second algorithme d'auto-optimisation qui vient compléter le précédent en permettant l'ajout et le retrait simultané de plusieurs éléments dupliqués du système administré. Cela permet de mieux répondre à des *variations brutales* de la charge correspondant à des pics de charge. La difficulté réside alors dans l'estimation du nombre d'éléments devant être rajoutés ou retirés. Pour cela, en plus de l'indicateur d'utilisation CPU qui permet de détecter le changement de charge, nous utilisons un indicateur de nombre de requêtes en cours de traitement sur les serveurs pour calculer le nombre de nœuds impliqués dans une reconfiguration globale. Nous avons en effet observé que la quantité de ressources (ie. nœuds) à mobiliser est une fonction linéaire du nombre de requêtes à traiter par l'application que nous avons utilisée pour l'évaluation. La Figure 7 décrit les résultats d'une expérimentation basée sur un système J2EE initialement constitué d'un serveur d'entreprise et d'un serveur de bases de données, avec un charge initiale de 100 clients web. Un pic de charge survient (+400 clients) et nous comparons alors le système auto-optimisé et le même système non administré, en termes de latence des requêtes clientes.

Après l'occurrence du pic de charge, le système basé sur Jade évolue directement vers une configuration où le niveau bases de données est constitué de 3 serveurs, ce qui permet d'absorber la charge et d'assurer une continuité de service, tandis que le système non administré par Jade s'écroule. Nous observons effectivement que le niveau bases de données s'écroule juste après le pic de charge, avec ou sans Jade (c.f. augmentation de la latence des requêtes). Toutefois, avec Jade, l'écroulement du serveur de bases de données est limité par la reconfiguration, ce qui explique les différences entre les latences mesurées dans les cas avec et sans Jade. [5].

[5] L'axe horizontal de la Figure 7 donne les instants de soumission et non de terminaison des requêtes.

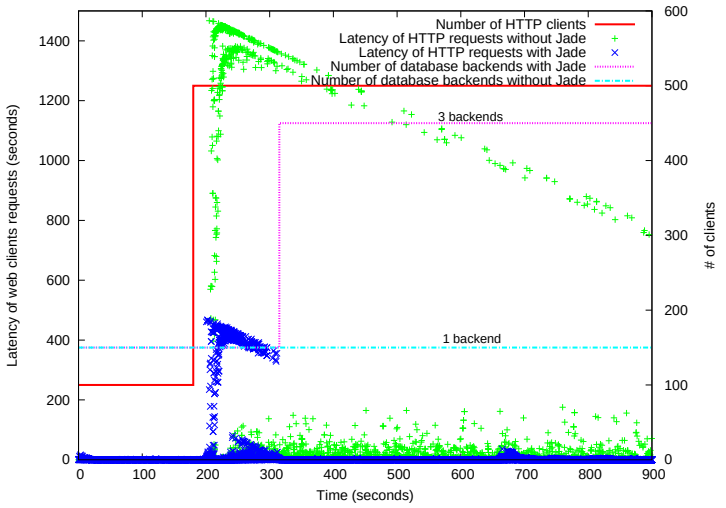


FIG. 7 – Comportement de l'étage de données en présence de pics de charge

Auto-optimisation à grain plus fin

Nous présentons maintenant l'évaluation de l'algorithme d'auto-optimisation considérant les ressources à la granularité du processus. Cette expérimentation s'appuie sur le banc d'essai TPC-C qui met en œuvre une base de données associée à un jeu de transactions pour modéliser les diverses activités d'un fournisseur pour la gestion, vente et distribution de produits ou de services. Ce banc d'essai fournit également un émulateur de clients qui permet de générer une charge paramétrable et de collecter des informations de performances sur le serveur de bases de données. Pour valider notre approche, nous déployons l'application TPC-C sur un serveur de bases de données multi-processus Postgres (version 8.1.3) et nous mesurons les débits et latences des requêtes clients générées par l'émulateur sur la base de données, en fonction, d'une part, du nombre de clients émulés, et d'autre part, du taux de parallélisme du serveur de bases de données. Les travaux préliminaires concernant l'auto-optimisation à grain plus fin sont actuellement en cours de réalisation et devrait nous permettre de conclure sur la validité de notre approche consistant à adapter dynamiquement le taux de parallélisme d'un serveur en fonction de la charge.

Sur-coût sur les performances

Nous nous intéressons ici à l'évaluation de l'éventuel sur-coût sur les performances du système administré lié à l'utilisation de la plate-forme d'auto-optimisation Jade. Pour ce faire, nous avons mesuré différents paramètres du système J2EE utilisé, dans un cas sans Jade, et, dans un autre cas, avec Jade, mais sans opération d'auto-administration. La Table 2 présente les valeurs mesurées de ces paramètres en termes de débit et de latences des requêtes web et d'utilisation CPU et mémoire.

Les taux d'utilisation CPU et mémoire correspondent à des moyennes sur l'ensemble des machines du système. Les mesures réalisées ne montrent pas de surcoût significatif au niveau des débits, latences et utilisation CPU. En revanche, l'espace mémoire utilisé est légèrement supérieur lorsque Jade est présent (20.1% vs. 17.5%) ; cela correspond aux composants internes à Jade déployés sur les machines du système.

	Débit (req. web/s)	Latence req. web (ms)	CPU (%)	Mémoire (%)
avec Jade	12	89	12.74	20.1
sans Jade	12	87	12.42	17.5

TAB. 2 – Surcoût

6. État de l'art

Nombre de travaux s'intéressent à l'administration des grappes de machines qui, pour des raisons économiques, sont partagées entre plusieurs applications. Une répartition statique des ressources de la grappe conduit à un gaspillage. C'est pourquoi ces travaux visent à fournir des mécanismes d'allocation dynamique.

Une partie de ces travaux repose sur la disponibilité de tous les composants logiciels sur l'ensemble des machines administrées. Dans cette configuration, l'administration des ressources s'appuie sur un routage adapté des requêtes vers les différents nœuds (voir Neptune [14] et DDSD [19]). D'autres travaux reposent sur un contrôle avancé de la répartition du temps CPU pour fournir des garanties sur les allocations de ressources (voir Cluster Reserves [3] et Sharc [17]).

L'autre partie de ces travaux considère l'administration des ressources à la granularité du nœud (isolant ainsi les applications, d'un point de vue sécurité). L'administration des ressources repose alors sur l'allocation dynamique de nœuds et le déploiement dynamique d'applications sur ces nœuds. On peut citer dans cette catégorie les projets Oceano [2], QuID [12], OnCall [10] et Cataclysm [16, 18].

La plate-forme Jade décrite dans ce papier tombe dans la seconde catégorie, en présentant une isolation forte entre les applications administrées et une administration des ressources à la granularité du nœud. Jade se distingue cependant sur plusieurs points.

Systèmes patrimoniaux. La plate-forme Jade permet d'abstraire tout système sous forme d'un composant administrable exhibant une interface d'administration uniforme. L'administration de systèmes patrimoniaux est ainsi rendue générique vis-à-vis des logiciels sous-jacents.

Architectures complexes. Les expérimentations réalisées sur l'auto-optimisation d'architectures J2EE multi-niveaux démontrent l'apport de Jade dans l'administration de systèmes répartis à architecture complexe alors que la plupart des travaux se cantonnent au simple modèle client-serveur.

7. Conclusion

Les applications web actuelles reposent généralement sur des systèmes multi-niveaux dupliqués, pour des raisons de scalabilité et de continuité de service. L'administration de tels systèmes reste une tâche complexe et fastidieuse. L'administration autonome apporte une réponse intéressante à ce problème, à travers la construction de systèmes auto-configurables, auto-optimisables et auto-adaptables. Cependant, deux problèmes restent ouverts : l'administration de systèmes patrimoniaux et l'administration d'architectures complexes. Dans cet article, nous avons précisément apporté une réponse à ces problèmes à travers une approche générique d'auto-administration de systèmes patrimoniaux avec architectures complexes. Nous nous sommes, plus particulièrement, intéressés à l'auto-optimisation de ces systèmes.

Nous avons ainsi conçu et mis en œuvre la plate-forme d'administration autonome Jade et, plus précisément, un gestionnaire d'auto-optimisation qui met en œuvre divers algorithmes d'optimisation. Ces algorithmes ont pour objectif de garantir des performances stables et acceptables dans différents cas : des montées graduelles de charge de l'application ou des montées brutales de cette charge, tout en considérant des granularités plus ou moins fines de ressources administrées. Nous avons illustré la validité de la solution proposée à travers une application de commerce électronique basée sur le modèle J2EE multi-niveaux et dupliquée sur des grappes de serveurs. Les résultats de l'évaluation expérimentale confirment la validité de l'approche, à la fois en termes de garantie de performances mais aussi en termes de généricité.

Plusieurs perspectives sont envisageables suite à ces premières réalisations et expérimentations. La première concerne l'application de la solution d'auto-optimisation proposée à d'autres domaines applicatifs tels que les systèmes sur grilles ou les environnements pair à pair. La seconde persceptive concerne l'étude de la coordination de plusieurs gestionnaires autonomes qui cohabitent dans Jade (par exemple, un gestionnaire d'auto-optimisation et un gestionnaire de tolérance aux pannes) ; ceci servirait à minimiser les interférences dues à des reconfigurations concurrentes commandées par différents gestionnaires.

Bibliographie

1. Amza (C.), Cecchet (E.), Chanda (A.), Cox (A.), Elnikety (S.), Gil (R.), Marguerite (J.), Rajamani (K.) et Zwaenepoel (W.). – Specification and Implementation of Dynamic Web Site Benchmarks. *In : IEEE 5th Annual Workshop on Workload Characterization (WWC-5)*. – Austin, TX, novembre 2002.
2. Appleby (K.), Fakhouri (S.), Fong (L.), Goldszmidt (G.) et Kalantar (M.). – Oceano - SLA based management of a computing utility. *In : 7th IFIP/IEEE International Symposium on Integrated Network Management*. – Seattle, WA, mai 2001.
3. Aron (M.), Druschel (P.), et Zwaenepoel (W.). – Cluster Reserves : a mechanism for resource management in cluster-based network servers. *In : International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS-2000)*. – Sant Clara, CA, juin 2000.
4. Bouchenak (S.), Boyer (F.), Hagimont (D.) et Krakowiak (S.). – Architecture-Based Autonomous Repair Management : An Application to J2EE Clusters. *In : 24th IEEE Symposium on Reliable Distributed Systems (SRDS-2005)*. – Orlando, FL, octobre 2005.
5. Burke (B.) et Labourey (S.). – Clustering With JBoss 3.0. octobre 2002. – http ://www.onjava.com/pub/a/onjava/2002/07/10/jboss.html.
6. Candea (G.), Kawamoto (S.), Fujiki (Y.), Friedman (G.) et Fox (A.). – A Microrebootable System : Design, Implementation, and Evaluation. *In : 6th Symposium on Operating Systems Design and Implementation (OSDI-2004)*. – San Francisco, CA, décembre 2004.
7. Cecchet (E.), Marguerite (J.) et Zwaenepoel (W.). – C-JDBC : Flexible Database Clustering Middleware. *In : USENIX Annual Technical Conference, Freenix track*. – Boston, MA, juin 2004. http ://c-jdbc.objectweb.org/.
8. Kephart (J. O.) et Chess (D. M.). – The Vision of Autonomic Computing. *IEEE Computer Magazine*, vol. 36, n1, 2003.
9. Nagaraja (K.), Oliveira (F.), Bianchini (R.), Martin (R. P.) et Nguyen (T. D.). – Understanding and Dealing with Operator Mistakes in Internet Services. *In : 6th Symposium on Operating System Design and Implementation (OSDI-2004)*. – San Francisco, CA, décembre 2004.
10. Norris (J.), Coleman (K.), Fox (A.) et Candea (G.). – OnCall : Defeating Spikes with a Free-Market Application Cluster. *In : 1st International Conference on Autonomic Computing (ICAC-2004)*. – mai 2004.
11. ObjectWeb Open Source Middleware. – C-JDBC : Clustered JDBC. http ://c-jdbc.objectweb.org/.
12. Ranjan (S.), Rolia (J.), Fu (H.) et Knightly (E.). – QoS-Driven Server Migration for Internet Data Centers. *In : 10th International Workshop on Quality of Service (IWQoS 2002)*. – Miami Beach, FL, mai 2002.
13. Saito (Y.), Bershad (B. N.) et Levy (H. M.). – Manageability, Availability and Performance in Porcupine : A Highly Scalable, Cluster-Based Mail Service. *ACM Transactions on Computer Systems*, vol. 18, août 2000.
14. Shen (K.), Tang (H.), Yang (T.) et Chu (L.). – Integrated resource management for cluster-based internet services. *In : 5th USENIX Symposium on Operating System Design and Implementation (OSDI-2002)*. – décembre 2002.
15. Sun Microsystems. – Java 2 Platform Enterprise Edition (J2EE). – http ://java.sun.com/j2ee/.
16. Urgaonkar (B.) et Shenoy (P.). – *Cataclysm : Handling Extreme Overloads in Internet Services*. – Technical report, Department of Computer Science, University of Massachusetts, novembre 2004.
17. Urgaonkar (B.) et Shenoy (P.). – Sharc : Managing CPU and network bandwidth in shared clusters. *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, n1, 2004.
18. Urgaonkar (B.), Shenoy (P.), Chandra (A.) et Goyal (P.). – Dynamic Provisiong of Multi-Tier Internet Applications. *In : 2nd International Conference on Autonomic Computing (ICAC-2005)*. – Seattle, WA, juin 2005.
19. Zhu (H.), Ti (H.) et Yang (Y.). – Demand-driven service differentiation in cluster-based network servers. *In : 20th Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM-2001)*. – Anchorage, AL, avril 2001.