Rien ne sert de prédire ; il faut servir ancien.

Céline Comte

Rien ne sert de prédire ; il faut servir ancien.

Céline Comte[1,2] †

[1] *Nokia Bell Labs France, Nozay, France*
[2] *Télécom ParisTech, Paris, France*

Optimiser l'utilisation des centres de données requiert des algorithmes efficaces et fiables pour répartir la charge entre les machines en s'adaptant à des contraintes variées, telles que la localité des données, qui restreignent les affectations possibles. Dans ce papier, nous proposons un algorithme simple qui réalise cette tâche sans avoir besoin de connaître le taux d'arrivée des requêtes ni la capacité des machines. Sa propriété essentielle, vérifiée à l'aide d'un modèle de files d'attente, est son insensibilité à la loi de la taille des requêtes : si chaque machine applique la discipline de service processor sharing et si les requêtes arrivent selon un processus de Poisson, alors la performance ne dépend de la loi de la taille des requêtes que par l'intermédiaire de sa moyenne. De premiers résultats numériques évaluant les taux de rejet des requêtes et d'occupation des machines indiquent de plus que cet algorithme surpasse le meilleur algorithme statique et est proche d'un algorithme idéal qui optimiserait constamment l'utilisation des ressources.

Une version longue de cet article a été présentée à IFIP Networking 2018 [Com18].

Mots-clefs : Répartition de la charge, contraintes d'affectation, insensibilité, théorie des files d'attente

1 Introduction

Les politiques de répartition de charge *équilibrées* ont été identifiées dans [BJP04] comme rendant la performance insensible à la loi de la taille des requêtes lorsque celles-ci arrivent selon un processus de Poisson et chaque machine applique la discipline de service processor sharing. De telles politiques donnent des intuitions robustes sur la performance que l'on peut attendre d'un système réel. Cependant, à l'exception de [AW12] qui supposait que chaque machine ne pouvait traiter qu'une seule requête à la foi, aucun algorithme déterministe n'a été proposé pour *réaliser* une politique équilibrée dans laquelle la décision d'affectation dépend de l'état du système. Notre première contribution, décrite en partie 2, est un algorithme simple à base de jetons qui réalise une telle politique dans le cas général où chaque machine peut traiter plusieurs requêtes en parallèle. Notre deuxième contribution, décrite en partie 3, est un modèle de files d'attente construit en appliquant des modèles existants de façon atypique. Ce modèle certifie l'insensibilité de l'algorithme et fournit des formules explicites pour prédire sa performance dans de petites grappes (jusqu'à dix machines environ). La partie 4 applique ces formules pour évaluer le gain de performance de notre algorithme par rapport à des politiques équilibrées où la décision d'affectation est indépendante de l'état du système.

2 Algorithme

Problème On considère une grappe de machines où des requêtes arrivent à des instants aléatoires. Un unique répartiteur affecte chaque requête à une machine à l'instant où elle arrive. Chaque machine applique la discipline de service processor sharing et admet une limite supérieure sur le nombre de requêtes qu'elle peut traiter en parallèle. Chaque requête a des contraintes qui restreignent l'ensemble des machines (dites *compatibles*) auxquelles elle peut être affectée. On suppose que le répartiteur peut déterminer les compatibilités d'une requête à son arrivée. L'objectif est de proposer un algorithme simple pour répartir la charge entre les machines, sans connaître à l'avance les compatibilités possibles des requêtes ni leur taux d'arrivée.

† Ce travail a été mené au LINCS, voir `https://www.lincs.fr`.

Solution Chaque machine envoie un jeton au répartiteur à chaque fois qu'elle termine de traiter une requête. Le répartiteur conserve ces jetons dans une liste ordonnée où les jetons les plus anciens sont en tête. Lorsqu'une requête arrive, le répartiteur : (i) vérifie ses compatibilités, (ii) parcourt la liste pour trouver le jeton le plus ancien qui identifie une machine compatible, (iii) affecte la requête à cette machine et supprime ce jeton de la liste. Si la liste ne contient aucun jeton compatible, la requête est rejetée. Lorsqu'une requête est compatible avec toutes les machines, l'algorithme l'affecte à la machine identifiée par le jeton le plus ancien. Lorsqu'une machine est vide, la liste contient autant de jetons que cette machine peut traiter de requêtes en parallèle. Au démarrage, les jetons, tous libres, sont placés dans la liste dans un ordre arbitraire.

Intuition Une méthode classique pour répartir la charge consiste à affecter chaque requête à la machine qui contient le moins de requêtes (ou la plus petite charge de travail). Notre approche est différente. Une machine a, certes, plus de chances de se voir affecter une requête si elle a plus de jetons dans la liste, mais la décision d'affectation dépend aussi de l'*ordre* dans lequel les jetons ont été reçus. Intuitivement, un jeton plus ancien qu'un autre identifie probablement une machine moins chargée ; il vaut donc mieux affecter une requête entrante à cette machine si elle est compatible afin de laisser les autres machines, probablement plus chargées, à des requêtes ultérieures. La propriété principale de cet algorithme, analysée en partie 3, est son insensibilité à la loi de la taille des requêtes. Des résultats numériques évaluent sa performance en partie 4.

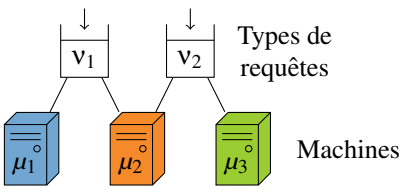
3 Analyse

Grappe de machines Les compatibilités des requêtes sont décrites par un graphe biparti appelé *graphe des compatibilités*. Plus précisément, on note I le nombre de machines et K le nombre de types de requêtes possibles ; le graphe contient une arête entre un type et une machine si les requêtes de ce type sont compatibles avec cette machine, comme illustré en FIGURE 1a. On néglige le temps de communication entre le répartiteur et les machines ainsi que le temps que le répartiteur passe à prendre la décision d'affectation (de sorte qu'une requête entre en service immédiatement après sont arrivée dans le système si elle est acceptée).

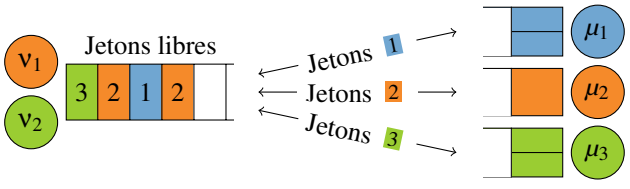
Nos hypothèses sur les statistiques du trafic sont les suivantes. Pour chaque $k = 1, \dots, K$, les requêtes de type k arrivent selon un processus de Poisson indépendant d'intensité positive ν_k. Les tailles des requêtes suivent une loi générale de moyenne unitaire. On suppose que cette loi ne dépend pas du type de la requête mais peut dépendre de la machine sur laquelle elle est traitée. Pour chaque $i = 1, \dots, I$, on note μ_i la capacité de service, positive, de la machine i et ℓ_i le nombre maximum de requêtes qu'elle peut traiter en parallèle.

Réseau de files d'attente Le modèle repose sur une interprétation légèrement différente de la notion de jeton. On suppose que chaque machine $i = 1, \dots, I$ possède exactement ℓ_i jetons qui se déplacent tour à tour entre cette machine et la liste du répartiteur. Chaque requête est affectée à la machine dont elle saisit un jeton et libère ce jeton lorsqu'elle quitte la machine à la fin de son service. Le temps qu'un jeton passe dans une machine est donc égal au temps de service de la requête tenant ce jeton. La structure du réseau de files d'attente, illustrée en FIGURE 1b, décrit le mouvement des jetons :

— une file d'attente M/M/1/ℓ_i de capacité μ_i, sous la discipline de service processor sharing, contient les jetons tenus par les requêtes en service à la machine i, pour chaque $i = 1, \dots, I$;
— une file d'attente multi-serveur, dont le fonctionnement est détaillé ci-dessous, contient les jetons libres ; son évolution reproduit celle de la liste des jetons libres au niveau du répartiteur.



(a) Graphe des compatibilités



(b) Réseau de files d'attente. Chaque machine a trois jetons. La couleur d'un serveur dans la première file est celle du jeton qu'il est en train de servir.

FIGURE 1: Modèle

Rien ne sert de prédire ; il faut servir ancien.

Cette dernière file contient autant de serveurs qu'il y a de types de requêtes. Considérons un serveur donné $k = 1, \ldots, K$. Un jeton est *servi* par ce serveur lorsqu'il est saisi par une requête de type k dans la liste du répartiteur ; il est *en service* sur ce serveur s'il est destiné à être saisi par la prochaine requête entrante de type k. Le serveur k applique la discipline de service premier arrivé, premier servi aux jetons des machines qui sont compatibles avec les requêtes de type k, en ignorant les autres jetons. L'hypothèse des arrivées poissonniennes garantit que le temps de service sur ce serveur suit une loi exponentielle de taux ν_k.

Insensibilité L'état du réseau est décrit par la séquence des jetons libres, rangés par ordre d'arrivée. Si la taille des requêtes suit une loi exponentielle de moyenne unitaire, alors cet état définit un processus de Markov irréductible dont la distribution stationnaire est donnée dans [Com18, Partie 3.3] en s'appuyant sur des résultats classiques de théorie des files d'attente. Si la taille des requêtes suit une loi quelconque de moyenne unitaire, alors le processus défini par l'état du réseau n'a pas, en général, la propriété de Markov. On peut cependant montrer que sa distribution en régime stationnaire reste inchangée ; la preuve pour des lois coxiennes, denses dans l'ensemble des lois de variables aléatoires positives, étend celle de [BP03].

4 Résultats numériques

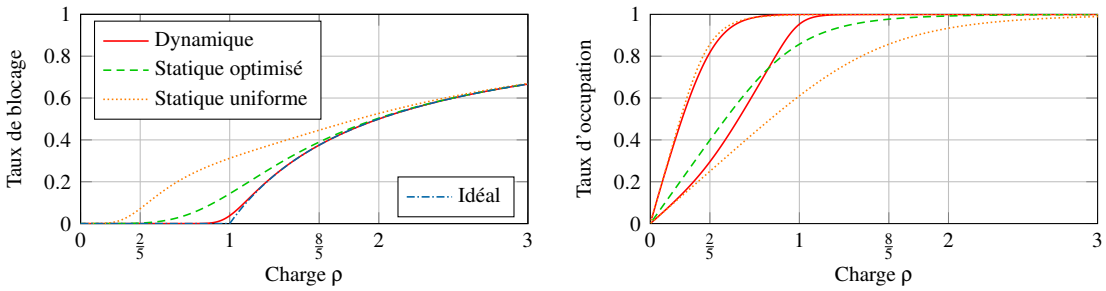
Paramètres Le nombre limité de requêtes qui peuvent être en service sur chaque machine garantit un taux de service minimum. On s'intéresse donc à deux autres métriques de performance, le taux de blocage des requêtes et le taux d'occupation de chaque machine. Cette dernière métrique, définie comme la fraction du temps que chaque machine passe à traiter des requêtes, est un bon indicateur de leur charge effective. Ces deux métriques sont tracées en fonction de la charge ρ, égale au rapport entre le taux d'arrivée des requêtes et la capacité des machines. Les hypothèses sur les statistiques du trafic sont comme en partie 3.

Notre algorithme, appelé *dynamique* dans la suite, est comparé à deux algorithmes (insensibles) dits statiques car ils affectent chaque requête à une machine compatible choisie au hasard, indépendamment de l'état du système, et la rejettent si cette machine est pleine. Les probabilités d'affectation sont uniformes sous *statique uniforme* et fixées de manière à égaliser les charges des machines autant que possible sous *statique optimisé*. Cette dernière solution nécessite de prédire les taux d'arrivée relatifs des requêtes.

Machines hétérogènes On considère une grappe de dix machines avec un seul type de requêtes, sans contrainte d'affectation. La moitié des machines a une capacité μ et l'autre moitié une capacité 4μ. Chaque machine peut traiter jusqu'à six requêtes en parallèle. Les résultats numériques sont regroupés en FIGURE 2.

Le taux de blocage sous *dynamique* est proche du taux *idéal*, celui que l'on peut espérer obtenir en optimisant constamment l'utilisation des ressources. Par rapport à *statique optimisé*, le gain de *dynamique* est maximal autour de la charge critique $\rho = 1$; c'est aussi l'endroit où sa différence avec *idéal* est la plus élevée. La performance de *statique uniforme* se détériore plus vite car les charges des machines sont déséquilibrées. Ceci souligne le besoin de prédire les paramètres du système pour rendre efficace une politique statique.

Toutes les machines ont le même taux d'occupation sous *statique optimisé* car leur charge est la même. Au contraire, sous *dynamique* et *statique uniforme*, le taux d'occupation des cinq premières machines (tracé



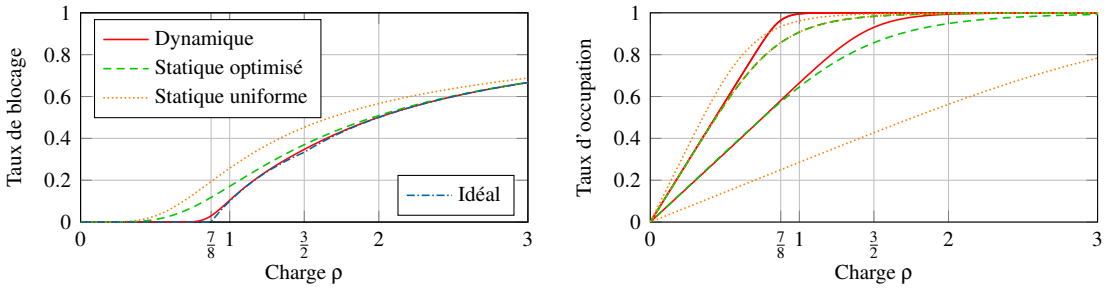
(a) Taux de bloquage des requêtes

(b) Taux d'occupation des machines.

FIGURE 2: Performance de l'algorithme de répartition de charge avec des machines hétérogènes.

du haut dans la FIGURE 2b)) est plus élevé que celui des cinq dernières (tracé du bas). Remarquons aussi que les taux sous *dynamique* et sous *statique uniforme* semblent avoir la même tangente en $\rho = 0$. Intuitivement, à très faible charge, (presque) chaque jeton est relâché avant que le jeton suivant ne soit saisi, de sorte que la distribution réalisée par *dynamique* est approximativement uniforme et indépendante des charges relatives des machines. La performance de *dynamique* s'améliore cependant à mesure que la charge ρ augmente.

Contraintes d'affectation Considérons maintenant une grappe de dix machines de même capacité μ et pouvant chacune traiter jusqu'à six requêtes en parallèle. Il y a deux types de requêtes. Celles de type 1 (resp. 2) arrivent au taux ν (resp. $\nu/4$) et sont compatibles avec les sept premières (resp. dernières) machines. Seules les quatre machines centrales sont partagées. Les résultats numériques sont regroupés en FIGURE 3.



(a) Taux de bloquage des requêtes

(b) Taux d'occupation des machines

FIGURE 3: Performance de l'algorithme de répartition de charge avec des contraintes d'affectation.

Les remarques concernant le taux de blocage sont similaires à celles du scénario précédent. La différence principale est que, même sous *idéal*, la performance se dégrade dès que la charge ρ dépasse $7/8$ à cause des contraintes d'affectation : les sept premières machines supportent au moins $4/5$ème des arrivées avec seulement $7/10$ème de la capacité de service, donc leur charge effective est au moins $8\rho/7$.

Sous *dynamique*, le taux d'occupation des sept premières machines augmente comme $8\rho/7$ et celui des trois dernières comme $2\rho/3$. Ceci suggère que la distribution de charge réalisée par *dynamique* est la même que celle réalisée par *statique optimisé*. En particulier, *dynamique* parvient à égaliser les charges effectives des sept premières machines bien que celles-ci aient des compatibilités différentes.

5 Conclusion

Nous avons conçu et évalué un algorithme insensible qui répartit la charge entre les machines d'une grappe en s'adaptant aux contraintes d'affectation des requêtes. La simplicité de cet algorithme laisse la place à de nombreuses variations que nous avons commencé à explorer dans [Com18]. Nous souhaitons maintenant mieux comprendre comment sa performance se compare à celle d'algorithmes non-insensibles.

Remerciement Un grand merci à Fabien Mathieu pour avoir suggéré ce titre original et adapté au sujet.

Références

[AW12] I. Adan and G. Weiss. A loss system with skill-based servers under assign to longest idle server policy. *Probability in the Engineering and Informational Sciences*, 26(3) :307–321, 2012.

[BJP04] T. Bonald, M. Jonckheere, and A. Proutière. Insensitive load balancing. *SIGMETRICS Perform. Eval. Rev.*, 32(1) :367–377, June 2004.

[BP03] T. Bonald and A. Proutière. Insensitive bandwidth sharing in data networks. *Queueing Syst.*, 44(1) :69–100, 2003.

[Com18] C. Comte. Dynamic Load Balancing with Tokens. In *17th International IFIP TC6 Networking Conference Networking 2018*, pages 343–351, Zurich, Switzerland, May 2018.