Prédiction d'allocation de ressources pour les grilles et les Clouds basée sur la recherche de motifs

Adrian Muresan

RenPar'20 / SympA'14 / CFSE 8
Saint-Malo, France, du 10 au 13 mai 2011

Prédiction d'allocation de ressources pour les grilles et les Clouds basée sur la recherche de motifs

Adrian Muresan

Université de Lyon - CNRS - ENS Lyon - UCB Lyon 1 - INRIA
Laboratoire LIP, UMR 5668. Lyon, France
Adrian.Muresan@ens-lyon.fr

Résumé
Le phénomène des Clouds permet d'avoir une gestion de ressources dynamiques. L'efficacité de cette gestion devient une question importante avec la taille croissante du nombre de ressources et du nombre d'utilisateurs. A cause d'un coût d'initialisation non-négligeable, une prédiction précise des besoins doit être faite. Dans cet article, nous proposons une approche pour la prédiction de la charge basée sur l'identification d'occurences similaires dans l'historique récent d'utilisation d'une plate-forme. Nous présentons en détail l'algorithme d'**auto-redimensionnement** qui utilise la méthode précédente pour assister le système à prendre des décisions d'allocation de ressources ainsi que des résultats d'expériences utilisant des traces de plate-formes de grilles et de Clouds. Nous présentons également une évaluation globale de l'approche, son potentiel et son utilité pour permettre une gestion optimisée des ressources dans les Clouds.

Mots-clés : Auto-redimensionnement, recherche de motifs

1. Introduction

L'évolution des services logiciels vers les Clouds a pris un essor nouveau avec l'utilisation efficace de ressources virtualisées. Ces plates-formes permettent l'ajout ou le retrait dynamique (ou élastique) de telles ressources durant l'exécution d'une application hébergée sur une Cloud. En allant plus loin, il est possible de gérer dynamiquement ces ressources au niveau du fournisseur de ressources ou du client lui-même à travers l'API du fournisseur. Lorsqu'un client de Cloud optimise sa gestion des ressources à travers une gestion élastique, il économise des dépenses.
Le problème principal de la gestion dynamique des ressources est que les nouvelles ressources ne sont pas obtenues instantanément et que la latence de démarrage ne peut être négligée. Afin de résoudre ce problème, les stratégies actuellement en place utilisent des approches prédictives en espérant provisionner les ressources en avance, réduisant ainsi cette latence.
L'idée de l'auto-similarité a été utilisé dans d'autres domaines comme le trafic web [3]. A partir de ces idées, une nouvelle stratégie de gestion des ressources automatiques au niveau du client peut être élaborée qui identifiera des motifs d'utilisation de ressources qui ont eu lieu dans le passé et qui ont un degré de similarité important avec motif courant.
Le système de prédiction qui est issu de ce travail utilise l'historique de l'utilisation des ressources par un client et tente de deviner de manière *intelligente* les demandes courantes à court terme. Cela ne constitue pas pour autant un système complet de gestion de ressources dynamiques vu qu'il y a d'autres éléments à prendre en compte. Dans nos travaux, nous nous focalisons uniquement sur la prédiction d'utilisation des ressources. L'impact qu'ont les autres facteurs sur les décisions de redimensionnement est un sujet de recherche important qui dépasse les frontières de nos travaux actuels. La principale contribution de cet article est l'élaboration d'un algorithme de prédiction d'utilisation de ressources basé sur la recherche de motifs ayant pour but de donner à un client la capacité de gérer automatiquement l'allocation de ses ressources et ceci de manière élastique et optimisée. Cette recherche de motif est réalisée en modifiant un algorithme de comparaison de chaînes de caractères permettant de trouver les

chaînes proches. Notre approche est conçue pour travailler au niveau du client de Cloud, par dessus la plateforme de Cloud.
La suite de cet article est organisée comme suit. La section suivante présente un tour d'horizon des approches existantes données dans la littérature. Ensuite, la section 3 présente notre algorithme et ses principaux concepts. Enfin, avant une conclusion et une description de travaux futurs, la Section 4 présente nos résultats d'expériences utilisant en entrée des traces issues de grilles et de Clouds actuellement opérationnels.

2. Travaux connexes

Il y a actuellement deux approches principales pour simplifier les décisions d'**auto-redimensionnement** au niveau du client basé sur l'utilisation passée des ressources. La première approche prend en entrée l'utilisation passée des serveurs et en déduit un modèle mathématique qui la modélise. La valeur suivante de demande de ressources est obtenue en instanciant le modèle à la prochaine itération. La seconde approche est réactive, basée sur la charge actuelle du serveur et des règles d'**auto-redimensionnement** qui sont mises en place par un opérateur humain (généralement un client de Cloud). Cette approche est souvent référée comme l'approche par règle élastique ou l'approche *SLA* [5].
Dans [10], une description comparative de trois algorithmes d'**auto-redimensionnement** est donnée : l'auto-régression d'ordre 1 (AR1), la régression linéaire et l'algorithme RightScale. Les deux premiers algorithmes appartiennent à la première catégorie d'algorithmes d'**auto-redimensionnement**. Ils consistent à identifier une séquence récurrente (ou respectivement un polynôme) avec une fenêtre glissante et en l'utilisant pour prédire la valeur suivante. L'algorithme RightScale appartient à la deuxième catégorie. Son approche consiste à utiliser un système de vote démocratique dans lequel chaque machine virtuelle possédée par le Cloud vote soit pour augmenter ou pour réduire en fonction de sa charge. La majorité décide de la décision de redimensionnement pour la plate-forme globale. Les trois algorithmes sont comparés à l'aide d'une métrique définie dans l'article.
Une forme plus complexe de gestion dynamique de ressources par règles (SLA) peut être décrite à l'aide de règles élastiques qui dictent quelle partie du client doit évoluer, dans quelle direction et pour quel volume. Dans [5], on peut trouver un tel exemple avec des règles à seuil. Celà est effectué en étendant l'OVF (*Open Virtualization Format*), un format ouvert interopérable et indépendant des plateeformes ou des fournisseurs qui est utilisé pour décrire les applications virtuelles (VAs). Cette approche est réactive. Les règles d'extensibilité ont l'intérêt de combiner les bonnes performances des algorithmes à seuil avec l'adaptation. Ils ont donc été largement utilisés dans les Clouds disponibles actuellement.
Les études autour de la prédiction de charge et sa modélisation ne sont pas nouvelles et il y a eu de nombreux travaux récents notamment autour des grilles de calcul. Dans [8], on peut trouver une étude fine de charges synthétiques plusieurs grilles de production et de recherche. Des métriques de performance utiles pour l'évaluation d'environnements de grilles sont décrites. Cet article décrit les schémas de soumission de ces environnements de grilles et présente quelques unes des approches actuelles pour les modéliser qui incluent la combinaison de distributions de Poisson pour les motifs quotidiens ou en utilisant une fonction polynomiale de degré 8. Les auteurs prétendent que ses approches basées sur les motifs ne fonctionneront peut être pas puisqu'elles sont indifférentes au inter-dépendences de charges. Les auteurs poursuivent en présentant le framework GRENCHMARK de génération de charge synthétique de grilles, d'exécution et d'analyse [6].
Dans [7], on peut trouver un effort d'intégration d'un environnement de développement d'applications sur grille appelé Ibis [14], un co-ordonnanceur appelé Koala [11] et le générateur de charges GRENCHMARK [6] avec pour but de fournir un framework complet de génération de traces et de test points-à-points.
Un modèle non-linéaire de prédiction de charge de grille est donné dans [4]. Les auteurs proposent un modèle de prédiction sous forme de séries de composants fonctionnels connus finis, généralement pris à partir d'une classe de fonction sigmoïde. L'erreur de prédiction expérimentale est inférieure à 14% avec une moyenne à 7.5%.
Dans [12], on peut trouver un système de gestion de ressource temps-réel pour les jeux muti-joueurs en ligne basé sur un modèle d'usage prédictif. Les auteurs proposent un modèle prédictif basé sur un réseau de neurones. A travers leurs expériences, l'approche par réseaux de neurones prouve sa meilleure

précision par rapport à d'autres méthodes de prédiction testées : moyenne, moyenne mobiles, dernière valeur et lissage exponentiel. L'erreur de prédiction obtenue durant les expériences a une valeur maximale de 33% et une valeur minimum de 4.94%.
Un des défauts de ces approches réactives est le fait qu'elles sont insensibles à la forme globale de la charge qu'elles sont en train de prédire puisqu'elles ne considèrent pas les états de charge récents pour leurs résultats. Au contraire, les approches prédictives considèrent les tendances, mais les approches présentées précédemment qui utilisent des modèles mathématiques ne considèrent pas des auto-similarités qui n'ont pas un comportement périodique répétitif. C'est la raison qui a motivé nos travaux.

3. Algorithme de redimensionnement basé sur l'appariement de chaîne

3.1. Le concept d'appariement de chaîne

L'utilisation d'un client de Cloud peut parfois avoir un comportement répétitif. Celà peut être causé par la répétition de tâches exécutées par le client ou par le comportement répétitif de l'utilisateur. Étant donné l'autosimilarité du trafic Web, il est concevable que les schémas d'utilisation des services en ligne aient une probabilité forte d'avoir déjà eu lieu dans une forme similaire. Nous pouvons donc inférer l'usage d'un client de Cloud en examinant son comportement dans le passé et en extrayant des schémas similaires. La stratégie d'utilisation des motifs a deux entrées : un ensemble de traces d'utilisations de clients de Clouds et le motif d'utilisation courant constitué des dernières mesures d'utilisation d'un client de Cloud. Les données historiques ainsi que les motifs sont associés à des tableaux de nombres à virgule flottantes représentant l'utilisation des ressources de calcul de la plateforme.
Le motif d'utilisation du client de Cloud est utilisé pour identifier le nombre de motifs dans l'historique qui sont proches du motif présent lui-même. Les motifs résultants les plus proches seront interpolés en utilisant une interpolation pondérée.
Le problème qui consiste à chercher un motif dans un tableau de données est très proche du problème d'appariement de chaînes de caractères. Ce problème a été largement étudié et en particulier autour de la bioinformatique. Il reste toutefois un peu éloigné de notre problème puisque nous ne sommes pas intéressés par des appariements approximés utilisant une distance d'édition (ou de Levenshtein) [1].
Pour le problème qui nous intéresse, la distance d'édition ne peut être utilisée puisque nous ne comparons pas des caractères mais des nombres flottants. Nous sommes intéressés par l'identification des sous-tableaux de longueurs égales (ou proches) et dont les différences en valeur absolue sont les plus proches de zéro possible. Une insertion ou une suppression dans le tableau identifié aurait un impact important dans la différence des valeurs flottantes.

3.2. Algorithme d'appariement de chaîne

Il y a plusieurs solutions pour le problème d'appariement de chaîne. Nous avons choisi l'algorithme Knuth-Morris-Pratt (KMP) pour ses bonnes performances [2]. Pour notre problème, un appariement approximatif est nécessaire car les chances de trouver un motif identique à celui que nous sommes en train de chercher sont très basses. Les motifs candidats trouvés qui sont trop différents sur des petits intervalles ou en globalité doivent être retirés. Les motifs trouvés doivent être ensuite interpolés avec différents poids sur le résultat final, basé sur leur similarité avec le motif identifié.
Il y a deux types d'erreurs d'approximations utilisées pour l'appariement approximé :

1. une erreur instantanée qui dicte la différence maximum par instance de temps - donné par la fonction $Distance()$
2. une erreur cumulative qui dicte la distance maximum pour le candidat dans son ensemble et qui est la somme d'erreurs instantanées - donnée par la fonction $CumulativeDistance()$.

La distance entre le motif que l'on essaye d'apparier et un motif candidat est calculé d'une manière indépendante de l'échelle en normalisant les deux valeurs de motifs avec une échelle commune. Pour réduire les erreurs d'approximation sur le calcul flottant, on peut choisir un calcul de distance qui n'utilise pas les divisions et ainsi calculer uniquement sur des valeurs entières.

3.3. Modification de KMP

Algorithm 1 Calculer-prefix(P, ER_INST)

```
1: m ← length(P)
2: π[0] ← -1
3: k ← -1
4: scaleK = P[0]
5: scaleQ = P[1]
6: for q ← 1 to m - 1 do
7:    dist ← Distance(P[k+1], scaleK, P[q], scaleQ)
8:    maxDistance ← ER_INST × scaleQ × P[k+1]
9:    while k > -1 and dist > maxDistance do
10:      k ← π[k]
11:      dist ← Distance(P[k+1], scaleK, P[q], scaleQ)
12:      scaleQ = P[q - (k+1)]
13:   end while
14:   if dist ≤ ER_INST × scaleQ × P[k+1] then
15:      k ← k+1
16:   end if
17:   π[q] ← k
18: end for
19: return π
```

La fonction de calcul de préfixe est modifiée selon l'algorithme 1. Cette fonction calcule le préfix approximatif de répétition d'un tableau de données d'entrée P, et le sauvegarde dans un tableau appelée π. Les calculs sont limités par l'erreur instantanée spécifiée par ER_INST. L'échelle des deux composants comparés sont représentés par la première valeur de chaque composant. Cela est discutable mais en pratique nous avons obtenu des bons résultats avec cette approche. Dans la fonction, $scaleK$ représente l'échelle du postfix du motif. La comparaison des lignes 9 et 14 garantissent que la distance courante ne diffère pas plus d'une erreur acceptable (en pourcentage) par rapport au motif que nous comparons. Dans la comparaison dans la ligne 14, le terme $scaleQ$ représente l'échelle de la donnée. Il est nécessaire pour amener le motif à la même échelle que la donnée.

L'algorithme d'appariement est modifié comme dans l'algorithme 2. La fonction trouve toutes les instances similaires d'un motif P dans un tableau de données T avec une erreur instantanée limitée par ER_INST et une erreur cumulative limitée par ER_CUMUL. Par rapport à l'algorithme KMP original, la différence principale est l'utilisation des distances instantanée et cumulative comme moyen de filtrer les appariements qui sont trop différents soit sur une unité de temps soit globalement.

Sur les lignes 10 et 16, on garantit que la différence instantanée entre le candidat identifié et le motif est inférieure à l'erreur acceptable. Afin de garantir une comparaison correcte, le terme motif doit être étendu jusqu'à la même taille que la donnée, et ainsi le terme $scaleT$ est utilisé dans la comparaison. Le filtrage par la distance cumulative est effectué entre les lignes 20 et 24. Le temps d'exécution de cette fonction est $\Theta(m)$ où m est la longueur des tableaux, qui dans notre cas est égal à la longueur de P. La ligne 22 de l'algorithme assure que la distance cumulative du candidat au motif lui-même est inférieure à l'erreur cumulative acceptée. Le motif lui-même est représenté par le terme $patternSum$ dans la comparaison. C'est une somme de tous les termes dans le motif et doit être calculé une seule fois, au début de l'algorithme. La somme du motif doit être amené à la même échelle que la séquence candidate et ainsi le terme $scaleT$ est utilisé. L'utilisation de l'erreur cumulée change le temps d'exécution de l'algorithme d'appariement jusqu'à $\Theta(n \times m)$ dans le pire cas, où n est la longueur de la chaîne à apparier et m la longueur du motif en entrée.

Une fois que des appariements approximés ont été trouvés, chaque appariement doit avoir une contribution au résultat final qui est proportionnelle à sa distance relative par rapport aux autres motifs identifiés. Cela correspond à une somme pondérée des appariements identifiés. Une fois que les poids ont été calculés, l'interpolation est effectuée entre les L éléments suivants après chaque appariement approximé. Le résultat est une séquence prédite de longueur L.

Algorithm 2 KMP-approx(T, P, ER_INST, ER_CUMUL)

```
1: n ← length(T)
2: m ← length(P)
3: π ← Calculate-prefix(P)
4: q ← -1
5: scaleP = P[0]
6: scaleT = T[0]
7: for i ← 0 to n - 1 do
8:    dist ← Distance(P[q+1], scaleP, T[i], scaleT)
9:    maxDist ← ER_INST × scaleT × P[q+1]
10:   while q > -1 and dist > maxDist do
11:      dist ← Distance(P[q+1], scaleP, T[i], scaleT)
12:      q ← π[q]
13:      scaleT = T[i - (q+1)]
14:      maxDist ← ER_INST × scaleT × P[q+1]
15:   end while
16:   if dist ≤ maxDist then
17:      q ← q+1
18:   end if
19:   if q = m-1 then
20:      dist ← CumulativeDistance(P, T, i - m + 1)
21:      maxDist ← ER_CUMUL × patternSum × scaleT
22:      if dist ≤ maxDist then
23:         StoreSolution(dist / scaleT, i - m + 1)
24:      end if
25:      q ← π[q]
26:      scaleP = P[q+1]
27:      scaleT = T[i - (q+1)]
28:   end if
29: end for
```

4. Résultats expérimentaux

Afin de valider notre modèle nous avons utilisé des traces réelles provenant d'une plate-forme Cloud et pour compléter cette validation nous avons également utilisé des traces provenant de grilles de production qui sont des environnements proches des Clouds d'un point de vue de l'allocation des ressources. Les grilles de production, comme les Clouds, permettent l'accès à des ressources sur demande comme mécanisme d'allocation de ressources principale. Par ailleurs, l'utilisation de traces de Grille dans nos expérimentations s'explique également par le fait qu'il n'y a très peu d'archives publiques de traces de plates-formes de Cloud et il est très difficile de se procurer ce genre de trace pour des raisons de confidentialité.

Dans toutes nos expérimentations, nous avons utilisé 100 secondes comme unité de temps discrétisé. Les prédictions de trace représentent le nombre total de CPUs utilisés par différents jobs exécutés en parallèle dans l'unité de temps des 100 secondes. Nous avons ciblé uniquement l'utilisation CPU. Cependant si cette information était disponible, notre approche reste valide.

4.1. Source des données

Nous avons testé notre approche avec des traces provenant d'un client Cloud et de trois grilles différentes qui proviennent de la *Grid Workload Archive* [13], chacune ayant des particularités d'utilisation, avec des différences essentielles dans la fréquence et l'amplitude des changements de façon globale.

Animoto [1] est une application client Cloud déployée sur la plateforme EC2 d'Amazon [2], qui est spécialisée dans l'organisation automatique de vidéo basée sur le contenu. L'utilisation de la plate-forme est vue au travers de l'activité de l'utilisateur. **LCG** [3] **(Large Hadron Collider Computing Grid)** propose des traces de plusieurs noeuds de la grille associée. au Large Hadron Collider. Son comportement est moyennement oscillant. **NorduGrid** [4] offre une amplitude d'oscilliation plus importante étant donnée que cette grille est plus hétérogène que la précédente. **SHARCNET** [5] est présentée comme le "cluster des clusters". Cette grille est volatile et est sujette à de très fortes amplitudes.

4.2. Analyse des données

Nous avons besoin de trouver une meilleure façon de choisir la taille du motif, afin d'avoir des résultats plus pertinents et éviter les perturbations autant que possible. La taille du motif doit être déterminée en fonction du temps requis pour répondre à une requête sur un serveur. En pratique nous utilisons des longueurs de motifs qui correspondent au temps moyen de la résolution d'une requête.

4.3. Configuration expérimentale

Toutes les expériences utilisent les traces des serveurs avec le même type de données en entrée comme décrit précédemment avec une unité de temps de 100 sec., et la mesure de la consommation des ressources est basée sur le nombre de CPUs utilisé pendant ces 100 sec. Une longueur de motif de 100 unités a été utilisé pour toutes les expériences (c'est 100×100 secondes soit approximativement 2.7 heures du temps du serveur). On a utilisé une unité de temps de 100 secondes pour compenser le temps d'installation des ressources virtuelles (82 secondes), tout en laissant une marge d'erreur pour calculer la prédiction.

Un second ensemble de métrique est également utilisé permettant une comparaison avec d'autres algorithmes **auto-redimensionnement** existants. Cette métrique a été proposée et utilisée par l'Université de Californie Santa Barabara (UCSB) pour comparer les performances de trois algorithmes **auto-redimensionnement** [10] : AR1, une Régression linéaire et un algorithme de vote démocratique.

Nous avons également mesuré le temps moyen d'exécution nécessaire pour évaluer une prédiction. Cela a une incidence sur l'utilisation pratique de la prédiction. Ainsi il est nécessaire de le soustraire au temps de prédiction, qui est de 100 secondes, pour calculer le temps effectif de la prédiction.

Nous avons utilisé deux versions de la métrique proposée par l'équipe de UCSB : un score instantané où l'on considère que le coût de la ressource est considéré par fraction d'une heure et un second score où nous prenons le maximum de prédiction pendant la durée de l'heure et que nous utilisons comme information statique pour l'allocation pour l'heure entière.

Métrique	A - A	A - L	L - L	L - N	N - L	S - N
Erreur minimum de prédiction (%)	0.0	0.0	0.0	0.0	0.0	0.0
Erreur max de prédiction (%)	100	856.87	53.4	100.0	1146.00	528.03
Erreur médiane de prédiction (%)	2.69	4.08	1.0	1.2	1.74	0.9
Erreur moyenne de prédiction (%)	5.42	7.4	1.749	7.32	35.38	375.65
Métrique UCSB (max par heure)	-1.39	-15.95	10.66	3.43	30.64	-3.23
Métrique UCSB (instantanée)	-18.38	-38.75	-2.68	-10.71	27.27	-2.06
Temps d'exécution moyen par instance (ms)	186.625	27.63	41.734	514.956	162.949	528.418

TABLE 1 – Résultats des expérimentations de prédictions avec les traces provenant des quatre autres sources de données : Animoto, LCG, Nordugrid et SHARCNET. Les expériences consistent à prédire l'utilisation d'une plate-forme à partir de trace d'une autre plate-forme comme historique de données par morceau de 100 secondes. La signification des colonnes est fournie ensuite. La convention de nommage **A - B** où A est une plate-forme dont l'usage est prédit à partir des données de B

4.4. Résultats

Nous avons effectué deux types de tests : 1. la *self-prediction* dans laquelle une trace est utilisée pour prédire son propre comportement, ignorant les correspondances exactes. 2. Les prédictions croisées dans laquelle on utilise les traces d'une autre source. La Table 1 présente les résultats obtenus.

1. http://animoto.com
2. http://aws.amazon.com/ec2
3. http://lcg.web.cern.ch/LCG/
4. http://www.nordugrid.org
5. http://www.sharcnet.ca

A titre de la comparaison nous pouvons considérer les résultats de l'évaluation des trois autres algorithmes obtenus dans [10]. Ces valeurs résultent d'expérimentations basées sur un usage aléatoire et le point commun de ces trois algorithmes est qu'ils implémentent la stratégie *Smartkill*, améliorant ainsi leur efficacité. Seules les mesures de l'UCSB (instantanée) sont disponibles :

– RightScale : 11.11
– Autorégression d'ordre 1 : 17.3
– Régression Linéaire : 10.8

Il est à noter que l'erreur de prédiction maximale est élevée, que l'erreur médiane est considérablement faible (entre 0.9% et 4.08% pour tous les tests). Cela nous conduit à penser que l'approche **autoredimensionnement** est envisageable. Il n'est pas surprenant que l'algorithme qui offre les meilleurs résultats soit celui qui s'applique sur des données historiques qui ont une forte similitudes avec celles qui sont prédites. Cette similarité est influencée par plusieurs paramètres qui dépendent du domaine du serveur pour lequel la charge du serveur est prédite. Il en découle que les résultats obtenus à partir des données du même domaine peuvent facilement être utilisés pour en prédire un autre.

Lorsque nous nous comparons aux autres approches, nous avons obtenu à la fois des résultats moins bons et meilleurs, qui dépendent des paramètres de l'algorithme. Cela permet de conclure qu'il est possible d'améliorer les résultats en pratique si les paramètres de l'algorithme sont correctement calibrés. Le plus important étant la longueur du motif et la pertinence du domaine de l'historique des données en respectant le domaine de la plate-forme qui a été utilisé pour réaliser la prédiction.

En accord avec [9], le temps total moyen nécessaire pour obtenir une instance d'Amazon EC2 de type `m1.small` est de 82 secondes. En ajoutant le temps de prédiction de 100 secondes, le temps d'exécution de l'algorithme obtenu peut, en pratique, être négligé (entre 28 et 518 ms).

Nous avons réalisé les expérimentations avec différents volumes d'historique de données et différentes tailles de motifs. Les résultats avec le taux d'erreur de prédiction pour chaque cas est présenté dans le Tableau 2. Cependant cela ne montre pas que l'algorithme fournit les meilleurs résultats possibles, cela montre qu'une amélioration possible passe par l'augmentation de la taille de l'historique des données et que nous trouvons le meilleur motif à considérer lorsque nous réalisons la prédiction. Les résultats montrent comment les prédictions se comportent lorsque la taille du motif varie ainsi que la taille de l'historique des données. Nous avons fait varier l'historique des données de 100% - complet, à 50%, 25% et 12.5% de l'ensemble. La longueur du motif varie de 1000 unités de temps à 500, 100, 50, 25, 12 et 2 unités de temps. La longueur du motif ne doit pas être trop petite. En effet, cela produirait trop de motifs candidats qui peuvent sortir de notre contexte et le résultat final serait alors trop "pollué". L'augmentation de la taille de la base de données dépend de l'application. Dans le cas ou l'application Cloud ne change pas, les prédictions seront plus précises quand la base de données augmentera.

Longueur du motif	**Longueur des données**			
	100.0%	50.0%	25.0%	12.5%
1000	5.3%	9.5%	19.7%	100%
500	3.7%	6.0%	8.6%	18.7%
100	1.0%	1.2%	1.3%	2.0%
50	0.6%	0.5%	0.9%	1.3%
25	0.3%	0.3%	0.4%	0.5%
12	0.2%	0.2%	0.2%	0.3%
2	98%	100%	100%	82%

TABLE 2 – Les erreurs de prédiction obtenues pour différentes longueurs d'historique de données et différentes tailles de motifs pour la plate-forme LCG.

5. Conclusions et travaux futurs

Un des bénéfices les plus importants du calcul sur architecture Cloud est la capacité pour les clients du Cloud à adapter le nombre de ressources utilisées en se basant sur leur utilisation courante. Cela a un grand impact sur les économies de coût en ne payant pas pour des ressources qui ne sont pas utilisées. Le redimentionnement dynamique est réalisé via la virtualisation. D'un point de vue de la virtualisation, l'augmentation ou la réduction de ressources a un temps de prise en compte non nulle. Cela implique

que la précision de la méthode de prédiction va fortement aider le client de plate-forme Cloud à prendre les bonnes décisions d'**auto-redimensionnement**.

Cet article présente un nouvel algorithme de prédiction pour l'utilisation des ressources. Il se base sur un historique de données pour identifier un motif de comportement similaire pour une fenêtre actuelle qui ressemble à une fenêtre du passé. L'algorithme prédit alors l'utilisation prochaine du système par extrapolation à partir du motif identifié dans le passé. Les expérimentations ont montré que l'algorithme donne de bons résultats lorsqu'il est associé à des données en entrée pertinentes et que la qualité des résultats peut être amélioré en augmentant la taille de l'historique des données. Le temps d'exécution de l'algorithme a été prouvé comme négligeable dans nos expérimentations. Il serait intéressant à présent d'étudier la pertinence d'un ensemble d'historiques de données pour un domaine applicatif spécifique.

Remerciements : Ce travail a été développé avec le support financier de l'ANR (Agence Nationale de la Recherche) via le projet SPADES référencé 08-ANR-SEGI-025. Les auteurs remercient Animoto et The Grid Workload Archive pour la qualité des traces qu'ils ont produits.

Bibliographie

1. William I. Chang and Thomas G. Marr. Approximate String Matching and Local Similarity. In *CPM '94 : Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 259–273, London, UK, 1994. Springer-Verlag.
2. Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms, Chapter 32 : String Matching*. McGraw-Hill Higher Education, 2001.
3. Mark Crovella and Azer Bestavros. Explaining World Wide Web Traffic Self-Similarity. Technical Report TR-95-015, Boston, MA, USA, 1995.
4. Nikolaos Doulamis, Anastasios Doulamis, Antonios Litke, Athanasios Panagakis, Theodora Varvarigou, and Emmanuel Varvarigos. Adjusted fair scheduling and non-linear workload prediction for QoS guarantees in grid computing. *Computer Communications*, 30(3) :499 – 515, 2007. Special Issue : Emerging Middleware for Next Generation Networks.
5. Fermín Galán, Americo Sampaio, Luis Rodero-Merino, Irit Loy, Victor Gil, and Luis M. Vaquero. Service specification in cloud environments based on extensions to open standards. In *COMSWARE '09 : Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, pages 1–12, New York, NY, USA, 2009. ACM.
6. GrenchMark. http ://grenchmark.st.ewi.tudelft.nl.
7. Alexandru Iosup, , Ru Iosup, Dick H. J. Epema, Jason Maassen, and Rob Van Nieuwpoort. Synthetic grid workloads with ibis, koala, and grenchmark. In *In Proceedigs of the CoreGRID Integrated Research in Grid Computing*, 2005.
8. Alexandru Iosup, Dick H. J. Epema, Carsten Franke, Alexander Papaspyrou, Lars Schley, Baiyi Song, and Ramin Yahyapour. On grid performance evaluation using synthetic workloads. In *JSSPP'06 : Proceedings of the 12th international conference on Job scheduling strategies for parallel processing*, pages 232–255, Berlin, Heidelberg, 2007. Springer-Verlag.
9. Alexandru Iosup, Simon Ostermann, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE TPDS (in print)*, November 2010.
10. Jonathan Kupferman, Jeff Silverman, Patricio Jara, and Jeff Browne. Scaling Into The Cloud. `http://cs.ucsb.edu/~jkupferman/docs/ScalingIntoTheClouds.pdf`, 2009.
11. Hashim Mohamed and Dick Epema. Koala : a co-allocating grid scheduler. *Concurr. Comput. : Pract. Exper.*, 20(16) :1851–1876, 2008.
12. Radu Prodan and Vlad Nae. Prediction-based real-time resource provisioning for massively multiplayer online games. *Future Generation Computer Systems*, 25(7) :785 – 793, 2009.
13. TUDelft University The Grid Workloads Archive. http ://gwa.ewi.tudelft.nl.
14. Rob V. van Nieuwpoort, Jason Maassen, Gosia Wrzesińska, Rutger F. H. Hofman, Ceriel J. H. Jacobs, Thilo Kielmann, and Henri E. Bal. Ibis : a flexible and efficient java-based grid programming environment : Research articles. *Concurr. Comput. : Pract. Exper.*, 17(7-8) :1079–1107, 2005.