Jade : un environnement d'administration autonome

Noël De Palma[β], Sara Bouchenak[α], Fabienne Boyer[α], Daniel Hagimont[δ], Sylvain Sicard[α], Christophe Taton[β]

α Université Joseph Fourier, Grenoble
β Institut National Polytechique de Grenoble
δ Institut National Polytechique de Toulouse

INRIA Rhône-Alpes
655 avenue de l'Europe
38334 Saint Ismier Cedex

RÉSUMÉ. *Cet article présente la conception, la réalisation et l'évaluation de Jade, un environnement pour l'administration autonome d'infrastructures logicelles patrimoniales. Jade est essentiellement composé de deux parties : un canevas pour l'encapsulation des ressources administrées, qui leur donne une interface d'administration uniforme, et un canevas de construction de gestionnaires autonomes, qui administrent à l'exécution un ensemble de ressources suivant une politique particulière.*
Nous rapportons nos expérimentations concernant des politiques d'administration visant l'auto-optimisation et l'auto-réparation d'une application J2EE s'exécutant sur une grappe de nœuds.

ABSTRACT. *This paper presents the design, implementation and evaluation of Jade, an environment for autonomic management of legacy software infrastructures. This environment relies on two main frameworks: a framework for the encapsulation of administrable resources, which provides the administered system's resources with a uniform management interface; and a framework for autonomic managers, which regulates a set of managed resources for a specific management aspect.*

We report on our experiments with self-optimization and self-repair for a J2EE application deployed in a cluster of machines.

MOTS-CLÉS :*administration autonome, auto-optimisation, auto-réparation, J2EE*

KEYWORDS: *autonomic management, self-optimization, self-repair, J2EE.*

1. Introduction

Les environnements informatiques d'aujourd'hui sont de plus en plus sophistiqués. Ils intègrent de nombreux logiciels complexes qui coopèrent dans le cadre d'une infrastructure logicielle, potentiellement à grande échelle. Ces logiciels se caractérisent par une grande hétérogénéité, en particulier en ce qui concerne les modèles de programmation utilisés pour les concevoir et les développer. De plus, ces logiciels fournissent des fonctions d'administration elles aussi marquées par une grande hétérogénéité en termes d'interfaces d'utilisation.

La conséquence est que l'administration de ces infrastructures logicielles est une tâche de plus en plus complexe, source d'erreurs, et consommatrice en ressources :

– humaines, car elle nécessite de nombreux administrateurs humains pour maîtriser cette complexité et réagir aux événements (par exemple des pannes) survenant,

– matérielles, car pour anticiper des incidents potentiels (pannes, surcharges), les administrateurs ont souvent recours à un surdimensionnement de l'infrastructure.

Les systèmes autonomes (Kephart *et al.*, 2003)(Ganek *et al.*, 2003), qui visent une gestion autonome de ressources (logicielles ou matérielles) dans des conditions changeantes, constituent une approche très prometteuse pour répondre à ce problème. Les bénéfices de cette approche sont les suivants :

– Le fait de programmer (dans un langage de haut niveau) les interventions d'administration réduit les erreurs de configuration et les efforts d'administration.

– L'administration autonome permet au système d'effectuer les reconfigurations nécessaires sans intervention humaine, ce qui économise les ressources humaines.

– L'administration autonome est également un moyen d'économie de ressources matérielles, car ces ressources peuvent être allouées dynamiquement à la demande, par les programmes d'administration autonomes, en réaction à des pannes ou des surcharges, au lieu d'être pré-allouées.

Dans cette direction de recherche, nous avons conçu et implanté Jade, un environnement permettant l'administration autonome d'applications réparties. L'environnement Jade est principalement composé de deux parties :

– un canevas pour l'encapsulation des éléments administrés (ME pour *Managed Elements*), permettant de manipuler les logiciels administrés à travers une interface uniforme. Les ME sont des composants élémentaires qui peuvent être assemblés dans le cadre d'une infrastructure logicielle qui peut évoluer au cours du temps.

– un canevas de construction de gestionnaires autonomes (AM pour *Autonomic Managers*) qui implantent des politiques d'administration autonome pour des aspects déterminés. Un AM est généralement implanté sous la forme d'une boucle de contrôle qui supervise et reconfigure suivant une certaine politique une infrastructure de ME.

Parmi les aspects d'administration qui peuvent être étudiés, nous avons porté notre attention sur les deux points suivants :

– l'auto-réparation qui consiste, en cas de panne, à ramener l'application dans un état équivalent à celui précédant la panne (Bouchenak *et al.*, 2005).

– l'auto-optimisation, qui consiste à maintenir le niveau de performance de l'application lorsqu'elle fait l'objet de fortes variations de la charge qui lui est soumise (Hagimont *et al.*, 2006).

Afin de valider nos propositions, nous nous sommes intéressés à l'administration autonome d'une grappe de machines utilisée pour implanter des applications Web, généralement des serveurs de commerce électronique. Plus précisément, nous présentons une validation de nos travaux à travers l'administration autonome d'une application Web structurée suivant une architecture J2EE sur une grappe de machines.

Le reste de cet article est structuré de la façon suivante. La section 2 présente nos motivations pour la conception d'un logiciel d'administration autonome. La section 3 présente les principes de conception que nous avons suivis et la section 4 décrit le prototype implanté. L'évaluation de cette réalisation est rapportée dans la section 5. Après un survol des travaux comparables (section 6), nous concluons l'article en section 7.

2. Besoins

Afin de préciser les besoins d'un logiciel d'administration autonome, nous les étudions dans le cadre de notre contexte applicatif privilégié, à savoir celui des applications J2EE en grappe.

2.1. Les applications J2EE en grappe

La spécification J2EE propose un cadre pour la conception de serveurs Web dynamiques (J2EE, 200x). Ces serveurs Web génèrent des pages Web dynamiquement en réponse à des requêtes HTTP. Un serveur J2EE est généralement composé de trois tiers, qui peuvent s'exécuter sur des machines différentes, comme illustré sur la figure 1 :

– Le frontal Web, qui est souvent composé de deux serveurs : un serveur Web et un serveur de Servlet. Le serveur Web, par exemple (Apache, 200x), traite les requêtes des clients. Une référence à une page statique est résolue par ce serveur alors qu'une référence à une page dynamique est transmise au serveur de Servlet. Le serveur de Servlet, par exemple (Tomcat, 200x), exécute des programmes Java (les Servlets) qui génèrent des pages Web à la volée en utilisant des données qui peuvent être obtenues d'un serveur EJB.

– Le serveur EJB, par exemple (Jonas, 200x), exécute le code fonctionnel de l'application, qui calcule les données demandées par les Servlets. Le serveur EJB gère des objets Java persistants qui sont conservés de façon stable dans un serveur de base de données.

– Le serveur de base de données, par exemple (MySQL, 200x), gère les données persistantes de l'application et est appelé par le serveur EJB pour charger et stocker ses objets.

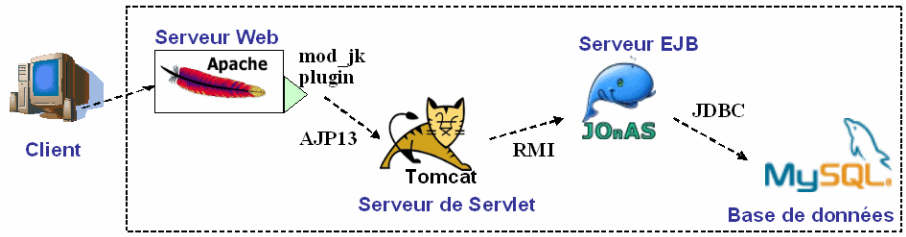

Figure 1. *Architecture d'une application J2EE*

Dans ce contexte, il est important d'assurer le passage à l'échelle et la disponibilité du serveur J2EE (He *et al.*, 2000)(Iyengar *et al.*, 1997). Le passage à l'échelle se définit comme sa capacité à traiter un nombre variable de requêtes (croissant ou décroissant) tout en maintenant un niveau de performance acceptable. La disponibilité est définie comme la capacité du serveur à répondre aux requêtes malgré l'occurrence de pannes (logicielles ou matérielles).

A la fois le passage à l'échelle et la disponibilité peuvent être obtenus par duplication des différents tiers sur une grappe de machines, comme illustré sur la figure 2.

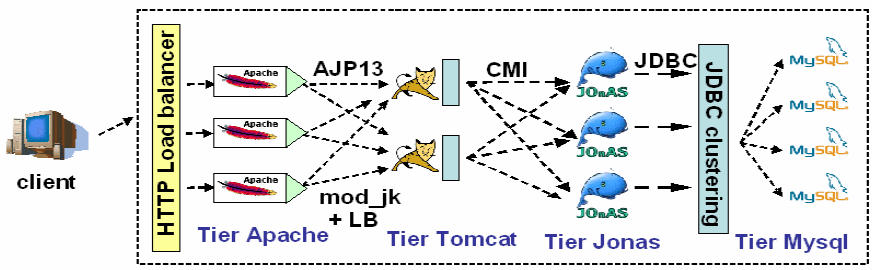

Figure 2. *Architecture J2EE sur une grappe*

Les schémas de duplication généralement employés dans les architectures J2EE dupliquent entièrement les serveurs sur plusieurs machines. Les requêtes clientes sont aiguillées vers les duplicas afin de répartir la charge ; la sélection du duplica

repose généralement sur une stratégie Round-Robin (tourniquet). Chaque tier implante une gestion de la cohérence particulière pour les données dupliquées ; les politiques de gestion de la cohérence sont précisées ultérieurement dans l'article lorsque nous présentons nos expérimentations.

2.2. Administration autonome

Un critère important de la configuration d'une architecture J2EE en grappe est le degré de duplication de chacun des tiers. Dans une configuration statique, l'administrateur choisit un degré de duplication pour chaque tier. Mais la configuration statique a de nombreux inconvénients que l'on peut analyser en fonction de l'objectif de la duplication :

– **disponibilité**. La duplication peut être utilisée pour masquer les conséquences des pannes et maintenir la disponibilité du service. Si on duplique statiquement un tier en N exemplaires, si N est trop grand on gaspille les ressources de la grappe, alors que si N est trop petit la disponibilité peut être compromise. Le choix du degré de duplication est difficile étant donné que le taux de panne est difficile à prédire. Une solution plus flexible consiste à permettre l'allocation dynamique des tiers en cas de panne.

– **passage à l'échelle**. La duplication peut être utilisée pour anticiper les augmentations de la charge soumise aux tiers. Comme précédemment, il est difficile de prédire la charge à laquelle seront soumis les tiers et de déterminer statiquement le degré de duplication nécessaire. Une solution plus flexible consiste à dupliquer dynamiquement un tier de l'architecture J2EE lorsque la charge sur ce tier augmente, et à réduire son degré de duplication lorsqu'il est sous-chargé. Ainsi, les machines de la grappe peuvent être allouées dynamiquement pour des tiers différents en fonction de leur charge.

Une réserve de machines peut être utilisée dynamiquement pour assurer la disponibilité et le passage à l'échelle, l'architecture J2EE étant gérée par des programmes d'administration autonomes.

2.3. Applications patrimoniales

Un besoin important pour l'administration des infrastructures logicielles est la capacité de gérer des logiciels patrimoniaux ayant des interfaces d'administration très hétérogènes. Nous ne voulons pas faire l'hypothèse que les applications administrées ont été conçues suivant un modèle particulier en se conformant à des interfaces particulières.

Dans la solution que nous proposons, les logiciels administrés sont considérés comme des boîtes noires que nous encapsulons dans des composants logiciels appelés éléments administrés (ou ME pour *Managed Elements*). Grâce à cette

encapsulation, les ME possèdent tous la même interface d'administration, ce qui permet de les administrer par des programmes.

3. Principes de conception

Dans cette section, nous présentons les principes généraux retenus pour la conception du système autonome Jade, puis nous présentons le modèle de composants Fractal qui est la base de la conception de notre canevas logiciel.

3.1. Organisation d'un système autonome

Nous avons adopté l'organisation générale décrite dans (Kephart *et al.*, 2003).

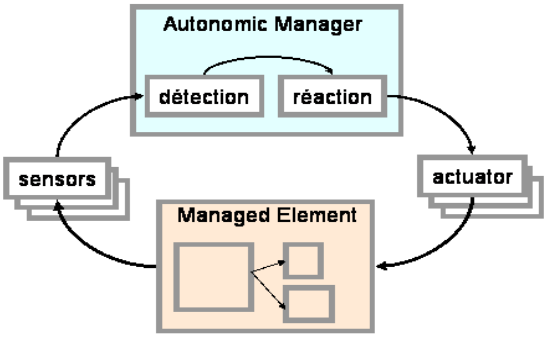

Figure 3. *Organisation d'un système autonome*

Les systèmes autonomes sont constitués de deux parties, les éléments administrés (Managed Element / ME) et les gestionnaires autonomes (Autonomic Manager / AM).

Les éléments administrés encapsulent les ressources logicielles et matérielles constituant le système. Ils fournissent des opérations d'administration permettant de surveiller leur exécution (sensors) et de modifier leurs paramétrages au sens large (actuators).

Un gestionnaire autonome est un élément logiciel chargé d'imposer un comportement sur les éléments administrés qu'il supervise. Ce comportement est assuré par une boucle de contrôle que le gestionnaire autonome implante. Le gestionnaire est divisé en 2 parties logiques distinctes. La partie détection surveille l'exécution d'un ou plusieurs éléments administrés grâce aux mesures provenant des sensors. La partie réaction réagit à des événements qui sont survenus sur le système administré en déclanchant un ensemble d'actions correctrices sur le système. Entre la

partie détection et la partie réaction, on peut trouver une partie appelée décision qui est chargée de décider de la réaction à enclencher en fonction d'une analyse des événements survenus.

Jade est un canevas logiciel pour la construction de système autonome. Nous avons décidé d'utiliser le modèle à composants Fractal pour l'implantation de Jade. La section suivante présente les principales caractéristiques de ce modèle à composants.

3.2. Le modèle à composants Fractal

Le modèle à composants Fractal (Bruneton *et al.*, 2006) permet de définir des composants primitifs et des composants composites. Un composant primitif contient du code exécutable. Un composant composite est construit comme un assemblage de sous-composants, les sous-composants pouvant être partagés entre plusieurs composants composites. Les composants primitifs ou composites exhibent des interfaces fonctionnelles clientes et serveurs, ces interfaces pouvant être liées pour créer des assemblages de composants.

En plus de ses interfaces fonctionnelles, un composant possède un ensemble d'interfaces de contrôle qui permettent de gérer des aspects non fonctionnels. Ces interfaces sont implantées par des contrôleurs qui implantent les politiques de gestion de ces aspects. D'autres interfaces/contrôleurs peuvent aussi être ajoutés. Nous décrivons les principaux contrôleurs définis par défaut dans le modèle Fractal.

– **AttributeController.** Un attribut est une propriété configurable d'un composant. Le contrôleur d'attributs implante une interface qui fournit les méthodes (*getter/setter*) permettant l'accès à ces attributs.

– **BindingController.** Le contrôleur de liaisons d'un composant est utilisé pour contrôler les connexions de ce composant vers d'autres composants. Il implante une interface permettant de consulter et modifier l'état des liaisons du composant.

– **LifecycleController.** Le gestionnaire de cycle de vie permet de contrôler l'exécution du composant auquel il est attaché. Il permet d'appeler les opérations classiques de contrôle de l'exécution (par exemple l'arrêt ou le démarrage du composant).

– **ContentController.** Le gestionnaire de contenu permet la gestion du contenu des composants composites. Son interface permet d'observer, d'ajouter, de retirer des sous-composants dans le composite et de manipuler les liaisons entre ces sous-composants.

Ces contrôleurs sont définis par défaut dans le modèle Fractal. Il est possible de modifier l'ensemble des contrôleurs associés à un composant (en ajouter, en enlever et en modifier).

Fractal est la base de notre canevas pour la construction de systèmes autonomes car :

– Fractal définit un modèle à composants hiérarchique, permettant la définition de composants composites et le partage de sous-composants entre composants composites. Il permet donc de représenter sous la forme d'une hiérarchie de composants des systèmes administrés très complexes.

– Fractal fournit une interface de contrôle uniforme et adaptable, permettant l'introspection (observation des attributs du composant) et la liaison dynamique (reconfiguration d'un assemblage de composants). Il fournit donc les moyens de mettre en œuvre une gestion à l'exécution (observations et modifications) sur des éléments administrés.

Nous avons utilisé les caractéristiques de Fractal[1] pour implanter une forme générique de ME fournissant une interface uniforme d'administration. Cette approche, que nous appelons *Administration Basée Composants* est présentée dans la section suivante.

4. Le canevas Jade

4.1. Administration Basée Composants

L'administration basée composants fournit une couche d'abstraction pour un ensemble d'éléments matériels ou logiciels. Un élément est encapsulé dans un composant Fractal primitif, qui implante ainsi un ME primitif. Des composants Fractal composites peuvent être définis pour construire des abstractions utiles à l'administration du système, formant ainsi des ME composites. L'administration basée composants est un moyen de :

– gérer des éléments patrimoniaux très hétéroclites à travers un modèle uniforme, au lieu d'utiliser des interfaces propres aux éléments, généralement des fichiers de configuration gérés à la main.

– gérer des environnements complexes avec différents points de vue. Par exemple, en utilisant des ME composites à différents niveaux d'abstraction, il est possible de représenter une infrastructure réseau, la configuration d'un environnement intergiciel[2] J2EE ou la configuration d'une application sur l'intergiciel J2EE.

Cette approche est illustrée sur la figure 4, pour le point de vue de l'environnement intergiciel J2EE. Dans cette configuration, un équilibreur de charge (LB) répartit les requêtes reçues entre deux exemplaires de serveur Apache. Les

[1] Et plus précisément son implantation sur la machine virtuelle Java, nommée Julia. Dans la suite, avec le nom Fractal, nous faisons référence à cette implantation.

[2] middleware

serveurs Apache sont connectés à deux serveurs Tomcat, qui sont connectés au même serveur MySQL[3].

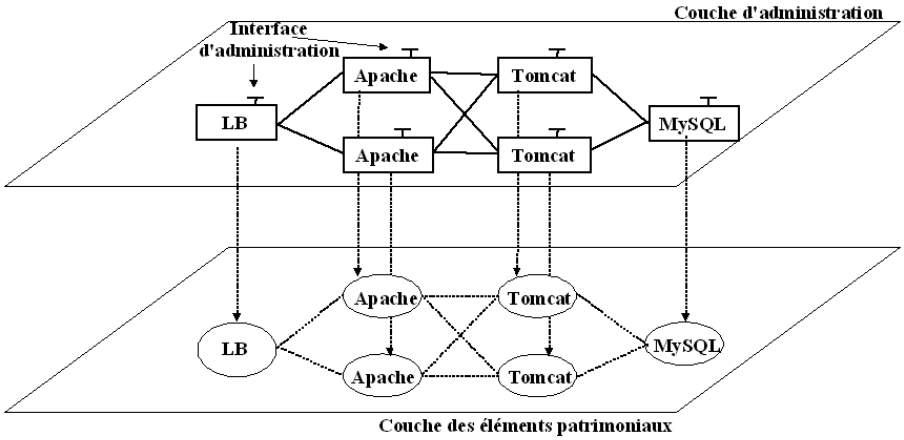

Figure 4. *Administration Basée Composants*

Dans la figure 4, les flèches verticales en pointillé représentent la relation d'encapsulation entre les composants (implantant les ME) et les éléments patrimoniaux administrés. Dans la couche patrimoniale, les lignes horizontales en pointillé représentent les relations (ou liaisons) entre les éléments, dont les implantations sont propriétaires. Ces liaisons sont représentées dans la couche d'administration sous la forme de liaisons entre composants (lignes pleines dans la couche du haut). Il convient de faire les deux remarques suivantes.

– La figure précédente ne montre qu'un seul point de vue, celui de la gestion des intergiciels de l'architecture J2EE. Il lui est associé un ME composite qui décrit cette architecture. De nombreux autres aspects d'administration sont gérés et ne sont pas représentés sur cette figure. C'est par exemple le cas des nœuds sur lesquels les intergiciels J2EE sont déployés, qui sont également représentés par des ME dans la couche d'administration.

– Cette structure possède des similitudes avec les architectures réflexives organisées en méta-niveaux. Mais on s'intéresse ici essentiellement à l'abstraction d'un environnement constitué de ressources très hétérogènes, en les représentant dans un environnement à composants homogènes.

Dans la couche d'administration, tous les composants (ME) fournissent la même interface (uniforme) d'administration pour les éléments encapsulés. Cette interface

[3] Pour des raisons de clarté, seuls certains ME sont représentés dans la figure 4 (des ME additionnels ont été utilisés dans notre expérimentation, représentant par exemple des applications déployées sur les serveurs J2EE) .

permet de d'observer et de contrôler les attributs des éléments, leurs liaisons et leur cycle de vie (démarrage, arrêt). L'implantation de cette interface est spécifique à chaque élément encapsulé (par exemple Apache, Tomcat, MySQL dans le cas de J2EE).

A partir de cette couche d'administration, des programmes d'administration sophistiqués peuvent être implantés, sans avoir à utiliser des interfaces de configuration propriétaires et complexes, qui sont cachées dans les composants d'encapsulation. La couche d'administration fournit toutes les fonctions nécessaires à l'implantation des programmes d'administration :

– **Introspection.** Un programme d'administration peut utiliser l'interface d'introspection des ME pour les observer. Par exemple, un tel programme peut consulter les attributs d'un ME Apache particulier (qui encapsule un serveur Apache) pour trouver que ce serveur Apache s'exécute sur la machine *node1* et reçoit ses requêtes sur le port 80. Il peut également observer l'architecture J2EE globale (qui est considérée comme un ME composite) pour découvrir qu'elle est composée de deux serveurs Apache connectés à deux serveurs Tomcat connectés à un serveur MySQL.

– **Reconfiguration.** Un programme d'administration peut également utiliser l'interface de reconfiguration des ME permettant de contrôler l'architecture à composants. Cette interface permet notamment de modifier les attributs des composants et les liaisons entre ces composants. Ces opérations de reconfiguration au niveau de la couche d'administration sont répercutées sur la couche patrimoniale. Un exemple de reconfiguration est l'ajout ou le retrait d'instances de serveur Apache en fonction de la charge courante du serveur.

Le fait que tous les composants (ME) fournissent la même interface (uniforme) d'administration pour les éléments encapsulés permet de construire des gestionnaires autonomiques génériques, dans le sens où ils peuvent contrôler n'importe quel ME. Cependant, dans certains cas, on veut pouvoir disposer de fonctions d'administration spécifiques au logiciel administré. Pour ce faire, les interfaces fournies par les ME peuvent être étendues.

4.2. Implantation des ME

Rappelons que les ME sont des entités qui représentent des éléments administrés, qui peuvent être manipulées par des programmes d'administration à travers une interface uniforme. Nous distinguons deux types de ME, qui correspondent simplement aux notions de composants primitifs et composants composites :

– les ME primitifs (composants primitifs) représentent des éléments patrimoniaux (matériels ou logiciels) pour en fournir une interface d'administration uniforme. Dans cette catégorie, nous trouvons les ME représentant des logiciels patrimoniaux comme Apache ou Tomcat.

– les ME composites (composants composites) représentent des abstractions utilisées dans une tâche d'administration, comme une machine (l'ensemble des logiciels s'exécutant sur cette machine), une sous-grappe de machines (un ensemble de machines) ou l'architecture J2EE implantant une application Web.

Les sections 4.2.1 et 4.2.2 décrivent les ME primitifs et composites dans le cas de l'administration d'une plate-forme J2EE.

4.2.1. *ME primitifs*

Dans le contexte des applications J2EE que nous avons utilisé pour valider notre environnement d'administration autonome, nous avons implanté un ensemble de ME primitifs pour les différents tiers de l'architecture J2EE (Apache, Tomcat et MySQL dans nos expérimentations). Nous avons également défini un ME pour représenter une application appelée RUBiS (Cecchet *et al.*, 2003) qui implante un site d'enchères similaire à eBay.com et permettant d'évaluer notre prototype dans des conditions très proches de la réalité.

Nous décrivons dans la suite comment nous encapsulons dans un ME primitif un logiciel existant en nous appuyant sur l'exemple de l'intergiciel Apache. De manière générale, il est apparu que le coût de réingénierie pour encapsuler des logiciels patrimoniaux est tout à fait acceptable (cet aspect est traité dans la section 5.1).

Le ME Apache fournit des contrôleurs *AttributeController*, *BindingController* et *LifeCycleController*, qui permettent l'administration du serveur Apache :

– Le contrôleur d'attributs permet de consulter et affecter des attributs configurables du ME Apache. Par exemple, le logiciel Apache possède un attribut *port*, défini dans son fichier de configuration *httpd.conf*. Un attribut *port* est donc géré dans le ME Apache et la modification de cet attribut est répercutée dans le fichier de configuration *httpd.conf*.

– Le contrôleur de liaisons permet de gérer des liaisons du ME Apache vers des ME Tomcat au travers de primitives de type *bind(..)/unbind(..)*. Dans Apache, les connexions vers des serveurs Tomcat sont définies dans le fichier de configuration *worker.properties*. En conséquence, la définition d'une liaison entre un ME Apache et un ME Tomcat (en utilisant l'interface *BindingController* du ME Apache) est répercutée dans le fichier *worker.properties*.

– Le contrôleur du cycle de vie permet d'arrêter et démarrer le ME au travers de primitives de type *start(..)/stop(..)*. L'implantation de ce contrôleur appelle les commandes (généralement appelées depuis un *shell* d'Unix) d'Apache permettant d'arrêter et démarrer le serveur Apache.

Les autres intergiciels Tomcat et MySQL sont encapsulés dans des ME primitifs de la même façon, et fournissent ainsi les mêmes interfaces d'administration.

4.2.2. *ME composites*

Pour permettre de modéliser des architectures complexes, nous définissons des ME architecturaux (*Architectural Managed Element* ou AME). Nous identifions plus précisément divers types d'AME modélisant deux types de ME 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 qui héberge une application (figure 5, organisation verticale). Un CluME (*Cluster ME*) représente une grappe de ME dupliqués (figure 5, organisation transversale). Enfin un MultiME (*Multi-Tiers ME*) est une combinaison de plusieurs ME pour constituer une architecture (figure 5, organisation horizontale).

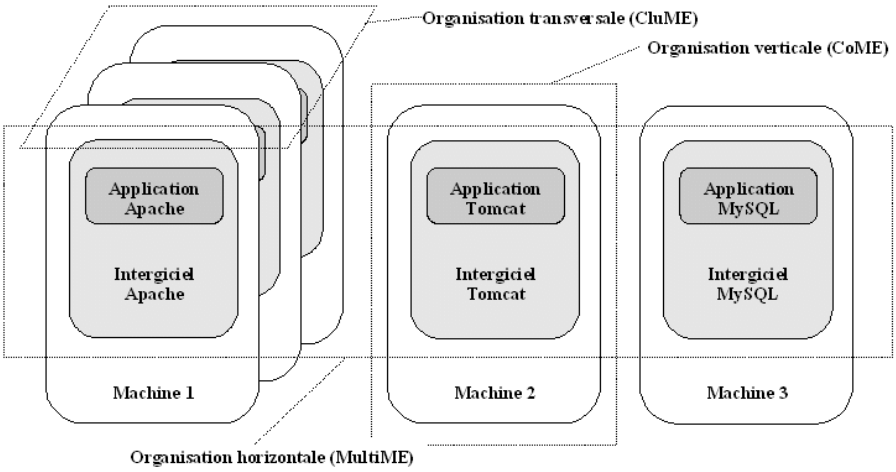

Figure 5. *ME composites dans une architecture J2EE*

Comme le montre la figure 5, les AME sont des structures qui se superposent. Il est possible de définir des AME plus complexes permettant de représenter les environnements administrés. Les AME présentés ici sont ceux que nous avons utilisés pour gérer un serveur J2EE en grappe. Nous donnons dans la suite deux exemples de l'utilisation de ces patrons.

– Administration des nœuds. Comme motivé dans la section 2.2, pour assurer le passage à l'échelle d'applications J2EE en grappe, nous nous intéressons au dimensionnement dynamique d'un schéma de duplication de composants. Pour ce faire, il faut être en mesure de déployer dynamiquement un élément logiciel sur une machine allouée dynamiquement dans la grappe. A cet effet, le CoME composite *ManagedNode* représente une machine de la grappe et possède un gestionnaire de contenu (*ContentController* en Fractal) permettant d'ajouter un sous-ME. Dans le cas présent, le gestionnaire de contenu est utilisé pour ajouter des sous-ME représentant

des logiciels à déployer sur la machine représentée. De plus, par introspection du CoME *ManagedNode* associé à une machine, on peut découvrir quels logiciels ont été déployés sur cette machine.

– Administration de la grappe. Pour gérer un groupe de nœuds dans la grappe, nous utilisons un ME composite de type CluME (appelé *ManagedCluster*). Il inclut l'ensemble des ME *ManagedNode* représentant ces nœuds. Il fournit une interface permettant d'allouer et libérer des nœuds suivant une politique d'allocation déterminée. Il inclut également des sondes permettant de monitorer ces nœuds afin de détecter les pannes de nœuds, et maintenir une vue cohérente des nœuds en terme de disponibilité.

4.3. Implantation des AM

Les gestionnaires autonomes (AM) implantent des boucles de contrôle dont le but est surveiller et gérer le système administré.

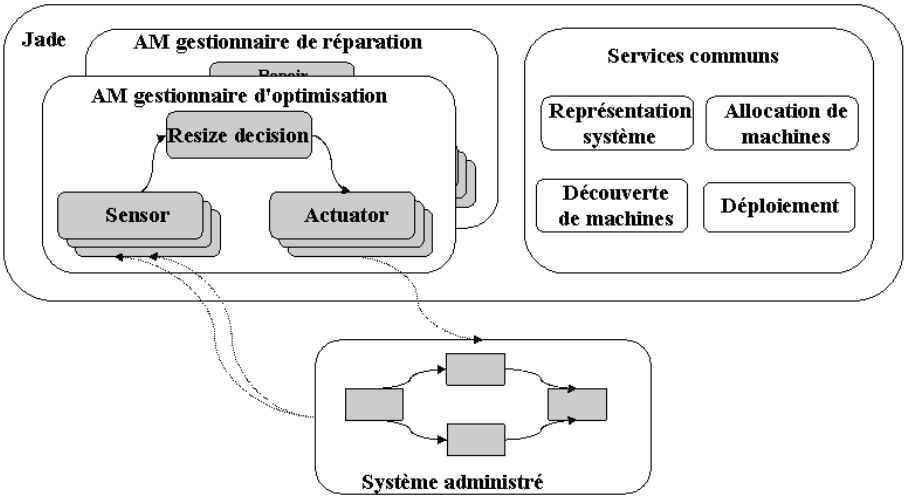

Figure 6. *Vue globale de Jade*

La figure 6 donne une vue d'ensemble de l'environnement d'administration autonome Jade. La figure montre deux AM qui sont respectivement responsables de la gestion de l'auto-réparation et de l'auto-optimisation. Ces deux AM utilisent les interfaces des ME pour implanter leur politique.

Chaque AM dans Jade définit une boucle de contrôle qui utilise des services communs, permettant notamment :

– la découverte dynamique de nœuds qui rejoignent la grappe. Un CoME *ManagedNode* est créé pour chaque nouveau nœud et inséré dans le CluME *ManagedCluster* de gestion de la grappe.

– la gestion d'une politique d'allocation des nœuds qui gère la réservation des nœuds pour les applications déployées sur la grappe.

– le déploiement de l'architecture logicielle d'une application sur la grappe. L'architecture logicielle à déployer est décrite dans le langage de description d'architecture (ADL pour *Architecture Description Language*) de Fractal.

– la gestion d'une Représentation Système qui est une sorte de sauvegarde du système, présentée ci-après dans la section Auto-réparation.

Comme évoqué précédemment, nous avons principalement expérimenté avec deux politiques d'administration autonome qui visent respectivement l'auto-réparation et l'auto-optimisation d'une architecture J2EE en grappe. Nous décrivons les algorithmes associés à ces deux politiques de gestion.

4.3.2. Auto-réparation

L'AM d'auto-réparation que nous avons mis en œuvre (Bouchenak *et al.*, 2005) traite les pannes franches des nœuds. Cet AM permet de réparer l'architecture logicielle J2EE lorsqu'une panne est détectée. Un ensemble de sondes est déployé afin de détecter les pannes de nœuds. Ces sondes s'appuient sur des consultations périodiques (*heartbeat*) pour surveiller les nœuds. Lorsqu'une panne est détectée, l'AM exécute un programme de réparation qui alloue une machine de remplacement, redéploie les ME nécessaires sur les nouveaux nœuds et remet en cohérence l'architecture logicielle administrée.

Pour la gestion de l'auto-réparation, nous avons cependant été confronté au problème suivant. Pour un élément logiciel administré (tel que Apache ou Tomcat), le ME primitif qui encapsule le logiciel administré est localisé sur la même machine que le logiciel. Ceci implique que si cette machine est défaillante, il n'est plus possible d'utiliser la couche d'administration pour connaître l'état du système avant la panne, étant donné que les ME qui étaient sur cette machine ne sont plus accessibles.

Pour résoudre ce problème, un service de Jade appelé la Représentation Système (SR pour *System Representation*) implante une sauvegarde de l'architecture globale des ME, c'est à dire des composants Fractal qui implantent ces ME. Le SR est une sorte de méta-niveau d'une application Fractal[4], qui contient une sauvegarde des caractéristiques architecturales de cette application : quels sont les composants instanciés, comment sont-ils liés, quelles sont les valeurs de leurs attributs, etc . Le SR est géré sur une seule machine, mais peut être dupliqué pour tolérer une panne de

[4] La couche d'administration composée des ME est un méta-niveau de la couche patrimoniale. Le SR est un méta-niveau de la couche d'administration.

cette machine. Il est maintenu causalement cohérent avec l'architecture de ME, dans le sens où toute reconfiguration sur l'architecture de ME est répercutée sur le SR.

L'AM d'auto-réparation introspecte le SR pour connaître la configuration de l'environnement logiciel administré avant la panne, puis après avoir alloué une nouvelle machine en utilisant le service d'allocation de nœuds, il re-crée sur cette machine les composants équivalents à ceux qui étaient créés sur la machine tombée en panne. En particulier, il met à jour pour les composants re-créés leurs attributs et leurs liaisons avec les autres composants de l'architecture logicielle. Si ces composants doivent effectuer une procédure de reprise d'état, alors l'exécution de cette procédure sera déclenchée par le gestionnaire d'auto-réparation avant le redémarrage des composants.

4.3.2. Auto-optimisation

La gestion d'auto-optimisation que nous avons mise en œuvre (Hagimont *et al.*, 2006) permet de traiter un phénomène de surcharge au niveau d'un tier répliqué d'une architecture J2EE. Un ensemble de sondes surveille les charges CPU moyennes au niveau des tiers de l'architecture J2EE. Lorsqu'une surcharge est détectée au niveau d'un tier (par exemple Tomcat), l'AM alloue une nouvelle machine, déploie sur cette machine le ME associé à ce tier (Tomcat) et l'insère dans l'architecture J2EE. Cette insertion nécessite de configurer correctement ce ME, principalement ses attributs et ses liaisons avec les autres composants de l'architecture J2EE (par exemple un nouvel exemplaire de serveur Tomcat doit être interconnecté avec les serveurs Apache et les serveurs MySQL de l'architecture).

De même, si la ressource CPU est sous-utilisée au niveau d'un tier, l'AM peut retirer une machine allouée à ce tier, afin de la placer dans une réserve de nœuds pour qu'elle puisse être utilisée ultérieurement pour un autre tier. Un possible effet "yoyo" est maîtrisé grâce à l'introduction d'une période réfractaire dans le gestionnaire d'auto-optimisation. Cela signifie qu'après une reconfiguration le gestionnaire ne peut plus être actif durant un certain temps (période durant laquelle le système est censé se stabiliser).

4.3.3. Implémentation des reconfigurations

Les interfaces des ME permettent de programmer des AM implantant des reconfigurations dynamiques et autonomes de ces architectures. Si on considère le scénario de la figure 7, un serveur Apache s'exécute sur la machine 1 et il est connecté à un serveur Tomcat s'exécutant sur la machine 2. L'objet de la reconfiguration est de remplacer ce serveur Tomcat par un autre qui s'exécute sur la machine 3.

Sans utiliser le service fourni par Jade, cette simple reconfiguration nécessite d'effectuer manuellement les opérations suivantes : se connecter à la machine 1,

arrêter le serveur Apache en utilisant le script *shutdown* d'Apache, éditer le fichier de configuration d'Apache (*workers.properties*) pour mettre à jour le nom de la machine vers laquelle le serveur Apache doit propager les requêtes, redémarrer le serveur Apache en utilisant le script *httpd*.

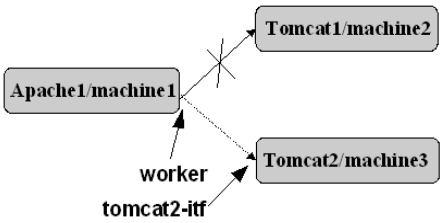

Figure 7. *Scénario de reconfiguration*

Avec Jade et une fois les logiciels encapsulés dans des ME (c'est à dire des composants Fractal), les interfaces de contrôle des ME peuvent être utilisées pour effectuer la reconfiguration souhaitée, en utilisant successivement le contrôleur de cycle de vie et le contrôleur de liaison comme suit :

// Arrêt d'Apache1/machine1

Fractal.getLifeCycleController(Apache1).stopFc();

// Destruction de la liaison Apache1/machine1 vers Tomcat1/machine2

Fractal.getBindingController(Apache1).unbindFc("worker0");

// Création d'une liaison Apache1/machine1 vers Tomcat2/machine2

Fractal.getBindingController(Apache1).bindFc("worker0", tomcat2-itf);

// Redémarrage de Apache1/machine1

Fractal.getLifeCycleController(Apache1).startFc();

4.3.4. Cohérence des ME dupliqués

Un problème important lorsque l'on gère la duplication de composants qui incluent un état modifiable est la gestion de la cohérence de cet état. Le problème ne se pose pas dans le cas du serveur Tomcat, car l'application que nous utilisons pour notre évaluation est composée de Servlets qui ne gèrent pas de données de session. Le problème ne se pose pas non plus pour le serveur Apache qui ne gère pas d'état modifiable. Par contre, le serveur de base de données contient des données modifiables. Pour permettre la duplication du serveur de base de données, nous utilisons (C-JDBC, 200x) qui est un équilibreur de charge qui répartit les requêtes de lecture sur des copies entières de la base de données (*full mirroring*) et assure la

cohérence en garantissant une mise à jour atomique sur les différentes copies lors des écritures.

Pour permettre l'allocation et le retrait de nœuds de l'ensemble des duplicas, tout en évitant de copier la base de données très volumineuse lors de l'allocation, la base de donnée n'est pas détruite lors du retrait et son état est resynchronisé lors de l'allocation. Pour ce faire, un journal des modifications a été ajouté à C-JDBC et reçoit toutes les modifications sur la base de donnée répliquée. Lorsqu'une base de données est ré-insérée dans l'ensemble, on suppose qu'elle possède en local une ancienne version de la base, et les modifications journalisées depuis le dernier retrait sont rejouées avant l'insertion.

5. Evaluation

5.1. Administration d'une application J2EE

Afin de valider le système Jade, notre domaine d'application privilégié est celui des applications J2EE en grappe. Pour les évaluations, nous avons utilisé l'application J2EE de commerce électronique RUBiS (Cecchet *et al.*, 2003) qui implante un site d'enchère modélisé d'après eBay.com.

L'application Rubis est déployée sur les intergiciels J2EE suivants : un serveur web Apache, un serveur d'entreprise Tomcat et un serveur de bases de données MySQL. De plus, chaque étage est dupliqué pour former, respectivement, une grappe de serveurs web dupliqués, une grappe de serveurs d'entreprise dupliqués et une grappe de serveurs de bases de données dupliqués.

Le site Rubis déployé sur un système J2EE multi-niveaux dupliqué est modélisé, dans Jade, par les ME suivants : un CoME pour chaque machine hébergeant ce site, un ME primitif pour chaque intergiciel s'exécutant sur une machine (i.e. Apache, Tomcat, MySQL), un ME primitif pour chaque application hébergée par un intergiciel (i.e. applications web, d'entreprise et de base de données), un CluME pour chaque grappe de ME dupliqués et un MultiME 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 le gestionnaire d'auto-optimisation, les AME ou les services de réservation de nœuds, d'installation de logiciels ou de déploiement. Quant au code spécifique impliqué dans nos expérimentations, il est constitué du code des ME primitifs encapsulant des logiciels patrimoniaux.

L'intégration d'un nouveau logiciel patrimonial dans Jade nécessite l'écriture du code d'un ME primitif constitué de 477 lignes de code Java et 16 lignes d'ADL en moyenne pour définir le composant de type ME (interfaces, packaging, etc).

		# classes Java	Code Java (lignes)	# fichiers ADL	ADL (lignes)
Code générique	AM d'auto-optimisation	14	2340	12	150
	AM d'auto-réparation	48	4750	26	430
	AME	4	840	4	200
	Représentation Système	40	6630	-	-
	Service de réservation	2	475	1	30
	Service d'installation	3	430	9	830
	Service de déploiement	21	2600	9	830
	Total	*132*	*18065*	***61***	***2470***
Code spécifique	Application Rubis - Apache - Tomcat - MySQL	 1 1 1	 150 150 150	 1 1 1	 11 11 11
	Intergiciel Apache Répartiteur requêtes	3 2	800 460	1 1	16 14
	Intergiciel Tomcat Répartiteur requêtes	3 2	550 460	1 1	12 14
	Intergiciel MySQL Répartiteur requêtes	4 2	760 810	3 1	40 14
	Total	***19***	***4290***	***11***	***143***
	Moyenne	***2***	***477***	***1***	***16***

Table 1. *Code générique et code spécifique aux applications dans Jade*

5.2. Mesures et observations

5.2.1. Environnement d'expérimentation

Nous utilisons l'application (Rubis, 200x) dans sa version 1.4.2 qui est accompagnée d'un émulateur de clients web pour générer une certaine charge. Pour les intergiciels de l'architecture J2EE, nous utilisons : le serveur web (Apache, 200x) version 1.3.29, le serveur d'entreprise (Tomcat, 200x) version 3.3.2 et le serveur de bases de données (Mysql, 200x) version 4.0.17. Pour les outils de répartition de requêtes permettant la duplication des intergiciels (Apache, Tomcat et MySQL), nous avons utilisé : (PLB, 200x) version 0.3 pour la répartition des requêtes aux serveurs Apache, le mécanisme de Tomcat 3.3.2 pour la répartition des requêtes aux serveurs

Tomcat, et (C-JDBC, 200x) version 2.0.2 pour la duplication et répartition des requêtes à une grappe de bases de données.

L'environnement matériel est constitué de sept nœuds Linux x86, 1.8 GHz/1 Go, interconnectées par un réseau Ethernet 100Mbits.

5.2.1. Auto-réparation

Dans une première expérimentation, nous considérons un scénario dans lequel le système est composé d'un serveur Apache, d'un serveur Tomcat et d'un serveur de base de données. Le système est soumis à une charge moyenne de 300 clients de l'application RUBiS. Nous provoquons une panne sur la machine hébergeant le serveur Apache et nous observons l'activité CPU sur cette machine. L'injection de panne est réalisée à l'aide d'une règle iptable (firewall linux) interdisant toute communication entrante et sortante à destination des autres nœuds de la grappe. La période des signaux heart beat est de 5 secondes.

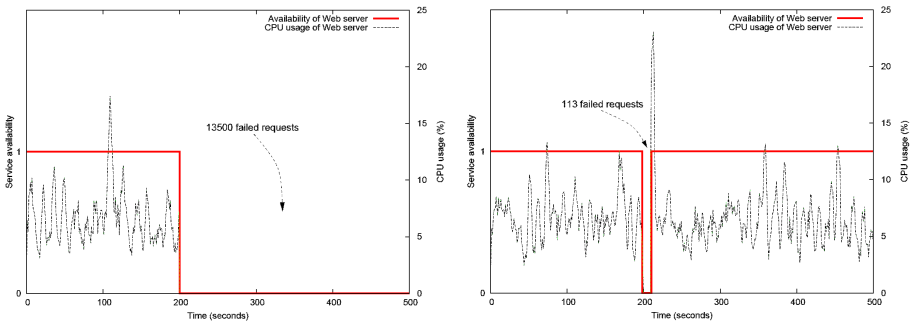

Figure 8. *Auto-réparation du serveur Apache*

Sur la figure 8 à gauche, sans l'auto-réparation implantée par Jade, lorsque la panne se produit (à l'instant t=200s), l'application RUBiS devient et reste indisponible jusqu'à la fin de l'expérimentation (provoquant une erreur HTTP pour 13500 requêtes). Lorsque le système d'auto-réparation de Jade est activé (figure 8 à droite), la panne est détectée et l'architecture J2EE est réparée en allouant une nouvelle machine et en re-déployant le serveur Apache sur cette nouvelle machine [5](en réalité pour cette expérimentation, nous avons redéployé le nouvel Apache sur la même machine que celle où nous avons provoqué la panne). La continuité de service est presque complètement garantie, étant donné que seulement 113 requêtes n'ont pu être traitées (pendant la période de réparation). Dans cette expérimentation,

[5] Lorsque le serveur Apache est relancé sur une machine différente, un mécanisme de redirection doit être utilisé entre les clients et le(s) serveur(s) Apache.

il n’y a pas de bufferisation des requêtes, qui imposerait une interception de celles-ci et par suite une intrusivité accrue sur la gestion de toutes les requêtes.

Dans une seconde expérimentation, le système est composé d'un serveur Apache, de deux serveurs Tomcat et d'un serveur de base de données. Nous provoquons une panne sur la machine hébergeant un des serveurs Tomcat. Sur la figure 9 à gauche, sans l'auto-réparation implantée par Jade, lorsque la panne se produit (à l'intant t=300s), l'application RUBiS reste disponible grâce au deuxième serveur, mais la charge du serveur affecté par la panne se reporte sur le serveur survivant (son occupation CPU passe de 25% à 50%). Avec le système d'auto-réparation de Jade (figure 9 à droite), on bénéficie à la fois de la continuité de service et de l'auto-réparation qui assure la stabilité des performances.

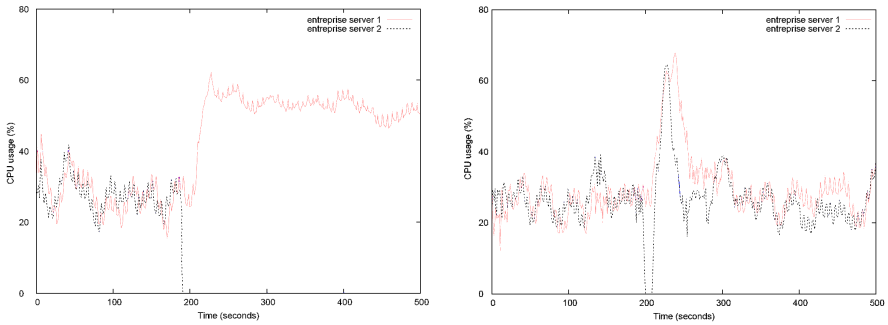

Figure 9. *Auto-réparation d'un serveur Tomcat dupliqué*

5.2.1. Auto-optimisation

Dans cette 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 une 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 sous-chargés, ce qui entraîne le retrait d’éléments dupliqués du niveau entreprise puis du niveau bases de données. La figure 10 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[6]. 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[7]. Notons que le paramétrage de cette sonde est réalisé en configurant le ME qui la représente.

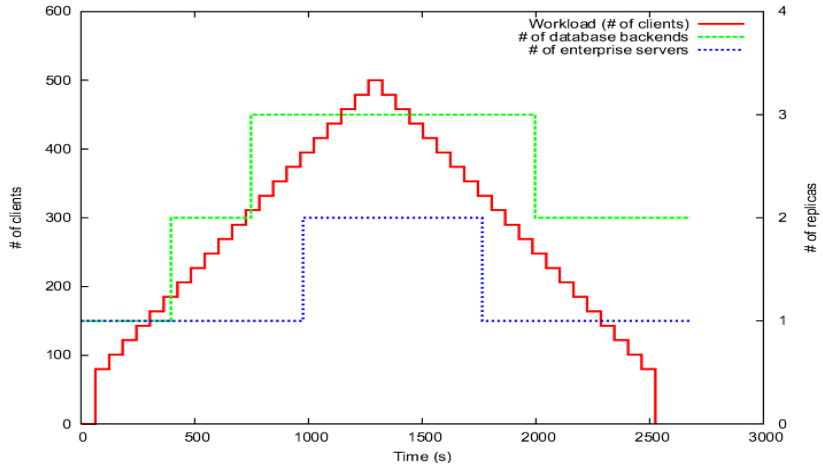

Figure 10. *Charge vs. degrés de duplication des serveurs*

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 11 en haut, nous considérons le tier base de données. Sans Jade, le CPU monte rapidement au maximum et ne redescend qu'en fin d'expérimentation. Avec Jade, lorsque le CPU atteint le seuil maximum autorisé pour le niveau base de données (représenté par la barre horizontale du haut), Jade alloue dynamiquement un nouveau nœud (courbe en escalier), régulant ainsi la charge. Les seuils sont fixés manuellement à l'issue d'une étude expérimentale (et manuelle) du système. Ils sont donc statiques et relativement délicats à fixer.

La figure 11 en bas présente des résultats similaires relatifs au niveau entreprise. Lorsque Jade est utilisé, la régulation de la charge (courbe du haut) s'effectue comme convenu. Notons qu'en l'absence de Jade, l'indice d'utilisation CPU reste

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

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

faible (courbe du bas) malgré la charge soumise 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.

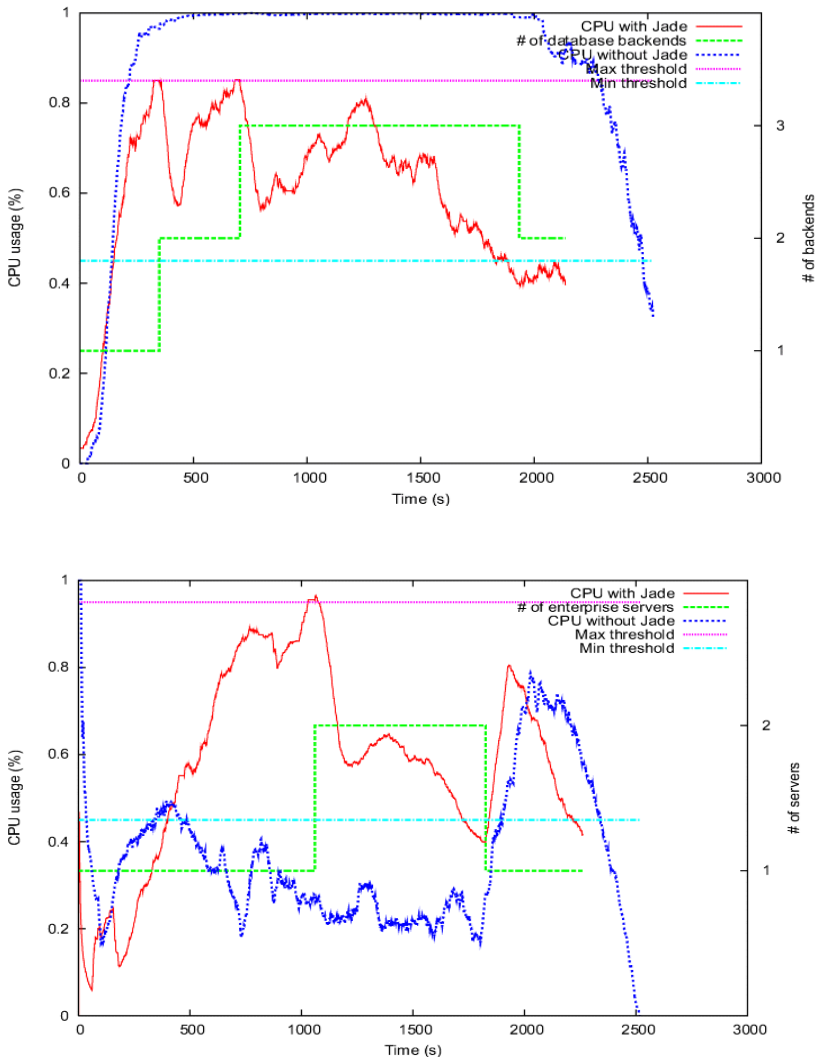

Figure 11. *Auto-optimisation avec Jade sur le tier MySQL et le tier Tomcat*

La figure 12 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.

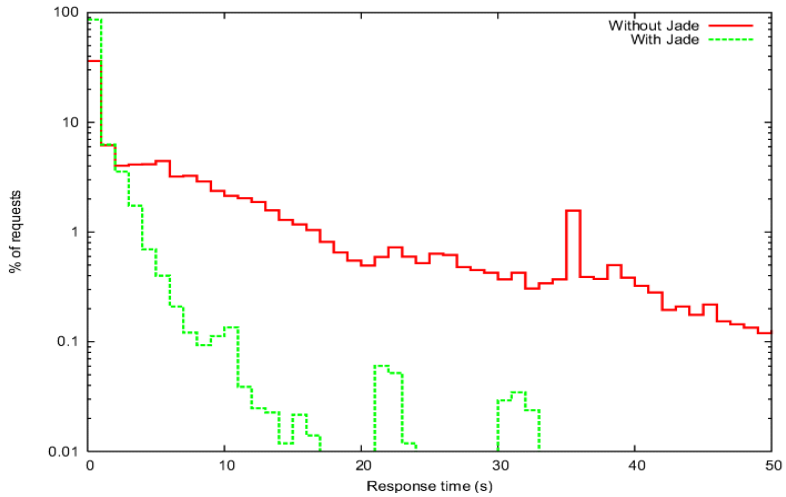

Figure 12. *Répartition des latences des requêtes clientes avec et sans Jade*

6. Comparaison aux autres travaux

L'informatique autonome est une approche très prometteuse pour simplifier la tâche difficile d'administration des infrastructures logicielles réparties ; elle vise à fournir des systèmes auto-réparables, auto-configurables (Kephart *et al.*, 2003).

Parmi les problèmes à résoudre, nous nous sommes concentrés sur la gestion de logiciels patrimoniaux et sur l'administration d'infrastructures complexes.

Dans la suite, nous survolons les travaux apparentés suivants deux directions : la gestion autonome basée sur des architectures logicielles, et la gestion dynamique de ressource dans les grappes de nœuds.

6.1. Gestion autonome basée sur des architectures logicielles

Une approche pour la gestion autonome de systèmes complexes consiste à s'appuyer sur la notion d'architecture logicielle (Dashofy *et al.*, 2002). Plusieurs projets de recherche ont exploré cette voie (Blair *et al.*, 2002)(Georgiadis *et al.*, 2002)(Oriezy *et al.*, 1999). Jade est un de ceux-ci, car nous exploitons une représentation du système administré sous la forme d'une architecture logicielle construite dans le modèle à composants Fractal. Cette représentation facilite le développement de gestionnaires autonomes implantant les politiques de gestion à l'exécution souhaitées. Le travail le plus proche de Jade est sans doute le système fondé sur Darwin décrit dans (Georgiadis *et al.*, 2002). Nous nous distinguons de ces travaux du fait que nous ciblons la gestion d'applications et infrastructures patrimoniales. Le modèle à composants Fractal est utilisé pour construire une représentation architecturale d'un environnement logiciel, sans faire d'hypothèse sur la nature de cet environnement. De plus, comparé à Darwin, l'exploitation du modèle à composants Fractal apporte une plus grande flexibilité à l'exécution.

6.2. Gestion dynamique de ressources dans les grappes de nœuds

Les politiques de gestion autonome que nous avons étudiées visent la gestion de ressources dans le cadre d'une application J2EE déployée dans une grappe de nœuds. De nombreux travaux se sont intéressés à la gestion dynamique de ressources dans des grappes de nœuds.

Une partie de ces travaux repose sur la disponibilité de tous les composants logiciels sur l’ensemble des nœuds administrés. Dans cette configuration, l’administration des ressources s’appuie sur un routage adapté des requêtes vers les différents nœuds (voir Neptune (Shen *et al.*, 2002) et DDSD (Zhu *et al.*, 2001)). 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 (Aron *et al.*, 2000) et Sharc (Urgaonkar *et al.*, 2004)).

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 (Appleby *et al.*, 2001), OnCall (Norris *et al.*, 2004) et Cataclysm (Urgaonkar *et al.*, 2004).

La plate-forme Jade décrite dans cet article 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 la forme d'un ensemble de composants administrables 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 les 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. Conclusions et perspectives

Cet article a présenté la conception, l'implantation et l'évaluation de Jade, un environnement d'administration autonome dont le but est de faciliter le développement de logiciels d'administration. Ces logiciels peuvent superviser une infrastructure logicielle et réagir à des événements comme des pannes ou des surcharges, et reconfigurer l'infrastructure en conséquence.

Jade fournit un canevas permettant la construction d'éléments administrables. Ces éléments peuvent encapsuler des logiciels patrimoniaux (comme Apache ou Tomcat dans les applications J2EE) ou être construits pour fournir des abstractions de plus haut niveau (comme des collections d'éléments ou des conteneurs). Les éléments administrables fournissent une interface uniforme, ce qui facilite l'implantation des programmes devant les administrer. Jade fournit également un canevas pour la construction de gestionnaires d'administration, implantant les programmes qui supervisent l'infrastructure logicielle administrée, par exemple pour réaliser une gestion d'auto-réparation ou d'auto-optimisation.

Afin de valider cette approche, nous avons appliqué Jade à l'administration d'une application J2EE, et plus précisément d'un site d'enchère similaire à eBay.com. Cette application s'appuie sur les intergiciels Apache, Tomcat et MySQL et ces intergiciels (ainsi que les applications qu'ils supportent) sont répliqués sur plusieurs nœuds d'une grappe afin d'assurer la disponibilité et le passage à l'échelle de l'application.

Pour qu'une nouvelle application puisse être administrée par Jade, il est nécessaire que les logiciels administrés soient encapsulés dans des composants. Cette encapsulation, si elle n'a pas déjà été réalisée, ne nécessite que quelques centaines de lignes de code. Une fois ces logiciels encapsulés, l'application bénéficie de nombreux services pouvant être réutilisés comme le déploiement, l'auto-réparation ou l'auto-optimisation. Une grande partie du code de ces services est générique et ces services peuvent être spécialisés en fonction de leur contexte d'utilisation.

Nous avons démontré l'utilisation de Jade pour gérer l'auto-réparation de l'infrastructure logicielle de notre application J2EE en grappe. L'auto-réparation consiste à ramener l'infrastructure logicielle dans un état équivalent à celui d'avant la panne. Nous avons également utilisé Jade pour gérer l'auto-optimisation de notre

application d'enchère. Le gestionnaire d'optimisation est capable d'allouer dynamiquement de nouveaux nœuds aux tiers MySQL ou Tomcat de l'architecture J2EE en cas de surcharge, et de retirer des nœuds en cas de sous-charge.

Nous poursuivons actuellement ces travaux dans différentes directions. Les algorithmes d'administration doivent pouvoir prendre en compte des situations complexes comme des pannes transitoires et des charges très fluctuantes. Par ailleurs, de nouveaux aspects d'administration doivent être considérés tels que par exemple la sécurité dans les environnements répartis. Nous prévoyons également de travailler sur l'association potentiellement conflictuelle de plusieurs politiques d'administration. Enfin, nous explorons l'utilisation de Jade pour l'administration d'autres infrastructures logicielles réparties comme celle des grilles de calcul ou celle des infrastructures pair-à-pair.

Ces travaux ont bénéficié du support de l'ANR à travers le projet Selfware (ANR-05-RNTL-01803).

12. Bibliographie

Apache, Apache HTTP Server, The Apache Software Foundation, http://httpd.apache.org, 200x.

Appleby K., Fakhouri S., Fong L., Goldszmidt G., Kalantar M., Krishnakumar S., Pazel D.P., Pershing J., Rochwerger B., « Oceano - SLA based management of a computing utility », *7th IFIP/IEEE International Symposium on Integrated Network Management*, 2001.

Aron M., Druschel P., and Zwaenepoel W., « Cluster Reserves: a mechanism for resource management in cluster-based network servers », *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ACM Press, 2000.

Blair G., Coulson G., Blair L., Duran-Limon H., Grace P., Moreira R., and Parlavantzas N., « Reflection, self-awareness and sef-healing in OpenORB », *1st Workshop on Self-Healing Systems, WOSS 2002, ACM*, 2002.

Bouchenak S., Boyer F., Hagimont D., Krakowiak S., Mos A., De Palma N., Quéma V., Stefani J. B., « Architecture-Based Autonomous Repair Management: An Application to J2EE Clusters », *24th IEEE Symposium on Reliable Distributed Systems (SRDS-2005)*, Orlando, FL, USA, October 2005.

Bruneton E., Coupaye T., Leclercq M., Quéma V., Stefani J.-B., « The Fractal Component Model and its Support in Java », Software Practice and Experience, special issue on Experiences with Auto-adaptive and Reconfigurable Systems, volume 36, number 11-12, 2006.

Cecchet E., Chanda A., Elnikety S., Marguerite J., Zwaenepoel W., « Performance Comparison of Middleware Architectures for Generating Dynamic Web Content », *4th ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, Brazil, June 2003.

Cecchet E., Marguerite J., Zwaenepoel W., « C-JDBC: Flexible Database Clustering Middleware », *USENIX Annual Technical Conference, Freenix track*, Boston, MA, June 2004.

C-JDBC, Clustered JDBC, The ObjectWeb Consortium, http://c-jdbc.objectweb.org, 200x.

Dashofy E. M., Van der Hoek A., Taylor R. N, « Towards architecture-based self-healing systems », *Proceedings WOSS '02, ACM*, 2002.

Ganek A. G., Corbi T. A., « The dawning of the autonomic computing era », *IBM Systems Journal - Special issue on autonomic computing*, Volume 42, Number 1, 2003.

Georgiadis I., Magee J., Kramer J., « Self-organizing software architecture for distributed systems », *1st Workshop on Self-Healing Systems, WOSS 2002, ACM*, 2002.

Hagimont D., Bouchenak S., De Palma N., Taton C., « Autonomic Management of Clustered Applications », *IEEE International Conference on Cluster Computing*, Barcelona September 25th-28th, 2006.

He X., Yang O., « Performance Evaluation of Distributed Web Servers under Commercial Workload », *Embedded Internet Conference 2000*, San Jose, CA, September 2000.

Iyengar A., MarcNair E., Nguyen T., « An Analysis ofWeb Server Performance », *IEEE Global Telecommunications Conference (GLOBECOM'97)*, Phoenix, AR, November 1997.

J2EE, Java 2 Platform Enterprise Edition, Sun Microsystems, http://java.sun.com/j2ee, 200x.

Jonas, Jonas Java Open Application Server, The ObjectWeb Consortium, http://jonas.objectweb.org, 200x.

Kephart J. O., Chess D. M., « The Vision of Autonomic Computing », *IEEE Computer Magazine*, Volume 36, Number 1, 2003.

MySQL, MySQL AB, http://www.mysql.com, 200x.

Norris J., Coleman K., Fox A., Candea G., « OnCall: Defeating Spikes with a Free-Market Application Cluster », *1st International Conference on Autonomic Computing*, May 2004.

Oriezy P., Gorlick M., Taylor R., Johnson G., Medvidovic N., Quilici A., Rosenblum D., Wolf A., « An Architecture-Based Approach to Self-Adaptive Software », *IEEE Intelligent Systems*, 14(3), 1999.

PLB, Pure Load Balancer, http://plb.sunsite.dk, 200x.

Rubis, The Rubis Benchmark, The ObjectWeb Consortium, http://rubis.objectweb.org, 200x.

Shen K., Tang H., Yang T., Chu L., « Integrated resource management for cluster-based internet services », *5th USENIX Symposium on Operating System Design and Implementation (OSDI-2002)*, December 2002.

Tomcat, Apache Tomcat, The Apache Software Foundation, http://tomcat.apache.org, 200x.

Urgaonkar B., Shenoy P., Chandra A., Goyal P., « Dynamic Provisiong of Multi-Tier Internet Applications », *2nd International Conference on Autonomic Computing (ICAC-2005)*, Seattle, WA, June 2005.

Urgaonkar B., Shenoy P., « Sharc: Managing CPU and network bandwidth in shared clusters », *IEEE Transactions on Parallel Distribibuted Systems*, 15(1):2-17, 2004.

Urgaonkar B., Shenoy P., « Cataclysm: Handling Extreme Overloads in Internet Services », Technical Report, Department of Computer Science, University of Massachusetts, November 2004.

Zhu H., Tang H., Yang T., « Demand-driven service differentiation in cluster-based network servers », IEEE Infocom 2001.

Sara Bouchenak est Maître de Conférences d'Informatique à l'Université Grenoble I depuis 2004. Elle est membre de l'équipe Sardes (UJF/INPG/CNRS/INRIA) où elle s'intéresse aux concepts sous-jacents à la construction de systèmes répartis, à la gestion de ressources, à la qualité de service et à l'autonomie de ces systèmes. Auparavant, elle a obtenu un DEA de l'Université Grenoble I en 1998 et un Doctorat en Informatique de l'Institut National Polytechnique de Grenoble en 2001. Elle a également été en visite à l'Ecole Polytechnique Fédérale de Lausanne en 2003.

Fabienne Boyer est Maître de conférences à l'Université Joseph Fourier et membre de l'équipe Sardes (UJF/INPG/CNRS/INRIA) au sein du laboratoire LIG (Laboratoire d'Informatique de Grenoble). Elle effectue ses recherches sur la construction d'infrastructures réparties autonomiques et travaille actuellement sur le système JADE. Elle a obtenu en 1993 un doctorat de l'Université Joseph Fourier et a ensuite rejoint l'équipe Sirac travaillant dans le domaine des applications réparties au sein du laboratoire LSR.

Noël De Palma est docteur de l'université Joseph Fourier de Grenoble depuis 2001. Il est maître de conférence à l'Institut National Polytechnique de Grenoble depuis 2002 où il enseigne la conception de systèmes distribués. Il effectue ses recherches au sein du projet Sardes (UJF/INPG/CNRS/INRIA) sur les systèmes autonomes et le système JADE en particulier. Il est actuellement chercheur invité au laboratoire SICS de Stockholm (KTH - The Royal Institute of Technology) sur les systèmes autonomes à grande échelle.

Daniel Hagimont est Professeur à l'Institut National Polytechnique de Toulouse (INPT) et membre du laboratoire de recherche IRIT où il anime un groupe travaillant sur les systèmes d'exploitation, les systèmes répartis et les intergiciels. Il a obtenu en 1993 un doctorat de l'Institut National Polytechnique de Grenoble. Après une année post-doctorale à l'Université de Colombie Britannique, Vancouver, en 1994, il a rejoint le centre INRIA de Grenoble en 1995. Il est Professeur à l'INPT depuis 2005.

Sylvain Sicard est doctorant à l'Institut National Polytechnique de Grenoble. Il travaille sur l'auto-réparation dans les infrastructures logicielles distribuées, dans le contexte des systèmes autonomes.

Christophe Taton est doctorant à l'Institut National Polytechnique de Grenoble. Il travaille sur l'auto-optimisation d'applications distribuées dans les systèmes autonomes.