HAL
open science

Performance, disponibilité et coût de services Internet adaptatifs

Jean Arnaud

To cite this version:

Jean Arnaud. Performance, disponibilité et coût de services Internet adaptatifs. Réseaux et télécommunications [cs.NI]. Université Joseph-Fourier - Grenoble I, 2010. Français. NNT : . tel-00529936

HAL Id: tel-00529936

https://theses.hal.science/tel-00529936v1

Submitted on 27 Oct 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

N° attribué par la bibliothèque

THÈSE

pour obtenir le grade de **DOCTEUR de l'Université de Grenoble**

Spécialité : « Informatique »

préparée au laboratoire LIG dans le cadre de **l'École Doctorale « Mathématiques, Sciences et Technologies de l'Information, Informatique »**

préparée et soutenue publiquement par

Jean Arnaud

le 21 Septembre 2010

Titre :

Performance, disponibilité et coût de services Internet adaptatifs

---

sous la direction de Sara Bouchenak et Jean-Bernard Stefani

JURY

Pr Jean-François Méhaut	Président
Dr Françoise Baude	Rapporteur
Pr Pierre Sens	Rapporteur
Dr Guillaume Pierre	Examinateur
Dr Sara Bouchenak	Examinateur
Pr Jean-Bernard Stefani	Examinateur

"There is one rule for the industrialist and that is : Make the best quality of goods possible at the lowest cost possible, paying the highest wages possible."

Henry Ford.

Remerciements

Je tiens tout d'abord à remercier Sara Bouchenak pour son encadrement et son attention tout au long de cette thèse. Son implication m'a permis de progresser et d'apprendre beaucoup sur le monde de la recherche. Merci à Jean-Bernard Stefani pour m'avoir fait confiance quant à mes directions de recherche, et permis d'effectuer cette expérience qui, bien que difficile, est extrêmement enrichissante.

Je voudrais ensuite remercier les membres du jury pour avoir accepté d'évaluer cette thèse, et plus largement toutes les personnes rencontrées au cours de ces dernières années qui se sont montrées intéressées par mes travaux.

Merci également à tous les membres, passés ou actuels, du projet Sardes pour leur compagnie au travail ou en dehors, en particulier Thomas, Sergueï, Stéphane, Christophe, Jeremy, Ludovic, Alan et Damien, ainsi que toutes celles et tous ceux que je ne nomme pas mais qui se seront bien évidemment reconnus. Je pense également à Émeline pour son soutien et ses encouragements tout au long de cette thèse, ainsi qu'à tous ceux qui, famille ou amis, m'ont motivé à poursuivre et à voir le côté positif des choses.

Et surtout, merci aux personnes avec lesquelles j'ai pu échanger, progresser et découvrir de nouveaux horizons.

Résumé

La multiplication des services Internet et la hausse de leur utilisation entraînent des charges importantes sur les services Internet. Ces charges peuvent de plus varier dans le temps, souvent de manière imprévisible. Cependant la qualité de service de ces applications doit toujours rester dans des limites acceptables, les utilisateurs pouvant choisir le service utilisé en fonction de la qualité de service perçue. Par ailleurs, l'impact économique et écologique de ces services peut devenir problématique, principalement à cause de la consommation électrique des serveurs. Les fournisseurs de services Internet cherchent donc à minimiser les coûts de fonctionnement, tout en préservant la qualité de service fournie aux utilisateurs. Plusieurs approches existent pour administrer des services Internet. Cependant, la plupart ne considèrent qu'un aspect de qualité de service, ne s'adaptent pas seules à toutes les variations de charge, ou se contentent d'une approche "au mieux" *(best-effort)*, sans garantie de la qualité de service fournie.

Dans cette thèse, nous proposons un contrôle adaptatif de services Internet, fournissant à la fois des garanties de performance et de disponibilité de service, tout en minimisant le coût de fonctionnement des services. Les contributions de cette thèse sont les suivantes. Tout d'abord, un modèle analytique de prédiction des performances, de la disponibilité et du coût d'un service Internet en fonction de charges et de configurations variables du service, est proposé. Puis une quantification de l'utilité du service Internet, en termes de niveaux de performance et de disponibilité et en termes de coût, est définie. Ensuite, une méthode de planification de capacité, permettant de calculer la configuration optimale d'un service Internet pour garantir des contraintes de qualité de service et minimiser le coût du service est proposée. Enfin, un contrôle adaptatif de services Internet est fourni pour prendre en compte tout type de variation de charge des services Internet et ceci via un calibrage en ligne automatique des modèles et planification de capacité sous-jacents.

L'approche proposée est implantée dans un prototype fonctionnel appelé MoKa. MoKa a été appliqué avec succès pour le contrôle d'un service Internet de vente en ligne constitué de serveurs Web et de serveurs de bases de données. Les expériences menées ont montré que le service contrôlé par MoKa était capable de s'adapter en ligne à diverses variations de charge et continuait ainsi de garantir les contraintes de qualité de service tout en effectuant des économies de ressources significatives.

Mots-clés : Contrôle, Modélisation, Optimisation, Planification de capacité, Qualité de service, SLA, Services Internet.

Abstract

The increasing amount of Internet services and their growing usage lead to huge, changing and unpredictable workloads on these services. However, the quality of service of these applications must remain within acceptable limits and respect the service level agreement (SLA) contracted with their users. Furthermore, the economical and ecological impact of these services could become a problem, mainly due to the servers energy consumption. Thus Internet service providers aim to minimize service costs while guaranteeing the quality of service. Several existing approaches aim to manage Internet services. However, most of them are restricted to one particular aspect of quality of service, can not adapt themselves to different types of workload variations, or are limited to a best-effort behavior, without any guaranty on provided quality of service.

In this thesis, we propose an adaptive control of Internet services, providing guaranties on service performance and availability, while minimizing the service cost. The contributions are the following. First, we propose an analytic model for predicting the performance, availability and cost of an Internet service depending on varying workloads and service configurations. Second, we quantify the Internet service utility, in terms of performance and availability level, and in terms of cost. Third, a capacity planning method is proposed in order to calculate the optimal configuration of an Internet service that matches quality of service constraints while minimizing the service cost. Finally, we provide an adaptive control of Internet services to take into account all kind of workload variations, through an automatic on-line monitoring, capacity planning and reconfiguration of Internet services.

The proposed approach is implented in a software prototype called MoKa. MoKa was successfully applied to the control of an e-commerce Internet service, consisting in several Web and database servers. The results of the experiments show that the MoKa controlled service is able to self-adapt when facing different workload variations, and to continue to guarantee the quality of service constraints whith significant savings in terms of resource usage.

Keywords : Control, Modeling, Optimization, Capacity Planning, Quality of Service, SLA, Internet Services

Table des matières

1 Introduction **1**
1.1 Contexte et motivations . . . 1
1.2 Objectifs et contributions scientifiques . . . 2
1.3 Organisation du document . . . 4

2 Contexte **5**
2.1 Organisation des services Internet . . . 5
2.2 Qualité et coût des services Internet . . . 9
2.3 Charge des services Internet . . . 10
2.4 Configuration des services Internet . . . 12
2.5 Synthèse . . . 13

3 État de l'art **15**
3.1 Présentation générale . . . 15
3.1.1 Types de contrôle . . . 15
3.1.2 Méthodes d'analyse et de décision . . . 17
3.2 Contrôle de configuration locale de services Internet . . . 17
3.2.1 Approches à base d'heuristiques . . . 17
3.2.2 Approches avec garanties . . . 22
3.3 Contrôle de configuration architecturale de services Internet . . . 23
3.3.1 Approches à base d'heuristiques . . . 23
3.3.2 Approches avec garanties . . . 24
3.4 Synthèse . . . 27

4 Motivations et contributions scientifiques **31**
4.1 Illustration du problème . . . 31
4.2 Objectifs . . . 36
4.3 Contributions scientifiques . . . 38
4.4 Synthèse . . . 40

5 Modélisation des services Internet **41**
5.1 Présentation générale . . . 41
5.1.1 Entrées du modèle . . . 41
5.1.2 Sorties du modèle . . . 42
5.2 Principes du modèle . . . 42
5.2.1 Files d'attente . . . 42
5.2.2 Files d'attente pour services multi-étagés dupliqués . . . 43

5.3 Algorithme de prédiction de performance, de disponibilité et de coût 46
5.4 Synthèse 48

6 Planification de capacité des services Internet 49
6.1 Objectifs 49
6.2 Fonction d'utilité 50
6.3 Algorithme de planification de capacité 51
6.4 Principes de preuves 55
6.5 Synthèse 58

7 Contrôle adaptatif de services Internet 61
7.1 Présentation générale 61
7.2 Monitorage 62
7.2.1 Sondes de monitorage 62
7.2.2 Monitorage global 64
7.3 Paramétrage interne 65
7.4 Contrôleur 70
7.5 Reconfiguration 72
7.6 MoKa : prototype de contrôleur adaptatif de services Internet 73
7.6.1 Logiciel MoKa 73
7.6.2 Détails de mise en oeuvre 76
7.6.3 Interfaces utilisateur 78
7.6.4 Données factuelles 82
7.7 Synthèse 82

8 Expérimentations et évaluations 85
8.1 Environnement d'évaluation 85
8.2 Évaluation du modèle 86
8.2.1 Scénario d'évaluation 86
8.2.2 Évaluation de la précision du modèle 88
8.3 Évaluation de la planification de capacité 90
8.3.1 Scénario d'évaluation 90
8.3.2 Évaluation de l'optimalité de la planification de capacité 90
8.4 Évaluation du contrôle adaptatif 91
8.4.1 Scénario d'évaluation 91
8.4.2 Évaluation de l'adaptabilité des services Internet 92
8.4.3 Autres techniques de contrôle 96
8.5 Synthèse 99

9 Conclusion et perspectives 101
9.1 Synthèse 101
9.2 Perspectives 102

Références bibliographiques 108

Table des figures

2.1 Schéma client/serveur 6
2.2 Serveur multiprogrammé 7
2.3 Exemple de service multi-étagé 7
2.4 Exemple de service Internet multi-étagé dupliqué 8
2.5 Variation de la quantité de charge au cours d'une journée 12

3.1 Exemple de contrôle d'admission 16
3.2 Approximation parabolique des performances - type de charge en lecture seule 18
3.3 Approximation parabolique des performances - type de charge en écriture . . 19

4.1 Impact de la configuration sur la performance 32
4.2 Impact de la configuration sur la disponibilité 33
4.3 Impact de la configuration sur le coût 33
4.4 Impact de la charge sur la performance 34
4.5 Impact de la charge sur la disponibilité 34
4.6 Impact de la charge sur le coût 35
4.7 Comportement non linéaire de la latence 36
4.8 Comportement non linéaire du débit 37

5.1 Modèle pour les services Internet multi-étagés 41
5.2 Réseau ouvert de files d'attente 43
5.3 Réseau fermé de files d'attente 43
5.4 Modélisation d'une application multi-étagée 44
5.5 Modélisation d'une application multi-étagée - prise en compte de la configuration locale 45

6.1 Planification de capacité pour les applications multi-étagées 49
6.2 Fonction d'utilité 50
6.3 Fonction d'utilité 52
6.4 Planification de capacité 53

7.1 Contrôleur adaptatif 62
7.2 Interaction client/serveur et *think time* 64
7.3 Recherche des paramètres internes optimaux du modèle 65
7.4 Divergence du modèle en fonction des variables internes 68
7.5 Évolution de la quantité de charge 69
7.6 Évolution des ratios de visite 69

7.7 Évolution des paramètres S_i 70
7.8 Latence prédite et latence mesurée 71
7.9 Débit prédit et débit mesuré 71
7.10 Taux de rejet prédit et taux de rejet mesuré 72
7.11 Principaux paquets de MoKa 73
7.12 Interface de programmation de modèle de services Internet 74
7.13 Javadoc de l'interface pour les algorithmes de planification de capacité 75
7.14 Javadoc de l'interface du contrôleur 75
7.15 Architecture du contrôleur 76
7.16 Architecture d'une application JMX 77
7.17 Interface Web de MoKa - évolution de la quantité de charge 80
7.18 Interface Web de MoKa - évolution du type de charge 81

8.1 Variation de charge 87
8.2 Latences mesurées et prédites 88
8.3 Débits mesurés et prédits 89
8.4 Taux de rejet mesurés et prédits 89
8.5 Précision de la planification de capacité - configuration architecturale 90
8.6 Précision de la planification de capacité - configuration locale 91
8.7 Charge du service Internet 92
8.8 Latences du service Internet 93
8.9 Taux de rejet du service Internet 93
8.10 Débit du service Internet 94
8.11 Évolution de la configuration architecturale du système contrôlé 95
8.12 Évolution de la configuration locale du système contrôlé 95
8.13 Évolution de la fonction d'utilité 96
8.14 Comparaison du coût des configurations 97
8.15 Latence 97
8.16 Taux de rejet 98
8.17 Coût 99

Chapitre 1

Introduction

1.1 Contexte et motivations

Internet permet désormais de fournir des services de toute nature, qui prennent le pas sur leur équivalent classique. Les exemples d'application des services Internet sont nombreux. Ils permettent entre autres de communiquer, d'écouter de la musique, de participer à des réseaux sociaux, ou encore d'acheter et de vendre des biens en ligne. Certains de ces services, comme eBay, Amazon, Youtube ou encore Facebook revendiquent des millions d'utilisateurs, et leur croissance en terme d'usagers se poursuit.

Depuis les pages HTML statiques des premiers sites Internet, l'évolution a mené à la construction de services en ligne toujours plus complexes, et proposant de nombreuses fonctionnalités. La plupart des services Internet actuels fournissent ainsi un contenu dynamique et personnalisé à chaque utilisateur. Cet accroissement des fonctionnalités et de la qualité perçue des services Internet est en partie la raison de leur succès actuel.

Les principaux critères de qualité de service perçus par les clients sont relatifs à la performance et à la disponibilité du service. Ainsi, la performance peut être représentée par la latence du service Internet, c'est-à-dire le temps nécessaire au traitement d'une requête, et la disponibilité, concernerait la probabilité qu'à une requête d'être traitée avec succès par le service ou rejetée pour cause de surcharge. La qualité de service d'une application est un critère fondamental. En effet, les clients cesseront d'utiliser un service Internet lent ou peu fiable. Un *contrat de qualité de service* est donc souvent utilisé pour spécifier les niveaux de performance et de disponibilité à respecter.

Plus de fonctionnalités sur un service Internet impliquent une charge de calcul plus importante pour la génération dynamique des réponses aux clients, ainsi qu'une complexité croissante dans l'architecture logicielle et matérielle de ces services. Pour répondre à ces défis, les architectures des services Internet ont évolué vers des systèmes dupliqués et multi-étagés, permettant de supporter des charges importantes pour des services complexes. Dans ce contexte, la configuration d'un service Internet concerne à la fois la quantité de serveurs hébergeant l'application, mais également le paramétrage de chacun de ces serveurs. Le coût d'un service Internet est directement lié au nombre de serveurs alloués à ce service, du fait de leur consommation électrique importante.

L'hébergement des services Internet est souvent externalisé dans des centres d'hébergement spécialisés, chargé de maintenir le fonctionnement de ces applications complexes. Ils mutualisent ainsi les moyens techniques et humains propres en hébergeant plusieurs ser-

vices Internet. La taille de ces centres d'hébergement a suivi la croissance en terme d'utilisateurs des services hébergés. Actuellement, des services Internet populaires peuvent nécessiter plusieurs dizaines de milliers de serveurs pour fonctionner. L'importante consommation électrique de ces centres d'hébergement a évidemment un coût financier et écologique non négligeable. A cette échelle, la configuration d'un service Internet est donc une problématique stratégique, impactant directement le coût de fonctionnement du service ainsi que la qualité de service fournie aux utilisateurs. En effet, allouer plus de ressources à un service Internet permet généralement d'améliorer sa qualité de service, mais accroît son coût.

Par ailleurs, la charge subie par un service Internet est directement liée au comportement des clients y accédant. Ce comportement n'étant pas constant au cours du temps, la plupart des services Internet doivent donc faire face à de fortes variations dans leur fréquentation. Par exemple un site de vente en ligne sera fortement sollicité avant une période de fêtes, et un site d'actualités verra sa charge augmenter lors d'évènements importants. Mais la plupart des variations dans la fréquentation des sites sont difficiles voire impossibles à prévoir. Cependant, malgré des variations importantes dans la quantité et le type des requêtes reçues, un service Internet doit continuer à fournir à ses clients une qualité de service respectant le contrat de qualité de service spécifié.

Nous cherchons donc à trouver une configuration permettant de respecter les contraintes de qualité de service pour une charge donnée. Généralement, il existe un nombre important (théoriquement infini) de configurations permettant de respecter le contrat de qualité de service. Parmi ces configurations, on s'intéresse à celle minimisant le coût du service. Cette configuration sera la configuration optimale pour une charge et un contrat de qualité de service donnés. La recherche de cette configuration optimale devra être effectuée de manière dynamique, pendant le fonctionnement du service Internet. En effet, la quantité et le type de charge étant variables, la configuration optimale peut évoluer au cours du temps. Ceci implique de pouvoir détecter en ligne les variations des caractéristiques de la charge. Il faut enfin pouvoir appliquer la configuration optimale calculée sans interrompre le fonctionnement du service.

1.2 Objectifs et contributions scientifiques

Nos principaux objectifs sont les suivants :
- Calculer la configuration optimale qui, d'une part, garantit le contrat de qualité de service (SLA) et, d'autre part, minimise le coût de fonctionnement du service Internet.
- Considérer les contrats de qualité de service qui stipulent à la fois des contraintes de performance et des contraintes de disponibilité de service.
- Fournir un contrôle adaptatif en ligne des services Internet qui détecte et prenne en compte automatiquement la variation de la quantité et du type de charge.

Les principales contributions de cette thèse sont les suivantes [3, 8, 7, 6, 5, 4] :
- Un *modèle analytique* de prédiction des performances, de la disponibilité et du coût d'un service Internet en fonction de charges et de configurations variables du service Internet.
- Une quantification de l'*utilité* du service Internet, en termes de niveaux de performance et de disponibilité et en termes de coût.
- Une méthode de *planification de capacité*, permettant de calculer la configuration opti-

male d'un service Internet pour garantir des contraintes de qualité de service et minimiser le coût du service.
– Un *contrôle adaptatif* de services Internet est fourni pour prendre en compte tout type de variation de charge des services Internet et ceci via un calibrage en ligne automatique des modèles et planification de capacité sous-jacents.

Modèle analytique

Nous proposons un modèle analytique qui fournit une estimation précise des performances, de la disponibilité et du coût d'un service Internet avec une configuration et une charge données. Ce modèle est basé sur la théorie des files d'attente et étend l'algorithme connu MVA [47]. Les nouvelles fonctionnalités suivantes sont proposées : la prise en compte de la duplication des serveurs dans les services Internet (configuration architecturale), la prise en compte du contrôle d'admission dans les serveurs (configuration locale), la prise en compte de différents types de charge des services, ainsi que la prédiction de la disponibilité et du coût des services Internet.

Les modèles à file d'attente classiquement utilisés sont limités par la multiplication du nombre de paramètres utilisés et par la difficulté du calibrage de ces paramètres. Nous proposons ici une solution de calibrage automatique du modèle, facilitant ainsi l'usage du modèle. Le chapitre 5 détaille la conception du modèle.

Fonction d'utilité

Nous proposons une fonction d'utilité agrégeant les critères de performance, de disponibilité et de coût d'un service Internet en une métrique unique. La fonction d'utilité prend en compte les objectifs de performance et de disponibilité tels que spécifiés par le SLA, ainsi que l'objectif de minimisation du coût. Ainsi, pour une charge donnée, différentes configurations d'un service Internet peuvent avoir différentes valeurs d'utilité. Une configuration ayant la valeur d'utilité la plus élevée est alors considérée comme une configuration optimale, la configuration recherchée par la planification de capacité.

Planification de capacité pour les services Internet

Nous proposons une méthode de planification de capacité permettant de trouver la configuration optimale d'un service Internet, pour une quantité et un type de charge donnés. La configuration optimale d'un service Internet est celle qui garantit les contraintes de performance et de disponibilité spécifiées par le SLA, et qui minimise le coût de fonctionnement du service. La planification de capacité se base sur le modèle analytique de prédiction de performance et de disponibilité, et sur la fonction d'utilité proposés pour déterminer la configuration optimale. Le chapitre 6 présente la méthode de planification de capacité proposée.

Contrôle adaptatif

Nous proposons un contrôle adaptatif de services Internet pour prendre en compte automatiquement toutes les variations de charge des services. Ceci concerne à la fois la variation de la quantité de charge et du type de charge, ce dernier n'ayant jamais été pris en compte auparavant. Le contrôle adaptatif proposé se base sur un calibrage en ligne des modèles et

planification de capacité sous-jacents. MoKa est un prototype mettant en oeuvre ce contrôle adaptatif, et a été appliqué avec succès à un service Internet de vente en ligne. Le chapitre 7 présente le contrôle adaptatif dans MoKa.

Évaluation et expérimentations

Nous avons conduit une campagne d'évaluation expérimentale basée sur le service Internet de vente en ligne TPC-W [56], déployé sur des serveurs Web et des serveurs de base de données. L'évaluation a pu montrer que la précision du modèle analytique proposé, l'optimalité de la configuration appliquée au service Internet pour garantir le SLA et minimiser le coût et enfin, le contrôle adaptatif qui détecte automatiquement toute variation de quantité et de type de charge et reconfigure le système contrôlé. Ces résultats sont présentés dans le chapitre 8.

1.3 Organisation du document

Le reste du document est organisé comme suit. Le chapitre 2 présente le contexte des services Internet. Dans le chapitre 3, nous étudions les travaux existants relatifs au contrôle des services Internet. Les motivations et les contributions scientifiques sont présentées dans le chapitre 4. Le chapitre 5 décrit la modélisation proposée pour les services Internet multi-étagés. Dans le chapitre 6, nous présentons la méthode de planification de capacité. Le chapitre 7 présente la réalisation du contrôle adaptatif de services Internet dans le prototype MoKa. Le chapitre 8 décrit l'évaluation de MoKa sur un service Internet en ligne. Le chapitre 9 présente enfin nos conclusions et les perspectives ouvertes par nos travaux.

Chapitre 2

Contexte

Ce chapitre présente le contexte technique de notre étude. Nous y définissons l'organisation des services Internet, leur qualité de service, leur charge et leur configuration.

2.1 Organisation des services Internet

La complexité croissante des applications proposées sur Internet ainsi que le nombre toujours plus important d'utilisateurs accédant à ces services sont deux défis majeurs auxquels doivent faire face les concepteurs de services Internet. Pour répondre à ces défis, les architectures des services Internet ont évolué, depuis les architectures client/serveur classiques vers des architectures multi-étagées avec duplication de serveurs.

Architecture client-serveur

Traditionnellement, les services Internet sont organisés selon l'architecture classique client-serveur. Lorsqu'un utilisateur, ou *client*, veut accéder à un service Internet, il émet une requête correspondant à sa demande. Cette requête est reçue via le réseau par le *serveur* hébergeant le service Internet. Ce dernier calcule une réponse appropriée à la requête, puis retourne une réponse au client, comme illustré sur la figure 2.1. Les clients peuvent utiliser différents protocoles pour communiquer avec le serveur. Ces protocoles dépendent principalement du type de service fourni par le service Internet. Pour les services basés sur un site Web, le protocole le plus fréquemment utilisé est HTTP *(HyperText Transfer Protocol)* ou sa version sécurisée HTTPS. D'autres protocoles sont également utilisés, comme IMAP pour les services de courrier électronique ou FTP pour le transfert de fichiers.

Un même serveur peut traiter les requêtes de différents clients accédant en concurrence au service. La hausse du trafic sur les services Internet a vite montré les limites d'un traitement séquentiel des requêtes. En effet, dans le modèle simple, chaque requête doit attendre la fin du traitement de la requête précédente pour s'exécuter. Afin d'augmenter le nombre de requêtes traitées, les serveurs utilisent le principe de la *multiprogrammation*. La multiprogrammation consiste à avoir plusieurs processus (ou processus légers, *threads*) de traitement des requêtes clients fonctionnant en parallèle dans un même serveur. Ainsi, chaque requête accédant au serveur voit son traitement confié à un des processus. Dans l'exemple de la figure 2.2, 3 clients accèdent à un serveur multiprogrammé. Trois processus fonctionnent donc en parallèle dans le serveur pour répondre aux clients. Le traitement des requêtes client en

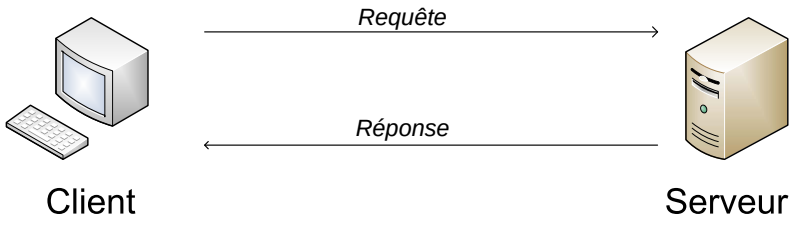

FIGURE 2.1 – Schéma client/serveur

parallèle, plutôt qu'en série, permet ainsi de recouvrir les temps d'inactivité de certains processus, par exemple en attente d'une entrée/sortie.

Architecture multi-étagée

L'augmentation de la charge subie par les services Internet mène à des problématiques de performance concernant le traitement des requêtes des clients, le serveur hébergeant le service Internet pouvant être surchargé. Parallèlement à une augmentation de la charge subie, la complexité des services Internet a également fortement augmenté. Des problématiques de présentation des résultats, au stockage des données, en passant par des calculs parfois complexes à réaliser, les tâches à effectuer par un service Internet sont nombreuses et spécifiques. Or ces problématiques se trouvent souvent mêlées à l'intérieur d'un même serveur. Ceci entraîne des difficultés de développement et de maintenance, et parfois des failles de sécurité dans le service Internet.

Pour des raisons de performances mais aussi de fiabilité, les différents traitements logiques d'un service Internet sont séparés sur autant de serveurs logiciels. Lorsque chaque serveur logiciel composant le service Internet est localisé sur une machine dédiée, ceci permet d'améliorer les performances et de faciliter les opérations de maintenance. Les architectures multi-étagées sont ainsi une évolution de l'architecture client-serveur basique, permettant de gérer la charge et la complexité croissante des services Internet en séparant les problématiques.

Une architecture multi-étagée est composée d'une série de M étages (*tiers* en anglais) $T_1, T_2, ..., T_M$. Chaque étage effectue une partie spécifique du traitement des requêtes. Par exemple, le service multi-étagé de la figure 2.3 est composé de 3 étages, T_1 générant les pages Web renvoyées aux clients, T_2 gérant la logique métier de l'application, et T_3 responsable du stockage des données à conserver, par exemple dans une base de données. Par exemple, un service de vente en ligne pourra être composé de trois étages. Un premier étage responsable

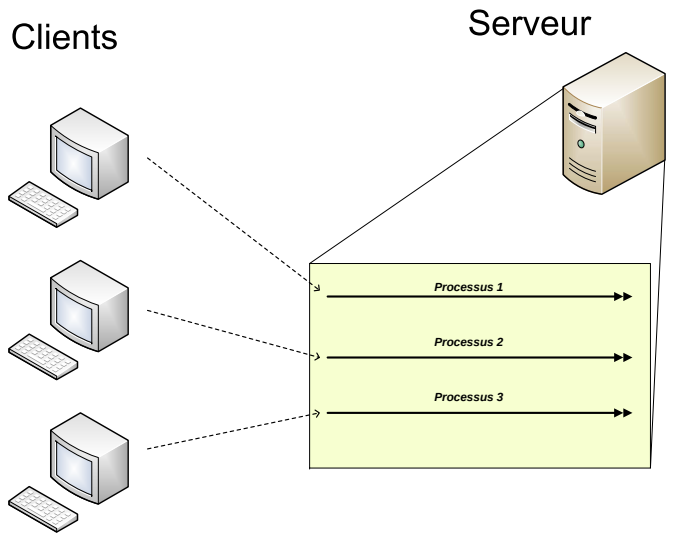

FIGURE 2.2 – Serveur multiprogrammé

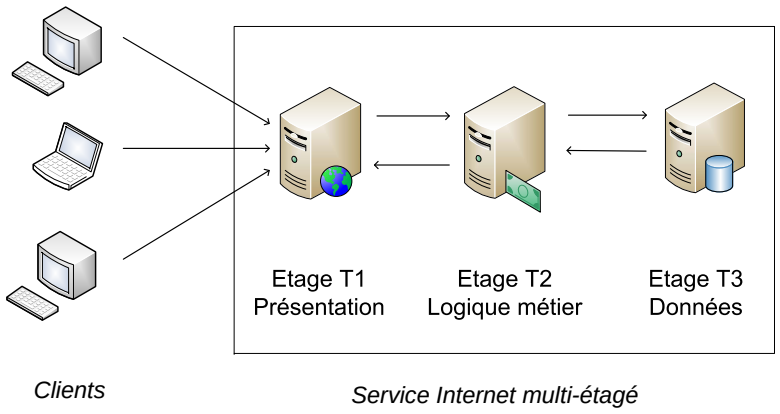

FIGURE 2.3 – Exemple de service multi-étagé

des interactions avec l'utilisateur via une interface graphique. Le deuxième étage sera responsable de la logique métier (calcul des prix, état des stocks, ...). Enfin, le troisième étage aura pour tâche le stockage et la récupération des données persistantes.

Quand un client envoie une requête à un service multi-étagé, la requête accède en premier à l'étage T_1, puis engendre éventuellement des requêtes dans les étages suivants ($T_2, T_3, ...$ jusqu'à T_M). Plus précisément, quand une requête est traitée sur un étage T_i, soit une réponse est renvoyée à l'étage T_{i-1} (ou au client si $i = 1$ et le traitement de la requête est alors terminé), soit une ou plusieurs requêtes sous-jacentes sont envoyées à l'étage T_{i+1} (si $i < M$). La communication entre les étages s'effectue de manière synchrone c'est-à-dire que chaque étage attend la réponse à sa première sous-requête avant d'en envoyer une seconde.

Architecture multi-étagée dupliquée

Pour supporter une charge toujours croissante, les serveurs hébergeant les services Internet sont dupliqués afin de traiter plus de requêtes simultanément. Ceci permet de répartir la charge entre les différents duplicats et de traiter une plus grande quantité de requêtes. La duplication consiste à ajouter un certain nombre de serveurs effectuant le même traitement qu'un serveur de base, afin d'augmenter le nombre de requêtes traitables en parallèle. La figure 2.4 présente un exemple de service Internet multi-étagé avec duplication. Ici, l'étage présentation est composé de 2 serveurs, l'étage métier de 3 serveurs, et l'étage données de 2 serveurs.

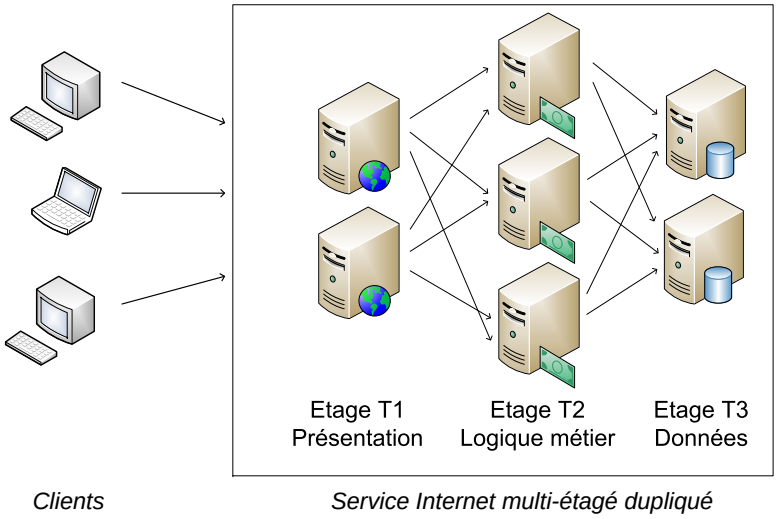

FIGURE 2.4 – Exemple de service Internet multi-étagé dupliqué

La duplication repose sur deux éléments : la répartition de la charge entre les différents duplicats, et le maintien de la cohérence entre les duplicats. Un répartiteur de charge est associé à un service dupliqué, afin de rediriger les requêtes entrantes vers les différents duplicats. Diverses politiques peuvent être suivies par un répartiteur de charge. La plus simple et la plus fréquente est la méthode du tourniquet, qui consiste à envoyer les requêtes entrantes sur chaque machine à tour de rôle, afin d'obtenir une distribution statistiquement

équivalente du nombre de requêtes sur chacun des duplicats [48]. Nous supposons, par la suite, l'utilisation de répartiteurs de charge équitables sur les différents étages des services Internet. Par ailleurs, tous les duplicats d'un serveur doivent fournir le même service. Ainsi, si un duplicat modifie des données partagées, ces mises à jour doivent être répercutées sur les autres duplicats.

2.2 Qualité et coût des services Internet

La qualité de service fait référence au niveau du traitement proposé par un serveur à ses clients. Pour des services Internet, la qualité de service prend en compte, entre autres, des objectifs de performance et de disponibilité [40].

Performance

La performance d'un service Internet fait référence à la rapidité de traitement des requêtes qui sont soumises au service. Nous considérons une des métriques de performance pertinentes dans le cas des services Internet, à savoir la latence et le débit.

La *latence* correspond au temps nécessaire au service Internet pour traiter une requête client. Cette valeur est critique pour le client, car cela correspond à la période pendant laquelle il attend la réponse à sa requête. La latence d'une requête à un service Internet est généralement exprimée en secondes ou en millisecondes et sera calculée à l'entrée du service, depuis la réception de la requête jusqu'à l'émission de la réponse correspondante. La latence est un critère à minimiser pour garantir des services Internet rapides. Nous notons ℓ la latence moyenne d'une requête à un service Internet.

Plusieurs études menées sur le comportement des utilisateurs face à un système informatique ont démontré l'influence de la latence sur le comportement des utilisateurs [41, 44, 45]. Le système est perçu comme répondant instantanément si la latence est inférieure à 0,1 seconde. En-dessous d'une seconde, la réflexion de l'utilisateur n'est toujours pas perturbée par la latence du service, mais l'utilisateur remarque le délai. Au delà de 10 secondes, l'utilisateur abandonne son interaction avec le système. La latence a donc un rôle clef dans la qualité de service de l'application, et l'image qu'ont les utilisateurs de cette dernière.

Le *débit* d'un service Internet est le nombre de requêtes traitées pendant un laps de temps donné. Du point de vue d'un client, le débit d'un serveur n'est pas mesurable, car il n'a pas connaissance de l'ensemble des requêtes en cours de traitement sur le service Internet. Un débit élevé permet cependant d'absorber une charge importante et donc d'améliorer la rentabilité du serveur. Dans la suite, le débit sera mesuré en requêtes traitées par secondes (req/s).

Disponibilité

La disponibilité d'un service Internet représente la chance qu'a une requête client d'être traitée par le service. Par exemple, un service surchargé ne pourra pas traiter toutes les requêtes qui lui sont adressées, et devra donc en rejeter.

La métrique du *taux de rejet* correspond au pourcentage de requêtes rejetées par le système. Ce taux doit être le plus faible possible pour garantir la meilleure disponibilité au service Internet. La disponibilité est un critère de qualité de service important, notamment

pour les sites de commerce en ligne, où toute requête rejetée est un manque à gagner potentiel.

SLA - Contrat de qualité de service

Parties en présence

Un contrat de qualité de service est signé entre un fournisseur d'applications et un client, ou entre un site d'hébergement de service (*cloud computing*) et la société exploitant le service Internet.

Le contrat de qualité de service, ou *SLA* pour *Service Level Agreement*, lie juridiquement les parties en présence, c'est à dire le fournisseur de service Internet et la société cliente. Le SLA peut porter sur un ou plusieurs objectifs de qualité de service [53]. Il peut prévoir également des compensations financières dans les cas où un des points du contrat viendrait à ne plus être respecté. Le respect du contrat de qualité de service est donc une problématique majeure pour un fournisseur de service. Les contraintes de qualité de service peuvent porter sur différentes métriques de qualité de service. Pour chaque métrique, l'objectif peut par exemple porter sur la valeur moyenne maximale à ne pas dépasser. Par exemple une valeur moyenne maximale à ne pas dépasser telle que :

> "La latence moyenne maximale sera inférieure à 2000 ms, avec un taux de rejet inférieur à 5%", ou bien "Moins de 1% des requêtes doivent être rejetées, et leur temps de traitement ne doit pas dépasser 1000 ms en moyenne".

Coût des services Internet

Le coût de fonctionnement d'un service Internet concerne principalement l'énergie consommée par les serveurs hébergeant ce service. En 2007, on estimait la consommation électrique de l'ensemble des serveurs informatique à environ 450 TWh pour la France et 4 000 TWh pour les États-Unis [30]. Cette consommation croît à un rythme constant de 17% par an depuis le début du siècle [31]. Dans la plupart des pays du monde, l'électricité est produite à partir de ressources fossiles (charbon, pétrole). Ce processus a un impact important sur l'environnement. Réduire le nombre de serveurs en fonctionnement pour un service Internet permettrait à la fois de réelles économies financières ainsi que de préserver l'environnement. Nous définissons le *coût* de fonctionnement d'un service Internet comme étant le nombre total de serveurs alloués à ce service.

2.3 Charge des services Internet

Quantité de charge

La quantité de charge d'un service Internet correspond au nombre de requêtes clients accédant simultanément au service Internet. On notera N la quantité de charge d'un service Internet.

Type de charge

Le type de charge est une caractérisation des requêtes émises par les clients au service Internet. Par exemple, le type de charge peut être caractérisé par des requêtes en lecture

seule, des requêtes en écriture, ou une combinaison des deux. Dans un service Internet avec un ensemble connu de requêtes, le type de charge peut être caractérisé par la probabilité d'occurrence de chaque requête [56]. A vrai dire, il n'existe pas de caractérisation formelle du type de charge d'un service Internet. Dans la suite, nous définissons le type de charge C d'un service Internet comme un n-uplet composé des éléments suivants :

- Le temps d'attente, ou temps de réflexion (*think time*) Z. Cette durée correspond au temps d'attente moyen des clients entre la réception de la réponse à une requête et l'émission de la requête suivante. Ce temps dépend du comportement des clients d'un service.
- Les ratios de visite Vi entre étages d'un service Internet multi-étagé. Ces ratios correspondent à l'impact moyen d'une requête client au service Internet sur chacun des étages du service. Par exemple une requête adressée au service (donc à l'étage T_1) entraînera V_i sous-requêtes à l'étage T_i. Par définition, V_1 est donc toujours égal à 1.
- Les temps de service S_i correspondent au temps moyen incompressible de traitement d'une requête sur l'étage T_i du service lorsque celui-ci n'est pas chargé. Ces temps de service dépendent du service Internet lui-même.
- Les délais de communication D_i correspondent au temps incompressible de transmission entre l'étage T_{i-1} et l'étage T_i de l'application. Ces délais dépendent surtout du matériel hébergeant le service Internet.

Les méthodes d'obtention de ces métriques sont détaillées dans le chapitre 7.

Variabilité de la charge

Un service Internet fait généralement face à une charge variable dans le temps. Les variations de charge peuvent concerner le type de charge ou la quantité de charge. Il faut noter qu'une variation de la quantité de charge n'implique pas forcément une variation du type de charge, et réciproquement. Les variations de type de charge concernent la manière dont les clients interagissent avec le service. Par exemple, un service de vente en ligne verra la proportion de client accédant à la partie commande du site augmenter lors de périodes de fêtes ou de promotion. Ces variations de comportement impliquent des modifications dans un ou plusieurs paramètres définissant le type de charge (voir section 2.3). Ainsi, dans le cas où la proportion de clients effectuant des achats sur un site de vente en ligne augmente, l'étage de stockage des données sera davantage sollicité. Ceci entraînera donc une hausse du ratio de visite (V_i) sur l'étage concerné. De même, les temps de service (S_i) peuvent être modifiés par le changement de comportement des clients. Si ces derniers accèdent plus souvent à des pages dynamiques demandant un temps de calcul important, le temps de service moyen sur l'étage en charge de la logique métier augmentera. Enfin, le temps d'attente (Z) sera différent entre des clients parcourant le site sans but précis et des clients étudiant attentivement les caractéristiques d'un produit par exemple.

Par ailleurs, la quantité de charge, correspondant au nombre de clients accédant au service Internet, peut varier fortement en fonction du type de service et de la période de temps considérés. Par exemple, un service de messagerie subira une charge importante le matin, puis plus faible le reste de la journée, car la majorité des utilisateurs commencent leur journée de travail par relever leur courrier électronique. La figure 2.5 présente un relevé de la quantité de charge mesurée au cours d'une journée sur un site d'actualités informatiques. On y constate une charge faible pendant la nuit, puis une augmentation progressive au cours de la matinée.

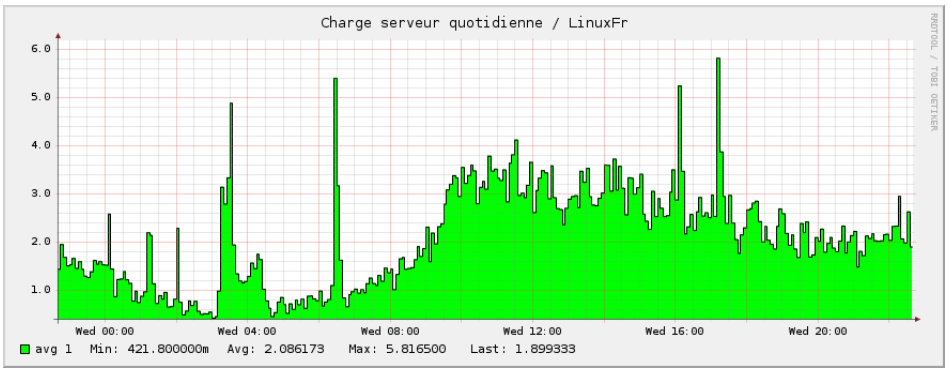

FIGURE 2.5 – Variation de la quantité de charge au cours d'une journée [32]

2.4 Configuration des services Internet

Nous définissons la configuration κ d'un service Internet comme étant un triplet $\kappa < M, AC, LC >$ où M est le nombre d'étages du service Internet, AC est la configuration architecturale et LC la configuration locale du service. Pour un service Internet donné, M est fixe et ne varie pas dans le temps, car il est fixé par les concepteurs de l'application. Les paramètres AC et LC, eux, peuvent évoluer dans le temps.

Configuration locale Pour éviter le phénomène d'écroulement (ou *trashing*) dans un serveur multiprogrammé, lorsqu'un trop grand nombre de clients accèdent au serveur, le nombre de clients concurrents est limité. Cette limite correspond au niveau de multiprogrammation maximal, ou *MPL* pour *MultiProgramming Level*. La valeur de cette limite est un paramètre de configuration retrouvé dans la plupart des serveurs multiprogrammés, tels que le serveur Web Apache [2] et le serveur de bases de données MySQL [52]. Nous définissons la *configuration locale* d'un service Internet multi-étagé comme étant le vecteur $LC < LC_1 \ldots LC_M >$, où LC_i est la valeur du MPL pour chacun des serveurs composant l'étage i de l'application multi-étagée.

Comme illustré dans la section 3.2.1, le problème de la détermination de ces paramètres obéit à un compromis. Avec un MPL faible, le nombre de requêtes traitées en parallèle par le serveur diminue, ce qui réduit les latences de traitement des requêtes mais augmente le taux de rejet. Inversement, avec un MPL élevé, beaucoup de clients peuvent d'accéder au serveur, entraînant des latences élevées pour les requêtes des clients, et éventuellement un phénomène d'écroulement du serveur [27]. En effet, en limitant le nombre de requêtes pouvant accéder de manière concurrente au service, on évite les phénomènes d'écroulement mais on rejette potentiellement des clients. Inversement, admettre une plus grande quantité de clients tentant d'accéder au service permet de diminuer le taux de rejet, mais en augmentant la latence moyenne de traitement des requêtes. Ce comportement est dû au fonctionnement en boucle fermée des services Internet, dont les clients doivent attendre la réponse à une requête avant d'émettre la requête suivante (voir section 5.2.2). La surcharge d'un service se traduit donc principalement par une augmentation de la latence, mais pas forcément du taux de rejet.

Configuration architecturale Nous définissons la *configuration architecturale* d'un service multi-étagé comme étant le vecteur $AC < AC_1 \ldots AC_M >$, où AC_i correspond au nombre de serveurs duplicats alloués à l'étage T_i du service Internet.

Tout comme avec la configuration locale, il faut trouver un compromis pour la valeur de la configuration architecturale. Pour un étage donné, si un nombre insuffisant de serveurs est alloué, un goulot d'étranglement (ou *bottleneck*) peut survenir à cet étage, réduisant la qualité de service. Inversement, si la configuration architecturale est trop élevée, des ressources sont allouées inutilement, augmentant de fait la coût de fonctionnement du serveur (voir section 2.2). Ajouter des ressources sur un étage tend à améliorer les performances de cet étage et ne les dégrade pas.

Un goulot d'étranglement peut survenir sur un seul étage de l'application et pas sur plusieurs étages simultanément [13]. Cependant, en fonction de la charge variable, ce point de congestion peut se déplacer entre les étages du service Internet.

2.5 Synthèse

Ce chapitre a présenté le contexte technique des services Internet. L'architecture de ces services est tout d'abord présentée, puis les critères de qualité de service de coût pour les services Internet. La charge subie par ces services a ensuite été caractérisée selon différents critères de quantité et de type de charge. Les notions de configurations locales et architecturales pour un service Internet ont également été définies. Dans le chapitre suivant, nous présentons les travaux relatifs au contrôle de performance, de disponibilité et de coût pour les services Internet.

Chapitre 3

État de l'art

Nous présentons dans ce chapitre les principaux travaux constituant l'état actuel des recherches dans le domaine du contrôle de services Internet pour la gestion de la performance et de la disponibilité de service. Deux familles de travaux sont présentés : le contrôle de configuration locale et le contrôle de configuration architecturale de services Internet. Enfin, une synthèse globale est proposée en fin de chapitre.

3.1 Présentation générale

La gestion de la configuration est un problème critique pour les fournisseurs de services Internet, car elle impacte directement la qualité du service rendu aux utilisateurs [34, 37, 38]. Traditionnellement, des administrateurs humains sont chargés de maintenir les performances et la disponibilité des services Internet, en utilisant des techniques de configuration ad-hoc basées sur l'expérience de l'administrateur [12, 42]. De nouvelles approches apparaissent pour simplifier et automatiser ces tâches. Ces approches diffèrent selon plusieurs critères :

- le *type de contrôle* effectué sur le système, pouvant être par exemple du contrôle d'admission sur les serveurs ou de l'approvisionnement de serveurs dans un service Internet réparti.
- la *méthode d'analyse et de prise de décision* utilisée, pouvant être basée sur des heuristiques ou sur des méthodes avec garantie d'optimalité.
- la *taille du service Internet* administré, allant d'un serveur seul à un système multi-étagé constitué d'une série de serveurs dupliqués.
- les objectifs du contrôle en termes de *performance, de disponibilité et de coût*
- la prise en compte de la variation de la quantité de charge vs. le type de charge.

Dans la suite, la section 3.1.1 présente les principaux types de contrôle utilisés dans le cadre des services Internet, et la section 3.1.2 présente une classification des méthodes d'analyse et décision.

3.1.1 Types de contrôle

La gestion de la performance, de la disponibilité et du coût des services Internet peut-être effectuée en utilisant différentes techniques pour contrôler le service. Différents types de contrôle peuvent être utilisés, comme la différentiation de service, le réordonnancement

de requêtes, la dégradation de service, le contrôle d'admission ou l'approvisionnement de serveurs [25]. Dans la suite, nous nous intéressons plus particulièrement à ces deux dernières techniques, utilisées dans le cadre de cette thèse.

Contrôle d'admission. Une des techniques fréquemment utilisées pour la configuration locale d'un serveur est le contrôle d'admission. Le contrôle d'admission consiste à limiter la quantité de requêtes à traiter par un serveur lorsque ce dernier fait face à une forte quantité de charge, en rejetant une partie des requêtes entrantes. La figure 3.1 présente un exemple de contrôle d'admission appliqué à un serveur. Ici, la capacité de traitement maximale du serveur pour un type de charge donné est de 100 requêtes en parallèle ; une partie des requêtes est donc rejetée par le contrôle d'admission pour éviter l'écroulement du serveur. Le contrôle d'admission permet de maintenir de bonnes performances pour le service Internet. Cependant, en cas de forte affluence sur un service, le maintien des performances basé sur le contrôle d'admission va entraîner un nombre important de rejets de requêtes, et donc une hausse du niveau d'indisponibilité du service. Par ailleurs, il est à noter que la capacité maximale de traitement d'un serveur (c'est-à-dire le degré de parallélisme maximal) est différent pour différents types de charges [27]. La détermination du contrôle d'admission à appliquer sera donc fonction du type de charge subi par le service.

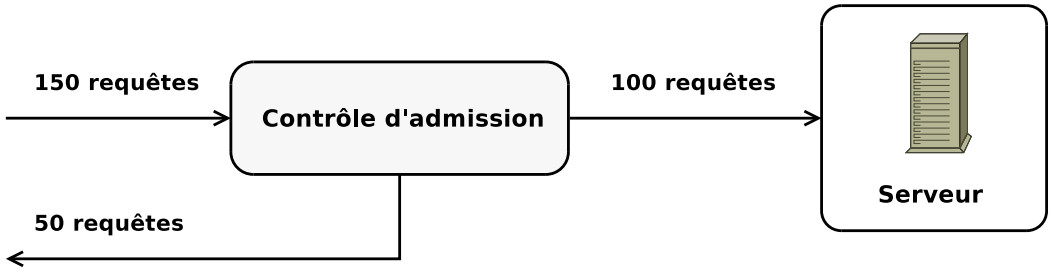

FIGURE 3.1 – Exemple de contrôle d'admission

Approvisionnement de serveurs. Si la charge subie par un service Internet augmente trop, l'augmentation de sa capacité de traitement peut-être réalisée en approvisionnant le service avec des serveurs supplémentaires, c'est-à-dire de nouveaux serveurs dédiés au traitement des requêtes. Ces serveurs peuvent être placés sur les différents étages de l'application. L'ajout de serveurs permet ainsi d'améliorer les performances et la disponibilité d'un service, et ne dégrade jamais ces paramètres. Cependant, l'ajout de serveurs sur un service augmente directement son coût de fonctionnement. En cas de baisse de charge, il faudra donc retirer les serveurs qui ne sont plus nécessaires au fonctionnement du service. Ainsi un bon approvisionnement de serveurs à un service Internet est celui qui permet de garantir les contraintes du SLA tout en minimisant le coût du service. La détermination de l'approvisionnement de serveurs est également fonction du type et de la quantité de charge subis par le service Internet.

Nous parlons ici de deux niveaux de contrôle des services Internet : le contrôle de la configuration locale (c-à-d contrôle d'admission) et le contrôle de la configuration architecturale (c-à-d approvisionnement de serveurs). Les travaux relatifs à ces deux niveaux de contrôle sont respectivement présentés en section 3.2 et en section 3.3.

3.1.2 Méthodes d'analyse et de décision

Nous présentons, dans cette section, les principales méthodes d'analyse et de prise de décision utilisées pour le contrôle de services Internet. Celles-ci sont classées en deux catégories : les méthodes basées sur des heuristiques, les méthodes avec garanties.

Approches basées sur des heuristiques. La première catégorie concernent les travaux se basant sur des heuristiques pour déduire les actions à entreprendre, afin d'atteindre les objectifs de qualité de service voulus [27, 17, 43, 14, 10]. Ces approches ont souvent l'avantage d'être simples à mettre en oeuvre, et donnent de bons résultats dans certaines conditions. Cependant, ces heuristiques offrent un comportement "au mieux" *(best-effort)*, et ne permettent pas de fournir de garanties sur la qualité de service fournie par le service Internet : elles ne donnent pas de garantie sur l'optimalité de la configuration d'un service Internet.

Approches avec garanties. Pour remédier aux limites des approches heuristiques, une deuxième catégorie de travaux cherche à fournir des garanties quant à la configuration calculée et appliquée au service Internet. La plupart de ces approches se basent sur une modélisation du service Internet sous-jacent.

Il existe, par exemple, des modèles linéaires et des modèles non linéaires [24, 57], des modèles basés sur la théorie des files d'attente et des modèles basés sur l'automatique et la théorie de la commande [46, 19, 35], des modèles considérant des serveurs centralisés et d'autres s'adressant à des services Internet répartis [10, 57, 49], des approches considérant une métrique de qualité de service et d'autres combinant plusieurs métriques [15, 39], des approches utilisant un type de contrôle et d'autres en employant plusieurs [43, 19]. Dans la suite, nous présentons les différentes approches de l'état de l'art.

3.2 Contrôle de configuration locale de services Internet

Dans cette section, nous présentons les approches existantes effectuant un contrôle au niveau local des services Internet, c'est-à-dire utilisant le contrôle d'admission sur le serveur composant le service. Parvenir à déterminer la configuration locale qui va améliorer la qualité de service d'un serveur est une problématique qui a donné lieu à de nombreuses études. Ces approches peuvent être basées sur des heuristiques (cf. section 3.2.1), ou être des approches avec garantie de qualité de service (cf. section 3.2.2).

3.2.1 Approches à base d'heuristiques

Les approches à base d'heuristiques sont souvent les premières utilisées, car elles se basent généralement sur l'expérience pratique des administrateurs des services.

Heiss et Wagner [27]

Heiss et Wagner étudient dans [27] le comportement d'un serveur multi-programmé centralisé, comme un serveur Web ou un serveur de base de données. La quantité et le type de charge subie par le serveur évolue au cours du temps. L'objectif est de déterminer la configuration locale, c'est-à-dire le contrôle d'admission, donnant les meilleures performances

pour le serveur. Le contrôle d'admission est ici réalisé en ajustant le degré maximal de multiprogrammation du serveur, ou *MPL*. Ce paramètre limite le nombre de clients autorisés à s'exécuter en parallèle dans le serveur. Les auteurs s'intéressent à la caractérisation d'une fonction de performance exprimée en fonction du MPL du serveur. La fonction de performance retenue est une parabole, ayant un optimum global. Cet optimum correspond à la configuration offrant le meilleur rendement pour le serveur avec le type de charge considéré.

L'approche proposée par Heiss et Wagner sert de base à plusieurs approches [43, 22, 54]. Nous avons donc validé expérimentalement cette approche en l'appliquant à l'administration d'un serveur de bases de données MySQL, dans le cadre du service de ventes aux enchères Rubis [1]. Les figures 3.2 et 3.3 présentent la valeur de la fonction de performance définie dans [27] en fonction du nombre de requêtes client admises en parallèle dans le serveur pour deux types de charge différents. La figure 3.2 présente la fonction de performance pour une charge en lecture seule, alors que la figure 3.3 présente la fonction de performance pour une charge comportant 20% de mises à jour dans la base de données. Comme on le voit, la valeur optimale (c'est-à-dire maximale) de la configuration locale est différente pour différents types de charge. Elle est de 225 pour le premier type de charge et de 122 pour le second type de charge.

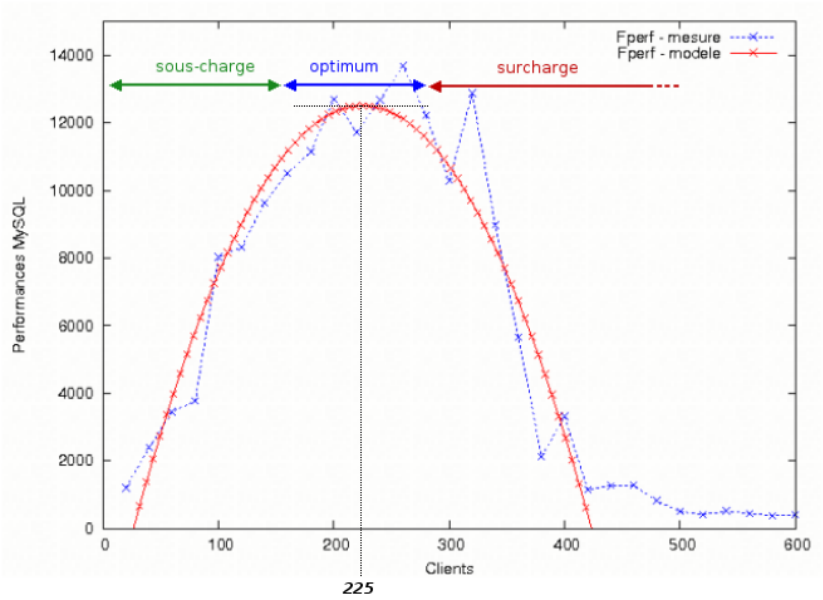

FIGURE 3.2 – Approximation parabolique des performances - type de charge en lecture seule

Par ailleurs, connaître les paramètres de la parabole supposerait d'avoir mesuré suffisamment de valeurs de la fonction de performance. Comme on ne dispose, a priori, pas de la valeur de la parabole, Heiss et Wagner proposent une heuristique pour faire évoluer pas à pas la configuration locale dans le sens améliorant la performance, c'est à dire dans le sens croissant de la pente de la fonction de performance. Cet algorithme connu sous le nom de *hill-climbing* tente de s'approcher de la configuration optimale après un nombre variable de pas, mais sans garantie d'optimalité.

Enfin, différents types de charge impliquent différents paramètres de la fonction de per-

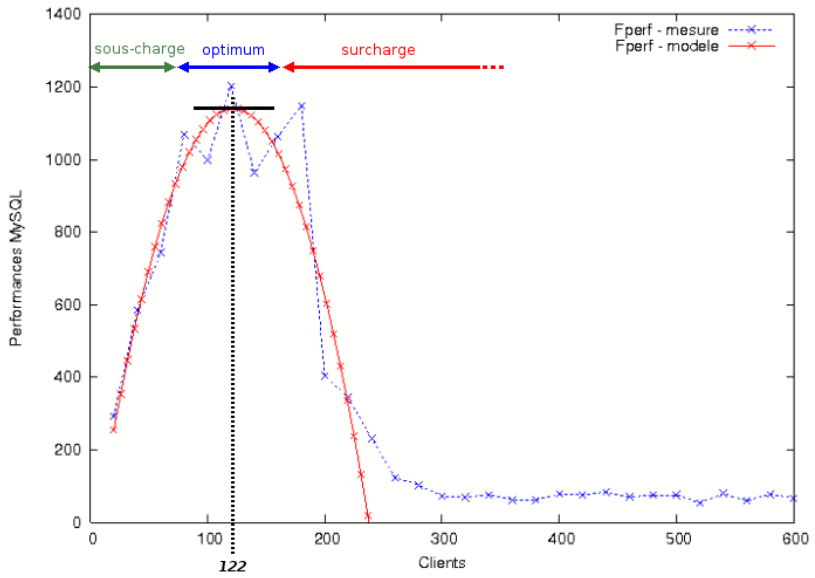

FIGURE 3.3 – Approximation parabolique des performances - type de charge en écriture

formance, et donc différents optimums pour la parabole. Cela suppose de connaître, a priori, les valeurs de toutes les paraboles, ce qui est peu réaliste. De plus, la méthode proposée ne s'applique qu'à des métriques de qualité de service ayant une forme de parabole, ce qui n'est pas le cas de toutes les métriques (comme la latence ou le taux de rejet).

Milan-Franco et al. [43]

Milan-Franco et al. proposent dans [43] une approche pour améliorer le débit d'une application faisant face à une quantité et un type de charge variables. L'architecture de l'application considérée est constituée d'un seul étage dupliqué. Les auteurs mettent en application les travaux de Heiss et Wagner pour déterminer la configuration optimale d'un serveur de base de données dupliqué, en fonction du type de charge à traiter [43]. Cet article présente une approche d'optimisation à 2 niveaux. En effet il combine la méthode d'approximation parabolique vue dans [27] pour optimiser le débit d'un serveur au niveau local, avec une reconfiguration au niveau de l'étage lorsque la première ne suffit plus. Au niveau de l'architecture de l'étage, le nombre de duplicats n'est pas modifié, mais la correspondance entre objet accédé par transaction et duplicat gérant la transaction évolue au cours du temps pour équilibrer la charge entre les noeuds. Les gains sont donc importants avec une charge inégalement répartie sur les noeuds.

Cette approche souffre des mêmes limitations que l'approche de Heiss et Wagner concernant les variations dans le type de la charge. Il faudrait également que le nombre de duplicats puissent évoluer lorsque la quantité de charge augmente, car le nombre de duplicats initialement présents risque d'être insuffisant pour garantir les performances voulues. Le paramétrage de l'auto-optimisation, c'est-à-dire la détermination des seuils et des intervalles de mesure reste enfin un problème en soit.

Chen et al. [17]

Chen et al. développent dans [17] une heuristique pour améliorer le temps de réponse et le débit d'un serveur Web. en présence d'une quantité de charge variable. Les auteurs utilisent pour cela un algorithme de contrôle d'admission des requêtes dans le serveur. Cette approche est basée sur une comparaison entre les estimations des temps de service des requêtes, et du temps de traitement disponible sur le serveur. Si le temps de traitement disponible sur le serveur est suffisant pour la durée estimée d'exécution de la requête, alors cette dernière est admise dans le serveur. L'approche proposée permet de réduire le délai moyen d'exécution des requêtes, cependant le type de système pris en compte est limité à un serveur unique. De plus, les variations de type de charge ne sont pas prises en compte.

Welsh et al. [59]

Welsh et al. proposent dans [59] une solution pour améliorer la latence et le débit d'un étage d'une application multi-étagée, faisant face à une quantité de charge variable. Les auteurs proposent une méthode de construction d'applications par assemblage d'étages, à travers le prototype SEDA. Chaque étage du système est contrôlé de manière indépendante pour simplifier sa configuration. Sur chaque étage, un contrôle d'admission est appliqué pour éviter la surcharge du ou des serveurs composant l'étage. Ce contrôle d'admission s'adapte pour respecter une contrainte de latence maximum selon une heuristique classique : le taux de multiprogrammation est augmenté en ajoutant un palier fixe lorsque les performances respectent la contrainte de latence, et diminué de manière multiplicative lorsque les objectifs de qualité de service ne sont plus respectés. Cela permet de trouver rapidement une configuration respectant la qualité de service, même si des ressources peuvent être gaspillées. L'avantage de ce type de contrôle est sa facilité de mise en oeuvre. Cependant, le fait de contrôler chaque étage de manière indépendante sans coordination globale entre les étages peut mener à un gaspillage de ressources, notamment lorsque une requête est rejetée par le dernier étage de l'application après avoir consommé du temps de calcul sur les étages précédents.

Diao et al. [19]

Diao et al. proposent dans [19] une méthode pour améliorer la consommation CPU et mémoire d'un serveur Web centralisé. Les auteurs proposent une méthode utilisant plusieurs paramètres de configuration locale du système, pour améliorer l'utilisation de plusieurs ressources matérielles. Les paramètres de configuration utilisés sont le degré maximal de multiprogrammation (MPL), et la durée au bout de laquelle les sessions sont considérées comme abandonnées. En configurant conjointement ces paramètres, les auteurs améliorent la consommation mémoire et l'utilisation du processeur. L'idée est que si l'on veut améliorer plusieurs paramètres simultanément, il faut une approche globale et non une juxtaposition d'améliorations indépendantes. En effet les différentes approches d'amélioration peuvent ne pas être totalement orthogonales. Par exemple améliorer un premier critère de qualité de service peut nuire à un second. Les expériences menées confirment que la juxtaposition d'améliorations au même niveau est moins efficace qu'une amélioration globale de plus haut niveau.

Cependant, les critères de performance de haut niveau (latence, débit, taux de rejet) ne

sont pas optimisés directement, mais impactés par l'optimisation de l'utilisation des ressources matérielles. De même, la quantité et le type de charge ne sont pas pris en compte directement. L'approche ne fournit donc pas de garanties sur la qualité de service obtenue.

Elnikety et al. [22]

Elnikety et al. présentent dans [22] deux principes pour améliorer la latence d'un serveur, particulièrement en cas de pic dans la quantité de charge. Le type de charge est hétérogène car on considère plusieurs classes de requêtes avec des temps de traitement très différents, et avec des quantités de charges pouvant varier. La solution proposée consiste en l'utilisation de deux méthodes de contrôle au niveau local. Premièrement, un contrôle d'admission cherche à calculer un sous-ensemble de requêtes à exécuter parmi les requêtes en entrée du système. Cette méthode permet de limiter l'écroulement dû à un nombre trop important de requêtes dans le système. Les auteurs se basent pour cela sur l'approche de Heiss et Wagner [27]. Deuxièmement, une méthode de réordonnancement de requêtes selon la technique du plus court d'abord *(SJF : shortest job first)* est utilisée pour diminuer le temps de réponse moyen du système. Ceci est réalisé à l'aide d'un proxy, qui maintient pour chaque classe de requête le temps moyen d'exécution, ce qui permet de prédire quelles seront les requêtes les plus rapides à se terminer. Le nombre de requêtes admissibles dans le système est donc continuellement calculé et mis à jour en fonction du temps d'exécution de ces requêtes. Dans ce cas, le contrôle d'admission n'est pas appliqué à plusieurs étages. En effet, le proxy Gatekeeper est seulement positionné devant le dernier étage de l'application.

L'estimation du temps d'exécution des requêtes de chaque classe se faisant à faible charge, il n'est pas garanti que ces temps restent valides lorsque la charge varie de manière brutale, ni que la quantité de charge du serveur soit faible suffisamment longtemps pour mesurer ces temps d'exécution. La méthode de contrôle d'admission qui se base sur ces temps d'exécution risque donc de se révéler approximative voire fausse si les délais fluctuent dans des proportions trop importantes.

Parekh et al. [46]

Parekh et al. présentent dans [46] la conception d'une boucle de rétroaction permettant de contrôler le nombre de requêtes en attente de traitement dans une file. Le but est de maximiser le gain obtenu lors d'une étape de reconfiguration. L'approche est appliquée à un serveur simple (Lotus Note). Cette méthode permet de minimiser les oscillations ainsi que le temps de stabilisation du système contrôlé. Les auteurs présentent une méthode de modélisation mathématique d'un serveur comme une fonction de transfert. Cette approche vise à modéliser un serveur Internet pour déterminer les paramètres de la boucle de commande d'auto-optimisation associée. Même s'il y a un modèle de l'application, le type de contrôle est donc basé sur une heuristique par palier de reconfiguration. Le type de charge est homogène, et il n'y a pas de variation de quantité de charge, le but de l'étude étant entre autres de comparer les temps de stabilisation du MPL à la valeur souhaitée. L'approche formelle donne de bons résultats, mais est limitée à des systèmes simples. En effet le type de systèmes pris en compte sont des serveurs seuls avec une charge constante. Enfin, les objectifs de qualité de service ne sont pas pris directement en compte, l'objectif étant ici d'atteindre un nombre voulu de clients dans la file d'attente du serveur, mais sans en déduire l'impact sur la latence, le débit, ou le taux de rejet de l'application.

3.2.2 Approches avec garanties

Nous avons vu que les approches basées sur des heuristiques ne permettent pas de fournir de garanties concernant la performance et la disponibilité des service Internet contrôlés. Or la majorité des services Internet sont désormais soumis à des contrats de qualité de service, spécifiant des critères de performance et de disponibilité à atteindre. Des approches permettant de garantir ces critères sont donc devenues nécessaires. La plupart se basent sur une modélisation de l'application pour calculer ses performances.

Diao et al. [20]

Diao et al. proposent dans [20] une solution pour optimiser la latence des systèmes multi-étagés faisant face à des quantités de charge variable. Dans l'approche proposée, chaque étage embarque un contrôleur autonome qui communique avec l'étage précédent et l'étage suivant. Les étages ont une connaissance des flux de requête entrants et sortant, et en déduisent les informations de qualité de service à comparer avec les objectifs de qualité de service. En cas de non respect du contrat de qualité de service sur un étage, le contrôleur détermine si le problème peut-être résolu en reconfigurant son étage, ou sinon transmet l'information à l'étage suivant. Afin de calculer la configuration à appliquer, les contrôleurs embarquent un modèle du service Internet. Ce modèle est basé sur un réseau fermé de files d'attente, et permet de calculer la latence de chaque étage. Le contrôle de qualité de service proposé est donc décentralisé, afin de permettre de diminuer la quantité de messages à envoyer entre le contrôleur et le service Internet, notamment pour les grands systèmes. De plus, la synchronisation des reconfigurations peut poser problème si les délais de communication sont trop importants, ou différents entre les étages.

Zhang et al. [60]

Zhang et al. cherchent dans [60] à optimiser le débit d'une application multi-étagée, en présence d'une quantité et d'un type de charge variables. Les auteurs relèvent la difficulté du paramétrage des modèles à file d'attente, souvent utilisés pour modéliser les systèmes multi-étagés. Ils proposent d'estimer la demande en temps de calcul d'une requête en utilisant une régression mathématique. Le modèle obtenu est ainsi plus simple, et ne nécessite pas de paramétrage complexe. Cette méthode est ensuite utilisée pour prédire la capacité de traitement d'une application multi-étagée en terme de clients accédant simultanément au service, ce qui permet d'optimiser le débit de cette application. Cependant, la définition du modèle nécessite de disposer de suffisamment de données concernant la charge des processeurs sur l'application, afin d'appliquer la méthode par régression. De plus, le modèle proposé ne prend pas en compte la duplication sur les étages du service Internet.

Menascé et al. [39]

Menascé et al. cherchent dans [39] à optimiser conjointement la latence, le débit et le taux de rejet d'un système multi-étagé pour que ce dernier respecte les contraintes de qualité de service données dans un SLA. Les auteurs utilisent pour cela une approche basée sur le contrôle d'admission pour contrôler chacun des étages d'un système multi-étagé [39]. La méthode de calcul de la configuration à appliquer se base à la fois sur un modèle de l'application contrôlée ainsi que sur un algorithme de *hill-climbing* (gravissement de colline). Ce

dernier utilise le modèle pour calculer les performances obtenues avec les différentes configurations parcourues jusqu'à trouver celle respectant le contrat de qualité de service. La validation expérimentale de l'approche a été réalisée en utilisant un banc d'essai représentant un site de vente en ligne multi-étagé. L'approche proposée est utilisable avec différents paramètres de configuration locale, comme le réordonnancement de requêtes, cependant l'allocation dynamique de ressources n'est pas prise en compte. De plus, on a vu que la recherche par hill-climbing utilisée ici peut rester bloquée sur un optimum local, et ne pas trouver la configuration optimale permettant de respecter le contrat de qualité de service.

Malrait et al. [36]

Dans [36], Malrait et al. cherchent à fournir des garanties de performance (latence) et de disponibilité sur un serveur simple, en présence de variations dans la quantité et le type de charge. Les auteurs utilisent pour cela du contrôle d'admission. La méthode de calcul de la valeur de la configuration locale est ici basée sur une approche par modélisation à base de flux. Ce type de modèle cherche à caractériser l'impact des variations dans les paramètres d'entrée du système (charge et configuration) sur des variables caractérisant l'état du système, appelées variables internes. Les sorties du modèle sont ensuite calculées à partir de ces variables internes, et pas directement à partir des entrées comme dans le cas d'un modèle à files d'attente par exemple. L'approche proposée ici permet de capturer l'impact des variations concernant à la fois le type et la quantité de charge subie par le serveur contrôlé, et d'optimiser la configuration locale du serveur pour optimiser l'un de ces paramètres en fixant une limite au second.

3.3 Contrôle de configuration architecturale de services Internet

Récemment plusieurs approches sont apparues pour gérer la planification de capacité dans le cadre des services Internet multi-étagés. Ces approches ont pour but de calculer et d'appliquer la configuration donnant les meilleurs résultats, en ajoutant ou en enlevant des serveurs au service en fonction de son contexte d'exécution. Certaines approches sont basées sur des heuristiques [16, 10], tandis que d'autres utilisent une modélisation du système administré, permettant d'atteindre la configuration optimale plus rapidement [58, 57, 49].

3.3.1 Approches à base d'heuristiques

Chen et al. [16]

Chen et al. présentent dans [16] une solution pour améliorer la latence d'une application multi-étagée faisant face à une quantité et un type de charge variables. Les auteurs proposent pour cela une heuristique d'allocation dynamique de serveurs sur le dernier étage de l'application multi-étagée. L'objectif de l'approche est d'allouer suffisamment de serveurs pour que la latence globale de l'application respecte un objectif de qualité de service donné. Les auteurs font donc l'hypothèse que la capacité de traitement du dernier étage de l'application constitue toujours le facteur limitant pour les performances. Deux paliers, *HighSLAThreshold* et *LowSLAThreshold*, sont définis afin de décider des reconfigurations. Lorsque la latence dépasse le premier, un serveur est ajouté sur l'étage, et lorsque la latence passe en-dessous

du second, un serveur est retiré. Ceci permet de conférer au système une propriété d'hystérésis évitant les oscillations dans la configuration. La simplicité du contrôleur lui permet d'être indépendant du type de charge reçue par le service Internet, et donne de bons résultats avec des variations graduelles de charge. Cependant, en présence de fortes variations dans la demande de service, la reconfiguration serveur par serveur risque d'être trop lente à converger vers la configuration optimale. De plus, la seule allocation de ressources sur le dernier étage de l'application ne garantit pas d'atteindre l'objectif de latence donné, car si la charge augmente de manière importante, les étages précédents peuvent constituer un goulot d'étranglement. Enfin, la détermination de la valeur des paliers reste un problème en soit.

Bouchenak et al. [10]

Bouchenak et al. présentent dans [10] une heuristique pour contrôler l'utilisation CPU des serveurs d'un service multi-étagé dupliqué, faisant face à des variations de quantité et de type de charge. L'objectif est ici de maintenir l'utilisation des processeurs dans un intervalle défini, afin d'utiliser au mieux chaque serveur. L'approche présentée permet d'ajuster automatiquement le nombre de duplicats d'un serveur en fonction de la charge. A chaque étage, une boucle de commande surveille l'utilisation du processeur sur les duplicats, et si ce taux d'utilisation dépasse un certain seuil, un serveur est ajouté. Inversement si le taux d'utilisation devient trop faible, on enlève un serveur pour libérer des ressources. L'approche proposée est donc similaire à celle de [16], mais se base sur l'utilisation du processeur et non sur la latence de l'application. Cette approche souffre donc des mêmes limitations que [16]. La reconfiguration par pas successifs induit des délais de réaction assez importants, ce qui peut être gênant si la charge évolue rapidement. De plus, seule une optimisation au niveau du degré de duplication sur les étages est effectuée. La configuration locale des serveurs n'est ainsi pas modifiée, ce qui peut éventuellement entraîner un gaspillage de ressources, ou inversement une surcharge sur les serveurs de l'application. Enfin, la méthode se basant sur l'utilisation d'une ressource matérielle, elle ne fournit pas de garanties concernant la qualité de service de l'application.

3.3.2 Approches avec garanties

La limitation commune aux approches basées sur les heuristiques est leur faible vitesse de convergence, et la possibilité de ne pas trouver la configuration optimale. Or dans un contexte de charge dynamique, trouver rapidement la configuration optimale pour le service Internet contrôlé est une nécessité. D'autres approches se basent donc sur une modélisation des services Internet pour caractériser précisément leur comportement.

Chase et al. [15]

Chase et al. présentent dans [15] l'architecture Muse, dont le but est de minimiser la consommation d'énergie, tout en respectant des objectifs de qualité de service concernant la latence, le débit et le taux de rejet d'une application sur un étage dupliqué d'une application multi-étagée. L'approche proposée se place dans un contexte de charge variable en termes de quantité et de charge. Pour atteindre leurs objectifs, les auteurs proposent d'adapter le nombre de serveurs utilisés par un hébergeur de services Internet. Ils utilisent pour cela un

modèle économique pour l'allocation de ressources. A chaque client est associée une fonction d'utilité permettant d'estimer le gain obtenu par le traitement d'une de ses requêtes. Les gains obtenus par le traitement des requêtes des clients sont comparés avec le coût associé aux ressources à allouer pour les traiter. Ainsi dans certains cas le système rejette des requêtes, plutôt que d'allouer un serveur qui sera très peu utilisé et diminuera la profitabilité de l'ensemble du service. La prise en compte du contrat de qualité de service (SLA) se fait en incluant des pénalités en cas de non respect du contrat. L'approche est validée sur un serveur Web Apache, et permet de réduire la consommation d'énergie tout en préservant le débit et la latence de l'application. Cependant, la difficulté d'utilisation réside dans le paramétrage du modèle économique utilisé, notamment dans les pénalités relatives au non-respect du SLA. La charge de calcul imposée par l'utilisation du modèle économique peut également être problématique, particulièrement dans le cadre d'un système à plus large échelle.

Urgaonkar et. al. [57]

Dans [57], Urgaonkar et al. proposent une solution pour améliorer la latence d'une application multi-étagée dupliquée en présence de variations dans la quantité de charge, afin de respecter une contrainte de latence maximum. Pour cela les auteurs effectuent de l'approvisionnement dynamique de serveurs, afin d'ajouter ou de retirer des serveurs en fonction de la quantité de charge courante. L'approvisionnement utilise une modélisation de l'ensemble des étages du service Internet multi-étagé. Cette modélisation est basée sur l'utilisation de l'algorithme MVA *(Mean Value Analysis)*, qui associe l'application modélisée à un réseau de files d'attente. Le point difficile est de calculer les données en entrée de l'algorithme. Par contre cette approche permet de gérer de manière globale les particularités des applications modélisées. Les auteurs proposent une application de leur modèle dans le cadre de l'allocation dynamique de serveurs pour les services Internet, afin d'ajouter ou de retirer des serveurs en fonction de la quantité de charge courante pour respecter un objectif de latence maximum. La méthode de planification de capacité proposée peut fournir des garanties concernant le respect de l'objectif de latence maximum, mais l'optimalité de la configuration calculée n'est pas prouvée.

L'approche proposée prend en compte des variations dans la quantité de charge, mais le type de charge est fixe et doit être détecté hors-ligne. De plus, le paramétrage du modèle proposé peut s'avérer ardu, ce qui déplace la complexité du contrôle du service Internet vers le modèle.

Sivasubramanian et al. [49]

Sivasubramanian et al. proposent dans [49] une méthode pour optimiser la latence d'un service multi-étagé construit par composition de service. Dans l'architecture proposée, chaque étage peut faire appel à un nombre variable de sous-étages. La composition de service est un cadre plus général que les architectures multi-étagées. En effet dans le cas des architectures à plusieurs étages, un étage peut interroger au plus l'étage suivant, alors que dans le cadre de la composition de services un étage peut en interroger plusieurs autres, éventuellement distants. Les variations de la quantité et du type de charge sont prises en compte. L'objectif de cette étude est d'allouer des ressources de manière efficace, dans le cadre de services Internet sous la contrainte d'un contrat de qualité de service (SLA). Le critère de performance considéré est uniquement le temps de réponse. Les auteurs se basent sur une modélisation

du service multi-étagé contrôlé afin de rajouter des ressources (caches, duplicats) à des emplacements non triviaux dans l'architecture. Ceci permet d'obtenir des temps de réponse optimaux tout en minimisant l'utilisation de ressources. Une méthode pour évaluer le taux de succès des caches est présentée, utilisant un système de caches virtuels, fonctionnant sur le principe de "pêche à la ligne". Le but de cette approche est d'effectuer de l'allocation de ressources de manière efficace en ayant un modèle du comportement de l'application. Le critère de performance est une latence maximale à ne pas dépasser, qui est donnée par un SLA. La charge, hétérogène, peut varier au cours du temps. Dans cette approche, on fait la supposition que le MPL de chaque serveur est fixe. On pourrait combiner cette approche avec les travaux présentés dans [27] pour obtenir un système optimisé sur les deux niveaux d'architecture.

Ghanbari et al. [24]

Ghanbari et al. présentent dans [24] une approche pour minimiser la latence des requêtes et la température des serveurs composant un service Internet, réduit à un étage dupliqué. Les variations dans la quantité et le type de charge sont prises en compte. Les auteurs proposent un modèle construit par apprentissage dynamique de la corrélation entre les différents paramètres de configuration considérés et les métriques de latence et de température. Dans le cadre d'une application multi-étagée, les auteurs mettent en oeuvre le modèle proposé pour l'allocation dynamique de serveurs sur le dernier étage de l'application, en fonction de la quantité de charge. Les auteurs mettent ainsi en évidence la corrélation entre configuration architecturale et latence du service pour une charge donnée.

Afin de pouvoir capturer correctement le comportement du système, un modèle par apprentissage nécessite un certain nombre de données mesurées sur le système réel en fonctionnement. Ces mesures sont des couples composés des paramètres d'entrée du modèle et des sorties correspondantes, et constituent la base d'entraînement du modèle. Cependant, dans le cadre des services Internet, si la nature de la charge varie, mais que la quantité de charge ne varie pas sur une plage suffisamment grande, le modèle ne disposera pas d'un jeu de données d'entraînement lui permettant de capturer correctement le comportement de l'application, pour l'ensemble des quantité et types de charge possibles. De plus, la prise en compte des paramètres de configuration d'un seul étage peut être problématique. En effet, la configuration des autres étages de l'application peut avoir un impact sur les performances de l'étage considéré. Une application de l'approche proposée ici à l'ensemble des paramètres de configuration d'un service Internet multi-étagé dupliqué peut être envisagée, mais le calibrage du modèle risque de nécessiter un jeu de données d'entraînement trop important pour être réellement utilisable.

Dejun et al. [28]

Dejun et al. proposent dans [28] une approche pour l'allocation de ressources dans le cadre de services Internet composés de plusieurs applications distinctes. L'objectif est ici de minimiser le coût du service, tout en respectant un objectif de qualité de service (latence). Pour cela, les auteurs proposent un modèle de service Internet permettant de prédire la latence d'une composition d'applications en fonction du nombre de ressources allouées à chacunes d'elles. Ce modèle est recalibré automatiquement lorsque son erreur dépasse un

certain seuil de tolérance. Les variations de type et de quantité de charges sont ainsi prises en compte.

Déterminer les seuils de recalibrage du modèle reste cependant une problématique ouverte. De plus, la configuration locale des serveurs composant le service Internet n'est pas prise en compte. Le fait de considérer uniquement l'adaptation de la configuration architecturale peut donc mener à un gaspillage de ressources.

3.4 Synthèse

Ce chapitre présente les principaux travaux existants dans les domaines de la configuration et la gestion de ressources pour les services Internet. Après avoir présenté les différents critères de comparaison retenus, nous détaillons les travaux existants en les classant en fonction du types de contrôle, et des méthodes d'analyse et de prise de décision appliqués.

Nous synthétisons dans le tableau 3.1 les caractéristiques des approches présentées. Pour chaque article, nous indiquons le type de contrôle effectué sur le système, si une heuristique est utilisée ou si des garanties sont fournies par la méthode d'analyse et de décision. La nature adaptative ou non du contrôle est indiquée. Nous précisons également la taille du service Internet contrôlé, les métriques de performance, de disponibilité ou de coût considérées comme objectif, et enfin si les variations dans la nature ou le type de la charge sont prises en compte.

Les approches heuristiques sont efficaces pour un contrôle simple, mais elles se trouvent confrontées au manque de garantie quand à la configuration calculée. Certaines heuristiques, comme l'approche par hill-climbing utilisée dans plusieurs approches [27, 43], peuvent également être lentes à converger, ou osciller autour de la valeur optimale. L'utilisation de modèles permet de résoudre cette difficulté, mais pose le problème du paramétrage parfois complexe du modèle utilisé.Les approches locales permettent donc d'optimiser la configuration du ou des serveurs composant l'application contrôlée. Cependant, la seule configuration locale ne permet pas d'optimiser à la fois la latence et le taux de rejet en présence d'une charge variable. En effet, la quantité de serveurs alloués à un service peut être insuffisante pour traiter les requêtes des clients en respectant les contraintes de qualité de service, et ceci quelle que soit la configuration locale de ces serveurs. Dans ce cas, la configuration locale ne suffit plus, et doit être complétée par de l'allocation dynamique de ressources, ou configuration architecturale.

La planification de capacité détermine la quantité de ressources à allouer à une application en fonction de sa charge et des contraintes de qualité de service à respecter. Des heuristiques basées sur des seuils peuvent être utilisées, soit en fonction de la charge du système [10], soit en fonction du respect ou non des contraintes de qualité de service [16]. Cependant la valeur de ces seuils impacte fortement l'efficacité du contrôle, et leur détermination n'obéit à aucune règle précise. De plus, la vitesse de convergence de ces approches peut être trop lente en cas de variations rapides et importantes de la charge. Des approches basées sur des modèles économiques [15] ou à files d'attente [57] permettent de calculer directement la configuration architecturale à appliquer. Une méthode de contrôle de la configuration architecturale basée sur un modèle construit par apprentissage est proposé dans [24]. Cependant, le fait de ne pas contrôler la configuration locale simultanément peut mener à un gaspillage de ressources. De plus, le paramétrage de ces modèles reste délicat [60].

Défis scientifiques ouverts

En résumé, les défis scientifiques ouverts à ce jour sont les suivants.

- Calculer la configuration optimale qui, d'une part, garantit le contrat de qualité de service (SLA) et, d'autre part, minimise le coût de fonctionnement du service Internet. La prise en compte simultanée de plusieurs critères de performance (latence, débit), de disponibilité et de coût est en effet rarement effectuée dans les approches existantes.
- Considérer les contrats de qualité de service qui stipulent à la fois des contraintes de performance et des contraintes de disponibilité de service. Des garanties doivent être fournies quand aux différents critères de qualité de service stipulés dans le SLA. La majorité des solutions existantes fournissant des garanties le font sur un seul critère de qualité de service.
- Fournir un contrôle adaptatif en ligne des services Internet multi-étagés et dupliqués, qui détecte et prenne en compte automatiquement la variation de la quantité et du type de charge. Les solutions existantes sont souvent limitées à des systèmes simples (serveur centralisé) ou alors ne proposent pas d'adaptation automatique du contrôle.

Dans cette étude, nous proposons une solution permettant de répondre à l'ensemble de ces problématiques. Une solution de planification de capacité pour calculer la configuration optimale d'un service Internet est proposée dans le chapitre 6. Cette solution est basée sur un modèle des services Internet multi-étagés et dupliqués, présenté dans le chapitre 5. Un contrôleur automatique capturant les variations dynamiques de la charge est proposé dans le chapitre 7.

Le chapitre suivant présente les problématiques auxquelles doivent faire face un grand nombre de fournisseurs de services Internet. Nous présentons le principe des solutions apportées à ces problématiques dans le reste de cette étude.

	Type de contrôle		Méthode d'analyse et décision		Contrôle adaptatif		Taille du système			Objectifs			Type de variation	
	contrôle d'admission	approvisionnement dynamique	heuristique	garanties	non	oui	centralisé	dupliqué	multi-étagé dupliqué	performance	disponibilité	coût	quantité de charge	type de charge
Heiss et Wagner [27]	X		X		X		X			X			X	
Milan-Franco et al. [43]	X		X		X			X		X			X	
Chen et al. [17]	X		X		X		X			X			X	
Welsh et al. [59]	X		X			X		X		X			X	
Diao et al. [19]	X		X		X		X							
Elnikety et al. [22]	X		X		X		X			X			X	
Parekh et al. [46]	X		X			X	X							
Diao et al. [20]	X			X	X				X	X				
Zhang et al. [60]	X			X	X				X	X			X	
Menascé et al. [39]	X			X	X				X	X	X		X	
Malrait et al. [36]	X			X	X		X			X	X		X	X
Chen et al. [16]		X	X		X			X		X			X	
Bouchenak et al. [10]		X	X		X				X				X	
Chase et al. [15]		X		X		X		X		X	X	X	X	
Urgaonkar et al. [57]		X		X	X				X	X			X	
Sivasubramanian et al. [49]		X		X	X			X		X			X	
Ghanbari et al. [24]		X		X		X		X		X			X	
Dejun et al. [28]		X		X		X			X	X		X	X	X

TABLE 3.1 – Résumé des approches

Chapitre 4

Motivations et contributions scientifiques

Dans ce chapitre, nous synthétisons les problématiques posées par la gestion de la performance, de la disponibilité et du coût de services Internet soumis à des quantités et des types de charge variables. Pour répondre à ces problématiques, nous présentons ensuite les objectifs de cette thèse et les contributions scientifiques, concernant une méthode de planification de capacité calculant la configuration optimale d'un service Internet, un modèle analytique de prédiction de performance et un contrôle adaptatif de service Internet prenant en compte les variations de quantité et de type de charge.

4.1 Illustration du problème

La configuration d'un service Internet, mais également la quantité et le type de sa charge ont un impact sur la performance, la disponibilité et le coût de ce service. Cette section illustre ces différents impacts à travers un exemple.

Impact de la configuration

Les figures 4.1, 4.2 et 4.3 illustrent l'impact de la configuration sur les performances, la disponibilité et le coût d'un service Internet. Nous utilisons ici un service Internet à 2 étages implantant le service de vente en ligne TPC-W (voir la section 8.1 pour plus de détails sur TPC-W). Deux configurations ad-hoc statiques sont prises en compte : $\kappa_1 < 2,\ AC < 1,1 >$, $LC < 500,500 >>$ et $\kappa_2 < 2,\ AC < 3,6 >,\ LC < 200,100 >>$. Dans la première configuration, la capacité de traitement du système vient d'une valeur élevée de la configuration locale. Pour la deuxième configuration, la configuration locale est laissée aux valeurs par défaut des serveurs sous-jacents (c'est-à-dire le serveur Web Tomcat et le serveur de base de données MySQL), et la capacité de traitement du système est principalement due à la configuration architecturale, donc en augmentant le nombre de serveurs alloués au service. Pour chaque configuration, le service Internet est exécuté avec une quantité de charge de 1000 clients accédant simultanément au service, et un type de charge ayant des temps de service de 9 et 14 ms, un ratio de visite de 2,92 sur le deuxième étage, et un temps de réflexion moyen des clients de 7s.

La figure 4.1 présente la latence obtenue avec chacune des configurations, tandis que la figure 4.2 donne le taux de rejet pour ces configurations. La configuration κ_1 induit de mauvaises performances ainsi qu'un taux de rejet important pour le service, car trop peu de serveurs sont affectés au fonctionnement du service. La configuration κ_2, qui a un coût supérieur, fournit de meilleures performances, mais toujours avec un taux de rejet important à cause d'une configuration locale inadéquate.

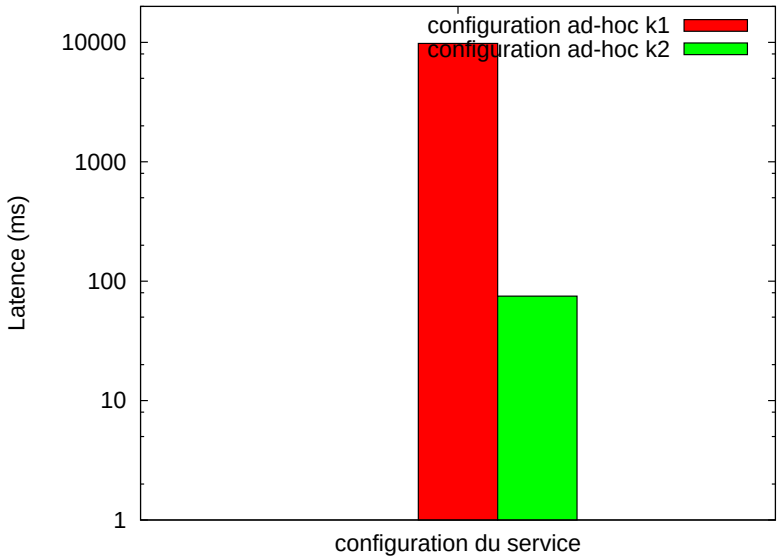

FIGURE 4.1 – Impact de la configuration sur la performance

Impact de la charge

Les figures 4.4, 4.5 et 4.6 présentent l'impact des variations du type et de la quantité de la charge sur la performance, la disponibilité et le coût du service Internet TPC-W. Plusieurs scénarios de variation de charge sont illustrés ici. Le type de charge est d'un type C_1 puis évolue vers un type C_2 au milieu de l'expérience. Pour chaque type de charge, la quantité de charge N évolue par paliers et avec des pics, pour reproduire, à la fois, des changements graduels et des changements subits de charge d'un service Internet. Une configuration ad-hoc du service Internet est utilisée, avec la configuration $\kappa < 2, AC < 2, 3 > LC < 500, 500 >>$. La configuration ad-hoc est comparée avec la configuration optimale du service, c'est-à-dire une configuration qui respecte l'ensemble des contraintes du SLA tout en minimisant le coût du service. Le contrat de qualité de service consiste ici en une latence maximum $\ell_{max} = 2s$ et un taux de rejet maximum $\alpha_{max} = 5\%$. Évidemment, les différentes charges impliquent des comportements différents du service Internet en termes de performance et de disponibilité. La figure 4.4 met en évidence l'impact de la quantité de charge sur la latence. On constate que pour une configuration et un type de charge donnés, l'évolution de la quantité de charge se traduit par une variation de la latence du service. La figure 4.4 met également en évidence l'impact du type de charge sur la latence. Le type de charge étant modifié au milieu de l'expérience (à 3000s), on constate que pour la même quantité de charge, la latence sera différente en fonction du type de charge. Par exemple, avec 1250 clients et la configuration ad-hoc

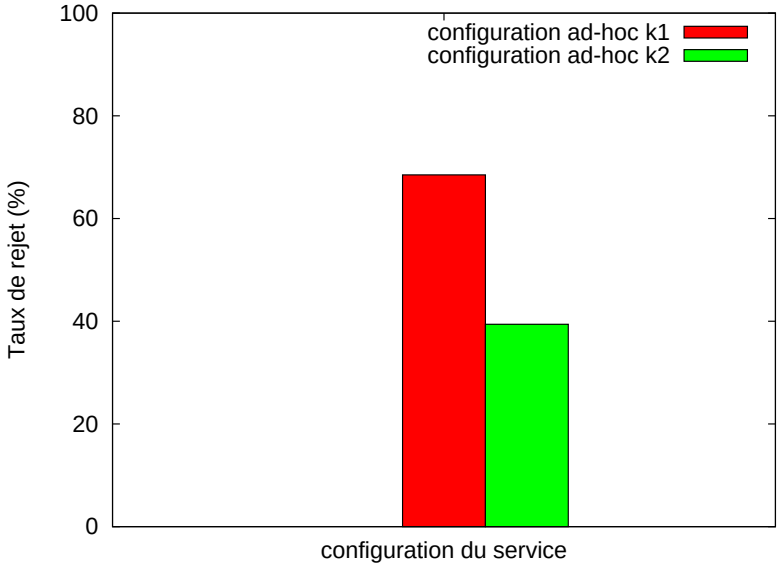

FIGURE 4.2 – Impact de la configuration sur la disponibilité

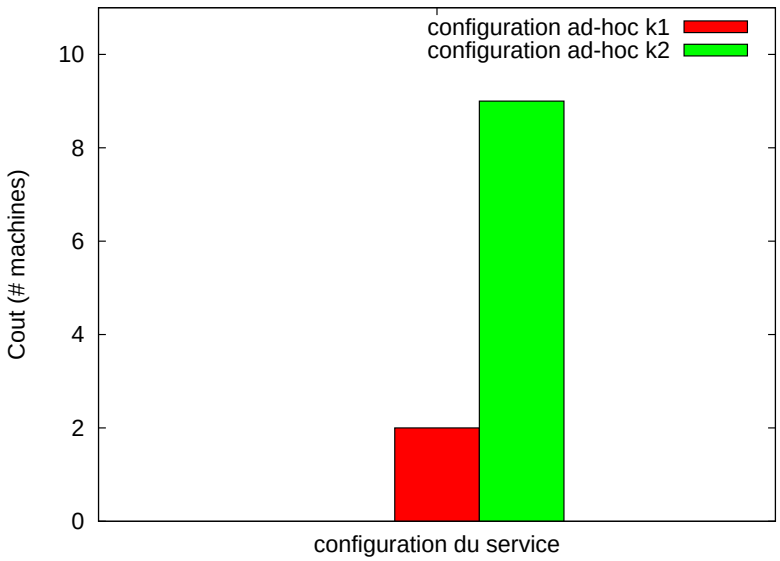

FIGURE 4.3 – Impact de la configuration sur le coût

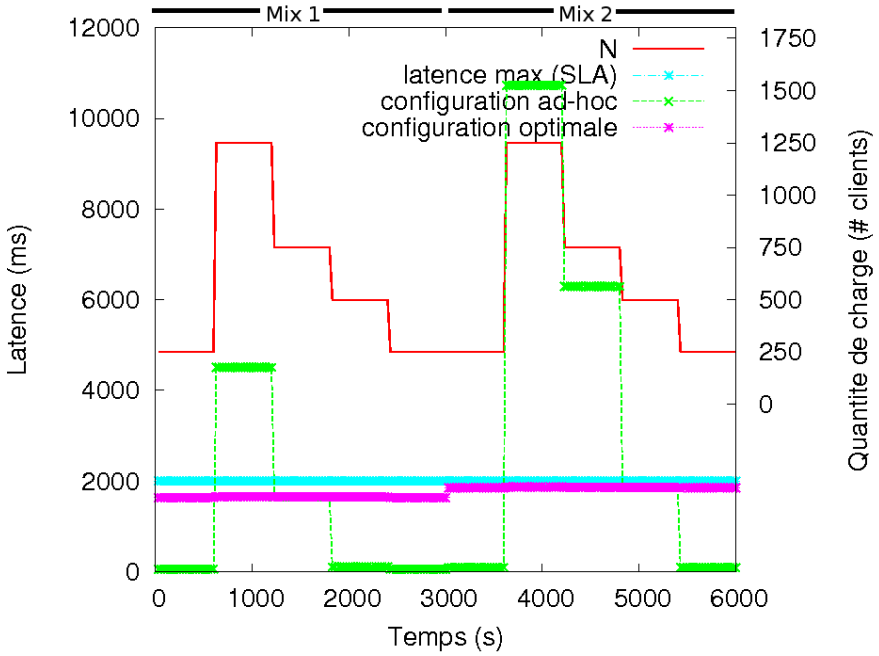

FIGURE 4.4 – Impact de la charge sur la performance

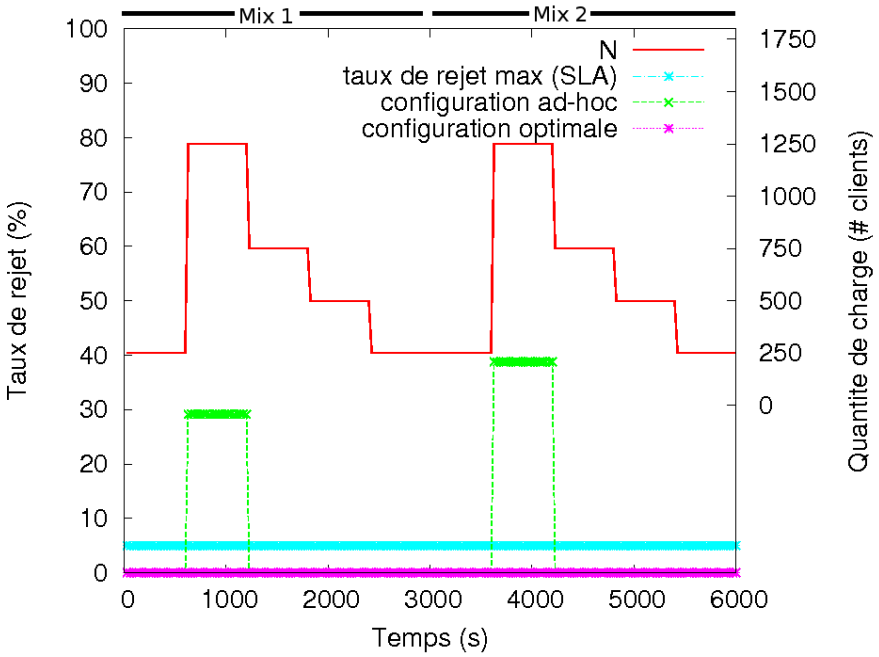

FIGURE 4.5 – Impact de la charge sur la disponibilité

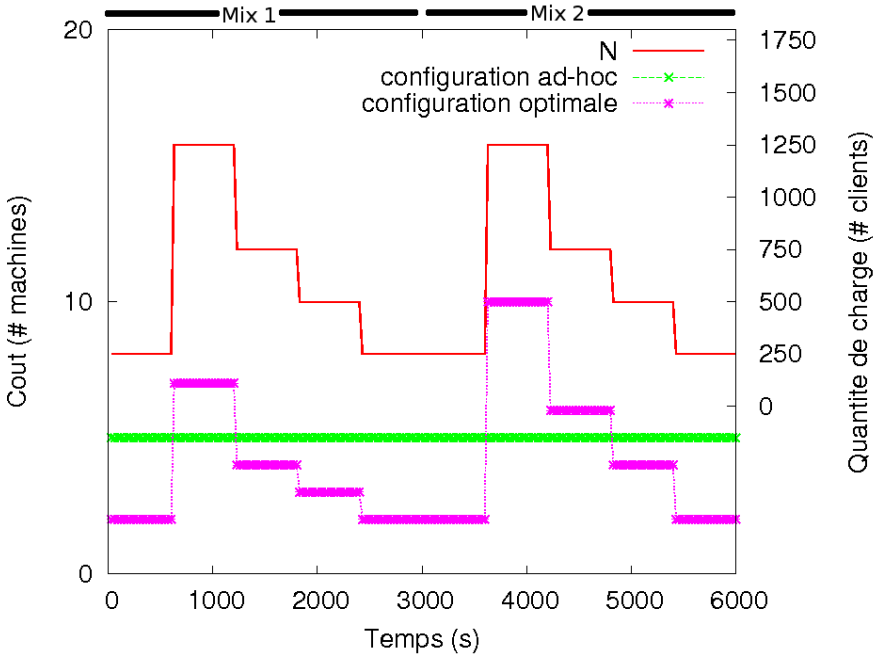

FIGURE 4.6 – Impact de la charge sur le coût

considérée, la latence est de moins de 5 secondes pour le premier type de charge, contre plus de 10 secondes avec le deuxième type de charge. La figure 4.5 met en évidence l'impact de la quantité et du type de charge sur la disponibilité du service. On constate qu'avec 1250 clients, le taux de rejet du service configuré avec la configuration ad-hoc est de 30% pour le premier type de charge, et de 40% pour le deuxième type de charge. La figure 4.6 présente l'évolution du coût du service en fonction de la charge, pour la configuration ad-hoc et la configuration optimale. La configuration ad-hoc étant par définition fixe, son coût n'évolue pas. La configuration optimale s'adapte aux variations de quantité et de type de charge. On peut vérifier que, pour une quantité de charge donnée, le nombre de serveurs alloués au service est plus important avec le deuxième type de charge qu'avec le premier type de charge.

En conclusion, la configuration et la charge d'un service Internet ont toutes deux un impact sur les performances, la disponibilité et le coût du service Internet. La quantité et le type de la charge sont des entrées exogènes du service Internet, dont les variations ne peuvent pas être contrôlées. Par conséquent, pour faire face aux variations de charge et respecter le SLA tout en minimisant le coût du service, la configuration du service doit être adaptée dynamiquement.

Comportement non linéaire des services Internet

L'adaptation dynamique de la configuration d'un service Internet peut être effectuée de différentes façons. Une méthode basée sur un contrôle linéaire du système, utilisant une règle de proportionnalité, est simple à mettre en oeuvre. Elle consiste à trouver par essais successifs une configuration respectant les contraintes de qualité de service pour une charge donnée, puis à calculer la configuration devant être appliquée pour les autres charges en

utilisant une règle de proportionnalité. Cependant, comme illustré sur les figures 4.7 et 4.8, le comportement des services Internet n'est pas linéaire, et une méthode par proportionnalité ne fournira donc pas de garanties concernant le respect des contraintes de qualité de service. La figure 4.7 présente la latence d'un service Internet en fonction de sa quantité de charge. La configuration du service est $\kappa(2, AC < 1, 1 >, LC < 2000, 2000 >)$. On constate que la latence du service n'est pas proportionnelle à la quantité de charge. De même, la figure 4.8 présente l'évolution du débit pour ce même service. Le débit du service n'évolue pas de manière linéaire en fonction de la quantité de charge. Une évolution linéaire de la charge n'impliquant pas une évolution linéaire des performances, l'utilisation d'un contrôle linéaire n'est donc pas valable dans le cadre de la configuration des services Internet.

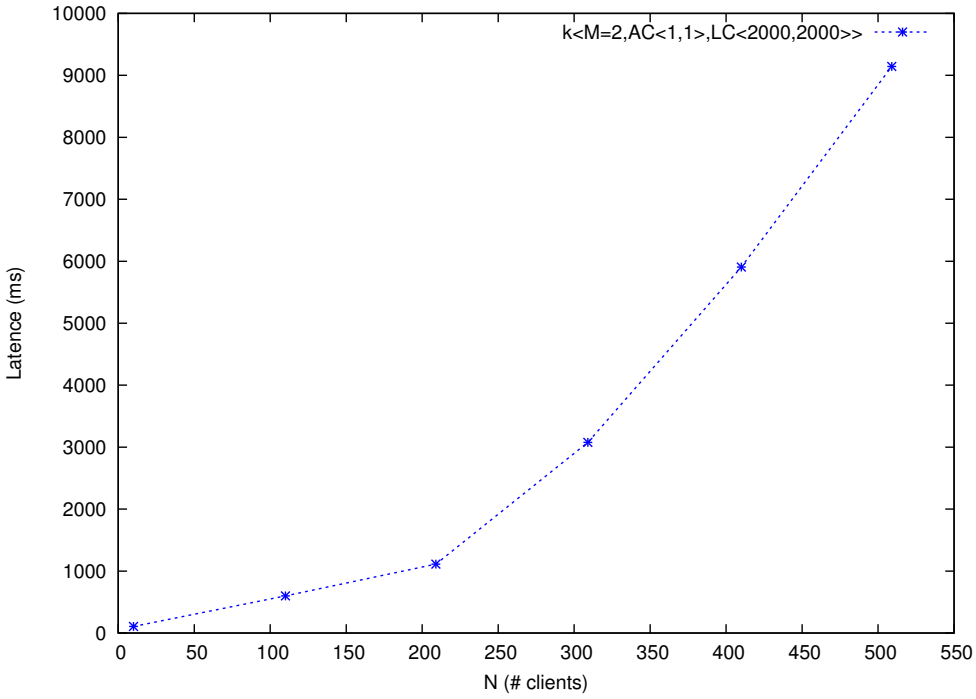

FIGURE 4.7 – Comportement non linéaire de la latence

4.2 Objectifs

Les objectifs de cette thèse sont les suivants :

- Premièrement, il faut pouvoir *quantifier la qualité d'une configuration* en fonction de son coût et du respect ou non des contraintes de qualité de service.
- Il faut ensuite pouvoir trouver la configuration optimale, qui permettra de *garantir l'ensemble des contraintes de qualité de service* tout en *minimisant le coût* du service Internet.
- Ceci suppose de pouvoir *estimer les performances, la disponibilité et le coût d'une configuration* de service Internet en fonction du type de charge et de la quantité de charge du service.
- Nous avons vu dans la section précédente que la charge évoluant au cours du temps, la configuration optimale va changer également. Il faut donc *prendre en compte la variabilité*

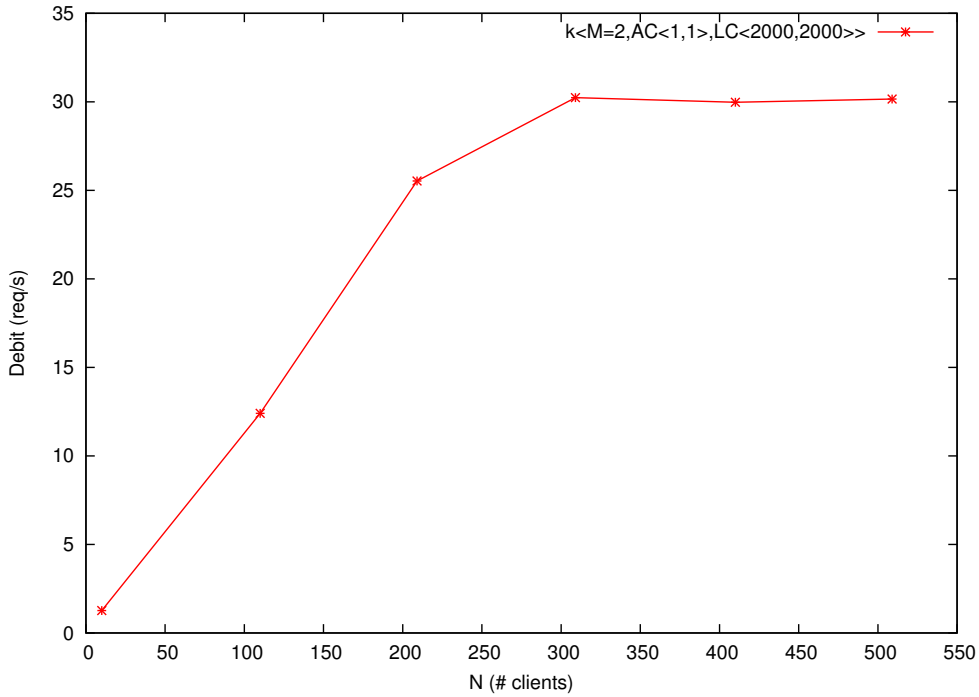

FIGURE 4.8 – Comportement non linéaire du débit

de la charge et adapter la configuration optimale lorsque le type ou la quantité de charge évolue.

Caractérisation de la qualité d'une configuration

Nous avons vu dans la section 4.1 qu'il n'existe pas de configuration optimale absolue, valable quelles que soient la charge ou les contraintes de qualité de service. Il est donc nécessaire de pouvoir quantifier la qualité d'une configuration (c-à-d celle ayant la plus haute utilité), afin de comparer les configurations entre elles pour déterminer la configuration optimale.

Garantie de qualité de service et minimisation du coût

Respecter les objectifs de performance et de disponibilité de service spécifiés dans le contrat de qualité de service (SLA) est primordial pour les fournisseurs de services Internet. Ces derniers doivent donc allouer le nombre nécessaire et suffisant de serveurs aux services Internet, et configurer localement ces derniers de manière appropriée afin de respecter les contraintes données dans le SLA. Il est possible de respecter ces contraintes de qualité de service en allouant un nombre de ressources largement supérieur au maximum nécessaire. Cette technique de sur-allocation est efficace en terme de performances, cependant elle a un coût trop important pour être une solution viable à long terme. Les fournisseurs de services Internet cherchent donc aujourd'hui des solutions pour allouer le nombre minimal de serveurs permettant de satisfaire les objectifs de qualité de service. Notre premier objectif est de trouver une configuration permettant au service Internet de garantir les objectifs du SLA

tout en minimisant le coût de ce service.

Estimation des performances, de la disponibilité et du coût

Afin de pouvoir calculer la configuration optimale d'un service Internet, il faut pouvoir estimer les performances, la disponibilité et le coût de ce service pour n'importe quelle configuration de ce service. Cette prédiction doit être précise malgré les variations de la quantité et de la nature de la charge.

Prise en compte de la variabilité de la charge

Un service Internet est sujet à des variations importantes concernant la quantité et le type de sa charge. La configuration des services Internet doit donc pouvoir s'adapter en fonction des variations de charge. Notre objectif est de fournir un contrôle adaptatif qui détecte automatiquement toute variation de quantité ou de type de charge, puis calcule et applique la configuration optimale en conséquence pour le service Internet.

En résumé, l'objectif de cette étude est de fournir une méthode adaptative et simple d'utilisation, permettant de configurer automatiquement les services Internet afin de garantir des objectifs de qualité de service tout en minimisant leur coût de fonctionnement, et ceci en présence de variations dynamiques de quantité et de type de charge.

4.3 Contributions scientifiques

Cette section présente les principales contributions scientifiques de cette thèse.

- Pour garantir la qualité de service et un coût minimum, nous définissons une *fonction d'utilité* pour les services Internet, qui permet d'agréger les différents critères de performance, disponibilité et coût d'une configuration.
- Nous proposons une *méthode de planification de capacité* pour les services Internet qui, pour un contrat de qualité de service et une charge donnés, calcule la configuration du service qui garantit les objectifs du SLA et minimise le coût du service.
- La planification de capacité est elle-même basée sur un *modèle pour les services Internet multi-étagés*, capable d'estimer la performance, la disponibilité et le coût d'un service Internet.
- Nous proposons un *contrôle adaptatif* des services Internet, avec un contrôle en ligne automatique de la configuration du service qui recalcule automatiquement la configuration optimale lorsque le type ou la quantité de la charge évolue.

Fonction d'utilité pour les services Internet

Nous proposons une fonction d'utilité permettant d'estimer la qualité d'une configuration vis-à-vis, d'une part, d'un ensemble d'objectifs donnés par le SLA, et d'autre part, du coût de cette configuration. Cette fonction d'utilité permet de comparer différentes configurations possibles pour un service Internet afin de choisir la meilleure. La fonction d'utilité que nous proposons est présentée dans le chapitre 6.

Planification de capacité pour les services Internet

La méthode de planification de capacité que nous proposons a pour but de calculer la configuration permettant de respecter les contraintes de qualité de service données tout en minimisant le coût de ce service. Il se base pour cela sur le modèle, calibré avec le contexte courant, ainsi que sur la fonction d'utilité vus précédemment. L'algorithme de planification de capacité effectue une recherche dans l'espace des configurations pour trouver la configuration optimale dans le contexte donné. Le chapitre 6 décrit la conception de l'algorithme de planification de capacité pour des services Internet multi-étagés.

Modèle de performance pour les services Internet.

Nous proposons un modèle analytique pour évaluer les performances, la disponibilité et le coût des services Internet hébergés sur des grappes de machines. Le modèle capture notamment le comportement d'une application multi-étagée, soumise à des variations de quantité et de type de charge, et dont la configuration peut varier. En fonction de ces paramètres, le modèle fournit les performances (latence), la disponibilité (taux d'échec) ainsi que le coût (nombre de serveurs) de la configuration. Nous détaillons le modèle dans le chapitre 5.

Services Internet adaptatifs

Nous avons vu que pour répondre simultanément aux problématiques de minimisation du coût et du respect de la qualité des services Internet, la configuration du système administrée doit être contrôlée dynamiquement. Nous proposons donc ici une solution complète pour la gestion autonome de la configuration des services Internet multi-étagés. La solution proposée consiste en une boucle de contrôle permettant d'administrer automatiquement ce type de services [29]. La boucle de contrôle est matérialisée par un contrôleur autonome administrant le service Internet pendant son fonctionnement. Le contrôleur effectue de manière dynamique les tâches suivantes :

- La collecte d'informations concernant l'environnement d'exécution du service. Ceci concerne en particulier les caractéristiques de la charge entrante, mais aussi la configuration du service Internet.
- La modélisation du système sous-jacent. Cette modélisation sera basée sur les informations collectées précédemment, et permettra de prédire les performances d'une configuration dans le contexte actuel du service Internet. Le paramétrage de ce modèle est effectué de manière automatique.
- La planification de capacité, effectuant le calcul de la configuration optimale en fonction des contraintes de qualité de service. Ce calcul utilise le modèle calibré pour la charge actuelle pour trouver la configuration optimale.
- L'application de cette configuration optimale sur le système administré, grâce à des actionneurs capable de reconfigurer dynamiquement les configurations locales et architecturales du service Internet.

Nous présentons dans le chapitre 7 la réalisation du prototype MoKa, un contrôleur autonome pour les services Internet multi-étagés implantant la solution proposée ici.

Contrôle adaptatif des services Internet

Le calibrage du modèle demande une connaissance de certains critères définissant la quantité et le type de la charge actuelle du service Internet. Afin de simplifier l'instrumentation requise, l'approche proposée comporte un système de calibrage automatique de certains paramètres du modèle. Grâce à ce paramétrage automatique, le modèle s'adapte automatiquement au type de charge du service Internet. Dans notre approche, le service Internet administré est adaptatif, mais également le contrôleur lui-même. Le paramétrage automatique du modèle est présenté dans la section 7.3.

4.4 Synthèse

Ce chapitre met en évidence les différents facteurs pouvant impacter la qualité et le coût des services Internet. Dans ce contexte, les objectifs retenus sont multiples. Premièrement, il faut pouvoir estimer la qualité de la configuration d'un service Internet. Le coût du service doit ensuite être minimisé sans que les contraintes de qualités de service cessent d'être respectées. De plus, les variations de la quantité et du type de charge doivent être détectées automatiquement. L'objectif de cette étude est de fournir une méthode adaptative et simple d'utilisation, permettant de configurer automatiquement les services Internet afin de garantir des objectifs de qualité de service tout en minimisant leur coût de fonctionnement, et ceci en présence de variations dynamiques de quantité et de type de charge. Les principales contributions scientifiques de notre approche concernent la définition d'une fonction d'utilité pour les services Internet, la construction d'un système de détection des variations dans les caractéristiques de la charge, un modèle de performance et de disponibilité pour les services Internet multi-étagés, et enfin une méthode de planification de capacité pour calculer et appliquer la configuration optimale en fonction de contraintes de qualité de service données et de la charge courante. La méthode proposée a été mise en oeuvre dans le prototype logiciel Moka, qui a permis de valider la faisabilité et l'intérêt de l'approche.

Les chapitres 5, 6 et 7 sont respectivement dédiés à la présentation du modèle, à la planification de capacité et à la réalisation du contrôle adaptatif avec le prototype MoKa.

Chapitre 5

Modélisation des services Internet

Nous présentons, dans ce chapitre une modélisation des services Internet multi-étagés. Cette modélisation permet de prédire la performance, la disponibilité et le coût de ces services. Les prédictions du modèle dépendent de la quantité et de la nature de la charge subie par le service, ainsi que de la configuration donnée pour le service.

5.1 Présentation générale

L'objectif de notre modèle est de fournir une estimation précise des performances, de la disponibilité et du coût d'un service Internet avec une configuration et une charge données. Les variables en entrée du modèle sont donc la quantité de charge, le type de charge, ainsi que la configuration du service Internet considéré. Le modèle produit en sortie la latence, le taux de rejet et le coût de ce service. La figure 5.1 schématise les entrées et les sorties du modèle proposé.

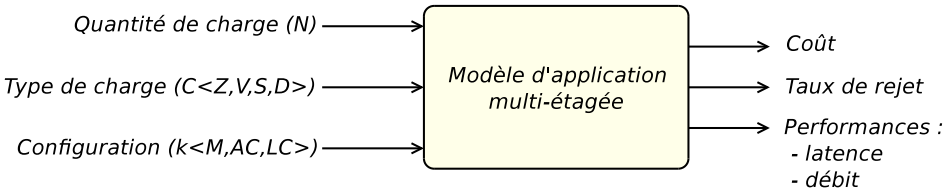

FIGURE 5.1 – Modèle pour les services Internet multi-étagés

5.1.1 Entrées du modèle

Les entrées du modèle peuvent être classées en deux catégories : (i) les entrées exogènes, concernant la quantité et le type de charge du service Internet, sur lesquelles on ne peut pas agir, et (ii) les entrées correspondant à la configuration du service Internet, qui peuvent être modifiées par un administrateur. Les entrées du modèle sont la configuration κ du système, le type de charge C et la quantité de charge N.

- $\kappa < M, AC, LC >$ correspond à la configuration du système pour laquelle on veut calculer les performances. Cette configuration n'est pas forcément la même que la confi-

guration actuelle. On peut par exemple vouloir vérifier les performances d'une configuration avant de l'appliquer, ou utiliser le modèle pour rechercher la configuration optimale.
- N est la quantité de charge, c'est à dire le nombre de clients tentant d'accéder de manière concurrente au service Internet.
- $C < Z, V, S, D >$ est le type de charge auquel est confronté le système. Il comprend le temps d'attente moyen des clients, les ratios de visite sur chaque étage (V_i), les temps de service sur chaque étage (S_i) ainsi que les délais de communication entre étages (D_i, cf. chapitre 2).

Dans la suite de ce chapitre, nous supposons les valeurs de ces entrées connues. Nous verrons, dans le chapitre 7 comment sont obtenues ces valeurs.

5.1.2 Sorties du modèle

Le modèle produit une estimation des performances, de la disponibilité et du coût du service configuré avec la configuration κ et dans le contexte de charge donnés en entrée. Comme indiqué sur la figure 5.1, notre modèle produit 4 sorties :
- la *latence* est le temps moyen nécessaire à l'exécution d'une requête client.
- le *débit* de l'application est le nombre de requêtes traitées avec succès par l'application à chaque seconde.
- le *taux de rejet* est le ratio du nombre de requêtes non admises dans le système par rapport au nombre total de requêtes.
- le *coût* d'une configuration est le nombre de serveurs utilisés par le service Internet.

5.2 Principes du modèle

Nous présentons ici les principes du modèle permettant de capturer le comportement des services multi-étagés soumis à des charges variables. Le modèle est basé sur la théorie des files d'attente. Nous choisissons de modéliser le service comme un réseau de files d'attente. En effet, ce type de modélisation permet de capturer de manière précise les phénomènes de contention pouvant apparaître sur les différents serveurs composant le système multi-étagé. Les bases théoriques des modèles à file d'attente ont été définies dans les années 1980 [47, 18].

5.2.1 Files d'attente

Une file d'attente représente à la fois une ressource critique en terme de performance, ainsi que la file de requêtes en attente pour accéder à cette ressource [38]. Il existe plusieurs façon d'ordonnancer les requêtes à l'intérieur de la file. Les files les plus courantes sont les files FIFO, pour *first in, first out* (premier entré, premier sorti). Dans ce type de file, chaque requête est assurée d'être traitée après la requête qui la précède dans la file, et avant la requête qui la suit dans cette même file. Nous considérons par la suite les files d'attente comme étant des files FIFO, comme c'est le cas dans la plupart des serveurs composant les services Internet. A chaque file d'attente et à chaque type de charge est associée un *temps de service*. Ce temps correspond au temps moyen incompressible nécessaire au traitement d'une requête de ce type de charge par la ressource critique (c-à-d le serveur). Dans notre cadre applicatif, le temps de service sur les ressources est indépendant de la quantité de charge. En effet, le

temps de traitement reste constant même si la latence augmente. Cette augmentation de latence est due au temps passé en attente de traitement dans la file, et non à la modification du temps de service.

Le choix du niveau de granularité pour les ressources représentées par les files d'attente influence la précision du modèle, mais aussi sa facilité d'utilisation. Une granularité fine permet d'obtenir un modèle avec une vue plus riche du système. Son utilisation est cependant rendue difficile par le nombre plus important de paramètres du modèle, ces derniers dépendant entre autres du nombre de files d'attente dans le système. Inversement, une granularité importante permet une utilisation plus facile du modèle en limitant la complexité du paramétrage de ce dernier. Le modèle peut cependant perdre en précision dans certains cas. Dans notre cadre applicatif, nous modélisons chaque serveur par une file d'attente.

5.2.2 Files d'attente pour services multi-étagés dupliqués

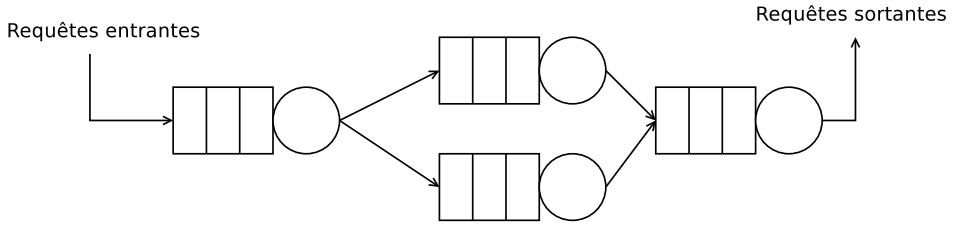

FIGURE 5.2 – Réseau ouvert de files d'attente

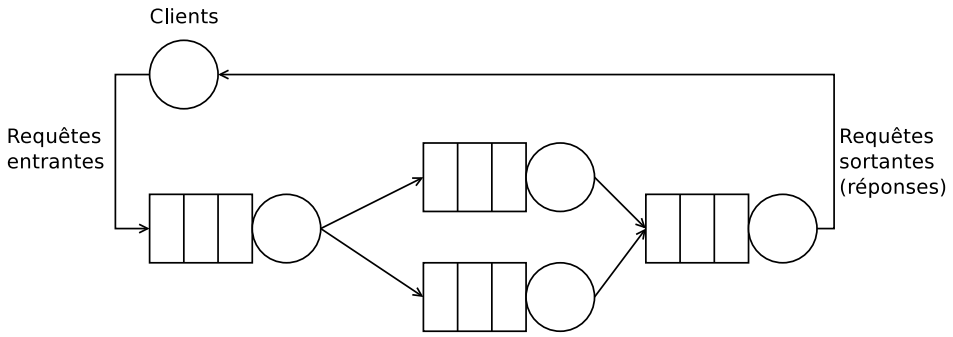

FIGURE 5.3 – Réseau fermé de files d'attente

Un service multi-étagé dupliqué est modélisé par un réseau de files d'attente. Les réseaux de files d'attente sont divisés en deux catégories, les réseaux ouverts (boucles ouvertes) et les réseaux fermés (boucles fermées). Dans les *réseaux ouverts*, les requêtes sont envoyées vers le système indépendamment de la vitesse de traitement de ce dernier. Les clients n'attendent donc pas la réponse à leur requête pour envoyer une nouvelle requête. La figure 5.2 présente un exemple de réseau ouvert. Dans les *réseaux fermés*, les clients attendent la réponse à leur requête avant d'en envoyer une autre. Nous supposons que toute requête admise dans le service sera traitée sans erreur, quel que soit le niveau de qualité de service fourni par le service. Les éventuelles erreurs applicatives pouvant résulter d'une surcharge du service Internet ne font pas l'objet de cette étude. Dans les réseaux fermés, les clients marquent un temps de réflexion entre la réception d'une réponse et l'émission de la requête suivante

(temps d'attente, ou *think time*). Dans les réseaux fermés, les clients sont donc vus comme un délai supplémentaire dans la boucle de traitement. La figure 5.3 présente un exemple de réseau fermé. Dans le cadre des services Internet, la modélisation se base sur un réseau fermé de files d'attente. En effet, dans les applications considérées les clients attendent la réponse à une requête avant d'en émettre une suivante. La figure 5.4 présente un exemple de modélisation de service Internet multi-étagé dupliqué.

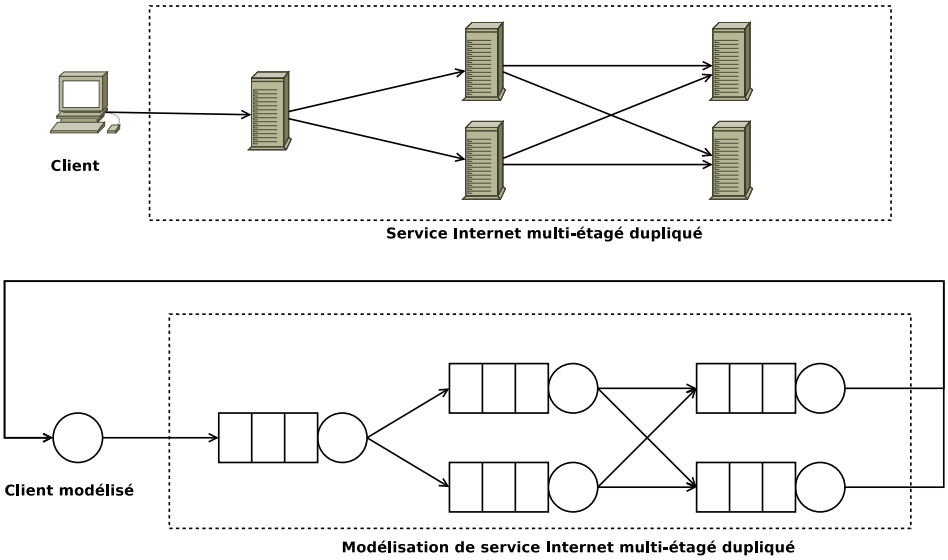

FIGURE 5.4 – Modélisation d'une application multi-étagée

La figure 5.5 présente un exemple de service Internet à 3 étages ayant une configuration $\kappa(3, AC < 1, 1, 2 >, LC < 20, 15, 3 >)$, une quantité de charge de 30 clients et un type de charge caractérisé entre autres par des ratios de visite $V < 1, 0.5, 2 >$. Cet exemple illustre le comportement des 30 requêtes clients tentant d'accéder aux différents étages et files d'attente. Ainsi, parmis les $Nt_1 = 30$ clients tentant d'accéder à l'étage T_1, $Nr_1 = 10$ sont rejetés à cause de la configuration locale (c'est-à-dire le contrôle d'admission) sur cet étage et $Na_1 = 20$ clients sont admis sur le serveur de T_1. Ensuite, les 20 requêtes clients traitées par T_1 génèrent $Nt_2 = 10$ sous-requêtes vers l'étage T_2 (du fait du ratio de visite $V_2 = 0.5$). L'ensemble des 10 requêtes sont admises sur le serveur de T_2, car ce nombre est inférieur à la configuration locale de T_2 ($Na2 = 10$). Enfin, les 10 requêtes sur T_2 induisent au total 40 requêtes sur T_3 (avec un ratio de visite $V_3 = 2$). Cependant, à cause de la communication synchrone entre les étages d'un service Internet, une requête sur T_2 induit à un instant donné au plus une requête vers T_3, et en moyenne 4 requêtes successives vers T_3. Parmis ces 10 requêtes, $Nr_3 = 4$ requêtes sont rejetées par la configuration locale de T_3, et $Na_3 = 6$ requêtes sont admises et réparties sur les deux serveurs composant cet étage. En résumé, sur les 30 requêtes client tentant d'accéder au service multi-étagé, un total de 14 sont rejetées et 16 sont traitées, ce qui donne un taux de rejet de 47%.

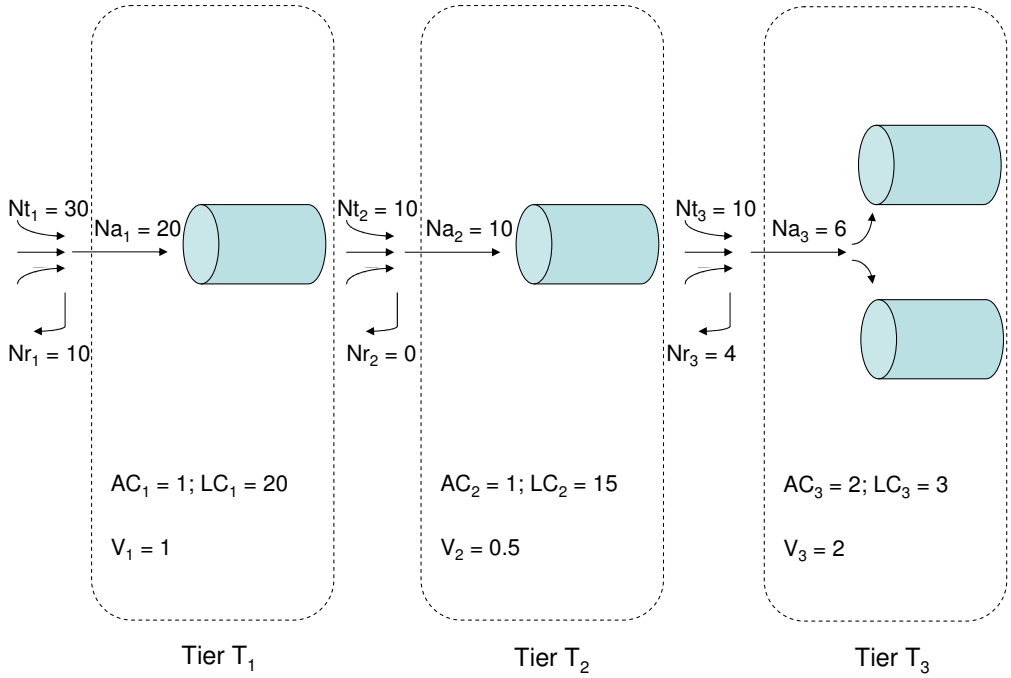

FIGURE 5.5 – Modélisation d'une application multi-étagée - prise en compte de la configuration locale

5.3 Algorithme de prédiction de performance, de disponibilité et de coût

L'algorithme 1 décrit comment le modèle prédit la latence, le taux de rejet et le coût d'un service Internet multi-étagé. Cet algorithme se base sur l'algorithme MVA (*Mean Value Analysis*) [47], et l'étend pour entre autres prendre en compte la duplication sur chaque étage (c'est-à-dire la configuration locale), les différents types de charge, et pour inclure les prédictions du taux de rejet et du coût. Pour prendre en compte la duplication sur un étage, notre modèle distribue la quantité de traitement à effectuer sur cet étage entre les différents duplicats composant l'étage. La distinction entre temps de service S_i et délai de communication entre étages D_i est nécessaire, car les traitements induisant les temps de service peuvent être répartis sur les duplicats, ce qui n'est pas le cas des délais D_i (ajouter plus de duplicats ne permet pas une communication plus rapide entre étages). L'algorithme proposé est constitué de quatre parties.

La première partie de l'algorithme (lignes 1 à 13), calcule la quantité de requêtes admises et la quantité de requêtes rejetées sur chaque étage du service multi-étagé. L'algorithme prend en entrée la configuration du service $\kappa(M, AC, LC)$, la quantité de charge N et le type de charge C avec les ratios de visite. Les lignes 1 à 6 de l'algorithme calculent Nt_i, le nombre de requêtes tentant d'accéder à l'étage T_i, en fonction des ratios de visite des étages. La ligne 6 garantit que si Na_{i-1} requêtes sont admises à l'étage T_{i-1}, il n'y aura pas plus que Na_{i-1} requêtes tentant d'accéder à T_i, car les communications entre les étages sont synchrones. Les lignes 7 à 9 appliquent le MPL de T_i pour calculer Na_i, le nombre de requêtes admises sur T_i, et Nr_i, le nombre de requêtes rejetées par T_i. De plus, parce qu'une requête admise sur T_i peut être rejetée par un des étages suivantes ($T_{i+1} \dots T_M$), les lignes 10 et 11 calculent Na'_i, le nombre de requêtes admises sur T_i et non rejetées par les étages suivants. Enfin, les lignes 12 et 13 produisent Nr, le nombre total de requêtes rejetées, et le nombre total de requêtes admises Na.

La deuxième partie de l'algorithme (lignes 14 à 32) est dédiée à la prédiction de la latence des requêtes adressées au service. Premièrement, les lignes 15 à 17 initialisent la longueur des files d'attente et la demande de service sur chaque étage. La demande de service W_i représente la quantité de travail nécessaire pour traiter une requête sur un étage T_i, à l'exclusion du délai de communication vers cet étage D_i. Les lignes 18 à 31 parcourent les différents étages de l'application, depuis le dernier vers le premier, afin d'estimer la latence cumulée des requêtes sur chaque étage T_i. La latence d'une requête admise sur l'étage T_i et admise sur les étages suivants $T_{i+1} \dots T_M$ est désignée par ℓa_i, et ℓr_i est la latence d'une requête admise par T_i mais rejetée par un des étages suivants. Dans ce dernier cas, la requête ne sera pas considérée comme admise, mais doit tout de même être prise en compte dans l'algorithme car elle a un impact sur la longueur des files, et donc sur les temps de réponse et la latence calculée pour les requêtes admises. Les lignes 19 à 22 introduisent les requêtes une par une à l'étage T_i, calculent la demande de service pour chaque duplicat de l'étage T_i, et estiment le temps de réponse des requêtes. A la ligne 22, la fonction *Max* est utilisée pour s'assurer que la demande de service n'est pas inférieure au temps incompressible de traitement W_i, dépendant du temps de service défini par le type de charge. Ensuite, les lignes 23 à 28 cumulent les temps de réponse pour calculer ℓa_i, la latence des requêtes admises aux étages $T_i \dots T_M$, et ℓr_i, la latence des requêtes admises par T_i mais rejetées par un des étages suivants. Ces valeurs sont ensuite utilisées des lignes 29 à 32 pour calculer la longueur des

Algorithme 1: Modélisation des services Internet multi-étagés

Input :
$\kappa(M, AC, LC)$: service configuration ; N : workload amount ; $C(Z, V, S, D)$: workload mix
Output : ℓ : service latency ; α : service abandon rate ; ω : service cost ; τ : throughput

1 **/* Admitted and rejected requests */**

2 **for** $i := 1 ; i \leq M ; i++$ **do**
3 **if** $i = 1$ **then**
4 $Nt_i := N$
5 **else**
6 $Nt_i := Min(Na_{i-1}, Na_{i-1} * \frac{V_i}{V_{i-1}})$
7 $MPL_i := LC_i \cdot AC_i$ /* total MPL at T_i */
8 $Na_i := Min(MPL_i, Nt_i)$ /* requests admitted at T_i */
9 $Nr_i := Nt_i - Na_i$ /* requests rejected at T_i */

10 **for** $i := 1 ; i \leq M ; i++$ **do**
11 $Na'_i := Na_i - \sum_{j=i+1}^{M} Nr_j$ /* requests admitted at $T_i..T_M$ */

12 $Nr := \sum_{i=1}^{M} Nr_i$ /* total rejected requests */
13 $Na := N - Nr$ /* total admitted requests */

14 **/* Service latency */**

15 **for** $i := 1 ; i \leq M ; i++$ **do**
16 $Ql_i := 0$ /* total queue length at T_i*/
17 $W_i := (S_i - D_i) \cdot V_i$ /* service demand at T_i*/

18 **for** $i := M ; i \geq 1 ; i--$ **do**
19 /* introduce requests one by one at T_i */
20 **for** $j := 1 ; j \leq Na_i ; j++$ **do**
21 $W'_i := \frac{(1 + Ql_i) \cdot W_i}{AC_i}$ /* service demand per server replica at T_i */
22 $R_i = Max(W'_i, W_i) + D_i \cdot V_i$ /* response time of request admitted at T_i */
23 **if** $i < M$ **then**
24 $\ell a_i := R_i + \ell a_{i+1}$ /* latency of request admitted at $T_i..T_M$ */
25 $\ell r_i := R_i + \ell r_{i+1}$ /* latency of request admitted at T_i and rejected at $T_{i+1}..T_M$ */
26 **else**
27 $\ell a_i := R_i$ /* latency of request admitted at $T_i..T_M$ */
28 $\ell r_i := R_i$ /* latency of request admitted at T_i and rejected at $T_{i+1}..T_M$ */
29 $\tau a_i = \frac{j \cdot Na'_i / Na_i}{\ell a_i + Z}$ /* throughput of requets admitted at $T_i..T_M$ */
30 $\tau r_i = \frac{j \cdot (Na_i - Na'_i)/Na_i}{\ell r_i + Z}$ /* throughput of requets admitted at T_i and rejected at $T_{i+1}..T_M$ */
31 $Ql_i = (\tau a_i + \tau r_i) \cdot R_i$ /* T_i's total queue length with Little's law */

32 $\ell := \ell a_1$ /* final latency of requests admitted at $T_1..T_M$ */

33 **/* Service throughput and abandon rate */**

34 $\tau a := \frac{Na}{\ell a_1 + Z}$ /* throughput of requets admitted at $T_1..T_M$ */
35 $\tau r := \frac{Nr - Nr_1}{\ell r_1 + Z}$ /* throughput of requets admitted at T_1 and rejected at $T_2..T_M$ */
36 $\tau r' := \frac{Nr_1}{Z}$ /* throughput of requests rejected at T_1 */
37 $\alpha := \frac{\tau r + \tau r'}{\tau r + \tau r' + \tau a}$ /* total abandon rate */
38 $\tau := \tau a$ /* total throughput */

39 **/* Service cost */**

40 $\omega := \sum_{i=1}^{M} AC_i ;$ /* total servers */

files en utilisant la loi de Little [33]. Enfin, la latence totale d'une requête client est définie à la ligne 32, comme étant la latence d'une requête admise sur T_1 et jamais rejetée par les étages suivants.

La troisième partie de l'algorithme (lignes 33 à 38) permet d'estimer le débit ainsi que le taux de rejet du service. L'algorithme commence par calculer τ_a, le débit de requêtes admises (et jamais rejetées) sur $T_1 \ldots T_M$, ainsi que τ_r, le débit de requêtes admises sur T_1 mais rejetées par un des étages suivants, et τ'_r, le débit de requêtes rejetées par T_1 à cause du contrôle d'admission. Ces différentes valeurs sont ensuite utilisées pour produire le taux de rejet global du service (ligne 37) et le débit du service (ligne 38).

Enfin, la quatrième et dernière partie de l'algorithme (lignes 39 et 40) calcule le coût total du service multi-étagé dupliqué, en terme de nombre de serveurs constituant le système.

Complexité algorithmique

La complexité de l'algorithme du modèle est fonction du nombre de clients accédant au système (N), ainsi que du nombre d'étages composant le système (M). Le nombre de clients N est considéré comme étant très supérieur à M ($N >> M$). L'équation 5.1 donne la complexité du modèle proposé.

$$O(N.M) \tag{5.1}$$

5.4 Synthèse

Nous présentons dans ce chapitre une modélisation des services Internet multi-étagés. Ce modèle prend en compte la quantité et le type de charge du service ainsi que sa configuration. Il calcule à partir de ces entrées la latence, la disponibilité, le débit et le coût du service. Nous donnons enfin la complexité du modèle proposé. Un système de paramétrage automatique du modèle, basé sur le monitorage en ligne du service Internet administré, est présenté dans le chapitre 7.

Le chapitre suivant présente une solution de planification de capacité pour les services Internet soumis à des contraintes de qualité de service. Cette planification de capacité se base sur le modèle décrit dans ce chapitre pour calculer les performances, la disponibilité et le coût d'une configuration.

Chapitre 6

Planification de capacité des services Internet

Nous proposons dans ce chapitre une méthode de planification de capacité pour les services Internet multi-étagés dupliqués soumis à des contraintes de qualité de service multiples, ainsi qu'une quantité et un type de charge variables dans le temps.

6.1 Objectifs

La planification de capacité a pour but de calculer la configuration optimale d'un service Internet multi-étagé dupliqué. Ce calcul se base sur un algorithme de recherche de la configuration respectant l'ensemble des contraintes de qualité de service tout en minimisant le coût de fonctionnement de ce service. La figure 6.1 présente le rôle de la planification de capacité. Elle prend en entrées des informations sur la quantité et le type de charge, ainsi que les critères de performance (latence maximum) et de disponibilité (taux de rejet maximum) à respecter. Dans la suite de ce chapitre, nous supposons les valeurs de ces entrées connues. Nous verrons, dans le chapitre 7 comment sont obtenues ces valeurs. La planification de capacité produit en sortie la configuration optimale du service Internet, respectant l'ensemble des contraintes avec le coût minimum. Pour cela, la planification de capacité se base sur le modèle décrit au chapitre 5, ainsi que sur une fonction d'utilité que nous présentons dans la section 6.2.

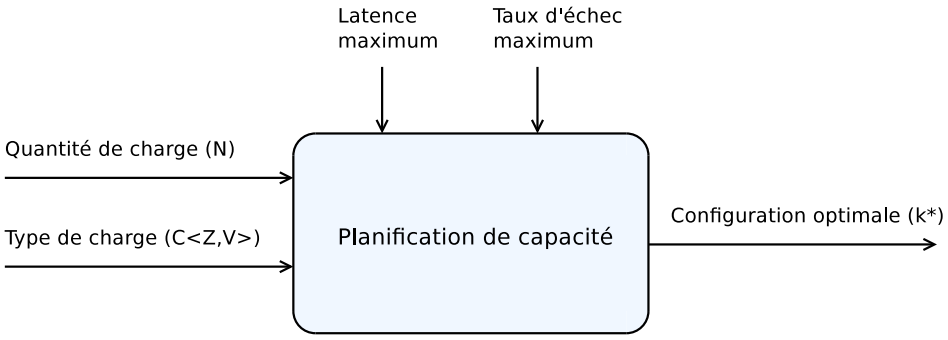

FIGURE 6.1 – Planification de capacité pour les applications multi-étagées

6.2 Fonction d'utilité

Le rôle de la fonction d'utilité est d'agréger les différents critères de performance, de disponibilité et de coût d'une configuration en une seule métrique, comme représenté sur la figure 6.2. La fonction d'utilité permet ainsi de synthétiser les objectifs de qualité de service et de coût à atteindre, puis de comparer les différentes configurations entre elles afin de choisir celle qui répond le mieux aux besoins. Ainsi, une configuration avec une valeur d'utilité élevée sera toujours meilleure qu'une configuration avec une valeur d'utilité plus faible.

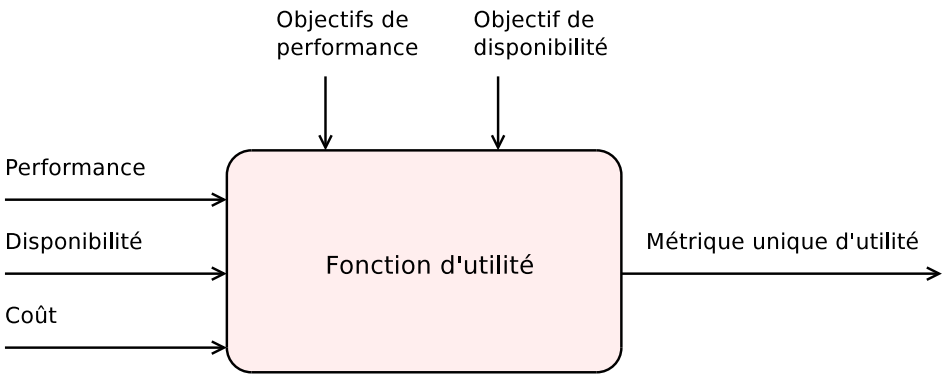

FIGURE 6.2 – Fonction d'utilité

Les critères pris en compte dans la fonction d'utilité sont de deux types.

- Les critères *logiques* sont vérifiés ou non. Dans notre cadre applicatif, les contraintes de qualité de service sont des critères logiques. En effet, le contrat de qualité de service (SLA) impose le respect de ces contraintes. On ne différencie donc pas les cas où le critère est faiblement ou largement respecté.
- Les critères *continus* prennent une valeur dans $\mathbb{R}$. La mesure du coût de la configuration est un critère continu, portant sur le nombre de machines allouées au service.

Définition

Étant donnés les objectifs de performance du contrat de qualité de service concernant la latence maximum ℓ_{max} et le taux d'échec maximum α_{max} qu'une application multi-étagée doit respecter, nous définissons la préférence de performance :

$$PP(\ell, \alpha) = (\ell \leq \ell_{max}) \cdot (\alpha \leq \alpha_{max}) \tag{6.1}$$

où ℓ et α représentent respectivement la latence et le taux d'échec d'un service Internet multi-étagé. A noter que $\forall \ell, \forall \alpha, PP(\ell, \alpha) \in \{0, 1\}$, suivant si les critères de qualité de service sont vérifiés ou non (critères logiques). A partir de la préférence de performance et du coût de l'application multi-étagée, nous définissons une fonction d'utilité Θ qui combine ces différents critères comme suit :

$$\Theta(\ell, \alpha, \omega) = \frac{M \cdot PP(\ell, \alpha)}{\omega} \tag{6.2}$$

où ω est le coût (en nombre de ressources) et M le nombre d'étages de l'application multi-étagée. Le nombre d'étages, M, est utilisé dans l'équation 6.2 à fin de normalisation. Ici,

$\forall \ell, \forall \alpha, \forall \omega, \Theta(\ell, \alpha, \omega) \in [0, 1]$, car $\omega \geq M$ (avec au moins un serveur par étage) et $PP(\ell, \alpha) \in 0, 1$.

Une valeur élevée de la fonction d'utilité signifie que, d'une part, les performances de l'application respectent les objectifs spécifiés dans le contrat de qualité de service (SLA) et que, d'autre part, le coût de fonctionnement de l'application est faible. Inversement, une valeur faible de la fonction d'utilité signifie que le coût de la configuration est élevé, ou les que les contraintes de qualité de service ne sont pas respectées. Dans ce dernier cas la valeur de la fonction d'utilité sera nulle.

Par soucis de concision, nous employons également la notation $PP(\kappa)$ pour désigner la préférence de performabilité d'une configuration κ donnée (produisant une latence et un taux de rejet pour une charge donnée) ainsi que la notation $\Theta(\kappa)$ pour désigner l'utilité d'une configuration.

Exemple

Afin d'illustrer le comportement de la fonction d'utilité, un ensemble de données synthétiques pour la configuration d'une application à 3 étages est donné dans le tableau 6.1 et la figure 6.3. Trois charges différentes sont considérées, avec respectivement 10, 100 et 1000 clients, avec $S =< 4, 11, 7 >$ et $V =< 1, 2.4, 4.3 >$. Les contraintes de qualité de service du SLA imposent $\ell_{max} < 1000$ ms et $\alpha_{max} < 25\%$. Dans le tableau 6.1, pour chaque charge, différentes configurations κ_i de l'application multi-étagée sont étudiées, avec différentes valeurs aux niveaux local et architectural. La figure 6.3 donne, pour chaque configuration, la valeur correspondante de la fonction d'utilité. Par exemple, avec une charge de 1000 clients, la plus haute valeur de la fonction d'utilité est obtenue avec la configuration κ_{16}, qui garantit les objectifs de performance avec un total de 9 ressources. Avec la même charge, si l'application a une configuration architecturale plus faible (donc moins de ressources), comme κ_{13}, ou une configuration locale plus faible (donc un MPL plus bas) comme κ_{14}, la préférence de performance pour la latence et le taux d'échec ne sont plus garantis. Inversement, si une configuration architecturale plus élevée comme κ_{17} ou κ_{18} est utilisée, cela augmente le coût de l'application multi-étagée sans améliorer les performances, et mène donc à une diminution de la fonction d'utilité.

Dans le cas d'une configuration locale plus élevée, comme κ_{15}, l'application sera soumise à un degré de concurrence plus élevé et ne pourra plus respecter les objectifs de performance, ce qui réduira la valeur de la fonction d'utilité à 0. De même, avec une charge de 100 clients et 10 clients, les configurations garantissant les performances de l'application avec un coût minimal sont celles maximisant la fonction d'utilité, respectivement κ_{10} et κ_2.

6.3 Algorithme de planification de capacité

L'objectif de la planification de capacité est de calculer une configuration optimale du service Internet multi-étagé dupliqué, pour une quantité et un type de charge donnés, afin de respecter les contraintes de latence et de taux de rejet du contrat de qualité de service, tout en minimisant le coût du service. Ainsi, une configuration optimale κ^* est une configuration qui a la plus haute valeur d'utilité θ^*.

La figure 6.4 illustre le fonctionnement de la planification de capacité. La quantité et le type de charge du service Internet sont utilisés pour calibrer le modèle. La planification de

Quantité de charge	Configuration
10 clients	$\kappa_1 < 3, AC < 1,1,1 >, LC < 7,50,50 >>$ $\kappa_2 < 3, AC < 1,1,1 >, LC < 100,100,50 >>$ $\kappa_3 < 3, AC < 1,2,1 >, LC < 100,50,100 >>$ $\kappa_4 < 3, AC < 1,2,2 >, LC < 200,100,150 >>$ $\kappa_5 < 3, AC < 2,2,3 >, LC < 200,200,100 >>$ $\kappa_6 < 3, AC < 2,3,4 >, LC < 300,200,200 >>$
100 clients	$\kappa_7 < 3, AC < 1,1,1 >, LC < 50,50,50 >>$ $\kappa_8 < 3, AC < 1,2,2 >, LC < 50,50,50 >>$ $\kappa_9 < 3, AC < 2,2,2 >, LC < 20,50,50 >>$ $\kappa_{10} < 3, AC < 1,2,2 >, LC < 100,50,50 >>$ $\kappa_{11} < 3, AC < 2,2,2 >, LC < 100,100,50 >>$ $\kappa_{12} < 3, AC < 2,4,4 >, LC < 200,100,200 >>$
1000 clients	$\kappa_{13} < 3, AC < 2,2,4 >, LC < 100,200,100 >>$ $\kappa_{14} < 3, AC < 2,3,4 >, LC < 200,200,200 >>$ $\kappa_{15} < 3, AC < 2,3,4 >, LC < 500,400,400 >>$ $\kappa_{16} < 3, AC < 2,3,4 >, LC < 450,200,200 >>$ $\kappa_{17} < 3, AC < 2,4,4 >, LC < 450,200,200 >>$ $\kappa_{18} < 3, AC < 4,3,4 >, LC < 450,200,200 >>$

TABLE 6.1 – Exemples de configurations et charges de service Internet

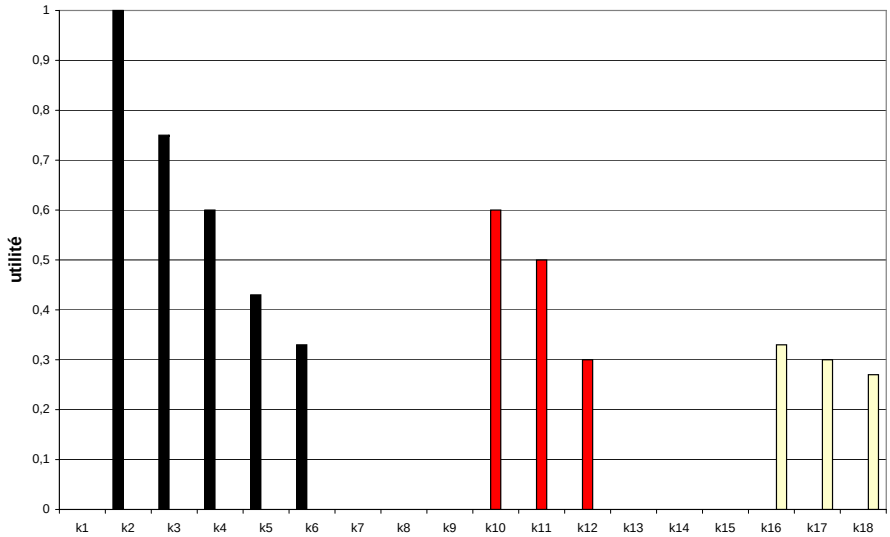

FIGURE 6.3 – Fonction d'utilité

capacité se base sur le modèle ainsi calibré pour obtenir les prédictions de performance, de disponibilité et de coût pour une configuration de base. La fonction d'utilité compare ensuite ces prédictions avec les objectifs de qualité de service spécifiés dans le SLA, et fournit une valeur d'utilité. Si les objectifs de qualité de service ne sont pas respectés ($PP(\kappa) = 0$) ou que des serveurs peuvent être économisés, la planification de capacité itère jusqu'à trouver la configuration optimale.

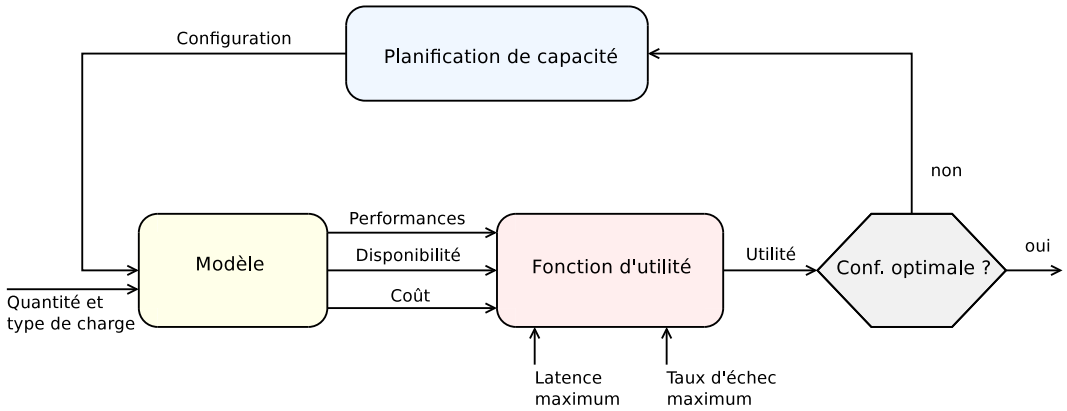

FIGURE 6.4 – Planification de capacité

La planification de capacité est donnée dans l'algorithme 2. L'algorithme prend en paramètre la quantité et le type de charge du service Internet, ainsi que le nombre d'étages de l'application, les contraintes de latence et de taux de rejet du contrat de qualité de service à respecter, et le modèle sous-jacent (cf. algorithme 1. L'algorithme est composé de deux parties principales : une première partie calcule une configuration garantissant les contraintes de qualité de service, puis une seconde partie réduit le coût du service.

La première partie de l'algorithme (lignes 6 à 17) augmente le nombre de serveurs alloués à chaque étage du service Internet (cf. ligne 11). Il adapte ensuite la configuration locale des serveurs pour distribuer les requêtes des clients sur l'ensemble des duplicats pour chaque étage (ligne 13). Cependant, si une requête sur un étage induit plus d'une sous-requête sur les étages suivants (c'est-à-dire $V_{i-1} \leq V_i$), et du fait des communications synchrones entre les étages, le nombre de requêtes concurrentes sur les étages suivants n'excède pas le nombre de requêtes concurrentes sur un étage (ligne 15). Après cela, la configuration du service ainsi obtenue est utilisée avec la quantité et le type de charge pour prédire la latence et le taux de rejet du service ; ceci se base sur le modèle analytique sous-jacent (cf. algorithme 1). Ce processus est répété jusqu'à ce qu'une configuration respectant le contrat de qualité de service soit trouvée.

La seconde partie de l'algorithme (lignes 18 à 50) a pour but de réduire le nombre de serveurs alloués au service pour minimiser son coût global. Pour chaque étage T_i, le nombre minimum de serveurs nécessaires pour garantir le contrat de qualité de service se situe entre 1 et la valeur de AC_i calculée dans la première partie de l'algorithme. Pour estimer la valeur minimum de AC_i, une recherche dichotomique est effectuée sur cet intervalle (lignes 20 à 22). La configuration locale LC_i est adaptée en fonction pour répartir la charge entre les duplicats (lignes 23 à 26). Ensuite, la latence et le taux de rejet pour la configuration du service résultante sont prédits. S'ils respectent les contraintes du contrat de qualité de service, le nombre de serveurs sur cet étage est réduit en poursuivant la recherche d'une valeur plus

Algorithme 2: Planification de capacité

Input :
N : workload amount
$C(Z, V, S, D)$: workload mix
Output :
$\kappa(M, AC, LC)$: service configuration
1 **Parameters :**
2 M : *#tiers of the service*
3 ℓ_{max} : *SLA latency constraint*
4 α_{max} : *SLA abandon rate constraint*
5 $MO(\kappa, N, C) \rightarrow (\ell, \alpha, \omega)$: *service prediction model*

6 **/* Meet SLA */**

7 **for** $i := 1 ; i \leq M ; i++$ **do**
8 $\quad AC_i := 0$

9 **repeat**
10 $\quad$ **for** $i := 1 ; i \leq M ; i++$ **do**
11 $\quad\quad AC_i++$ /* *Increase number of servers* */
12 $\quad\quad$ **if** $(i = 1)$ *or else* $(V_{i-1} > V_i)$ **then**
13 $\quad\quad\quad LC_i := \frac{N \cdot V_i}{AC_i}$ /* *A request on* T_{i-1} *induces at most one request on* T_i */
14 $\quad\quad$ **else**
15 $\quad\quad\quad LC_i := \frac{LC_{i-1} \cdot AC_{i-1}}{AC_i}$ /* *A request on* T_{i-1} *induces in total more than one request on* T_i */
16 $\quad\quad (\ell, \alpha, \omega) := MO(\kappa(M, AC, LC), N, C)$ /* *Call the prediction model* */
17 **until** $PP(\ell, \alpha) = 1$ /* κ *guarantees SLA* */;

18 **/* Minimize cost */**

19 **for** $i := 1 ; i \leq M ; i++$ **do**
20 $\quad minAC := 1 ; maxAC := AC_i$
21 $\quad$ **while** $minAC \neq maxAC$ **do**
22 $\quad\quad AC_i := minAC + \frac{maxAC - minAC}{2}$ /* *Dichotomic search to reduce* AC_i*, and thus* ω*/
23 $\quad\quad$ **if** $(i = 1)$ *or else* $(V_{i-1} > V_i)$ **then**
24 $\quad\quad\quad LC_i := \frac{N \cdot V_i}{AC_i}$ /* *A request on* T_{i-1} *induces at most one request on* T_i */
25 $\quad\quad$ **else**
26 $\quad\quad\quad LC_i := \frac{LC_{i-1} \cdot AC_{i-1}}{AC_i}$ /* *A request on* T_{i-1} *induces in total more than one request on* T_i */
27 $\quad\quad (\ell, \alpha, \omega) := MO(\kappa(M, AC, LC), N, C)$ /* *Call the prediction model* */
28 $\quad\quad$ **if** $PP(\ell, \alpha) = 1$ **then**
29 $\quad\quad\quad maxAC := AC_i$ /* *Enough servers at* T_i*, search in* $minAC..AC_i$ */
30 $\quad\quad$ **else**
31 $\quad\quad\quad$ **if** $\alpha > \alpha_{max}$ **then**
32 $\quad\quad\quad\quad minAC := AC_i + 1$ /* *Too few servers, search in* $AC_i + 1..maxAC$ */
33 $\quad\quad\quad$ **else**
34 $\quad\quad\quad\quad$ /* $\ell > \ell_{max}$ *: too high client concurrency level, reduce* LC *to decrease* ℓ */
35 $\quad\quad\quad\quad$ **for** $j := 1 ; j \leq M ; j++$ **do**
36 $\quad\quad\quad\quad\quad minLC := 1 ;$
37 $\quad\quad\quad\quad\quad maxLC := LC_j$
38 $\quad\quad\quad\quad\quad$ **while** $minLC \neq maxLC$ **do**
39 $\quad\quad\quad\quad\quad\quad LC_j := minLC + \frac{maxLC - minLC}{2}$ /* *Dichotomic search to increase* LC_j*, and thus reduce* ℓ */
40 $\quad\quad\quad\quad\quad\quad (\ell, \alpha, \omega) := MO(\kappa(M, AC, LC), N, C)$ /* *Call the prediction model* */
41 $\quad\quad\quad\quad\quad\quad$ **if** $PP(\ell, \alpha) = 1$ **then**
42 $\quad\quad\quad\quad\quad\quad\quad maxLC := LC_j$ /* *Enough concurrency at* T_j*, search in* $minLC..LC_j$ */
43 $\quad\quad\quad\quad\quad\quad$ **else**
44 $\quad\quad\quad\quad\quad\quad\quad minLC := LC_j + 1$ /* *Low concurrency at* T_j*, search in* $LC_j + 1..maxLC$ */
45 $\quad\quad\quad\quad$ /* *End of dichotomic search on* LC *at* $T_1..T_M$ *for current value of* AC_i */
46 $\quad\quad\quad\quad$ **if** $PP(\ell, \alpha) = 1$ **then**
47 $\quad\quad\quad\quad\quad maxAC := AC_i$ /* *Enough concurrency and servers at* T_i*, search in* $minAC..AC_i$ */
48 $\quad\quad\quad\quad$ **else**
49 $\quad\quad\quad\quad\quad minAC := AC_i + 1$ /* *Maximum concurrency but too few servers at* T_i*, search in* $AC_i + 1..maxAC$ */

50 /* *End of dichotomic search on* AC *at* $T_1..T_M$ */

faible de AC_i dans la partie inférieure de l'intervalle, soit $[minAC..AC_i]$ (lignes 28 et 29). Dans le cas contraire, la nouvelle configuration ne respecte pas l'objectif de taux de rejet ou ne respecte pas l'objectif de latence. Dans le premier cas, trop peu de serveurs sont alloués au service, et la recherche se poursuit pour trouver une valeur plus élevée de AC_i dans l'intervalle $]AC_i..maxAC]$ (lignes 31 et 32). Autrement, si l'objectif de taux de rejet est respecté mais pas celui de latence, il peut y avoir deux causes. Soit le taux de multiprogrammation est trop élevé, ce qui augmente la latence. Dans ce cas, des valeurs plus faibles de $LC_1 \ldots LC_M$ sont calculées en utilisant une recherche dichotomique (lignes 35 à 44). Si la nouvelle valeur de la configuration locale permet de respecter le contrat de qualité de service, l'algorithme poursuit sa recherche d'une configuration architecturale inférieure (lignes 46 à 47). Dans le cas contraire, cela signifie que trop peu de serveurs sont alloués au service, et l'algorithme poursuit sa recherche d'une configuration architecturale supérieure (lignes 48 à 49).

La méthode de planification de capacité proposée a une complexité algorithmique donnée dans l'équation 6.3, où M est le nombre d'étages du service Internet multi-étagé, N est la quantité de charge du service, AC_{max} et LC_{max} sont les valeurs maximales respectives de la configuration architecturale et de la configuration locale, utilisées dans les boucles itératives de l'algorithme 2, aux lignes 21 et 38. On rappelle que la complexité du modèle est de $O(M.N)$.

$$O(M^2.N.\log_2(AC_{max} + M^3.N.\log_2(LC_{max})) \tag{6.3}$$

De plus, les coûts logarithmiques des opérations sur AC_{max} et LC_{max} sont dus aux recherches dichotomiques sur les configurations architecturales et locales. A titre de comparaison, la complexité algorithmique d'une recherche exhaustive de la configuration architecturale et locale optimale pour un service Internet multi-étagé est de $O(M^2.N.AC_{max}.LC_{max})$. Ainsi, la méthode de planification de capacité proposée dans cette thèse est plus efficace que la recherche exhaustive de plusieurs ordres de magnitude, dépendant de la taille du service Internet sous-jacent.

L'algorithme de planification de capacité est complété par une partie optionnelle présentée dans l'algorithme 3. Ceci est dû au fait que l'algorithme 2 peut calculer une configuration locale trop restrictive. En effet, la configuration architecturale et locale produite par l'algorithme 2 permet de respecter le contrat de qualité de service pour une valeur donnée de la quantité de charge N. Mais il se peut que pour une quantité de charge supérieure N', la configuration locale calculée pour N ne permette pas de garantir le SLA, bien que la configuration architecturale puisse être suffisante pour traiter cette charge supplémentaire. Ceci peut entraîner de futures reconfigurations supplémentaires du service pour traiter des charges plus importantes. L'algorithme 3 cherche à réduire les reconfigurations futures du service en calculant, sur la base du résultat de l'algorithme 2, la plus haute valeur de la configuration locale permettant de garantir le contrat de qualité de service pour N, mais également éventuellement pour des valeurs supérieures.

6.4 Principes de preuves

Cette section commence par décrire les hypothèse sous-jacentes aux services Internet multi-étagés, avant de présenter les principes des preuves d'optimalité et de terminaison de la méthode de planification de capacité proposée.

Algorithme 3: Complément pour planification de capacité

Input :
N : workload amount
$C(Z, V, S, D)$: workload mix
Output :
$\kappa(M, AC, LC)$: service configuration
1 **Parameters :**
2 M : *#tiers of the service*
3 ℓ_{max} : *SLA latency constraint*
4 α_{max} : *SLA abandon rate constraint*
5 $MO(\kappa, N, C) \rightarrow (\ell, \alpha, \omega)$: *service prediction model*

6 **/* Reduce future service reconfiguration and oscillations */**

7 $N' := N$
8 $guaranteedSLA := true$

9 **repeat**
10 $N' := N' + 1$ /* *Introduce a new client request in* $T_1..T_M$ */
11 **for** $i := 1; i \leq M; i++$ **do**
12 **if** $(i = 1)$ *or else* $(V_{i-1} > V_i)$ **then**
13 $LC_i := \frac{N' \cdot V_i}{AC_i}$ /* *A request on* T_{i-1} *induces at most one request on* T_i */
14 **else**
15 $LC_i := \frac{LC_{i-1} \cdot AC_{i-1}}{AC_i}$ /* *A request on* T_{i-1} *induces in total more than one request on* T_i */
16 /* *After introducing a new client request in* $T_1..T_M$ */
17 $(\ell, \alpha, \omega) := MO(\kappa(M, AC, LC), N', C)$ /* *Call the prediction model with increased client requests* */
18 **if** $PP(\ell, \alpha) = 0$ **then**
19 $guaranteedSLA := false$
20 **until** $guaranteedSLA = false$;

21 /* $guaranteedSLA = false$: *reduce admitted client requests in* $T_1..T_M$ */
22 $N' := N' - 1$
23 **for** $i := 1; i \leq M; i++$ **do**
24 **if** $(i = 1) \vee (i > 1 \wedge V_{i-1} > V_i)$ **then**
25 $LC_i := \frac{N' \cdot V_i}{AC_i}$ /* *A request on* T_{i-1} *induces at most one request on* T_i */
26 **else**
27 $LC_i := \frac{LC_{i-1} \cdot AC_{i-1}}{AC_i}$ /* *A request on* T_{i-1} *induces in total more than one request on* T_i */

Hypothèses

Nous présentons ci-dessous les hypothèses sur lesquelles se basent les principes de preuve d'optimalité et de terminaison de la planification de capacité. Ces hypothèses reflètent des propriétés générales observées sur les services Internet.

Hypothèse H1 : Les objectifs de qualité de service spécifiés dans le contrat de qualité de service sont réalistes, c'est-à-dire qu'il est possible d'atteindre les contraintes de latence et de taux de rejet avec suffisamment de serveurs alloués au service Internet.

Hypothèse H2 : Ajouter des serveurs à un service Internet multi-étagé ne dégrade pas les performances et la disponibilité de ce service. De plus, si un goulot d'étranglement est présent à un étage, ajouter suffisamment de serveurs sur cet étage permet, à terme, d'améliorer la latence et le taux de rejet de ce service, et finalement de supprimer le goulot d'étranglement de cet étage.

Hypothèse H3 : Augmenter le niveau de multiprogrammation (augmenter le MPL) d'un serveur peut augmenter la latence du service mais va réduire le taux de rejet de ce service. Diminuer le niveau de multiprogrammation permet de réduire la latence du service et augmenter le taux de rejet.

Optimalité de la planification de capacité

Une configuration optimale d'un service Internet pour une charge donnée est une configuration garantissant le contrat de qualité de service avec un coût minimal pour le service. Soit $\kappa^*(M, AC^*, LC^*)$ la configuration optimale d'un service Internet multi-étagé constitué de M étages. Ainsi :

$$PP(\kappa^*) = 1$$

Cette configuration κ^* existe toujours grâce à l'hypothèse H1 qui indique que le contrat de qualité de service est toujours atteignable. De plus, soit $\kappa(M, AC, LC)$ une configuration de ce service. On a :

$$\forall \kappa, PP(\kappa) = 1 \Longrightarrow \sum_{i=1}^{M} AC_i \geq \sum_{i=1}^{M} AC_i^*$$

Donc :

$$\forall \kappa, \theta(\kappa) \leq \theta(\kappa^*) \wedge \theta(\kappa^*) > 0$$

Dans la suite, nous montrons tout d'abord que la configuration produite par l'algorithme de planification de capacité respecte le contrat de qualité de service (SLA), puis nous montrons que cette configuration a un coût minimal. Étant donnée $\kappa(M, AC, LC)$ la configuration produite par l'algorithme de planification de capacité (algorithme 2). Supposons que κ ne garantisse pas le SLA. Premièrement, les lignes 6 à 17 de l'algorithme itèrent et augmentent la quantité de serveurs alloués au service Internet jusqu'à ce que le SLA soit respecté. En effet, grâce aux hypothèses H1 et H2, cette boucle va se terminer avec une configuration respectant le SLA à la ligne 17. Ensuite, supposons que le reste de l'algorithme (lignes 18 à 50) produise une configuration qui ne respecte pas le SLA. Ceci correspond à un des trois cas suivants : ligne 31, ligne 43 ou ligne 48. Dans le cas des lignes 31 et 48, le nombre de serveurs alloués au service est augmenté, ce qui va permettre à terme de respecter le SLA (hypothèses H1 et H2).

La ligne 43 correspond au cas où la contrainte de taux de rejet n'est pas respectée, induisant une augmentation du niveau de multiprogrammation des serveurs. Ceci va soit permettre de respecter les contraintes du SLA grâce à l'hypothèse H3 (cf. ligne 46), soit sera suivi par une augmentation du nombre de serveurs alloués au service, ce qui va permettre de respecter le SLA (cf. hypothèses H1 et H2, et ligne 48). Finalement, ceci contredit la supposition que la configuration produite par l'algorithme de planification de capacité ne respecte pas le SLA.

Supposons maintenant que la configuration $\kappa(M, AC, LC)$ produite par l'algorithme de planification de capacité, qui garantit le SLA, n'a pas un coût minimal, c'est-à-dire :

$$(PP(\kappa) = 1) \wedge \sum_{i=1}^{M} AC_i > \sum_{i=1}^{M} AC_i^* \tag{6.4}$$

Par définition, retirer un serveur de la configuration optimale d'un service entraînera le non respect du SLA et l'apparition d'un goulot d'étranglement à l'étage où le serveur a été retiré :

$$\forall \kappa, \exists i \in [1, M], AC_i < AC_i^* \Rightarrow PP(\kappa) = 0$$

Ainsi, si la configuration κ calculée par l'algorithme de planification de capacité n'a pas un coût minimal :

$$\exists i \in [1, M], PP(\kappa) = 1 \wedge AC_i > AC_i^*$$

Ceci signifie que dans l'algorithme 2, la recherche dichotomique sur AC_i a été effectuée sur l'intervalle haut ($]AC_i \ldots maxAC]$) plutôt que sur l'intervalle bas ($[minAC \ldots AC_i]$) Ceci correspond à un des deux cas suivants : la ligne 32 ou la ligne 49 de l'algorithme 2. Cependant, dans les deux cas, les contraintes de qualité de service ne sont pas respectées, ce qui contredit l'équation 6.4, et donc contredit la supposition que la configuration produite par l'algorithme de planification de capacité n'a pas un coût minimal.

Terminaison de la planification de capacité

L'algorithme du modèle donné dans l'algorithme 1 termine en $0(M.N)$ étapes de calcul. De plus, l'algorithme 2 décrivant la planification de capacité est constitué de 3 parties successives. La première partie (lignes 6 à 9) se termine après M étapes. La deuxième partie (lignes 9 à 17) itère jusqu'à trouver une configuration du service permettant de garantir les contraintes du SLA. En se basant sur les hypothèses H1 et H2, cette deuxième partie de l'algorithme se termine également après $O(M^2.N.\log_2(AC_{max}) + M^3.N.\log_2(LC_{max}))$ étapes de calcul. Ainsi, l'algorithme 2 de planification de capacité est garanti de se terminer.

6.5 Synthèse

Dans ce chapitre, nous avons présenté une solution pour la planification de capacité des services Internet soumis à des contraintes de qualité de service. Cette solution se base sur une fonction d'utilité pour quantifier et comparer les configurations, ainsi que sur un algorithme de planification de capacité permettant de trouver la configuration optimale d'une

application multi-étagée, en fonction d'objectifs de haut niveau concernant la qualité de service.

La mise en oeuvre concrète d'un contrôleur adaptatif de services Internet est présentée dans le chapitre suivant.

Chapitre 7

Contrôle adaptatif de services Internet

Nous proposons dans ce chapitre un contrôleur autonome pour la configuration automatique de services Internet. Ce chapitre présente l'architecture du contrôleur, ainsi que ses détails de mise en oeuvre dans le prototype logiciel MoKa.

7.1 Présentation générale

Nous avons entièrement conçu et mis en oeuvre un contrôleur logiciel nommé MOKA, pour la modélisation et la planification de capacité des services Internet multi-étagés [4]. Ce prototype intègre donc entre autres l'implantation du modèle analytique proposé dans le chapitre 5 et effectue le paramétrage en ligne automatique de ce modèle. MoKa comprend également la méthode de planification de capacité vue dans le chapitre 6. Par ailleurs, MOKA a été conçu comme un outil ouvert, capable d'intégrer différents modèles de performance, différentes méthodes de planification de capacité et de comparer leur comportement.

Le contrôleur que nous proposons est constitué des différentes parties suivantes :

- Une implantation de l'algorithme du *modèle analytique* et de la méthode de *planification de capacité* pour le calcul de la configuration optimale qui garantit le SLA et minimise le coût du service, étant donnés la quantité et le type de charge du service.
- Des *sondes de monitorage* permettent d'effectuer une observation en ligne et de fournir au contrôleur les informations nécessaires concernant la quantité et le type de charge évoluant dans le temps.
- Des *actionneurs de reconfiguration* permettent d'appliquer en ligne au service Internet les configurations locale et architecturale optimales calculées par le contrôleur.

Le fonctionnement du contrôleur est schématisé sur la figure 7.1. Le contrôleur collecte dynamiquement les informations de charge, de performance et de disponibilité fournies par le système de monitorage, et utilise ces informations pour calibrer automatiquement le modèle. L'algorithme de planification de capacité utilise ensuite le modèle pour calculer la configuration optimale. Cette dernière est appliquée en utilisant les actionneurs de reconfiguration. Les objectifs de qualité de service (latence maximum et taux de rejet maximum) sont spécifiés au contrôleur par l'administrateur.

Nous présentons, dans la suite, le système de monitorage, le paramétrage automatique du contrôle de services Internet, puis nous présentons le contrôleur et les actionneurs. La mise en oeuvre de la solution proposée est ensuite présentée, à travers la réalisation du contrôleur MoKa, dont nous présentons les principales caractéristiques.

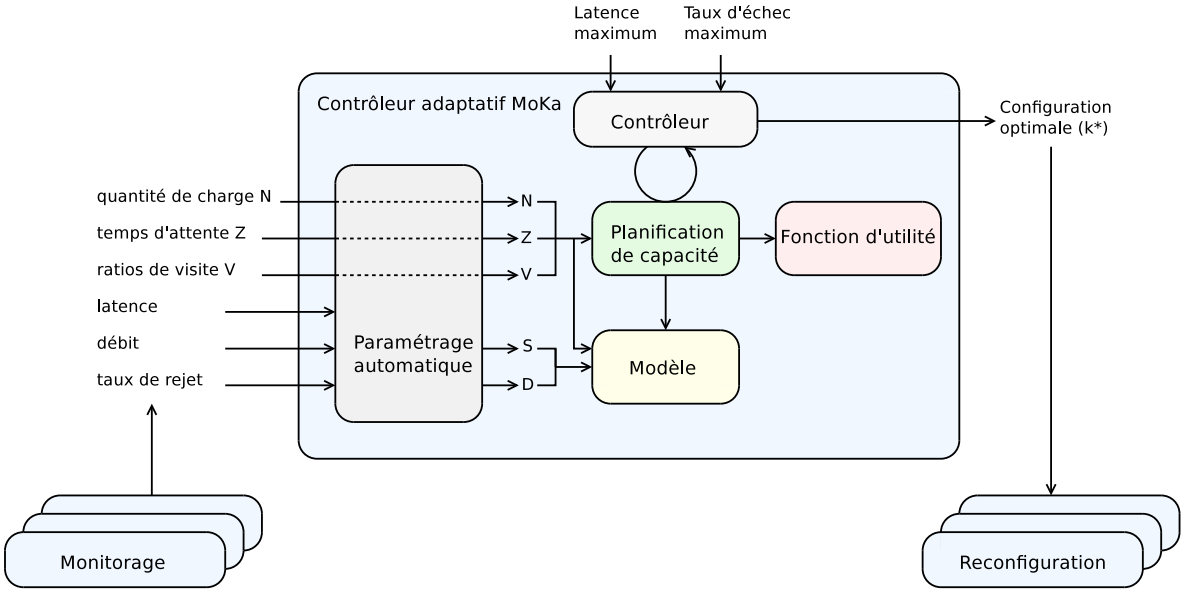

FIGURE 7.1 – Contrôleur adaptatif

7.2 Monitorage

Le système de monitorage est indispensable pour obtenir une connaissance complète et à jour du service Internet administré. Les informations à collecter concernent les entrées à fournir au modèle et à l'algorithme de planification de capacité.

7.2.1 Sondes de monitorage

Les entrées du modèle devant être monitorées sur le service Internet sont présentées dans la section 5.1.1. Elle comprennent la quantité et le type de charge. Pour le paramétrage automatique du modèle, la latence, le débit, le taux d'échec, ainsi que la configuration locale et architecturale du service Internet administré sont également requis.

Le monitorage en ligne du service Internet est effectué en utilisant des sondes qui mesurent périodiquement l'état du service et collectent des informations sur la quantité de charge N, le temps d'attente Z et les ratios de visite $V < V_1 \ldots V_M >$ du service Internet. Ensuite, les moyennes de ces valeurs sont calculées (voir section 7.2.2) et fournies en entrée du contrôle adaptatif, pour être utilisées par le modèle et la méthode de planification de capacité.

Les métriques mesurées sur chaque étage de l'application sont donc :

- la *latence* moyenne des requêtes
- le *débit* moyen de requêtes traitées
- le *taux de rejet* des requêtes traitées

Le premier étage de l'application est également instrumenté pour fournir, en plus des mesures précédentes :

- le *temps d'attente*, ou *think time* moyen des clients
- la *quantité de charge* accédant à l'application

L'ensemble de ces mesures permet ensuite de définir la quantité et le type de la charge actuelle, ainsi que la performance et la disponibilité du service Internet administré.

Latence. La latence globale ℓ du service est mesurée sur le premier étage du service Internet. Pour chaque requête client, on stocke sa date d'arrivée et la date de fin de son traitement. La différence entre les deux permet d'obtenir la latence de cette requête, ℓ étant égal à la moyenne de ces latences. La latence sur chaque étage du service contrôlé est également mesurée.

Débit. Le débit global X du service est également mesuré sur le premier étage du service Internet. Il correspond au nombre de requêtes traitées par unité de temps. Le débit de chaque étage du service contrôlé est également mesuré.

Taux de rejet. Le taux de rejet α du service est également mesuré sur le premier étage de l'application. Il correspond au ratio entre le débit de requêtes rejetées par le contrôle d'admission, et le débit total de requêtes, comme représenté par l'équation 7.1.

$$\alpha = \frac{X_{err}}{X + X_{err}} \tag{7.1}$$

Le taux de rejet sur chaque étage du service contrôlé est également mesuré.

Quantité de charge. La quantité de charge accédant à un service Internet est mesurée sur le premier étage du service.

Temps d'attente. Le temps d'attente Z des clients est mesuré par le proxy à l'entrée du premier étage du service Internet. Comme représenté sur la figure 7.2, les clients attendent la réception de la réponse à leur requête avant d'envoyer la requête suivante. De plus, après la réception d'une réponse, les clients ont besoin d'un certain temps de réflexion pour décider de la prochaine action (*think time)*. En l'absence de surcharge, la valeur de ce paramètre influe sur la quantité de requêtes envoyées au serveur par un nombre donné de clients.

Ratios de visite. Les ratios de visite *(visit ratio)* expriment le nombre moyen de requêtes induites sur chacun des étages par une requête client [18]. Comme expliqué dans la section 2.1, le traitement d'une requête sur un étage peut entraîner un nombre variable de sous-requêtes vers les étages suivants. V_i correspond donc au nombre moyen de requêtes généré sur l'étage T_i par une requête client accédant au service. Le ratio de visite V_i à l'étage T_i est le rapport entre le débit X_i sur l'étage T_i et le débit de requête client X_1. Indépendamment du système considéré, V_1 sera toujours égal à 1, car les requêtes reçues sur cet étage sont les requêtes des clients. L'équation 7.2 présente le calcul des ratios de visite pour une quantité de charge non nulle ($X_1 > 0$).

$$\forall i \in [1, M], V_i = \frac{X_i}{X_1} \tag{7.2}$$

Les valeurs des *temps de service* (S_i) et des *délais de transmission* (D_i) sont calculées automatiquement, comme présenté dans la section 7.3.

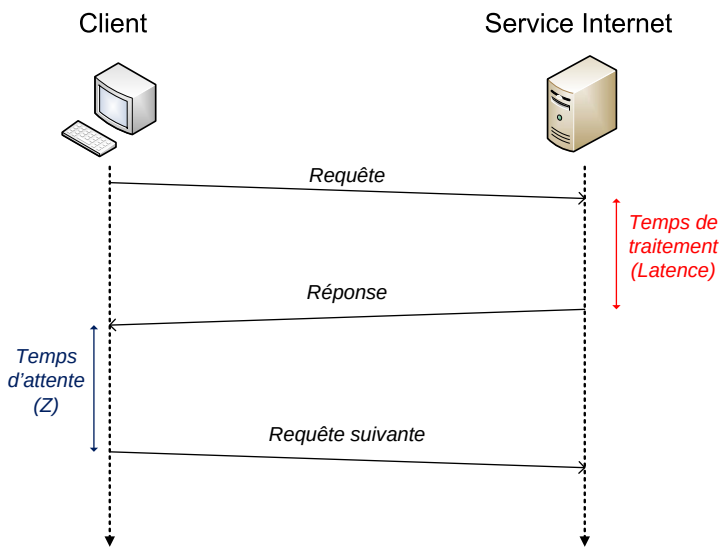

FIGURE 7.2 – Interaction client/serveur et *think time*

7.2.2 Monitorage global

Fréquence de monitorage

Les sondes de monitorage fournissent les informations collectées à intervalle régulier. La valeur de cet intervalle paramétrable est fixé par défaut à 10 secondes, ce qui constitue un bon compromis entre réactivité du système de monitorage et limitation des communications sur le réseau.

Fenêtre de mesure

Les valeurs renvoyées par les sondes de monitorage sont agrégées et fournissent une estimation de la valeur moyenne monitorée. En effet, la nature des requêtes émises par les clients peut varier au sein d'un même type de charge. La valeur instantanée d'une valeur mesurée sur le système, par exemple la latence, peut ne pas correspondre à la latence moyenne réellement observée par la majorité des clients. Lors de la collecte des mesures, nous utilisons donc une fenêtre glissante pour effectuer cette moyenne. Cette fenêtre glissante est de plus pondérée de manière exponentielle pour donner plus de poids aux données récentes. Nous utilisons une moyenne glissante exponentielle, ou *EWMA (Exponentially Weighted Moving Average)*, permettant de rendre le système de monitorage plus précis dans les mesures effectuées, tout en conservant une bonne réactivité aux changements de valeurs moyennes [11]. La durée de la fenêtre de mesure est paramétrable, avec une valeur par défaut de 60 secondes.

7.3 Paramétrage interne

L'objectif est de calculer les valeurs des paramètres S_i et D_i donnant au modèle la meilleure précision. Pour cela, nous calculons les écarts entre les valeurs de latence, de débit et de taux d'échec mesurées sur le système réel, et celles produites par le modèle. On cherche à obtenir les paramètres internes (S_i et D_i) du modèle minimisant les écarts pour chacune de ces métriques. La figure 7.3 schématise l'approche décrite.

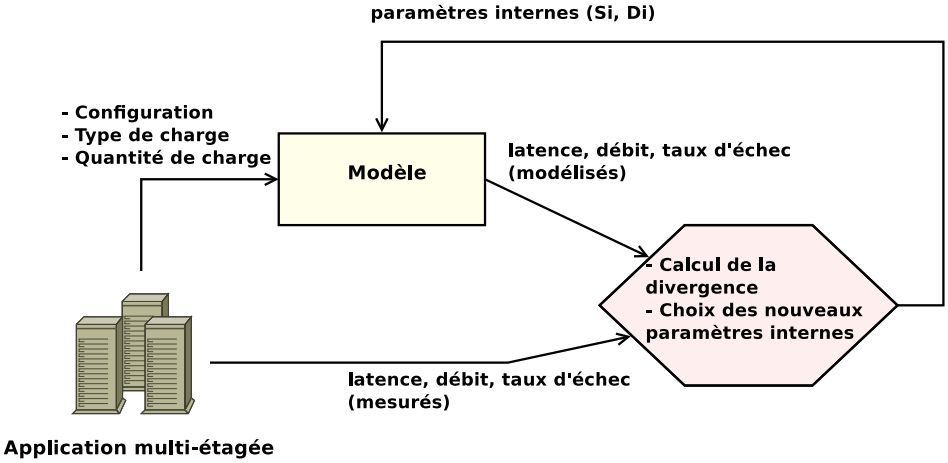

FIGURE 7.3 – Recherche des paramètres internes optimaux du modèle

Cette recherche s'effectue en ligne, de manière dynamique et complètement automatisée, pour que les paramètres du modèle correspondent toujours au type de charge supporté par le service Internet modélisé. Il faut noter également que l'espace de recherche est important, car les valeurs des paramètres doivent être obtenues avec précision, et ces paramètres sont définies dans l'espace des réels positifs ($\mathbb{R}+$). Enfin, le nombre de ces paramètres augmente avec le nombre d'étages de l'application, car pour chaque étage T_i de l'application il faut trouver les valeurs des paramètres S_i et D_i correspondant. On a vu cependant que D_1 est toujours égal à 0 quel que soit le service Internet considéré. Le nombre de paramètres à trouver est donc égal à $2 \times M - 1$. Un calcul exhaustif des prédictions du modèle pour l'ensemble des valeurs de l'espace de paramétrage est donc inenvisageable.

Calcul de S_i et D_i

Nous utilisons pour ce calcul une méthode automatisée basée sur la technique de la descente de gradient. La descente de gradient est une méthode d'optimisation permettant de trouver le minimum local d'une fonction. Ici la fonction concernée est la fonction donnant l'écart entre les prédictions du modèle et les mesures sur le système réel, présentée dans l'équation 7.3. Dans cette fonction, ℓ_{mod}, τ_{mod} et α_{mod} correspondent respectivement à la latence, au débit et au taux de rejet prédits par le modèle, alors que ℓ, τ et α sont respectivement les valeurs de latence, débit et taux de rejet mesurés sur l'application. L'écart obtenu est exprimé en pourcentage moyen.

$$f_{div}(\ell_{mod}, \tau_{mod}, \alpha_{mod}) = \frac{1}{3} \times \left(\frac{|\ell_{mod} - \ell| * 100}{\ell} + \frac{|\tau_{mod} - \tau| * 100}{\tau} + \frac{|\alpha_{mod} - \alpha| * 100}{\alpha} \right) \quad (7.3)$$

La technique par descente de gradient consiste à faire évoluer par étape la valeur des paramètres internes du modèle. A chaque étape, on fait évoluer le paramétrage dans le sens inverse du gradient de la fonction f_{div} au point courant. Autrement dit, on cherche la modification des paramètres qui donnera la plus forte diminution de l'écart moyen. La valeur de cette modification peut être soit un pas fixe, soit variable. Nous utilisons un pas variable afin de converger plus rapidement vers le paramétrage optimal, et éviter ensuite les phénomènes d'oscillations autour de cet optimal, qui peuvent subvenir si la modification de paramétrage menant vers la valeur optimale est plus petite que le pas d'adaptation standard.

L'algorithme se termine après un certain nombre d'itérations, définies par le paramètres $step_size$. L'algorithme 4 présente l'algorithme de descente de gradient pour le paramétrage automatique du modèle. L'algorithme prend en entrée la configuration du service, la quantité de charge N, le temps d'attente Z et les ratios de visite V. La ligne 1 initialise nb_steps, le nombre d'itérations de l'algorithme de paramétrage automatique, à une valeur constante. La valeur initiale de l'ajustement des paramètres à chaque itération est notée, $step_size$, et initialisée à une valeur constante à la ligne 2. Des lignes 3 à 6, l'algorithme commence par initialiser les temps de service et les délais à des valeurs par défaut. Le délai D_1 est initialisé à 1, sa valeur ne changeant pas par la suite. L'algorithme itère ensuite (lignes 8 à 43) pour trouver la valeur optimale des paramètres. A chaque itération, l'algorithme teste les effets d'une modification de chacun des paramètres (lignes 10 à 36), sauf D_1 qui est constant. Pour chacun de ces paramètres, on évalue les effets de l'ajout du pas d'évolution $evoStep$ ainsi que du retrait de ce pas d'évolution (lignes 12 à 36). Dans le cas où la valeur d'un paramètre deviendrait négative, on n'effectue pas le test concerné (lignes 18, 19, 22 et 23). En effet un temps de traitement ou un délai de transmission ne peut être inférieur à 0. La fonction f_{div} est utilisée pour calculer la divergence des prédictions du modèle avec les mesures effectuées sur le système contrôlé.

On calibre ensuite le modèle pour chaque évolution des paramètres, afin de conserver le paramétrage qui minimise l'écart entre les prédictions du modèle et les mesures sur le système réel (lignes 24 à 36). La modification de paramétrage donnant le meilleur gain est conservée pour les itérations suivantes (lignes 37 à 40). De plus, si aucune amélioration n'est possible, le pas d'évolution est réduit pour gagner en précision (lignes 41 à 43). L'algorithme se poursuit ensuite avec l'itération suivante. A la sortie de l'algorithme, les paramètres optimaux S et D sont renvoyés.

Toutefois, pour que cette méthode puisse être utilisée dans notre cadre applicatif, il faut s'assurer que tout optimum local de la fonction f_{div} soit également un optimum global. Cette vérification est importante pour s'assurer que la recherche ne renvoie pas de valeurs qui ne correspondent pas aux paramètres optimaux du modèle. Nous avons effectué une phase préliminaire d'expérimentation afin de vérifier que la fonction f_{div} ne comporte pas d'optimum local qui ne soit pas un optimum global. La figure 7.4 présente l'écart entre prédiction du modèle et mesure réelle en fonction des paramètres internes S_1 et S_2, pour un service Internet composé de deux étages. Cette figure permet d'obtenir une visualisation de l'espace de recherche parcouru par l'algorithme 4. Les zones de la figure ayant la plus faible valeur correspondent donc au paramétrage optimal du modèle. Comme on peut le voir, il n'existe pas d'optimum local différent d'un optimum global. Malgré des variations dans la configuration du service et la quantité ou le type de charge pris en compte, les expériences montrent que la conformation de l'espace de recherche reste identique, sans optimums locaux. Ces expériences confortent donc l'idée que l'algorithme 4 va évoluer vers la zone d'optimalité.

Algorithme 4: Algorithme de paramétrage automatique du modèle

```
Input :
κ(M, AC, LC) : service configuration
N : workload amount
Z think time
V : visit ratios
MO(κ, N, C) → (ℓ, τ, α) : service prediction model
Output :
S : service times ; D : delays between tiers
1  nb_steps = VALUE1 ;   /* research iterations */
2  step_size = VALUE2 ;  /* parameter adjustment value */
3  for i = 0; i < M; i++ do
4      bestS_i = step_size;
5      bestD_i = step_size;
6  bestD_1 = 1;
7  evoStep = step_size;
8  for step = 0; step < nb_steps; step++ do
9      /* Evolution for each space dimensions */
10     for dim = 0; dim < (2 * M − 1); dim++ do
11         /* Evolution direction in this dimension */
12         for evo = −evoStep; evo <= evoStep; evo+ = 2 * evoStep do
13             for i = 0; i < M; i++ do
14                 S_i = bestS_i;
15                 D_i = bestD_i;
16             if dim<M then
17                 S_dim = S_dim + evo;
18                 if S_dim < 0 then
19                     continue ; /* Limit research space to ℝ+ */
20             else
21                 D_(dim−M+1) = D_(dim−M+1) + evo;
22                 if D_dim < 0 then
23                     continue ; /* Limit research space to ℝ+ */
24             /* Limit research space to ℝ+ */
25             /* Call the prediction model for the new configuration */
26             C(Z, V, S, D);
27             (ℓ_mod, τ_mod, α_mod) := MO(κ(M, AC, LC), N, C);
28             /* Call the prediction model for the best configuration */
29             bestC(Z, V, bestS, bestD);
30             (ℓ_modbest, τ_modbest, α_modbest) := MO(κ(M, AC, LC), N, bestC);
31             /* Test if the deviation is reduced */
32             gain = f_div(ℓ_modbest, τ_modbest, α_modbest) − f_div(ℓ_mod, τ_mod, α_mod);
33             if gain > bestGain then
34                 bestGain = gain;
35                 bestDim = dim;
36                 bestEvo = evo;
37     if bestDim < M then
38         bestS_(bestDim) = bestS_(bestDim) + bestEvo;
39     else
40         bestD_(bestDim−M+1) = bestD_(bestDim−M+1) + bestEvo;
41     /* Avoid oscillations around optimal value */
42     if bestGain <= 0 then
43         evoStep = evoStep/2;
44 S = bestS;
45 D = bestD;
```

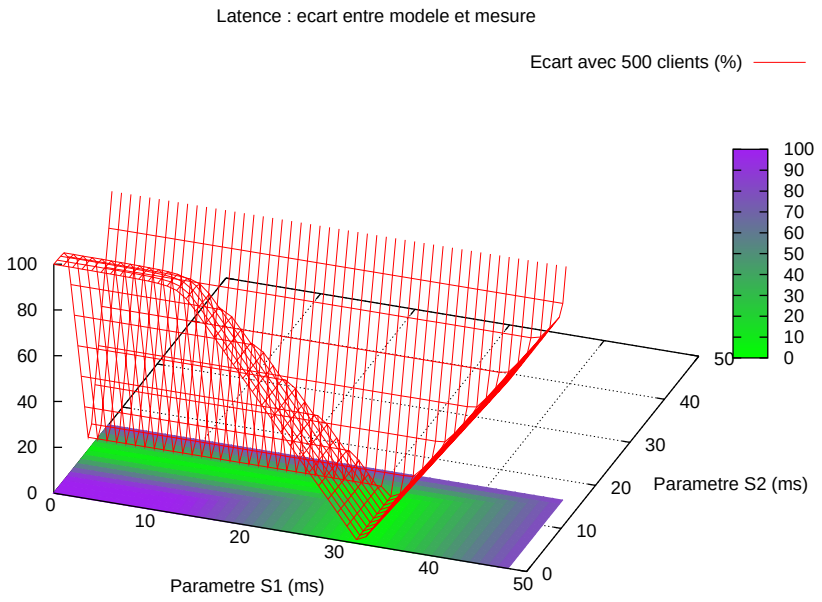

FIGURE 7.4 – Divergence du modèle en fonction des variables internes

Stabilité du paramétrage interne

Afin d'obtenir un modèle précis, il faut s'assurer que les paramètres du modèle produits par l'algorithme 4 restent les mêmes pour un type de charge donné. En effet, ces paramètres dépendent uniquement du type de charge. Ils doivent donc rester constants même si les autres paramètres du modèles évoluent, comme par exemple la quantité de charge N. L'expérience suivante présente l'évolution des valeurs des paramètres S_i dans un modèle auto-calibré pour un système soumis à la fois à des variations de type de charge et de quantité de charge. Le service Internet de vente en ligne TPC-W a été utilisé pour cette expérience, avec une configuration $\kappa < M = 2, AC < 1, 1 >, LC < 500, 500 >>$. La figure 7.5 présente l'évolution de la quantité de charge, suivant des paliers et des pics de charge. Le type de charge change au milieu de l'expérience (à 3000s). Le temps d'attente des clients étant constant à $Z = 7000$ ms pour cette expérience, seul les ratios de visite et les paramètres à calculer automatiquement (S_i et D_i) sont modifiés parmi les caractéristiques du type de charge. L'évolution des ratios de visite V_i est représentée sur la figure 7.6. Par définition, le ratio de visite sur le premier étage est toujours constant à 1. Le ratio de visite sur le deuxième étage évolue lors du changement de type de charge, car les ratios de visite sont une caractéristique du type de charge. L'instabilité dans la mesure de V_2 est due au pic de charge survenant à 600s (la quantité de charge passe de 250 à 1250 clients), entraînant une surcharge temporaire du premier étage pouvant ne pas se propager immédiatement sur le deuxième étage. Les ratios de visite étant définis par les rapports entre les débits des étages, la baisse du débit sur le premier étage entraîne une hausse temporaire du ratio de visite V_2.

La figure 7.7 présente l'évolution des paramètres S_i calculés automatiquement par le modèle en fonction de son contexte courant. Comme on peut le voir, les valeurs des paramètres internes calculées par le modèle restent stabilisées autour d'une valeur constante pour chaque type de charge.

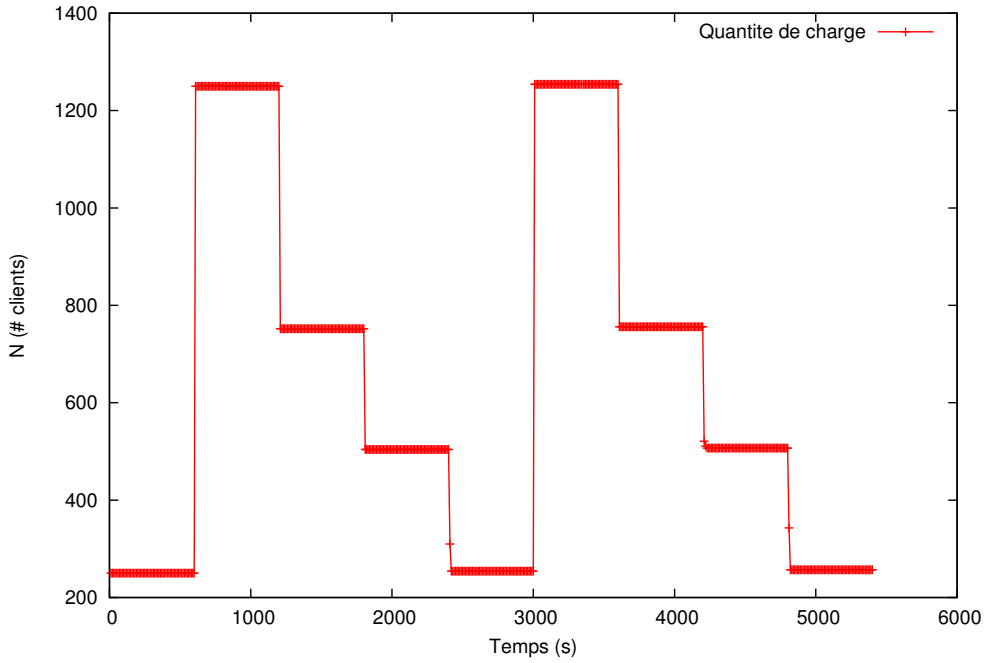

FIGURE 7.5 – Évolution de la quantité de charge

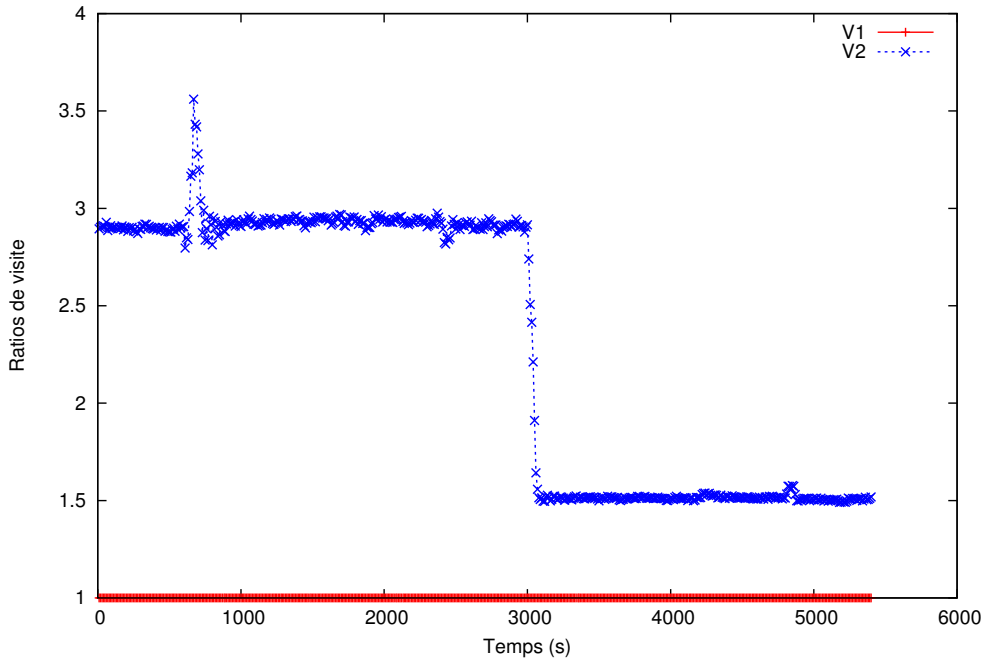

FIGURE 7.6 – Évolution des ratios de visite

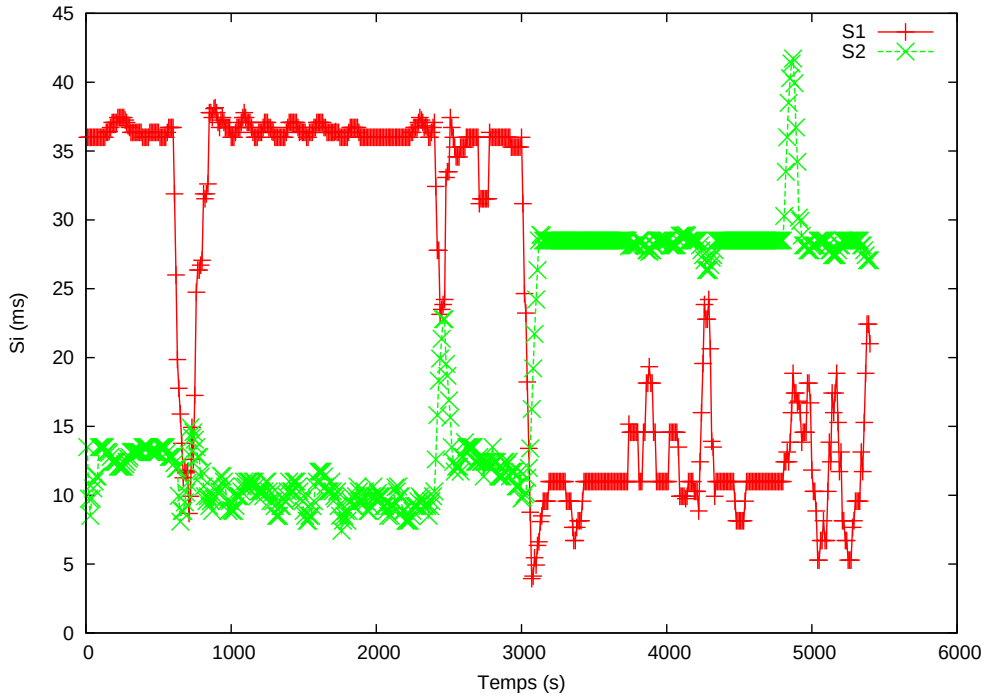

FIGURE 7.7 – Évolution des paramètres S_i

La figure 7.8 permet de comparer la latence du système réel avec la latence prédite par le modèle. On constate que malgré les variations de quantité et de type de charge, le paramétrage automatique du modèle lui permet de rester précis. La figure 7.9 compare le débit prédit et celui mesuré. Le débit prédit ne dévie pas de la valeur mesurée malgré les variations de charge. La figure 7.10 compare enfin le taux de rejet mesuré sur le service et celui prédit par le modèle. Comme on peut le voir, les prédictions du modèle sont correctes pour chacun des critères de performance et de disponibilité produits par le modèle. Une évaluation plus détaillée de la précision du modèle est effectuée dans le chapitre 8.

7.4 Contrôleur

Le monitorage en ligne et le paramétrage interne permettent de calibrer automatiquement le modèle de service Internet proposé ainsi que les méthodes de planification de capacité avec leurs valeurs d'entrée, sans nécessiter de calibrage manuel ou de profilage de la part de l'administrateur (voir figures 5.1 et 6.1). Ainsi, la méthode de planification de capacité est appelée à intervalles réguliers par le contrôleur pour calculer la configuration optimale κ^* pour la quantité de charge actuelle N et le type de charge actuel $C(Z, V, S, D)$. Cette configuration optimale respecte les objectifs de performance et de disponibilité du SLA tout en minimisant le coût du service Internet. Le contrôleur aurait pu être conçu pour seulement réagir en cas de non respect du contrat de qualité de service, entraînant le calcul d'une nouvelle configuration optimale (au lieu de le faire périodiquement). Cependant, en cas de baisse de charge, ceci peut mener à des situations où le SLA est garanti, mais avec trop de serveurs alloués au service Internet, et donc un coût non minimal pour le service.

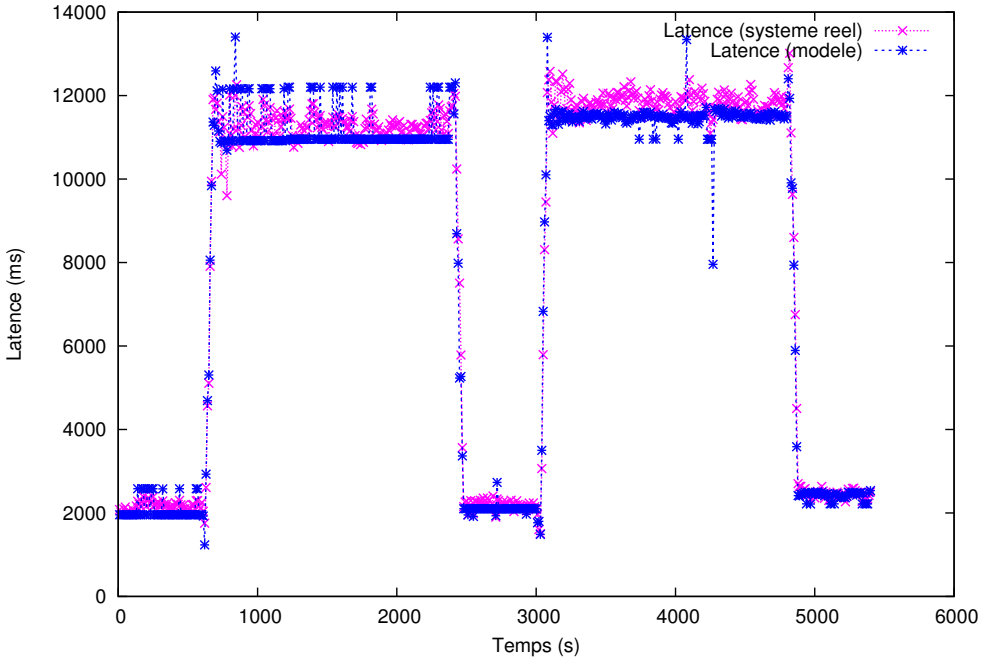

FIGURE 7.8 – Latence prédite et latence mesurée

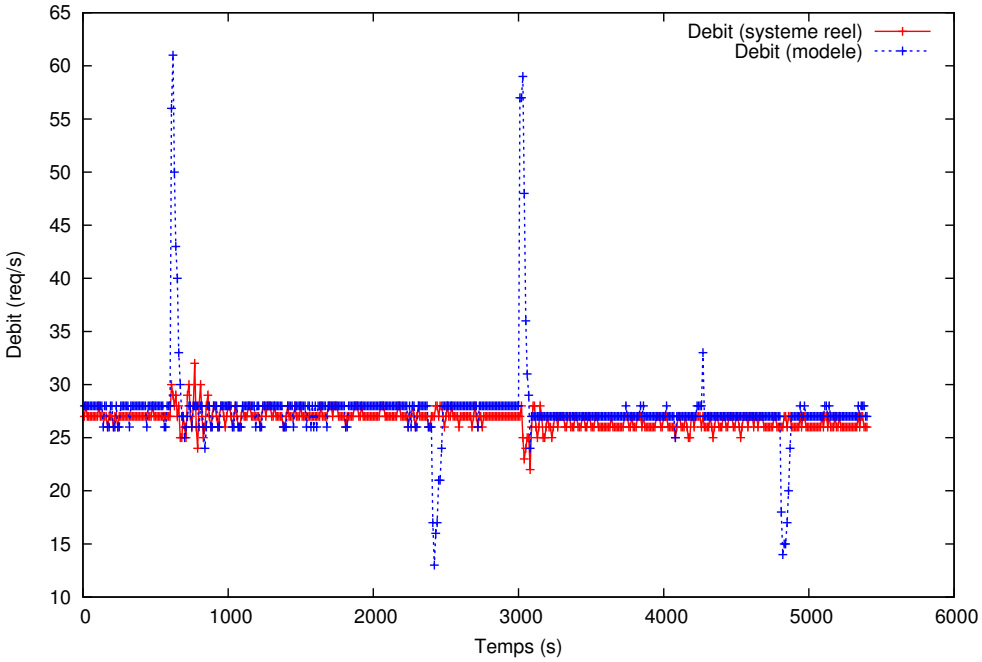

FIGURE 7.9 – Débit prédit et débit mesuré

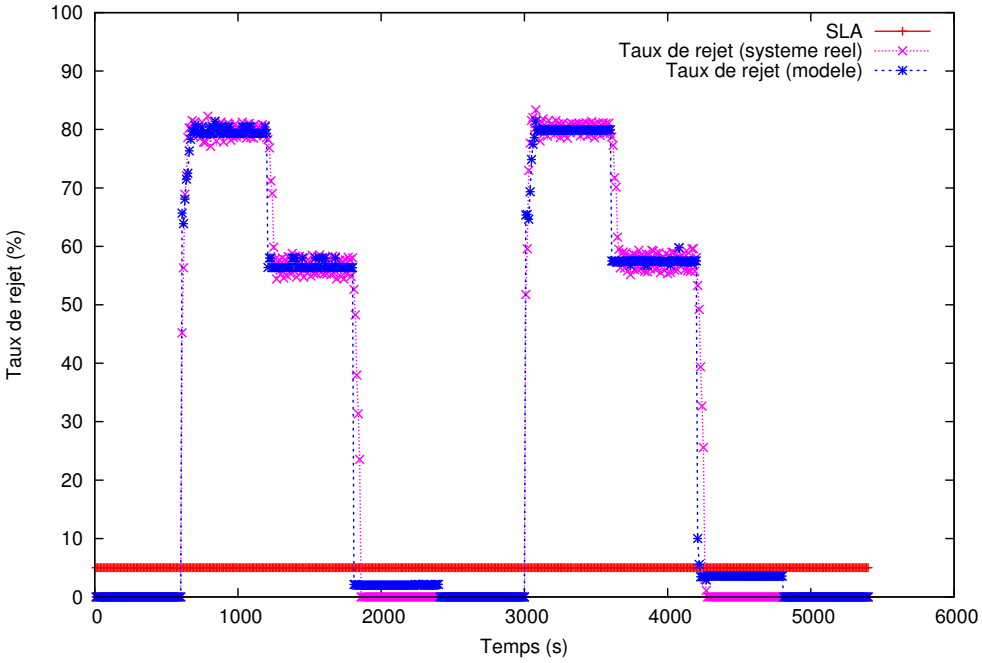

FIGURE 7.10 – Taux de rejet prédit et taux de rejet mesuré

Enfin, la nouvelle configuration κ^* est comparée avec la configuration courante du service Internet. Les différences de configuration locale ou architecturale déclenchent une reconfiguration du service Internet, telle que l'allocation ou le retrait d'un serveur pour appliquer la configuration architecturale, et l'ajustement dynamique du MPL pour l'application de la configuration locale.

7.5 Reconfiguration

Une fois la configuration optimale calculée par l'algorithme de planification de capacité, des actionneurs sont responsables de l'application de cette configuration optimale au service Internet administré. Les actionneurs sont capables de reconfigurer le service afin de modifier une partie seulement ou l'intégralité des configurations locales et architecturales de ce service. Les actionneurs agissent indépendamment les uns des autres. Ainsi, lors d'une reconfiguration, chaque actionneur reçoit la nouvelle valeur du paramètre de configuration qu'il a à appliquer. Le contrôleur de MoKa est cependant chargé de coordonner l'ensemble des actionneurs lors de l'application d'une reconfiguration, comme illustré sur la figure 7.15.

Configuration locale

La configuration locale est modifiée au niveau des proxies se trouvant devant chaque étage, le contrôle d'admission étant effectué à ce niveau. Chaque proxy expose donc une méthode permettant de définir la configuration locale des serveurs de l'étage, tous les serveurs sur un étage ayant la même configuration locale. Lors d'une modification de la configuration locale, le contrôle d'admission est dynamiquement modifié afin de s'assurer que le nombre

de requêtes autorisées à s'exécuter sur chaque serveur ne dépasse pas la nouvelle valeur. Dans le cas où le système compte un nombre de requêtes en cours d'exécution supérieur à la nouvelle valeur de configuration locale, aucune nouvelle requête n'est admise avant la fin du traitement des précédentes.

Configuration architecturale

Une modification de la configuration architecturale de l'application nécessite d'ajouter ou de retirer des serveurs au service Internet en cours de fonctionnement, et ce pour chacun des étages.

Dans le cas d'une augmentation de la configuration architecturale sur un étage, les opérations effectuées donc :

- l'allocation de serveurs depuis une liste de serveurs disponibles
- le lancement d'un serveur applicatif
- la mise à jour de la politique de distribution de charge de l'étage pour prendre en compte le nouveau serveur

Dans le cas d'une diminution de la configuration architecturale, les opérations sont similaires, mais dans l'ordre inverse. On effectue dans l'ordre :

- la mise à jour de la politique de distribution de charge de l'étage pour retirer un serveur
- l'arrêt du serveur applicatif
- l'ajout du serveur à la liste des machines disponibles

Lors de l'ajout ou de la suppression d'un serveur sur un étage, les opérations listées ci-dessus doivent être exécutées de manière synchrone, afin d'éviter des erreurs dans le traitement des requêtes client.

7.6 MoKa : prototype de contrôleur adaptatif de services Internet

7.6.1 Logiciel MoKa

Le prototype MoKa est organisé en différents paquets, dont les principaux sont représentés sur la figure 7.11.

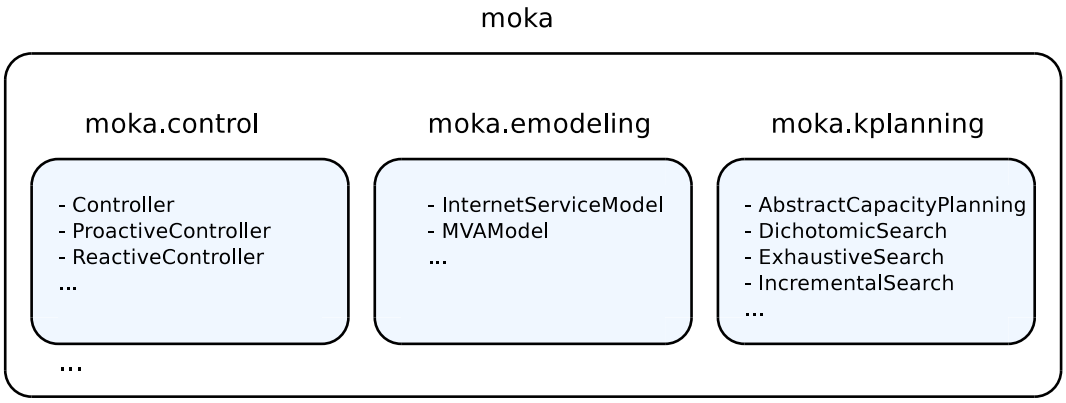

FIGURE 7.11 – Principaux paquets de MoKa

- *moka.control* contient les politiques de déclenchement du contrôleur MoKa.

- *moka.emodeling* contient les modèles de service Internet utilisé par le contrôleur, notamment le modèle présenté au chapitre 5.
- *moka.kplanning* contient les algorithmes de planification de capacité pouvant être utilisés par Moka. Ce paquet contient la planification de capacité présentée dans le chapitre 6, ainsi qu'un algorithme de recherche exhaustive.

Un modèle hérite de l'interface *InternetServiceModel*. La figure 7.12 présente cette interface. L'interface *InternetServiceModel* définit une seule méthode, *calculate*, calculant la latence, le débit et le taux de rejet du service Internet en fonction de la configuration, du type de charge et de la quantité de charge passés en paramètres.

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

moka.emodeling

Interface InternetServiceModel

All Known Implementing Classes:
MVAModel

```
public interface InternetServiceModel
```

Interface for a model that predicts the quality-of-service of an Internet service

Author:
Jean Arnaud, Sara Bouchenak

Method Summary	
InternetServiceQoS	**calculate**(GlobalConfiguration configuration, int workloadAmount, WorkloadMix workloadMix) Interface for calculating the quality of the Internet service (e.g. performance, availability, cost, etc.) according to underlying system configuration, workload characteristics and workload amount

FIGURE 7.12 – Interface de programmation de modèle de services Internet

La planification de capacité hérite de l'interface *CapacityPlanningSearch*. La figure 7.13 présente cette interface. Les méthodes de l'interface *CapacityPlanningSearch* sont les suivantes :

- *calculate* renvoie une configuration optimale pour le service Internet, en fonction d'une quantité de charge, d'un type de charge et de contraintes de qualité de service.
- les méthodes *getCost* permettent d'évaluer automatiquement le temps de calcul nécessaire à l'algorithme de planification de capacité, et ainsi de le comparer à d'autres algorithmes.
- *setInternetServiceModel* indique le modèle de service Internet à utiliser par la planification de capacité.
- *setWorkloadProperties* et *setProperties* mettent à jour les informations concernant la quantité de charge, le type de charge et les contraintes de qualité de service.

L'interface *controller* est présentée sur la figure 7.14. Elle définit les méthodes suivantes :

- *start* et *stop* permettent respectivement de démarrer et d'arrêter le fonctionnement du contrôleur.
- *isStarted* indique si le contrôleur est actuellement démarré.
- *setCapacityPlanning* définit la méthode de planification de capacité à utiliser.
- *setInternetServiceModel* définit le modèle de service Internet à utiliser.
- *setQoSConstraints* permet de spécifier les contraintes de qualité de service (SLA) à respecter.

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

moka.kplanning

Interface CapacityPlanning

All Known Implementing Classes:
AbstractCapacityPlanning, DichotomicSearch, ExhaustiveSearch, IncrementalSearch

public interface **CapacityPlanning**

Interface for an Internet service capacity planning method

Author:
Jean Arnaud, Sara Bouchenak

Method Summary

GlobalConfiguration	**calculate**() Calculates and returns an optimized configuration for an Internet service, according to current constraints
GlobalConfiguration	**calculate**(int workloadAmount, WorkloadMix workloadMix, int M, InternetServiceQoS constraints) Calculates and returns an optimized configuration for an Internet service, according to given constraints
double	**getCost**() Evaluate the cost of the underlying capacity planning algorithm.
double	**getCost**(int minWorkloadAmount, int maxWorkloadAmount, WorkloadMix workloadMix, int M, InternetServiceQoS constraints) Calculate the cost of the underlying capacity planning algorithm, in average, for different workload amounts.
double	**getCost**(int workloadAmount, WorkloadMix workloadMix, int M, InternetServiceQoS constraints) Evaluate the cost of the underlying capacity planning algorithm.
void	**setInternetServiceModel**(InternetServiceModel model) Associates an Internet service model with this capacity planning search algorithm.
void	**setProperties**(int workloadAmount, WorkloadMix workloadMix, int M, InternetServiceQoS constraints) Assigns workload amount, workload mix, the amount of tiers and the QoS constraints to the underlying Internet service
void	**setWorkloadProperties**(WorkloadMix workloadMix, InternetServiceQoS constraints) Assigns workload and performance characteristics to the underlying Internet service.

FIGURE 7.13 – Javadoc de l'interface pour les algorithmes de planification de capacité

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

moka.control

Interface Controller

All Known Implementing Classes:
ProactiveController, ReactiveController

public interface **Controller**

Interface for a controller of an Internet service

Author:
Jean Arnaud, Sara Bouchenak

Method Summary

boolean	**isStarted**() Indicates if the controller is started
void	**setCapacityPlanning**(CapacityPlanning kplan) Define the capacity planning to use
void	**setInternetServiceModel**(InternetServiceModel model) Define the Internet service model to use
void	**setQoSConstraints**(InternetServiceQoS constraints) Define the QoS constraints to match
void	**start**() Start the controller
void	**stop**() Stop the controller

FIGURE 7.14 – Javadoc de l'interface du contrôleur

7.6.2 Détails de mise en oeuvre

Pour des raisons de facilité d'utilisation et de sécurité, les systèmes de monitorage et de reconfiguration doivent être les moins invasifs possibles. Idéalement, le service Internet contrôlé ne doit pas être modifié pour intégrer ces éléments. Les systèmes de monitorage et de reconfiguration sont donc basés sur des proxies afin d'éviter la modification des services sous-jacents.

La figure 7.15 représente l'utilisation de MoKa sur un service multi-étagé à 3 étages, avec une configuration architecturale $AC =< 3, 2, 4 >$. Les implantations du modèle analytique et de l'algorithme de planification de capacité sont intégrées au contrôleur MoKa, qui fonctionne sur un serveur dédié. Des proxies positionnés sur chaque étage de l'application embarquent les sondes de monitorage et les actionneurs de reconfiguration. Les proxies sont positionnés sur chacun des serveurs du service Internet. Ils interceptent les sous-requêtes émises vers les étages suivants. Les serveurs du premier étage du service comportent également un proxy en entrée, afin de mesurer certaines caractéristiques de la charge du service (temps d'attente et quantité de charge). Le contrôleur MoKa communique avec chaque proxy pour collecter les informations de monitorage et transmettre les opérations de reconfiguration à effectuer.

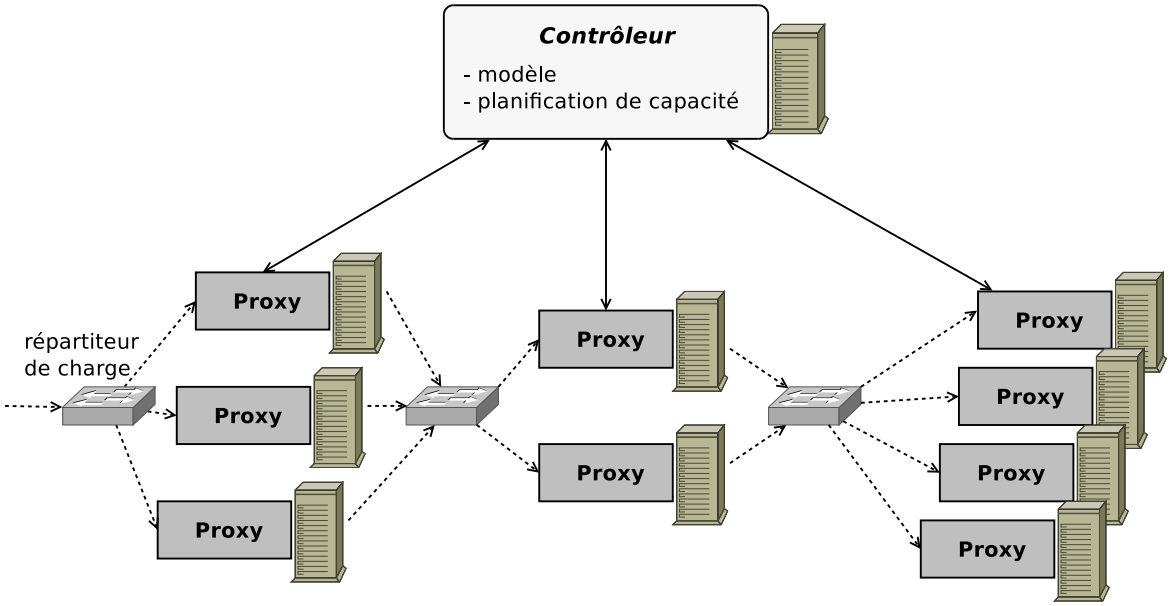

FIGURE 7.15 – Architecture du contrôleur

JMX

L'ensemble des informations collectées par le système de monitorage sont transférées au contrôleur MoKa en utilisant JMX *(Java Management Extensions)*, une technologie standardisée basée sur Java. JMX permet de collecter des informations sur une application distante, ainsi que de modifier dynamiquement son fonctionnement. Une application JMX se compose de 3 niveaux, illustrés sur la figure 7.16.

– Le niveau *instrumentation* contient l'ensemble des ressources instrumentées. Ces ressources sont appelées *MBean (Managed Bean)*, un type particulier de JavaBean. Bien qu'il existe plusieurs types de MBean, nous utilisons ici uniquement des MBeans standard, dont les fonctionnalités suffisent pour notre approche.
– Au niveau *agent*, on définit le serveur de MBean. Les MBean correspondant aux ressources instrumentées sont tous enregistrés dans ce serveur. Ce dernier contrôle les MBean et les expose aux divers agents d'administration à distance.
– Le niveau *administration* englobe tous les clients accédant à distance aux attributs et méthodes exposées par les MBean enregistrés sur le serveur JMX.

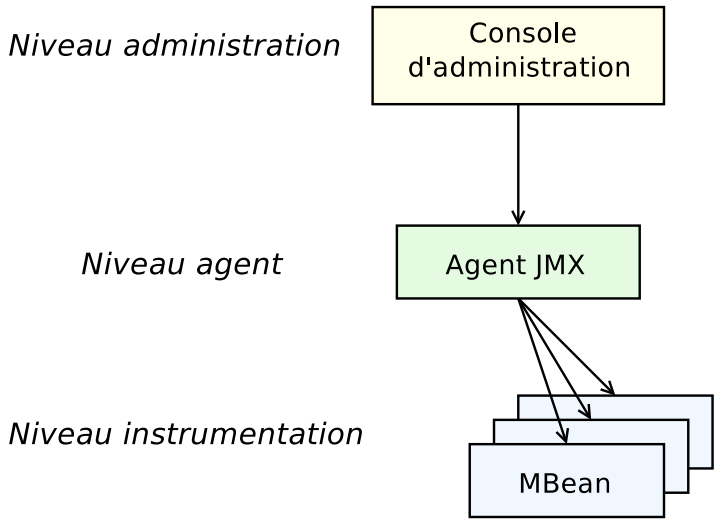

FIGURE 7.16 – Architecture d'une application JMX

JMX est utilisé ici pour permettre de connecter les différentes briques de MoKa, à savoir les proxies MoKa (sondes et actionneurs), avec le contrôleur MoKa. L'utilisation de JMX dans notre cadre applicatif se justifie pour plusieurs raisons. Premièrement, JMX est facile à mettre en oeuvre, et limite la quantité de code à écrire pour l'administration des services Internet. Ensuite, le fait que JMX soit standardisé permet de connecter notre contrôleur avec différentes applications sans avoir à réécrire les connecteurs. De plus, JMX permet d'obtenir un système d'administration modulaire, les différents MBean pouvant être connectés ou non à l'agent d'administration. Ceci permet donc de faciliter le passage à l'échelle de notre système d'administration. Enfin JMX est une partie intégrante du J2SE, et bénéficie de l'expérience d'une communauté importante d'utilisateurs.

AOP / AspectJ

Afin de ne pas modifier le code source de l'application, nous avons construit un système de monitorage non invasif basé sur la programmation par aspects. Nous utilisons les techniques de conception par aspect et l'injection autour d'aspects pour instrumenter, de manière transparente, les services Internet sous-jacents et les connecter aux sondes et aux actionneurs de MoKa. L'utilisation de la programmation orientée aspects (POA ou *AOP* -

aspect oriented programming) permet en effet de tisser les aspects non fonctionnels de monitorage à l'application administrée sans avoir à modifier le code source de cette dernière.

Sondes de monitorage

Dans le service Internet utilisé pour nos expériences, le monitorage du temps d'attente des clients a été réalisé grâce à AspectJ [21], l'extension orientée aspects de Java. Grâce à AspectJ, nous interceptons tous les appels aux méthodes des servlets constituant le premier étage du service Internet. Les clients étant identifiés par des sessions, nous pouvons calculer en particulier le temps moyen entre l'envoi d'une réponse à un client et la réception de la requête suivante. Ceci correspond au temps d'attente moyen Z. Les mesures de latence et de débit sur le premier étage de l'application sont effectuées simultanément à la mesure du temps d'attente. Pour les autres étages du service Internet, nous utilisons également l'injection d'aspects dans l'application pour obtenir les informations de latence et débit. Les différences concernent le type de méthode à instrumenter, en fonction du type de protocole utilisé pour communiquer avec l'étage. Nous interceptons par exemple tous les appels aux méthodes JDBC [51] utilisées par le premier étage pour communiquer avec le deuxième étage, hébergeant la base de données. L'utilisation d'aspects permet dans la plupart des cas de ne pas avoir à connaître le code source du service Internet contrôlé lors de la construction du système de monitorage.

Actionneurs de reconfiguration.

Les opérations de lancement et d'arrêt des serveurs applicatifs composant le service Internet sont décrites dans des scripts. Ceci permet de pouvoir étendre facilement la gamme des serveurs contrôlables par MoKa, contrôler un nouveau type de serveur nécessitant seulement d'écrire les scripts de démarrage et d'arrêt du serveur.

7.6.3 Interfaces utilisateur

Notre contrôleur a été conçu pour être facilement interfacé avec d'autres applications. En effet, le contexte d'utilisation de l'outil d'administration concerne surtout des entreprises importantes, ayant souvent un système d'information préexistant. Il est donc nécessaire de fournir une méthode simple d'intégration de notre outil dans la majorité des environnements informatiques. Comme on l'a vu en 7.6.2, le choix de JMX répond à cette problématique de simplicité et de généricité. Le contrôleur de MoKa contient donc un serveur JMX listant l'ensemble des attributs et des opérations exposées aux utilisateurs. Les informations exposées via JMX par le contrôleur sont :

- les informations de performance (latence, débit), de disponibilité (taux de rejet) et de coût mesurées sur le service Internet
- la configuration optimale calculée par le modèle
- la configuration actuelle du service Internet
- les informations concernant les différents objectifs de qualité de service du SLA

Les opérations possibles sur le contrôleur sont :

- la modification de un ou plusieurs objectifs de qualité de service
- la définition d'une liste de serveurs disponibles, pouvant être alloués par le contrôleur aux différents étages de l'application

Interface dynamique. Nous avons développé une interface Web dynamique pour MoKa, afin de permettre aux administrateurs d'utiliser MoKa à distance depuis un navigateur Web. L'interface permet de surveiller l'évolution dynamique des performances et de la configuration d'un système administré par MoKa, ainsi que de changer dynamiquement les contraintes de qualité de service. L'interface permet également de vérifier les prédictions du modèle par rapport aux mesures effectuées.

Cette interface a été écrite en Java, à l'aide du canevas logiciel GWT [26]. GWT permet en effet de construire des applications Web 2.0 complexes de manière efficace. La figure 7.17 présente l'interface utilisateur de MoKa, affichant la quantité de charge et les latences modélisées et mesurées pour un service Internet en fonctionnement. Comme on le voit, le contrôleur détecte automatiquement les variations dans la quantité de charge. Les informations à afficher dans l'interface graphique peuvent être modifiées pendant le fonctionnement de l'application grâce au menu de gauche. Ce menu propose d'afficher les mesures concernant la charge (quantité de charge, ratios de visite, temps d'attente), de qualité de service (latence, débit, taux de rejet), les prédictions du modèle ainsi que les paramètres du modèle calculés automatiquement (S_i et D_i) . Le contrôleur est capable d'afficher l'évolution récente de la valeur de chacun de ces paramètres sur une fenêtre glissante de durée paramétrable. La fréquence de mise à jour de l'interface graphique est également paramétrable. L'interface proposée permet enfin de modifier les contraintes de qualité de service devant être respectées par le service Internet pendant son fonctionnement (SLA).

La figure 7.18 affiche l'interface graphique de MoKa lors d'un changement dans le type de charge du service Internet administré (la quantité de charge reste constante). Lors du changement, à $16:07$, le ratio de visite sur le deuxième étage du service évolue pour passer de 3 à 1,5. Le paramétrage du modèle évolue en conséquence, et les paramètres internes calculés automatiquement (S_i) sont adaptés. Aucune intervention humaine n'a été nécessaire pour prendre en compte ce changement dans le type de charge.

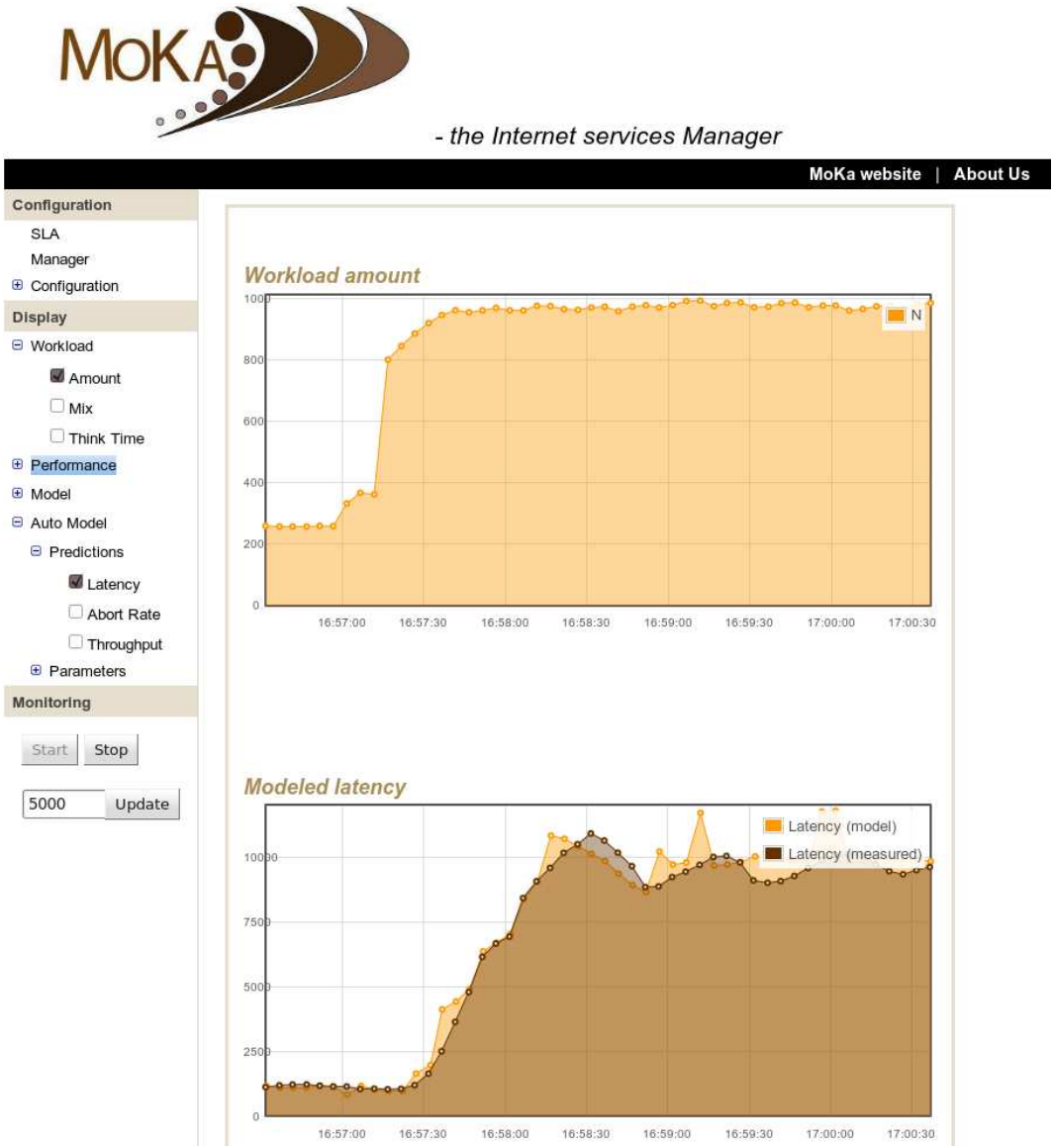

FIGURE 7.17 – Interface Web de MoKa - évolution de la quantité de charge

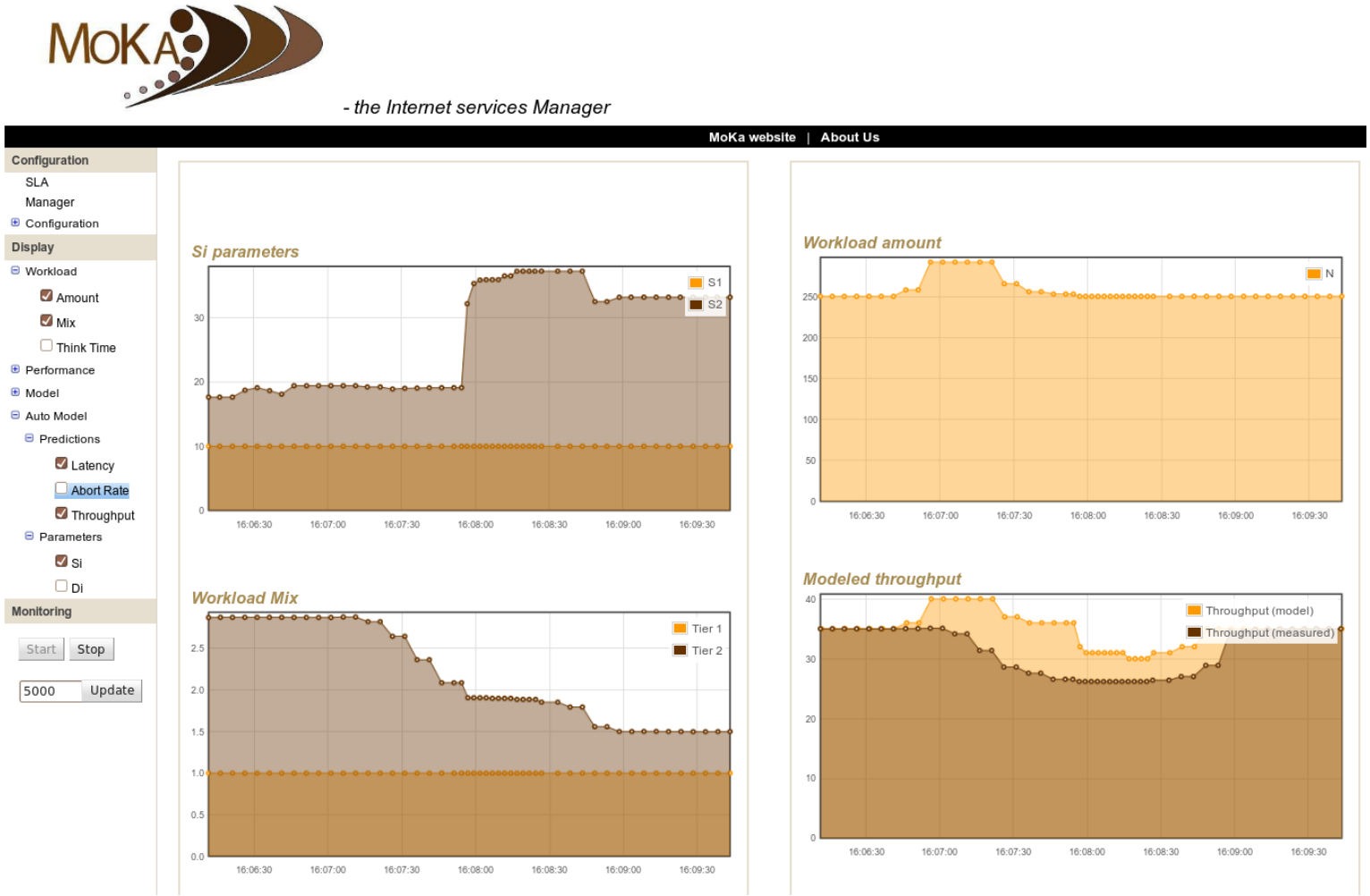

FIGURE 7.18 – Interface Web de MoKa - évolution du type de charge

7.6.4 Données factuelles

Logiciel MoKa

Le tableau 7.1 détaille la quantité de lignes de codes composant chaque partie de MoKa. L'ensemble du prototype a nécessité l'écriture de 12900 lignes de codes et de script, réparties dans trois catégories.

Le contrôleur autonome de Moka, embarquant la modélisation et la planification de capacité en a requis la majeure partie, soit 7500 lignes. Les modèles et les algorithmes de planification de capacité sont dans des modules séparés de l'architecture générale du contrôleur. Ceci permet d'ajouter plus facilement des modèles ou des algorithmes de planification de capacité. L'architecture centrale de Moka comprend donc le système de communication JMX, les opérations de traitement et de centralisation des données monitorées sur l'application, ainsi que le gestionnaire d'administration autonome. Un service annexe permet également l'archivage des actions du gestionnaire autonome, ainsi que des informations détaillées sur le fonctionnement du service administré.

Les sondes de monitorage et les actionneurs de reconfiguration dépendent fortement du service Internet administré. Pour le service de librairie en ligne utilisé dans nos expériences, 3400 lignes ont été nécessaires.

L'interface graphique dynamique est proposée en tant que programme indépendant, collectant les informations à afficher et effectuant les modifications sur le service administré uniquement via JMX. L'écriture de l'interface graphique a nécessité 2000 lignes de code.

Module	**Lignes de code Java**	**Lignes de code AspectJ**	**Lignes de script**
Architecture du contrôleur	2500	0	0
Modélisation	2000	0	0
Planification de capacité	3000	0	0
Monitorage et actionneurs	1300	300	1800
Interfaces graphiques	2000	0	0
TOTAL	10800	300	1800

TABLE 7.1 – Lignes de code du prototype MoKa

7.7 Synthèse

Nous avons présenté dans ce chapitre le prototype MoKa pour l'administration autonome de services Internet soumis à des contraintes de qualité de service. Ce prototype comporte des sondes de monitorage permettant de collecter les informations nécessaires sur le service Internet administré, des actionneurs de reconfiguration, et un contrôleur autonome. Ce dernier implémente la modélisation présentée dans le chapitre 5 et la planification de capacité présentée dans le chapitre 6. Ce chapitre détaille également le paramétrage en ligne automatique du modèle, ce paramétrage utilisant les informations collectées par le système de monitorage.

Cette conception permet au contrôleur d'être utilisé sans modifications dans différents contextes, seuls les sondes et les actionneurs devant être adaptés au service administré.

Le chapitre suivant présente une validation expérimentale de la solution proposée, et plus spécifiquement du modèle et de la planification de capacité intégrée au contrôleur. Une

évaluation plus globale des gains obtenus grâce à notre approche, en termes de qualité de service et de coût, est également proposé dans le chapitre 8.

Chapitre 8

Expérimentations et évaluations

Ce chapitre présente l'évaluation expérimentale du contrôle adaptatif avec MoKa. Nous commençons par détailler l'environnement d'évaluation. Nous évaluons ensuite deux aspects de notre proposition : le modèle analytique pour la précision de ses prédictions, et la planification de capacité pour l'optimalité de son résultat. Enfin, nous présentons l'évaluation du contrôle adaptatif de MoKa appliqué à un service Internet subissant des variations de charge dans le temps.

8.1 Environnement d'évaluation

Environnement matériel

Nos expériences ont été conduites sur des machines de la plate-forme Grid'5000, une grille de calcul dédiée à la recherche [9]. Chaque machine comporte deux processeurs Intel Xeon fonctionnant à 2,33 GHz, et embarque 4 Go de mémoire. Un réseau Gigabit Ethernet dédié permet d'interconnecter l'ensemble des machines.

Environnement logiciel

Les machines utilisaient le système d'exploitation Linux 2.6.26. Pour nos expériences avec les services Internet multi-étagés, nous avons utilisé la version 5.5 du serveur Web et de servlets Tomcat[23], ainsi que la version 5.0 du serveur de base de données MySQL [52]. Par ailleurs, pour la mise en oeuvre de MoKa, nous avons utilisé Java 1.6 [50] et la version 1.6 d'AspectJ [21].

Banc d'essai de service Internet

Les bancs d'essais, ou *benchmarks* sont des applications permettant de mesurer le comportement et les performances d'un système. Utiliser un banc d'essai permet de disposer d'une application réaliste et de s'assurer ainsi de l'intérêt des solutions validées grâce à ce banc d'essai. De plus, les bancs d'essais reconnus permettent de comparer les résultats de différentes approches, et de garantir la reproductibilité des expériences.

Nous avons utilisé le banc d'essai TPC-W [56] pour les expériences de validation de notre approche. TPC-W a été développé par le *Transaction Processing Performance Council* (TPC), et

représente un standard largement utilisé pour reproduire le comportement des services Internet. TPC-W est une application de commerce électronique de type *Amazon.com*. Un émulateur client génère un certain nombre de clients en parallèle, chaque client produisant des interactions avec le service Internet. Ces interactions sont par exemple la recherche d'informations sur un article, des ajouts dans le panier ou des commandes en ligne, ou encore la consultation des meilleures ventes du moment. La mise en oeuvre de TPC-W utilisée est basée sur une architecture à deux étages [55]. Le premier étage est responsable de la logique métier et de la génération des pages renvoyées aux clients. Le deuxième étage est en charge du stockage des données persistantes.

Nous considérons dans la suite deux types de charge. Dans le premier type de charge $C1$, les clients consultent surtout les meilleurs ventes du moment, tandis que dans le second type de charge $C2$, les clients effectuent principalement des recherches d'articles.

8.2 Évaluation du modèle analytique

Dans cette section, nous évaluons la précision du modèle. Pour cela nous comparons les estimations produites par le modèle et les mesures effectuées sur un service Internet réellement déployé.

8.2.1 Scénario d'évaluation

Les expériences de validation de la précision du modèle ont été réalisées avec le banc d'essai TPC-W. Le scénario d'évaluation devant couvrir un maximum de points dans l'espace de paramétrage du modèle, nous faisons pour cela évaluer le type de charge et la quantité de charge comme présenté sur la figure 8.1. Deux types de charge sont utilisés. Le premier correspond à des clients effectuant des consultations d'articles sur le service Internet *(browsing mix)*. Le deuxième est une version modifiée du premier type de charge entraînant davantage de sous-requêtes vers la base de données. Le type de charge est modifié au milieu de l'expérience (au temps 5000 s). La quantité de charge est également modifiée au cours du temps. Des variations graduelles ainsi que des pics de charge sont générés pour correspondre aux cas réel rencontrés par les services Internet. La variation de quantité de charge est la même pour les deux types de charge, permettant ainsi de comparer l'influence du type de charge sur le modèle.

Des configurations locales et architecturales différentes sont utilisées afin de s'assurer que le modèle capture bien l'influence de la configuration sur les performances et la disponibilité des services Internet. Nous utilisons une configuration minimale κ_1, ainsi qu'une configuration plus importante κ_2. La configuration locale de κ_2 utilise les valeurs par défaut des serveurs utilisés (200 pour le premier étage et 100 pour le deuxième étage [23, 52]). La configuration architecturale de κ_2 suppose que l'administrateur ait constaté que le deuxième étage est le plus sollicité, et nécessite donc plus de serveurs. tableau 8.1 présente le détail des configurations utilisées lors des expériences. La configuration architecturale de κ_1 est la configuration minimale du service, tandis que la configuration locale de κ_1 est fixée à une valeur importante pour éviter trop de rejet de clients.

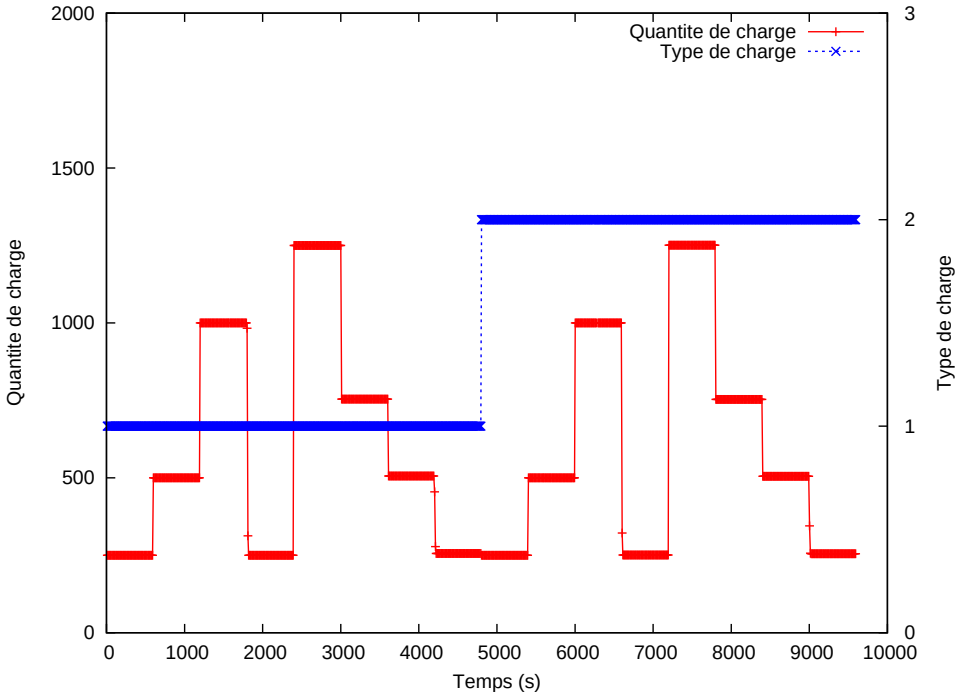

FIGURE 8.1 – Variation de charge

Configuration	M	AC	LC
κ_1	2	$< 1, 1 >$	$< 500, 500 >$
κ_2	2	$< 3, 6 >$	$< 200, 100 >$

TABLE 8.1 – Configurations ad-hoc utilisées avec TPC-W

8.2.2 Évaluation de la précision du modèle

Le modèle est capable de s'adapter aux variations de quantité et de type de charge pour prévoir correctement la latence, le débit et le taux de rejet pour chacune des configurations considérées. La figure 8.2 compare les latences mesurées sur le système réel et prédites par le modèle, pour les configurations κ_1 et κ_2. Pour la configuration κ_1, la moyenne des écarts entre modèle et mesure est de 8%. Pour κ_2, l'écart moyen est de 25 %, mais ceci est dû à la faible valeur de la latence. En effet, la valeur absolue de cet écart moyen est de seulement 20 ms. On constate une prédiction correcte du modèle quelle que soit la quantité ou le type de charge pour les deux configurations considérées.

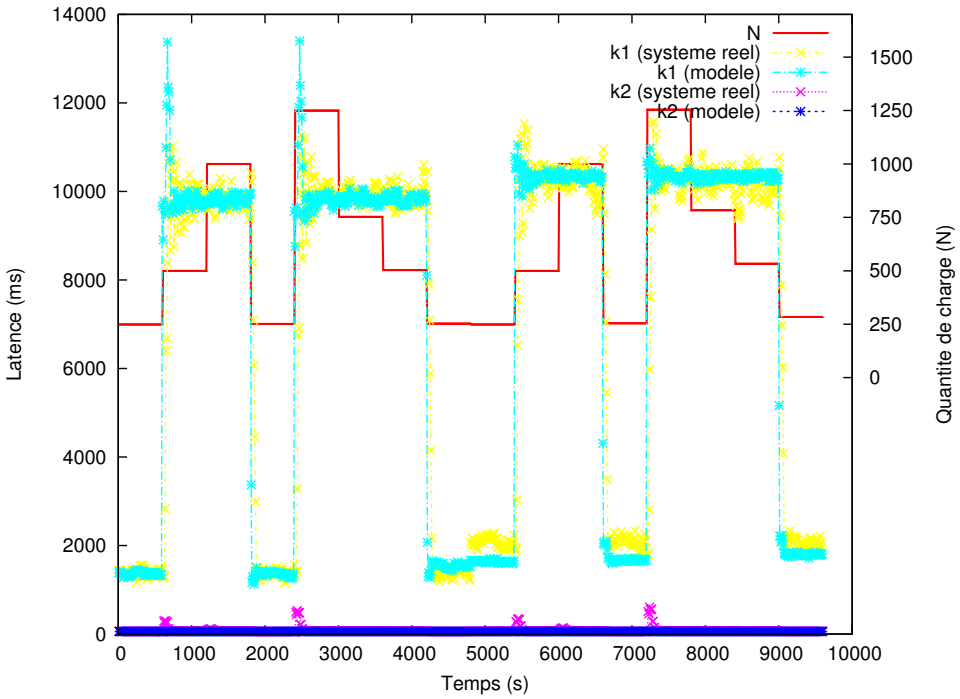

FIGURE 8.2 – Latences mesurées et prédites

La figure 8.3 compare les débits mesurés et calculés par le modèle pour les configurations κ_1 et κ_2. L'écart moyen entre mesure et prédiction du débit est de moins de 3% pour κ_1. Pour κ_2, l'écart moyen est égal à 4%. La modélisation du débit reste donc précise malgré les variations importantes des caractéristiques de la charge. Le modèle prévoit donc correctement le comportement du débit pour les deux configurations.

Les variations de taux de rejet en fonction de la charge, représentées sur la figure 8.4, sont également capturées correctement par le modèle pour les deux configurations considérées. Les écart moyens entre modèle et mesure pour κ_1 et κ_2 sont en effet de 2% et 1%. Les expériences montrent donc que le modèle est capable de prédire correctement les performances et la disponibilité d'un service Internet multi-étagé faisant face à des variations de la quantité et de la nature de sa charge.

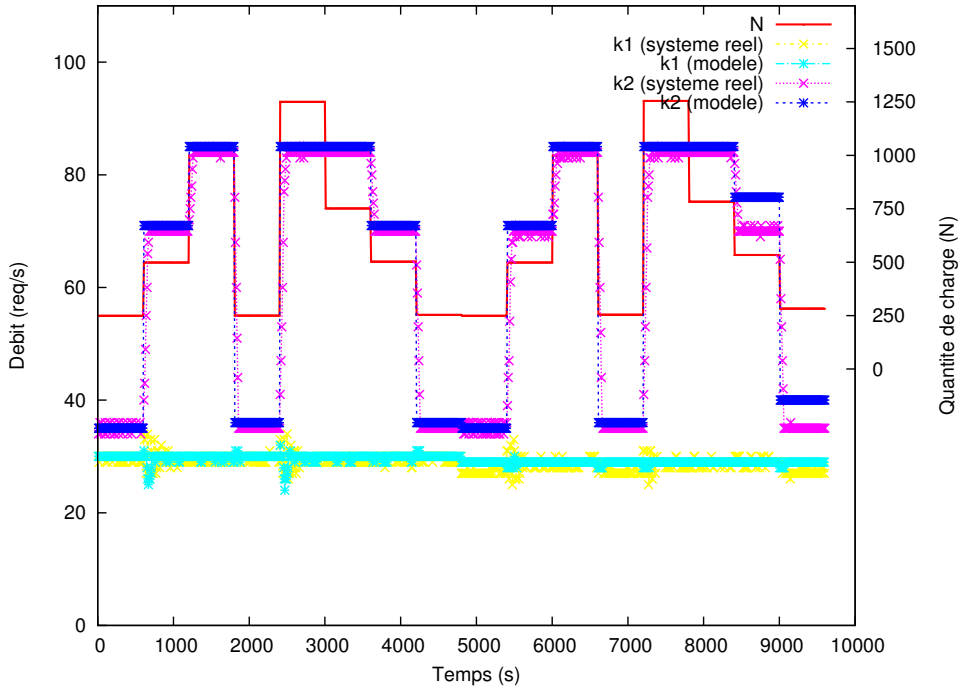

FIGURE 8.3 – Débits mesurés et prédits

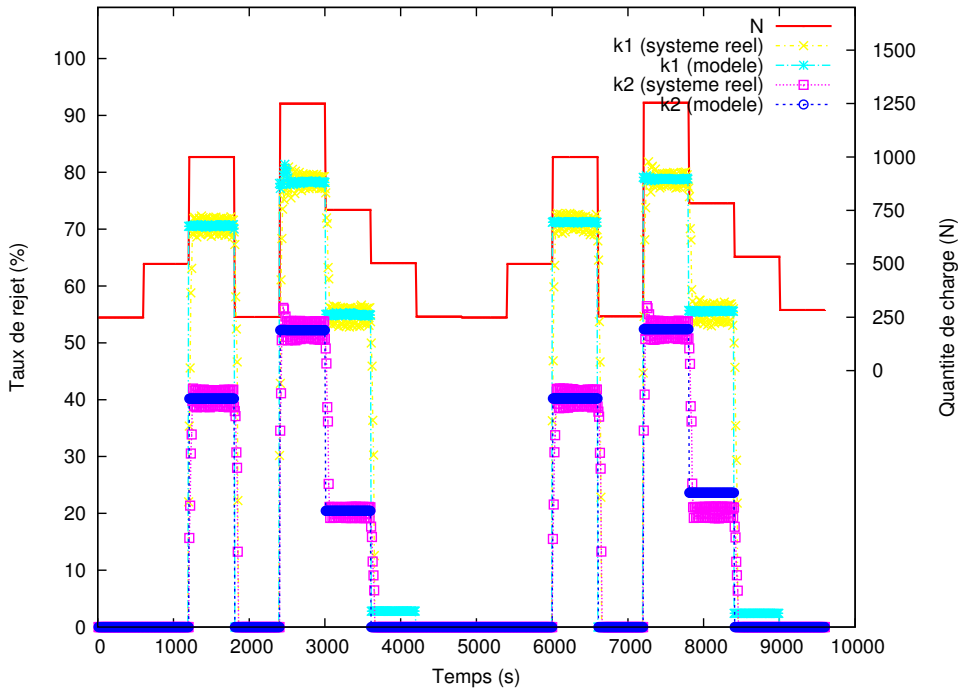

FIGURE 8.4 – Taux de rejet mesurés et prédits

8.3 Évaluation de la planification de capacité

8.3.1 Scénario d'évaluation

L'optimalité de la planification de capacité s'évalue en mesurant la différence entre la configuration calculée par l'algorithme de planification de capacité et la configuration optimale pour le contexte donné. La configuration optimale peut toujours être trouvée en effectuant une recherche exhaustive dans l'espace des configurations. Même si cette recherche est extrêmement coûteuse, elle garantit de trouver la configuration optimale.

Nous comparons les prédictions de notre solution de planification de capacité avec l'algorithme de recherche exhaustive. Le contrat de qualité de service utilisé par les algorithmes de planification de capacité est représenté dans le tableau 8.2.

ℓ_{max}	α_{max}
$1000\ ms$	10%

TABLE 8.2 – Contraintes de qualité de service du SLA

8.3.2 Évaluation de l'optimalité de la planification de capacité

La figure 8.5 présente la configuration architecturale calculée par l'algorithme de recherche exhaustive et l'algorithme de planification de capacité intégré dans MoKa, pour chacun des étages de l'application. On constate que les configurations calculées sont équivalentes, et ceci pour toutes les valeurs de type de charge et de quantité de charge considérées.

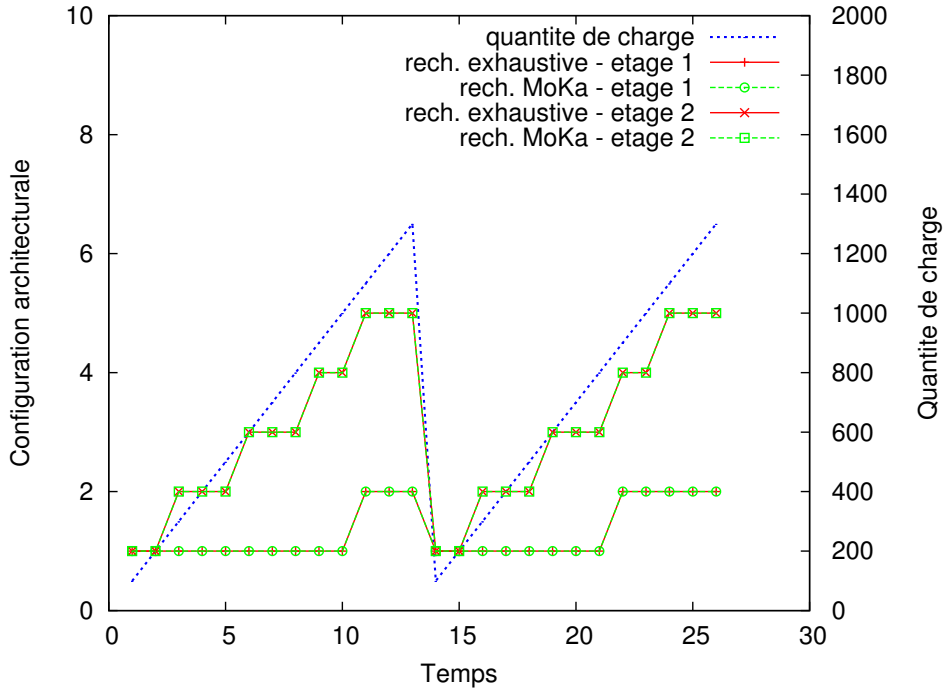

FIGURE 8.5 – Précision de la planification de capacité - configuration architecturale

Les configurations locales calculées par chacun des algorithmes sur les différents étages de l'application sont présentées sur la figure 8.6. On constate que les configurations sont identiques entre la recherche exhaustive et l'algorithme proposé dans MoKa, quelle que soit la quantité et le type de charge. Notre algorithme a donc la même précision qu'une recherche exhaustive.

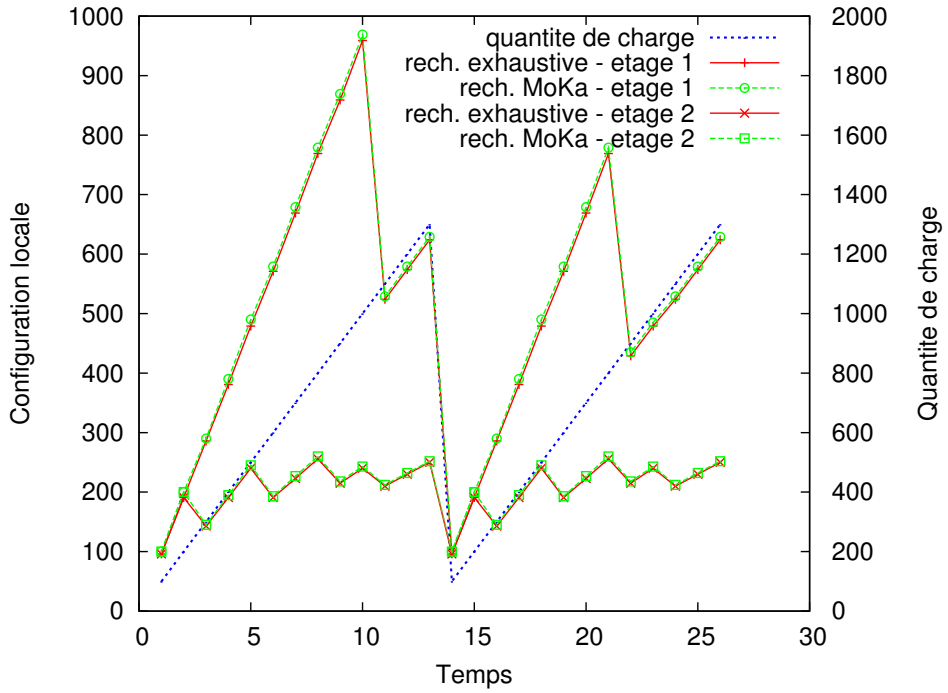

FIGURE 8.6 – Précision de la planification de capacité - configuration locale

8.4 Évaluation du contrôle adaptatif

8.4.1 Scénario d'évaluation

Nous comparons dans cette section le comportement d'un système dont la configuration est gérée par notre contrôleur autonome (κ_3), avec les comportements des systèmes configurés de manière ad-hoc (κ_1 et κ_2). Les configurations ad-hoc sont celles présentées dans le tableau 8.1. Nous mettons ainsi en évidence les gains obtenus concernant le coût de fonctionnement du service Internet, ainsi que le respect des différents critères de performance et de disponibilité. Le banc d'essai utilisé dans cette expérience est TPC-W. Le contrat de qualité de service que doit respecter le service Internet est représenté dans le tableau 8.3.

ℓ_{max}	α_{max}
$1000\ ms$	5%

TABLE 8.3 – Contraintes de qualité de service du SLA

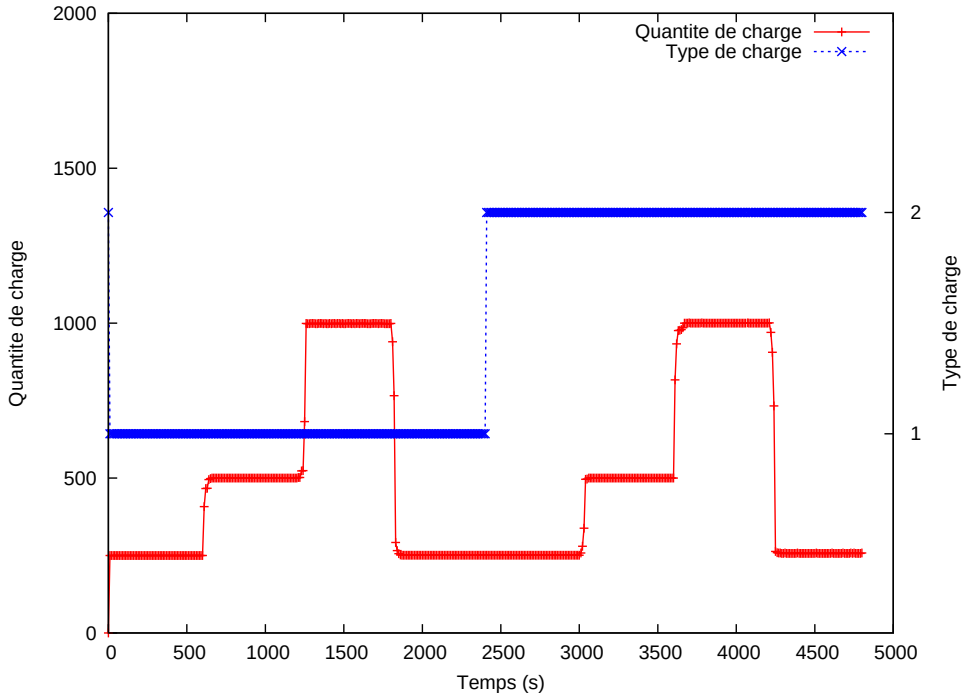

FIGURE 8.7 – Charge du service Internet

8.4.2 Évaluation de l'adaptabilité des services Internet

Les évolutions de la quantité et du type de charge lors de cette expérience sont représentées sur la figure 8.7. Le type de charge change au milieu de l'expérience, tandis que la quantité de charge comprend des évolutions en palier, ainsi que des pics de charge. La figure 8.8 compare les latences des configurations ad-hoc avec celle des configurations contrôlées par MoKa. La majorité du temps, la configuration κ_1 ne respecte pas le critère de latence maximum du contrat de qualité de service, à cause d'une configuration trop faible. Au cours de l'expérience, la latence moyenne de κ_1 est de 7030 ms. La configuration κ_2 ainsi que la configuration contrôlée κ_3 permettent d'obtenir une latence respectant l'objectif fixé, avec respectivement 76 ms et 670 ms de latence moyenne.

Les taux de rejet sont représentés sur la figure 8.9. On y constate un fort taux de rejet pour les configurations κ_1 et κ_2 lorsque le système fait face à de fortes quantités de charge, entre 1200s et 1800s, et entre 3600s et 4200s. La configuration autonome de κ_3 lui permet de continuer à respecter l'objectif de 5% de rejet maximum pendant ces périodes, après les délais nécessaires pour reconfigurer le service.

L'évolution du débit pour chacune des configurations est représentée sur la figure 8.10. On constate que le débit de κ_1 reste constant tout au long de l'expérience, a une valeur moyenne de 26 requêtes par seconde. A cause de sa faible configuration architecturale, le système atteint en effet sa capacité de traitement maximale même avec une faible charge. Les augmentations de quantité de charge se traduisent donc par une augmentation du taux de rejet pour cette configuration. La configuration architecturale plus élevée de κ_2 lui permet de traiter plus de charge, mais on constate un phénomène identique à celui observé avec κ_1 lors des périodes de forte charge. La configuration architecturale de κ_2 est suffisante, mais

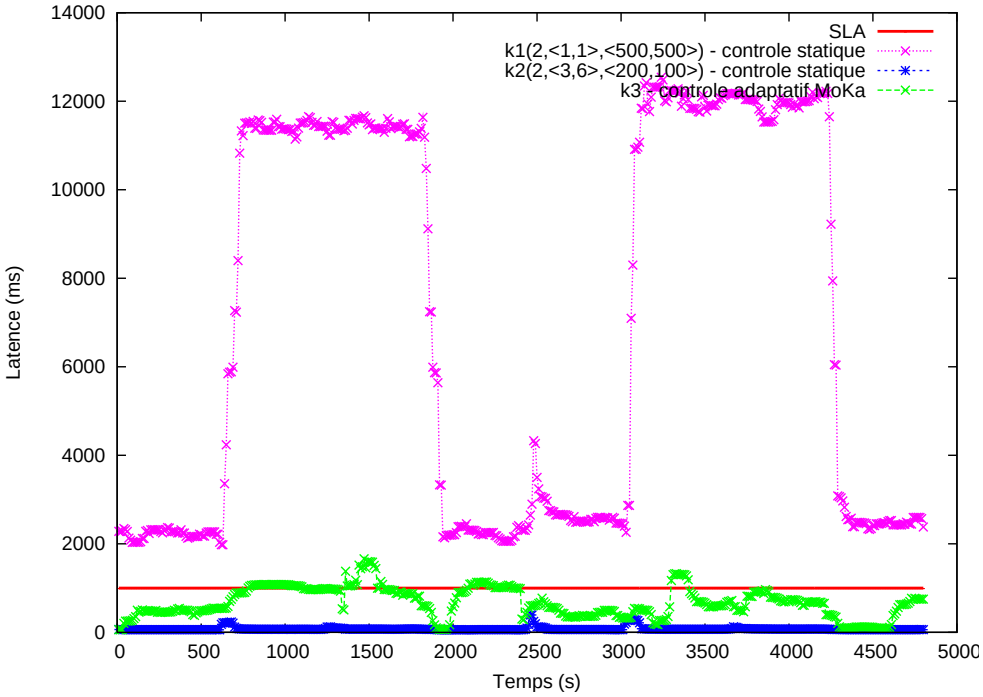

FIGURE 8.8 – Latences du service Internet

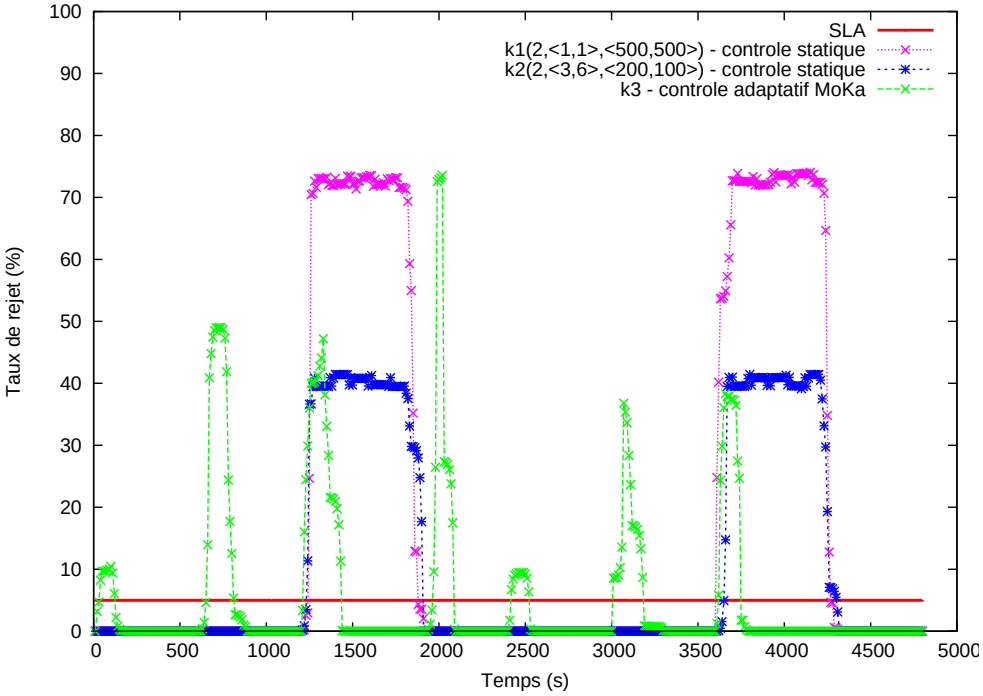

FIGURE 8.9 – Taux de rejet du service Internet

sa configuration locale limite l'accès des requêtes au service Internet. Cela se traduit par une très bonne latence, mais également par un taux de rejet important. Le débit moyen de κ_2 est de 55 requêtes par seconde. Avec κ_3, on constate un débit de requêtes traitées évoluant suivant la quantité de charge, ce qui signifie que le système absorbe correctement la charge. Ceci est vérifié par le faible taux de rejet de la configuration contrôlée κ_3. Le débit moyen avec κ_3 est de 60 requêtes par seconde.

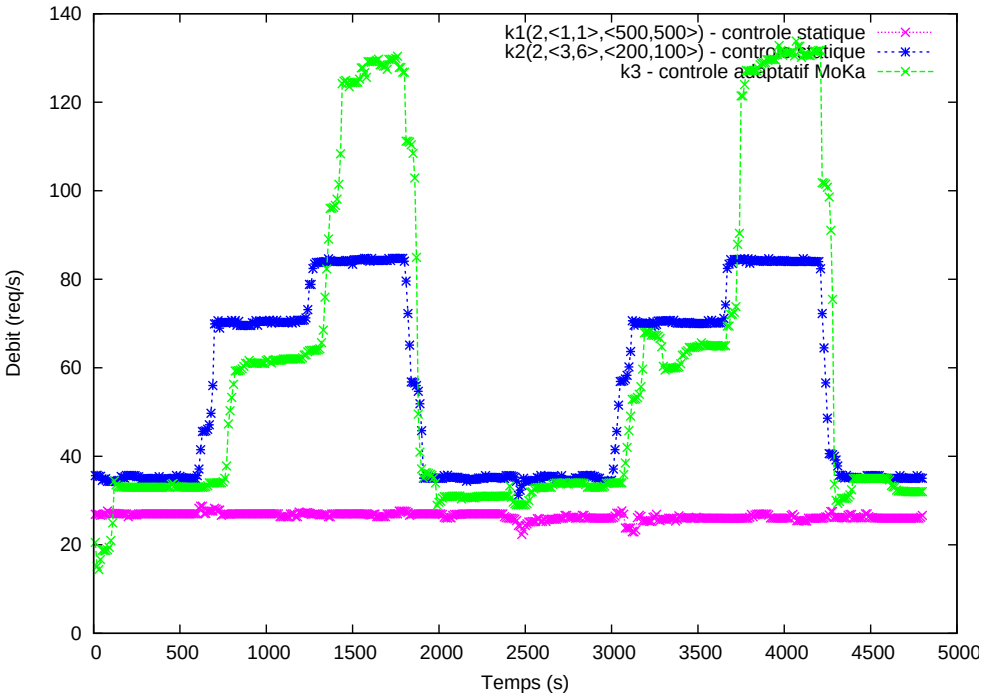

FIGURE 8.10 – Débit du service Internet

L'ensemble des caractéristiques de la configuration architecturale et de la configuration locale de κ_3 sont représentées respectivement dans les figures 8.11 et 8.12. On constate que l'évolution de la charge impacte surtout le deuxième étage, alors que le premier étage nécessite des ressources supplémentaires uniquement lors des pics de charge. La configuration locale sur le premier étage s'adapte afin d'éviter un écroulement du serveur entre l'apparition d'un éventuel pic de charge et le moment où la nouvelle configuration adéquate serait appliquée. La configuration locale sur le deuxième étage reste plus faible que celle du premier étage, car le temps de service du deuxième étage est plus élevé que le temps de service du premier étage. La méthode de contrôle utilisée par MoKa étant basée sur un calcul périodique de la configuration optimale, la configuration locale et architecturale du service Internet est modifiée à intervalles réguliers. On constate cependant qu'en l'absence de variation de quantité de charge ou de type de charge, la valeur de cette configuration évolue peu, et reste stabilisée autour d'une valeur moyenne. Les pics dans la configuration locale de l'étage 2 sont des mécanismes de protection de MoKa, pour éviter de rejeter des requêtes lors de la reconfiguration simultanée des configurations locale et architecturale du service.

La figure 8.13 présente l'évolution de la fonction d'utilité pour les configurations κ_1, κ_2 et κ_3. La configuration κ_1 ne respectant jamais les objectifs de qualité de service, sa valeur d'utilité reste à 0. La majorité du temps, la configuration κ_3 contrôlée par MoKa donne la

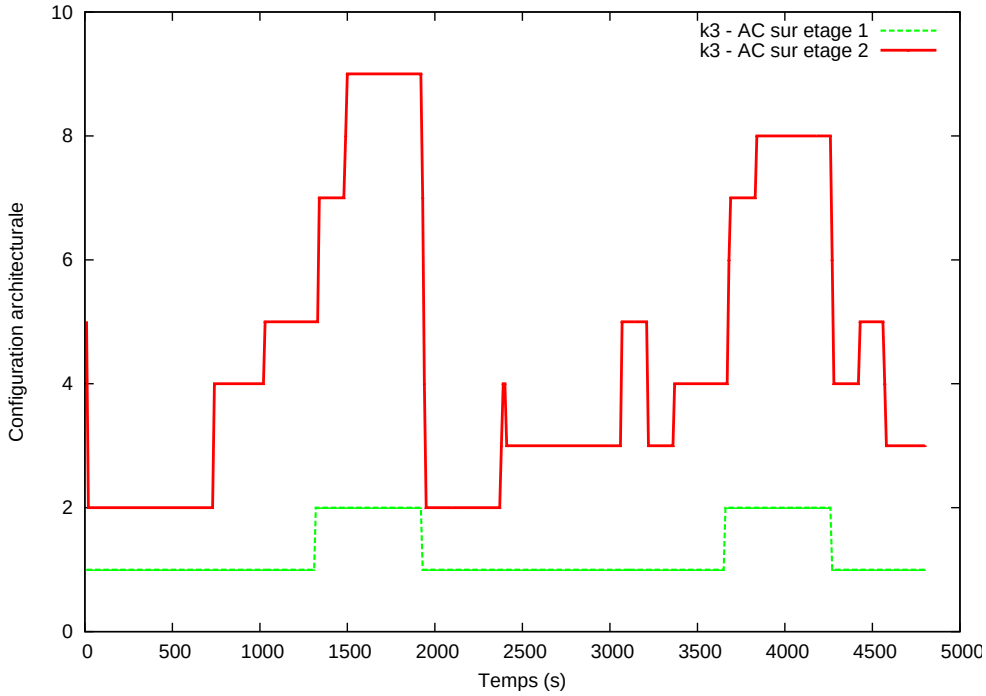

FIGURE 8.11 – Évolution de la configuration architecturale du système contrôlé

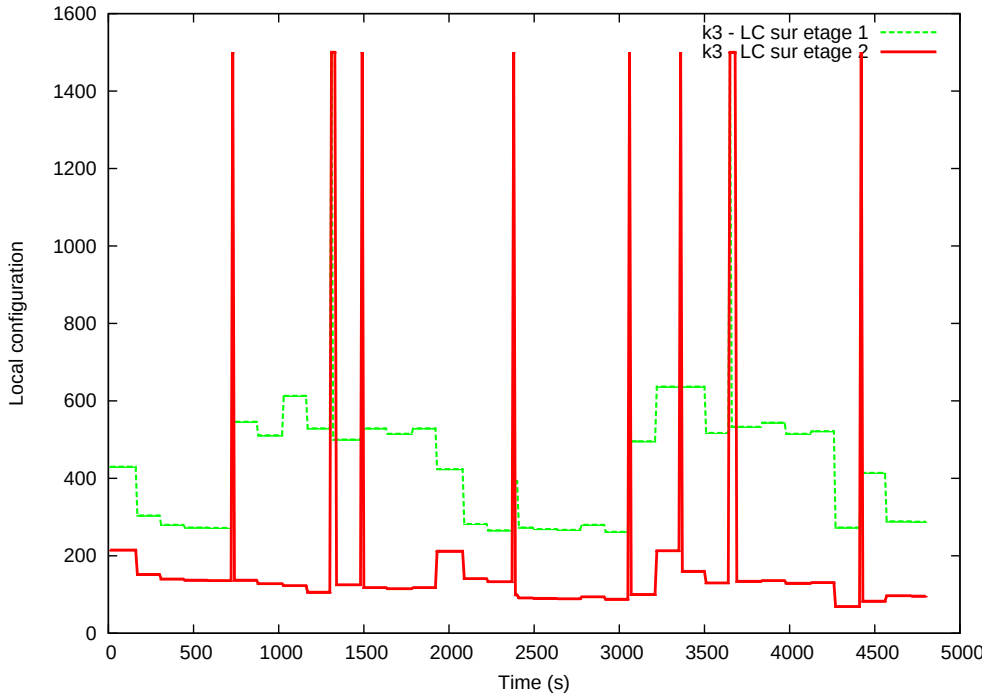

FIGURE 8.12 – Évolution de la configuration locale du système contrôlé

meilleure valeur d'utilité, devant κ_2.

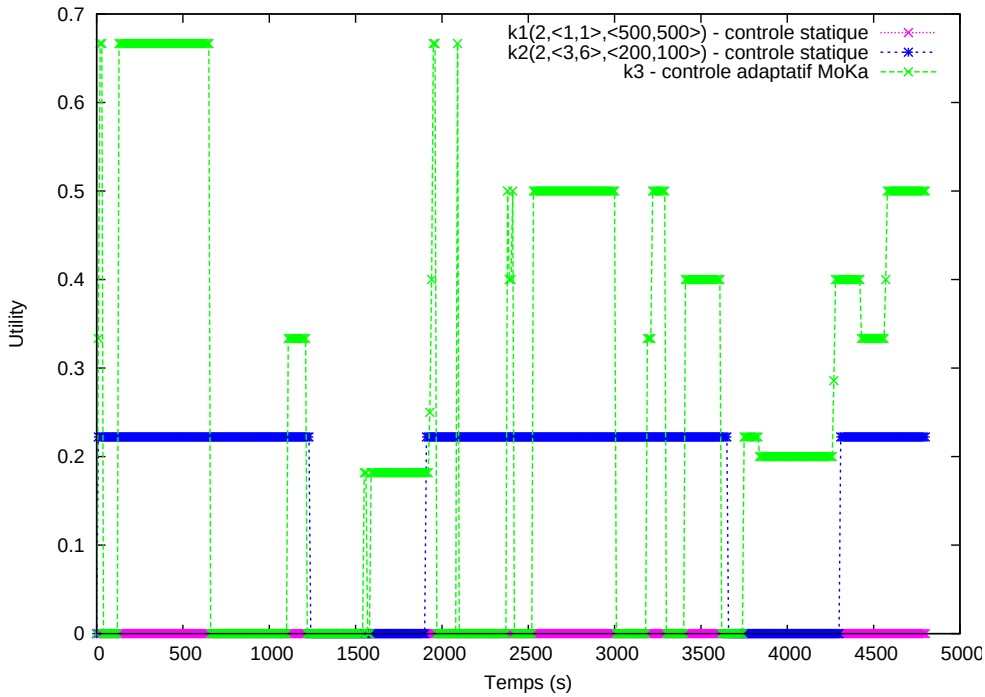

FIGURE 8.13 – Évolution de la fonction d'utilité

Enfin, le coût des différentes configuration considérées, c'est-à-dire la quantité de serveurs utilisés, est représenté sur la figure 8.14. Si le coût des configurations ad-hoc est fixe par définition, on constate que les variations du coût de κ_3 dépassent rarement le coût de κ_2, pour des performances et une disponibilité meilleures. Le coût moyen de κ_3 sur la durée de l'expérience est de 5,7 serveurs, soit une économie de 36% par rapport à κ_2. Le coût de κ_1 est le plus faible, mais cette configuration ne permet pas de respecter les contraintes de qualité de service du SLA. La configuration gérée par notre contrôleur, κ_3, est donc capable de respecter l'ensemble des contraintes de qualité de service tout en minimisant le coût de fonctionnement du service Internet.

8.4.3 Autres techniques de contrôle

Les approches existantes se concentrent soit sur la configuration locale (contrôle d'admission) soit sur la configuration architecturale (approvisionnement de ressources) des services Internet (voir tableau 3.1). Ceci ne permet pas d'atteindre la configuration optimale et entraîne souvent un gaspillage de serveurs. L'expérience suivante compare des services Internet uniquement optimisés au niveau local, des système uniquement optimisés au niveau architectural, et des systèmes où l'optimisation est faite sur les deux niveaux de l'architecture. Les configurations considérées sont présentées dans le tableau 8.4. Lors de ces expériences, les contraintes de qualité de service sont $\ell_{max} = 1000$ ms et $\alpha_{max} = 10$ %.

$AA - OL_1$ et $AA - OL_2$ représentent deux configurations avec une configuration architecturale ad-hoc, et une configuration locale optimisée avec un contrôle d'admission comme présenté dans [39]. $OA - AL_1$ et $OA - AL_2$ représentent deux configurations dont la confi-

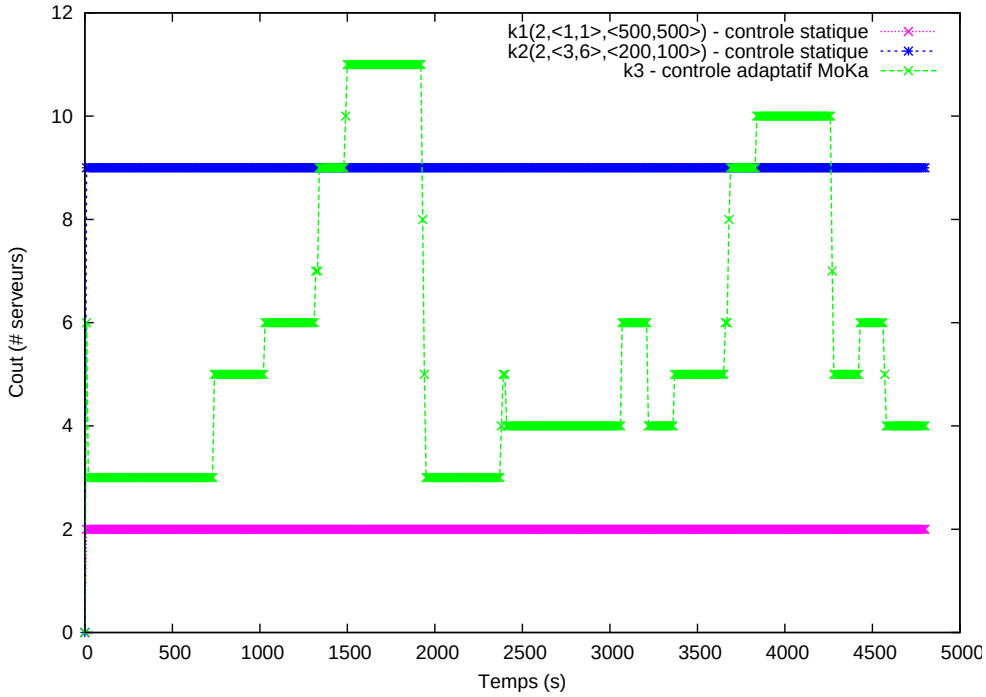

FIGURE 8.14 – Comparaison du coût des configurations

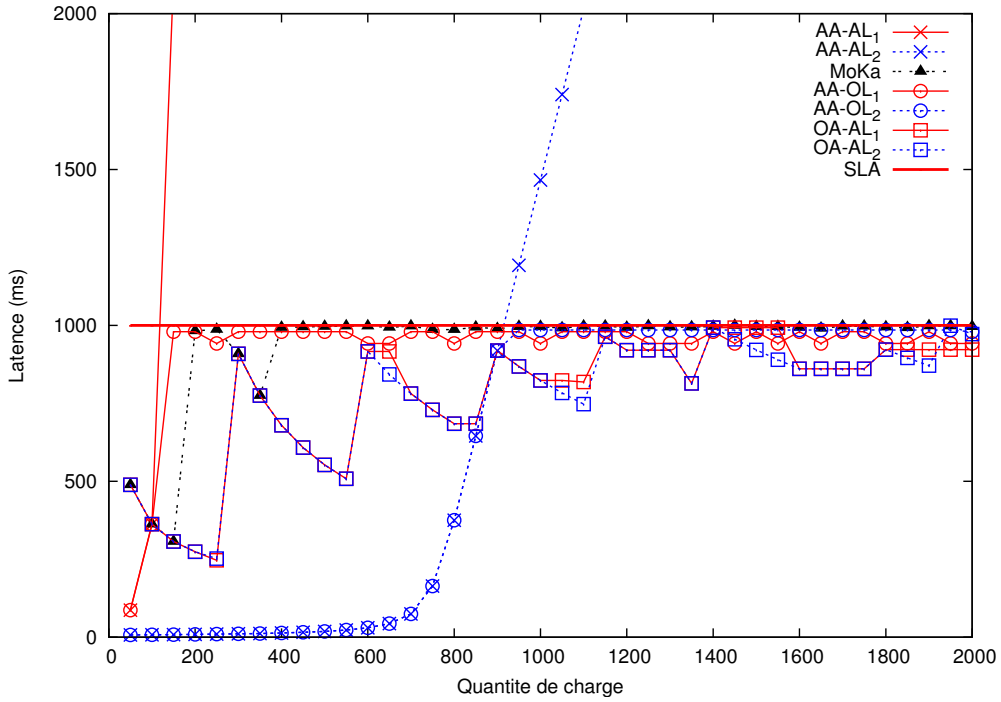

FIGURE 8.15 – Latence

Config.	architecturale	locale
AA-AL_1	Ad-hoc $AC < 1, 2 >$	Ad-hoc $LC < 200, 100 >$
AA-AL_2	Ad-hoc $AC < 6, 15 >$	Ad-hoc $LC < 300, 200 >$
MoKa	MoKa-based control	
AA-OL_1	Ad-hoc $AC < 1, 2 >$	Optimisée
AA-OL_2	Ad-hoc $AC < 6, 15 >$	Optimisée
OA-AL_1	Optimisée	Ad-hoc $LC < 200, 100 >$
OA-AL_2	Optimisée	Ad-hoc $LC < 300, 200 >$

TABLE 8.4 – Configurations du service Internet

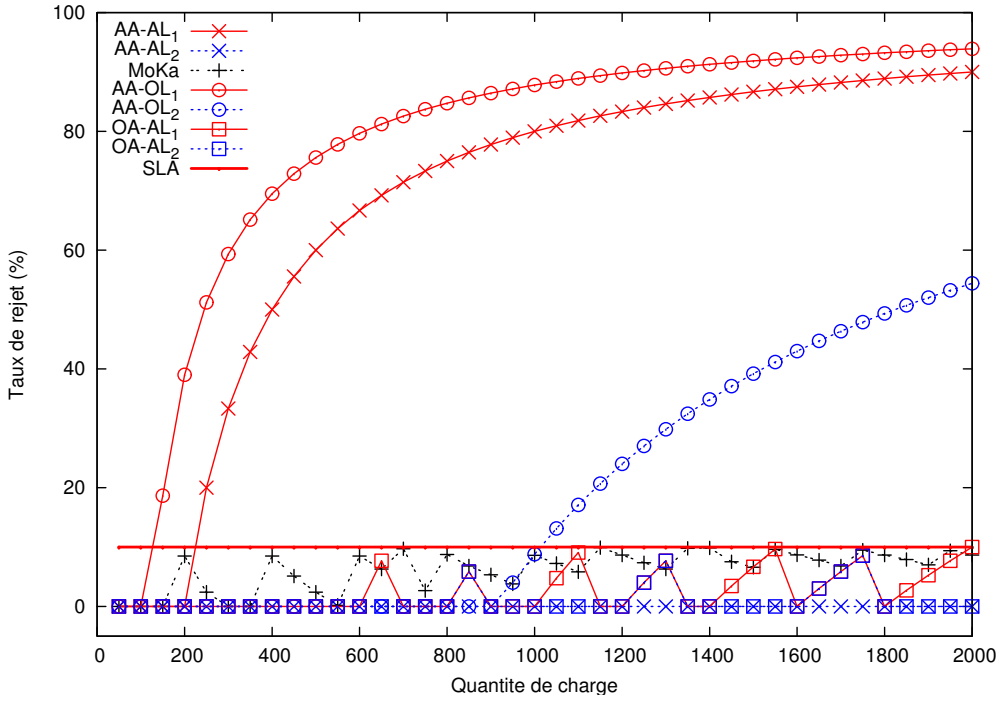

FIGURE 8.16 – Taux de rejet

guration locale est fixée de manière ad-hoc, et la configuration architecturale est optimisée suivant une approche similaire à [57]. La figure 8.15 montre que les configurations $AA-OL_1$ et $AA-OL_2$ vérifient la contrainte de latence maximum car elles sont capables d'optimiser leur configuration locale en limitant la concurrence sur les serveurs. Cependant, ceci a un impact direct sur le taux de rejet qui croît lorsque la charge augmente, et donc ne respecte plus la contrainte de taux de rejet comme on peut le voir sur la figure 8.16. Le taux de rejet croît plus tard avec $AA-OL_2$ qu'avec $AA-OL_1$ car le premier représente une configuration avec plus de ressources qui absorbent une partie de la charge. Enfin la configuration calculée par MoKa est optimisée à la fois localement et architecturalement, et permet de garantir à la fois les contraintes α_{max} et ℓ_{max}. Les configurations $OA - AL_1$ et $OA-AL_2$, avec une configuration architecturale optimisée et une configuration locale ad-hoc sont également capables de garantir les contraintes de latence et de taux de rejet, comme illustré sur les figures 8.15 et 8.16. Mais le coût de ces configurations est élevé, comme nous allons le voir.

En plus de la comparaison des performances, nous évaluons et comparons aussi le coût

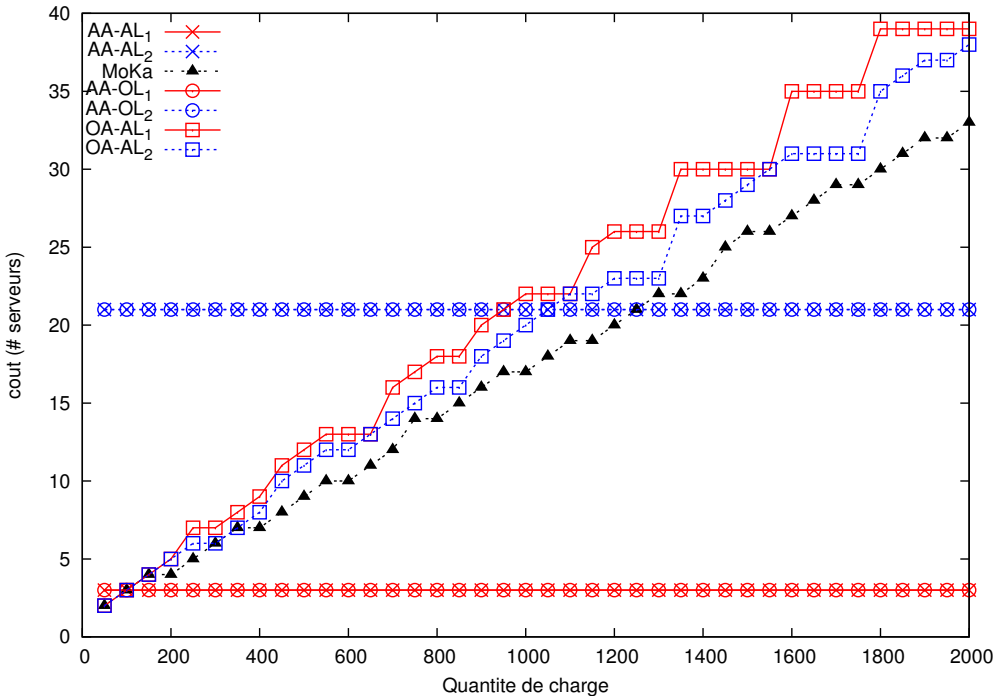

FIGURE 8.17 – Coût

(i.e. le nombre de serveurs) de toutes les configurations comme décrit sur la figure 8.17. Évidemment, $AA - AL_1$, $AA - AL_2$, $AA - OL_1$ et $AA - OL_2$ ont un coût constant car leur configuration architecturale est fixée de manière ad-hoc, avec un coût de 3 pour $AA - AL_1$ et $AA - OL_1$, et un coût de 21 pour $AA - AL_2$ et $AA - OL_2$. Par contre le coût de $OA - AL_1$, $OA - AL_2$ et $OA - OL$ croît lorsque la charge augmente, car leur configuration architecturale est optimisée en fonction. $OA - AL_1$ a un coût plus élevé que $OA - AL_2$. Ceci est du au fait que $OA - AL_1$ a une configuration locale plus faible que celle de $OA - AL_2$, et donc une plus faible concurrence sur les serveurs est autorisée avec $OA - AL_1$. Donc, pour que $OA - AL_1$ respecte la contrainte de taux de rejet α_{max}, cette configuration a besoin d'augmenter son nombre de serveurs et ainsi accroître le nombre global de requêtes traitées en parallèle. $OA - OL$ présente un coût plus faible que $OA - AL_1$ et $OA - AL_2$ car, à l'inverse de ces dernières, $OA - OL$ est capable d'optimiser la configuration locale du système et donc de maximiser l'usage des serveurs lorsque c'est possible, avant d'utiliser des serveurs supplémentaires.

En résumé, nos expériences montrent que comparée aux approches avec seulement une configuration locale où jusqu'à 94% des requêtes peuvent être rejetées, et comparée aux approches avec seulement une configuration architecturale où jusqu'à 15% des serveurs peuvent être gaspillés, une approche qui combine à la fois configuration locale et architecturale permet de garantir le SLA tout en minimisant le coût du service.

8.5 Synthèse

Ce chapitre présente l'évaluation du prototype MoKa de modélisation et de planification de capacité pour les services Internet multi-étagés. L'évaluation est réalisée à l'aide d'un banc d'essai et porte sur plusieurs points. Premièrement, la précision du modèle est évaluée,

afin de vérifier que ses prédictions sont correctes dans l'ensemble des contextes existants. Deuxièmement, la précision et l'efficacité de la planification de capacité sont évalués, pour s'assurer que la configuration calculée par notre solution est bien la configuration optimale, et que le temps total de calcul de cette configuration permet bien l'utilisation en ligne de notre approche. Enfin, nous évaluons les bénéfices apportés par notre solution en termes de garanties de respect du contrat de qualité de service, et de minimisation du coût des services Internet administrés.

Chapitre 9

Conclusion et perspectives

9.1 Synthèse

Dans cette thèse, nous nous intéressons aux problématiques posées par la variation dynamique de la quantité et du type de la charge dans des services Internet multi-étagés. Des pics de charge ou des changements dans le comportement des clients peuvent fortement impacter la qualité de service de ces applications. Or ces dernières sont soumises à des contrats de qualité de service stricts, devant être respectés malgré les fluctuations dans la charge subie. Pour pouvoir supporter une charge importante, les hébergeurs de services Internet allouent parfois plusieurs milliers de serveurs au fonctionnement d'un service, afin d'assurer une capacité de traitement suffisante pour les requêtes des clients. Cependant, les coûts économiques et écologiques liés au fonctionnement des serveurs deviennent problématiques à cette échelle. Afin de limiter ces coûts, les hébergeurs cherchent donc à limiter la quantité de ressources matérielles allouées au fonctionnement des services Internet. Minimiser le coût d'un service tout en améliorant sa performance sont cependant deux objectifs antagonistes. De plus, une configuration adaptée à un moment donné ne le sera pas quel que soit le contexte, car la quantité et la nature de la charge évoluent en permanence. Afin de réduire les coûts de fonctionnements des services Internet tout en préservant leur qualité de service, la configuration de ces services doit être dynamiquement adaptée.

Cette étude a pour but de fournir une solution autonome d'adaptation automatique de la configuration des services Internet, pour que ces derniers respectent un ensemble de contraintes de qualité de service malgré une variation dans la quantité ou la nature de leur charge. Cette solution est basée sur les éléments suivants.

Tout d'abord, nous fournissons une *fonction d'utilité* pour les services Internet, permettant d'agréger les différents critères de performance, de disponibilité et de coût d'une configuration et à les comparer au contrat de qualité de service.

Un *modèle* de service Internet multi-étagé et dupliqué est également proposé. Ce modèle permet de prédire les performances, la disponibilité et le coût d'un service avec une configuration et une charge données.

Nous proposons ensuite un algorithme de *planification de capacité* pour les services Internet, qui calcule la configuration du service garantissant les objectifs spécifiés dans le contrat de qualité de service tout en minimisant le coût du service. Cette planification de capacité utilise le modèle vu précédemment, ainsi que la fonction d'utilité pour comparer l'utilité d'une configuration par rapport aux contraintes de qualité de service données.

Un prototype de contrôleur autonome de services Internet, nommé MoKa, a été conçu et réalisé pour valider l'approche proposé. Ce prototype intègre la fonction d'utilité, le modèle et l'algorithme de planification de capacité permettant de calculer la configuration optimale du service Internet Ce prototype prend en charge l'ensemble des tâches concernant la boucle de contrôle, depuis la collecte d'informations sur le service Internet jusqu'à l'application de la configuration optimale. En effet, la collecte des mesures concernant la quantité ou le type de charge sont réalisées par un système de monitorage en ligne de l'application administrée. Les variations dans la quantité ou le type de charge sont détectées automatiquement, et le modèle recalibré en conséquence. De même, la configuration optimale calculée par l'algorithme de planification de capacité est automatiquement appliquée au système administré, sans nécessiter l'arrêt et le redémarrage de celui-ci. La prise en compte de nouveaux objectifs de qualité de service peut également se faire dynamiquement.

La diffusion de ce prototype en tant que logiciel libre est en cours, afin de permettre à la communauté industrielle et scientifique de réutiliser le travail effectué dans le cadre de cette thèse [4]. Les résultats de ces travaux ont été présentés dans des publications et manifestations scientifiques [3, 8, 7, 6, 5].

9.2 Perspectives

Les perspectives scientifiques ouvertes par ces travaux sont nombreuses. Premièrement, les techniques de configuration au niveau local (contrôle d'admission) et au niveau architectural (approvisionnement de serveurs) utilisées pourraient être complétées par d'autres techniques de configuration. En particulier, la différentiation de service, consistant à traiter les clients avec différents niveaux de priorité, pourrait être intégrée dans notre prototype afin de permettre aux services Internet administrés de fournir différents niveaux de garantie de qualité de service à différentes classes de clients.

Nous avons vu que les problématiques de gestion de qualité de service et de contrôle deviennent stratégiques dans le cadre des services Internet. Dans cette thèse, nous avons suivi une démarche de conception et de mise en oeuvre qui tentait de limiter l'intrusivité sur le service logiciel sous-jacent. Cependant, à l'heure actuelle, ces problématiques n'apparaissent pas clairement lors de la conception des applications. Nous pensons que la définition d'un langage permettant d'intégrer dès la phase de conception des applications la prise en compte de la qualité de service avec des sondes de monitorage applicatives et des actionneurs de reconfiguration applicatifs, permettrait de faciliter et de fiabiliser la gestion de performance pour les services Internet.

L'application de notre approche à d'autres contextes applicatifs pourrait également être considérée. Les services de messagerie d'entreprise ou les réseaux de capteurs font face à des problématiques proches de celles rencontrées dans le cadre du contrôle des services Internet. De manière générale, les résultats de cette thèse trouveraient leur application pour le contrôle de services de *cloud computing* à différents niveaux.

L'application de l'approche proposée dans cette thèse à l'administration de services à grande échelle est limitée par l'utilisation d'un contrôleur centralisé. Un contrôleur unique pose des problèmes de passage à l'échelle si la quantité de messages à recevoir et envoyer devient trop importante.

Enfin, fournir des garanties temporelles concernant la durée des reconfigurations et le temps nécessaire pour atteindre les objectifs de qualité de service permettrait d'utiliser l'ap-

proche proposée dans le cadre des systèmes temps-réel.

Bibliographie

[1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *The IEEE 5th Annual Workshop on Workload Characterization (WWC(5)*, Austin, TX, November 2002.

[2] Apache. Apache http server. http ://httpd.apache.org/.

[3] Jean Arnaud. Automated control of internet services. In *5th International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID 2010)*, Paris, France, April 2010.

[4] Jean Arnaud and Sara Bouchenak. Moka. http ://sardes.inrialpes.fr/research/moka/.

[5] Jean Arnaud and Sara Bouchenak. Gestion de ressources dans les services internet. In *CFSE*, Fribourg, Suisse, February 2008.

[6] Jean Arnaud and Sara Bouchenak. Modeling and capacity planning of multi-tier systems. Technical Report RR-6730, INRIA, November 2008.

[7] Jean Arnaud and Sara Bouchenak. Moka : Optimisation de services internet multi-étagés. In *Conférence Internationale sur les NOuvelles TEchnologie de la REpartition (NOTERE 2009)*, Montreal, Canada, July 2009.

[8] Jean Arnaud and Sara Bouchenak. Adaptive internet services through performance and availability control. In *25th ACM Symposium On Applied Computing (SAC 2010)*, Sierre, Switzerland, March 22 - 26 2010. ACM Press.

[9] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lantéri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E-G. Talbi, and I. Touche. Grid'5000 : a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4) :481–494, November 2006.

[10] Sara Bouchenak, Noël De Palma, Daniel Hagimont, and Christophe Taton. Autonomic management of clustered applications. *IEEE International Conference on Cluster Computing*, September 2006.

[11] George E.P. Box, Alberto Luceno, and Maria Paniagua-Quinones. *Statistical control by monitoring and adjustment*. Wiley ; 2 edition, 2009.

[12] Martin Brown. Optimizing apache server performance, February 2008. http ://www.serverwatch.com/tutorials/article.php/3436911.

[13] Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite, and Willy Zwaenepoel. Performance comparison of middleware architectures for generating dynamic web content. In *ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, Brazil, June 2003.

[14] Abhishek Chandra, Prashant Pradhan, Renu Tewari, Sambit Sahu, and Prashant Shenoy. An observation-based approach towards self-managing web servers. *Comput. Commun.*, 29(8) :1174–1188, 2006.

[15] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01 : Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116, New York, NY, USA, 2001. ACM.

[16] J. Chen, G. Soundararajan, and C. Amza. Autonomic provisioning of backend databases in dynamic content web servers. In *The 3rd IEEE International Conference on Autonomic Computing (ICAC 2006)*, Dublin, Ireland, June 2006.

[17] Xiangping Chen, Huamin Chen, and Prasant Mohapatra. Aces : An efficient admission control scheme for QoS-aware web servers. *Computer Communications*, 26(14) :1581–1593, 2003.

[18] Peter J. Denning and Jeffrey P. Buzen. The operational analysis of queueing network models. *ACM Comput. Surv.*, 10(3) :225–261, 1978.

[19] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury. Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server. In *Network Operations and Management Symposium*, April 2002.

[20] Yixin Diao, Joseph L. Hellerstein, Sujay Parekh, Hidayatullah Shaikh, and Maheswaran Surendra. Controlling quality of service in multi-tier web applications. In *ICDCS '06 : Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 25, Washington, DC, USA, 2006. IEEE Computer Society.

[21] Eclipse. Aspectj. http ://www.eclipse.org/aspectj/.

[22] Sameh Elnikety, Erich Nahum, John Tracey, and Willy Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. In *WWW '04 : Proceedings of the 13th international conference on World Wide Web*, pages 276–286, New York, NY, USA, 2004. ACM Press.

[23] The Apache Software Foundation. Apache tomcat. http ://tomcat.apache.org/.

[24] Saeed Ghanbari, Gokul Soundararajan, Jin Chen, and Cristiana Amza. Adaptive learning of metric correlations for temperature-aware database provisioning. In *ICAC*, page 26, 2007.

[25] Jordi Guitart, Jordi Torres, and Eduard Ayguadé. A survey on performance management for internet applications. *Concurr. Comput. : Pract. Exper.*, 22(1) :68–106, 2010.

[26] Robert Hanson and Adam Tacy. *GWT in Action : Easy Ajax with the Google Web Toolkit*. Manning Publications Co., Greenwich, CT, USA, 2007.

[27] Hans-Ulrich Heiss and Roger Wagner. Adaptive load control in transaction processing systems. In *VLDB*, pages 47–54, 1991.

[28] Dejun Jiang, Guillaume Pierre, and Chi-Hung Chi. Autonomous resource provisioning for multi-service web applications. In *WWW '10 : Proceedings of the 19th international conference on World wide web*, pages 471–480, New York, NY, USA, 2010. ACM.

[29] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1) :41–50, January 2003.

[30] Jonathan G. Koomey. Estimating total power consumption by servers in the u.s. and the world. *Analytics Press*, 2007.

[31] Jonathan G. Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(034008), 2008.

[32] Linuxfr. http ://linuxfr.org.

[33] J.D.C. Little. A proof of the queuing formula l =lambda.w. *Operations Research*, 9 :383–387, 1961.

[34] Chris Loosley, Frank Douglas, and Alex Mimo. *High-Performance Client/Server*. John Wiley & Sons, November 1997.

[35] Luc Malrait, Sara Bouchenak, and Nicolas Marchand. Fluid modeling and control for server system performance and availability. In *DSN*, pages 389–398, 2009.

[36] Luc Malrait, Sara Bouchenak, and Nicolas Marchand. Experience with conser : A system for server control through fluid modeling. *IEEE Transactions on Computers*, 2010.

[37] Evan Marcus and Hal Stern. *Blueprints for High Availability*. Wiley, September 2003.

[38] D. A. Menascé and V. A. F. Almeida. *Capacity Planning for Web Services : Metrics, Models, and Methods*. Prentice Hall, 2001.

[39] D. A. Menascé, D. Barbara, and R. Dodge. Preserving qos of e-commerce sites through self-tuning : A performance model approach. In *ACM Conference on Electronic Commerce (EC'01)*, Tampa, FL, October 2001.

[40] D. Menasce and V. Almeida. *Capacity Planning for Web Services : metrics, models, and methods*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[41] Daniel A. Menasce, Lawrence W. Dowdy, and Virgilio A. F. Almeida. *Performance by Design : Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[42] Microsoft. Optimizing database performance. http ://msdn.microsoft.com/en-us/library/aa273605(SQL.80).aspx.

[43] J.M. Milan-Franco, Ricardo Jiménez-Peris, Marta Patino-Martinez, and Bettina Kemme. Adaptive middleware for data replication. In *Middleware '04 : Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 175–194, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[44] Robert B. Miller. Response time in man-computer conversational transactions. In *AFIPS '68 (Fall, part I) : Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277, New York, NY, USA, 1968. ACM.

[45] Jakob Nielsen. Usability engineering. In Allen B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 1440–1460. CRC Press, 1997.

[46] Sujay S. Parekh, Neha Gandhi, Joseph L. Hellerstein, Dawn M. Tilbury, T. S. Jayram, and Joseph P. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Systems*, 23(1-2) :127–141, 2002.

[47] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *J. ACM*, 27(2), 1980.

[48] Marshall Rose. Dns support for load balancing. http ://tools.ietf.org/html/rfc1794.

[49] Swaminathan Sivasubramanian, Guillaume Pierre, Maarten van Steen, and Sandjai Bhulai. Sla-driven resource provisioning of multi-tier internet applications. Technical report, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 2006.

[50] Sun. Java. http ://java.sun.com/javase/.

[51] Sun. Java database connectivity. http ://java.sun.com/javase/technologies/database/.

[52] Sun. Mysql. http ://www.mysql.com/.

[53] Sun. Service level agreement in the datacenter.

[54] Christophe Taton, Sara Bouchenak, Noël De Palma, Daniel Hagimont, Sacha Krakowiak, and Jean Arnaud. Administration autonome de services Internet : Expérience avec l'auto-optimisation. In *5ème Conférence Française sur les Systèmes d'Exploitation (CFSE 2006)*, Le Canet en Roussillon, France, October 2006.

[55] Ross Dickson Timothy Heil Milo Martin Collin McCurdy Ravi Rajwar Eric Weglarz Craig Zilles Todd Bezenek, Trey Cain and Mikko Lipasti. Pharm. http ://www.ece.wisc.edu/ pharm/tpcw.shtml.

[56] TPC. Tpc-w transactional web e-commerce benchmark. http ://www.tpc.org/tpcw/.

[57] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. Analytic modeling of multitier internet applications. *ACM Transactions on the Web (ACM TWEB)*, 1(1) :2, 2007.

[58] D. Villela, P. Pradhan, and D. Rubenstein. Provisioning servers in the application tier for e-commerce systems. *ACM Trans. Interet Technol.*, 7(1), 2007.

[59] Matt Welsh, David Culler, and Eric Brewer. Seda : an architecture for well-conditioned, scalable internet services. volume 35, pages 230–243, New York, NY, USA, 2001. ACM.

[60] Q. Zhang, L. Cherkasova, and N. Mi. A regression-based analytic model for capacity planning of multi-tier applications. *Journal of Cluster Computing*, 11(3), 2008.