RenPar'18 / SympA'2008 / CFSE'6
Fribourg, février 2008

Gestion de ressources dans les services Internet

Jean Arnaud, Sara Bouchenak

Université Grenoble I – INRIA - LIG, Grenoble, France

{Jean.Arnaud, Sara.Bouchenak}@inria.fr

Résumé
Cet article présente une nouvelle solution de gestion de ressources dans les services Internet multi-niveaux tels que J2EE. L'approche proposée vise à maximiser les performances des services Internet tout en minimisant leur coût de fonctionnement. Elle est basée sur une modélisation de l'application et un algorithme efficace de calcul d'une configuration optimale des services Internet.

Mots-clés : services Internet, J2EE, gestion de ressources, optimisation

1. Introduction

1.1. Contexte et motivations

Il existe une grande variété de services Internet, tels que des serveurs Web, de courier électronique, de commerce en ligne, de vidéo à la demande ou des serveurs d'entreprise. Les services Internet, et en particulier les serveurs d'entreprise, sont généralement organisés selon une architecture multi-niveaux. Une architecture multi-niveaux est constituée d'une série de serveurs, gérant chacun une partie du service. A titre d'exemple, un service de vente en ligne à trois niveaux est constitué d'un niveau présentation (serveur Web), d'un niveau métier (servlets/EJB) et un niveau données (SGBD), tel qu'illustré en figure 1.

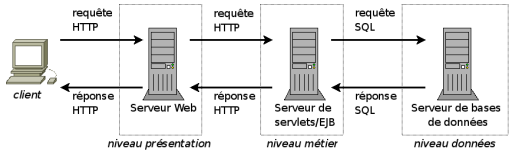


FIG. 1 – Architecture multi-niveaux

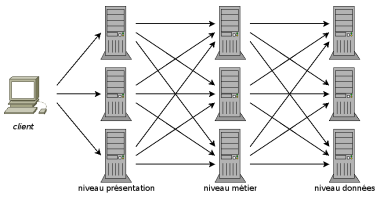


FIG. 2 – Architecture multi-niveaux et duplication

La demande actuelle concernant les services Internet est importante et continue de croître. Elle génère une charge toujours plus importante sur les serveurs en termes de nombre de requêtes de clients en concurrence sur les serveurs. Soumis à de fortes charges, ces services ont parfois du mal à garantir une qualité de service suffisante à leurs clients, tels qu'un temps de traitement de requête raisonnable. Une des techniques classiquement utilisées pour faire face à des trafics importants est la duplication. Il s'agit ici de fournir plusieurs instances (copies) d'un même serveur et de répartir la charge (requêtes des clients) sur les différentes copies du serveur. La charge est ainsi divisée et le système peut traiter plus de requêtes simultanément (voir figure 2).

Dans ce contexte, une question à laquelle l'administrateur du système doit répondre est la suivante : quelle est la quantité de ressources (i.e. machines) devant être allouées au système ? Une approche pessimiste classiquement utilisée est de considérer le cas le pire en terme de charge (le pic de charge maximum du système) et d'allouer le nombre de ressources nécessaires pour traiter cette charge. Une telle solution

peut naturellement mener à un gaspillage de ressources qui seraient sous-utilisées. Une autre approche plutôt optimiste se base sur le principe de l'*overbooking* tel qu'utilisé dans les systèmes de réservation aériens [11]. Une telle solution peut mener à des situations de surcharge du système. Ainsi les principaux défis des services Internet actuels sont de :

- Trouver la configuration optimale (e.g. nombre de ressources) qui maximise les performances du système (e.g. temps de traitement des requêtes clients) tout en minimisant le coût du système (e.g. nombre de machines utilisées).
- Permettre de faire évoluer la configuration du système pour que celle-ci s'adapte à la variation de la charge du système dans le temps.

1.2. Contributions scientifiques

Les principales contributions scientifiques de cet article sont :

- Une approche d'allocation de ressources aux services Internet qui soit *optimale* à la fois en terme de maximisation de performances du système et en terme de minimisation du coût inhérent à la quantité de ressources utilisées. Cette approche se base sur une modélisation du service Internet et propose un algorithme efficace de calcul de la configuration optimale.
- Une prise en compte de la configuration globale d'un service Internet multi-niveaux et ceci à deux dimensions : (i) une dimension macro qui considère le nombre de serveurs alloués à chaque niveau d'un système multi-niveaux (ce que nous appelons par la suite une *configuration architecturale*), et (ii) une dimension micro qui considère le paramétrage interne des serveurs (que nous appelons par la suite *configuration locale*). En effet le paramétrage interne (i.e. configuration locale) des serveurs peut influer de manière considérable sur les performances du système. L'intérêt de combiner la configuration locale et la configuration architecturale dans une solution de gestion de ressources est de tirer parti au maximum des ressources que possède déjà un service Internet avant d'allouer de nouvelles ressources au service, et donc d'être plus économe.
- Une proposition de solution de reconfiguration dynamique de services multi-niveaux qui recalcule la configuration optimale du système en fonction de la variation de la charge du système.

1.3. Plan de l'article

Le reste de l'article est organisé comme suit. La section 2 présente le cadre de nos travaux. La section 3 définit les principes de conception de la solution de gestion de ressources proposée. Les sections 4 et 5 décrivent respectivement les résultats de l'évaluation expérimentale menée et les principaux travaux apparentés. Enfin, la section 6 présente une conclusion de ces travaux.

2. Services Internet multi-niveaux

La plupart des services Internet modernes sont organisés selon une architecture multi-niveaux, afin d'améliorer les performances mais aussi pour faciliter le passage à l'échelle et la composition de services. Par exemple dans le cas d'une architecture à trois niveaux, la couche présentation reçoit les requêtes des clients Web, puis interroge la couche métier pour construire la page réponse. Le niveau métier peut éventuellement avoir besoin d'accéder au niveau données pour y mettre à jour et/ou y récupérer des données requises pour la construction de la réponse.

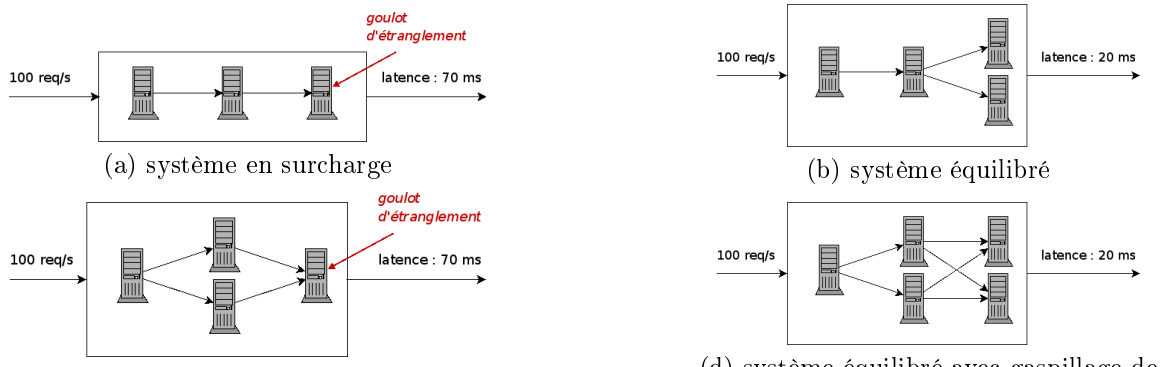
Configuration architecturale. Une des techniques classiquement utilisées pour permettre le passage à l'échelle des services Internet et faire face à une charge importante est la duplication des serveurs. La charge est ainsi répartie entre les différents duplicas des serveurs. Dans le cas de services Internet multi-niveaux, la duplication de serveurs peut intervenir à chacun des niveaux du service (voir figure 2).

Configuration locale. Actuellement, la majorité des serveurs composant les services Internet sont multiprogrammés, c'est-à-dire qu'ils supportent l'exécution de plusieurs processus (ou processus légers) à la fois. Cette technique permet d'augmenter le parallélisme d'un serveur pour que celui-ci traite les requêtes de plusieurs clients en parallèle.

2.1. Impact de la configuration architecturale sur les performances

La quantité de ressources à allouer à un service Internet dépend de la charge (requêtes clients) que doit traiter ce service. De plus, dans le cas de services multi-niveaux, certains niveaux peuvent être plus surchargés que d'autres. Il est important ici d'allouer des ressources aux niveaux les plus surchargés

car autrement aucun gain de performances ne sera constaté. A titre d'exemple, prenons le cas d'un service Internet à trois niveaux. Suivant la configuration architecturale en terme de ressources affectées à chacun des niveaux, les performances (ici le temps de traitement d'une requête client ou *latence*) peuvent varier de manière importante. La figure 3 présente, pour une même charge, quatre configurations architecturales présentant des différences de performances et d'économie de ressources. Dans la première configuration, le système est en surcharge à cause d'un goulot d'étranglement au niveau données. Dans la deuxième configuration, le système est équilibré car une ressource a été ajoutée au niveau présentant un goulot d'étranglement. Dans la troisième configuration, le système reste en surcharge même si une ressource supplémentaire a été allouée, car celle-ci n'a pas été ajoutée au niveau comportant le goulot d'étranglement. Enfin dans la quatrième configuration, le système est en équilibre mais gaspille une ressource qui aurait pu être économisée car non nécessaire pour traiter la charge.



(a) système en surcharge

(b) système équilibré

(c) système en surcharge avec gaspillage de ressources

(d) système équilibré avec gaspillage de ressources

FIG. 3 – Différentes configurations architecturales

2.2. Impact de la configuration locale sur les performances

Le degré de parallélisme d'un serveur multi-programmé a un impact direct sur les performances du serveur. Si le degré est faible, cela limite la contention sur le serveur et permet d'améliorer (i.e. diminuer) le temps de réponse aux requêtes. Mais dans un tel cas, le serveur peut-être sous-utilisé car limitant trop le parallélisme. Inversement, si le degré de parallélisme est élevé, cela améliorera (augmentera) l'utilisation du serveur, mais avec le risque d'obtenir des temps de réponse aux clients très longs voire infinis dans le cas de l'écroulement du système pour cause de parallélisme intense.

3. Principes de conception

3.1. Objectifs

L'objectif de ces travaux est double : maximiser les performances des services Internet tout en minimisant leur coût de fonctionnement.

Performances. Le premier objectif est d'atteindre une qualité de service donnée en terme de performances. La qualité de service peut porter sur différents critères, tels que la latence (ou temps de réponse à une requête client), le débit (ou la quantité de requêtes que le serveur traite par unité de temps). Dans un contrat de qualité de service de type SLA *(Service Level Agreement)*, il est par exemple demandé de garantir que le temps de réponse moyen des requêtes clients ne dépasse pas une latence maximale donnée. Un de nos objectifs est de fournir une solution de gestion de ressources des services Internet avec une garantie de latence bornée.

Coût. Le deuxième objectif est de minimiser la quantité de ressources (nombre de machines physiques) utilisées pour le fonctionnement du service Internet. En effet le fonctionnement d'un serveur est coûteux en énergie et entretien. L'objectif de l'approche proposée ici est de limiter les dépenses de fonctionnement de ces services.

Ainsi notre approche a pour objectif de calculer la *configuration optimale* d'un service Internet, à la fois en termes de performances mais aussi en termes de coût. Pour cela nous proposons une approche à deux niveaux : calcul de la *configuration architecturale optimale* et le calcul de la *configuration locale optimale*.

3.2. Propriétés

Nous décrivons ci-dessous quelques propriétés sous-jacentes au système considéré. Ces propriétés sont nécessaires à la compréhension de la suite de l'article.

Quantité de charge. La quantité de charge d'un service Internet fait référence au nombre de requêtes de clients concurrents accédant au service. Cette quantité peut être variable dans le temps.

Classe de charge. Il peut exister des classes de charge de service Internet de différentes natures. Par exemple une classe de charge constituée des requêtes clients, n'effectuant que des lectures de données, et une classe de charge de requêtes effectuant des mises à jour de données. Ces classes de charge reflètent des comportements changeants dans l'utilisation du service Internet.

Homogénéité des ressources. Les services Internet multi-niveaux sont généralement déployés sur des grappes de machines (i.e. ressources) homogènes. En particulier, toutes les machines d'un même niveau dans un service multi-niveaux sont de même architecture matérielle.

Répartiteur de charge. Pour répartir la charge entre les différentes resources dupliquées d'un niveau dans un système multi-niveaux, plusieurs algorithmes peuvent être appliqués (en pratique les algorithmes les plus déployés sont les algorithmes les plus simples, (ex. *round-robin*). Dans la suite, nous considérons une répartition de charge équitable entre les différents réplicas d'un même niveau.

3.3. Configuration architecturale

La configuration architecturale d'un service Internet multi-niveaux définit le degré de duplication de chaque niveau du service, c'est-à-dire le nombre de ressources affectées à chaque niveau. Par exemple, dans la figure 3.b, la configuration architecturale du service Internet à trois niveaux est définie par un 3-uplet $CA = [r_1, r_2, r_3]$ où, $r_1 = 1$, $r_2 = 2$ et $r_3 = 2$. Donc, le service utilise au total k ressources avec $k = \sum_{i=1}^{M} r_i$, où M est le nombre de niveaux du service. Une configuration architecturale est dite optimale si, pour une charge donnée, elle garantit le contrat de qualité de service requis (i.e. latence requêtes client bornée) avec le minimum de ressources Notons CA^* une configuration architecturale optimale pour une charge donnée et k^* le nombre total de ressources d'une configuration architecturale optimale.

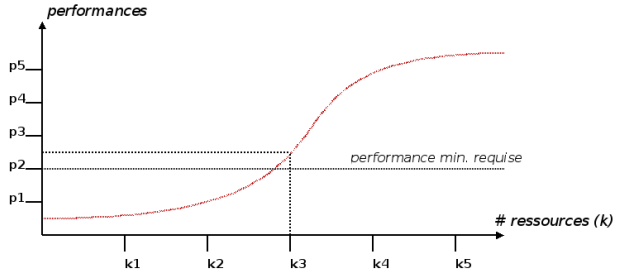


FIG. 4 – Performances d'un service en fonction de la quantité de ressources, pour une charge donnée

La figure 4 illustre, pour une charge donnée, la variation des performances d'un système en fonction du nombre total de ressources allouées au système. Dans cet exemple, pour garantir la performance minimale requise (p_2), nous constatons qu'il faut au minimum k_3 ressources. Si le système dispose de moins de k_3 ressources, le contrat de qualité de service n'est pas respecté. Inversement, si le système dispose de plus de k_3 ressources, certaines ressources peuvent être gaspillées au regard du contrat minimal de qualité de service. Dans ce cas, $k^* = k_3$.

Notre approche de calcul de configuration architecturale optimale se base :

– d'une part sur une modélisation du service Internet qui permet de calculer, pour une configuration architecturale donnée et une charge donnée les performances du système

– et d'autre part sur un algorithme de recherche d'une configuration architecturale optimale pour un système avec une charge donnée.

3.3.1. Modélisation de la configuration architecturale

La modélisation du système au niveau architectural suit l'approche proposée dans [10]. Cette approche utilise un réseau de files d'attente pour modéliser l'application (voir figure 5). Le réseau de files sert à représenter une application multi-niveaux. Chaque file représente un serveur de l'application, et les requêtes passent d'une file à l'autre (i.e. d'un niveau à l'autre) suivant des lois probabilistes.

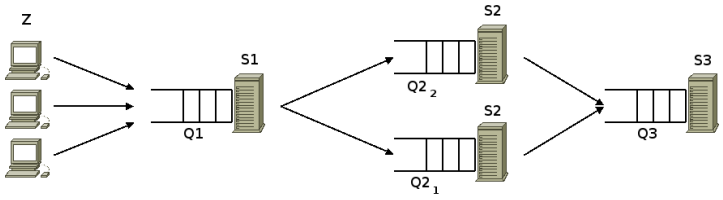


FIG. 5 – Exemple de réseau de files d'attente

Sur la base de ce modèle, l'algorithme MVA (*Mean Value Analysis*) est utilisé pour calculer le temps de réponse moyen (latence) perçu par l'utilisateur final à l'envoi d'une requête [7]. Ainsi, pour une application multi-niveaux constituée de M niveaux, l'algorithme MVA prend les paramètres suivants en entrée :

– M : Nombre de niveaux de l'application.
– N : Quantité de charge (nombre de sessions clientes) à l'entrée du service Internet multi-niveaux.
– $\bar{S}_m (1 \leq m \leq M)$: Temps de traitement moyen à chaque niveau (ou temps de service).
A chaque niveau de l'application, les requêtes sont retardées d'une durée correspondant au temps de service de la requête à ce niveau. Ce temps de service dépend, par exemple, du temps processeur requis pour le traitement de la requête sur ce serveur. Ce temps de service est calculé à faible charge, lorsqu'il n'y a pas de contention sur le serveur.
– $V_m (1 \leq m \leq M)$: ratio de visite au niveau m.
Le ratio de visite d'un niveau d'une application multi-niveaux est le nombre moyen de fois que ce niveau est appelé pendant l'exécution d'une requête.
– $\bar{Z}$: temps d'attente moyen du client (ou *think time*).
Le temps d'attente moyen du client est le temps moyen passé entre le moment où le client reçoit la réponse à une requête et le moment où ce même client envoie la requête suivante. Ce temps d'attente moyen montre à quelle vitesse les clients envoient leurs requêtes au service.
– $[r_1 \dots r_M]$: une configuration architecturale du système.

Une fois le modèle correctement paramétré, l'algorithme MVA est capable de prédire le temps de réponse moyen aux requêtes clients pour la configuration architecturale considérée (paramètre à faire varier jusqu'à trouver la valeur optimale). Notons que dans les paramètres passés à l'algorithme MVA, N définit la quantité de charge, et $\bar{S}_m$, V_m et $\bar{Z}$ définissent la nature de la charge.

3.3.2. Algorithme de calcul d'une configuration architecturale optimale

Nous proposons un algorithme de calcul d'une configuration architecturale optimale. Cet algorithme prend en entrée le nombre de niveaux M d'une application multi-niveaux, une quantité et une classe de charge données pour l'application (c.f. section 3.2) et produit en sortie la configuration architecturale optimale qui garantit une performance requise. L'algorithme est constitué de deux phases principales. La première phase a pour objectif de calculer une première configuration architecturale qui garantisse l'objectif de performance fixé, même si cette configuration peut ne pas être économe en terme de ressources utilisées. La deuxième phase de l'algorithme a pour objectif, en partant de cette première configuration architecturale, de réduire le nombre de ressources utilisées à leur strict minimum pour que les deux objectifs de performance et de coût soient garantis (c.f. section 3.1), ce qui résulte en une configuration archicturale optimale. Par exemple, en considérant le système illustré par la figure 4, la première phase de notre algorithme peut produire une configuration architecturale dont le nombre total de ressources est k_5. Puis la seconde phase de l'algorithme permet de réduire ce nombre à k_3 dans une configuration

optimale. Dans la deuxième phase de l'algorithme, la recherche du degré de duplication optimal se fait par dichotomie pour chaque niveau de l'application, ce qui a un coût logarithmique.

Algorithme 1 : Algorithme de configuration architecturale

Entrées : M, N, classe_charge ($\bar{S}_m$, V_m, $\bar{Z}$), latence_max
Sorties : $CA^* = [r_1^* \dots r_M^*]$ (configuration architecturale optimale)

```
1  début
2      [r_1* ... r_M*] = [1 ... 1] ;
3      /* Phase 1 : garantie de performances */
4      latence = MVA(M,N,classe_charge, [r_1 ... r_M]) ;
5      tant que latence>latence_max faire
6          [r_1 ... r_M] = [r_1 + 1 ... r_M + 1] ;
7          latence = MVA(M,N,classe_charge, [r_1 ... r_M]) ;
8      /* Phase 2 : minimiser le coût */
9      pour tous les m = M ;m ≥ 1 ;m − − faire
10         /* recherche dichotomique de r_m* dans l'intervalle [1 ... r_m] */
11         latence = MVA(M,N,classe_charge, [r_1 ... r_m/2 ... r_M]) ;
12         si latence > latence_max alors
13             poursuivre recherche dichotomique de r_m* dans l'intervalle ]r_m/2 ... r_m]
14         sinon
15             poursuivre recherche dichotomique de r_m* dans l'intervalle [1 ... r_m/2]
16         A la fin de la recherche dichotomique au niveau m : [r_1 ... r_m* ... r_M*]
17     retourner [r_1* ... r_M*] ;
18 fin
```

3.4. Configuration locale

L'objet de la configuration locale proposée ici est d'augmenter la rentabilité des serveurs utilisés par un service multi-niveaux. Autrement dit, on souhaiterait trouver une configuration locale des serveurs qui maximise l'usage des ressources (i.e. performances système telles que le débit des serveurs ou leur vitesse de traitement des requêtes) Nous nous intéressons plus particulièrement à la configuration du degré de parallélisme maximal dans un serveur (c.f. section 2). Notre objectif est de trouver le degré de parallélisme optimal qui garantisse au système les meilleures performances tout en lui évitant l'écroulement.

Nous utilisons la modélisation proposée dans [3], et illustrée sur la figure 6. Cette figure modélise le comportement classique d'un serveur multiprogrammé, en terme de variation de sa performance en fonction de la quantité de charge. Nous constatons qu'en l'absence de limitation du degré de parallélisme dans le serveur, lorsque la quantité de charge augmente, le serveur passe par trois phases : (i) une phase de sous-charge, (ii) une phase d'utilisation optimale et maximale du serveur, et enfin (iii) une phase de surcharge jusqu'à l'écroulement du serveur. Idéalement, le serveur devrait pouvoir fournir une performance allant jusqu'à *perf_max* sans écroulement du système. Pour cela, le degré de parallélisme du serveur devrait être fixé à l'antécédent de *perf_max*. Cette dernière valeur représente la configuration locale optimale du serveur.

Le modèle utilisé est une approche par approximation parabolique . Elle consiste à trouver les paramètres de la parabole se rapprochant le plus du comportement d'un serveur multiprogrammé. L'équation standard d'une parabole est de la forme $y = ax^2 + bx + c$. Comme nous cherchons à représenter les performances du système en fonction de la quantité de charge, nous pouvons écrire :

$$\text{performances} = a \times \text{quantite\_charge}^2 + b \times \text{quantite\_charge} + c \quad (1)$$

A noter que les valeurs des paramètres a, b et c peuvent changer en fonction des variations de la classe de charge du serveur. Ainsi le calcul de la configuration locale optimale (CL^*) revient à calculer l'antécédent du maximum, qui est $CL^* = \frac{-b}{2a}$.

Pour calibrer notre modèle, nous faisons fonctionner le serveur instrumenté pour récupérer des informations sur les performances du système, en fonction de la quantité de charge. Une fois les données obtenues,

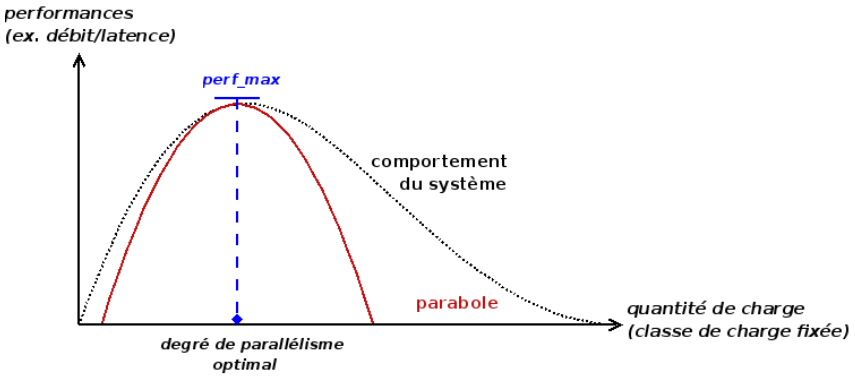


FIG. 6 – Approximation parabolique des performances d'un serveur multiprogrammé (d'après [3])

nous utilisons la méthode des moindres carrés pour trouver les paramètres a, b et c qui minimiseront l'écart moyen entre la parabole et les données mesurées.

3.5. Configuration globale

Nous avons combiné la configuration architecturale d'un service Internet multi-niveaux à la configuration locale des différents serveurs composant ce service, dans une configuration que nous appelons globale. A notre connaissance, c'est la première fois que ces types de configuration ont été combinés, dans le but d'optimiser les performances et l'usage des ressources dans un service Internet. La charge des services Internet étant variable dans le temps, une configuration (globale) optimale à un instant donné peut ne plus l'être à l'instant suivant. Il faut alors recalculer la nouvelle configuration globale optimale, c'est-à-dire la nouvelle configuration architecturale optimale et la nouvelle configuration locale optimale de chaque serveur. Nous constatons que la configurations architecturale d'un service multi-niveaux peut être impactée par la quantité ou la classe de charge reçue par le service (c.f. section 3.3). Autrement dit, si un de ces deux paramètres change, la configuration architecturale optimale peut changer. Quant à la configuration locale, elle est impactée par la classe de charge uniquement (c.f. section 3.4). L'algorithme 2 décrit la combinaison de ces deux types de configuration dynamique.

Algorithme 2 : Configuration globale

Entrées : M, N_{t-1}, classe_charge$_{t-1}$, $[CA^*, [CL_1^* \dots CL_M^*]]_{t-1}$ (configuration globale)

```
1  début
2      /* Reconfiguration locale */
3      si classe_charge_t ≠ classe_charge_{t-1} alors
4          [CL_1^* ... CL_M^*]_t = calcul de la nouvelle configuration locale optimale de chaque serveur
5          appliquer les nouvelles configurations
6      /* Reconfiguration architecturale */
7      si classe_charge_t ≠ classe_charge_{t-1} ou N_t ≠ N_{t-1} alors
8          CA_t^* = calcul de la nouvelle configuration architecturale optimale
9          appliquer la nouvelle configuration
10 fin
```

4. Evaluation expérimentale

4.1. Environement d'évaluation

Pour nos expériences, nous avons utilisé dans la plate-forme Grid'5000 la grappe de machines située à Lille. Le matériel est principalement composé de bi-processeurs AMD Opteron et Intel Xeon, reliés par un

réseau de communication à 10 Gb/s. Notre environnement logiciel était constitué des élément suivants : Linux v. 2.6.18 comme système d'exploitation, le serveur de Servlets (métier) Apache Tomcat v. 5.5.23, le serveur de bases de données MySQL v. 5.0.37 et le répartiteur de charge de bases de données Sequoia v. 2.10.6. Nous utilisons une application multi-niveaux de commerce électronique, et plus précisément une application de ventes aux enchères modélisant eBay. L'application RUBiS[1] (*Rice University Bidding System*) est un banc d'essai permettant 26 interactions, comme ajouter de nouveaux utilisateurs, ajouter des articles au panier, passer une commande, etc. La classe de charge étant paramétrable, nous avons utilisé le *browsing mix* (100% de requêtes Web en lecture seule) et le *bidding mix* (15% de requêtes Web en écriture). Un émulateur génère des clients Web qui envoient des requêtes de consultation, de vente et d'enchères sur le site Web via son interface. Chaque session d'un client est une suite d'interactions, séparées par un temps d'attente entre requêtes moyen de 7 secondes, conformément à la norme TPC-W[2]. Une phase de chauffe à lieu au début de chaque expérience, pendant laquelle on fait fonctionner le système sans collecter de mesures sur ses performances.

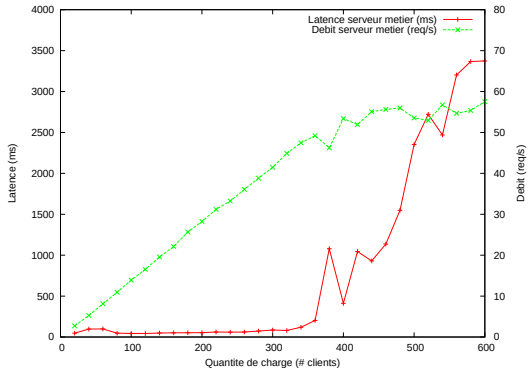


FIG. 7 – Latence et débit du serveur métier (Tomcat) - classe de charge en lecture *(browsing mix)*

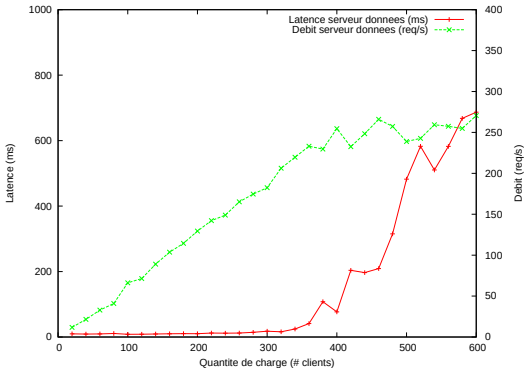


FIG. 8 – Latence et débit du serveur données (MySQL) - classe de charge en lecture *(browsing mix)*

4.2. Evaluation de la configuration locale

Dans le cadre de l'évaluation de la configuration locale, nous nous sommes basés sur l'application Rubis constituée d'un serveur au niveau métier et d'un serveur au niveau données. A noter que la configuration locale d'un serveur est orthogonale au nombre de duplicas dans l'application multi-niveaux. Notre objectif était de calculer la configuration locale optimale du serveur métier et la configuration locale optimale du serveur de données, en considérant différentes classes de charge de l'application. Une première phase dans nos expériences a été de calibrer le modèle sous-jacent à la configuration locale et ceci pour chacune des classes de charge de l'application.

Donc soit f la fonction à déterminer, ici $f(x) = ax^2 + bx + c$. Les paramètres à déterminer sont donc a, b et c. On va chercher les valeurs de ces paramètres qui minimiseront la somme des écarts au carré : $S = \sum_{i=1}^{n}(y_i - f(x_i))^2$

Les y_i sont des valeurs mesurées sur le système réel. Pour obtenir ces valeurs, nous avons développé des outils de mesures de performances (latence, débit) que nous avons appliqué au système. Nous avons ensuite fait fonctionner le système à modéliser avec une classe de charge et en faisant varier la quantité de charge pour couvrir l'ensemble de l'espace d'entrée des paramètres. Les résultats de ces mesures de performances sont décrits tout d'abord dans les figures 7 et 8, résultats de l'application Rubis avec une classe de charge de requêtes en lecture seule (*browsing mix*). Le nombre de clients augmente régulièrement par palier de 20 au cours de l'expérience. Chaque point représente une exécution de 5 minutes. La limite sur le taux de parallélisme maximal est fixée à une valeur très élevée sur les deux niveaux de l'application (1000 pour la base MySQL et 5000 pour le serveur Tomcat).

[1] http ://rubis.objectweb.org/

[2] http ://www.tpc.org

On voit qu'à partir d'un certain nombre de clients, le système commence à saturer. Le nombre de requêtes traitées n'augmente plus, et le temps de traitement des requêtes croit linéairement. Notre but va être de caractériser le nombre de client optimal devant être admis dans le système pour avoir les meilleurs performances (ratio débit/latence). Pour cela nous traçons la courbe de performance au niveau MySQL, qui correspond à ce ratio. la figure 9 présente ce ratio de performance. Il faut garder à l'esprit que la valeur du ratio de performance n'a pas de sens en soit, seule la forme de la courbe et donc la comparaison entre différentes valeurs de performance a un intérêt.

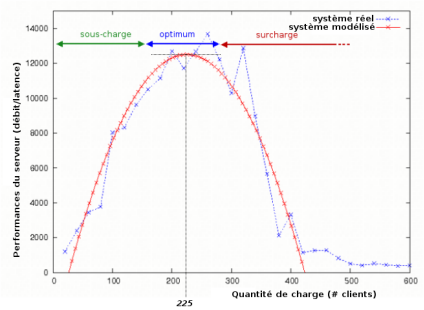


FIG. 9 – Performances du service au niveau données (MySQL) - classe de charge en lecture *(browsing mix)*

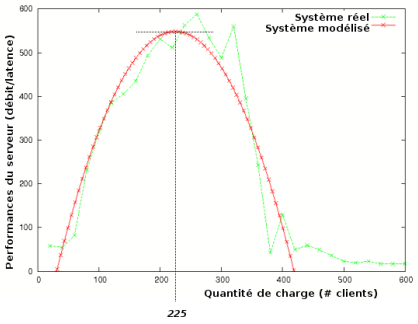


FIG. 10 – Performances du niveau métier (Tomcat) - *browsing mix*

Conformément à la théorie, ce ratio a la forme d'une parabole. Nous effectuons donc une regression quadratique pour obtenir les paramètres de cette parabole ($a = -0.3177, b = 142.9, c = -3553.5$) et déterminer l'antécédent du maximum. Dans notre cas, le maximum de performance est atteint avec un nombre de clients aux alentours de 225, ce qui correspond à une configuration locale optimale du serveur de données de $CL_2^* = \frac{-b}{2a} = 225$ clients maximum en parallèle.
Nous avons également trouvé une configuration locale optimale pour le serveur métier $CL_1^* = 225$. Ceci s'explique par le fait que la base de données constitue le goulot d'étranglement de l'application. Les performances du niveau métier sont donc limitées par celles du niveau données.

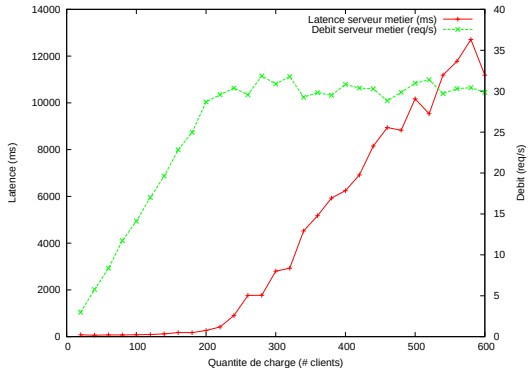


FIG. 11 – Latence et débit du serveur métier (Tomcat) - classe de charge en écriture *(bidding mix)*

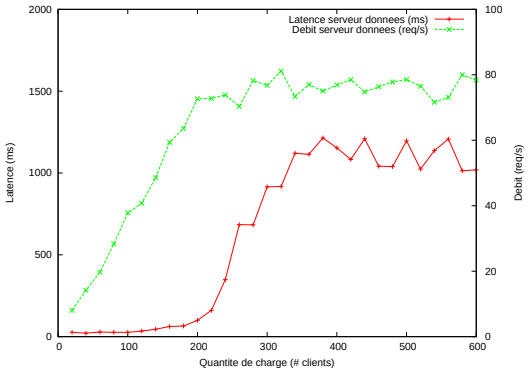


FIG. 12 – Latence et débit du serveur données (MySQL) - classe de charge en écriture *(bidding mix)*

De manière similaire, nous avons effectué ces expérimentations avec une autre classe de charge de l'application, comportant des requêtes en lecture et en écriture *(bidding mix)*. Les résultats sont présentés sur les figures 11 et 12. Ainsi, l'approximation parabolique donne les paramètres $a = -0.085, b = 20.8, c = -127.8$ pour la parabole. La configuration locale optimale a donc pour valeur $CL_2^* = 122$ clients maximum en parallèle.

Ainsi en passant de 0% de requêtes Web en écriture à 15%, le nombre de clients optimal à diminué de plus de 100 clients. Ceci s'explique par le fait que des lectures peuvent être plus facilement parallélisées que des mises à jour, car ces dernières nécessitent des verrous en accès sur la base de données. Nous avons peut donc résumer en disant que plus le nombre d'écritures est important, plus le nombre de clients optimal sera faible. Comme la proportion de lectures / écritures évolue au cours du temps, nous avons donc bien besoin d'une approche permettant de reconfigurer dynamiquement le degré de parallélisme maximal. La figure 14 illustre le déplacement des paraboles approximant la courbe de performance des serveurs MySQL avec des charges sans mises à jour *(browsing)* et avec 15% de mises à jour *(bidding)*.

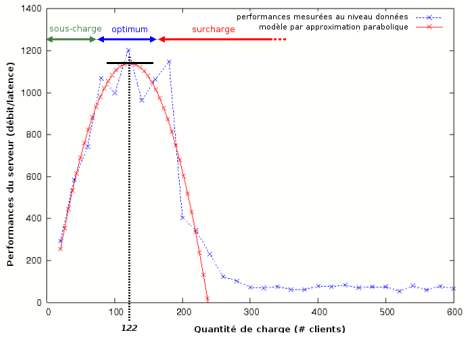


FIG. 13 – Performances du service au niveau données (MySQL) - classe de charge en écriture *(bidding mix)*

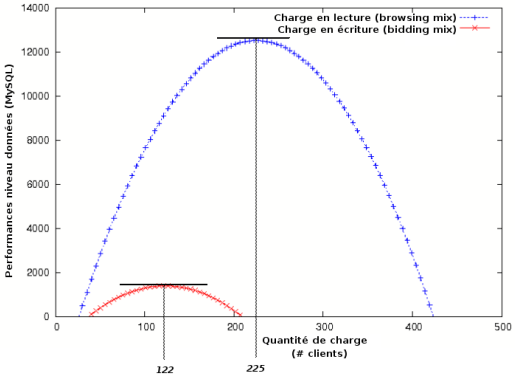


FIG. 14 – Comparaison des performances de différentes classes de charge

4.3. Evaluation de la configuration architecturale

Une première phase de cette évaluation a été de calibrer le modèle sous-jacent à la configuration architecturale de l'application multi-niveaux Rubis. Pour cela, nous avons soumis l'application à une faible charge et mesuré ses performances pour pouvoir produire les paramètres du modèle qui sont décrits dans le tableau 1.

Paramètre	Signification	Valeur
M	Nombre de niveaux dans l'application	2
N	Quantité de charge (# clients)	varie
$\bar{Z}$	Temps d'attente chez le client (think time)	7 s
$\bar{S}_1$	Temps de service au niveau métier	13,876
$\bar{S}_2$	Temps de service au niveau données	11,8
V_1	Ratio de visite au niveau métier	1
V_2	Ratio de visite au niveau données	2,68
$CA = [r_1, r_2]$	Configuration architecturale de l'application multi-niveaux	$[1, 1]$

TAB. 1 – Calibration du modèle de configuration architecturale

Une fois le modèle paramétré, nous pouvons comparer la prédiction du modèle avec les données mesurées. La figure 15 présente le temps de réponse du système en fonction du nombre de clients, ainsi que la prédiction du modèle pour ce même nombre de client. On peut vérifier que le modèle concorde bien avec la réalité, car l'écart moyen entre la prédiction du modèle et la mesure effectuée sur le système réel est de 242 ms, ce qui donne une erreur moyenne de 12% environ. Cette erreur est en partie due au temps d'expérience pour les mesures, chaque point représentant la moyenne des latences sur une exécution de 7 minutes. Comme RUBiS envoie des requêtes ayant des durées d'exécution variables, la

latence moyenne peut donc fluctuer au cours d'une expérience si sa durée n'est pas assez longue. En diminuant le nombre de points de mesure mais en augmentant la durée d'expérience à chaque point nous obtiendrions probablement une erreur moyenne moindre. On peut également trouver sur cette figure une prédiction de la configuration architecturale optimale en fonction de la charge, pour un objectif de latence à 2000 ms maximum. En dessous de 300 clients, un seul serveur au niveau données permet d'atteindre l'objectif de qualité de service. Au dessus de cette valeur, un serveur supplémentaire est nécessaire.

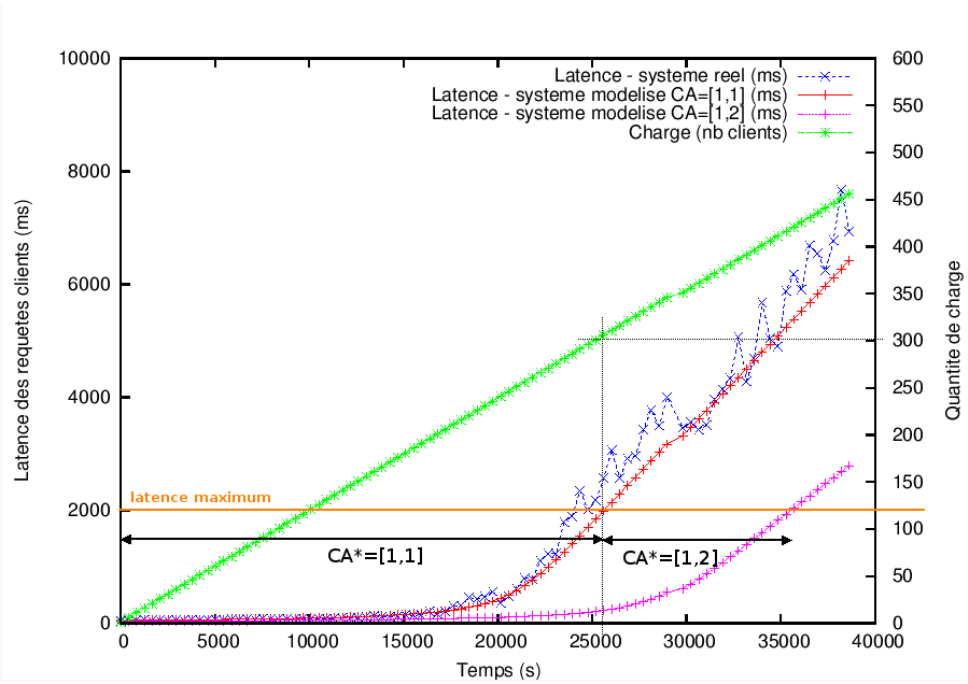


FIG. 15 – Comparaison des performances du système réel et du système modélisé

5. Etat de l'art

Afin d'améliorer les performances des services Internet, deux types d'approches sont généralement envisagés. La première approche consiste, lors de surcharge, à limiter le nombre de clients admis dans le système et à traiter seulement les tâches les plus importantes. La plupart des travaux suivant cette approche utilisent un proxy pour filtrer les requêtes en entrée dans le système [2]. Une seconde approche est d'adapter dynamiquement le nombre de ressources affectées au service en fonction de la charge. Si ces deux types d'approches ne sont pas incompatibles, elles ont néammoins des objectifs assez différents. L'approche que nous proposons s'inscrit dans la seconde voie, celle de l'allocation dynamique de ressources. Les différentes recherches effectuées dans ce domaine peuvent se classer suivant différents critères.

Un premier est le type de système pris en compte. Jusqu'à maintenant, la plupart des travaux portent sur des systèmes simples composés d'un seul serveur [3, 6], ou alors d'un serveur répliqué [5]. La prise en compte d'applications plus complexes de type multi-niveaux n'est apparue que récemment avec des approches basées sur l'utilisation de réseaux de files d'attente pour modéliser le système administré [10, 8]. Les réseaux de files d'attente sont à la fois simples d'utilisation et suffisament puissants pour caractériser des systèmes complexes. L'approche que nous proposons dans cet article se situe dans cette dernière catégorie. Les méthodes de reconfiguration évoluent également. En effet la plupart des méthodes réactives *ad hoc* employées jusqu'alors sont efficaces mais ne permettent pas d'anticiper les évolutions des performances du système [9]. Le besoin de modèles mathématiques permettant de prédire le comportement des systèmes a donné lieu à de nombreux travaux qui ont permis de développer des modèles plus ou moins complexes des systèmes à administrer, ainsi que la définition d'une terminologie précise proche de l'automatique [1, 5, 4, 6].

Enfin la grande majorité des approches se placent soit au niveau architectural, en cherchant à optimiser le nombre de ressources allouées au système [10], soit au niveau local en cherchant à optimiser les paramètres de configuration des serveurs de l'application [3]. Notre approche se différencie de [10] dans l'algorithme de recherche de la configuration architecturale optimale. Dans [10], l'algorithme est basé sur une recherche exhaustive des différentes configurations, alors que l'algorithme que nous proposons est basé sur une recherche dichotomique et donc plus efficace (avec un facteur allant jusqu'à M^2, M étant le nombre de niveaux de l'application).
Par ailleurs, le prototype Middle-R [5] prend en compte les deux niveaux (local et architectural) de l'application, mais influe uniquement sur le paramétrage de l'application et n'effectue pas d'allocation de ressources. Ainsi, dans l'état actuel de nos connaissances, il n'existe pas d'approche ayant intégré à la fois l'optimisation de la configuration locale de chaque serveur et l'optimisation de l'allocation de ressources au niveau architectural. Notre algorithme global, en synchronisant deux modèles de l'application pour deux niveaux d'optimisation, permet de répondre à ce besoin.

6. Conclusion

Nous avons présenté dans cet article une approche permettant d'améliorer les performances d'un service Internet multi-niveaux, tout en économisant le nombre de ressources allouées à ce service. Pour cela, nous avons développé un algorithme permettant de calculer de manière efficace une configuration optimale d'un service Internet. Cette configuration optimale concerne à la fois un paramétrage optimal de chacun des serveurs composants l'application, ainsi que l'allocation du nombre de ressources minimal nécessaire pour respecter les contraintes de qualité de service de l'application considérée. Ces travaux ouvrent plusieurs perspectives, notamment l'application de cette approche pour des systèmes à très grande échelle, domaine dans lequel l'économie de ressources peut s'avérer particulièrement critique.

Bibliographie

1. 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.
2. S. Elnikety, E. Nahum, J. Tracey, and W. 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*, New York, NY, USA, 2004.
3. H-U. Heiss and R. Wagner. Adaptive load control in transaction processing systems. In *VLDB*, 1991.
4. D.A. Menascé and V. Almeida. *Capacity Planning for Web Services : metrics, models, and methods.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
5. J.M. Milan-Franco, R. Jiménez-Peris, M. Patino-Martinez, and B. Kemme. Adaptive middleware for data replication. In *Middleware '04 : Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, New York, NY, USA, 2004.
6. S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real Time Systems Journal*, 23(1-2), 2002.
7. M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *J. ACM*, 27(2), 1980.
8. S. Sivasubramanian, P. Guillaume, M. van Steen, and S. Bhulai. Sla-driven resource provisioning of multi-tier internet applications. Technical report, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 2006.
9. C. Taton, S. Bouchenak, N.D. Palma, D. Hagimont, S. Krakowiak, and J. 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.
10. B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. Analytic modeling of multitier internet applications. *ACM Trans. Web*, 1(1) :2, 2007.
11. B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms, 2002.