ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE
CONCENTRATION GÉNIE LOGICIEL
M. Sc. A.

PAR
Carine LACROIX

MÉCANISME DE SURVEILLANCE DYNAMIQUE DU TRAFIC ET DE RESSOURCES DANS DES SYSTÈMES IMS VIRTUALISÉS ET DISTRIBUÉS

MONTRÉAL, LE 03 MAI 2016

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Mme Nadjia Kara, directrice de mémoire
Département de génie logiciel et de TI à l'École de technologie supérieure

Mme Ghizlane El Boussaidi, présidente du jury
Département de génie logiciel et de TI à l'École de technologie supérieure

M. Chamseddine Talhi, membre du jury
Département de génie logiciel et de TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 07 AVRIL 2016

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Madame Nadjia Kara, ma directrice de mémoire, pour son accueil chaleureux dans son équipe et pour m'avoir encadrée et guidée sur ce vaste projet.

Mes collègues Souhila B., Abderrahmane E., et Hani N. pour leur aide aussi bien technique que morale.

Samuel D. pour le temps qu'il a passé à faire en sorte que techniquement tout fonctionne pour le mieux dans ce projet.

Mes parents pour leur soutien moral dans les périodes difficiles et leur soutien financier qui m'a permis de me concentrer pleinement à mon travail de recherche.

Mon copain, ma sœur, mon frère et tous mes amis de France qui m'ont soutenu et encouragé.

Mes colocataires pour tous les bons moments passés au Québec avec elles.

Une pensée à ceux qui nous ont quittés trop tôt

MÉCANISME DE SURVEILLANCE DYNAMIQUE DU TRAFIC ET DES RESSOURCES DANS DES SYSTÈMES IMS VIRTUALISÉS ET DISTRIBUÉS

Carine LACROIX

RÉSUMÉ

Le Cloud Computing est une infrastructure permettant l'exploitation de la puissance de calcul et de stockage de serveurs informatiques distants reliés par un réseau. Grâce à cette technologie, les gestionnaires d'infrastructure ont l'opportunité de créer des systèmes élastiques qui s'adaptent rapidement aux besoins en ressources de leurs clients, quelle que soit leur localisation. Le problème est alors de savoir quand et combien de ressources vont être nécessaires aux utilisateurs. Le gestionnaire de l'infrastructure a donc besoin de pouvoir prédire la quantité de ressources utiles à chaque instant. Cette prédiction servira à rendre disponible la quantité de ressources suffisantes aux clients afin de ne pas violer le contrat de service signé entre ces derniers et le gestionnaire. En même temps, cette prédiction permettra d'optimiser l'utilisation des ressources afin de réaliser des économies d'énergie et d'argent.

Au cours de ce travail, nous nous sommes intéressés à la gestion dynamique des ressources d'un système IP Multimedia Subsystem (IMS) virtualisé. Pour cela, nous avons développé un outil permettant d'adapter le nombre de cœurs utiles aux processeurs des nœuds S-CSCF du système. Le système virtualisé dans notre étude s'inspire de ceux utilisés par les opérateurs de réseaux fixes et mobiles pour fournir des services multimédias à leurs abonnés. Nous avons cherché à savoir s'il était possible de prédire le trafic des utilisateurs sur le réseau afin de pouvoir gérer de manière dynamique et proactive les ressources de ses nœuds CSCF (call/session control functions).

Avant de pouvoir développer l'outil de gestion des ressources mentionné précédemment, nous avons étudié le système IMS que nous avons virtualisé. Pour cela, nous avons détecté et analysé plusieurs métriques haut et bas niveau pour les nœuds pris en compte. Les différentes simulations réalisées, la collecte des données liées aux métriques suivies et l'analyse de ces données nous ont permis une meilleure compréhension du système. Nous avons alors pu constater que les nœuds de type P-CSCF du système ne semblent pas totalement fiables, car ils s'arrêtent parfois sans raison apparente. Nous avons aussi pu mettre en exergue que la principale cause d'arrêt du système était un manque de ressources CPU des nœuds de type S-CSCF. En étudiant de plus près ces nœuds, nous avons remarqué, grâce à la simulation de différents scénarios, que la charge de leur CPU en fonction du nombre d'appels par seconde suit un modèle exponentiel. Bien que nous ne puissions pas prédire le comportement des utilisateurs, le fait de surveiller cette ressource nous permet alors d'allouer plus de cœurs aux processeurs de ces nœuds avant qu'ils ne s'arrêtent pour manque de ressources. Cette allocation dynamique et proactive du CPU permet à notre système d'en optimiser l'utilisation.

Mots-clés: Cloud computing, système IMS, conteneur S-CSCF, gestion dynamique, proactive, prédiction CPU

DYNAMIC MONITORING MECHANISM OF TRAFFIC AND RESOURCES IN DISTRIBUTED AND VIRTUALIZED IMS SYSTEMS

Carine LACROIX

ABSTRACT

Cloud computing exploits computing power and storage in mainframe through a network. With this technology, infrastructure managers have the opportunity to create flexible systems that adapt quickly to their customers' needs regardless of their location. The problem is to successfully predict the moment when more resources will be required by the client. This is why infrastructure managers need to constantly predict adequate amount of resources required by theirs clients. This prediction aims to deploy enough resources for the client to respect the level agreement signed between the two parties, while the utilization of these resources remain optimized in order to save energy and money.

During this project, we have focused on the dynamic management of a virtualized IP Multimedia Subsystem (IMS) system's resources. With this goal, we have developed a tool adapting the number of useful cores for processors of S-CSCF nodes in the system. The system virtualized during the project looks like systems used by operators of landline and mobile services to provide multimedia services for their subscribers. We have tried to find out whether we could predict users' traffic on the network to be able to dynamically and proactively manage resources of its CSCF (call/session control functions) nodes.

Before developing the previously mentioned tool, we have studied the virtualized IMS system. To make this possible, we have detected and analyzed several high and low level metrics for studied nodes. Through simulation of different scenarios, collection of the studied metrics data and analysis of this data, we can better understand the system. We can also note that P-CSCF nodes in the system seem not totally reliable because they stop without explicit reasons. We have also found that the main cause of system shutdown was a lack of CPU for S-CSCF containers. By studying them more closely we have noticed, through simulation of different scenarios, that the workload of their CPU as a function of number of calls per second follows an exponential model. Although we cannot predict the behavior of users, monitoring this resource allows us to allocate more cores at S-CSCF containers before they stop due to overload. This dynamic and proactive CPU allocation lets our system to optimize resources usage.

Keywords: Cloud computing, IMS network, S-CSCF container, dynamically managed, proactively managed, CPU prediction

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CPU	central processing unit (unité centrale de traitement)
CGroup	control group
CPS	call per second (appel par seconde)
CSCF	call/session control function
DLS-SVM	dynamic least squares support vector machine
ETSI	European Telecommunications Standards Institute
I-CSCF	interrogating CSCF
P-CSCF	proxy CSCF
S-CSCF	serving CSCF
HSS	home subscriber server
IaaS	infrastructure as a service
PaaS	platform as a service
IaaS	infrastructure as a service
IHS	inadequately handled scenarios
IMS	IP multimedia subsystem
LS-SVM	least squares support vector machine
LXC	linux conteneurs
MAA	monitorage analyse et action, le nom de l'outil développé durant ce mémoire
MAE	mean absolute error
NGN	next-generation network (réseau de nouvelle génération)
NIST	national institute of standards and technology
ns	network simulator, logiciel libre

OSI	open systems interconnection
OS	operating system
RBF	radial basis function
SIP	session initiation protocol
SLA	service level agreements
SOA	architecture oriented service (architecture orientée service)
SVM	support vector machine, ou en français, machine à vecteurs de support
SVR	support vector regression
TIC	technologies de l'information et de la communication
UIT ou ITU	union internationale des télécommunications
UNESCO	organisation des nations unies pour l'éducation, la science et la culture
VM	virtual machine

LISTE DES SYMBOLES ET UNITÉS DE MESURE

cps	call per second (appel par seconde)
GHz	giga hertz
Mo	mega octet
ms	milliseconde
sec	seconde

INTRODUCTION

Le nombre d'abonnements à la téléphonie mobile n'a cessé d'augmenter ces dix dernières années atteignant un taux de pénétration de presque 100% (figure 0.1).

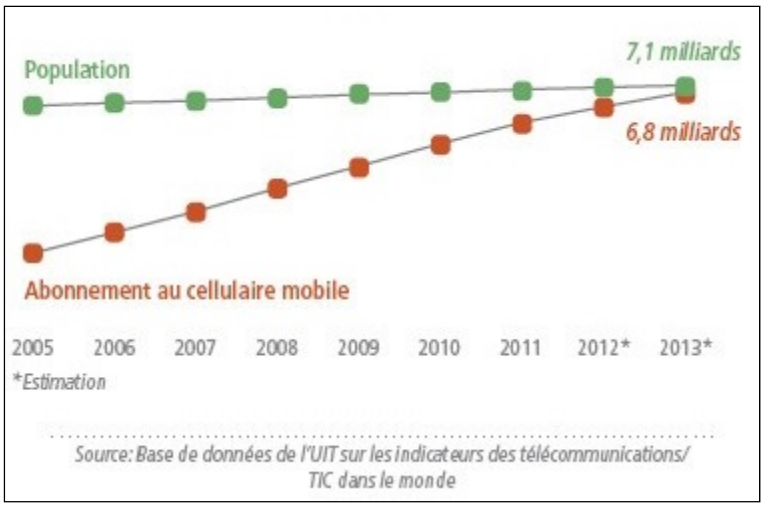

Figure 0.1 Convergence du nombre d'abonnements
au cellulaire mobile et de la population mondiale
Tirée de ITU News (Sanou 2013)

Les gens sont donc de plus en plus mobiles et leur besoin en connectivité ne cesse d'augmenter et d'évoluer. Malheureusement, les réseaux de téléphonie mobile actuels ne couvrent pas l'ensemble de la planète et leurs capacités sont limitées. Les propriétaires d'infrastructure et les fournisseurs de service doivent donc faire évoluer leurs réseaux. Étant donné qu'Internet permet déjà de relier les personnes n'importe où dans le monde et supporte de grosses charges de données, il a semblé intéressant de trouver un système fiable pour faire converger l'Internet et le monde de la téléphonie mobile. L'architecture IP multimedia subsystem (IMS) a été développée dans cet objectif. Cette architecture standardisée de type réseau de nouvelle génération (NGN) permet aux opérateurs de réseaux fixes et mobiles de fournir des services multimédias à leurs abonnées. L'objectif actuel est, pour les propriétaires d'infrastructure, de rendre ces réseaux de communication le plus efficace et rentable possible. En effet, l'architecture physique de cette technologie, basée sur l'informatique en nuage, permet d'exploiter un ensemble distant de ressources telles que le CPU, la mémoire vive ou

encore la bande passante, pour qu'elles soient disponibles de manière quasiment infinie. Si ces ressources sont disponibles en permanence, la consommation électrique des machines sur lesquelles elles sont installées représente un coût énorme de plusieurs billions de dollars. En revanche, si ces ressources ne sont pas disponibles lorsqu'un client en a besoin, le contrat de service (SLA) signé entre le fournisseur de service et le client est violé. Cette violation du SLA entraîne une amende à payer pour le propriétaire de l'infrastructure. L'idée est donc de trouver une manière efficace et fiable de gérer de manière dynamique et proactive les ressources du système IMS afin que seules les ressources nécessaires soient disponibles, mais que jamais elles ne viennent à manquer.

0.1 Problématique

Le développement d'un réseau de communication basé sur Internet doit relever plusieurs défis pour être efficace et rentable aux propriétaires des infrastructures. Il nécessite la construction d'énormes infrastructures matérielles. Étant donné la situation économique et écologique du vingt-et-unième siècle, le développement rapide des services proposés sur Internet et la croissance du nombre d'abonnés, il faut que le système déployé soit adaptatif afin d'être économique et soucieux de l'environnement. Les techniques de virtualisation, très populaire en ce moment, permettent de mettre en place des systèmes non monolithiques, distribués et flexibles. Grâce à leur architecture modulaire, les réseaux IP Multimedia Subsystem (IMS) font partie des réseaux pouvant adapter leurs ressources de manière dynamique. Il est donc intéressant, comme l'ont aussi pensé Nemati et al. (Nemati 2014), de virtualiser un système IMS et de le doter d'un mécanisme de gestion de l'élasticité pour optimiser l'utilisation de ses ressources et ainsi réduire les coûts qu'il génère. En effet, un système optimisé permettrait d'utiliser que les ressources utiles. Les autres ressources pourraient être éteintes afin d'économiser de l'énergie et de l'argent. L'objectif est alors de prévoir quand le réseau est surdimensionné afin de libérer des ressources, de minimiser leur gaspillage et même, si possible, d'éteindre des machines physiques. Les systèmes IMS possèdent une architecture modulaire. Plusieurs modules IMS peuvent alors être regroupés sur une même machine physique. Pour maintenir une bonne qualité de service, il faut pouvoir

prédire quand le trafic sera trop important pour les ressources actuellement disponibles afin d'en allouer plus. Le défi est donc de réussir à monitorer les ressources consommées par le système IMS et de prédire la quantité de ressources nécessaires à son bon fonctionnement. Le relevé de la consommation de ressources du système et l'analyse des données récoltées permettront de construire des modèles afin de prédire le besoin futur en ressource. Cette prédiction permettrait alors une optimisation de l'utilisation des ressources par le système. Un outil de prédiction semble donc nécessaire. Nous proposons de développer un tel outil dans le cadre de ce travail de recherche.

0.2 Objectif de la recherche

L'objectif principal de ce projet est de développer un système permettant la gestion dynamique des ressources des modules CSCF (call/session control function) de l'infrastructure de réseau de type IMS. Pour cela, nous allons commencer par mettre en place un système IMS virtualisé et distribué. Ce système sera soumis à différents jeux de test afin d'expérimenter différents profils de trafic. Nous pourrons alors suivre et analyser ses performances. Cette première étude nous permettra de développer des stratégies de prédiction de consommation des ressources par le système étudié. Basées sur les prédictions développées, nous pourrons alors implémenter une stratégie d'allocation des ressources pour un système IMS virtualisé. Pour la validation, nous proposons de tester et d'analyser les performances des modèles de prédiction proposés pour différents profils de trafic.

0.3 Méthodologie de travail et contributions

Pour atteindre l'objectif fixé, nous proposons de subdiviser le travail en trois parties. La première partie sera le monitorage des ressources consommées par un système IMS virtualisé. Ce relevé de données est essentiel à la prédiction mais l'est aussi à notre phase d'analyse du système afin de mieux comprendre son fonctionnement. Pour la télémétrie des ressources, nous n'avons pas trouvé d'outil libre permettant de mesurer et d'enregistrer les valeurs surveillées. Nous avons donc développé notre propre outil. La seconde partie, celle de prédiction, est la plus importante du projet. Il s'agit d'identifier s'il y a une corrélation et

un lien de cause à effet entre la consommation de ressources par le système et le nombre d'appels par seconde qu'il gère. Si une corrélation est remarquée, il faudra construire des algorithmes de prédiction des ressources nécessaires au bon fonctionnement du système. La troisième et dernière partie consiste à agir sur le système en fonction des prédictions réalisées et des valeurs mesurées pour mettre à disposition les ressources nécessaires. L'ensemble de ces phases formera un outil de gestion automatique et dynamique des modules du système IMS virtualisé. Dans cette étude, nous nous intéresserons plus particulièrement au module S-CSCF. Contrairement aux travaux de recherches déjà réalisés dans le domaine, l'approche que nous proposons est proactive et non réactive.

0.4 Organisation du rapport

Ce mémoire est organisé en quatre chapitres. Le premier présente une revue de l'état de l'art. Nous commencerons par décrire les différents concepts de la virtualisation et du nuage informatique. Ensuite, nous analyserons l'architecture modulaire d'un système IMS. Pour finir, nous analyserons les techniques existantes d'analyse du trafic dans le nuage informatique et de prédiction de consommation telle que celle du CPU. Le second chapitre décrit les motivations, les principes et les défis pour le développement d'un nouveau modèle de gestion proactive des ressources des modules CSCF d'un système IMS virtualisé. Nous y trouverons aussi l'architecture de l'outil développé. Le chapitre trois présente les différents algorithmes proposés pour la prédiction de la consommation en ressources physiques des modules CSCF. Pour cela, le chapitre commence par des étapes d'analyse d'un système IMS virtualisé afin de mieux le comprendre. Le quatrième et dernier chapitre propose une analyse des performances de l'outil développé et des algorithmes de prédiction définis. Ce mémoire se terminera par une conclusion générale sur notre travail de recherche.

CHAPITRE 1

REVUE DE LA LITTÉRATURE

1.1 Le concept de virtualisation

Comme le définit N. Triki (Triki 2013), « la virtualisation est un procédé informatique qui consiste à ajouter une couche d'abstraction entre les ressources physiques et la représentation logique d'un système ou d'un réseau informatique ». Afin de mieux comprendre ce concept, nous analysons les quatre principales manières de virtualiser un système.

1.1.1 La virtualisation complète

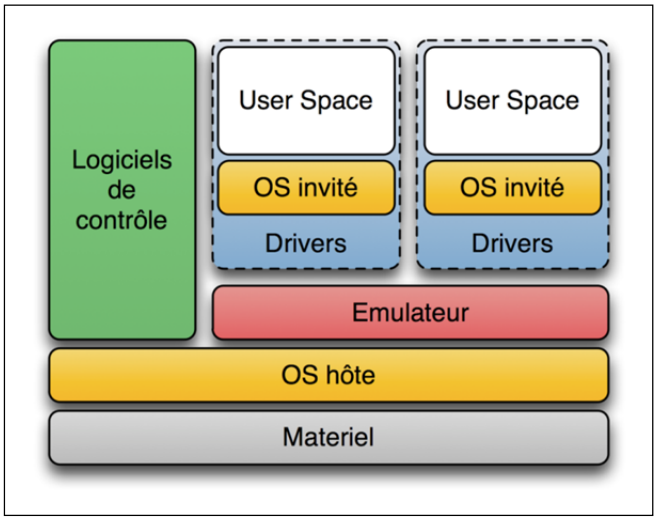

Figure 1.1 Architecture de virtualisation complète
Tirée de ArcticLinux (ArcticDesign 2014)

La virtualisation complète permet de lancer une ou plusieurs machines virtuelles sur une machine physique munie d'un système d'exploitation. Chaque machine virtuelle sera munie de son propre système d'exploitation et fonctionnera sur la machine réelle comme un simple logiciel. Comme représenté sur la figure 1.1, on a alors plusieurs systèmes d'exploitation invités qui s'exécutent en parallèle sur un système hôte et tous sont hébergés sur la même

machine physique. Les différents systèmes invités n'ont pas conscience de partager la machine et ont l'impression de gérer toutes les ressources matérielles. Comme l'explique L. Bonnet dans son état de l'art (Bonnet-Bearstech), ce type de virtualisation nécessite que le système hôte récupère au vol toutes les instructions lancées par un système invité afin de réellement les exécuter. Cette opération est coûteuse en temps et est complexe, car, la plupart du temps, les instructions ne peuvent pas être simplement transmises au matériel et doivent être interprétées. D'après D.A. Menascé (Menascé 2005), les performances du système sont donc réduites par l'empilement des différentes couches d'abstraction entre le système invité et le matériel.

1.1.2 La paravirtualisation

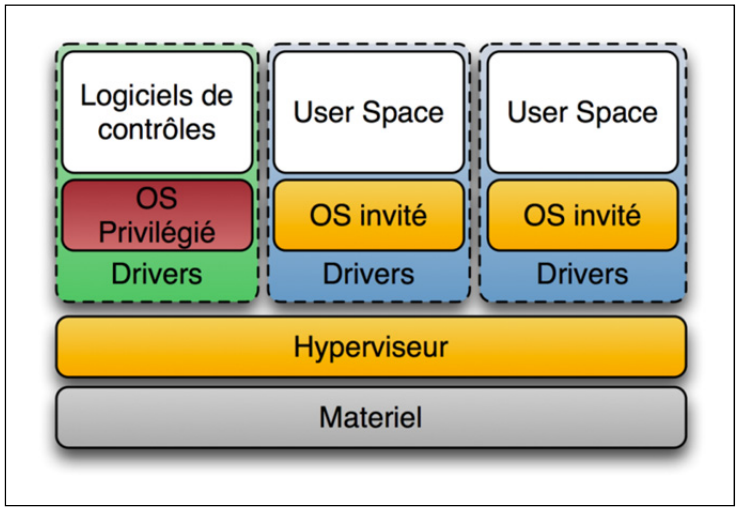

Figure 1.2 Architecture de paravirtualisation
Tirée de ArcticLinux (ArcticDesign 2014)

La paravirtualisation, en comparaison à la virtualisation complète, vise à diminuer le nombre de couches d'abstraction. On peut voir cela sur la figure 1.2. Comme décrit par R. Gadhgadhi (Gadhgadhi 2013), on émule dans ce cas encore un système d'exploitation sur un autre grâce à une machine virtuelle, mais cette fois-ci, le système invité est conscient de ne pas gérer directement les ressources matérielles. Ce dernier a donc subi quelques modifications afin de collaborer avec le système hôte pour être plus performant, les instructions envoyées prennent

donc en compte le fait qu'elles vont devoir être interprétées par le système qui les exécutera. L'inconvénient de cette solution est que pour modifier un système d'exploitation il faut avoir accès au code source et la permission du détenteur des droits de le modifier. Il est donc possible de modifier un système libre tel que GNU/Linux et les systèmes BSD (Berkeley Software Distribution, licence libre utilisée pour la distribution de logiciels) mais pas les systèmes propriétaires comme Microsoft Windows et Mac OS. De plus, comme l'explique L. Bonnet (Bonnet-Bearstech), ce système n'est pas encore vraiment optimal, car il y a encore de nombreuses couches d'abstraction entre le système invité et le matériel.

1.1.3 Un système hyperviseur

Le système hyperviseur, pensé par Popek et Goldberg dans les années 70 (Popek and Goldberg 1974), a pour but de réduire encore le nombre de couches d'abstraction entre le système invité et le matériel. L'idée est donc de laisser aux systèmes invités un accès, bien que contrôlé, au matériel. Le système hyperviseur a donc été développé pour être un système d'exploitation minimaliste qui sert à allumer la machine puis à lancer les différents systèmes invités qui devront ensuite passer par lui pour soumettre leurs instructions au matériel. L'hyperviseur s'assurera juste que le système invité a les droits pour exécuter ce qu'il souhaite exécuter. Cette technique de virtualisation permet ainsi de meilleures performances que les deux méthodes précédemment présentées. En effet, comme l'expliquent R. Sailer et al. dans (Sailer, Valdez, et al. 2005) et (Sailer, Jaeger, et al. 2005), avec un système hyperviseur, le système invité n'est pas vu comme un processus par le système hôte. L'hyperviseur régule l'accès des systèmes invités aux ressources matérielles, il n'y accède pas pour eux. Les systèmes invités peuvent alors exploiter toutes les performances de la machine hôte et de ses périphériques. Cette technique nécessite donc un contrôle de l'accès au matériel et de l'utilisation des ressources plus fins mais permet un gain significatif des performances. De plus, ce système de virtualisation ne veille plus à ce qu'un système invité n'en dérange pas un autre, comme dans les solutions décrites auparavant, la granularité est plus fine, il veille à ce qu'un programme utilisateur ne dérange pas les autres programmes du système. Mais cette solution, comme la précédente, nécessite que le système hôte soit

modifié. En effet, il faut adapter ces couches bas niveau pour qu'elles communiquent avec l'hyperviseur. Ces modifications sont très lourdes et peuvent augmenter la complexité du code. De plus, il faut avoir accès au code source du système d'exploitation que l'on souhaite utiliser et cela est impossible avec tous les systèmes propriétaires. Comme l'explique L. Bonnet (Bonnet-Bearstech), cette méthode de virtualisation permet donc de meilleures performances que celles précédemment décrites, mais les modifications du système d'exploitation devant être réalisées pour cela sont complexes et même parfois impossibles dans le cas de système d'exploitation propriétaire.

1.1.4 L'isolation ou virtualisation du système d'exploitation

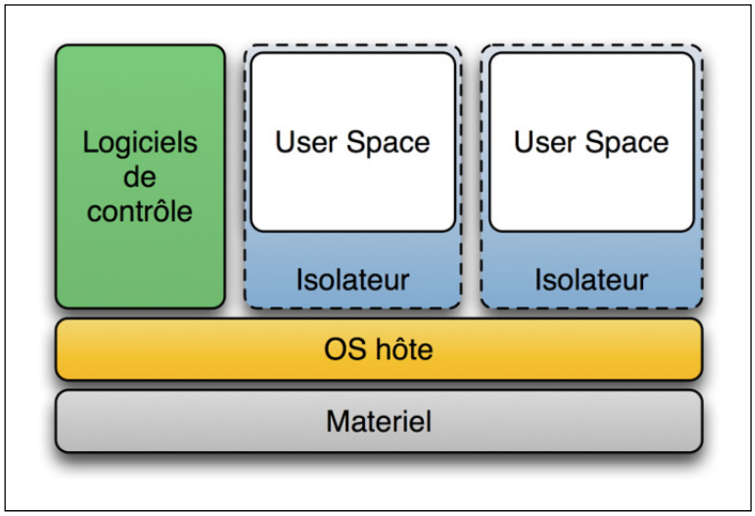

Figure 1.3 Architecture de l'isolation
Tirée de ArcticLinux (ArcticDesign 2014)

L'isolation est la dernière idée présentée ici et décrite par F. Santy (Santy 2013). Cette technique, illustrée à la figure 1.3, cherche à éliminer la couche restante entre le système qui souhaite exécuter des instructions, et le matériel. Pour ce faire, on ne va plus créer des machines virtuelles sur une machine réelle pour avoir chaque processus qui s'exécute dans un environnement à part. On va plutôt isoler tous ces processus sur un même système. Dans ce cas, tous les processus doivent pouvoir s'exécuter sur le même système d'exploitation où ils seront isolés les uns des autres. Un environnement, duplication du système hôte, est créé

pour chacun d'eux avec pour ressource seulement ce dont le processus a besoin pour fonctionner. Par la suite, cet environnement est nommé isolateur. Les processus ne peuvent pas sortir de cet environnement, ils ont donc une vision très limitée de l'état de la machine sur laquelle ils s'exécutent et ne savent pas quels autres processus s'exécutent en même temps qu'eux. Cela permet d'assurer la sécurité du système et des autres processus. Cette solution donne les meilleures performances en termes de rapidité d'exécution et permet à l'administrateur du système de gérer comme il le souhaite les ressources attribuées à chaque processus de manière indépendante. L'inconvénient est que chaque système invité est du même type que le système hôte. On ne pourra plus utiliser les techniques de virtualisation pour faire fonctionner sur sa machine physique un autre système d'exploitation que celui déjà installé.

Quelle que soit la solution retenue et mise en place, on peut donc donner l'illusion d'avoir un système dédié à un service avec en réalité une seule machine physique.

1.1.5 Avantages et inconvénients de la virtualisation

- **Avantages de la virtualisation**

Les avantages de la virtualisation sont nombreux. Tout d'abord, comme mentionné par Studnia et al. (Studnia et al. 2012), la virtualisation permet d'augmenter la sécurité. En effet si un processus est victime d'une attaque ou d'une défaillance cela n'affectera pas les autres processus présents sur la machine qui continueront de fonctionner, car ils sont indépendants les uns des autres. Ensuite, comme l'expliquent Rosenblum et Garfinkel (Rosenblum and Garfinkel 2005), la virtualisation permet d'installer le système d'exploitation voulu sur la machine physique souhaitée bien que cette dernière possède déjà un système d'exploitation. Le système d'exploitation que l'on souhaite utiliser peut être différent de celui installé sur la machine physique. Un autre avantage, décrit par Sahoo et al. (Sahoo, Mohapatra, and Lath 2010), est la flexibilité. En effet, il est facile d'ajouter ou de supprimer une machine virtuelle d'une machine physique et on peut aussi déplacer une machine virtuelle d'une machine

physique à une autre grâce à des outils de migration. Pour finir, la virtualisation a aussi l'avantage de permettre de réaliser des économies. Comme l'explique Callow (Callow 2000), ces économies peuvent être réalisées en réduisant la taille du parc de machines. Si on peut lancer sur une même machine physique plusieurs systèmes, on aura un plus petit nombre de machines à acheter et par la suite à gérer. De plus, d'après L. Barroso et U Hölzle (Barroso and Hölzle 2007), deux machines utilisées à 25% de leurs capacités consomment plus d'énergie qu'une machine utilisée à 50% de ses capacités, on peut alors penser à regrouper plusieurs systèmes qui consomment peu de ressources sur une même machine physique pour en éteindre certaines et ainsi réaliser des économies d'énergie favorable aussi bien aux fournisseurs du réseau qu'à la planète. Cette économie d'énergie peut représenter des gains énormes aux fournisseurs d'infrastructure. En effet, si on reprend l'exemple d'Hamilton (Hamilton 2009), la consommation électrique d'un centre de données pour le nuage informatique coûte plusieurs billions de dollars aux fournisseurs et grâce à la virtualisation ils pourraient réduire leur facture de plusieurs millions.

- **Inconvénients de la virtualisation**

La virtualisation a aussi quelques inconvénients. Comme l'expliquent Sahoo et al. (Sahoo, Mohapatra, and Lath 2010), si plusieurs processus s'exécutent sur une même machine physique il faut que cette dernière soit suffisamment puissante pour se substituer à plusieurs autres. De plus, si cette dernière tombe en panne tous les processus lancés sur celle-ci vont cesser de fonctionner. Pour finir, la virtualisation a un gros défaut, comme le mentionne Santy (Santy 2013), les techniques de virtualisation ne sont pas encore encadrées par beaucoup de standards. Le manque de standard peut poser problème pour l'interopérabilité d'un système. L'interopérabilité est cruciale pour l'économie. En effet, elle permet à un ensemble de systèmes de pouvoir communiquer en employant des structures et des types de données communs, sans que des interfaces supplémentaires soient développées pour cela.

Malgré ces inconvénients et grâce à tout ce que permet la virtualisation, nous allons avoir besoin d'utiliser cette technique pour isoler des processus lors du déploiement de notre

système IMS. Dans notre étude, la performance du système déployé est un point important, nous avons donc besoin de rester proches du matériel de notre infrastructure physique. Nous privilégierons donc la virtualisation par l'isolation quand cela est possible.

1.2 Le nuage informatique

Selon Mell et Grance (Mell and Grance 2011), « le nuage informatique est un modèle pratique, à la demande, [qui] permet d'établir un accès, par le réseau, à un réservoir partagé de ressources informatiques configurables (réseau, serveurs, stockage, applications et services) qui peuvent être rapidement approvisionnées et libérées en minimisant les efforts de gestion ou les interactions [des utilisateurs du système] avec le fournisseur de service ». Le nuage informatique est donc un concept qui permet d'offrir différents types de ressources matérielles et logicielles aux individus et aux entreprises à travers Internet.

1.2.1 Les cinq caractéristiques essentielles du nuage

Selon la définition donnée par le NIST (Mell and Grance 2011) du nuage informatique, ce dernier doit répondre à cinq caractéristiques essentielles :

- **Libre-service à la demande :** L'utilisateur doit pouvoir allouer et libérer des ressources rapidement et sans avoir besoin d'une validation humaine de la part du fournisseur de service.
- **Large accès au réseau :** L'utilisateur doit également pouvoir accéder au réseau quel que soit l'appareil qu'il utilise grâce à un client lourd ou léger.
- **Concentration des ressources :** L'utilisateur doit pouvoir choisir, à un haut niveau (pays, centre de données), où stocker ses données. Il a cependant conscience que les ressources informatiques du fournisseur sont partagées entre plusieurs clients et réparties sur plusieurs machines.
- **Élasticité rapide :** Le réseau doit être élastique, c'est-à-dire qu'il doit être capable de s'adapter aux besoins en ressource des utilisateurs du système en approvisionnant et

désapprovisionnant des ressources matérielles de manière automatique. Pour un client du fournisseur d'infrastructure, les ressources matérielles doivent sembler illimitées.

- **Service quantifiable :** Le fournisseur et le consommateur concerné doivent pouvoir surveiller et contrôler en toute transparence la quantité de ressources consommées.

Ces cinq caractéristiques permettent à l'utilisateur, comme l'expliquent Sadiku et al. dans leur article (Sadiku, Musa, and Momoh 2014), une réduction des coûts, une facilité d'utilisation, une bonne qualité de service et une grande fiabilité du nuage informatique. En contrepartie, l'utilisateur du nuage informatique expose ses données à des pertes de confidentialité et de sécurité. Comme nous pouvons le lire dans l'article de B. Grobauer et al. (Grobauer, Walloschek, and Stöcker 2011), ces deux risques font l'objet de nombreuses recherches afin d'être mieux contrôlés.

1.2.2 Les quatre modèles de déploiement du nuage

L'infrastructure du nuage informatique peut varier suivant quatre modèles en fonction de qui la possède et du niveau d'accès que doivent avoir les utilisateurs pour s'y connecter. Les différentes architectures exposées ici peuvent varier en termes d'investissement pour une entreprise, de sécurité des données, de fiabilité et de performance pour un utilisateur.

- **Le nuage privé**

Comme défini par Dillon et al. (Dillon, Wu, and Chang 2010), le nuage privé est un nuage informatique où l'entreprise qui possède l'infrastructure est la même que celle qui en utilise les ressources. La gestion des ressources et l'hébergement de l'infrastructure peuvent cependant être réalisés par un tiers. L'entreprise offre donc le service du nuage informatique uniquement à ses employés. Il est estimé, par Arumtec (Arumtec), que les entreprises qui possèdent ce genre d'infrastructure n'utilisent en moyenne que 5 à 15% de leurs ressources. Ce type d'architecture réduit donc l'intérêt du nuage, car encore beaucoup de ressources énergétiques sont gaspillées par un trop grand nombre de ressources matérielles disponibles et non nécessaires. Ce modèle est cependant souvent apprécié, car il a l'avantage de

permettre un haut degré de contrôle sur la fiabilité et la sécurité des données, car le réseau n'est pas partagé avec d'autres utilisateurs. Les accès en sont donc théoriquement contrôlés.

- **Le nuage communautaire**

L'infrastructure du nuage, dans ce cas-ci et selon Marinos et al. (Marinos and Briscoe 2009), est la propriété d'une communauté de personnes partageant des intérêts communs. La gestion du nuage peut être faite par la communauté elle-même ou par un organisme tiers via Internet. L'avantage de cette architecture, par rapport à celle précédemment exposée, est le partage des coûts entre les différents utilisateurs. Un exemple de déploiement de cette architecture pourrait être au sein d'une université où les différents membres sont des étudiants, des professeurs et du personnel administratif répartis dans différents domaines d'étude et différents lieux. L'ensemble des utilisateurs, avec ce modèle, a encore un niveau élevé de fiabilité et de sécurité de ses données. En effet, l'accès au nuage est contrôlé et réservé à un groupe de personnes sélectionnées.

- **Le nuage public**

Comme le décrit Armbrust et al. (Armbrust et al. 2010), l'infrastructure du nuage public appartient et est gérée par des entreprises, des instituts universitaires et le gouvernement. L'accès au nuage est ouvert à tous les utilisateurs. Pour ces derniers, ce modèle permet un faible coût d'investissement, une évolutivité régulière et une optimisation des coûts d'exploitation, car, selon Ahmed et al. (Ahmed et al. 2012), il n'est facturé aux utilisateurs que les ressources consommées. De plus, selon Chaisiri et al. (Chaisiri, Lee, and Niyato 2012), le transfert des problèmes technologiques est reporté sur le fournisseur d'infrastructure, car les utilisateurs de l'infrastructure ne la gèrent absolument pas et tout est fait pour qu'ils n'aient pas conscience de sa complexité. En contrepartie, ils n'ont pas un contrôle précis sur le stockage de leurs données. Ce dernier point fait souvent préférer aux entreprises les nuages privés ou communautaires.

- **Le nuage hybride**

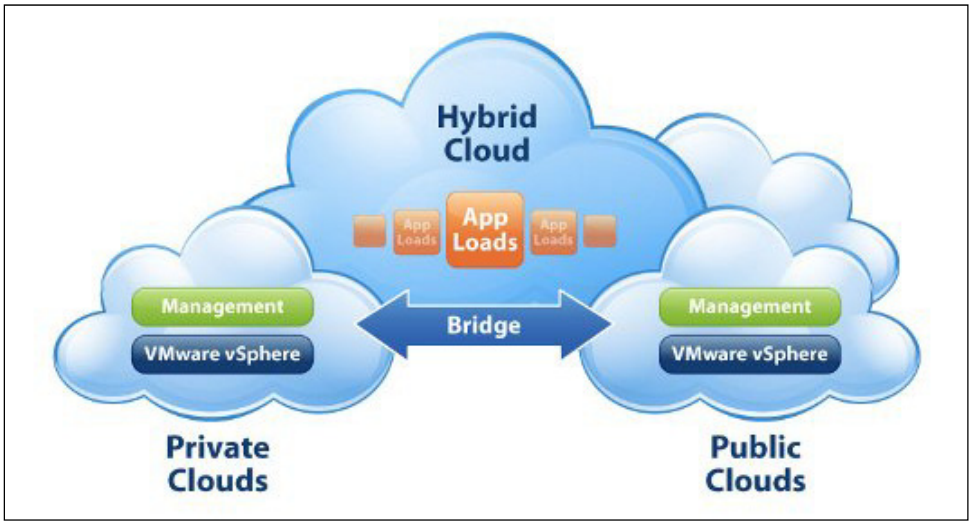

Figure 1.4 Représentation du nuage hybride avec VMWare
Tirée de (VMWare 2012)

Selon Sotomayor et al. (Sotomayor et al. 2009), l'infrastructure de ce type de nuage est la combinaison entre deux ou trois des modèles présentés ci-dessus. Le but est de tirer parti des avantages de chaque modèle exploité. Un exemple d'utilisation d'un nuage hybride serait celui d'une entreprise dont le nuage est à la fois privé et public. L'utilisation du nuage privé garantit la sécurité des données et le nuage public permet une flexibilité du réseau afin de pouvoir supporter une augmentation du trafic sans que la qualité de service soit amoindrie.

Les nuages peuvent être liés entre eux créant un nuage de nuage. Il y a deux intérêts majeurs à cela pour des fournisseurs d'infrastructure. Le premier est stratégique. En effet, en reliant deux nuages qui couvrent chacun une zone géographique, les fournisseurs agrandissent la zone physique d'où l'utilisateur du réseau peut se connecter. Le deuxième intérêt est économique, chaque fournisseur d'infrastructure va louer une partie de ses ressources à l'autre. Cette entente permet au propriétaire de l'infrastructure de rentabiliser du matériel non exploité et au locataire de mettre à disposition de ses clients des ressources sans avoir directement investi dans l'infrastructure.

1.2.3 Les trois couches de l'architecture du nuage

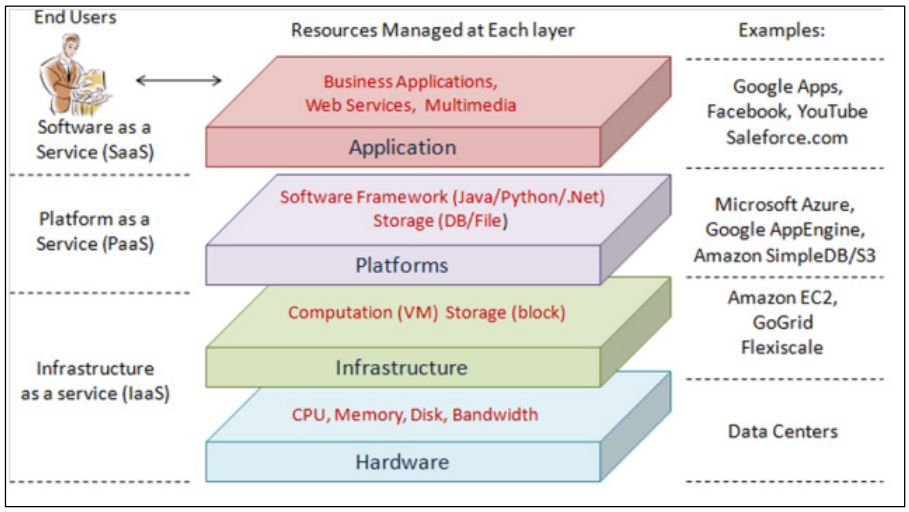

Figure 1.5 Architecture du nuage informatique
Tirée de (Zhang, Cheng, and Boutaba 2010)

B. Chee et C. Franklin (Chee and Franklin Jr 2010) définissent, d'un point de vue économique, l'informatique dans le nuage comme une offre commerciale où les différents acteurs du domaine proposent ou souscrivent à des abonnements afin de rendre disponible ou d'accéder à des services externes. Chaque gestionnaire d'une des couches du nuage, présentées ci-dessous et illustrées à la figure 1.5, doit donc avoir une entente avec un gestionnaire d'une couche inférieure pour en utiliser les services. Selon le National Institute of Standards and Technology, les trois couches de services de l'infonuagique sont l'infrastructure en tant que service, la plateforme en tant que service et le logiciel en tant que service.

- **Logiciel en tant que service (SaaS)**

Grâce à cette couche du nuage informatique, un fournisseur propose l'utilisation d'un logiciel à un client via Internet. Le client peut soit manipuler les applications via un navigateur web soit en les installant sur son appareil tel que son PC. L'utilisateur des logiciels n'a pas à se soucier de la maintenance de l'outil. Gmail est un exemple de service fournit par Google. Le

fournisseur du service ne contrôle pas l'infrastructure du nuage, ni le réseau, les serveurs ou l'espace mémoire, car il utilise pour cela les services de la couche inférieure du nuage.

- **Plateforme en tant que service (PaaS)**

Le fournisseur de ce service ne permet pas de contrôler le réseau, les serveurs, le système d'exploitation ou le stockage. Il met à disposition des utilisateurs de la couche SaaS un environnement d'hébergement et de développement sur le nuage grâce à des langages de programmation et des outils spécifiques. Par exemple, Google App Engine (Zahariev 2009) est une plateforme utilisée par des fournisseurs de services de la couche supérieure pour concevoir et héberger des applications web basées sur les serveurs de Google.

- **Infrastructure en tant que service (IaaS)**

Ce service est celui de plus bas niveau et consiste à offrir un accès à un parc informatique virtualisé. Selon Djawida (Djawida 2010), les fournisseurs de cette couche disposent d'une infrastructure matérielle qui leur procure une puissance de calcul et de stockage quasiment illimitée, car elle peut théoriquement être étendue autant que nécessaire. Ces ressources semblent infinies, car le nuage permet de les mutualiser grâce à un système où l'ensemble des machines virtuelles sont lancées sur une même infrastructure physique. Le concept de nuage informatique est uniquement possible grâce à la virtualisation et à des mécanismes de gestion et d'optimisation des ressources matérielles. Grâce à cette couche, le consommateur dispose de machines virtuelles sur lesquelles il peut installer le système d'exploitation et les applications qu'il souhaite.

Nous distinguons alors les quatre acteurs principaux dans le domaine du nuage informatique qui sont: le fournisseur d'infrastructure, le fournisseur de plateforme, le fournisseur de services et l'utilisateur des services. Comme représenté sur la figure 1.6 et comme l'expliquent Patel et al. (Patel, Ranabahu, and Sheth 2009), tous ces acteurs sont liés par le

SLA, pour Service Level Agreements, un contrat où le niveau de service est formellement défini.

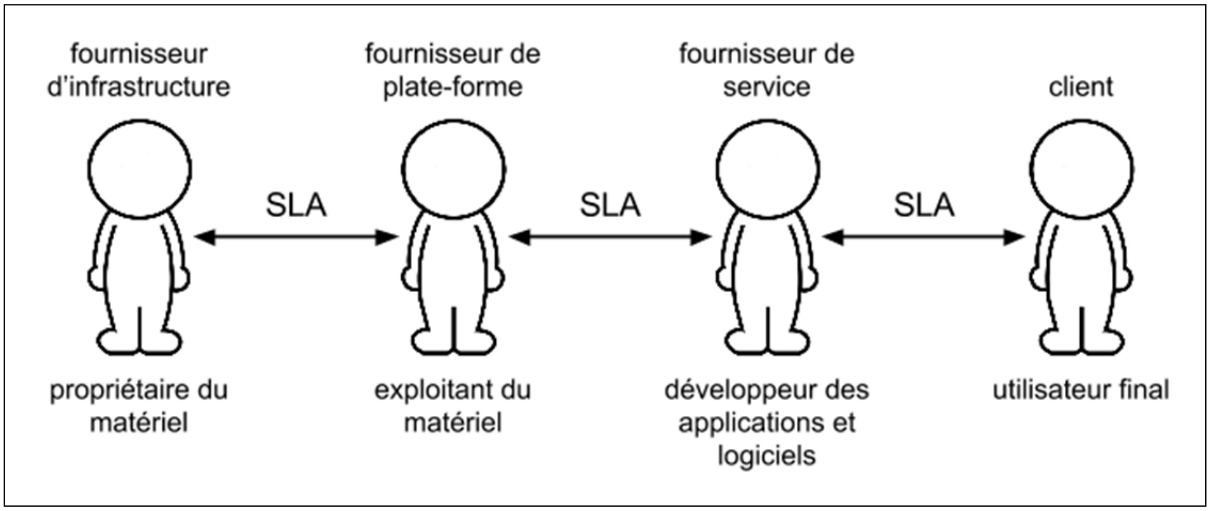

Figure 1.6 Les acteurs du nuage liés par le SLA

1.2.4 Un contrat entre les acteurs du nuage

Le SLA cible les services qu'un fournisseur doit offrir à son client. Il définit aussi de manière formelle la qualité minimum de ces services. Comme l'explique S. Frey et al. (Frey, Reich, and Lüthje 2013), le document spécifie la qualité de service (niveau de service) à travers des indicateurs clés de performances (Key performance indicators – KPI) qui représentent les exigences et les attentes du service. Parmi ces indicateurs, nous retrouvons le temps de réponse, le débit et la disponibilité. Il spécifie aussi la facturation des clients pour leur consommation en ressources du nuage et les pénalités appliquées en cas de non-respect du SLA.

1.3 L'architecture des réseaux de prochaine génération

Comme l'a formulé Tharan (Tharan 2004), les réseaux actuels, dits réseaux de nouvelle génération (NGN), doivent permettre à des utilisateurs fixes ou mobiles d'accéder depuis l'appareil de leur choix à de nombreux services d'échange d'information comme la voix, les données ou le contenu audiovisuel. IMS (IP Multimedia Subsystem) est un exemple de

système adopté par l'industrie pour fournir des services multimédias basés sur IP tels que *Voice over LTE* (*VoLTE*) (Poikselkä et al. 2012) et *Rich communications* (Patil and Sawant 2012). European Telecommunications Standards Institute (ETSI) a identifié IMS comme une infrastructure de service qui pourrait bénéficier du concept du nuage informatique pour améliorer son architecture actuelle. En effet, IMS est un système monolithique offrant peu de flexibilité dans la sélection des composants qui le constituent et dans son déploiement (système IMS centralisé versus virtualisé). La virtualisation du système IMS permet d'améliorer l'extensibilité, de simplifier les mises à jour et l'implémentation ainsi que de réduire les coûts d'investissement et d'opération (Zhiqun et al. 2013). Dans notre sujet de recherche, nous nous intéressons donc à la virtualisation d'un système IMS.

1.3.1 Le modèle en couche de l'architecture IMS

Nous retrouvons dans les systèmes IMS les différentes couches du modèle OSI regroupées dans trois couches plus génériques. Ces dernières sont illustrées sur la figure 1.7.

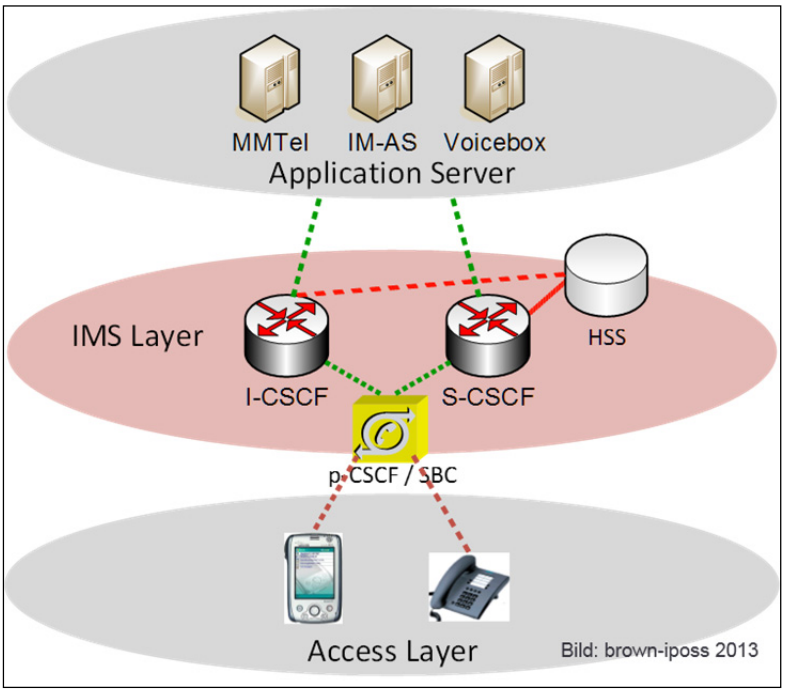

Figure 1.7 Couche d'un système IMS
Tirée de (Brown 2013)

Selon le standard (Patil and Sawant 2012), la couche IMS est le cœur du système IMS. Elle assure l'établissement des appels et gère les sessions. Ses composantes fonctionnelles sont les suivantes :

- P-CSCF : c'est le point de contact entre l'infrastructure du réseau matériel et le domaine IMS. Son rôle est, principalement, d'être le point d'entrée et de sortie, sur le système IMS, des messages SIP qu'il compresse ou décompresse, avant de les redistribuer aux bonnes entités.
- I-CSCF : ce point du réseau sert de passerelle entre deux systèmes IMS. Il permet d'assigner à un utilisateur un S-CSCF.
- S-CSCF : ce point du réseau prend en charge le contrôle des sessions des utilisateurs afin que ces derniers puissent faire appel à des services. Sa tâche est principalement l'établissement des sessions, la prise de décision pour le routage ou encore la vérification du contenu des médias. Ce module représente le nœud central d'un système IMS qui relie non seulement plusieurs modules P-CSCFs, mais aussi d'autres modules IMS tels que HSS et des serveurs d'application multimédia.

La deuxième partie importante de cette couche du système IMS est la base de données HSS (Home Subscriver Server). Cette base de données regroupe les informations essentielles sur tous les utilisateurs ainsi que leurs données d'identification, leurs autorisations pour les différents services et leur localisation.

Comme le décrit Grangeversanne et al. (Grangeversanne), la couche applicative du système IMS héberge sur différents serveurs d'application les services gérés par le système IMS. Tous ces modules communiquent en utilisant le protocole SIP, sauf le HSS qui utilise le protocole DIAMETER pour communiquer avec l'I-CSCF et le S-CSCF. On retrouve cela sur la figure 1.8. Le protocole SIP est très simple d'utilisation et à l'avantage d'être portable, il est donc utilisable aussi bien avec les téléphones portables qu'avec les téléphones intelligents. L'utilisation de DIAMETER est privilégiée avec la base de données HSS, car il fait partie des protocoles sécurisés qui permettent l'authentification, l'autorisation et la

localisation des utilisateurs comme l'expliquent les auteurs de ce livre blanc (InterLinkNetworks 2002).

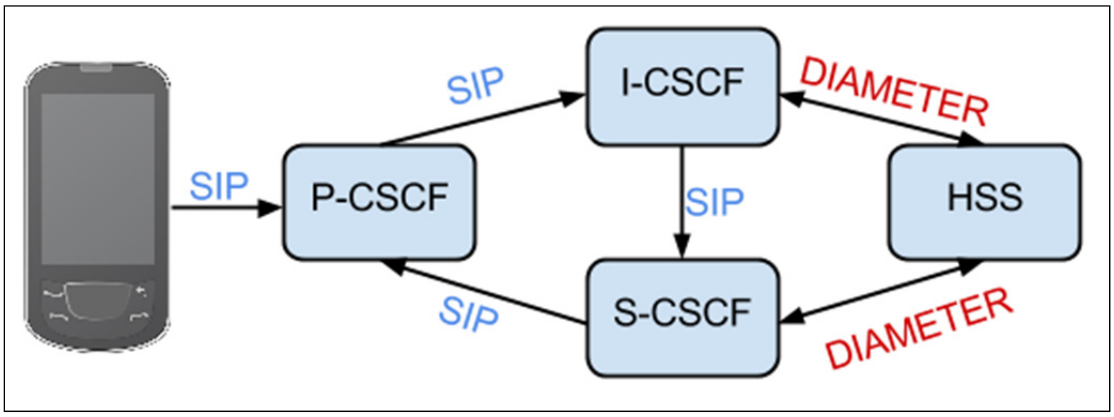

Figure 1.8 Résumé des protocoles utilisés entre les composants IMS

Les systèmes IMS sont décrits par Ericsson (Ericsson 2007) comme ayant une architecture horizontale. Ils permettent donc la réutilisation de fonctions communes de base et une forte interaction entre les différents services de la couche applicative. Pour cela, le système dissocie la technologie pour accéder à un service et la prestation de ce service. Il sépare aussi les fonctions de contrôle et de transport des données. Grâce à ces caractéristiques des systèmes IMS, leur gestion est simplifiée ce qui peut entraîner des économies d'argent. Ces caractéristiques laissent également penser que les systèmes IMS permettront aux opérateurs de créer et de déployer de nouveaux services innovants comme la communication IP en temps réel via de simples plug-ins (Ozçelebi, Radovanovic, and Lukkien 2007), à l'aide d'un Framework (Oumina and Ranc 2007) ou encore suivant le principe SOA (Lee et al. 2009).

1.3.2 La virtualisation d'un système IMS

La virtualisation serait une technique très utile dans le cadre des systèmes IMS. En effet, comme le mentionnent les auteurs du livre blanc sur la virtualisation des réseaux (Chiosi 2012), les systèmes IMS, principalement utilisés par des fournisseurs de téléphonie, ont besoin d'être distribués et robustes pour supporter le trafic de tous les clients. Ils ont aussi

besoin d'être flexibles pour offrir différents types de déploiement, par exemple la sélection d'un sous-ensemble de composants IMS pour une application multimédia donnée. De plus, la demande de service des clients varie et évolue rapidement et pour lancer un nouveau service il est souvent nécessaire de disposer de davantage de ressources. Le coût croissant de l'énergie et le cycle de vie de plus en plus court des machines font de la virtualisation une technique très intéressante, car elle permettrait d'exécuter tous les services voulus sur des parcs de machines ayant une grande puissance et dont le matériel est régulièrement changé pour rester à la pointe de l'innovation. Certains auteurs, comme Lu et Pan (Lu et al. 2013), ont cherché à virtualiser les systèmes IMS. Ces auteurs ont proposé une plateforme dans le nuage pour virtualiser le cœur d'un système IMS. Le cœur du système IMS est la partie du système utilisée pour le contrôle des sessions et des médias. Leur solution présente l'avantage d'équilibrer automatiquement la charge de travail et de gérer dynamiquement les ressources du système grâce à un système de migration à chaud des machines virtuelles pour éviter de violer le SLA. L'inconvénient de leur solution est le temps de réponse élevé du système, de l'ordre d'une trentaine de secondes, pour réagir lors d'une surcharge de travail.

1.3.3 Les caractéristiques d'un système IMS virtualisé

Les ressources énergétiques sont de plus en plus coûteuses et toutes les ressources des machines déployées pour la virtualisation d'un système IMS ne sont pas utiles au même moment. Par exemple, si un module IMS possède une capacité de traitement CPU trop grande par rapport à la charge du trafic qu'il doit traiter, alors nous devons lui enlever un certain nombre de cœurs CPU afin de les rendre disponibles pour d'autres modules ou d'autres systèmes. Comme l'analyse L. Bonnet (Bonnet-Bearstech), éteindre les machines pendant les périodes où elles ne sont pas utilisées représenterait une importante économie d'argent et éviterait une pollution inutile de notre environnement. Les systèmes IMS, par leur structure, peuvent être virtualisés et ainsi bénéficier de certaines caractéristiques des nuages informatiques. La mise à l'échelle et l'élasticité de ces derniers sont les principales caractéristiques exploitées dans les travaux sur le sujet. Bellavista et al. (Bellavista et al. 2012) ont dans cet article déployé un système IMS dont les ressources s'adaptent

automatiquement et dynamiquement en fonction du trafic que les auteurs génèrent eux-mêmes. Umair a, pour sa thèse (Umair 2013), virtualisé un système IMS afin de pouvoir implémenter une base de données pour la partie HSS de l'infrastructure IMS qui puisse grossir selon le besoin et avoir les ressources nécessaires pour satisfaire tous les appels émis par les utilisateurs. En effet, il a été remarqué qu'un sous dimensionnement du HSS peut faire augmenter les temps de latence dans l'échange d'information. Virtualiser pour utiliser la caractéristique élastique et la possibilité d'évolutivité des infrastructures dans le nuage informatique nous permettra de faire varier la taille de notre système au besoin à un instant t. Ainsi nous pourrons disposer des ressources nécessaires en cas de forte influence sur le réseau et réaliser des économies d'énergie en éteignant certaines ressources lorsque ces dernières ne sont plus utilisées.

Un projet open source, nommé OpenIMSCore (OpenSourceIms), a été lancé en 2006 pour promouvoir la technologie IMS et permettre, principalement aux projets de recherche universitaire, de déployer leur propre système IMS. Ce projet offre une implémentation open source des fonctions indispensables à IMS.

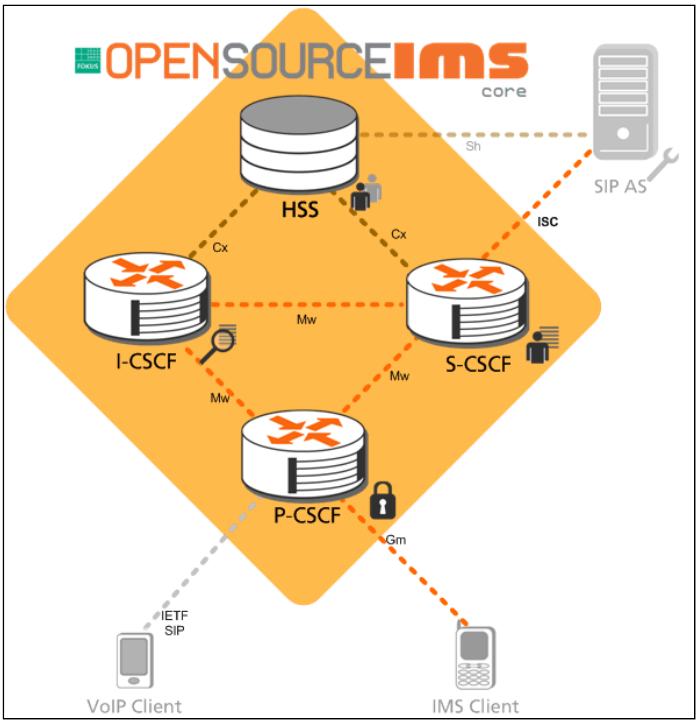

Figure 1.9 Architecture IMS d'OpenIMS Core
Tirée de OpenIMS Core (OpenSourceIms)

Comme le montre la figure 1.9, on y retrouve la base de données des utilisateurs (HSS) et les trois modules CSCF implémentés sur un serveur monolithique. Nous utiliserons cette solution dans notre projet de recherche pour déployer un système IMS virtualisé et distribué. Lorsque nous aurons construit notre système, nous chercherons à l'analyser. Pour cela, nous souhaitons pouvoir étudier ce système et le comportement de ses ressources physiques telles que le CPU et la mémoire, en fonction du trafic reçu.

1.4 Analyse du trafic et des ressources physiques d'un système IMS

Le système sur lequel nous allons travailler est un système IMS. Nous souhaitons pour notre étude en faire un système distribué et virtualisé. Lorsque nous aurons déployé notre système, nous aurons besoin d'étudier son comportement en fonction du trafic qu'il reçoit. Le fait de virtualiser le système IMS nous permettra, comme nous l'avons mentionné précédemment et comme l'explique L. Liao et al. (Liao, Leung, and Chen 2014), de pouvoir gérer

dynamiquement les ressources physiques du système. Apprendre à le connaître est donc essentiel pour détecter les périodes où des ressources physiques non nécessaires peuvent être retirées. Cela nous permettra aussi d'allouer plus de ressources au système si besoin. Le but étant de les lui allouer avant qu'il ne soit en famine afin d'assurer en permanence une bonne qualité de service pour les utilisateurs d'un tel système.

1.4.1 Télémétrer les ressources

Les ressources consommées par la virtualisation d'un système IMS sont nombreuses. Il y a donc un grand nombre de valeurs que nous pouvons relever à partir de différentes mesures. Ces mesures peuvent être faites à différents niveaux. Il peut, entre autres, être intéressant de télémétrer l'utilisation du CPU, le temps de réponse d'une machine, la bande passante, le débit, le taux d'arrivée des trames, l'utilisation de la mémoire ou encore la consommation d'énergie. Ces mesures peuvent être faites à différentes granularités, en effet elles peuvent être réalisées en prenant comme élément mesuré un composant d'une machine, une machine virtuelle, une machine hôte ou même le système IMS en général.

Tous les fournisseurs d'infrastructure ont leur propre système pour télémétrer les ressources consommées par leurs clients que ce soit en mémoire, en bande passante ou en puissance de calcul. Dans le nuage informatique, certains outils open source ont été proposés afin de permettre à tous les usagers de suivre la consommation de ressources qu'ils utilisent. Nous retrouvons parmi ces projets :

- *Xen Server Monitoring* ('Manage Engine') qui permet de télémétrer l'utilisation du CPU, de la mémoire et des écritures et lectures sur le disque de chaque machine virtuelle déployée. Ce projet s'adresse plus particulièrement à tous les systèmes virtualisés basés sur le logiciel *Xen* qui est un hyperviseur de machine virtuelle.
- Linux qui propose de virtualiser des ressources grâce à la technique de l'isolation avec son système nommé Linux Containers (LXC). Cette technique repose sur la fonctionnalité des *CGroups* (control groups) du noyau Linux. Ces *CGroups* permettent, entre autres, de limiter et d'isoler des ressources physiques d'une

machine utilisée avec le système d'exploitation Linux. Grâce à ces caractéristiques, il est possible de contrôler les ressources utilisables par un conteneur LXC. D'après Oracle (Oracle), ces *CGroups* peuvent être eux aussi télémétrés en observant la valeur de certains paramètres comme les entrées/sorties, la mémoire ou le CPU. Une simple ligne de commande donnée dans la documentation Oracle permet en console de visualiser la valeur de la métrique voulue. Il serait alors envisageable, comme cela a déjà été fait de nombreuses fois, par exemple par Deloumeau et Tanzilli ('LXC Web Panel' 2013), de développer une application pour relever les valeurs de ces paramètres.

- *OpenStack*, qui offre un ensemble de logiciels permettant de déployer des infrastructures dans le nuage. Parmi tous les logiciels proposés, il y en a un, *Ceilometer* (OpenStack), qui permet de télémétrer, quelle que soit la technique de virtualisation mise en place, les ressources consommées par ces infrastructures.
- *Docker* (Yegulalp 2014), un logiciel libre, qui permet de créer des conteneurs en s'appuyant sur l'isolation et les capacités du noyau Linux (surtout LXC et les *CGroups*). L'avantage de *Docker*, selon (Avram 2013), est donc de reprendre le format standard des conteneurs Linux, LXC, et de proposer en plus une API de haut niveau. Grâce à ce logiciel, il est alors possible de créer des conteneurs mais aussi de gérer leurs ressources comme cela est possible avec les *CGroups*. L'utilisateur peut aussi exécuter des processus en les isolant les uns des autres grâce à la solution de virtualisation utilisée.

Les outils présentés ici peuvent être utilisés pour suivre les ressources physiques d'un système IMS. En effet, dans notre recherche, et comme l'ont fait L. Liao et al. (Liao, Leung, and Chen 2014) et M. Fakhfakh et al. (Fakhfakh et al. 2009), *Xen* peut être utilisé pour virtualiser des éléments du système IMS tel que le HSS. L'outil de télémétrie associé permet donc de suivre les ressources des éléments du système ainsi virtualisé. D'autres modules de notre système IMS, comme les CSCF, seront virtualisés en utilisant le logiciel libre OpenIMSCore dans des LXC. Nous pourrons alors utiliser les *Cgroups* pour suivre les ressources consommées par ces modules et contrôler leur accès à ces ressources.

1.4.2 Mesure de l'efficacité d'un système de communication

L'efficacité des systèmes de communication dépend de la manière dont se déroulent les échanges d'informations. Pour évaluer l'efficacité d'un système, il y a donc certaines métriques qui peuvent être mesurées pour surveiller la qualité d'une transmission de données. Parmi toutes les mesures possibles, nous allons lister les cinq qui, d'après la littérature, sont les plus utilisées.

- **Le débit moyen** Ce débit représente la moyenne des vitesses de transmission des paquets sur le réseau dans un intervalle de temps fixé.
- **Le délai de transmission** Ce délai correspond au temps nécessaire à un nœud pour envoyer un paquet de données sur le réseau.
- **Le temps de latence** Ce temps correspond au temps nécessaire à un paquet de données pour passer d'un nœud A à un nœud B du réseau.
- **Le temps de réponse** Ce temps correspond au temps écoulé entre le moment où la source d'un paquet de données l'envoie sur le réseau et celui où elle reçoit la réponse du nœud destinataire du paquet. C'est donc la somme des délais de transmission et des temps de latence lors d'un échange entre deux nœuds du système.
- **Le taux d'insatisfaction** Ce taux représente le pourcentage de clients qui n'ont pas pu avoir accès au réseau par manque de ressources.
- **Le taux de perte** Ce taux représente le pourcentage de paquets perdus lors de leur transition sur le réseau.

Toutes ces métriques peuvent faire partie du contrat de service SLA. Dans ce cas, les valeurs sont fixées dans le contrat et les métriques associées ne doivent pas les dépasser. Si une limite est franchie, il y a violation du contrat de service, car la qualité de service du système est inférieure à la qualité attendue.

1.4.3 Prédiction de la charge de trafic

Selon Silvestri (Silvestri 2014), l'idée générale pour la prévision du trafic sur le réseau est d'abord d'observer l'évolution du trafic dans un intervalle de temps proche de l'instant t.

Cette observation permet de connaître l'évolution du réseau en termes d'utilisation des ressources et, par la suite, de proposer une prévision avec une incertitude la plus faible possible.

Tout d'abord pour connaître l'évolution d'un système proche de l'instant présent il faut le télémétrer. Certains outils, comme *Xen server monitoring* ou *Docker,* mentionnés dans la sous-section 1.4.1, permettent de le faire.

Il faut ensuite, à partir des valeurs mesurées, prédire le trafic à venir afin d'avoir une idée des ressources qui vont être nécessaires. Par exemple, Powers et al. (Powers, Goldszmidt, and Cohen 2005) proposent des techniques d'exploration de données (data mining) pour prédire les périodes où le CPU, la mémoire, la bande passante et les entrées/sorties d'un système informatisé seront exploités au-delà des seuils fixés. Comme le dit Silvestri (Silvestri 2014), une autre méthode pour prévoir le trafic serait d'avoir des modèles de prédictions auxquels se référer. C'est l'idée reprise en 2015 par A. Khan et al. (Khan et al. 2012) qui ont proposé une méthode de prédiction de la charge de travail des machines virtuelles d'un nuage privé grâce à la détection de modèles du trafic. Pour construire ces modèles, ils s'appuient sur des données récoltées sur un nuage informatique réel. Ce dernier est partagé par environ 1000 entreprises qui utilisent chacune entre 100 et 5000 machines virtuelles. Leur article montre que la détection de modèle et la prédiction basée sur l'automate de Markov à états cachés, hidden markov models en anglais, permettent d'obtenir des résultats fiables. L'idée de modèle a également servi à N. Roy et al. (Roy, Dubey, and Gokhale 2011). Pour ces chercheurs, les modèles construits lors de leur étude ne se basent pas sur d'anciennes données collectées au cours de précédents enregistrements, mais sur les données récoltées lors d'un enregistrement courant des métriques d'un système. En effet, ils tronquent les données enregistrées depuis le début de la surveillance de leur système en sous-jeux de données. Chaque sous-jeu constitue un enregistrement pour la détection de modèle. Cette détection, liée à l'utilisation du modèle statistique ARMA, pour autoregressive moving average, leur permet de comprendre et de prédire le nombre de machines virtuelles dont ils

vont avoir besoin pour satisfaire la charge de travail imposée par tous les clients connectés sur le nuage informatique qu'ils surveillent.

1.5 Les statistiques comme outil de prédiction

D'un point de vue théorique, la statistique est à la fois une science, une méthode et une technique. Souvent utilisées au pluriel pour montrer la diversité de la science, les statistiques sont très utilisées en informatique. Comme le prétend JW. Tukey, un des plus grands statisticiens du vingtième siècle et inventeur de la boîte à moustaches, il y a deux approches en statistiques. Les statistiques exploratoires qui visent à l'exploration des données afin d'avoir une idée qualitative de leurs propriétés et les statistiques confirmatoires. Ces dernières viennent après une étude exploratoire qui permet de faire des hypothèses et sert à les confirmer ou à les infirmer en recourant à d'autres techniques statistiques. Une étude statistique commence donc par la collecte de données, qui dans notre cas est traitée dans la rubrique 1.4.1 de ce mémoire, et se poursuit par le traitement et l'interprétation de ces données. Lors de notre étude, nous avons cherché à prédire les ressources dont va avoir besoin le système pour fonctionner de manière fiable et sans gaspillage. Comme nous pouvons le lire ci-dessus, à la section 1.4.3., certains chercheurs gèrent leurs ressources en termes de machines virtuelles et se servent de modèles pour définir si une machine virtuelle de plus va ou non être nécessaire au bon fonctionnement de leur système. Dans notre cas, la granularité est plus fine. En effet, nous ne nous intéressons pas aux machines virtuelles mais à leurs ressources telles que le CPU ou la mémoire. Nous nous sommes alors intéressés à la régression. La régression est une branche des statistiques qui regroupe les méthodes de prédiction quantitative. Le terme régression, introduit à la fin du dix-neuvième siècle par le statisticien britannique Francis Galton, est l'ensemble des méthodes statistiques utilisées pour analyser la relation d'une variable par rapport à une ou plusieurs autres. Dans la suite de ce mémoire, nous verrons que nous avons lié le temps au nombre d'appels par seconde et le nombre d'appels par seconde à la charge du CPU. Dans la littérature, on retrouve souvent le temps lié à la charge du CPU lors d'analyse de processus autorégressif. Durant notre travail, nous avons utilisé la régression linéaire et exponentielle. Nous parlerons donc de ces

régressions qui nous ont servi, dans ce mémoire, à proposer un outil de prédiction du CPU. De plus, nous avons défini des modèles de prédiction basés sur les techniques de machines à vecteur de support (SVM). Nous avons ensuite comparé les résultats obtenus à partir des différents modèles proposés. Nous présenterons donc aussi les méthodes SVM dans cette revue de la littérature.

1.5.1 Les méthodes de régression linéaire et non-linéaire

Le besoin de prédiction des ressources d'un système fait l'objet de nombreuses études. Certaines d'entre elles s'appuient sur des modèles de régressions linéaires et non-linéaires. Ces types de régression sont généralement employés sur des séries temporelles. La simplicité de ces méthodes dont le but est de trouver l'équation de la fonction qui représente au mieux les données relevées lors d'une observation fait qu'elles sont très utilisées. En effet, elles ne nécessitent pas de lourd calcul et peuvent donc être déployées facilement dans tous les outils de prédiction implémentés. PA. Dinda et DR. O'Hallaron utilisent, dans leur article (Dinda, 2000), la régression linéaire pour prédire la charge de travail de leurs machines hôtes en fonction du temps. Le modèle linéaire est souvent utilisé sous la forme du modèle ARMA, *autoregressive-moving-average* en anglais, proposé par P. Whittle en 1951 (Whitle 1951). Ce modèle est un processus temporel discret principalement utilisé pour comprendre et éventuellement prédire les futures valeurs d'une série. Y. Zhang et al. utilisent, quant à eux, la régression polynomiale pour déterminer une fonction qui suit au plus près leurs relevés. Dans leur article (Zhang, Sun, and Inoguchi 2006), ils utilisent ensuite cette fonction pour prédire la charge du CPU sur leur grille informatique. Une grille informatique est, selon Z. Juhasz et al. (Juhasz, Kacsuk, and Kranzlmüller 2004), « une infrastructure virtuelle constituée d'un ensemble de ressources informatiques potentiellement partagées, distribuées, hétérogènes, délocalisées et autonomes ». Les résultats de prédiction présentés par Y. Zhang et al. semblent bons, car la moyenne de leurs erreurs absolues est toujours inférieure à 12% et atteint même pour certains jeux de données seulement 0.08%. Y. Jiang et al. montrent dans leur article (Jiang, Perng, et al. 2013) que les résultats des méthodes présentées ici varient beaucoup selon le jeu de données. Pour cela, ils ont utilisé la régression linéaire qu'ils

comparent à d'autres méthodes comme celles des machines à support de vecteur (SVM). Dans la moitié de leur cas, les résultats obtenus avec la régression linéaire sont comparables à ceux obtenus avec les méthodes SVM, mais dans l'autre moitié des cas ils sont environ trois fois plus mauvais. Les méthodes SVM, bien que pas toujours fiable d'après leurs résultats, le sont beaucoup plus. Nous allons donc les présenter ci-dessous. Dans le chapitre 3 de ce mémoire, nous avons utilisé la régression non-linéaire, et plus particulièrement la régression exponentielle pour la prédiction de la charge du CPU consommé par le système IMS. Nous verrons que cette dernière nous a permis de prédire le CPU en fonction du nombre d'appels par seconde de manière relativement fiable. Nous avons aussi comparé nos résultats avec ceux obtenus avec l'approche SVM.

1.5.2 Les méthodes de régression des machines à support de vecteur

Les techniques de séparateurs à vaste marge, *Support Vector Machine* en anglais et abrégées SVM, souvent citées dans la littérature, semblent être de bonnes méthodes de prédiction bien que ce soit d'abord des méthodes de classification. Elles reposent, comme le décrivent KR. Müller et al. (Müller et al. 2001), sur deux notions clés : la marge maximale et la fonction noyau. La première notion, la marge, est la distance entre la courbe construite pour séparer deux types de données parmi toutes celles relevées lors d'une observation et la frontière à l'intérieur de laquelle on ignore les données qui ne seraient pas du bon côté de la courbe. Les points sur la frontière forment les vecteurs supports. On parle ici de marge maximale parce qu'il a été montré par la théorie de Vapnik-Chervonenkis que la meilleure frontière de séparation est celle qui maximise la marge, car elle minimise la complexité de la fonction mathématique qui caractérise les données. Le problème est donc de déterminer la frontière séparatrice optimale grâce à un jeu de données d'entraînement afin que l'algorithme puisse ensuite prédire le type des prochains résultats. Dans le cas où les données sont linéaires, des algorithmes d'optimisation quadratique permettent de résoudre le problème. Dans les autres cas, on utilise la deuxième notion des SVM, la fonction noyau. Cette deuxième notion consiste en une transformation de l'espace de représentation des données d'entrée. Le but est d'obtenir un espace de plus grande dimension dans lequel la fonction construite pour séparer

les données relevées sera linéaire. Dans la pratique, on ne connait pas la transformation à appliquer pour obtenir l'effet voulu. La fonction noyau est donc utilisée. Elle permet de réaliser les calculs pour passer d'un espace à un autre et de déterminer la fonction cherchée pour séparer les données. Les noyaux usuels les plus utilisés avec SVM sont le noyau linéaire, le noyau polynomial et le noyau gaussien. Dans la littérature, personne n'a montré qu'un certain noyau donne toujours de meilleurs résultats. Dans la suite de ce travail, nous avons choisi d'utiliser le noyau gaussien, nommé RBF, car ce dernier est, comme l'explique CW. Hsu et al. (Hsu, Chang, and Lin 2003), un bon choix par défaut pour les problèmes non linéaires avec peu de caractéristiques. Cela est notre cas, car nous nous préoccupons que du CPU. Comme mentionné précédemment les SVM permettent de résoudre des problèmes de discrimination, mais aussi, grâce à certaines méthodes, ils peuvent être appliqués pour résoudre des problèmes de régression.

Les méthodes SVM commencent à être utilisées pour la prédiction des ressources dans les réseaux de communications, mais elles le sont depuis longtemps pour des prédictions en temps réel dans des domaines tels que la finance (Adhikari and Agrawal 2013) ou la chimie (Fan, Li, and Song 2006). P. Huang et al. utilisent également les méthodes de régression des SVM dans leur article (Huang et al. 2015) pour prédire la charge de travail de leur réseau. Pour tester leur méthode, ils ont utilisé des données relevées pendant un mois sur un centre de données Google composé de 12 000 hôtes hétérogènes. Leur étude montre que SVM est une méthode fiable pour la prédiction des besoins CPU sur un nuage informatique. En effet, leur prédiction avec dix minutes d'avance a une précision de 80%. Cette dernière est au plus bas pour une prédiction avec une heure d'avance, 70%, puis remonte jusqu'à 75% pour une prédiction avec trois heures d'avance. L'erreur quadratique moyenne varie quant à elle entre 1.5 et 2.7 pour leur étude. Elle est la plus faible pour une prédiction avec dix minutes d'avance et la plus haute pour une prédiction avec une demi-heure d'avance. Dans le cadre de ce projet, nous proposons d'appliquer ces méthodes pour la prédiction des ressources dans un système IMS virtualisé et distribué.

1.6 Conclusion du chapitre

Dans ce chapitre, nous avons fait un tour d'horizon sur la virtualisation, l'architecture des systèmes IMS et la prédiction du trafic sur celui-ci. Nous avons vu que le nuage informatique et la virtualisation sont extrêmement liés. En effet, la virtualisation permet la personnification du nuage rendant ainsi possible de le particulariser à un besoin qui peut fluctuer dans le temps. Pour cela, la virtualisation permet, par exemple, grâce à son élasticité, de faire évoluer un système virtuel dynamiquement en ajoutant ou en supprimant des ressources à la demande. Il est alors possible de créer un nuage personnalisé, aux capacités contrôlées ou, au contraire, de mettre à la disposition de tous une capacité de calcul et un espace de stockage quasiment infini. Tout cela sera intéressant dans notre travail de recherche, car nous voulons qu'un système IMS virtualisé gère de manière autonome et dynamique ses ressources afin d'en optimiser l'utilisation. Une optimisation des ressources physiques du système IMS permettrait de réduire le nombre de machines nécessaires à son bon fonctionnement. Avec moins de machines, le système serait alors plus rentable et son impact sur l'environnement serait plus faible. Pour atteindre cet objectif, nous nous concentrerons dans le prochain chapitre sur le développement d'un outil de gestion de ressources pour système IMS virtualisé. Pour cela, nous commencerons par étudier l'existant puis nous proposerons une architecture pour notre outil qui s'adapte le mieux possible à la virtualisation par isolation que nous avons choisie pour notre système IMS.

CHAPITRE 2

OUTIL DE GESTION DU SYSTÈME IMS

Dans ce chapitre, nous allons présenter les motivations qui nous ont incitées à nous intéresser à l'analyse de mécanismes de surveillance de trafic et de gestion de ressources dans un système IMS virtualisé. De plus, nous verrons également quelques méthodes déjà proposées pour gérer dynamiquement l'allocation et la libération de ressources dans un système virtualisé. Ensuite, nous présenterons l'architecture de notre outil de gestion dynamique et proactive des ressources d'un système IMS virtualisé.

2.1 Motivations

Les outils les plus fonctionnels de gestion dynamique du système IMS actuellement proposés sont des systèmes réactifs. Ils attendent pour allouer ou libérer des ressources de franchir un certain seuil d'utilisation. Bellavista et al. (Bellavista et al. 2012) proposent, par exemple, un outil pour gérer de manière dynamique un système IMS en fonction de la valeur de la charge du CPU. Ils utilisent cette ressource comme référence, car ils ont remarqué qu'elle était la ressource la plus critique des systèmes IMS. Mais dans ce cas, comme dans celui de Nemati et al. (Nemati 2014), nous constatons que la mise à niveau du système, c'est-à-dire le déploiement de nouvelles ressources physiques utilisables, s'effectue quand le système est déjà en manque de ressources. L'objectif ici est donc de construire un outil proactif qui permettrait d'anticiper ce manque, ou l'abondance de ressources, avant que ce ne soit le cas. Nous proposons de développer cet outil pour les systèmes IMS virtualisés.

De nombreux travaux de recherche ont proposé différentes solutions pour la gestion dynamique des systèmes de nuage informatique. En effet, certains chercheurs ont déjà construit des systèmes proactifs. Jiang et al. (Jiang, Perng, et al. 2013), par exemple, veulent pouvoir créer ou supprimer des VM sur leurs machines physiques, afin de réagir à la variation de la charge de trafic de manière automatique. Étant donné qu'il est impossible de demander à un utilisateur d'annoncer son utilisation future des ressources d'un système, ils

ont alors défini un moyen de la prédire. Ils ont construit leur système de prédiction en se basant sur une mesure qu'ils ont appelée le coût de prédiction du nuage (CPC, Cloud Prediction Cost). Leur but est de minimiser ce coût. Pour mesurer cette valeur, ils proposent de calculer la différence entre les ressources que dispose le système à un instant t et les ressources dont il aurait besoin. Si le système dispose de plus de ressources que nécessaire il y a des coûts de gaspillage des ressources, si le système manque de ressources les coûts viennent de la pénalité financière à reverser pour avoir violé le SLA. En fonction de l'évolution de cette mesure, ils prédisent ce qu'il faut faire pour les secondes à venir. C'est-à-dire le nombre de VM qu'il faut éteindre ou allumer pour réduire au maximum leur fonction de coût. Leur solution fonctionne relativement bien, car leurs VMs sont créées en quelques secondes. Nous pouvons cependant voir dans la présentation de leurs résultats que le nombre de VM en activité est très variable. Leur système semble passer beaucoup de temps à construire et détruire des VM et nous pouvons nous demander si cela ne consomme pas aussi des ressources du système. Malgré cela, leur travail a été fructueux, car leurs résultats expérimentaux démontrent que leur approche permet d'améliorer la qualité de service de leur système tout en conservant un faible coût dû au manque ou à l'abondance de ressources.

Jiang et al. (Jiang, Lu, et al. 2013) ont, quant à eux, construit un système de prédiction du trafic sur Internet en se basant sur des techniques d'apprentissage automatique (machine learning) pour analyser des enregistrements de trafic et ainsi prédire celui à venir. Grâce aux modèles déterminés lors de leur analyse du trafic et en cherchant à minimiser une relation qui lie le coût de fonctionnement de leur système et son temps de latence, ils prédisent les ressources physiques nécessaires à leur système pour supporter le trafic à venir. Leur solution permet également de réduire le nombre de violations du SLA tout en optimisant le coût de fonctionnement du système.

Pour finir, Gong et al. (Gong, Gu, and Wilkes 2010) proposent une solution pour gérer dynamiquement les ressources du nuage informatique en prédisant le trafic. Pour cela, ils ont relevé un ensemble de modèles d'évolution du trafic sur le nuage informatique en s'appuyant sur des traces du trafic réel de Google. Ils ont ensuite développé un outil, PRESS (PRedictive

Elastic reSource Scaling), pour télémétrer la charge du CPU, la mémoire, la bande passante et les entrées/sorties d'un réseau afin d'identifier le modèle qui s'adapte le mieux au cas présent. Si un des modèles précédemment relevés correspond à l'état actuel du réseau, ils alloueront à ce dernier autant de ressources qu'il en était nécessaire au bon fonctionnement du réseau lors de l'enregistrement de leur modèle. Si aucun modèle ne s'adapte, ils utilisent des techniques de statistique, dont les chaînes de Markov, pour prédire l'évolution du trafic. Selon leur analyse, leur solution consomme peu de ressources et permet une bonne prédiction des ressources nécessaires à leur réseau. En effet, leur prédiction sous-estime le nombre de ressources nécessaires d'au maximum 5%. Une sous-estimation peut engendrer une violation du SLA. Pour éviter ce problème, ils ajoutent une marge pour avoir toujours un peu plus de ressources disponibles que ce qui est vraiment nécessaire. Cette marge est utile, car même si le trafic sur leur réseau suit régulièrement les mêmes modèles, il reste spontané et une fluctuation peut survenir à tout moment. Leur solution, d'après leurs résultats, ne surestime quasiment pas le nombre de ressources nécessaires au réseau qu'ils gèrent. Le nombre de ressources gaspillées est donc nul.

La différence majeure entre ces solutions et celle que nous cherchons à construire est que leur système gère dynamiquement les ressources d'un nuage informatique en général et que nous nous souhaitons nous concentrer sur la gestion dynamique d'un système IMS virtualisé. À terme, nous souhaitons que les ressources d'un système IMS virtualisé soient adaptées sans intervention humaine et de manière proactive pour garantir une bonne qualité de service aux utilisateurs. Pour cela, nous nous inspirerons des techniques présentées ici afin de construire notre outil de gestion d'un système IMS virtualisé.

2.2 MAPE-K : un modèle de conception de systèmes autogérés

Pour le développement de systèmes autogérés, nous retrouvons dans la littérature plusieurs modèles de conception. Nous avons choisi durant ce mémoire de nous inspirer du modèle MAPE-K, car il permet de répondre à notre besoin de surveillance du système IMS virtualisé et d'adaptation de ses ressources en fonction de différentes analyses que nous pourrons faire.

Cette solution, présentée par Kephart et Chess (Kephart and Chess 2003), propose un cycle de quatre phases :

- la télémétrie des ressources, d'où le M pour monitore
- l'analyse des valeurs relevées, d'où le A pour analyze
- la planification, d'où le P pour planning
- l'exécution du plan planifié, d'où le E pour execution

Toutes ces phases partagent une base de connaissance, d'où le K pour knowledge. Cette base de connaissance permet le partage de valeurs entre les quatre phases du cycle (MAPE). Elle permet aussi de sauvegarder des informations sur le système suivi, l'historique des décisions prises lors des cycles MAPE et les symptômes reliés à un problème du système géré.

Cette solution est, par exemple, utilisée par Bjorkqvist et al. (Bjorkqvist, Chen, and Binder 2012) qui proposent une méthode pour équilibrer la charge de travail d'un nuage informatique en distribuant le trafic vers les machines virtuelles les moins sollicitées. Leur méthode propose aussi un contrôleur qui surveille les taux d'arrivée des requêtes et la performance des machines virtuelles afin de réaliser une mise à niveau dynamique des ressources et la facturation.

Mao et al. (Mao, Li, and Humphrey 2010) utilisent aussi le modèle MAPE-K dans leur mécanisme de mise à l'échelle automatique de leur nuage informatique. Selon eux, le problème d'aujourd'hui n'est plus la performance, car nous pouvons théoriquement avoir un nombre illimité de ressources, mais le budget, car l'utilisation de ces ressources coûte cher. Ils proposent un mécanisme d'adaptation des ressources du nuage basé sur quatre composants : le premier suit les performances de leur nuage, le second enregistre un historique des décisions prises pour augmenter ou diminuer les ressources disponibles du nuage suivi, le troisième est le composant qui prend les décisions en fonction des informations obtenues grâce aux deux composants précédents, le dernier gère les VMs, il les allume ou les éteint en fonction de la décision prise par le troisième composant.

Plus récemment, S. Zareian et al. (Zareian et al. 2015) ont repris le modèle MAPE-K pour développer un outil nommé K-Feed qui permet de monitorer et analyser les ressources des applications accessibles sur le nuage informatique afin de prédire leurs performances. Cette prédiction est construite grâce à des techniques d'apprentissage automatique, machine learning en anglais, et est réalisée grâce aux algorithmes proposés par Weka (Hall et al. 2009), une suite de logiciels d'apprentissage automatique.

2.3 Description de l'outil construit

Dans notre étude, nous avons virtualisé un système IMS. Ce système est composé des éléments CSCF et de la base de données HSS. Nous avons également déployé un simulateur pour générer du trafic sur notre système IMS. Les composants du système IMS virtualisé et le simulateur peuvent être hébergés sur une ou plusieurs machines physiques en fonction des ressources matérielles de celles-ci. Pour virtualiser les différents modules du système, le système hyperviseur, présenté à la section 1.1.3, et l'isolateur, décrit à la section 1.1.4, seront utilisés. Le fait d'utiliser ces techniques de virtualisation pour isoler les processus permet une meilleure gestion de leurs ressources, car nous pouvons les maîtriser. Cela permet aussi une meilleure sécurité des composants comme expliqué à la section 1.1.5. Dans ce contexte, nous avons construit un outil de gestion de notre système IMS dont l'architecture de base est représentée à la figure 2.1. Dans cette architecture, à cause des ressources physiques de chaque machine à disposition, nous avons choisi d'utiliser trois machines hôtes pour virtualiser tous les modules de notre système IMS. Sur une première machine, nous avons déployé un système hyperviseur pour virtualiser le simulateur de trafic. Sur une deuxième machine, nous avons virtualisé le HSS en appliquant la même technique de virtualisation que précédemment. Les systèmes hyperviseurs construits ont été réalisés avec *Xen*. Sur la troisième machine, nous avons construit des conteneurs en appliquant la technique de l'isolateur proposée par *LXC*. Chaque conteneur est un module CSCF du système IMS virtualisé.

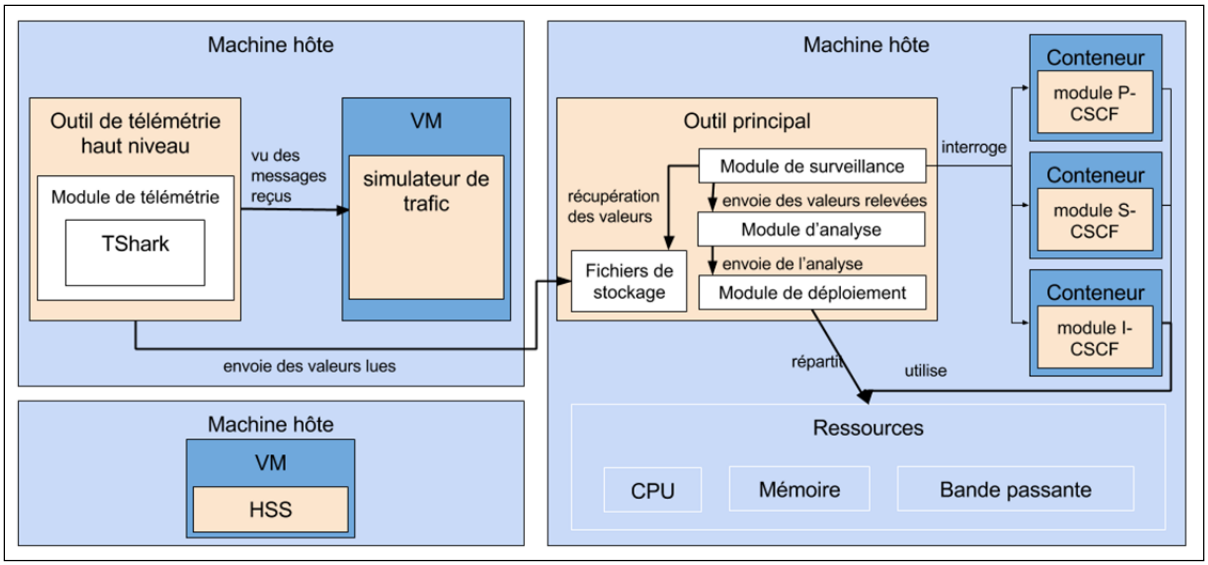

Figure 2.1 Organisation multi-outil de la gestion des ressources d'un système IMS virtualisé

Nous retrouvons, dans chaque article présenté à la section précédente, l'idée de monitorer le système. Dans notre cas, nous pouvons soit utiliser un outil comme Ceilometer (OpenStack) ou Docker (Yegulalp 2014), soit construire notre propre outil. Nous avons choisi de construire notre propre outil, car Ceilometer nous paraissait trop lourd étant donné que pour l'installer il est nécessaire d'installer beaucoup d'autres modules. Quant à Docker, il ne nous convenait pas, car nous ne pouvions pas enregistrer les valeurs qu'il reporte. De plus, les outils proposés ci-dessus ne suivent que l'état des ressources physiques et nous souhaitions aussi surveiller notre système au niveau service.

Nous avons donc choisi d'implémenter un premier logiciel pour relever les valeurs des ressources de la machine physique A et des systèmes virtualisés qu'elle héberge (modules CSCF d'un système IMS), c'est-à-dire la charge du CPU et de la bande passante ainsi que l'utilisation de la mémoire et des entrées/sorties. Ce dernier est appelé *module de surveillance* (voir figure 2.1).

Pour surveiller notre système au niveau service, nous avons développé un autre outil, afin d'observer le délai d'établissement d'une session, le temps de réponse des composants du système et le nombre de messages RINGING et BYE reçus par notre système. Ce suivi est

réalisé par le module de télémétrie de l'outil nommé *Outil de télémétrie haut niveau* installé dans la machine B (voir figure 2.1). Cet outil de télémétrie a été placé sur la machine B, car c'est aussi sur cette machine que le trafic est généré. Cela permet donc à l'outil d'être plus efficace, car il n'est ainsi pas nécessaire d'utiliser des techniques coûteuses en temps et en bande passante telles que SSH pour accéder aux informations utiles. En effet, en plaçant l'outil de télémétrie sur la même machine physique que la VM, tout est en local sur une même machine.

Selon le modèle MAPE-K, ce monitorage est suivi d'une phase d'analyse. Pour que celle-ci soit la plus probante possible, nous avons décidé que les deux outils de monitorage devaient transmettre les valeurs relevées à un même module d'analyse. L'*Outil de télémétrie haut niveau* dépose donc les valeurs collectées dans une base de connaissance. Cette dernière est représentée par *les fichiers de stockage* illustrés à la figure 2.1. Les *modules de surveillance* de l'*outil principal* transfèreront ensuite au *module d'analyse* toutes les valeurs récoltées, quel que soit le module qui les a relevées. La phase d'analyse pourra ainsi prédire les ressources physiques nécessaires au bon fonctionnement des modules CSCFs gérés. Les algorithmes développés pour cette prédiction sont détaillés dans le chapitre 3 de ce mémoire.

Les deux dernières phases du modèle, la planification et l'exécution, sont quant à elles réalisées par le *module de déploiement* (voir figure 2.1). Ce module doit pouvoir allouer et retirer à tous les éléments gérés du système IMS des ressources physiques de la machine sur laquelle ils sont déployés (machine A).

Dans la suite de notre travail, l'outil dit principal a été nommé MAA pour Monitorage, Analyse et Action. Son architecture est détaillée à l'annexe IV. À son démarrage, il récupère la liste des conteneurs actifs. L'outil doit ensuite, en même temps, récupérer les valeurs des métriques étudiées, analyser les valeurs relevées et agir en fonction de l'analyse réalisée. Afin que toutes ces tâches puissent être réalisées en parallèle, l'outil crée trois threads principaux, un par tâche. Ces threads communiquent entre eux grâce à des *flags* et des

variables globales protégées par des sémaphores. Dans la suite de la description, nous détaillerons davantage l'architecture de chaque module présenté ici.

2.3.1 Architecture du module de télémétrie des services

L'outil de télémétrie haut niveau est l'outil permettant de monitorer notre système au niveau service. Il est constitué du module de télémétrie qui, grâce à l'outil TShark, relève le délai d'établissement d'une session, le temps de réponse des composants CSCFs du système et le nombre de messages RINGING et BYE reçus par le système IMS virtualisé. Notre outil de télémétrie haut niveau se compose donc de deux threads. Un premier qui récupère les valeurs et un second qui envoie les valeurs relevées dans les fichiers de stockage de valeurs lues de l'outil principal. Plus de détails sur sa conception sont disponibles à l'annexe I.

Cette stratégie d'implémentation du module de télémétrie n'a cependant pas pu être utilisée. En effet, elle est théoriquement ce dont nous avons besoin, mais en pratique elle n'est pas suffisamment rapide pour avoir un système de gestion de ressources proactif efficace. Lorsque nous l'avons implémenté, nous l'avons testé grâce à de simples simulations, et nous nous sommes aperçus que les valeurs que nous collections n'étaient pas relevées en temps réel. Ce phénomène s'explique par l'importance du trafic reçu par le système IMS virtualisé, beaucoup de messages y circulent, et l'outil TShark n'a pas le temps de tous les analyser en temps réel. Ce goulot d'étranglement ne nous permet donc pas d'utiliser cette stratégie comme nous l'avons pensé, car nous avons besoin des données en temps réel si nous souhaitons pouvoir prédire leur évolution. Pour pallier ce problème, nous avons alors modifié l'outil haut niveau pour qu'il nous fournisse le nombre de sessions SIP établies chaque seconde par le simulateur de trafic (nombre d'appels par seconde, abrégé cps pour *call per second*) et le nombre d'entre elles qui sont interrompues de manière non attendue (SI). Cette modification de l'outil de télémétrie des ressources haut niveau a été faite car ainsi nous pouvons suivre en temps réel le trafic sur le système IMS virtualisé. Si notre outil venait a être utilisé par des entreprises qui déploient des systèmes IMS, il faudrait soit utiliser les outils qu'elles ont déjà pour suivre l'évolution du trafic sur le système, soit remettre en place

l'outil TShark, mais sur une machine suffisamment puissante pour que tout le trafic soit analysé en temps réel.

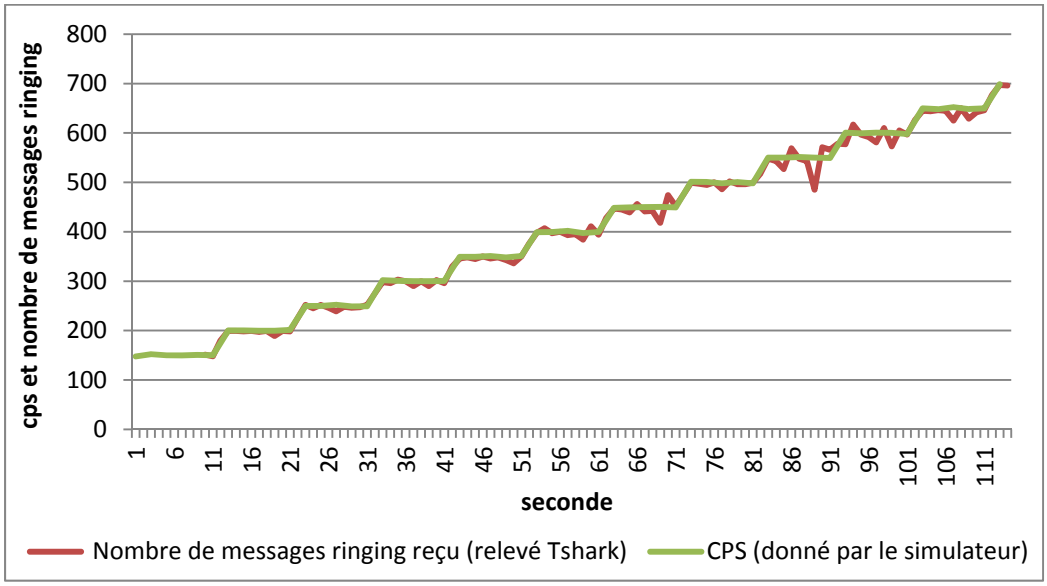

Figure 2.2 Erreur entre les valeurs relevées avec TShark
et celles retournées par le simulateur

Sur la figure 2.2, la courbe verte représente le nombre d'appels par seconde (CPS) que le simulateur envoie sur le réseau. La courbe rouge, quant à elle, reflète le nombre de messages RINGING que TShark voit passer sur le réseau. Or nous savons qu'un message RINGING est un des messages envoyés lors de l'établissement d'un appel. Si nous calculons le taux d'erreur entre ces deux courbes, nous pouvons constater que l'utilisation des valeurs renvoyées par le simulateur plutôt que celles relevées par TShark était similaire. En effet, le taux d'erreur entre les deux techniques est négligeable puisqu'il est au maximum de 13.6% (la différence maximale entre les deux courbes étant de 75cps) et en moyenne de 2.2%. Nous utiliserons donc dans notre travail les valeurs renvoyées par le simulateur.

2.3.2 Architecture du module de surveillance des ressources

Le premier des trois grands modules de notre outil principal est celui pour télémétrer le système d'un point de vue ressource. Les activités de ce module ont été représentées sur le diagramme illustré à la figure 2.3.

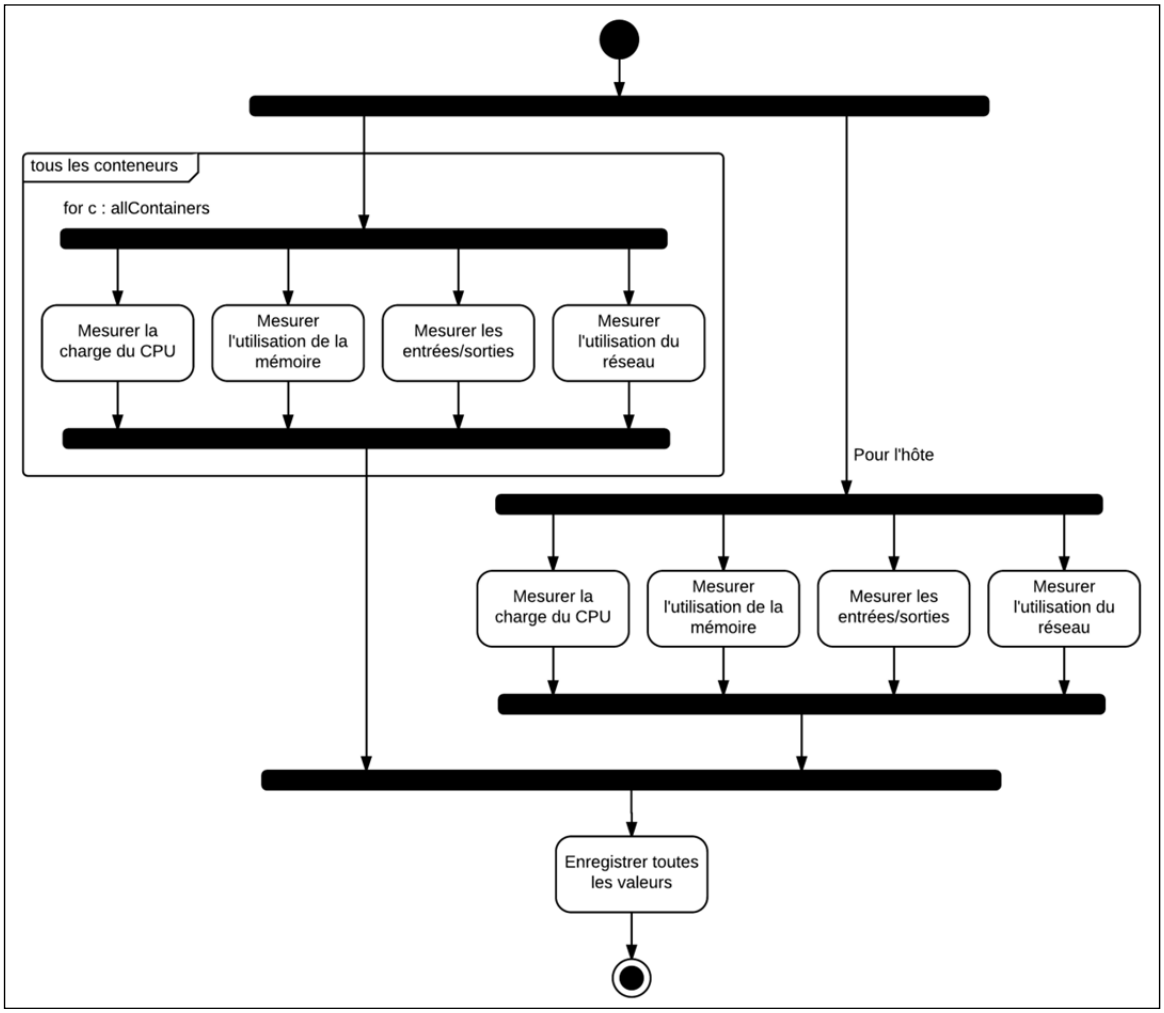

Figure 2.3 Diagramme d'activité du module de surveillance des ressources bas niveau

Nous remarquons sur ce diagramme que l'outil doit simultanément surveiller les ressources de tous les composants CSCFs du système IMS déployés dans les conteneurs ainsi que celles de l'hôte. Pour que ces tâches de télémétrie soient réalisées en parallèle, une architecture multithread a été implémentée. Un thread est créé pour chaque type de ressource et pour

chaque composant CSCF à surveiller. Un thread est aussi exécuté pour chaque type de ressources de l'hôte. L'outil est configuré afin de choisir quelle ressource de quel composant CSCF nous voulons analyser. Cette architecture nous permet d'implémenter un système qui exécute le diagramme de séquence présenté à l'annexe II. Ce diagramme nous montre que chaque thread relève à intervalle de temps régulier et communique au contrôleur de l'outil, la valeur courante de la métrique qui lui est attribuée. Les scripts utilisés pour relever les valeurs à mesurer sont présentés en annexe III. Ce module exécute aussi un thread afin de récupérer les valeurs des métriques de haut niveau dans les fichiers de stockage.

2.3.3 Architecture du module d'analyse et de prédiction

Le second module de MAA est celui d'analyse et de prédiction de l'utilisation des ressources par les conteneurs. Il prend en entrée les valeurs relevées par le module présenté à la section 2.3.2. En fonction des tendances des valeurs relevées et des algorithmes construits et détaillés au chapitre 3, le module prédit l'évolution du trafic. Ce module lance une analyse à chaque fois que le module de surveillance des ressources lève un *flag* leur permettant de se synchroniser. À la fin de l'analyse des données, si des valeurs particulières ont été relevées et impliquent une modification dans la gestion des ressources, le module de prédiction lève un *flag* lui permettant de se synchroniser avec le module de déploiement.

2.3.4 Architecture du module de déploiement

Le troisième module de MAA, à partir des prédictions faites par le module décrit à la section 2.3.3, détermine et exécute un plan d'action pour que le système IMS virtualisé dispose des ressources qui lui sont nécessaires et sans, non plus, en gaspiller. Dans la suite de ce mémoire, nous nous sommes davantage concentrés sur la gestion du CPU pour un conteneur de type S-CSCF car, comme nous l'expliquerons dans le chapitre 3, cette ressource pour ce type de composant du système IMS est la plus critique. Dans ce cas, l'outil développé se concentre donc sur l'augmentation ou la diminution du nombre de cœurs virtuels que peut utiliser ce conteneur. Seule cette ressource sera donc gérée par l'outil développé, mais en

théorie il peut aussi permettre d'adapter la capacité de mémoire et la largeur de la bande passante utilisable par chaque conteneur.

2.4 Conclusion du chapitre

Dans ce chapitre, nous avons présenté l'outil, MAA, que nous avons développé pour monitorer, analyser et ajuster les ressources du système IMS virtualisé. Nous avons aussi vu que pour suivre le trafic, il était préférable de mettre en place un sous-outil sur la machine par laquelle arrive le trafic sur le système. Le fait que cet outil soit sur la même machine physique que la VM qui génère le trafic permet des gains de performances étant donné que l'accès aux données pour leur monitorage est en local. Nous avons également décrit dans ce chapitre le modèle de conception suivi pour créer notre système autogéré. Le système MAPE-K choisi présente quatre phases que nous avons reprises dans notre outil de gestion dynamique et proactive des ressources d'un système IMS virtualisé. MAA est donc composé de différents threads qui sont tous ordonnancés par le thread principal de l'application. Ces threads permettent de relever la consommation courante en ressources des modules surveillés du système IMS virtualisé. L'un d'entre eux est destiné à l'analyse de l'ensemble des valeurs relevées. Un autre a pour rôle la planification et l'exécution d'un plan d'action pour adapter au mieux les ressources disponibles de la machine hôte aux différents modules du système IMS virtualisé. La procédure d'installation et de configuration de l'outil MAA est présentée en Annexe V.

CHAPITRE 3

ALGORITHME DE PRÉDICTION DU TRAFIC

Dans ce chapitre, nous poserons les hypothèses que nous avons faites pour la suite de notre travail de recherche. Nous décrirons également les algorithmes de prédiction du trafic développés pour un système IMS virtualisé.

3.1 Hypothèses

Dans notre étude, nous faisons l'hypothèse qu'il existe une corrélation entre les valeurs relevées au niveau des ressources et au niveau service. Afin de trouver cette corrélation, nous faisons également l'hypothèse que les valeurs relevées en temps réel traduisent bien l'état du réseau et des ressources. Nous restons cependant conscients que les ordinateurs exécutent souvent en tâches de fond et sans prévenir des processus, mais ces derniers consomment une quantité de ressources négligeable face à celles consommées par notre système IMS comme nous pouvons le voir sur la figure 3.1. Par ailleurs, dans le cas où notre système ne relèverait pas de valeur à une seconde donnée pour une métrique précise, nous faisons l'hypothèse que nous pouvons utiliser le résultat de la seconde précédente. Cette hypothèse se justifie par le fait que le nombre d'appels par seconde ne varie pas brutalement et donc que le système ne doit pas non plus évoluer soudainement (à chaque seconde). Une valeur prise à la seconde précédente doit donc être proche de celle que l'on aurait obtenue.

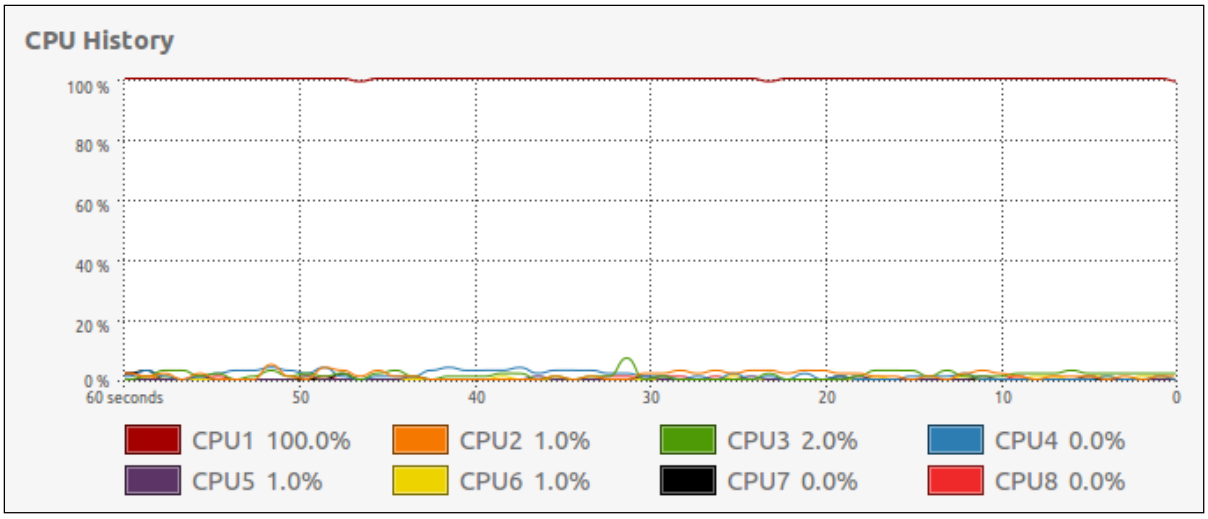

Figure 3.1 Vue des ressources consommées quand notre système IMS est éteint

Grâce à l'outil de visualisation des ressources de Linux, nous pouvons observer sur la figure 3.1 que le système au repos n'utilise que le cœur 1 (CPU1) de la machine pour fonctionner comme nous l'avons contraint à le faire.

Nous faisons également l'hypothèse que les entrées/sorties ne sont pas à prendre en compte, car elles ne sont pas sollicitées par notre système IMS. Les données sur la mémoire de la machine hôte ne seront pas non plus analysées, car cette ressource est abondante dans notre système. En effet, lors d'analyses préliminaires, nous observons que, dans le pire cas simulé, les conteneurs ne consomment que 10% de la mémoire disponible. Nous avons donc donné aux conteneurs un accès illimité à la mémoire de l'ordinateur.

Nous ferons aussi l'hypothèse que nos outils de télémétrie n'empêchent pas le système de bien fonctionner et que les ressources qu'ils consomment eux même pour réaliser les tâches qui leur sont demandées n'ont pas d'impact sur les simulations. Pour satisfaire à cette hypothèse, nous ne pouvons pas utiliser le premier outil de télémétrie pour les ressources du système que nous avons développé. En effet, comme expliqué dans la section 2.3.1, celui-ci utilise l'outil TShark qui est trop lent. Lorsque nous simulons beaucoup d'appels, nous sommes donc confrontés à deux problèmes : avoir des données en temps réel et avoir des

données exhaustives. Pour avoir les données en temps réel, cela nécessite de suivre en temps réel l'échange de messages sur le réseau, mais l'outil n'a pas le temps de tout analyser et dans le cas où le nombre d'appels est important il nous en manque et les résultats sont erronés. Pour avoir des données exhaustives, il faut récupérer tous les messages échangés, mais dans ce cas, le relevé ne se fait plus en temps réel, il peut nous falloir plusieurs minutes pour relever une seule seconde de la simulation. De plus, pour s'exécuter et récupérer tous les messages, cet outil consomme énormément de ressources CPU créant une famine pour les conteneurs.

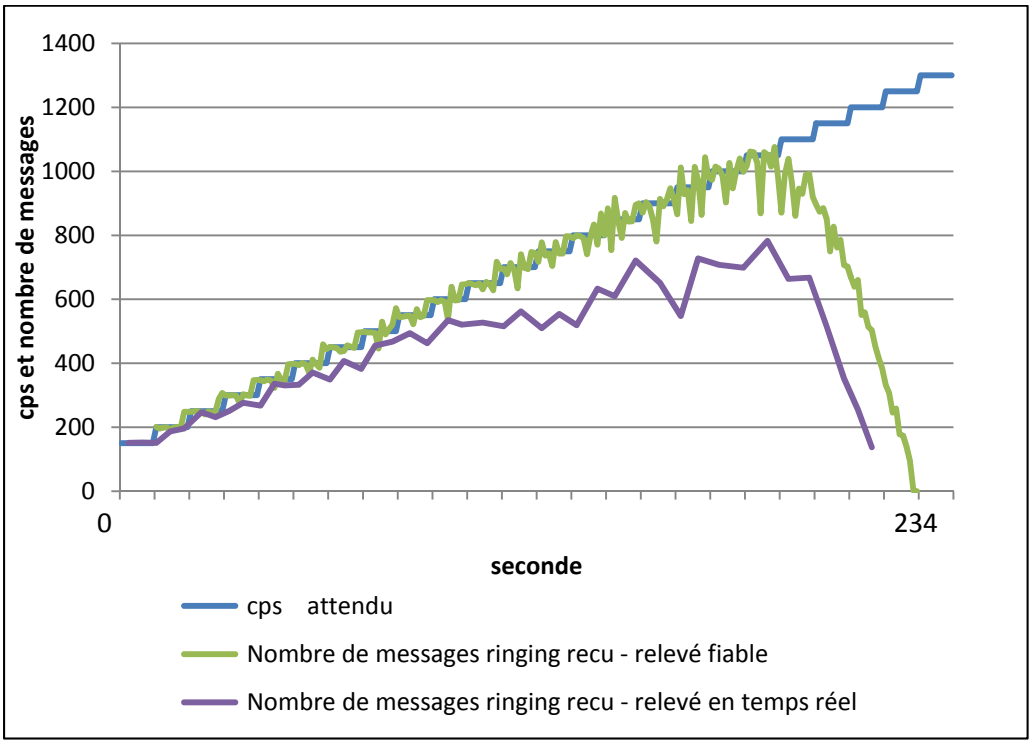

Figure 3.2 Comparaison des relevés du nombre de messages RINGING reçus dans le cas temps réel ou non

La comparaison entre les relevés des messages RINGING reçus en temps réel ou non est illustrée à la figure 3.2. Le relevé dit fiable, qui permet de collecter toutes les valeurs, a été réalisé en presque 8 heures alors que la simulation ne dure que quelques minutes. L'outil de télémétrie haut niveau avec TShark ne sera donc utilisé que dans notre phase d'analyse préliminaire du système IMS virtualisé pour apprendre à mieux l'appréhender. Nous

utiliserons ensuite, comme mentionné à la section 2.3.1, les valeurs du CPS fournies par le simulateur. L'outil de monitorage des ressources bas niveau a été, quant à lui, exploité, car il ne consomme qu'une quantité raisonnable de ressources comme nous pouvons le lire sur la figure 3.3.

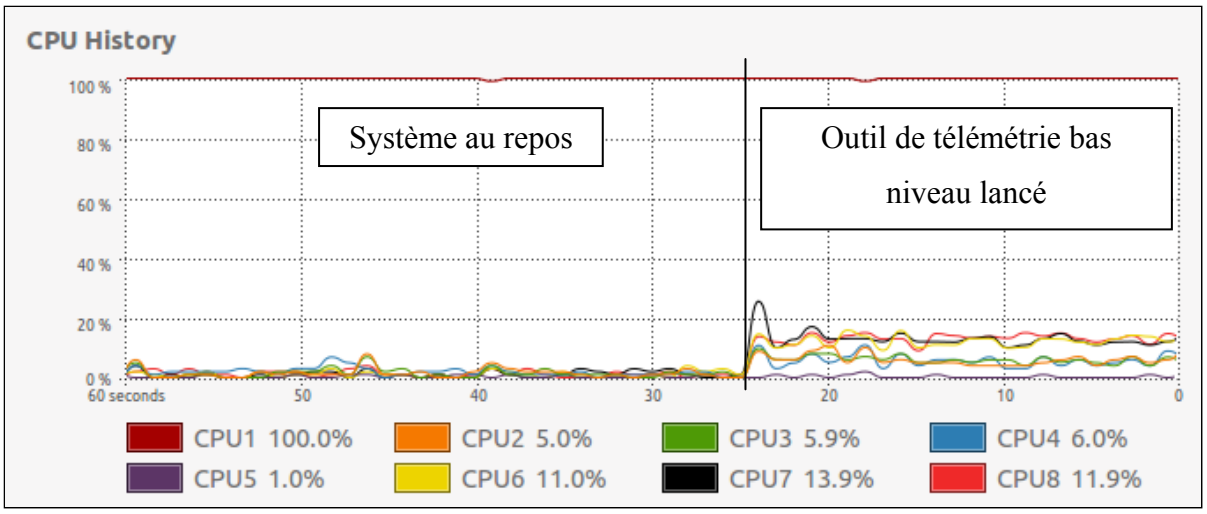

Figure 3.3 Vue des ressources consommées par l'outil de télémétrie bas niveau

Grâce à l'outil de télémétrie de Linux, nous pouvons visualiser les ressources CPU consommées par l'outil de monitorage développé. Cette consommation est, en partie, due au fait qu'une requête est envoyée chaque seconde à chaque conteneur pour leur demander les ressources qu'ils utilisent au niveau du CPU, de la mémoire et de la bande passante. Pour remplacer l'outil de monitorage système, nous avons inclus à celui-ci une fonction qui va lire le fichier log du simulateur de trafic, car ce dernier enregistre aussi le nombre d'appels simulés qui n'a pas abouti.

Pour finir, à la fin de chaque simulation, nous calculons le coefficient de Spearman qui lie les différentes métriques deux à deux. La corrélation de Spearman est utilisée pour étudier la relation entre deux variables qui ne suivent pas une loi normale. Cette méthode ne se base pas sur les valeurs exactes des variables analysées mais sur le rang de ces valeurs. La corrélation de Spearman permet donc de mesurer l'intensité de la liaison entre les deux

variables analysées. Nous faisons l'hypothèse que ce résultat peut être exploité et n'est pas biaisé par l'influence des autres métriques sur celles que nous avons choisies d'étudier.

3.2 Analyse préliminaire du système IMS virtualisé

Une analyse préliminaire a été réalisée afin d'observer la réaction du système IMS virtualisé en fonction de la charge du trafic généré et des ressources allouées aux différents modules. Pour ce faire, nous avons réalisé plusieurs simulations. Lors de chaque simulation, nous avons suivi, au niveau machine et au niveau service, plusieurs métriques. Chaque valeur représentée dans la suite de ce mémoire l'est en fonction du temps écoulé depuis que le test sur le système IMS est lancé. Un test commence après une phase d'enregistrement des utilisateurs et de préchauffage du système.

3.2.1 Choix des différentes métriques relevées

Comme expliqué au chapitre 2, nous avons décidé de suivre les ressources bas et haut niveau consommées par notre système IMS. Pour ce faire, nous avons réalisé plusieurs simulations avec un même jeu de test qui fait croître et diminuer le nombre d'appels par seconde et où le S-CSCF dispose de quatre cœurs. Au plus bas niveau, nous avons relevé :

- ***charge du CPU consommée par chaque conteneur et par les processeurs de l'hôte alloués à ce conteneur :*** Nous avons isolé chaque conteneur pour qu'il ne puisse utiliser qu'un nombre limité de cœurs de l'hôte. Cette limitation nous permet de contrôler et de mieux observer la charge CPU utilisée par chaque conteneur. Sur la figure 3.4 et pour chaque figure de ce mémoire, la courbe nommée « charge du CPU xcscf », comme la courbe bleue ici, représente la charge du CPU consommée par le conteneur xcscf. Pour obtenir cette valeur, nous interrogeons le conteneur grâce aux commandes fournies par LXC, l'outil de virtualisation. Ces commandes sont détaillées à l'annexe III. Ici, nous avons tracé la courbe des valeurs obtenues pour le S-CSCF dans le cas où ce dernier peut utiliser jusqu'à quatre cœurs de l'hôte. La courbe nommée « somme CPU scscf » représente la somme de la consommation en CPU de chaque cœur de l'hôte alloué au conteneur mentionné. Dans ce cas-ci,

c'est donc la somme de la consommation en CPU faite par les quatre cœurs attribués au S-CSCF.

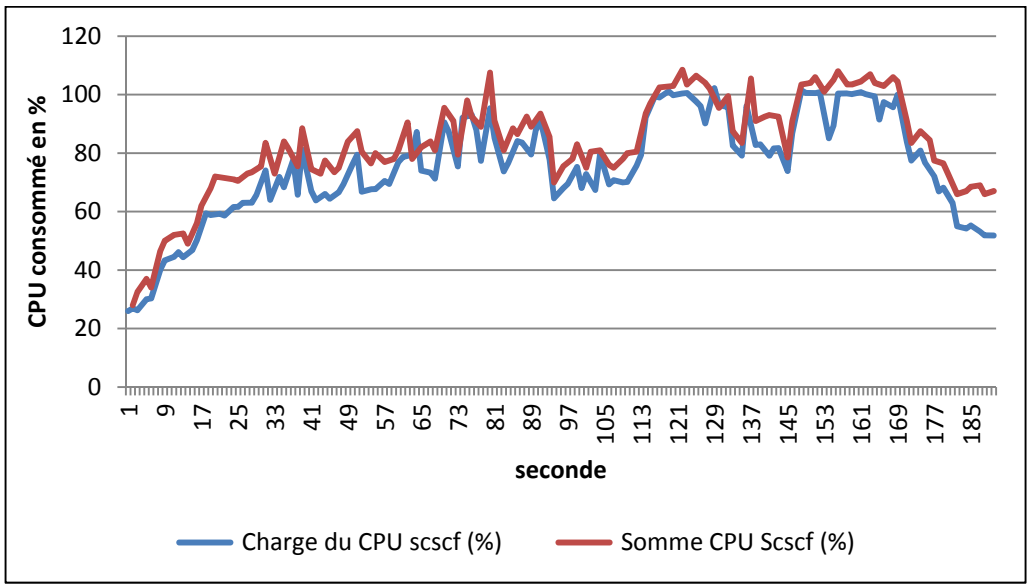

Figure 3.4 Relation entre le CPU utilisé par un conteneur et par les processeurs de l'hôte qui lui sont alloués

Sur la figure 3.4, la courbe bleue illustre donc la consommation du CPU selon les valeurs fournies par le conteneur scscf. La courbe rouge représente la somme de la consommation du CPU de chaque cœur alloué au conteneur scscf. Nous pouvons alors constater que les deux relevés sont semblables. Les cœurs alloués au conteneur scscf lui sont donc totalement consacrés comme nous le souhaitions et les relevés effectués auprès du conteneur sont fiables.

- ***mémoire utilisée par chaque conteneur :*** Comme pour la charge du CPU, nous relevons la consommation de la mémoire de chaque conteneur. Les commandes utilisées pour obtenir les résultats sont détaillées à l'annexe III. Sur la figure 3.5, comme sur toutes les figures de ce mémoire, les courbes nommées « Memoire » et suivies du nom d'un conteneur (voir figure 3.5) représentent les courbes des valeurs relevées pour le conteneur concerné.

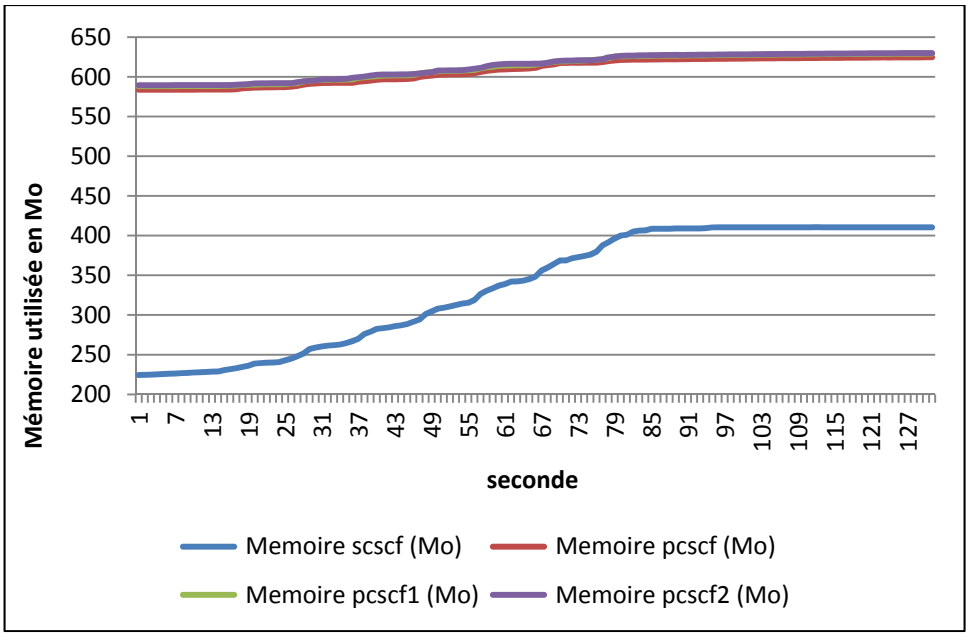

Figure 3.5 Courbes de la mémoire utilisée par les conteneurs

Nous pouvons constater sur la figure 3.5 que tous les conteneurs de type P-CSCF utilisent la mémoire de la même manière. On remarque également que l'utilisation de la mémoire par le conteneur de type S-CSCF est moins importante que celle par les P-CSCF.

- ***bande passante utilisée par chaque conteneur et par l'hôte :*** Comme pour les deux précédentes métriques, nous interrogeons chaque conteneur ainsi que l'hôte afin de collecter les valeurs de l'utilisation de la bande passante pour chacun. Les commandes permettant d'obtenir ces valeurs sont détaillées à l'annexe III. Sur la figure 3.6 ou dans celles de ce mémoire, la courbe représentant les valeurs de la bande passante utilisée par un conteneur est appelée « bande passante X » et est suivie du nom du conteneur. C'est le cas des courbes en bleu et en rouge sur la figure 3.6. Les courbes représentants la bande passante utilisée par l'hôte sont nommées « bande passante X hote », il s'agit des courbes verte et violette illustrées à la figure 3.6. Dans les deux systèmes de notation, X peut être Rx dans le cas où l'élément étudié reçoit les messages et Tx dans le cas où il les émet.

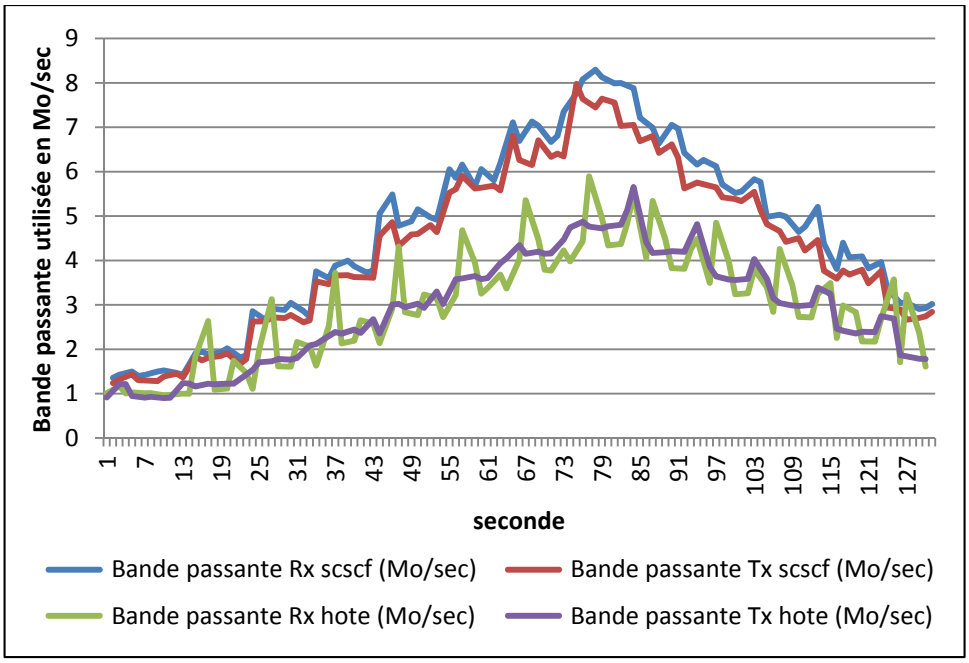

Figure 3.6 Relation entre la bande passante utilisée
par des conteneurs et par l'hôte

Pour les métriques de haut niveau, nous avions décidé de relever les valeurs suivantes :

- ***délai d'établissement d'un appel :*** Pour calculer le délai d'établissement d'un appel, nous calculons le temps écoulé entre le moment où un utilisateur envoie un message INVITE sur le système IMS pour demander l'ouverture d'une session et le moment où il reçoit le message RINGING qui signifie que la personne qu'il a contactée lui a répondu et que la session est ouverte.
- ***temps de réponse des conteneurs :*** Pour calculer le temps de réponse d'un module CSCF, nous calculons le temps qui s'est écoulé entre le moment où il reçoit un message INVITE et le moment où il envoie un message RINGING.
- ***nombre de messages RINGING reçus :*** Lorsqu'un utilisateur veut établir une session avec un autre utilisateur, il envoie un message INVITE sur le système IMS. Quand l'utilisateur contacté décroche, il envoie un message RINGING au système IMS. En comptabilisant le nombre de messages RINGING, nous pouvons connaître le nombre de sessions qui ont été ouvertes. Grâce à ces informations, l'outil de

monitorage peut alors mesurer le nombre de sessions établies et le nombre de sessions échouées.

- ***nombre de messages BYE reçus :*** L'envoi d'un message BYE du client au système IMS signifie qu'il souhaite terminer une session en cours. Si l'outil de monitorage connaît le nombre de sessions qui ont été ouvertes et le nombre de messages BYE qui ont été reçus, il peut calculer le nombre de sessions qui ont été interrompues par le système et non terminées selon la volonté d'un utilisateur.

L'enchaînement des messages INVITE, RINGING et BYE mentionnés précédemment est décrit sur la figure 3.7. Ce que les auteurs de la figure ont nommé *Serveur* est dans notre cas un système IMS virtualisé.

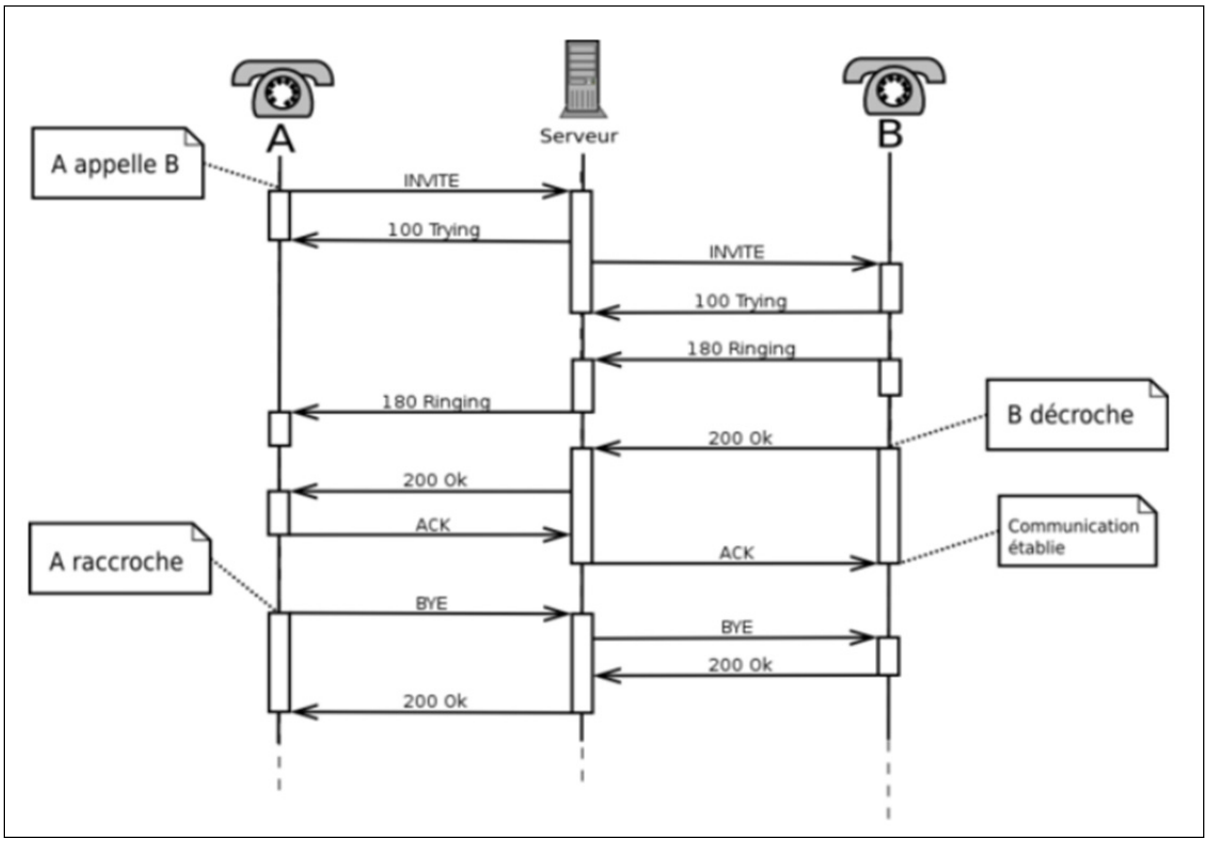

Figure 3.7 Le protocole SIP
Tirée de (Gourong 2010)

Pour l'obtention des valeurs de haut niveau dont nous avons besoin pour l'analyse préliminaire de notre système, nous utilisons TShark. Cet outil est très performant, mais comme déjà mentionné précédemment nous ne pourrons pas l'utiliser lors du déploiement de l'outil MAA que nous construisons. Pour l'utilisation de MAA, nous avons donc choisi de nous contenter des valeurs relevées par le simulateur de trafic, à savoir le nombre de communications initiées et l'IHS. IHS signifie *inadequately handled scenarios*. Il représente le pourcentage de sessions qui ont été interrompues de manière non volontaire depuis le début de la simulation. Le simulateur de trafic s'arrête automatiquement lorsque cette valeur dépasse les 70%. À partir de cette information, nous pouvons aussi obtenir le nombre de sessions interrompues involontairement chaque seconde.

3.2.2 Analyse de l'utilisation de la mémoire

Pour les conteneurs de type I-CSCF, quel que soit le jeu de test utilisé lors des simulations, l'utilisation de la mémoire par un conteneur de type I-CSCF est presque constante dans le temps (voir figure 3.8). Cette observation vient du fait que le conteneur de type I-CSCF n'est pas utilisé dans notre cas car il sert à connecter plusieurs systèmes IMS et nous n'en utilisons qu'un.

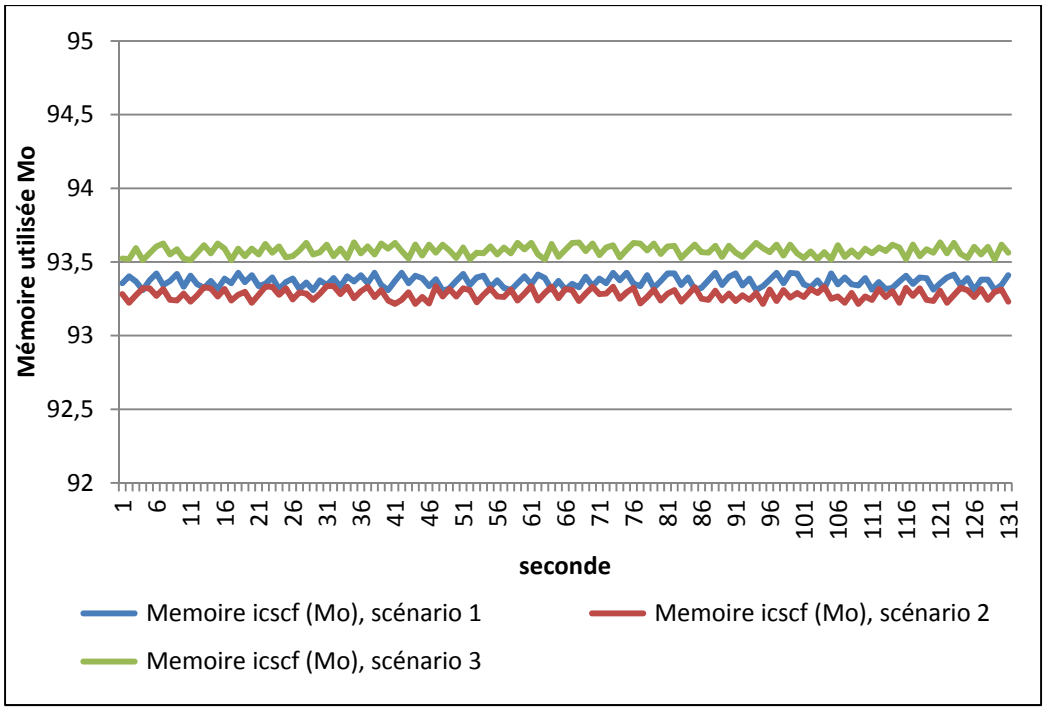

Figure 3.8 Exemple de consommation de la mémoire par l'I-CSCF
pour les différents jeux de test

Pour les conteneurs de type P-CSCF, il n'y a pas de relation visible entre le nombre d'appels par seconde et la mémoire consommée par le P-CSCF. Cependant, dans le cas où le P-CSCF cesse de fonctionner alors que le système IMS est toujours opérationnel, nous remarquons une chute brutale de la consommation de la mémoire par le conteneur hébergeant ce P-CSCF (voir figure 3.9). Cette chute de consommation nous permettra de détecter qu'un P-CSCF s'est arrêté et de prédire que le système IMS va s'arrêter à son tour parce que le manque d'un P-CSCF le met en souffrance (trafic reçu supérieur à la capacité du système IMS). Nous ne savons pas pourquoi un P-CSCF s'arrête. Nous pensons que cela est dû à un problème dans son implémentation. En effet, il s'arrête à chaque fois que la différence d'utilisation de la mémoire par le cache entre l'instant où le conteneur est lancé et celui où il s'arrête, atteint 250 Mo. Il n'est pas possible dans notre cas de relancer un P-CSCF pour le réintégrer dans le système IMS sans redémarrer ce dernier.

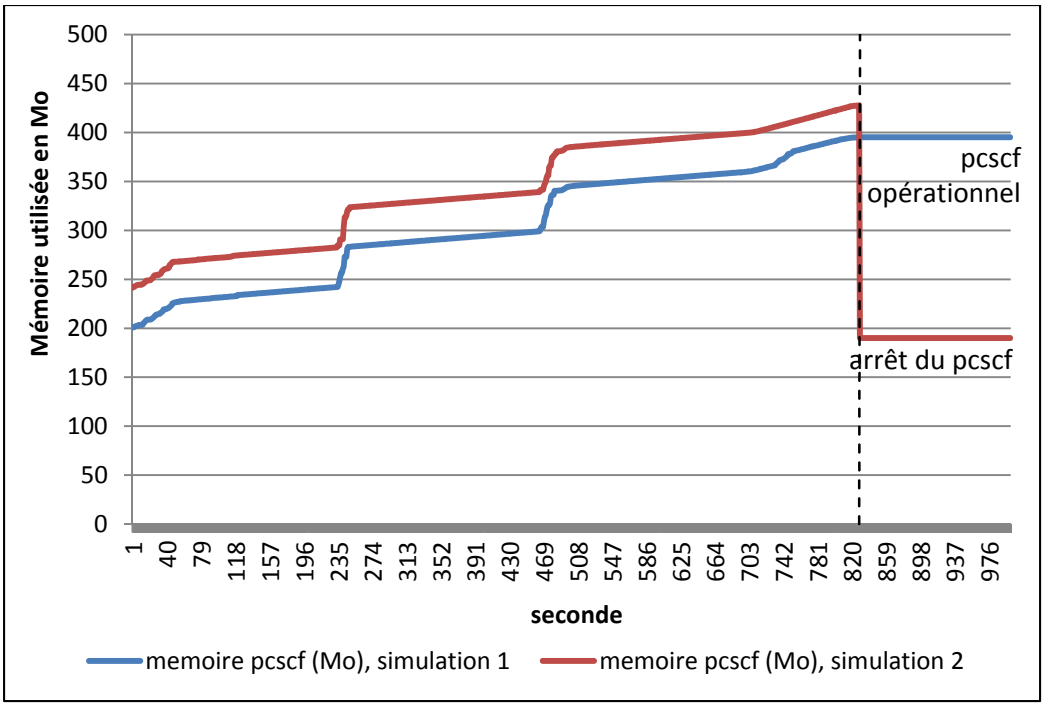

Figure 3.9 Consommation de la mémoire par le P-CSCF
dans deux cas différents

Les données représentées sur la figure 3.9 ont été collectées pour le même pcscf lors de simulations avec le même jeu de test où le profil de trafic est croissant. Lors de la première simulation, le conteneur pcscf est lancé juste avant le début de la simulation. Après 1000 secondes, la simulation est finie et le pcscf est toujours opérationnel. Lorsqu'une simulation est terminée, toutes les sessions ouvertes pendant celle-ci sont clôturées et la mémoire utilisée par le pcscf redescend, mais la valeur atteinte est plus élevée que lors du lancement du conteneur. Sans redémarrer ce dernier, la même simulation est rejouée. Cette fois-ci, le conteneur pcscf ne peut pas gérer l'augmentation de la mémoire et il cesse de fonctionner. Lorsqu'il s'arrête, la mémoire qu'il consomme redevient égale à celle qu'il consommait à son lancement. On peut alors remarquer que la différence entre la mémoire consommée au démarrage du conteneur et au moment où il a cessé de fonctionner est de 250Mo.

Pour les conteneurs de type S-CSCF, il n'y a pas de relation exploitable entre le nombre d'appels par seconde et la mémoire consommée par le S-CSCF. En effet, comme le montrent

les figures 3.10, 3.11 et 3.12, lorsque le nombre d'appels par seconde croît ou est constant l'utilisation de la mémoire augmente. Elle stagne si le nombre d'appels par seconde diminue ou si le système arrête de fonctionner.

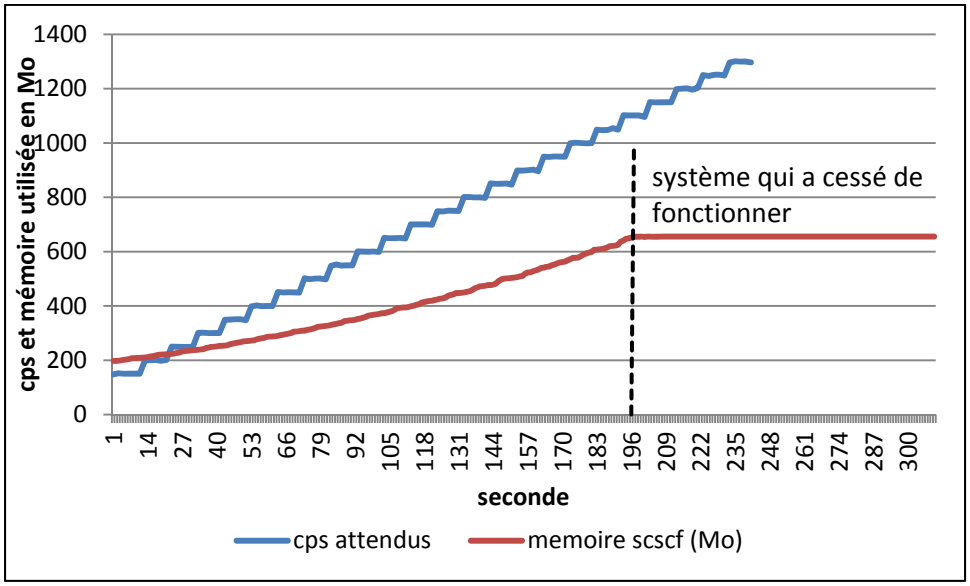

Figure 3.10 Exemple de la consommation de la mémoire par le S-CSCF lorsque le CPS croît

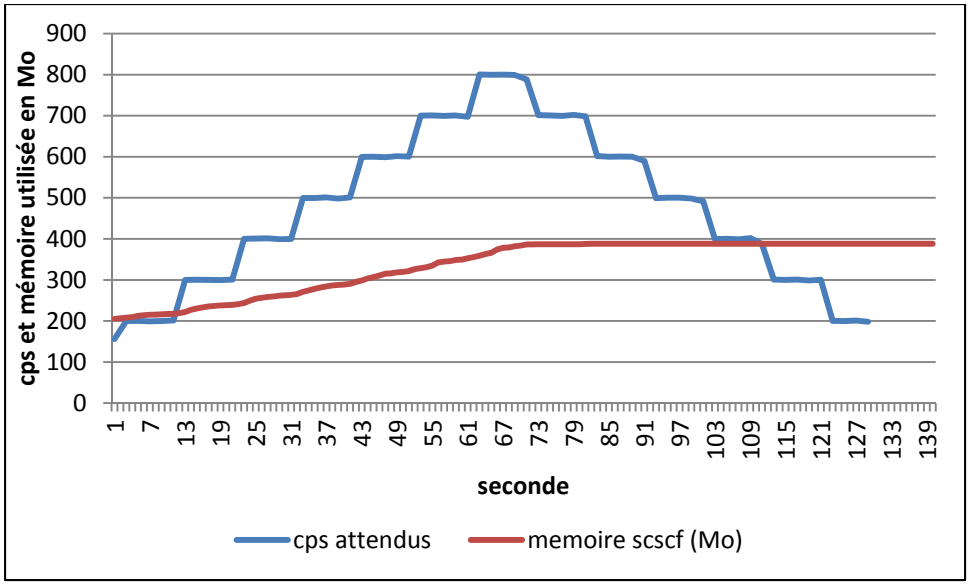

Figure 3.11 Exemple de la consommation de la mémoire par le S-CSCF lorsque le CPS croît puis diminue

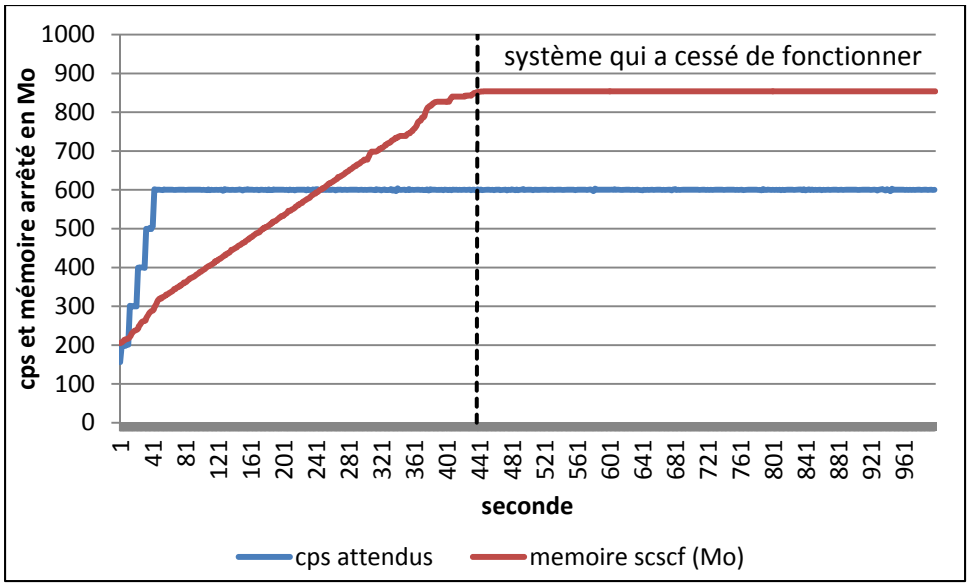

Figure 3.12 Exemple de la consommation de la mémoire
par le S-CSCF lorsque le CPS est constant

3.2.3 Analyse de l'utilisation de la bande passante

Pour l'hôte et pour chaque conteneur, nous avons enregistré l'utilisation de la bande passante. Comme le montre la figure 3.13, qui est un exemple réalisé lors d'une simulation où le CPS croît, les courbes suivent la même tendance et sont très proches.

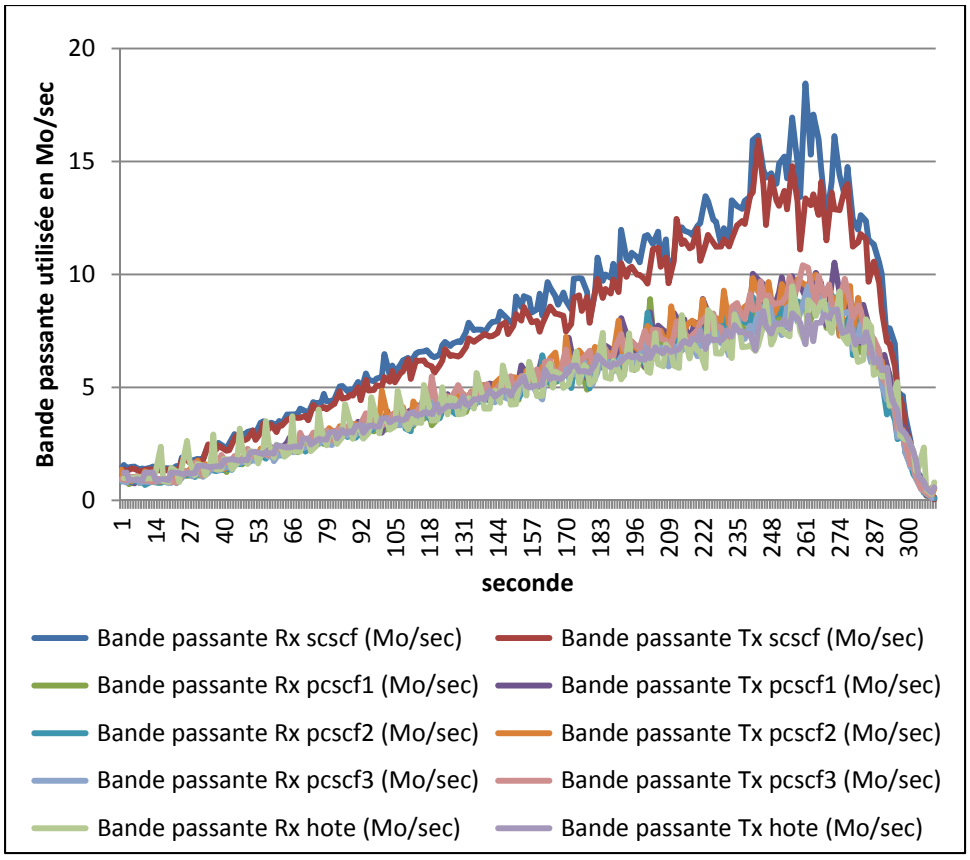

Figure 3.13 Exemple de l'utilisation de la bande passante
par les conteneurs lorsque le CPS croît

Pour l'hôte, comme pour les conteneurs de type P-CSCF ou S-CSCF, nous enregistrons aussi une forte relation entre les valeurs récoltées pour la bande passante et celles du nombre d'appels émis par seconde. En effet, si nous traçons les courbes pour ces deux métriques, nous pouvons voir qu'elles suivent la même tendance quand le système IMS n'est pas surchargé. Quand le système atteint un certain seuil, des messages se perdent et doivent être réémis. Cette réémission se voit sur les courbes de la bande passante par une oscillation très prononcée de cette dernière juste avant la chute (environ 250ème seconde). Cette affirmation se confirme lorsque nous traçons la courbe des données de service liées aux messages reçus par le système IMS. Ces données ont été récoltées avec l'outil TShark. Nous obtenons alors les courbes illustrées à la figure 3.14 qui mettent en relation la bande passante et le nombre de messages RINGING reçus par le système IMS.

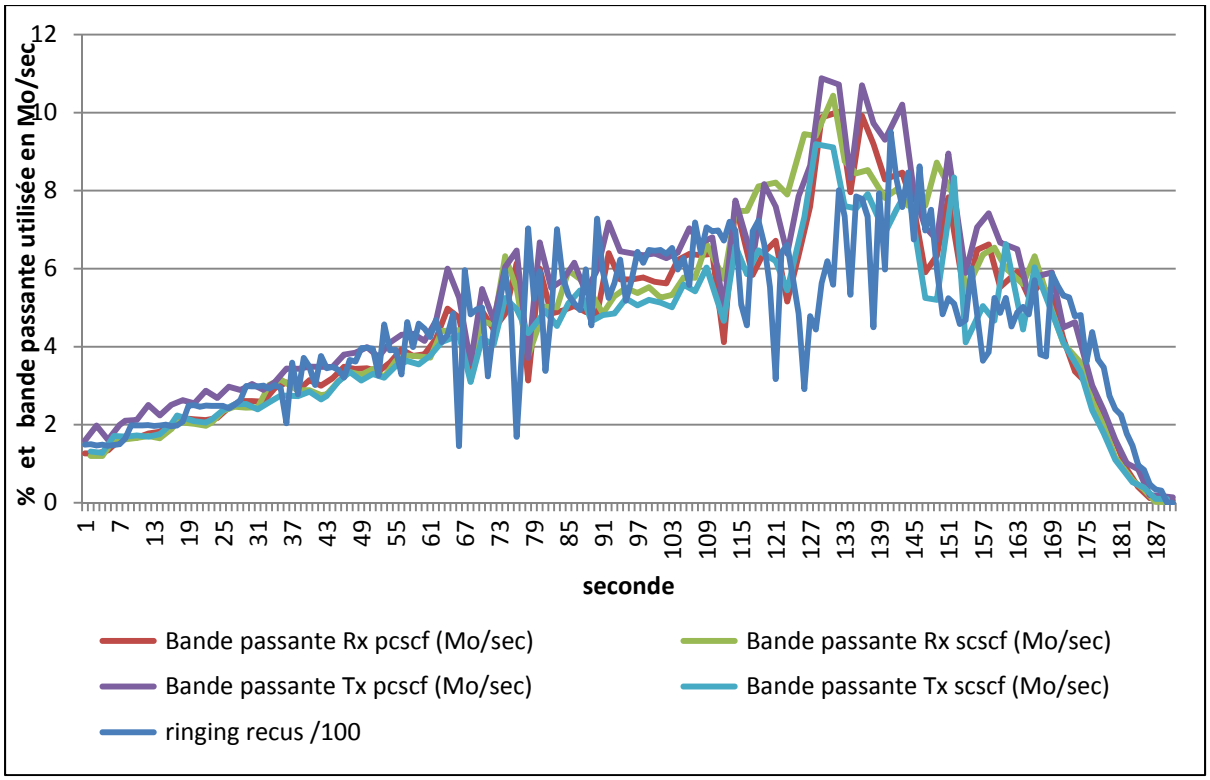

Figure 3.14 Courbes du nombre de messages RINGING reçus par le système IMS et de la bande passante consommée par chaque conteneur lorsque le CPS croît

Cette constatation nous laisse penser qu'il pourrait être intéressant de déterminer l'intervalle de temps durant lequel de fortes oscillations de la bande passante se produisent et de corréler cette fluctuation avec le taux de messages perdus. De cette manière, nous pourrions détecter de fortes oscillations inattendues de la bande passante afin de détecter des messages perdus et ainsi prédire que le système va s'arrêter car il souffre.

3.2.4 Analyse de la charge du CPU

Pour les conteneurs de type I-CSCF, quelle que soit la simulation testée, la charge du CPU de ce conteneur et celle du cœur associé restent constantes. Cela s'explique par le fait que ce conteneur n'est pas sollicité dans notre réseau IMS. Pour les conteneurs de type P-CSCF, nous constatons que, comme la bande passante, la courbe de la consommation du CPU suit le même modèle que celle du nombre d'appels émis par seconde. Pour le S-CSCF, la charge du CPU augmente lorsque le cps croît ou est constant et elle diminue lorsque le cps décroît.

Le CPU du conteneur S-CSCF et des cœurs qui lui sont alloués peut atteindre les 100% avant que le système ne s'arrête. Même si la charge maximale du système est atteinte, il peut encore être sauvé si davantage de ressources sont fournies au conteneur. À partir de la figure 3.15, nous pouvons observer qu'en ajoutant un cœur au S-CSCF, quand les siens arrivent à saturation, la charge du CPU du conteneur et la valeur du IHS décroissent. Le fait que cette dernière valeur chute nous prouve que le système fonctionne de nouveau comme voulu, car il ne perd plus de messages.

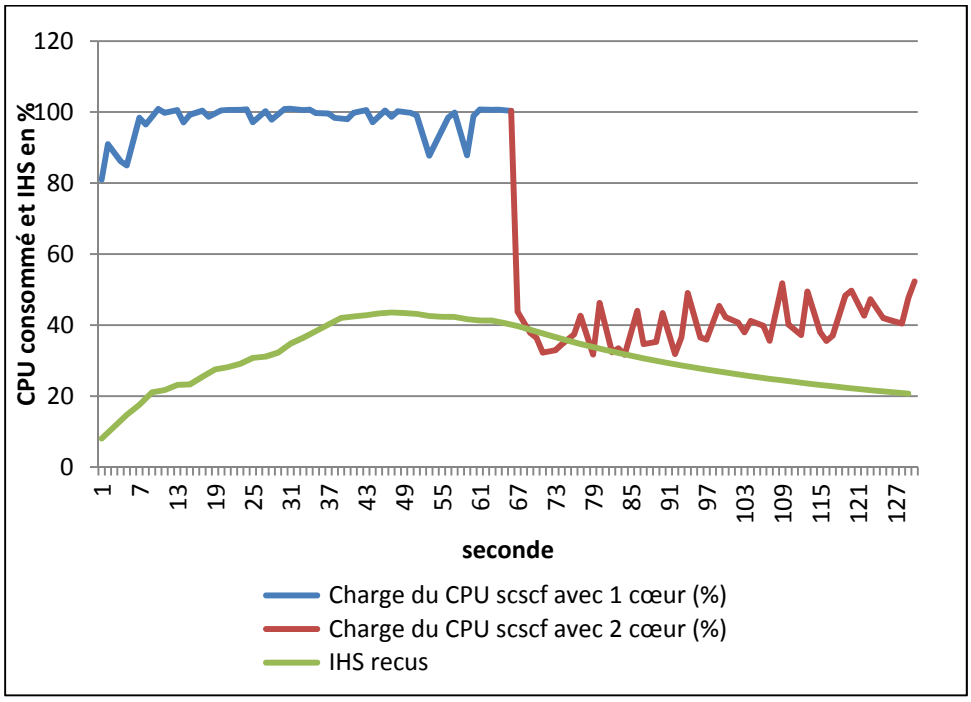

Figure 3.15 Charge du CPU pour une simulation avec un CPS constant et une augmentation du nombre de cœurs alloués au S-CSCF

Selon la configuration de notre système IMS, un nouveau cœur d'un CPU met environ 10 secondes à entrer en fonction. Cette valeur est essentielle pour la suite de notre étude, car notre algorithme devra prendre en compte le temps de déploiement du nouveau cœur pour l'allouer au S-CSCF au meilleur moment. En effet, le but ici est d'allouer un nouveau cœur au système juste avant que celui-ci ne vienne à manquer afin d'éviter le gaspillage. Il faut cependant ne pas l'allouer trop tard car cela pourrait entraîner une violation du SLA et des pénalités pour le fournisseur d'infrastructure. La valeur de cet intervalle de temps est illustrée

à la figure 3.16. C'est la différence entre le moment où nous avons ajouté un nouveau cœur à la liste de ceux que le conteneur utilise et le moment à partir duquel nous pouvons observer que la charge CPU du nouveau cœur augmente et celles des cœurs déjà en fonction diminuent. Au début de la simulation permettant de réaliser la figure 3.16, le conteneur S-CSCF ne pouvait utiliser que le cœur 6 de l'hôte. Vers la 113ème seconde, le cœur numéro 7 de l'hôte a été alloué au conteneur S-CSCF. Nous voyons alors que sa charge de travail augmente aux alentours de la 123ème seconde et que celle du cœur 6 de l'hôte diminue. Les deux cœurs sont alors utilisés par le conteneur S-CSCF.

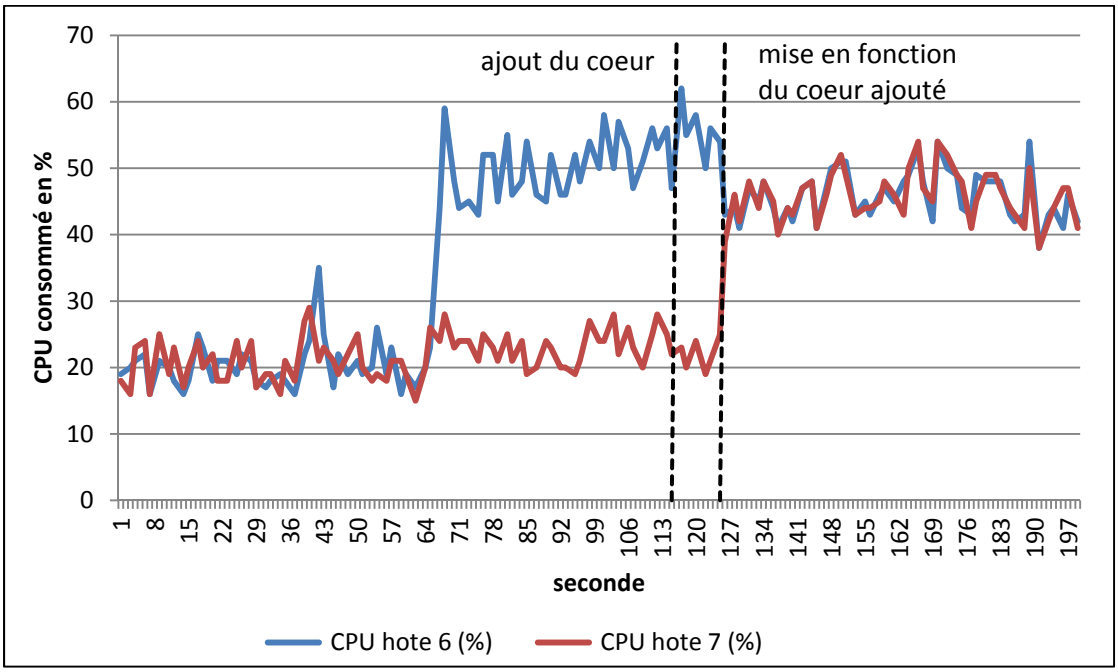

Figure 3.16 Temps de réaction lors de l'ajout d'un cœur
au conteneur S-CSCF

De plus, le système IMS entre dans un état d'instabilité (écart brusque de la consommation du CPU) avant même que la charge CPU du S-CSCF ait atteint les 100%. Entre 70 et 80% de la consommation du CPU, nous constatons un palier à partir duquel la charge du CPU monte plus brusquement alors que le nombre de cps augmente toujours avec le même taux. La figure 3.17 montre ce seuil pour différentes configurations du S-CSCF (1 cœur, 2 cœurs ou 3 cœurs de CPU). Sur cette figure, nous constatons que plus le nombre de cœurs alloués au S-

CSCF est grand plus tard ce seuil est atteint. La charge du CPU finit par atteindre son maximum (100%) et dans les cas où nous ne faisons rien le système s'arrête faute de ressources. Il faut donc prévoir d'augmenter le nombre de cœurs du S-CSCF dans le cas où nous franchissons ce seuil de 70%.

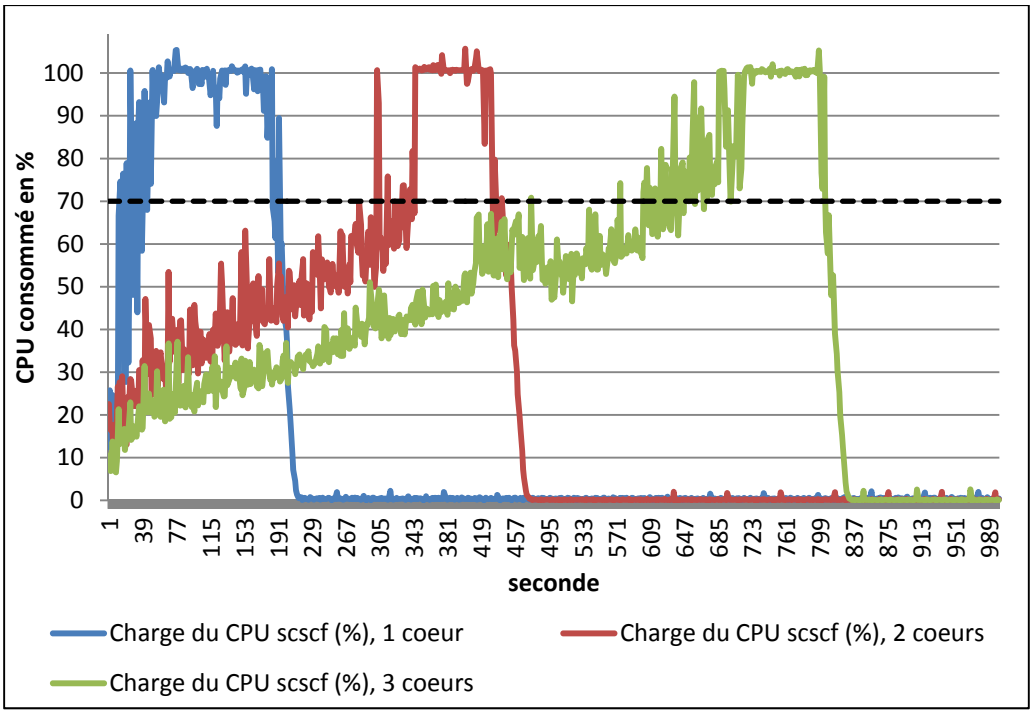

Figure 3.17 Exemple de la charge du CPU en fonction du nombre de cœurs disponible au S-CSCF pour un CPS qui croît de manière constante

3.2.5 Analyse du taux de sessions interrompues

L'IHS, *inadequately handled scenarios*, est une valeur calculée par le système de simulation de trafic. Cette valeur représente le pourcentage de messages perdus pendant une étape de la simulation. Nous ne pouvons donc pas exploiter cette valeur en l'état. En effet, dans le cas d'une étape de la simulation où nous réussissons à établir un grand nombre de sessions avant que le système ne commence à souffrir, l'IHS varie plus lentement que dans le cas d'une simulation où le trafic généré amène rapidement le système à sa fin. Ces situations sont illustrées par les figures 3.18 et 3.19. À partir de celles-ci, nous pouvons observer que dans

un cas, détecter que l'IHS franchi le seuil de 5% nous permettrait de dire que le système n'a plus suffisamment de ressources CPU, et que dans l'autre cas, le seuil est franchi une fois que ce manque de ressources a été fatal au système (le système a cessé de fonctionner).

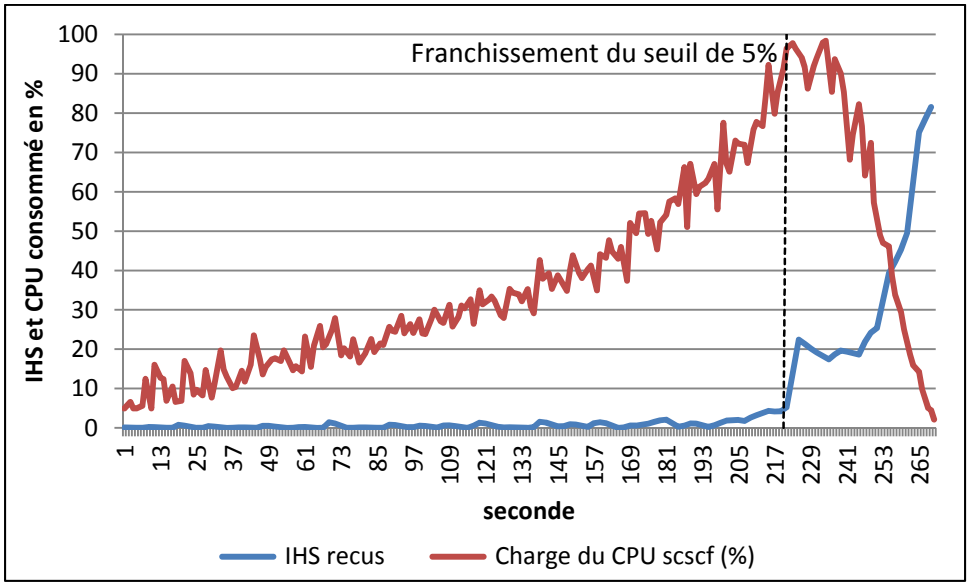

Figure 3.18 Exemple où 5% de l'IHS indique que le système est à son maximum

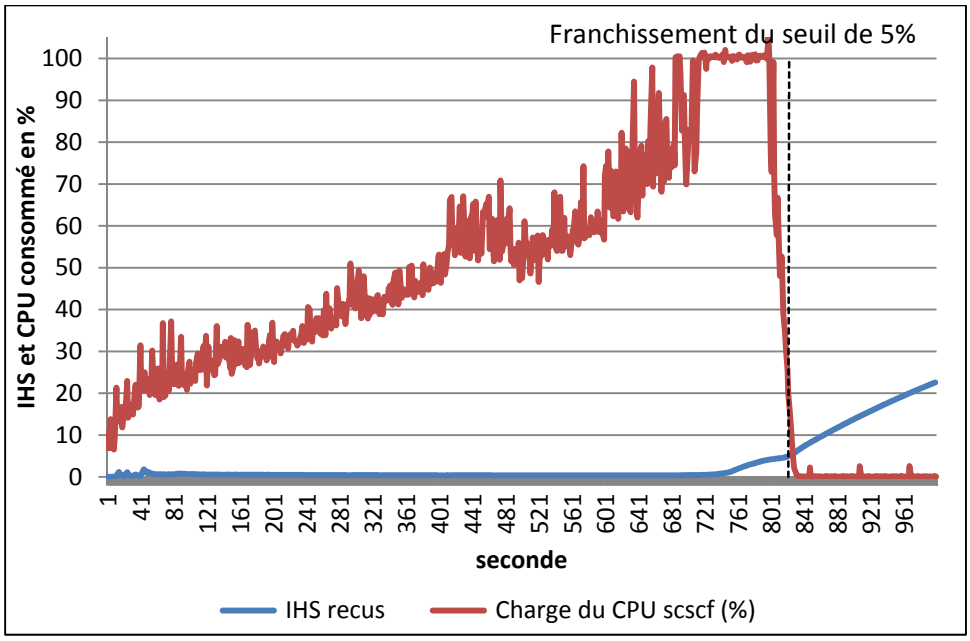

Figure 3.19 Exemple où 5% de l'IHS indique trop tard que le système souffre

Malgré le fait que le système de simulation soit construit pour s'arrêter suivant la valeur d'IHS, nous allons récupérer le pourcentage de sessions interrompues chaque seconde pour détecter une souffrance du système et ainsi prédire un arrêt prématuré du système. Le pourcentage de sessions interrompues ne dépend pas du nombre de sessions déjà établies, il reflète donc à chaque seconde le taux de sessions que le système ne peut pas gérer. Nous obtenons les figures 3.20 et 3.21 pour les mêmes simulations que précédemment.

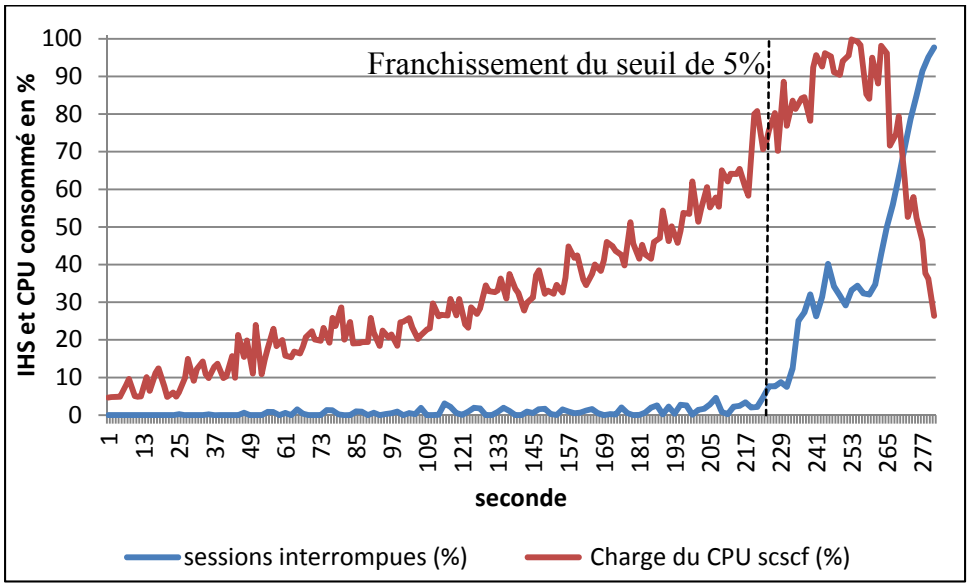

Figure 3.20 Mise en relation du pourcentage de sessions interrompues et de la charge du CPU du scscf pour la même simulation que pour 3.18

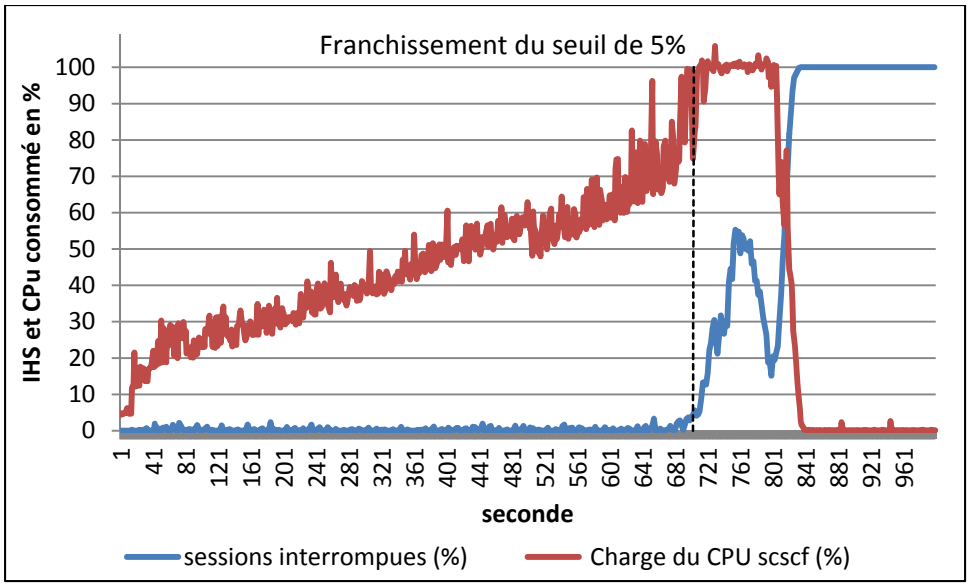

Figure 3.21 Mise en relation du pourcentage de sessions interrompues et de la charge du CPU du scscf pour la même simulation que pour 3.19

À partir de ces figures, nous remarquons que si nous détectons un dépassement du seuil de 5% de sessions interrompues alors nous pouvons prédire que le système va s'arrêter. Dans les deux cas, le seuil est franchi avant que le système cesse de fonctionner. Cela nous permettra donc de pouvoir augmenter le nombre de ressources attribuées au système IMS avant qu'il ne s'arrête.

3.3 Algorithmes de prédiction de la charge du CPU

Dans cette partie, nous présenterons les algorithmes développés pour permettre au système d'adapter ses ressources CPU de manière autonome. Ces algorithmes seront inspirés par les analyses faites dans la section 3.2.

3.3.1 Règles tirées de l'analyse préliminaire

Dans l'analyse préliminaire, présentée à la section 3.2, nous avons remarqué certains signes qui pourraient nous permettre de prédire une fin prématurée du système. Nos principales observations sont résumées dans ce qui suit :

- une chute brutale de la mémoire d'un conteneur de type P-CSCF permet d'anticiper que ce dernier s'est arrêté. Il est possible de vérifier cela en comptant le nombre de P-CSCF détecté par la machine hôte. Dans le cas où il y en a moins qu'au début de la simulation, nous savons que celle-ci va s'arrêter, car de nombreuses sessions ne pourront plus être établies. Il nous sera alors possible d'annoncer la fin du système.
- de fortes oscillations de la bande passante des conteneurs de type P-CSCF et S-CSCF permettent de détecter que le système a dû renvoyer des messages suite à une perte de ces derniers. Connaissant le nombre d'appels par seconde émis à chaque instant, nous pouvons prévoir une certaine tendance des valeurs de la bande passante des conteneurs. Dans le cas de fortes oscillations non prévues, nous pouvons donc détecter une souffrance du système et ainsi prédire que si rien ne change le système cessera de fonctionner.
- de fortes oscillations de la charge du CPU des conteneurs de type S-CSCF peuvent laisser penser que le système souffre. Cependant, cet évènement n'est pas fiable, car nous ne l'observons pas sur toutes les simulations ou alors pas avec la même intensité.
- un dépassement du seuil de 70% de la charge du CPU du conteneur de type S-CSCF permet de détecter une surcharge de ce dernier. Nous pouvons donc affirmer, dans ce cas-là, que si le nombre d'appels par seconde continue de croître le système va manquer de ressources et s'arrêter, car le conteneur ne pourra plus supporter la charge de trafic reçue. Le dépassement de ce seuil ou les fortes oscillations de la bande passante nous permettent donc de détecter le même problème. Nous nous concentrerons alors sur le seuil du CPU, car ce dernier nous est plus facile à détecter de manière fiable.
- un dépassement du seuil de 5% de sessions interrompues par le système nous permet de détecter une souffrance de ce dernier. En effet, si le système ferme des sessions prématurément c'est qu'il y a un problème et qu'il, ne pouvant plus gérer le nombre de sessions demandées, va arrêter de fonctionner. Le problème peut provenir du conteneur de type S-CSCF et dans ce cas nous aurons aussi détecté le dépassement du seuil de 70% de sa charge. Le problème peut également provenir de la chute d'un

conteneur de type P-CSCF et dans ce cas nous aurons détecté une chute brutale de la mémoire d'un de ces derniers. Le problème peut aussi venir d'une autre cause qui nous sera inconnue, nous ne pourrons alors pas l'expliquer mais nous pourrons également le détecter grâce au suivi du nombre de sessions interrompues.

Ainsi, nous proposons de détecter un de ces signes pour pouvoir prédire que le système souffre et va avoir besoin d'un cœur de CPU supplémentaire pour continuer de fonctionner. Pour cela, l'outil de monitorage développé suivra les ressources suivantes : le CPU du conteneur de type S-CSCF et la mémoire des conteneurs de type P-CSCF. Bien que l'outil MAA développé puisse suivre simultanément toutes les métriques présentées à la section 3.2, le suivi de ces métriques est suffisant dans notre cas car nous nous concentrons sur la gestion dynamique et proactive du CPU du conteneur S-CSCF.

Au fil du développement de notre outil de monitorage, pour l'adaptation dynamique des ressources CPU du conteneur de type S-CSCF, nous avons implémenté différents algorithmes. Ces derniers nous ont permis de mieux comprendre et appréhender le système IMS et ses réactions face à l'ajout d'un nouveau cœur ou à une surcharge. Les approches de gestion de ressource CPU du S-CSCF que nous proposons sont décrites dans ce qui suit.

3.3.2 Algorithme de décision par seuil non prédictif

Pour le premier algorithme que nous avons développé, nous ne savions pas comment prédire les valeurs des métriques monitorées. Nous avons alors décidé de nous limiter à l'analyse des données du CPU du S-CSCF et de la mémoire des P-CSCF. Nous avons ainsi développé les règles suivantes pour la gestion dynamique du nombre de cœurs alloués au S-CSCF :

- Si chute brutale de la mémoire d'un conteneur de type P-CSCF
  Alors alerter que la fin du système IMS est proche
- S'il y a un dépassement du seuil maximal de la charge du CPU pour le S-CSCF et un dépassement de 5% des sessions interrompues lorsque le nombre de cps est croissant

Alors,

Allouer un nouveau cœur CPU si possible

Sinon alerter de la fin proche du système

- S'il y a une chute du CPU du S-CSCF sous le seuil minimal et que moins de 5% de sessions sont interrompues lorsque le nombre de cps est décroissant

  Alors,

  Enlever un cœur CPU si possible

  Sinon ne rien faire

- Si plus de 10% de sessions sont interrompues

  Alors,

  Allouer un nouveau cœur CPU si possible

  Sinon alerter de la fin proche du système

L'algorithme a été testé avec différentes simulations. Bien que les conclusions tirées soient issues de tous les tests réalisés, dans la suite de cette section, seuls les résultats obtenus avec une simulation qui augmente le nombre d'appels par seconde de 150 à 1200 par ajout de 50 cps toutes les 10 secondes puis le diminue de 50 toutes les 10 secondes pour revenir à 150 cps, seront présentés. Ce choix a été fait car cette simulation regroupe différents profils de charge de trafic (croissant et décroissant, nécessitant l'ajout et le retrait de cœurs CPU).

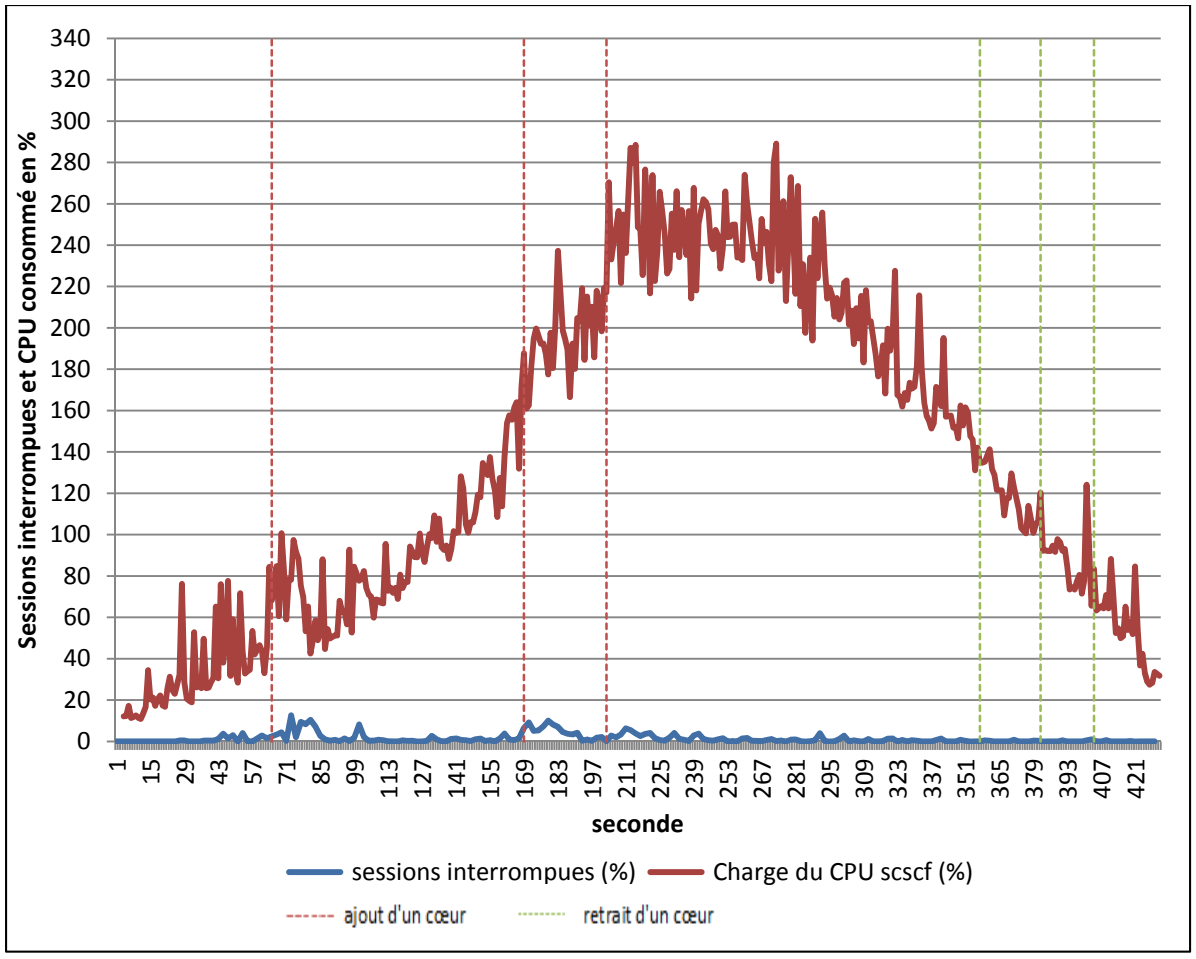

Figure 3.22 Pourcentage de la charge du CPU et du nombre de sessions interrompues lors d'une simulation où la gestion du nombre de cœurs du S-CSCF est automatique et basée sur des seuils

Tous les relevés des métriques commencent après les étapes d'enregistrement des utilisateurs et de préchauffage du système IMS. Sur la figure 3.22, nous pouvons observer que l'outil développé gère automatiquement les cœurs alloués au conteneur de type S-CSCF comme nous l'avons défini précédemment. Il permet donc au système de fonctionner en utilisant les ressources nécessaires et en limitant les excès. Dans note cas, le seuil maximal utilisé est de 70% (seuil pour l'ajout d'un cœur CPU), comme déterminé à la sous-section 3.2.4, et le seuil minimal est de 40% (seuil pour le retrait d'un cœur CPU). Ce seuil minimal a été défini de manière arbitraire afin de conserver des tests sans augmentation soudaine du nombre de

sessions interrompues et en essayant de minimiser le gaspillage de ressources. Avec ces seuils, nous obtenons l'utilisation moyenne de la charge du CPU résumée dans le tableau 3.1.

Tableau 3.1 Utilisation du CPU par le S-CSCF en fonction du nombre de cœurs qui lui est alloué pour l'algorithme par seuil non prédictif

Nombre de cœurs alloué au S-CSCF	1	2	3	4	3	2	1
Utilisation moyenne du CPU (%)	50	60	63	50	40	45	50

Nous aimerions améliorer l'utilisation moyenne de cette charge par le conteneur de type S-CSCF. En effet, nous souhaiterions une utilisation moyenne tout le temps supérieure à 50% et la plus haute possible (voir tableau 3.1). Une manière d'avoir ce résultat est de changer le seuil minimal. Cela est envisageable, car nous avons constaté que le seuil minimal pouvait être augmenté sans que le système en souffre. Lorsqu'un cœur va être retiré, la charge du conteneur de type S-CSCF sera répartie sur les cœurs restants. Il faut donc que la répartition sur les cœurs restants ne dépasse pas le seuil maximal. Le seuil minimal sera donc désormais calculé grâce à la formule suivante :

$$seuil_{min} = seuil_{max} * (nbCoeurSCSCF - 1) \quad (3.1)$$

Nous aimerions également augmenter le seuil maximal, mais cela n'est pas possible. Quand nous avons essayé, le nombre de sessions interrompues faisait des pics et nous avons noté des fluctuations de la charge CPU du système IMS. Cela est en partie dû au fait qu'il faille 10 secondes au conteneur pour commencer à utiliser un cœur nouvellement alloué. L'idée serait donc de pouvoir prédire avec 10 secondes d'avance le manque de ressources CPU du S-CSCF afin de pouvoir allouer un cœur supplémentaire au bon moment sans perturber le système et sans gaspiller la ressource CPU du système.

3.3.3 Algorithme de décision par seuil prédictif

Dans la précédente analyse, nous avions pu voir que la courbe du CPU du S-CSCF suit une forme exponentielle. Nous continuons donc à relever les valeurs de la même manière que précédemment, mais nous allons en plus essayer de prédire les valeurs avec dix secondes

d'avance. Nous avons choisi cet intervalle parce qu'il correspond au temps nécessaire pour qu'un cœur nouvellement alloué au S-CSCF soit totalement intégré au système. Dans notre démarche de prédiction, nous essayons de prédire les valeurs du CPU que lorsque nous sommes dans une phase de croissance du nombre d'appels par seconde. Il est important de noter que dans le cas d'une prédiction non fiable, si nous prévoyons une valeur trop élevée nous allouerons un cœur alors que cela n'était pas nécessaire, notre utilisation du CPU ne sera alors pas optimale, mais le système continuera de fonctionner. Au contraire, si nous prévoyons, toujours dans une phase de croissance, une valeur trop faible, le système risque d'être en difficulté et le nombre de sessions interrompues de manière non souhaitée va augmenter déclenchant l'état d'urgence du système ce qui entraînera l'ajout d'un cœur. Dans le cas d'une prédiction fiable, nous allouerons le cœur au bon moment pour que ce dernier soit opérationnel lorsque le système en aura réellement besoin. Dans les phases de décroissance du nombre d'appels par seconde, nous avons choisi de ne pas prédire la valeur du CPU. Bien que la prédiction soit possible et ayant été réalisée et testée comme fiable, nous choisirons de retirer un cœur du système que lorsque la valeur réelle de la consommation du CPU est inférieure à un certain seuil. Ce seuil est calculé grâce à la formule de $seuil_{min}$ présenté à la section 3.3.2. Cette décision est justifiée par le fait que si nous enlevons un cœur quand nous prédisons que la valeur va être suffisamment basse alors nous engendrons une période de famine pour le système. Cette famine survient entre le moment où nous avons prédit la valeur faible puis retiré le cœur et le moment où le système n'avait réellement plus besoin du cœur. L'algorithme que nous avons réalisé dans cette partie suit la logique exposée dans l'algorithme VI.1 présenté à l'annexe VI.

Pour la prédiction du nombre d'appels par seconde, nous retrouvons l'algorithme VI.4 de l'annexe VI dans lequel nous proposons d'utiliser le modèle de régression linéaire. Cet algorithme doit donc être adapté en cas de changement de modèle pour que la prédiction soit la plus fiable possible, car elle est utilisée pour la prédiction du CPU. Nous proposons, pour la prédiction du CPU (voir algorithme VI.5 de l'annexe VI), d'utiliser le modèle de régression exponentielle. Ces modèles de régression ont été choisis après l'analyse des résultats obtenus pour ces deux métriques. Nous fixons nous même le modèle du nombre

d'appels par seconde dans le simulateur. Ce dernier est croissant, constant ou décroissant donc un modèle linéaire semble le plus adapté pour représenter l'évolution de cette métrique. Pour le CPU, nous avons observé le CPU en fonction du cps. Comme nous l'avons précédemment expliqué, la prédiction du CPU nous intéresse qu'aux phases où le cps est croissant. En analysant les données relevées lorsque le cps est croissant, nous avons obtenu pour le CPU en fonction du cps une courbe de tendance dont le modèle est exponentiel. C'est donc suite à cette observation que nous avons décidé d'utiliser le modèle exponentiel pour prédire l'évolution du CPU en fonction du cps. Les prédictions du cps et du CPU se basent donc sur des techniques de régression. La fiabilité de ces techniques est mesurée par le coefficient de régression qui détermine si la régression utilisée est adaptée pour décrire la distribution des valeurs fournies pour la calculer. Dans le cas où le coefficient est faible, nous avons décidé de ne pas prendre en compte la valeur prédite pour éviter que l'outil MAA prenne une mauvaise décision à cause d'une mauvaise prédiction. Nous avons pris cette décision car nous avons constaté que ce coefficient remonte rapidement. En effet, les mauvaises prédictions surgissent après un changement de pente du cps car les valeurs alors enregistrées n'ont plus la bonne tendance et il faut attendre quelques secondes que de nouvelles valeurs soient relevées pour en avoir suffisamment pour que le modèle de régression choisi soit efficace. Les algorithmes pour calculer les coefficients pour les deux régressions étudiées sont reportés en annexe VII.

Pour la prédiction de la charge du CPU du S-CSCF avec 10 secondes d'avance, nous utilisons l'algorithme VI.5 présenté à l'annexe VI. Il nous permet d'obtenir les résultats illustrés par les figures 3.23, 3.24 et 3.25. Les courbes présentées sur ces figures ont été tracées à partir des valeurs d'une simulation réalisée avec le même profil de trafic que celui décrit à la sous-section 3.3.2.

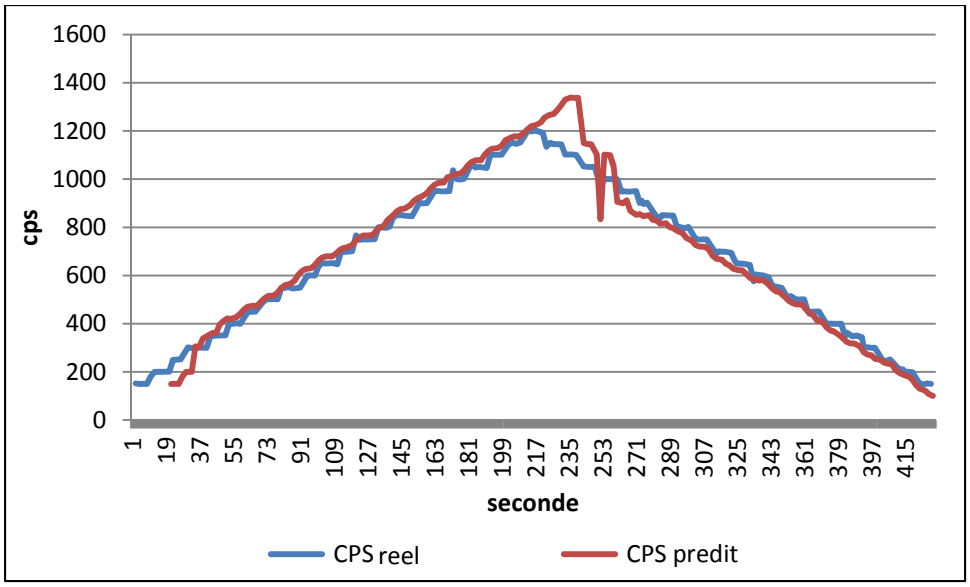

Figure 3.23 CPS réel et prédit avec 10 secondes d'avance

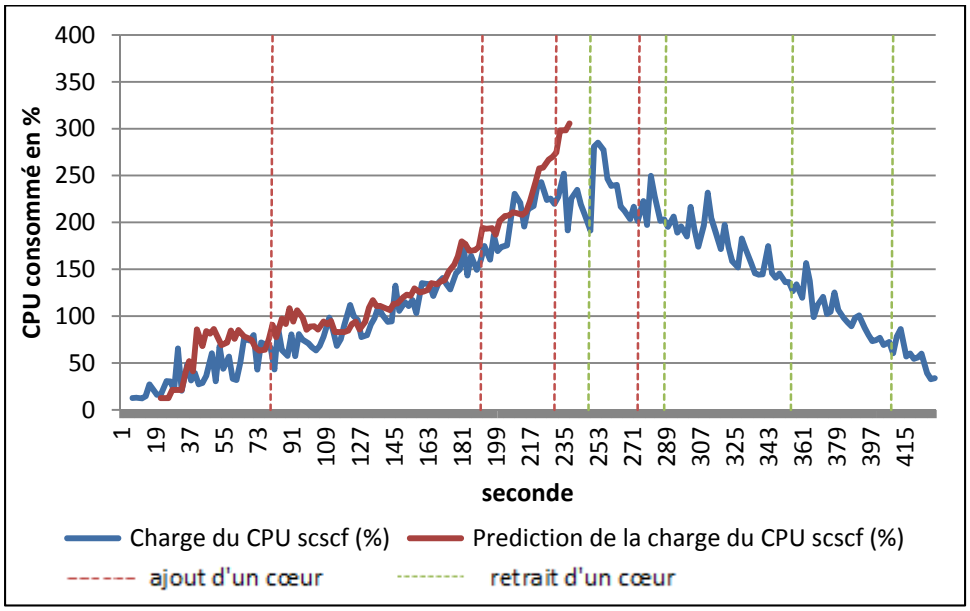

Figure 3.24 Pourcentage de la charge du CPU pour une simulation où la gestion du nombre de cœurs du S-CSCF est automatique, prédictive et basée sur des seuils

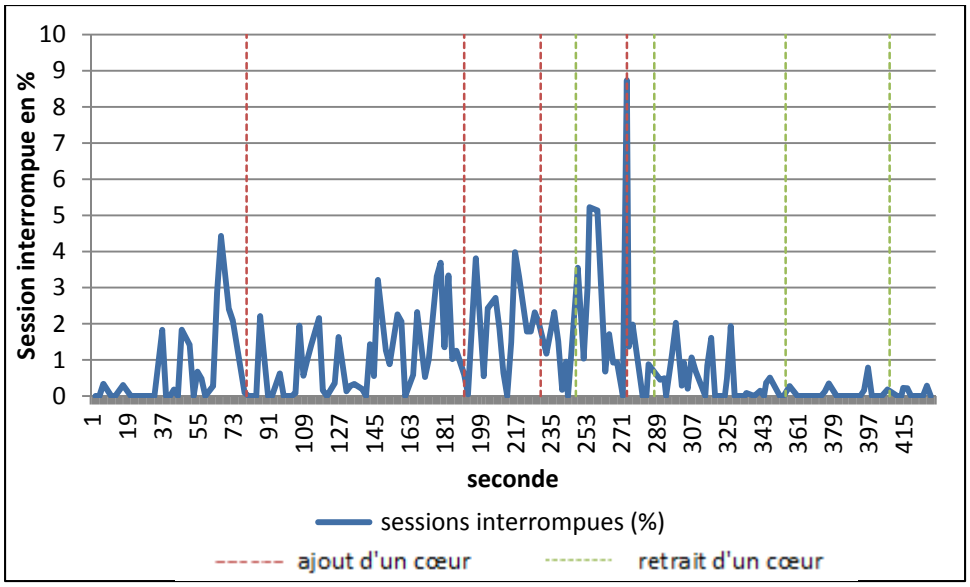

Figure 3.25 Pourcentage de sessions interrompues de manière non souhaitée

Sur l'ensemble des simulations réalisées, la moyenne de la différence entre les valeurs relevées et prédites pour le cps est de 4.5%. Pour le CPU, elle est de 3.3%. La prédiction du cps et du CPU consommé par le conteneur de type S-CSCF sont suffisamment fiables pour pouvoir prédire leurs comportements avec 10 secondes d'avance. Cette prédiction nous permet d'allouer un cœur supplémentaire au S-CSCF que lorsque cela est vraiment utile (voir figure 3.23 et 3.24). Dans ce cas, nous nous apercevons que les cœurs virtuels de la machine physique sont utilisés de manière plus optimale. Dans le cas où la prédiction ne permet pas d'anticiper un pic du CPU inattendu, l'outil de monitorage et d'analyse réagit en adaptant le nombre de cœurs alloué au S-CSCF en fonction du pourcentage de sessions interrompues qui, selon nos paramètres, ne doit pas dépasser 5% (voir figure 3.25).

Tableau 3.2 Utilisation du CPU par le S-CSCF en fonction du nombre de cœurs qui lui est alloué pour l'algorithme par seuil prédictif

Nombre de cœurs alloué au S-CSCF	1	2	3	4	3	4	3	2	1
Utilisation moyenne du CPU (%)	50	50	66	60	83	60	57	50	52

Nous aimerions optimiser encore plus cette utilisation, mais cela n'est pas forcément possible car les cœurs ne sont pas seulement utilisés par le conteneur. En effet, la machine hôte les exploite aussi pour l'outil que nous avons développé comme cela a été montré à la section 3.1. Donc, bien que le conteneur ne les utilise qu'en moyenne à 54% et au maximum à 85%, les cœurs sont en réalité exploités, en moyenne, à 70%. Ce résultat est acceptable et nous pouvons conclure qu'aucun cœur n'est sous-utilisé par le S-CSCF lorsqu'il lui est alloué.

L'utilisation des cœurs virtuels de la machine hôte par notre outil est principalement due aux appels transmis toutes les secondes aux conteneurs et au calcul des paramètres de l'exponentielle pour prédire le CPU consommé par le S-CSCF, spécifiquement dans le cas où nous réalisons l'analyse avec un grand nombre de données. Ce calcul étant réalisé toutes les secondes, nous avons dû réduire le nombre de données en entrée des fonctions servant à prédire le cps et le CPU. Cette réduction du nombre de données n'entraîne pas des résultats de moins bonne qualité. En effet, le coefficient de régression reste identique. Les valeurs prédites sont prises en compte dans le cas où le coefficient de régression est supérieur à 0.3, car nous nous sommes aperçus que toutes les valeurs prédites lorsque le coefficient de régression est inférieur à cette valeur sont aberrantes. Le coefficient est le plus faible dans les zones proches du changement de tendance pour le cps. En effet, dans ces zones les anciennes valeurs ne sont plus exploitables, car l'évolution du système IMS change. De nouvelles valeurs sont relevées toutes les secondes, ce qui permet d'obtenir rapidement une bonne régression (le coefficient de régression se rapproche rapidement de 0.8, valeur jugée bonne dans la littérature). D'après les simulations effectuées, la taille du tableau de données en entrée des fonctions de prédiction doit donc être supérieure à 10. Pour choisir la taille maximale du tableau nous servant à calculer les coefficients des différentes régressions réalisées ici, nous avons relevé le temps d'exécution de notre algorithme pour différentes tailles du tableau de données. Nous obtenons alors le graphique de la figure 3.26 sur lequel nous constatons que pour un tableau de taille 30 au maximum, le temps d'exécution est en moyenne inférieur à 400 ms et au plus égal à 900 ms. Cela est suffisant pour nous car notre échelle de temps est la seconde. Nous avons donc fixé dans MAA une variable afin de brider la taille du tableau de données en entrée des fonctions de prédiction du cps et du CPU à 30.

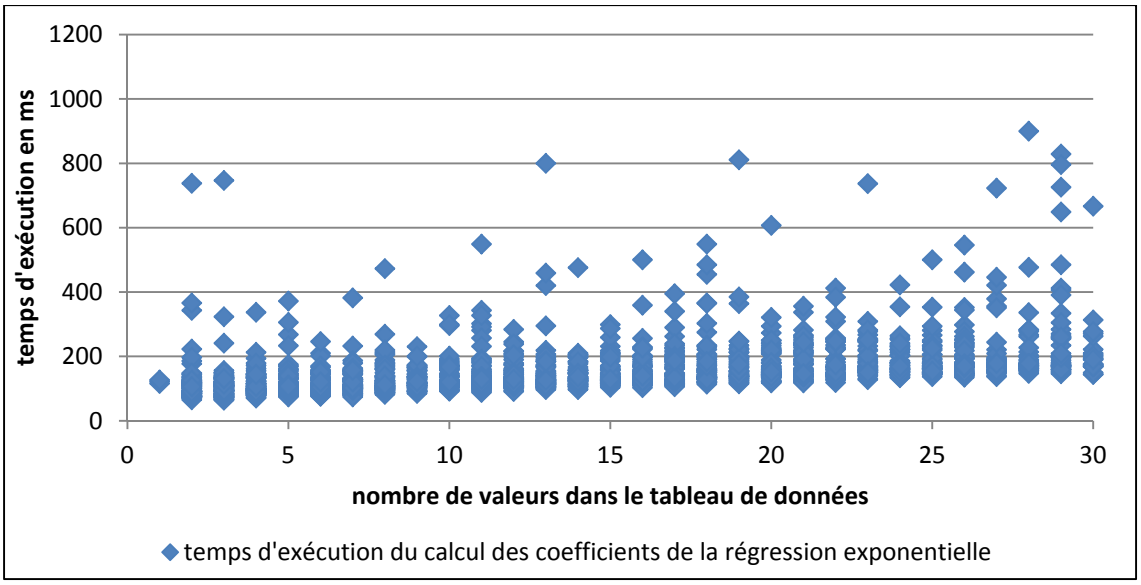

Figure 3.26 Temps d'exécution en milliseconde du calcul des coefficients de la régression exponentielle pour différentes tailles du tableau de données en entrée

Malgré ce temps d'exécution correct, nous avons cherché à trouver une solution qui n'implique plus ce calcul dont la complexité est de 6 O(N). Le calcul de la complexité est détaillé à l'Annexe VII. Dans la proposition suivante, nous essayerons d'exploiter un phénomène remarqué au fil des simulations, à savoir, la similitude des coefficients de l'exponentielle. En effet, les coefficients de l'exponentielle semblent être les mêmes pour des simulations où le profil du trafic généré est identique et où le système IMS conserve la même configuration en terme de modules déployés (leurs distributions et les ressources qui leur sont allouées).

3.3.4 Algorithme de décision par seuil prédictif moins gourmand

Dans les simulations réalisées pour tester l'algorithme VI.1, nous nous sommes aperçus que les paramètres de l'exponentielle restaient relativement constants. En utilisant l'algorithme précédent et en faisant varier le jeu de test et donc le profil de trafic généré, nous nous sommes aperçus que les paramètres pouvaient être prédits si le lambda de la simulation est connu. Ce paramètre, lambda, sera défini comme le quotient de deux différences. Au numérateur, la différence entre le nombre d'appels par seconde relevé à deux instants

distincts. Le dénominateur représente le temps écoulé entre les deux relevés du cps du numérateur.

$$\lambda = \frac{cps_{t'} - cps_t}{t' - t} \tag{3.2}$$

Nous avons donc décidé, dans ce cas, de construire une base de données regroupant les coefficients de la régression exponentielle qui correspondent le mieux aux données relevées lors des simulations, pour différents lambda. Une phase de configuration sera alors nécessaire pour initialiser cette base de données. Ainsi, lors de l'utilisation de notre outil, une fois celui-ci configuré pour la machine sur laquelle il est déployé, nous pourrons récupérer dans cette base de données la valeur des coefficients de l'exponentielle en fonction du lambda mesuré pendant l'utilisation du système IMS. Ces coefficients nous permettront alors de pouvoir prédire les valeurs de la charge du CPU tant que le lambda ne varie pas. Nous éviterons ainsi une des tâches lourdes que devait réaliser l'outil d'analyse avec la précédente proposition, à savoir, le calcul chaque seconde des coefficients de l'exponentielle. Dans le cas où nous n'avons pas, dans la base de données, les coefficients correspondant au lambda courant, nous appliquons l'algorithme VI.5 pour prédire la valeur du CPU. Comme dans la sous-section 3.3.3, pour les phases où le cps décroît, nous n'ôtons pas des cœurs au système par prédiction du CPU mais grâce à la valeur lue à l'instant t. Dans ce cas-ci, la seule fonction qui change par rapport l'algorithme VI.1 est donc celle de prédiction du CPU. Nous allons la décrire sous sa nouvelle forme dans l'algorithme VIII.1 de l'annexe VIII.

Avant de tester notre algorithme, nous avons dû créer la base de données. Pour ce faire, nous avons lancé plusieurs simulations en allouant, à chaque fois, au système toutes les ressources qu'il peut utiliser dans le pire cas. Le pire cas étant, ici, celui qui nécessite le plus de ressources CPU pour le conteneur S-CSCF. Dans notre cas, nous disposons d'au maximum quatre cœurs pour le S-CSCF, nous lui avons donc alloué les quatre. A chaque simulation, nous avons enregistré le lambda, la tendance du cps (croissant, constant ou décroissant) et les différents coefficients de la régression exponentielle calculés avec l'algorithme VI.5 précédemment présenté. Lorsque que la base de données possédait, pour chaque tendance du cps, des valeurs pour une plage suffisamment large de lambda ([lambda attendu pendant la

simulation-1; lambda attendu pendant la simulation+1]), nous avons pu lancer des simulations utilisant l'algorithme VIII.1. Nous avons ainsi obtenu les résultats illustrés aux figures 3.27 et 3.28 pour une simulation avec le même profil de trafic généré que celui présenté à la section 3.3.2.

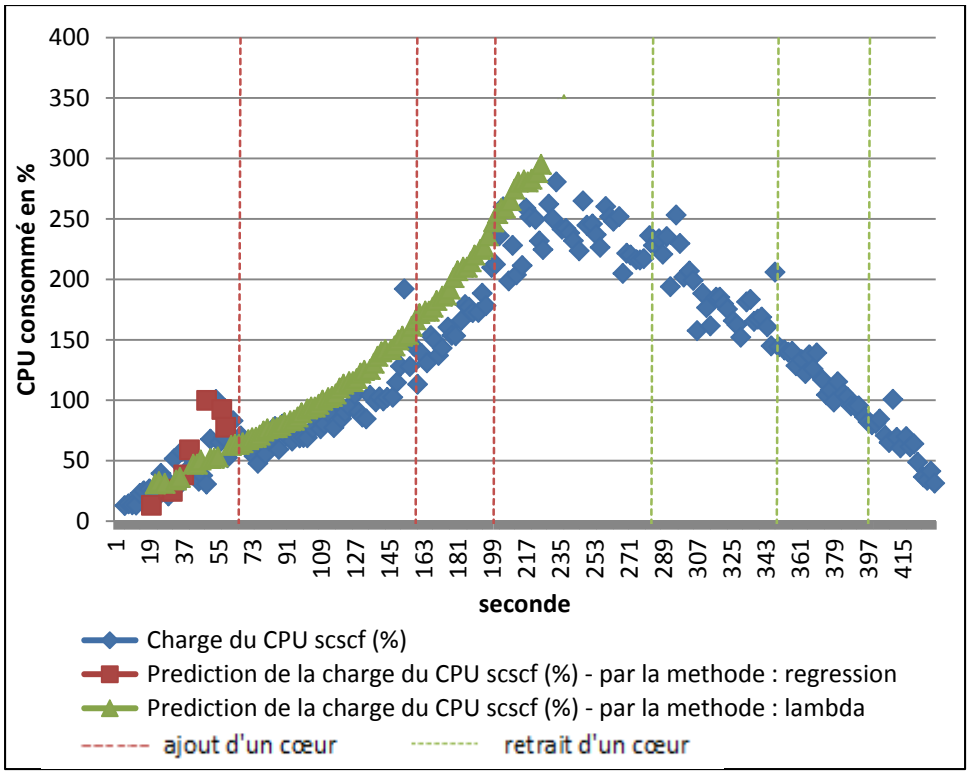

Figure 3.27 Pourcentage de la charge du CPU lors d'une simulation où la gestion du nombre de cœurs du S-CSCF est automatique, prédictive et basée sur des seuils

Sur la figure 3.27, nous pouvons visualiser trois ensembles de points. Les points bleus correspondent aux valeurs relevées, en temps réel, au niveau de la charge du CPU consommée par le conteneur de type S-CSCF. Les points rouges correspondent à la prédiction faite par l'algorithme VI.5. Nous pouvons constater qu'il permet de prédire les quelques points que l'algorithme VIII.1, testé ici, ne permet pas de prédire car le lambda calculé n'était pas dans la base de données. Les points verts sont les points prédits grâce à l'algorithme VIII.1 testé ici. Sur l'ensemble des simulations réalisées pour tester cet algorithme, la moyenne de l'erreur entre les valeurs relevées et prédites est de 9.9%.

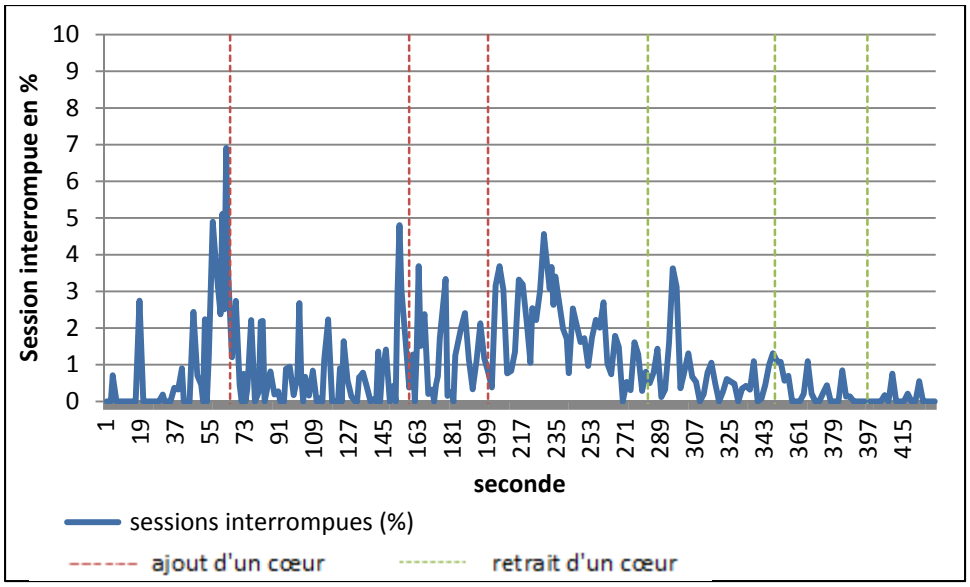

Figure 3.28 Pourcentage de sessions interrompues de manière non souhaitée

Dans ce cas, après configuration, la base de données est adaptée à l'ordinateur sur lequel l'outil est déployé. La base de données ne peut pas forcément être partagée entre plusieurs ordinateurs, car la consommation du CPU faite par un conteneur S-CSCF peut varier d'une machine hôte à une autre. Elle permet donc de prédire relativement fidèlement la charge consommée du CPU de la machine hôte concernée. Nous constatons cependant, lors de plusieurs simulations, qu'un problème se répète. En effet, comme nous pouvons le voir sur la courbe du CPU relevée, au début de chaque simulation la charge du CPU a tendance à augmenter légèrement. Cette augmentation peut se situer, comme c'est le cas dans nos simulations, dans une zone où la charge maximale du CPU est presque atteinte et donc à un instant où un nouveau cœur va prochainement être alloué au S-CSCF. Mais notre prédiction, se basant sur un modèle d'exponentielle construit dans un cas idéal, ne nous permet pas de prédire ces valeurs entraînant une augmentation du nombre de sessions interrompues de manière non souhaitée (voir figure 3.28). Le système IMS entre alors dans un état critique et un cœur est ajouté. Cela représente un problème puisque nous pourrions franchir le seuil du nombre maximal de sessions interrompues par seconde fixé par le SLA à chaque simulation. L'autre contrainte de cet algorithme est de déterminer le moment à partir duquel la base de données est suffisamment complète pour que l'algorithme soit efficace. Dans notre cas, nous

avions besoin d'une dizaine de simulations pour que la base de données couvre pour chaque tendance du cps des plages suffisamment larges de lambda (une plage est un ensemble défini par [lambda attendu-1;lambda attendu+1]). Un dernier problème que nous avons rencontré lors de certains tests est celui des valeurs aberrantes enregistrées dans la base de données pendant la phase de configuration. En effet, si lors d'une simulation de configuration un problème se produit, une valeur aberrante peut être enregistrée dans la base de données et cette dernière aura des répercussions sur toutes les simulations futures qui l'utiliseront. Nous recommanderons donc davantage l'algorithme VI.1 étant donné qu'il est plus fiable et même s'il nécessite plus de temps de traitement, il reste relativement rapide par rapport à notre échelle de temps.

3.3.5 Algorithmes de prédiction SVM

Afin d'essayer d'améliorer la prédiction que nous faisons du CPU, nous avons testé d'autres méthodes s'appuyant sur les SVM décrits à la sous-section 1.5.2. Parmi toutes les techniques des SVM, nous nous sommes concentrés sur celles adaptées à la régression. Nous avons alors étudié deux techniques principales que nous allons détailler ci-dessous. La première SVR pour *Support Vector Regression* en anglais, et la seconde LS-SVM pour *Least Squares Support Vector Machine* qui peut elle-même se décliner en deux sous-techniques en fonction du jeu de données en entrée. Dans un cas (dynamique), nous donnerons que les N dernières valeurs mesurées, dans l'autre cas, nous donnerons toutes les valeurs relevées.

SVR

En 1996, V. Vapnik et al. (Smola and Vapnik 1997) proposent une méthode SVM pour résoudre des problèmes de régression. Dans cette méthode, les valeurs en entrée sont des données d'entraînement afin de déterminer le meilleur modèle pour prédire de nouvelles valeurs. Le modèle est ici représenté par une fonction f. Le but de cette méthode est de déterminer la fonction f la plus monotone possible, mais qui regroupe le plus de points possible. Pour ce faire, on détermine ε, cette variable représente la largeur de la marge autour

de *f*. Tous les points du jeu d'entraînement situé dans la surface de la marge autour de *f* sont considérés comme étant sur *f*. Cette méthode fixe aussi la variable *C*. La variable *C* est un coefficient de contrôle de l'influence de l'erreur sur la largeur de la bande. On obtient alors une fonction de coût à minimiser. Ce coût est la somme entre la complexité de la fonction *f* et les distances ξ pondérées par *C* entre les points en dehors de la marge et la marge.

$$coût = Complexité(f) + C \sum \xi \tag{3.3}$$

Minimiser cette fonction permet d'avoir une fonction *f* qui ne soit pas trop complexe, mais qui représente fidèlement les données du jeu d'entraînement. La figure 3.29 illustre les termes présentés ici.

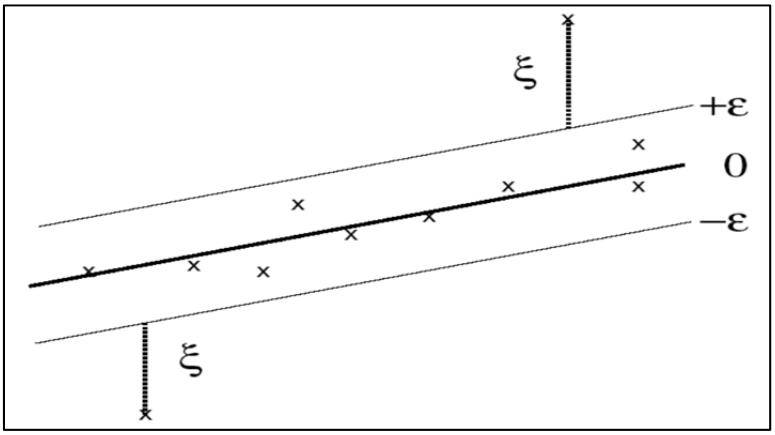

Figure 3.29 Représentation graphique de la marge ε et de la distance ξ entre les points hors de la marge et celle-ci
Tiré de (Schölkopf and Smola 2002)

Pour tester cette méthode, nous avons utilisé la bibliothèque LIBSVM (Chang and Lin 2011) développée par CC. Chang et CJ. Lin. Le code Matlab écrit pour exécuter la méthode SVR est disponible en annexe IX. Lors de la recherche des paramètres optimaux à cette méthode, nous avons utilisé la technique de validation croisée. Cette technique consiste à calculer pour chaque combinaison de paramètres possibles la fidélité du résultat (en essayant de prédire des résultats que nous avons déjà et en calculant l'erreur réalisée) et à garder les paramètres qui donnent les meilleurs résultats. Pour appliquer cette technique, nous avons restreint l'ensemble des valeurs possibles pour les paramètres à $[2^{-5}; 2^{15}]$ pour C et à $[2^{0}; 2^{15}]$ pour ε.

Ces choix d'intervalles sont ceux conseillés par les auteurs de la bibliothèque LIBSVM dans l'article (Chang and Lin 2011) associé à la bibliothèque utilisée.

LS-SVM et DLS-SVM

En 1999, JAK. Suykens et al. (J.A.K. Suykens 2002) proposent une méthode des SVM pour résoudre des problèmes de régression, la méthode LS-SVM pour *Least Squares Support Vector Machine*. Comme l'expliquent RM. Baladin et EI. Lomakina dans leur article (Balabin and Lomakina 2011), cette méthode ressemble beaucoup à la méthode SVR précédemment décrite. Dans cette méthode aussi, il faut donner en entrée un jeu d'entraînement à partir duquel sera déterminé et testé un modèle servant à prédire de prochaines valeurs. La principale différence entre les deux méthodes repose sur la notion de marge. En effet, dans la méthode LS-SVM cette dernière n'existe pas. Les données expérimentales du jeu d'entraînement et la fonction f recherchée pour décrire le modèle sont comparées par la méthode des moindres carrés. Toutes les erreurs entre les données relevées et la fonction mathématique sont prises en compte. Un argument γ, comparable à C dans la méthode SVR, est alors défini pour contrôler la forme de la courbe du coût. Dans ce cas aussi, le coût est la fonction que l'on cherche à minimiser. Il lie la complexité de la fonction f du modèle et l'erreur. Le second paramètre de cette méthode est σ, il correspond à la largeur du noyau RBF utilisé par la méthode de régression. La figure 3.30 peut alors illustrer la différence du calcul du coût avec les deux méthodes présentées ici. Sur chaque graphique de la figure, l'axe des abscisses représente l'erreur (calculée comme étant la distance entre les points du jeu de données et la fonction f. Elle est considérée nulle pour les points dans la marge pour la méthode SVR). L'ordonnée de chaque graphique est le coût à minimiser.

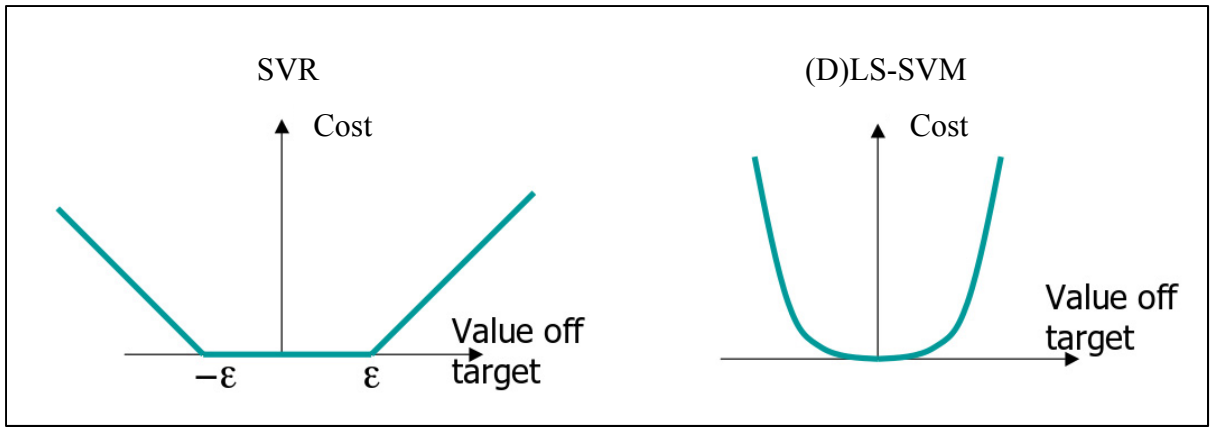

Figure 3.30 Comparaison du calcul du coût pour les modèles de SVR et LS-SVM
Tiré de (Schrater, 2002)

Une autre méthode, dynamic LS-SVM (DLS-SVM), présentée par Y. Fan et al. (Fan, Li, and Song 2006) et issue de LS-SVM a aussi été testée. La différence entre LS-SVM et DLS-SVM repose sur le jeu de données fourni à la méthode. Dans le cas de LS-SVM, toutes les données récoltées depuis le début de la simulation sont fournies pour calculer la régression. Dans le cas de DLS-SVM, la taille du jeu de données est fixe, donc uniquement les *n* derniers relevés sont fournis à la méthode. La valeur *n* représente la taille du jeu de données que nous avons fixé.

Pour implémenter un script Matlab pour ces méthodes, nous avons utilisé la bibliothèque LS-SVMlab1.8 développée par JAK. Suykens et al. (J.A.K. Suykens 2002). Le script développé est présenté en annexe X. Comme pour SVR, le choix des paramètres de cette méthode s'effectue par validation croisée. Dans ce cas, les intervalles de recherche sont définis dans la bibliothèque utilisée. Les méthodes LS-SVM et DLS-SVM sont donc testées grâce au même script Matlab, la différence se faisant dans les données en entrée.

Dans tous les cas, le but des méthodes est de minimiser le coût en identifiant les paramètres du modèle de prédiction. Pour appliquer ces méthodes, il faut donc identifier ces paramètres. Pour cela, quelle que soit la méthode utilisée, chaque phase de prédiction est précédée d'une phase de configuration. Lors de cette phase, nous calculons sur un petit échantillon de

données la précision de la prédiction (erreur entre les valeurs prédites et celles déjà enregistrées) des deux méthodes en faisant varier tous les paramètres. Nous allons ensuite garder et appliquer aux phases suivantes les paramètres pour lesquels la précision est la plus haute. Il y a ensuite l'étape de l'entraînement. Durant cette étape, la méthode cherche le modèle avec une partie des données d'entrée qui lui permet de prédire le plus fidèlement possible les autres données qu'elle a reçues en entrée. Durant cette phase, les paramètres utilisés sont ceux déterminés précédemment dans la phase de configuration. Pour finir, la dernière étape est celle de la prédiction. Durant cette étape, grâce aux paramètres déterminés à la phase de configuration et au modèle trouvé à la phase d'entraînement, les techniques de SVM utilisées ici nous permettent alors de prédire les valeurs du CPU consommé par le conteneur S-CSCF.

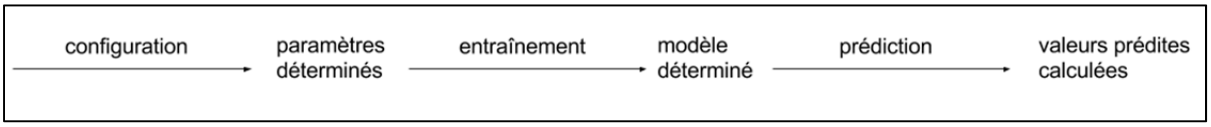

Figure 3.31 Enchaînement des étapes pour prédire des valeurs avec les méthodes SVM

Pour pouvoir prédire des valeurs, il nous faut donc un premier jeu de données pour la configuration, puis un second pour l'entraînement. Nous souhaitons appliquer les méthodes SVM pour prédire les valeurs du CPU d'une simulation de nos jeux de test. Si par la suite nous déployons ces méthodes dans MAA, nous aurons besoin que les méthodes SVM utilisées nous donnent une prédiction fiable mais aussi rapide et qui nous permet de réagir suffisant vite. Nous avons constaté que ces méthodes nous permettent de prédire les valeurs à venir dans un intervalle de temps relativement court. En effet, un évènement perturbateur peut survenir et entraîner une modification du jeu de test, ce qui aura pour conséquence de modifier le modèle de prédiction et donc les valeurs prédites elles-mêmes. Dans cette optique, nous avons choisi des tailles de jeu de test les plus petites possible afin que l'exécution des algorithmes SVR et DLS-SVM soit rapide et reflète au mieux l'état actuel de la consommation du CPU. Le nombre de données ne doit pas non plus être trop faible pour que ces dernières reflètent la complexité du jeu de test. Nous avons alors choisi d'exécuter l'algorithme XI.1 qui regroupe les phases de configuration, d'entraînement et de prédiction,

toutes les 5 secondes. Cet algorithme prendra en entrée les 25 dernières secondes de simulation et prédira les 5 prochaines. Les 25 secondes de simulation utilisées par les algorithmes de prédiction avec les méthodes SVR et DLS-SVM seront découpées en deux. Les 5 premières secondes serviront à la phase de configuration et les 20 dernières à la phase d'entraînement. Ce choix est arbitraire et a été fait car comme mentionné ci-dessus nous avons besoin d'un jeu de données en entrée qui ne soit ni trop petit ni trop grand. 25 secondes semblent être un bon compromis qui se confirme lors des tests préliminaires réalisés. Nous avons ensuite fait le choix de découper ce temps en $\frac{1}{5}$ pour la configuration et $\frac{4}{5}$ pour l'entraînement, car, lors d'analyses préliminaires, nous nous sommes aperçus que ce découpage était le plus efficace pour obtenir un modèle et des paramètres optimaux. Grâce aux bibliothèques Matlab présentées précédemment, nous avons pu tester l'algorithme XI.1 de l'annexe XI avec toutes les méthodes de SVM que nous avons choisies pour la régression. Les résultats sont présentés au chapitre 4.

3.3.6 Traitement des données recueillies

Dans tous les algorithmes décrits précédemment, nous avons appliqué certaines règles pour l'analyse des données. Ainsi nous avons fixé que si une valeur est supérieure ou inférieure à une valeur possible alors elle est ignorée et la seconde étudiée le sera avec la valeur de la seconde précédente. Par exemple, avec trois cœurs, une charge CPU supérieure à 300% est ignorée car cela est impossible. Pour la même raison, une valeur négative pour le CPU ne sera aussi pas prise en compte. Des résultats aberrants peuvent être obtenus dans certains cas, car pour les calculer nous relevons une valeur à deux intervalles de temps distincts et nous faisons la différence de ces valeurs que nous divisons ensuite par la durée de l'intervalle. L'erreur peut alors venir de la précision des opérations mathématiques.

3.4 Conclusion du chapitre

Dans ce chapitre, nous avons commencé par analyser les résultats obtenus pour différentes simulations sur le système IMS virtualisé. À partir de ces observations, nous avons proposé

plusieurs algorithmes pour prédire le CPU consommé par le conteneur S-CSCF du système IMS. Celui proposé à la sous-section 3.3.2 n'était pas prédictif, il ne nous intéresse donc pas pour la suite de notre étude, car nous souhaitons développer un outil proactif pour la gestion dynamique du CPU du conteneur S-CSCF du système IMS virtualisé. Cet algorithme nous a cependant permis de remarquer que le CPU en fonction du cps suit un modèle exponentiel lorsque le cps est croissant. Nous avons donc proposé un second algorithme, l'algorithme VI.1, qui s'appuie sur la régression exponentielle pour prédire le CPU en fonction du cps. Le cps est, quant à lui, prédit par régression linéaire du cps en fonction du temps. Comme cet algorithme est un peu gourmand en ressources pour calculer les coefficients des régressions utilisées, nous avons proposé l'algorithme VIII.1. Cet algorithme nécessite une phase de configuration pour construire une base de données qui regroupe les coefficients des régressions en fonction de la tendance du cps et du lambda. Une fois la base de données complète, elle est utilisée pendant les simulations pour déterminer les coefficients à utiliser pour la régression exponentielle afin de prédire de CPU. Bien que théoriquement cette méthode semble fiable car les coefficients de la régression restent les mêmes pour des simulations similaires, elle ne l'est pas en pratique. En effet, nous nous sommes aperçus que beaucoup de facteurs pouvaient fausser les résultats de la prédiction et entraîner des mauvaises prises de décision par MAA. Nous avons donc privilégié l'algorithme VI.1 pour construire MAA. Bien qu'avec cet algorithme les prédictions soient fiables et que l'outil permette de gérer de manière proactive et dynamique le système IMS virtualisé, nous avons voulu essayer de proposer une nouvelle technique de prédiction du CPU. Nous avons alors étudié la régression grâce aux techniques des SVM. Dans le chapitre 4, nous comparerons ces différentes techniques de régression à travers une analyse des performances de l'outil.

CHAPITRE 4

ANALYSE DES PERFORMANCES

Les résultats présentés tout au long de ce mémoire sont des exemples de résultats reflétant ceux obtenus lors de nombreux tests. Dans cette partie, nous commencerons par détailler la plateforme de test mise en place puis des scénarios que nous allons simuler pour tester notre outil de gestion dynamique et proactive des ressources d'un système IMS virtualisé. Nous proposerons ensuite une synthèse des résultats. Nous testerons aussi la fiabilité de l'outil sur un nombre important de simulations. Pour finir, nous comparerons les résultats de prédiction obtenus avec l'algorithme VI.1 à ceux obtenus grâce aux méthodes des SVM présentés à la sous-section 3.3.5.

4.1 Configuration de la plateforme de test

Pour la réalisation de ce projet, nous disposons de trois machines physiques. La première dispose d'un processeur de 8 cœurs avec une fréquence de 3.60 GHz et un disque dur de 250 Go. Le système d'exploitation qui y est installé est Ubuntu 14.04. Sur cette machine est déployé notre système IMS basé sur le logiciel libre OpenIMSCore. Cet outil nous permet de créer les composants CSCF du système IMS à savoir I-CSCF, P-CSCF et S-CSCF. Chacun de ces composants est placé dans un conteneur LXC. Ces derniers sont reliés à un groupe de contrôle, *CGroup*, qui nous permettra de distribuer les ressources de la machine physique aux conteneurs de la manière souhaitée. Depuis cette première machine physique, nous pouvons également accéder aux deux autres. Sur la deuxième machine de notre nuage informatique privé, qui possède 2 cœurs et 12 Go de mémoire, nous avons virtualisé grâce à la technologie Xen le HSS. La dernière machine dispose de 4 cœurs et de 4 Go de mémoire. Elle nous permet, aussi grâce à la technologie Xen, de virtualiser les six agents SIPp chargés d'établir, de modifier et de terminer les sessions SIP afin de simuler les appels que nous allons monitorer. Plus nous pouvons virtualiser d'agents, plus nous pouvons simuler d'appels. L'architecture déployée est illustrée à la figure 4.1.

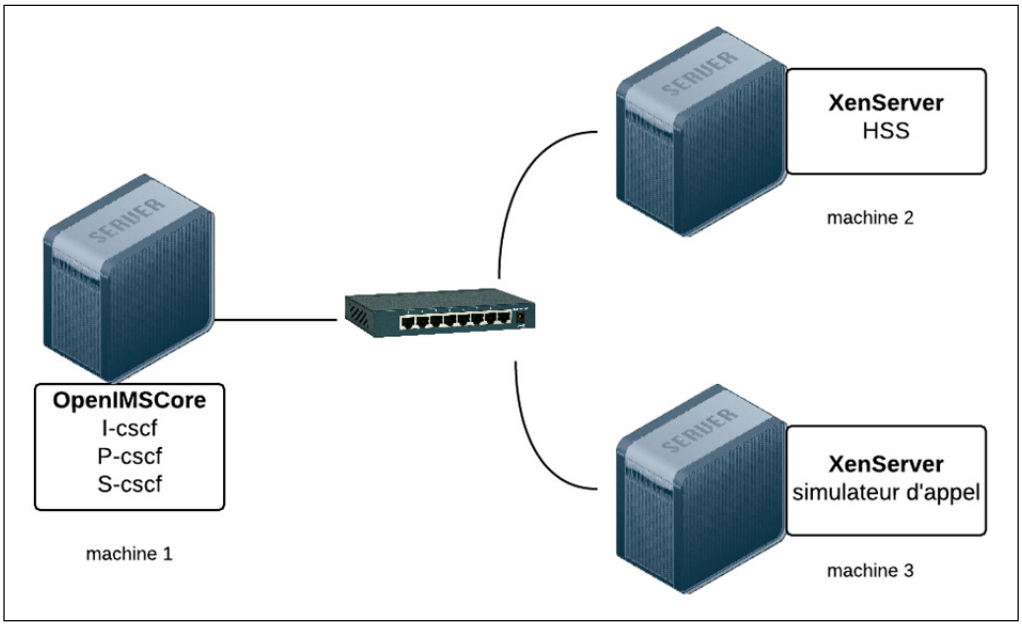

Figure 4.1 Infrastructure expérimentale

L'objectif final est de gérer dynamiquement et de manière proactive les ressources du système IMS, mais nous nous sommes concentrés dans ce mémoire sur la gestion des ressources du module S-CSCF. Nous avons décidé de commencer par ce module car, selon Gong et al. (Gong, Gu, and Wilkes 2010) et Bellavista et al. (Bellavista et al. 2012), ce module représente le goulot d'étranglement du système IMS. Pour atteindre notre objectif et visualiser le mieux cela, nous avons configuré notre système IMS virtualisé en fixant certains paramètres :

- Le nombre d'agents SIP
- Le nombre de conteneurs P-CSCF et le nombre de cœurs que nous leur allouons
- Le nombre de cœurs restants dans la machine 1 pouvant être alloués au S-CSCF

Dans nos simulations, nous avons fixé le nombre d'I-CSCF à un et nous lui avons attribué un cœur. Chaque processeur de notre machine physique a été isolé afin que la machine hôte ne puisse utiliser que le cœur numéro 1 pour les tâches qu'elle exécute en fond. Ce cœur est partagé avec le conteneur I-CSCF. Nous faisons l'hypothèse que, dans notre cas, cela ne pose pas de problème, car ce conteneur sert de passerelle entre les différents systèmes IMS. Mais, comme nous travaillons sur un seul système IMS, ce conteneur ne sera pas vraiment sollicité. Le fait d'isoler les cœurs et de les allouer aux conteneurs nous permet de suivre les

ressources CPU qu'ils consomment. Pour les autres conteneurs, nous hésitions sur le nombre de P-CSCF à créer et sur le nombre de processeurs à leur allouer. Après plusieurs tests, nous avons choisi trois P-CSCF avec un cœur chacun. En effet, les résultats des tests ont montré qu'avec cette configuration nous pouvions simuler plus d'appels et que le système ne s'arrêtait pas parce que les composants de type P-CSCF étaient surchargés. Cela est important pour nous, car ce que nous cherchons à faire ici c'est gérer dynamiquement les ressources du composant S-CSCF. Les autres cœurs de la machine physique seront attribués au S-CSCF. Nous souhaitions garder le plus de cœurs possibles disponibles pour ce composant. En effet, nous avons remarqué que si nous faisons plusieurs fois la même simulation en n'augmentant seulement le nombre de cœurs du S-CSCF, nous pouvons simuler plus d'appels par seconde avant que le système ne s'arrête. Nous allons donc tester l'algorithme VI.1 afin de gérer de manière dynamique et proactive le nombre de cœurs alloué au S-CSCF. Cette valeur pourra varier entre un et quatre étant donné que la machine sur laquelle les conteneurs sont créés possède huit cœurs et qu'un d'entre eux est attribué à I-CSCF et que trois autres sont alloués aux P-CSCF. L'ordinateur sur lequel nous simulons les appels nous permet de créer au maximum six agents SIPp. Nous allons donc rattacher deux agents par composant IMS de type P-CSCF. Ces agents généreront des sessions SIP qui seront transmises au P-CSCF correspondant. Les simulations réalisées seront limitées à cause des ressources du système construit. Cependant, cela n'est pas un problème car nous savons qu'avec plus de machines ou avec des machines plus puissantes, nous pourrions augmenter ces limites. Les résultats et conclusions resteraient les mêmes.

4.2 Description des jeux de test simulés

Lors de la réalisation des relevés de valeurs avec la configuration que nous avons proposée ci-dessus, nous appliquons plusieurs jeux de test. Toutes les simulations commencent avec une phase de préchauffement du système avant que le jeu de test déterminé soit lancé. Le préchauffement du système porte le nombre d'appels par seconde de 0 à 150 cps. Le premier jeu de test créé permet d'atteindre les limites du système et de le faire cesser de manière prématurée. Pour cela la simulation des appels commence à 150 cps et augmente de 50 cps

toutes les 10 secondes jusqu'à ce que le système cesse de fonctionner. Le second nous permet de visualiser le comportement du système quand le nombre d'appels reçus augmente et diminue. Il n'a pas pour but de faire cesser de fonctionner le système, ni de le stresser, mais plutôt de voir, dans le cas d'un réseau bien dimensionné pour la charge qu'il va devoir supporter, son comportement sur une période de pointe. Pour cela, le second jeu de test commence à 150 cps et augmente de 50 cps toutes les 10 secondes jusqu'à atteindre 1200 cps puis décroît de 50 cps toutes les 10 secondes jusqu'à revenir à 150 cps. Le troisième jeu de test nous permet de visualiser le comportement du système quand le nombre d'appels est constant. Pour cela, nous simulerons 600 appels par seconde. Pour finir, un quatrième jeu de test nous a été proposé. Ce jeu semble défavorable à notre outil car, comme expliqué à la section 4.4, il est constitué de nombreuses phases où le cps est constant. Or, lors de la construction de MAA, nous avons davantage privilégié la gestion des phases où le cps est croissant ou décroissant. Ce quatrième jeu de test est illustré à la figure 4.2.

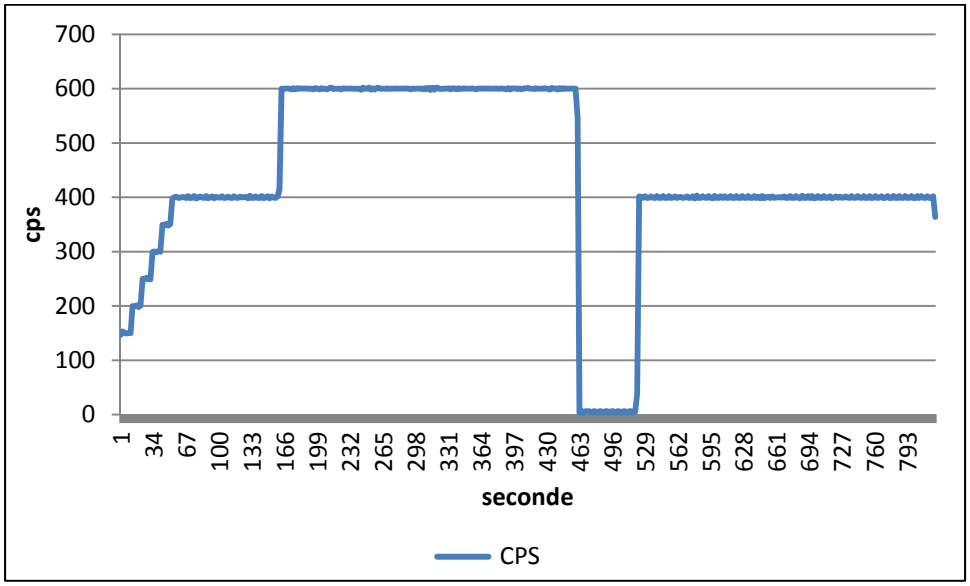

Figure 4.2 Représentation graphique du jeu de test 4

4.3 Analyse du choix des seuils pour la charge du CPU

Tout au long de notre étude, nous nous sommes servis des seuils du CPU déterminés à la partie 3.2.4 de ce document. Nous avons donc utilisé la valeur de 70% comme seuil maximal pour la charge du CPU d'un cœur alloué à un conteneur de type S-CSCF. Cependant, au cours de notre étude, nous nous sommes demandé si nous pouvions augmenter ou si nous devrions diminuer cette valeur. Nous avons donc augmenté la valeur du seuil à 90% puis à 80%. Dans les deux cas, nous nous sommes aperçus que le nombre de sessions interrompues de manière non souhaitée augmentait énormément. Ceci n'est pas acceptable car la solution proposée doit respecter le contrat de service et minimiser la violation du SLA. Nous avons donc abandonné l'idée d'augmenter le seuil. Mais même avec un seuil de 70%, nous avons des moments dans les simulations où le nombre de sessions interrompues atteints les 5%. Afin d'essayer de remédier à ce problème, nous avons cherché pour quelles valeurs du seuil nous n'aurions plus de pics du nombre de sessions interrompues. Nous avons donc modifié l'outil pour qu'il adapte le seuil maximal du CPU en fonction des seuils prélevés lors des simulations précédentes. Pour cela, à chaque simulation, nous enregistrons les valeurs du CPU pour lesquelles le nombre de sessions interrompues atteint les 5%. Ainsi, lors de la simulation suivante, le seuil est calculé en faisant la moyenne de toutes les valeurs enregistrées. Avec cent simulations, nous obtenons la tendance illustrée sur la figure 4.3 pour le seuil maximal du CPU.

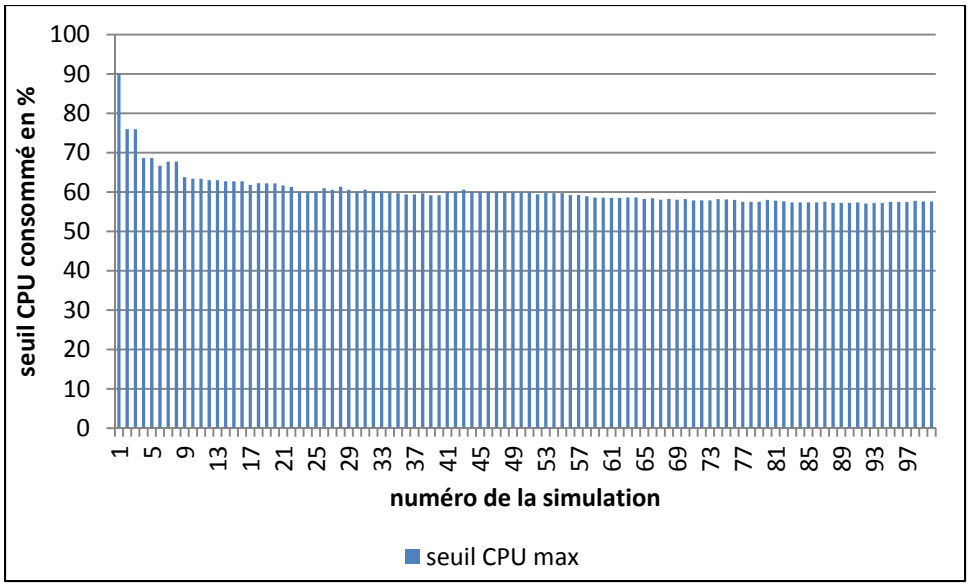

Figure 4.3 Évolution du seuil maximal pour le CPU

Sur le diagramme de la figure 4.3, chaque barre représente la valeur du seuil maximal pris pour une simulation avec le jeu de test 2. Nous nous apercevons alors que le seuil tend vers 57%. Ce seuil est bien plus bas que celui de 70% que nous avions déterminé précédemment. Cependant, nous ne l'avons pas utilisé, car nous nous sommes rendu compte qu'il ne faisait pas diminuer de manière notoire le nombre de fois où le nombre de sessions interrompues atteint les 5%. En effet, pour les mêmes simulations que celles qui nous ont permis de déterminer les 57%, nous obtenons le diagramme en barres de la figure 4.4 pour le nombre de fois où le nombre de sessions interrompues atteint les 5%.

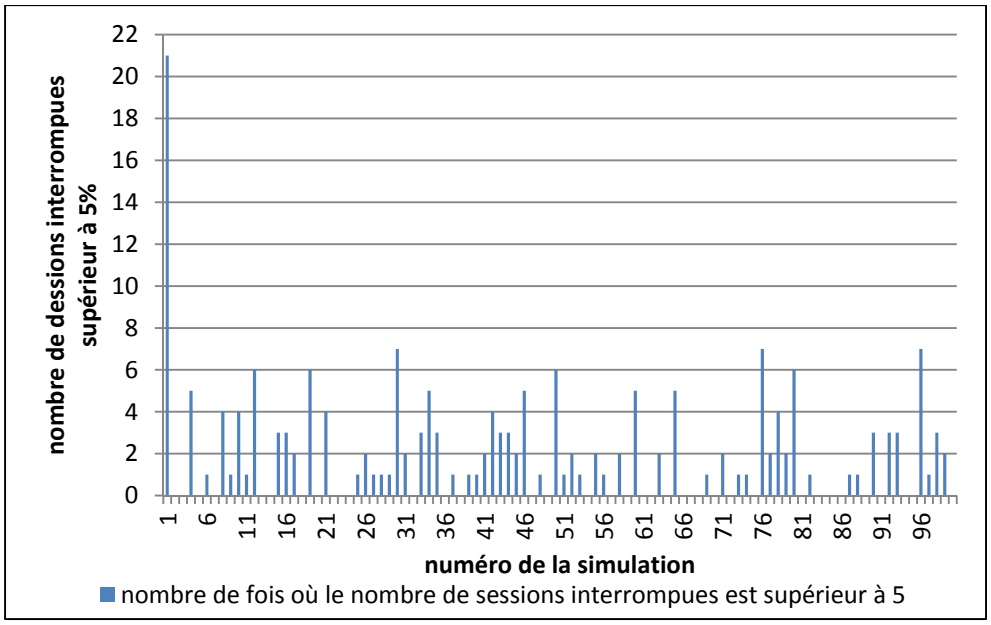

Figure 4.4 Évolution du nombre de fois où le nombre de sessions interrompues est supérieur à 5%

Sur ce diagramme de la figure 4.4, nous remarquons que le nombre de fois où le nombre de sessions interrompues est supérieur à 5% est en moyenne égal à 2. Nous remarquons également que dans le cas où nous fixons le seuil maximal du CPU à 70%, nous obtenons des résultats similaires (voir figure 4.8). Nous avons donc gardé pour toutes nos simulations un seuil maximal de 70% pour la charge du CPU des conteneurs de type S-CSCF. C'est avec ce seuil que nous avons exécuté les simulations présentées dans la section 4.4.

4.4 Analyse des performances de l'algorithme par seuil prédictif

Grâce à notre outil d'automatisation des tests présenté en annexe XII, nous avons réalisé un nombre important de simulations pour chacun de nos jeux de test. Ci-dessous, afin de pouvoir juger de la fiabilité de notre outil de gestion dynamique et proactive des ressources du S-CSCF de notre système IMS, nous allons présenter les résultats obtenus pour chaque jeu de tests présenté à la section 4.2.

Les premiers résultats que nous allons présenter sont ceux du jeu 1. Pour rappel, le premier jeu correspond à un jeu de test qui débute à 150 cps et où le nombre d'appels augmente de 50 cps toutes les 10 secondes. Le but de ce jeu de test est de déterminer le maximum d'appels par seconde que peut supporter le système et d'analyser son comportement avant qu'il ne cesse de fonctionner, car il est surchargé. Nous avons réalisé 100 simulations du jeu de test 1 dans différents cas de figure. En premier, nous avons réalisé les tests sur des simulations où notre outil, MAA, était lancé pour gérer les ressources CPU du conteneur S-CSCF. Ensuite, nous avons relancé la simulation, mais en désactivant MAA. Nous avons, dans ce cas, fixé le nombre de cœurs pour toute la simulation à un, deux, trois ou quatre. Nous obtenons alors les résultats regroupés dans le tableau 4.1.

Tableau 4.1 Moyenne du nombre maximum d'appels par seconde pour le jeu 1

	Avec MAA	Sans MAA			
		1 cœur	2 cœurs	3 cœurs	4 cœurs
M(max(cps))	1584 cps	939 cps	1314 cps	1500 cps	1581 cps

Le tableau 4.1 reporte la moyenne du nombre maximal d'appels par seconde, M(max(cps)), réalisé pour les simulations pour chaque cas de tests du jeu 1. Nous constatons que les valeurs pour les simulations avec MAA et sans MAA mais avec 4 cœurs sont proches. La différence notoire entre ces deux simulations se trouve dans la consommation des ressources. En effet, avec MAA, le système utilise davantage les ressources CPU à sa disposition, il y a donc moins de gaspillage. L'utilisation des ressources CPU est donc beaucoup plus optimisée avec MAA. Nous retrouvons cette observation sur la figure 4.5.

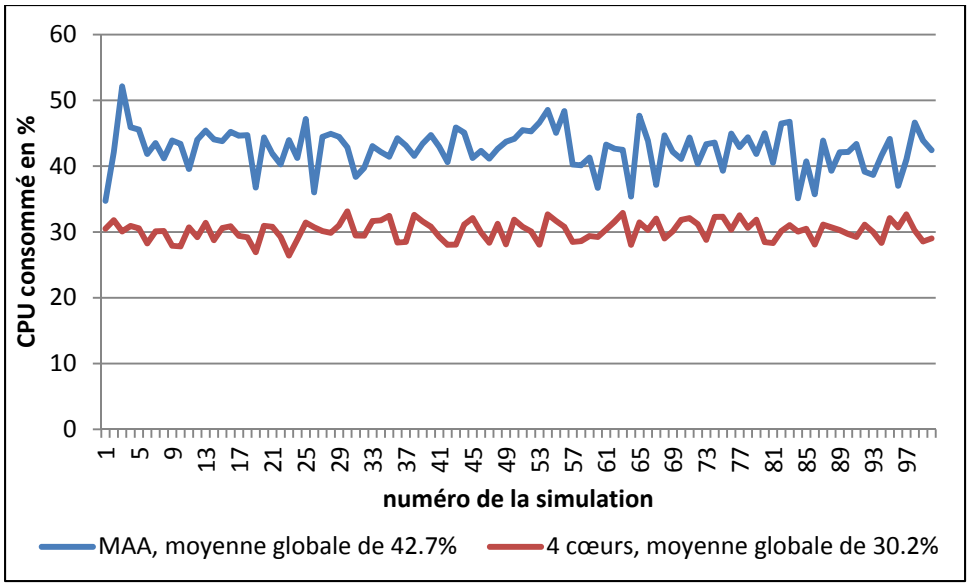

Figure 4.5 Moyenne d'utilisation du CPU pour les simulations du jeu 1

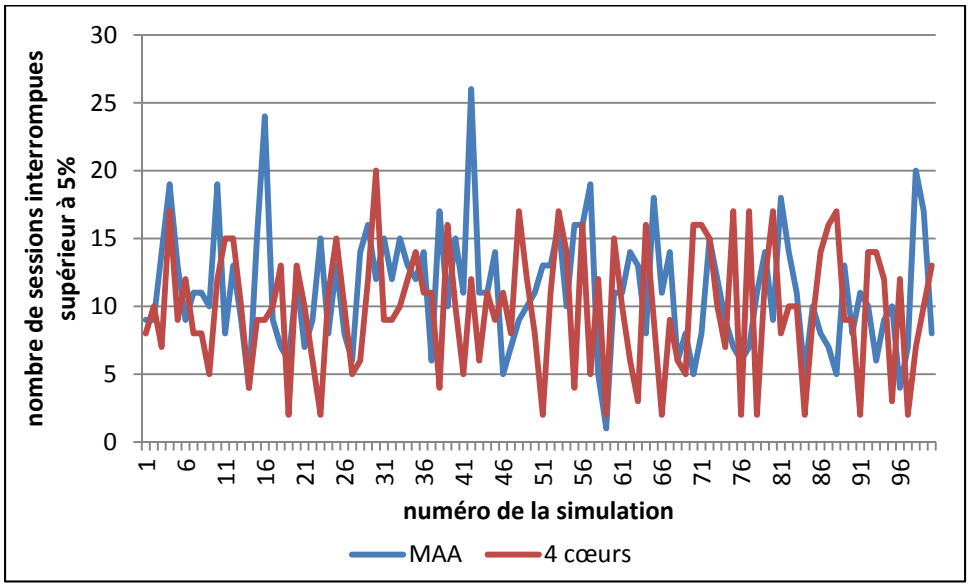

Figure 4.6 Nombre de fois où le nombre de sessions interrompues lors des simulations du jeu 1 dépasse le seuil de 5%

D'après la figure 4.5, les ressources CPU sont plus optimisées avec MAA car lorsque l'outil gère la quantité de ressources disponibles pour le système IMS virtualisé, ce dernier en consomme 42.7%. Pour cela, MAA adapte le nombre de cœurs alloués au système (de 1 à 4) en fonction de ces besoins. Sans MAA et en fixant le nombre de cœurs maximum au système

IMS virtualisé (4 cœurs), le système IMS ne consomme en moyenne que 30.2% des ressources à sa disposition, il y a donc plus de gaspillage. MAA permet donc une réelle réduction du gaspillage des ressources CPU par le S-CSCF. Comme nous pouvons le constater sur la figure 4.6, cette optimisation ne se fait pas au détriment de la qualité de service. En effet, avec MAA, la moyenne du nombre de fois où le système interrompt plus de 5% de sessions est de 11.3% contre 10.1% avec en permanence 4 cœurs. Ces résultats sont relativement proches. Étant donné que les ressources physiques sont à optimiser car elles représentent un coût financier et écologique non négligeable lorsqu'elles sont utilisées à grandes échelles, nous préférerons l'utilisation de MAA avec ce jeu de test. Cette préférence est due au fait que l'utilisation de notre outil permet d'atteindre le même maximum d'appels par seconde (voir tableau 4.1), tout en optimisant les ressources CPU (voir figure 4.5) et sans augmenter considérablement le nombre de fois où le nombre de sessions interrompues dépasse 5% (voir figure 4.6).

Les prochains résultats que nous allons présenter sont les 100 simulations du jeu de test 2. Ce jeu commence avec 150 appels par seconde et augmente de 50 cps toutes les 10 secondes pour arriver à 1200 cps, puis il décroît de 50 cps toutes les 10 secondes pour revenir à 150. Comme précédemment, nous avons réalisé les tests avec MAA et sans MAA mais en faisant varier le nombre de cœurs alloués au S-CSCF de un à quatre. Nous constatons qu'avec 1 et 2 cœurs aucun test ne se termine. Cela s'explique par le fait qu'avec 1 ou 2 cœurs, le S-CSCF n'a pas assez de ressources pour supporter la charge de travail qui lui est demandé lors de la simulation. Avec 3 ou 4 cœurs et avec MAA, toutes les simulations sont exécutées avec succès.

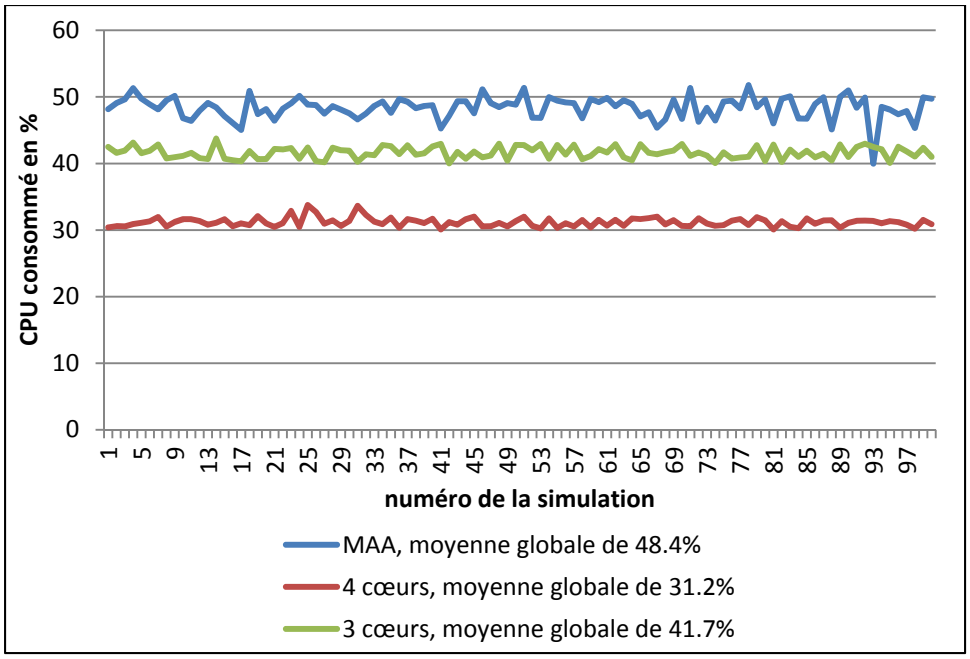

Figure 4.7 Moyenne d'utilisation du CPU pour les simulations du jeu 2

La figure 4.7 montre que la consommation du CPU est plus élevée avec MAA. En effet, sans MAA, le système a, sur une grande partie de la simulation, trop de ressources. Nous pouvons alors encore conclure que MAA permet une bonne optimisation des ressources CPU car il permet d'adapter le nombre de cœurs alloués au système IMS virtualisé en fonction du trafic. Comme nous pouvons le constater sur la figure 4.8, cette optimisation ne se fait pas au détriment de la qualité de service du système qui reste similaire pour les trois configurations de simulation. En effet, la moyenne du nombre de fois où le système interrompt plus de 5% de sessions est de 1.7% pour 4 cœurs en permanence, 2.1% avec MAA et 3.6% avec 3 cœurs en permanence.

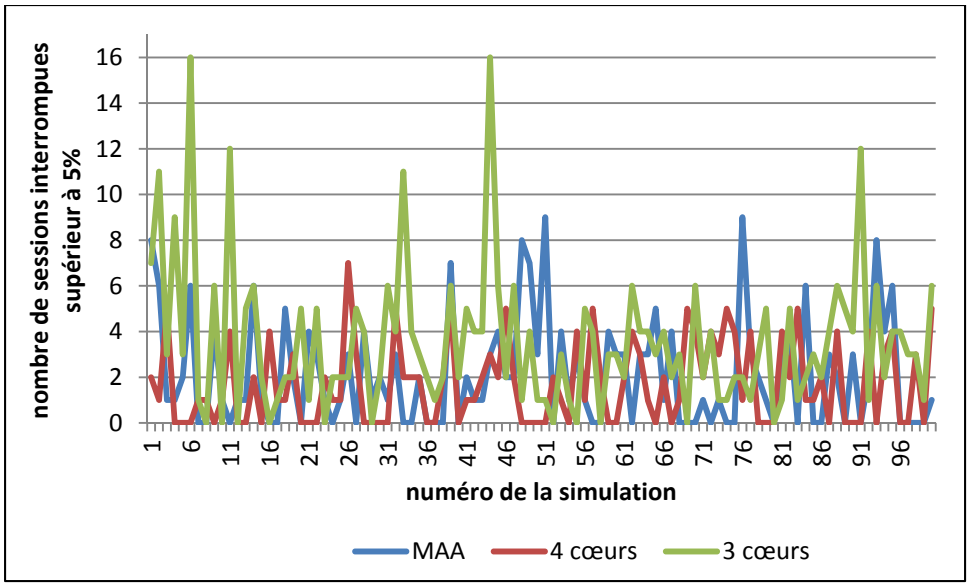

Figure 4.8 Nombre de fois où le nombre de sessions interrompues lors des simulations du jeu 2 dépasse le seuil de 5%

L'outil est relativement rapide à exécuter son algorithme de décision pour savoir s'il faut ou non ajouter ou retirer un cœur au système. En effet, la figure 4.9 montre que le temps d'exécution moyen est de 2 ms, ce qui est bien inférieur à la seconde, notre temps de référence. Cette rapidité d'exécution permet de prendre les décisions à temps et d'éviter un retard lors de l'ajout d'un cœur. Un retard créerait une période de famine en termes de ressources CPU. Cela aurait pour conséquence de faire augmenter le nombre de sessions interrompues.

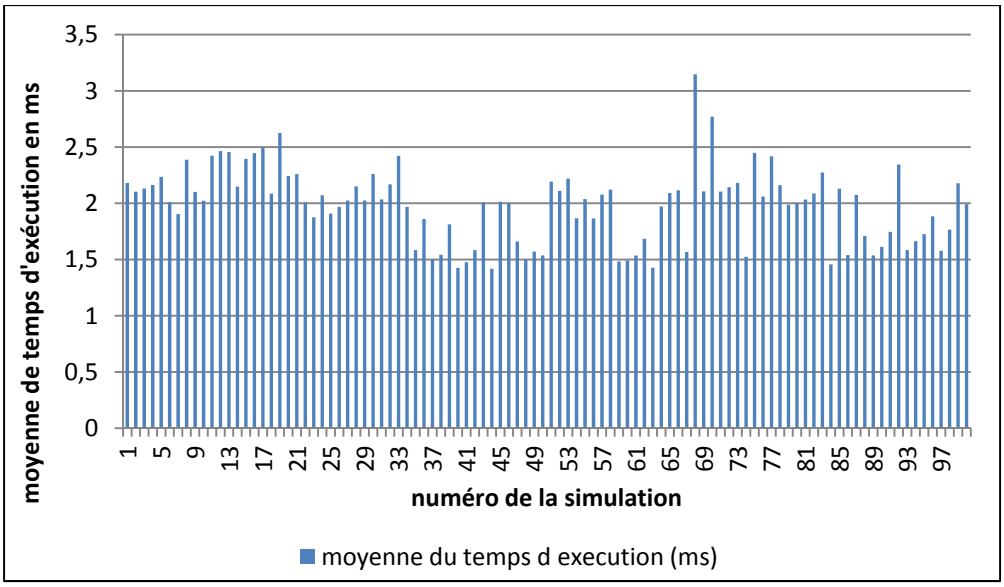

Figure 4.9 Moyenne du temps d'exécution des simulations du jeu 2 avec MAA

Avec ce jeu de test, aussi, nous pouvons conclure que notre outil permet une optimisation des ressources CPU de la machine hôte sans impacter sur la qualité de service du système IMS virtualisé. De plus, l'optimisation est relativement bonne car même si une moyenne de 50% de consommation de CPU semble faible il ne faut pas oublier que le seuil maximal est de 70% et non de 100%.

Nous allons ensuite réaliser 100 simulations du jeu de test 3. Ce jeu se compose d'une phase d'appels par seconde constante et égale à 600 cps. Le but de ce jeu de test est d'observer le comportement du système IMS et la réaction de MAA lors d'une période où le nombre d'utilisateurs ne varie pas. Comme pour les deux jeux précédents, les tests vont être réalisés avec MAA et sans MAA mais en faisant varier le nombre de cœurs alloué au S-CSCF de un à quatre. Dans ces cinq cas, nous constatons que le système finit par cesser de fonctionner. Cependant, l'arrêt du système n'est pas dû à la même raison pour chaque configuration. En effet, avec un, deux et trois cœurs, le système s'arrête quand il vient à manquer de ressources CPU pour le S-CSCF, alors que pour les simulations avec quatre cœurs et avec MAA, il s'arrête car les conteneurs de type P-CSCF cessent de fonctionner. Ce comportement a été expliqué précédemment à la section 3.2.2. Pour le système IMS, avec chaque configuration

testée, nous obtenons les résultats suivant pour le temps moyen de fonctionnement de tous les conteneurs de type P-CSCF, M(Tpcscf), et pour le temps moyen de fonctionnement du système, M(Tsys).

Tableau 4.2 Moyenne du temps de fonctionnement des conteneurs P-CSCF et du système IMS pour le jeu 3

	Avec MAA	Sans MAA			
		1 cœur	2 cœurs	3 cœurs	4 cœurs
M(Tpcscf)	814 sec.				815 sec.
M(Tsys)	828 sec.	222 sec.	468 sec.	797 sec.	831 sec.

Avec MAA ou sans MAA mais avec 4 cœurs, nous constatons que la durée de vie du système est équivalente mais la différence se fait, dans ce cas aussi, sur l'optimisation des ressources CPU. En effet, comme nous pouvons le lire sur le graphique 4.10, le système consomme plus de ressources CPU avec MAA. Cela signifie qu'il consomme plus des ressources mises à sa disposition et donc qu'il en gaspille moins. Cela s'explique par le fait que MAA permet d'adapter le nombre de cœurs alloués (de 1 à 4) au conteneur S-CSCF du système IMS virtualisé en fonction de la charge de travail attendue.

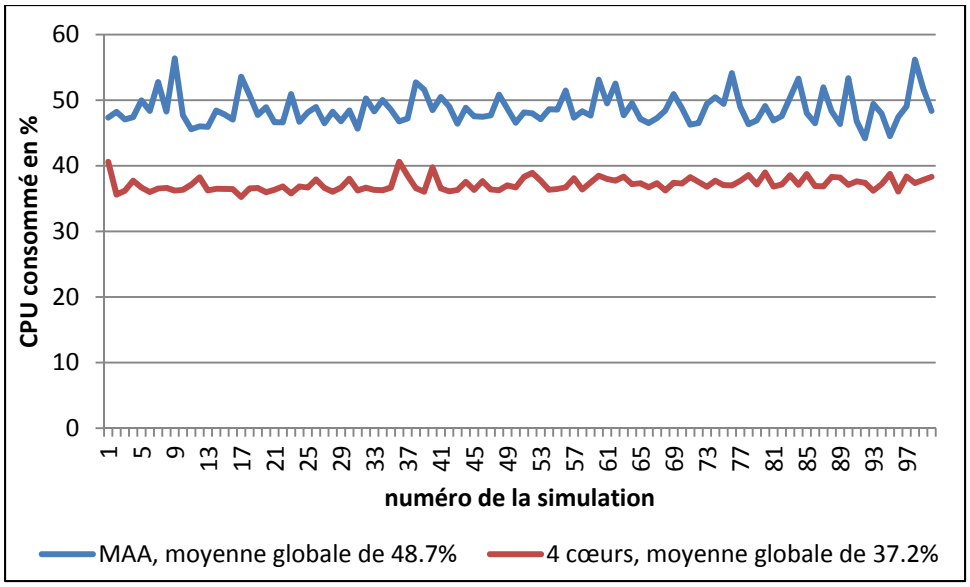

Figure 4.10 Moyenne d'utilisation du CPU pour les simulations du jeu 3

L'optimisation des ressources CPU, dans ce cas aussi, ne se fait pas au détriment de la qualité de service car, comme le montre la figure 4.11, le nombre de sessions interrompues lors des simulations avec MAA et lors des simulations sans MAA et avec 4 cœurs sont proches. En effet, la moyenne du nombre de fois où le système interrompt plus de 5% de sessions est de 6.5% avec MAA et de 4.8% avec 4 cœurs en permanence. Avec ce jeu de test comme avec les deux jeux précédents, la moyenne légèrement supérieure avec MAA est due au fait que l'outil ne prédit pas l'imprévisible, c'est-à-dire les changements du cps. Lors de ces changements, il lui faut un temps d'adaptation pour retrouver une prédiction fiable, durant cette période, le S-CSCF a pu manquer légèrement de ressources CPU.

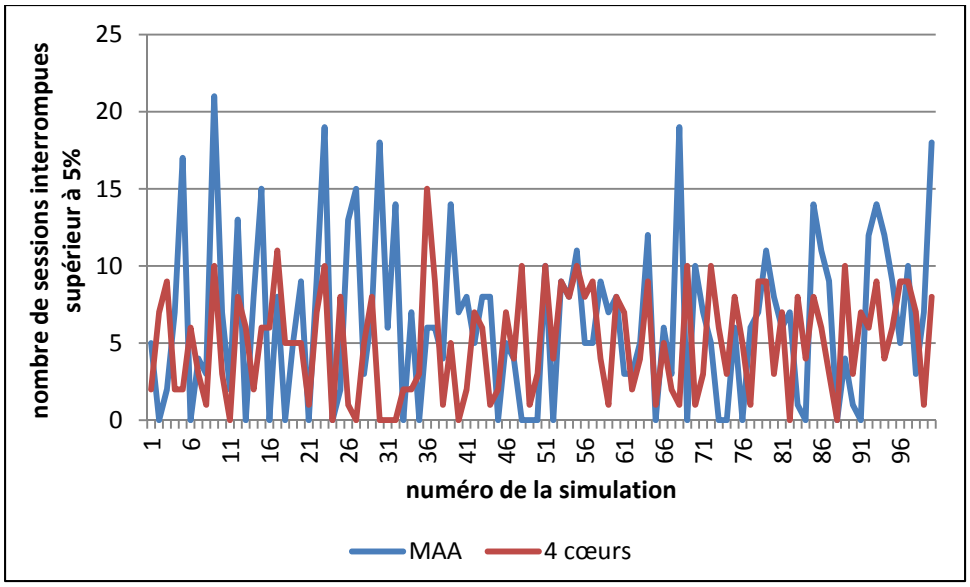

Figure 4.11 Nombre de fois où le nombre de sessions interrompues lors des simulations du jeu 3 dépasse le seuil de 5%

Avec ce jeu de test aussi nous pouvons conclure que l'utilisation de l'outil MAA est bénéfique, car il permet d'optimiser les ressources CPU consommées. En effet, si le conteneur S-CSCF consomme plus de CPU alors il y a moins de gaspillage de ressources. Cela est dû au fait que MAA adapte le nombre de cœurs alloués au S-CSCF (de 1 à 4) en fonction de la prédiction qu'il fait de la future charge de travail du conteneur.

L'optimisation moyenne du CPU pour ces trois jeux de test est de 46.5%. Cela peut paraître faible si on se base sur le fait que la charge du CPU peut varier de 0 à 100%, mais, dans notre cas, afin de garantir une bonne qualité de service, nous avons restreint l'étendue de la charge du CPU de 0 à 70%. Sur cette nouvelle échelle, une charge moyenne de 46.5% est bonne. En effet, comme présenté à la figure 4.12, elle correspondrait à une charge de 66.4% si on reportait nos 70% sur une échelle de 0 à 100.

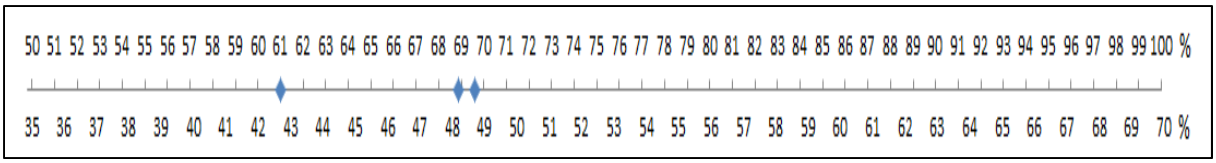

Figure 4.12 Correspondance des moyennes globales du CPU obtenues pour chaque jeu sur les différentes échelles (100% et 70%)

Pour finir, afin de tester notre outil et notre algorithme dans un cas défavorable, nous avons aussi considéré le jeu de test 4. Pour rappel, ce jeu de test a une première phase où le cps augmente, de 150 à 400 cps avec un lambda (comme défini à la section 3.4) de 5, puis quatre phases où il est constant. La première phase constante est à 400 cps, la seconde à 600cps, la troisième à 15 cps et la dernière à 400cps. Ce jeu de test semble ne pas correspondre au modèle exponentiel utilisé dans notre algorithme, car il est composé de nombreuses phases où le nombre d'appels par seconde est constant. La figure 4.13 montre un exemple de valeurs pour une des phases constantes, ici celle à 600 cps.

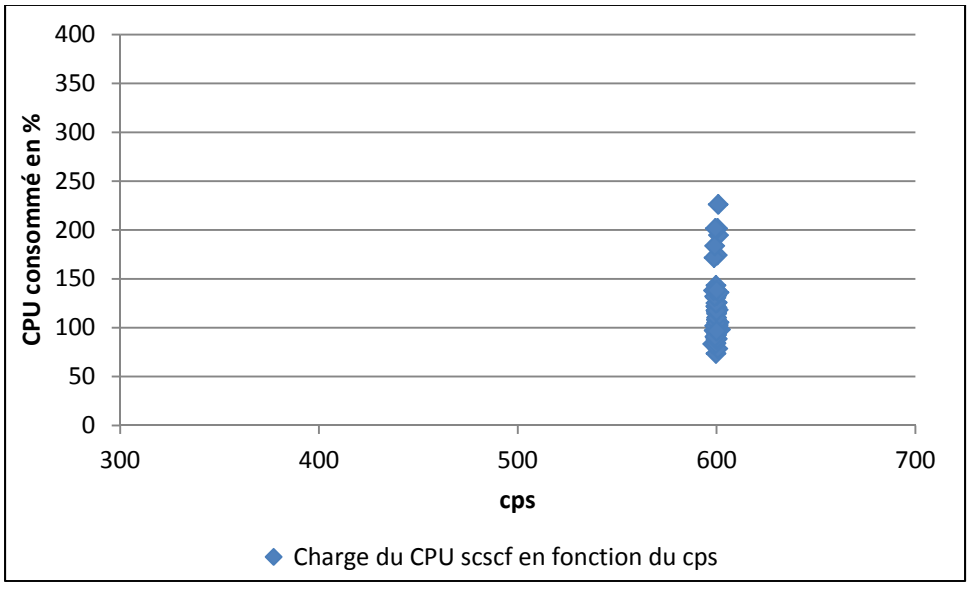

Figure 4.13 Charge du CPU en fonction du cps pour la phase constante à 600 cps du jeu de test 4

Nous avons remarqué dans notre étude que même lorsque le cps est constant, le CPU en fonction du temps peut-être caractérisé par une courbe de tendance exponentielle. Mais dans notre algorithme, pour nous adapter au lambda qui peut varier d'un jeu de test à l'autre, nous étudions le CPU en fonction du cps. Ce jeu de test n'est en réalité pas si mauvais que ça pour notre modèle car l'exponentielle peut prendre une forme telle qu'elle s'adapte à ce cas où le cps ne varie pas. Nous remarquons ceci sur la figure 4.14 où la courbe bleue représente les points calculés avec les coefficients de la régression exponentielle retournés par notre algorithme pour un instant t de la phase constante à 600 cps et où les autres points sont les points de la courbe du CPU en fonction du cps de la figure 4.13.

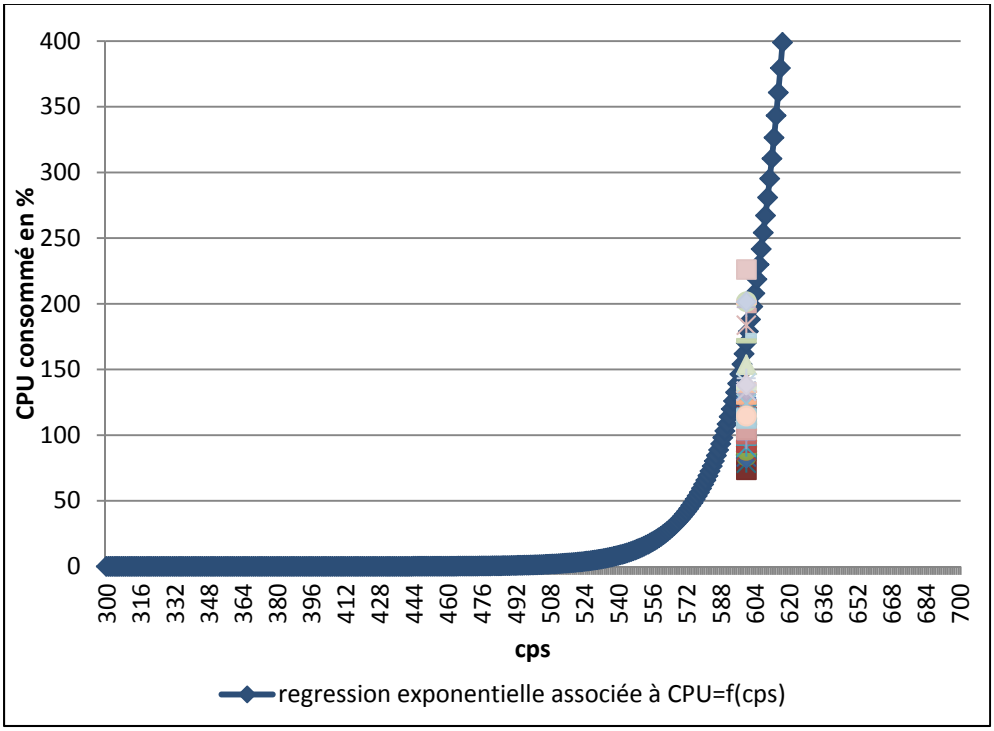

Figure 4.14 Régression exponentielle associée aux points de la fonction CPU=f(cps) de la figure 4.13

Si nous traçons la courbe de la prédiction du CPU par notre algorithme, nous obtenons la figure 4.15.

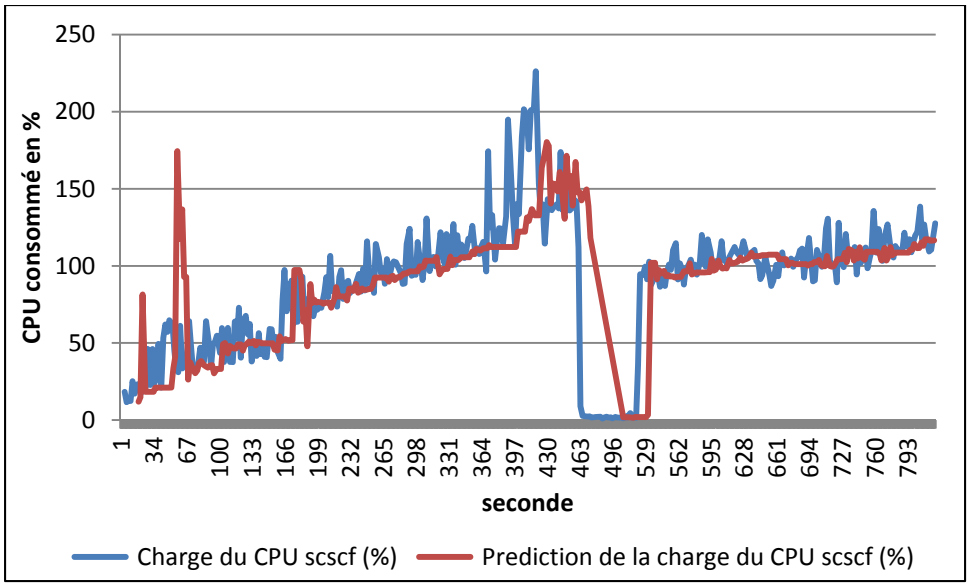

Figure 4.15 Pourcentage de la charge du CPU lors
de la simulation du jeu de test 4

Si nous traçons la courbe de la différence de CPU entre la valeur relevée et la valeur prédite, nous obtenons la courbe bleue de la figure 4.16. Nous remarquons alors les zones où la prédiction a été la plus mauvaise et celles où elle a causé des risques au système car elle a sous-estimé la charge du CPU attendue. Dans ce cas, c'est la partie de contrôle du nombre de sessions interrompues de manière non souhaitée qui a permis de réagir rapidement et d'adapter le nombre de ressources au système pour éviter qu'il ne cesse de fonctionner. Nous pouvons voir cela pour la 430ème seconde. En effet, la différence de prédiction était de presque 30% entre le CPU réel et le CPU prédit et c'est un pic du nombre de sessions interrompues qui a entraîné l'ajout d'un cœur et le retour à la normale du système. Dans ce cas aussi, notre outil semble donc fonctionner de manière relativement fiable, car la prédiction est majoritairement bonne et dans les zones où elle ne l'est pas le fait de suivre le nombre de sessions interrompues non volontairement permet de maintenir un système IMS fonctionnel.

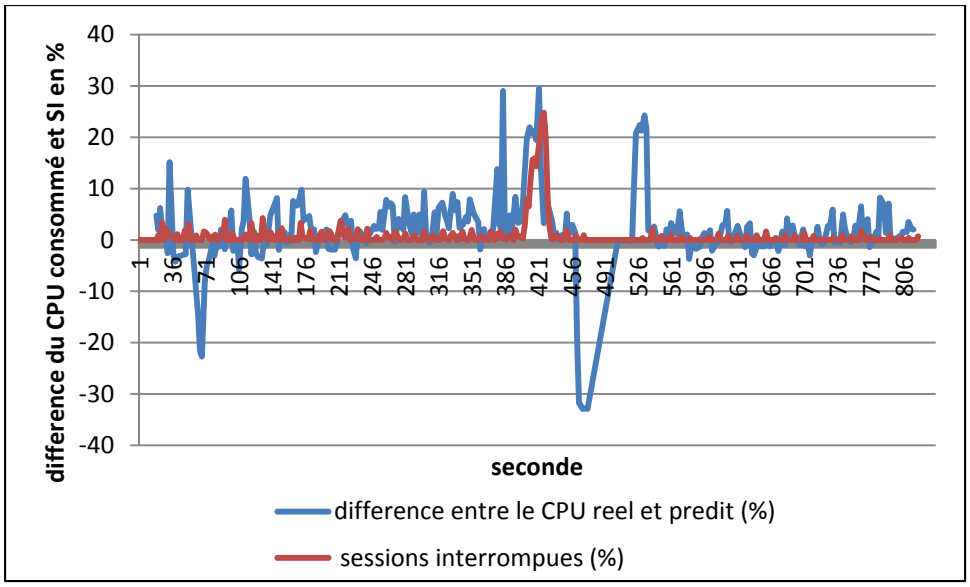

Figure 4.16 Erreur de prédiction et nombre de sessions interrompues de manière non souhaitée pour le jeu de test 4

Pour calculer les valeurs de la courbe bleue intitulée « difference entre le CPU reel et predit », nous avons appliqué la formule suivante :

$$difference\ entre\ le\ CPU\ reel\ et\ predit =$$
$$valeur\ relevee\ du\ CPU\ (reel) -\ valeur\ predite\ du\ CPU$$

Comme pour la section précédente, nous avons réalisé 100 simulations de ce jeu de test afin de tester la fiabilité de MAA. Nous obtenons alors les résultats suivants :

Tableau 4.3 Résultats des simulations du jeu de test 4

Pourcentage de simulations arrivées à terme	100%
Temps d'exécution moyen de l'algorithme de MAA	2.01 ms
Moyenne du nombre de fois où le système interrompt plus de 5% de sessions	12/400
Moyenne de la charge CPU consommée par le S-CSCF	40.6%

Nous remarquons qu'avec ce jeu de test aussi l'utilisation de l'outil peut être bénéfique. En effet, l'outil ne perturbe pas le système IMS dans son fonctionnement car nous constatons

que le jeu de test se termine dans 100% des cas et que le nombre de sessions interrompues n'est pas plus élevé quand notre outil supervise la gestion des ressources. Le faible temps d'exécution de l'algorithme nous permet de dire que malgré les calculs que doit faire l'outil chaque seconde, l'outil ne consomme pas beaucoup de ressources car 2 ms ne représentent que 3.33% d'une seconde qui est notre temps de référence pour la prédiction. Pour finir, MAA permet une charge moyenne du CPU à 40.6% dans ce cas. Le scénario non favorable à notre méthode de prédiction explique ce faible résultat qui reste cependant satisfaisant car notre CPU maximal est fixé à 70%.

4.5 Analyse des performances des algorithmes de prédiction SVR et (D)LS-SVM

Comme nous l'avons vu au cours de ce chapitre, l'optimisation du CPU consommé par le système IMS que nous utilisons dépend beaucoup du jeu de test utilisé lors de nos simulations. Nous avons alors voulu comparer les prédictions basées sur un seuil avec celles que nous aurions pu avoir grâce aux techniques de séparateurs à vaste marge présentées dans la revue de la littérature de ce mémoire et à la section 3.3.5. Pour ce faire, nous avons appliqué ces techniques pour la prédiction du CPU sur des simulations réalisées précédemment avec les jeux de test présentés à la section 4.2.

Nous avons exécuté le script Matlab implémenté selon l'algorithme XI.1 afin de comparer les différentes méthodes. En effet, l'algorithme permet de visualiser les prédictions du CPU consommé par le conteneur S-CSCF faites grâce aux méthodes SVR et DLS-SVM, mais aussi celles faites par notre outil MAA. Chaque groupe de figures, 4.17 à 4.22, 4.23 à 4.28, 4.29 à 4.34 et 4.35 à 4.40, regroupe cinq figures. La première des cinq figures illustre les prédictions obtenues du CPU consommé par le S-CSCF par chaque méthode et les valeurs relevées (réelles) (voir les figures 4.17, 4.23, 4.29 et 4.35). La seconde figure représente l'erreur faite pour les prédictions du CPU consommé par le S-CSCF (voir les figures 4.18, 4.24, 4.30 et 4.36). Les erreurs ont été calculées selon la méthode de l'erreur absolue moyenne (MAE) qui calcule la moyenne arithmétique des valeurs absolues de la distance entre la valeur relevée et la valeur prédite. Cette méthode est le plus souvent utilisée pour

comparer plusieurs méthodes comme nous cherchons ici à comparer les prévisions du CPU consommé faites par l'algorithme VI.1 implémenté dans MAA, SVR, LS-SVM et DLS-SVM. Les trois dernières figures montrent l'évolution des paramètres utilisés par les méthodes SVM mises en place (voir les figures 4.19 à 4.22, 4.25 à 4.28, 4.31 à 4.34 et 4.37 à 4.40).

Les figures 4.17 à 4.22 ont été réalisées avec les valeurs relevées lors d'une simulation du jeu de test 1, à savoir, une augmentation constante du cps jusqu'à ce que le système cesse de fonctionner.

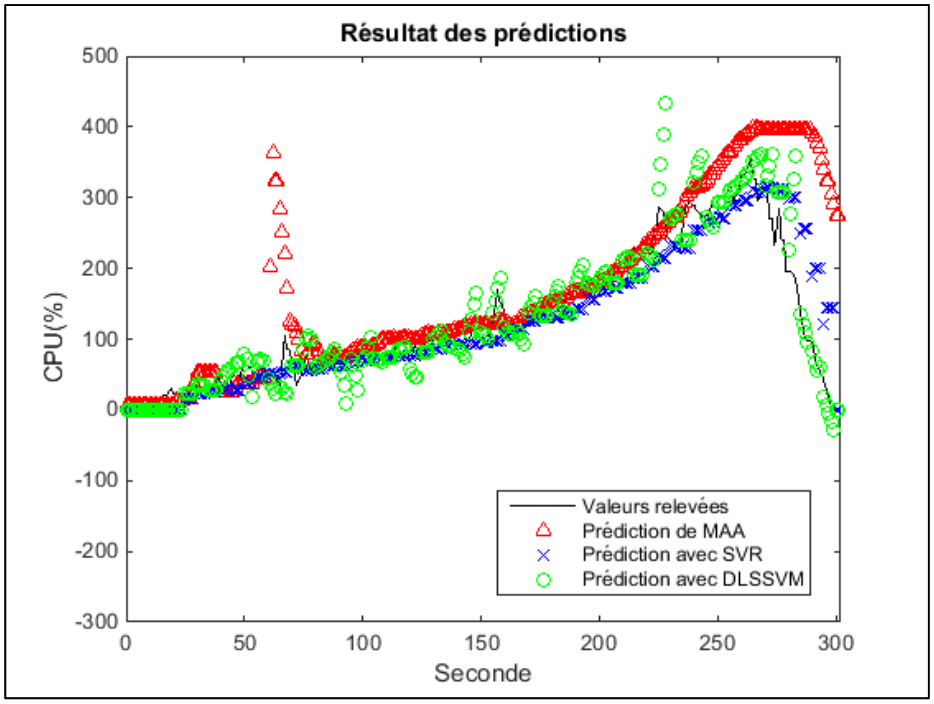

Figure 4.17 Prédiction du CPU consommé avec SVR, DLS-SVM et MAA pour le jeu 1

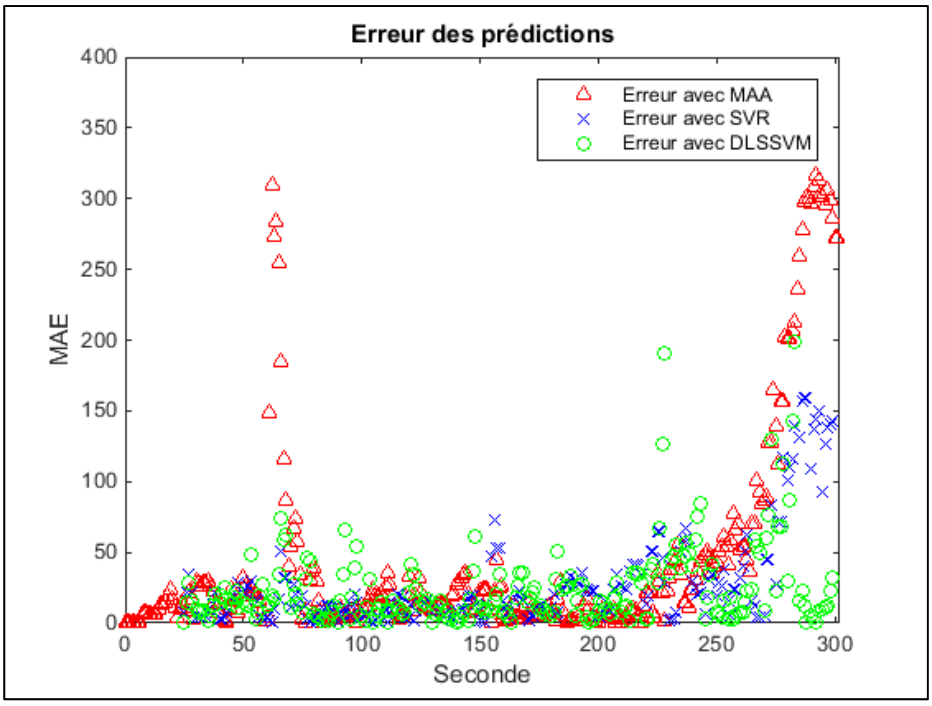

Figure 4.18 Erreur de la prédiction réalisée avec SVR, DLS-SVM et MAA pour le jeu 1

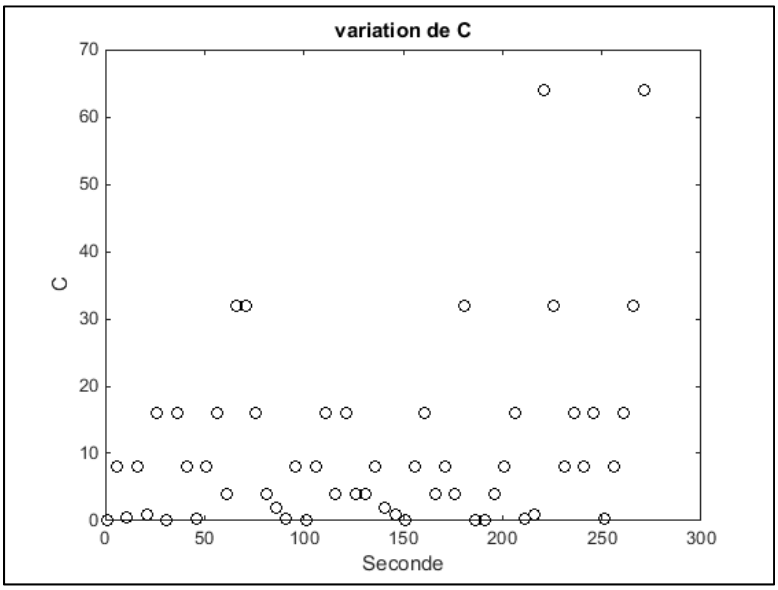

Figure 4.19 Variation du paramètre C de la méthode SVR pour le jeu 1

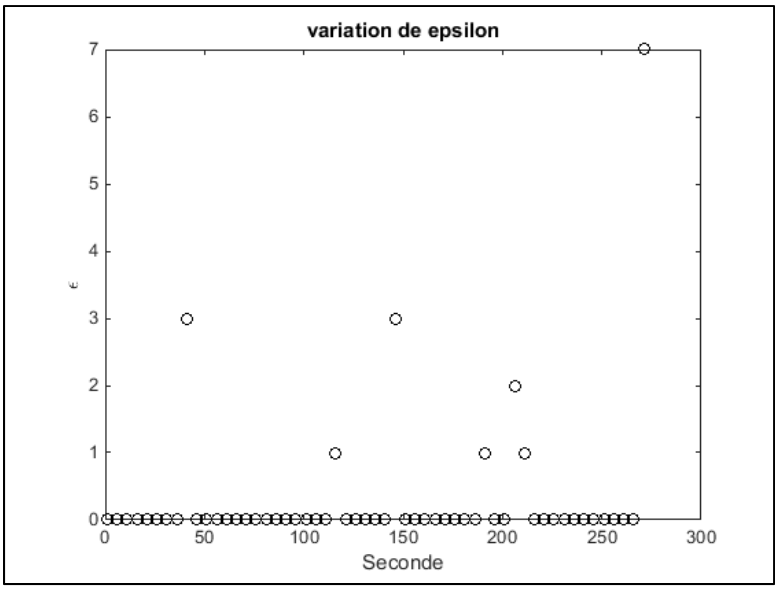

Figure 4.20 Variation du paramètre ε de la méthode SVR pour le jeu 1

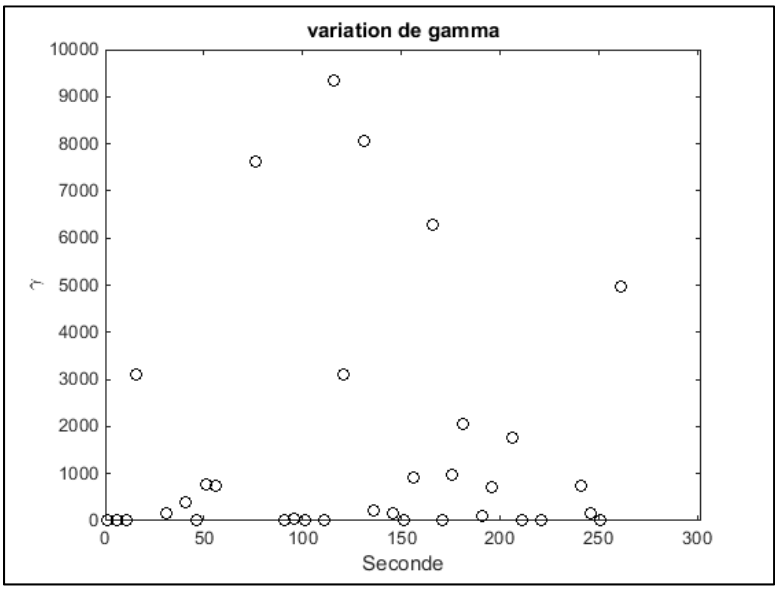

Figure 4.21 Variation du paramètre γ de la méthode DLS-SVM pour le jeu 1

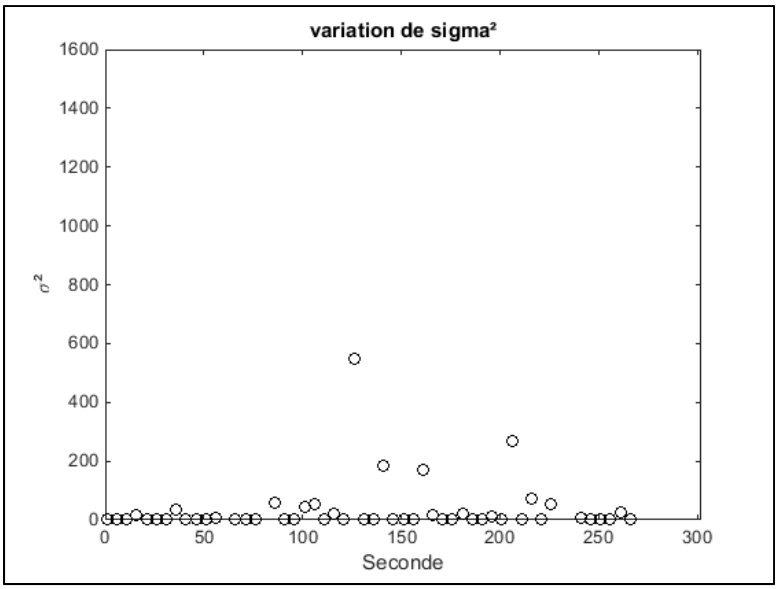

Figure 4.22 Variation du paramètre σ de la méthode DLS-SVM pour le jeu 1

Les figures 4.23 à 4.28 sont celles tracées à partir d'une simulation du jeu de test 2.

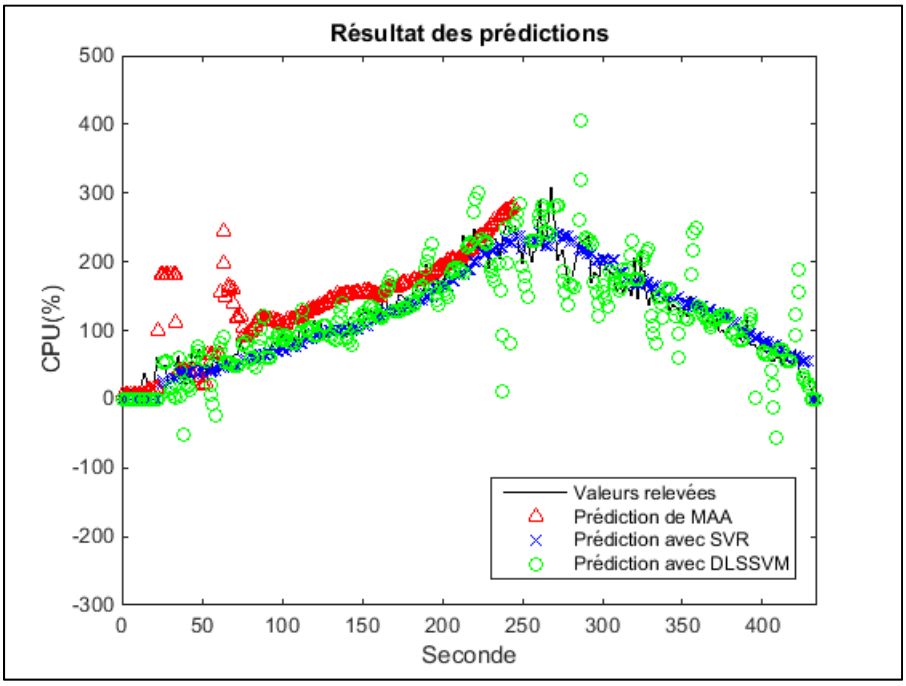

Figure 4.23 Prédiction du CPU consommé avec SVR, DLS-SVM et MAA pour le jeu 2

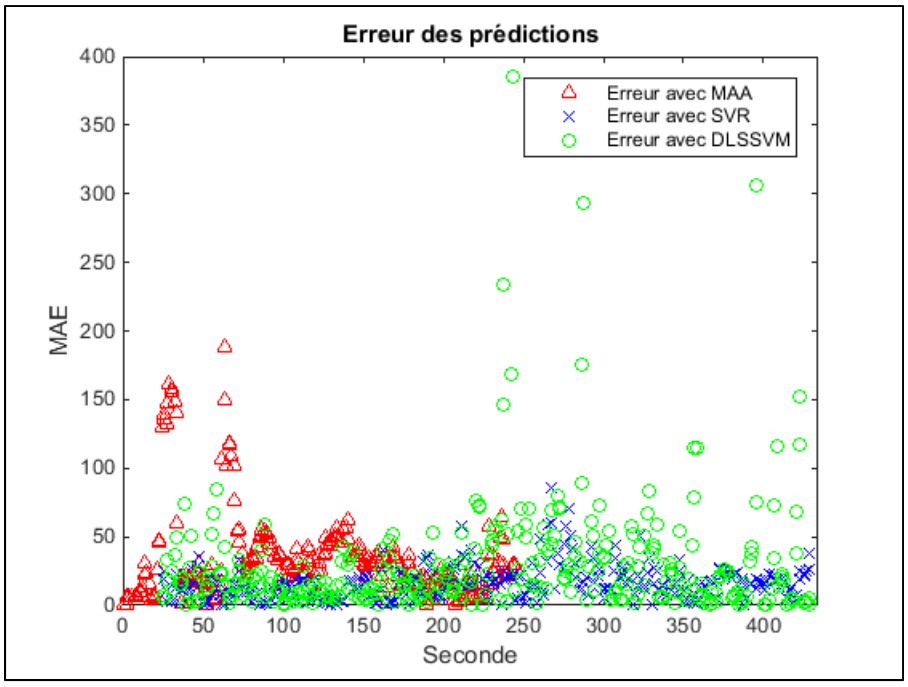

Figure 4.24 Erreur de la prédiction réalisée avec SVR, DLS-SVM et MAA pour le jeu 2

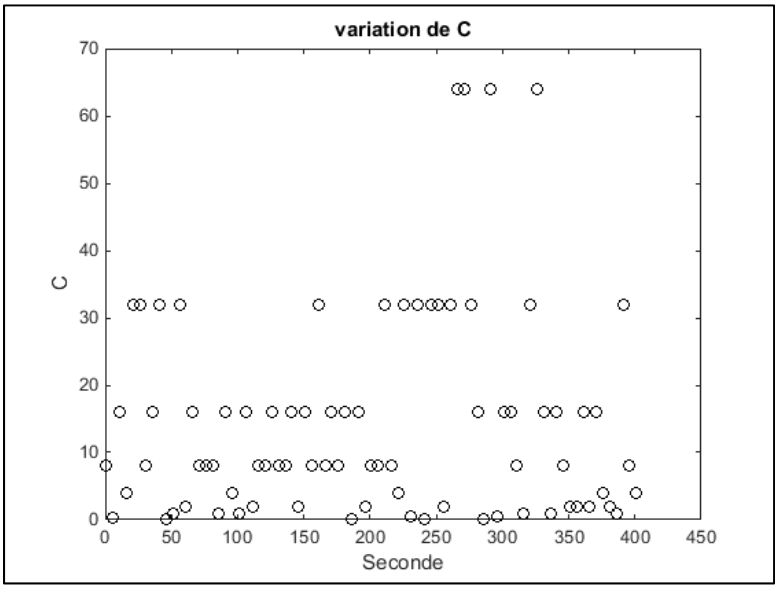

Figure 4.25 Variation du paramètre C de la méthode SVR pour le jeu 2

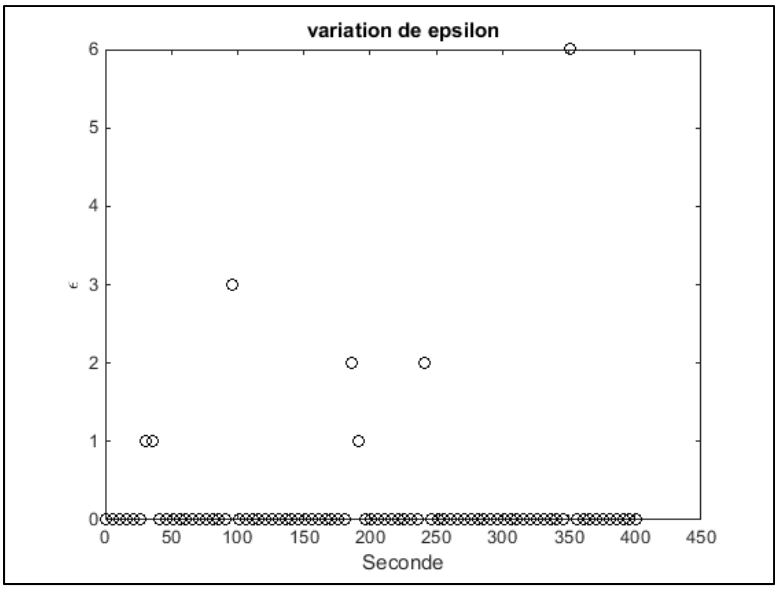

Figure 4.26 Variation du paramètre ε de la méthode SVR pour le jeu 2

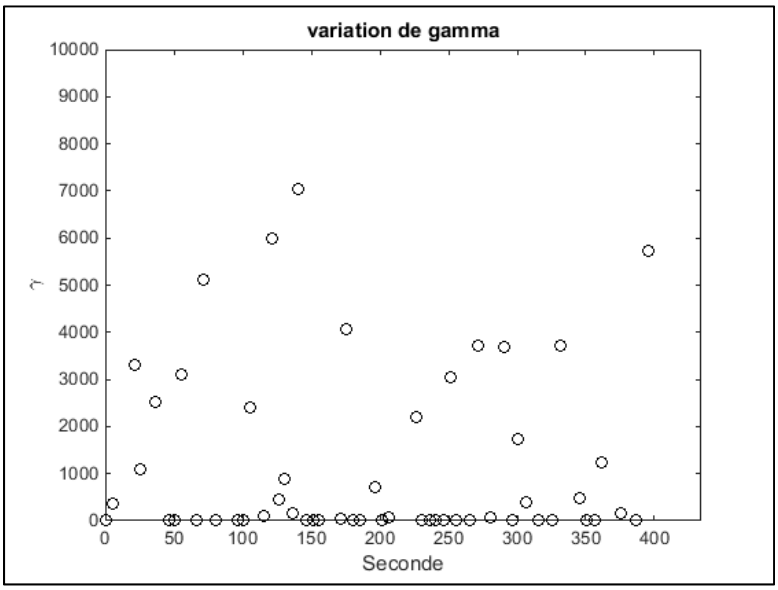

Figure 4.27 Variation du paramètre γ de la méthode DLS-SVM pour le jeu 2

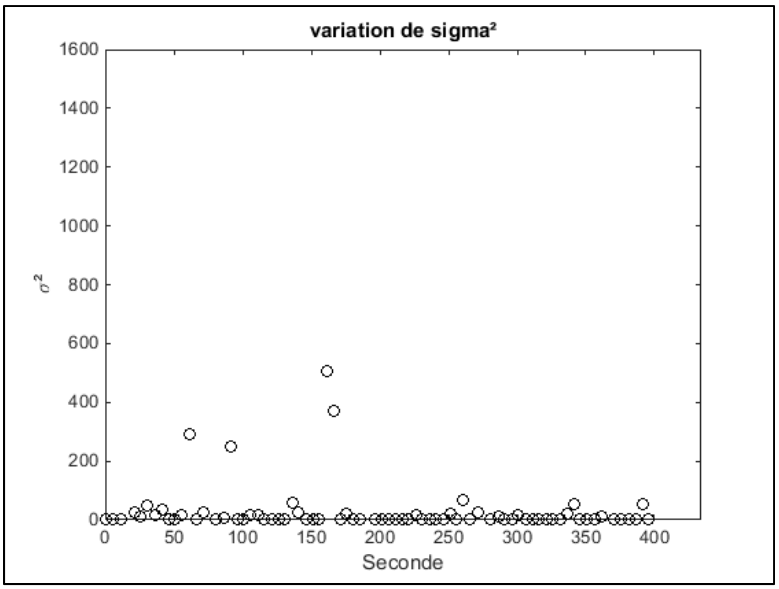

Figure 4.28 Variation du paramètre σ de la méthode DLS-SVM pour le jeu 2

Les figures 4.29 à 4.34 sont tracées à partir des résultats d'une simulation du jeu de test 3.

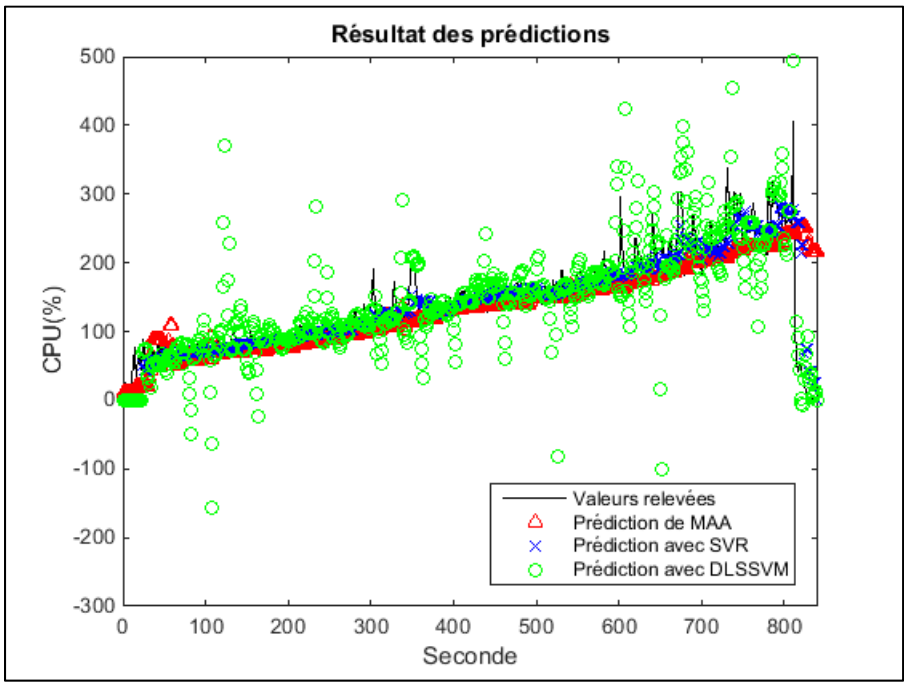

Figure 4.29 Prédiction du CPU consommé avec SVR, DLS-SVM et MAA pour le jeu 3

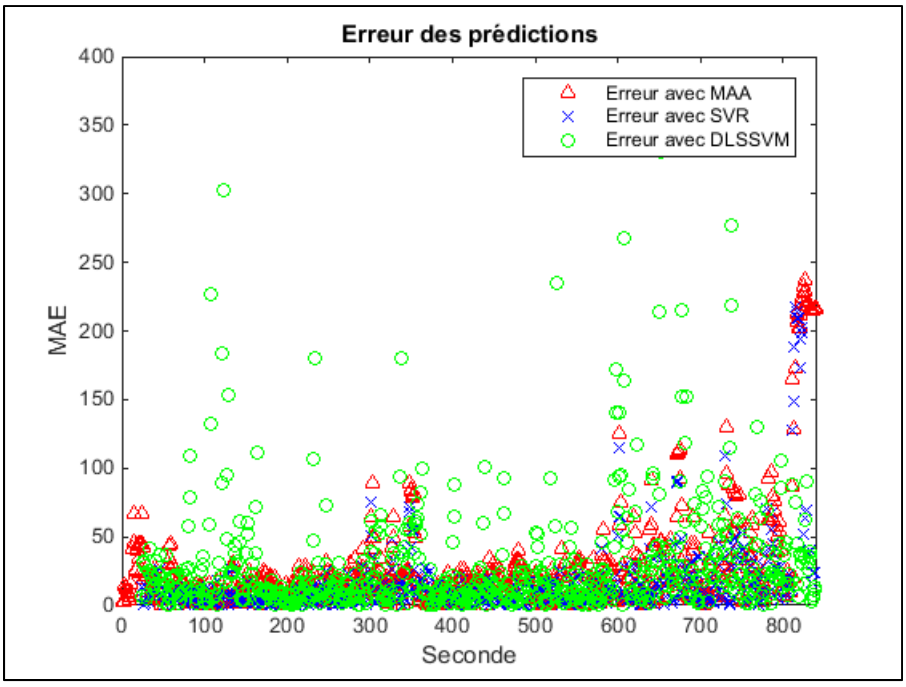

Figure 4.30 Erreur de la prédiction réalisée avec SVR, DLS-SVM et MAA pour le jeu 3

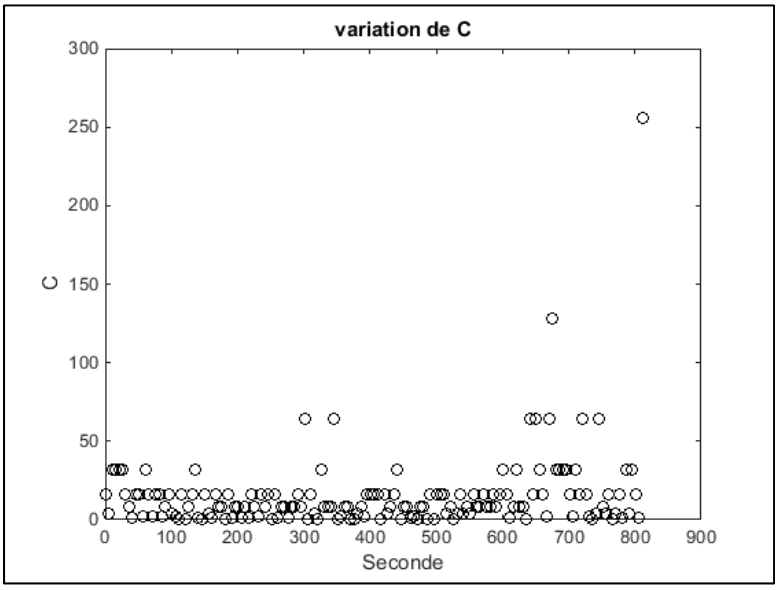

Figure 4.31 Variation du paramètre C de la méthode SVR pour le jeu 3

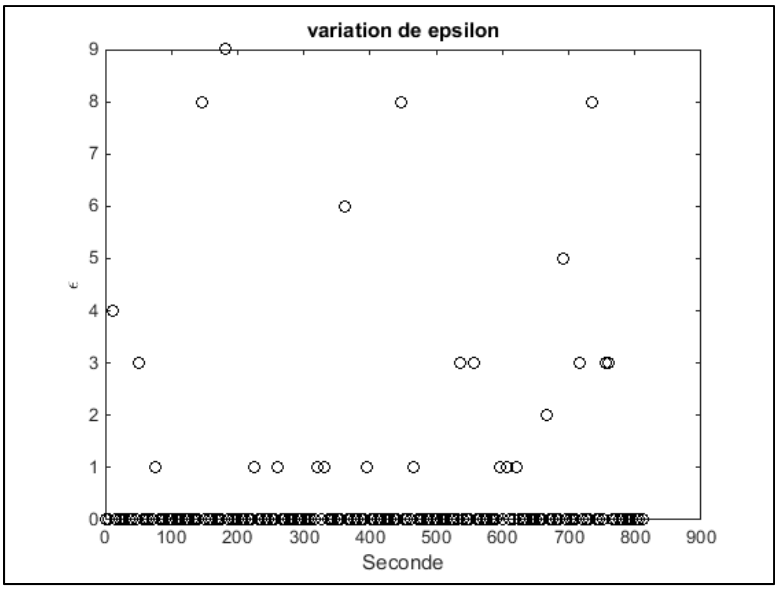

Figure 4.32 Variation du paramètre ε de la méthode SVR pour le jeu 3

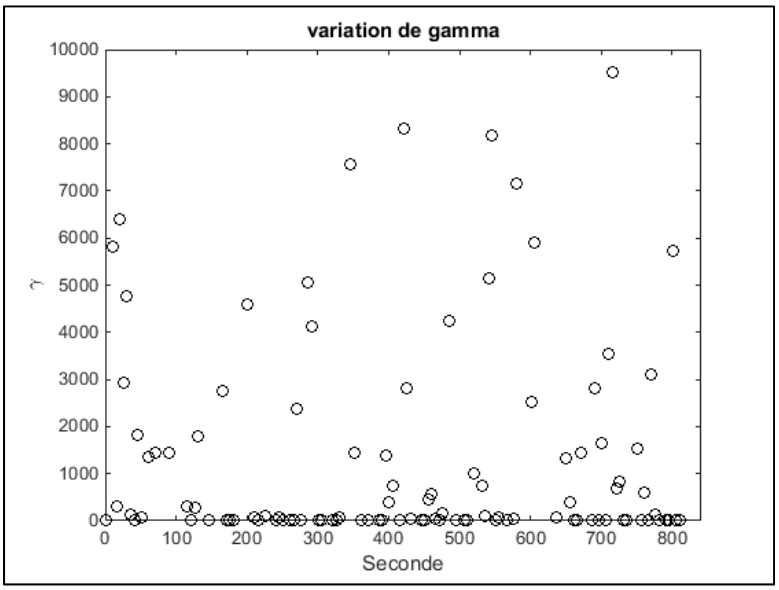

Figure 4.33 Variation du paramètre γ de la méthode DLS-SVM pour le jeu 3

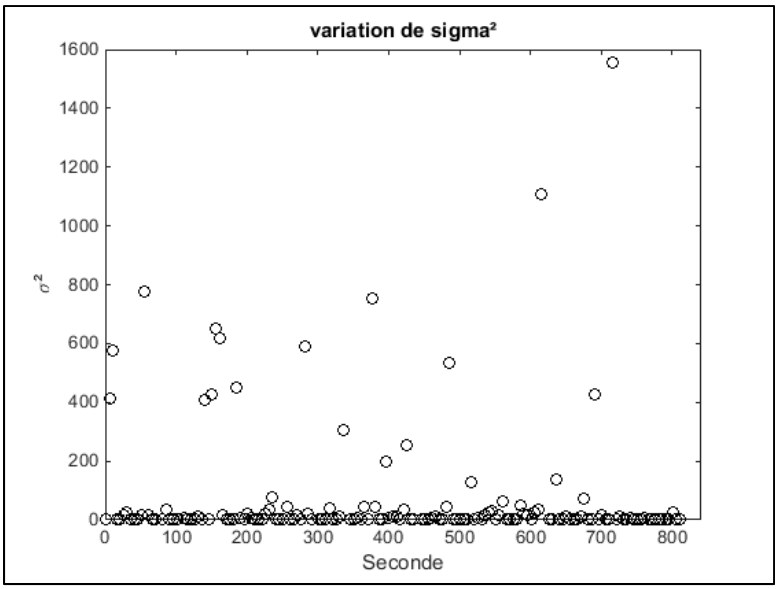

Figure 4.34 Variation du paramètre σ de la méthode DLS-SVM pour le jeu 3

Les résultats présentés sur les figures ci-dessus montrent que la méthode DLS-SVM est très sensible aux variations des relevés du CPU. Cela est un inconvénient, car en s'appuyant sur une montée ou une descente brusque de la courbe du CPU relevé, cette méthode prédit des valeurs trop hautes ou trop basses. La variation peut pourtant n'être que de courte durée mais la prédiction est malheureusement fausse. Cependant, cette caractéristique de la méthode fait que cette dernière est la plus rapide à réagir à un changement de pente soudain de CPU. En effet, nous pouvons lire sur la figure 4.17 que la méthode DLS-SVM réagit environ 6 secondes après le changement de pente du CPU alors que la méthode SVR réagit après environ 13 secondes et MAA après environ 23 secondes. Nous avons ensuite voulu vérifier que la méthode DLS-SVM est plus rapide à réagir que la méthode SVR qui est elle-même plus rapide que MAA avec le jeu de test 4. Les figures 4.35 à 4.40 illustrent les résultats obtenus.

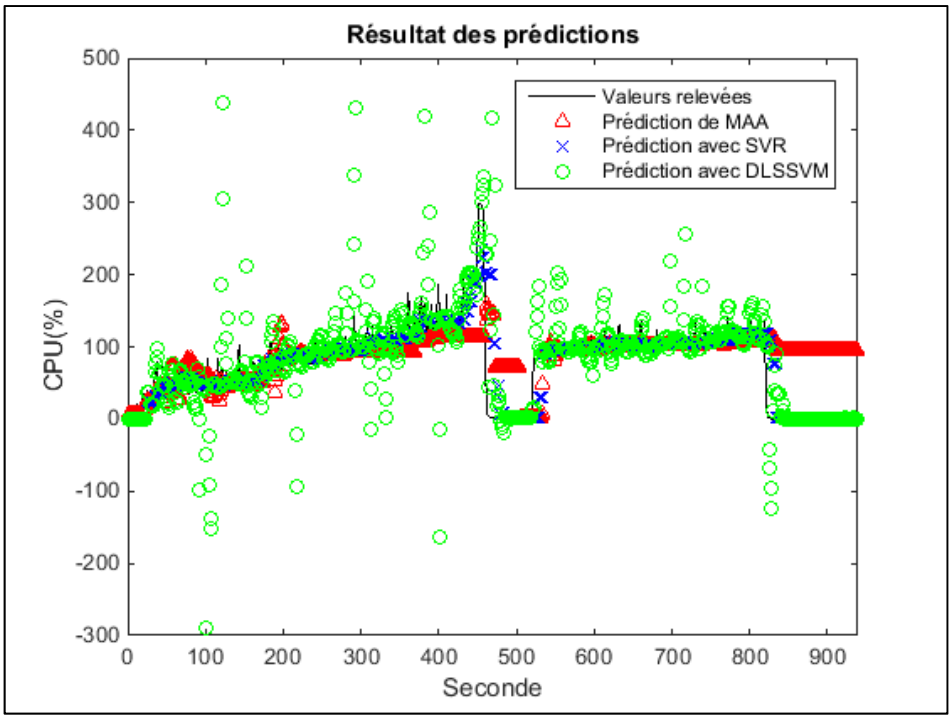

Figure 4.35 Prédiction du CPU consommé avec SVR, DLS-SVM et MAA pour le jeu 4

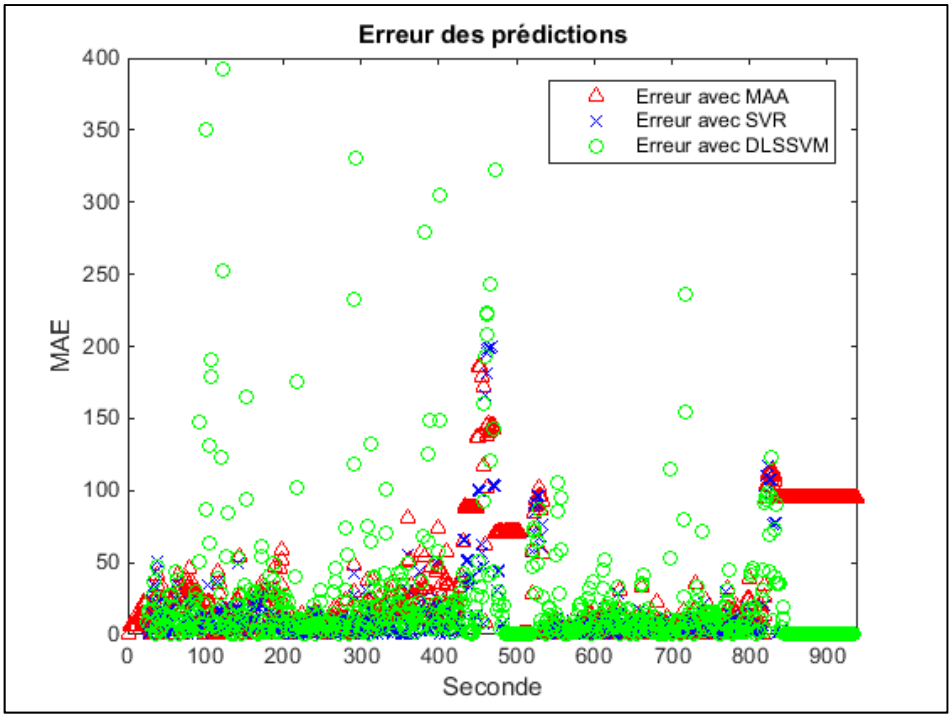

Figure 4.36 Erreur de la prédiction réalisée avec SVR, DLS-SVM et MAA pour le jeu 4

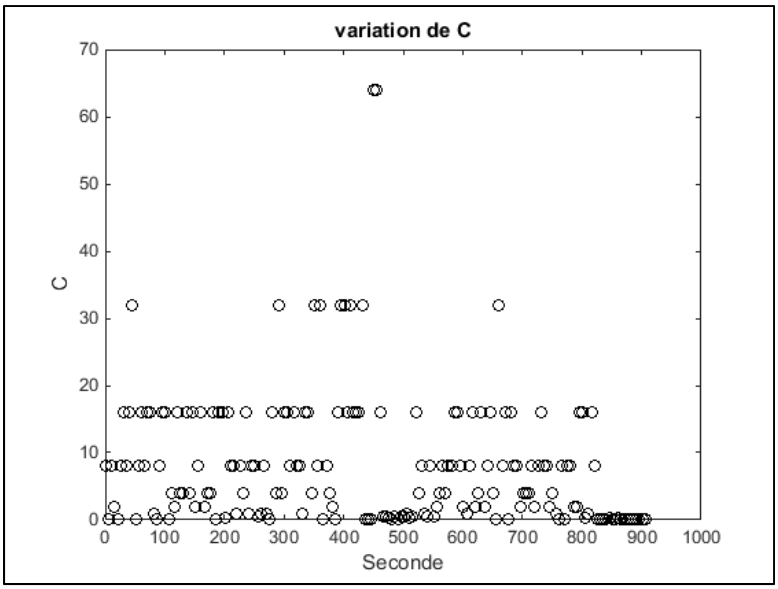

Figure 4.37 Variation du paramètre C de la méthode SVR pour le jeu 4

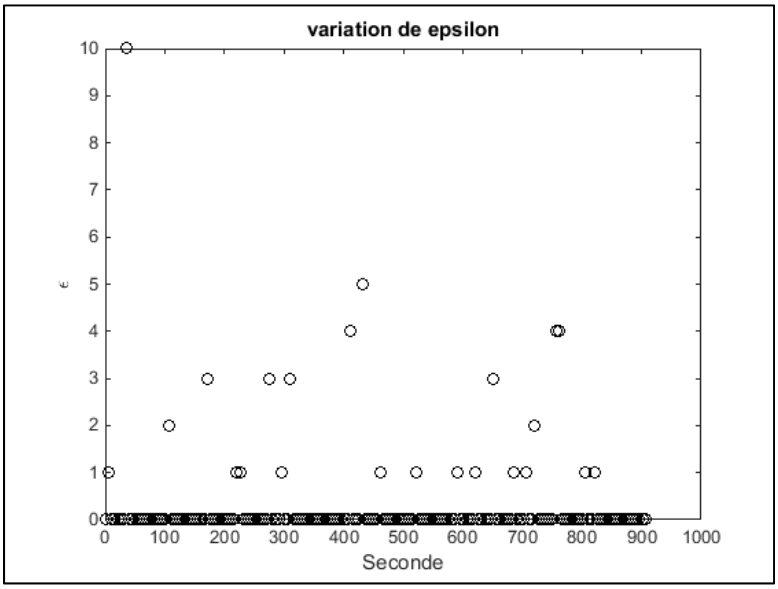

Figure 4.38 Variation du paramètre ε de la méthode SVR pour le jeu 4

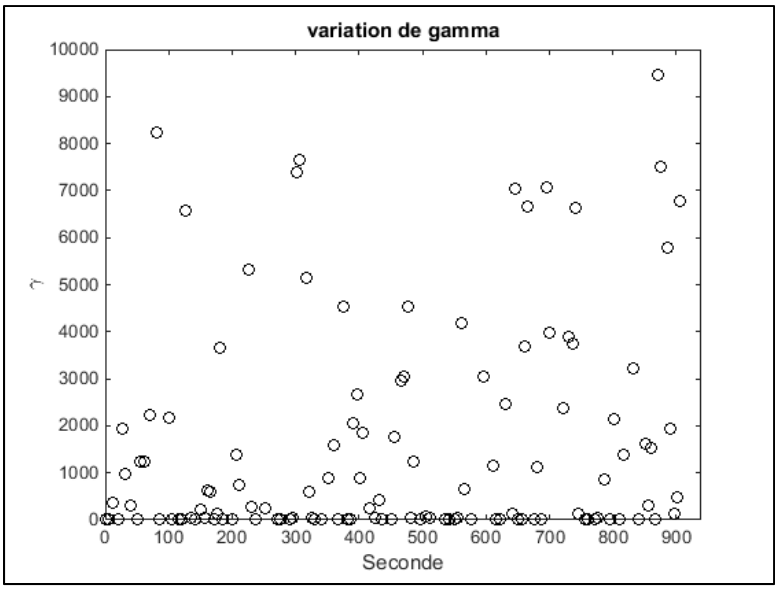

Figure 4.39 Variation du paramètre γ de la méthode DLS-SVM pour le jeu 4

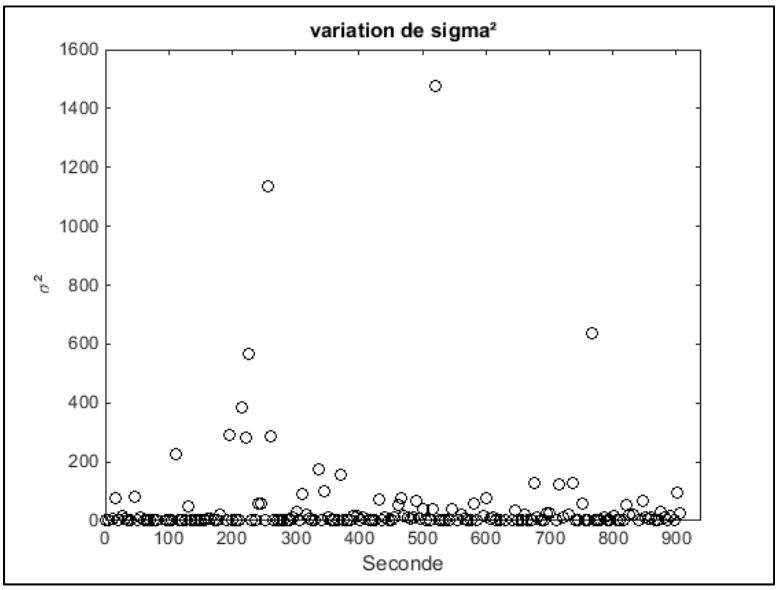

Figure 4.40 Variation du paramètre σ de la méthode DLS-SVM pour le jeu 4

Dans ce cas aussi, les méthodes SVR et DLS-SVM semblent fiables. Contrairement à MAA, elles réagissent rapidement à tous les brusques changements de pente du CPU. Cependant, pour ce jeu de test comme pour les précédents, la méthode DLS-SVM a le défaut de réagir à toutes les oscillations du CPU ce qui entraîne des erreurs de prédiction. Afin de nous assurer

que les erreurs de prédiction viennent, comme les figures le laissent penser, des oscillations des données d'entrée, nous avons réalisé deux tests supplémentaires avec cette méthode. Le premier consiste à refaire les mêmes tests, mais en appliquant un filtre de Kalman (Kalman 1960) sur les relevées afin d'éliminer le bruit et donc d'atténuer les oscillations du CPU. Le second test supplémentaire consiste à appliquer la méthode LS-SVM (décrite à la section 3.3.5) aux jeux de tests. Cette dernière utilise les mêmes algorithmes de régression que DLS-SVM mais prend en entrée toutes les mesures du CPU récoltées. Les figures 4.41, 4.42 et 4.43 illustrent les résultats obtenus pour le jeu de tests 2.

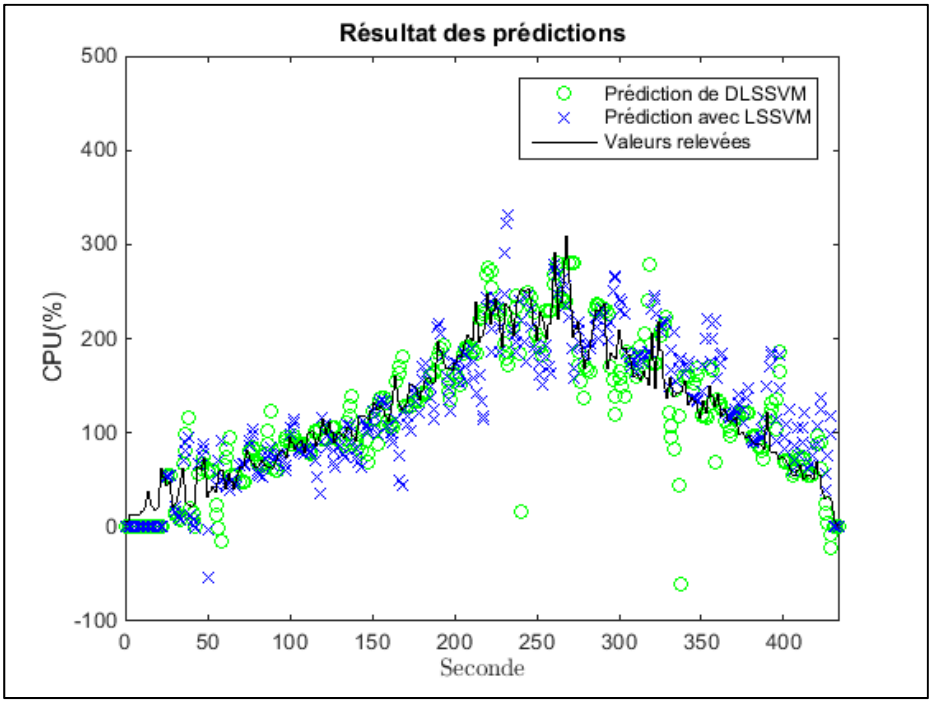

Figure 4.41 Comparaison des méthodes DLS-SVM et LS-SVM pour la prédiction du CPU pour le jeu 2

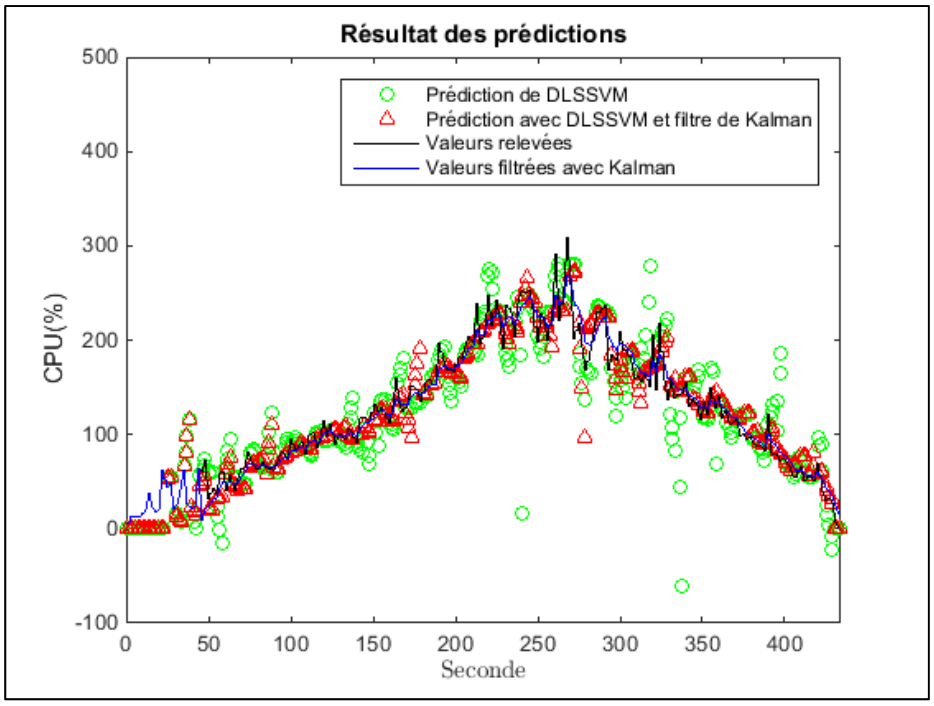

Figure 4.42 Comparaison de la méthode DLS-SVM avec et sans filtre de Kalman pour la prédiction du CPU pour le jeu 2

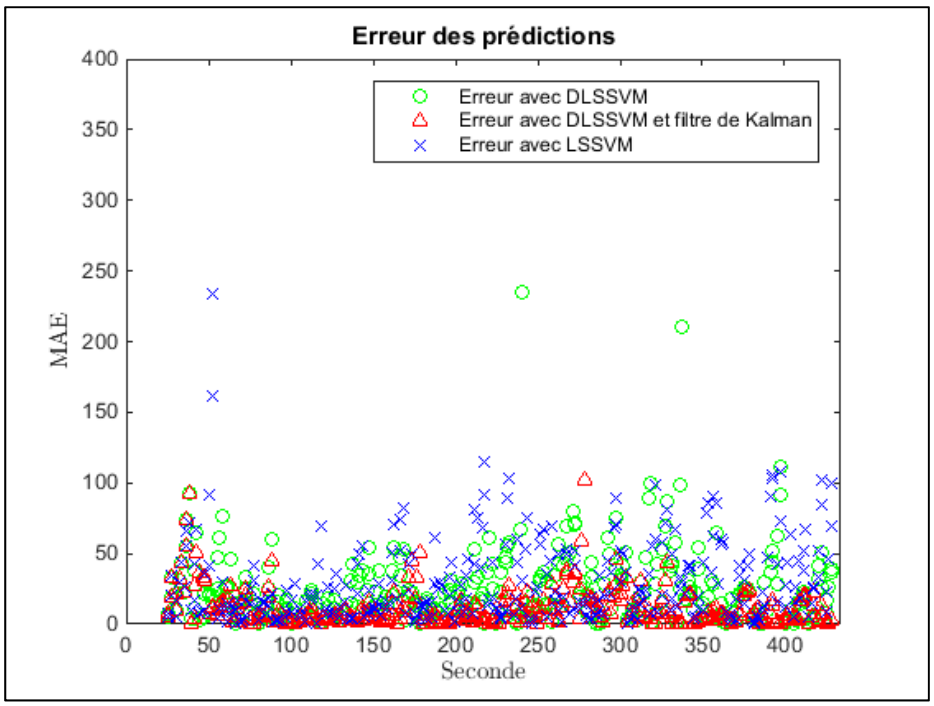

Figure 4.43 Erreur de la prédiction réalisée avec DLS-SVM, DLS-SVM avec le filtre de Kalman et LS-SVM pour le jeu 2

Sur la figure 4.41, nous pouvons constater que pour le jeu de test 2, les méthodes DLS-SVM et LS-SVM prédisent des résultats proches pour la prédiction du CPU consommé par le S-CSCF. Aucune des deux méthodes ne semble, sur cette figure, être meilleure ou plus rapide. Sur la figure 4.42, nous constatons qu'appliquer un filtre de Kalman sur les données en entrée avant d'appliquer la méthode DLS-SVM peut-être bénéfique à la prédiction du CPU. En effet, les valeurs récupérées en sortie du test qui lie le filtre de Kalman et la méthode DLS-SVM sont moins éparpillées autour de la courbe représentant les valeurs réelles que les valeurs calculées par la méthode DLS-SVM sur le jeu de données originel en entrée. Pour finir, en analysant la figure 4.43, nous remarquons que pour le jeu de test 2, la technique qui semble avoir la plus faible erreur à tous les moments du test est celle qui lie le filtre de Kalman et la méthode DLS-SVM.

Afin de comparer toutes les méthodes analysées, nous avons calculé la moyenne des moyennes de l'erreur absolue obtenues lors de chaque prédiction. Les résultats ont été regroupés dans le tableau 4.4.

Tableau 4.4 Moyenne des MAE pour des méthodes de prédiction du CPU

	SVR	DLS-SVM	DLS-SVM + kalman	LS-SVM	MAA
jeu 1	28.28	26.14	**10.53**	24.94	46.72
jeu 2	16.31	29.10	**11.31**	29.88	34.22
jeu 3	**16.10**	35.01	17.14	20.73	26.55
jeu 4	**13.37**	29.98	14.06	20.26	30.36

Selon les résultats obtenus et avec les jeux de test que nous avons construit, la méthode DLS-SVM avec des données nettoyées grâce au filtre de Kalman est la plus fiable pour prédire le CPU. En effet, c'est pour cette méthode que nous avons la plus faible moyenne des MAE avec les jeux 1 et 2. Avec les jeux 3 et 4, la méthode SVR est légèrement meilleure que la méthode DLS-SVM avec Kalman, mais pour les jeux 1 et 2 elle est largement moins bonne. Cependant, si les données d'entrée ne sont pas nettoyées ou que trop de bruit persiste, la

méthode perd beaucoup en efficacité et celle qui offre une meilleure moyenne MAE est alors la méthode SVR. D'après la littérature, en nous appuyant sur l'article de J. Ye et T. Xiong (Ye and Xiong 2007) et celui de R. Adhikari (Adhikari and Agrawal 2013) ainsi que sur les travaux de B. Üstün (Üstün 2003), nous ne pouvons pas conclure qu'une méthode SVM particulière apportera une meilleure prédiction que les autres. Aucune de ces méthodes ne peut donc être privilégiée, car la fiabilité varie en fonction des données en entrée. En effet, ces données influent sur la qualité de la prédiction faite par les méthodes SVM, car elles permettent le choix des paramètres (C, ε, γ et σ) et la détermination du modèle qui sera appliqué pour la prédiction.

Pour tester, sur notre système IMS virtualisé, les méthodes de régression de SVM mentionnées ici, il faudrait les implémenter en Python ou modifier MAA pour qu'il interagisse avec Matlab. L'implémentation de la méthode SVR a déjà été réalisée et est disponible en Python grâce à la bibliothèque Scikit-learn (Pedregosa et al. 2011). Cependant, cette bibliothèque, tout comme les fonctions Matlab implémentées pour nos tests, ne pourrait pas être directement utilisée, car elles mettent plus de 5 secondes à s'exécuter. La documentation de la bibliothèque Scikit-learn indique, par exemple, que la fonction pour déterminer les paramètres optimaux à utiliser avec SVR se déroule en 6.52 secondes. De plus, les scripts Matlab que nous avons développés ne prédisent le CPU qu'avec 5 secondes d'avance et nous avons besoin, comme nous l'expliquons à la section 3.2.4, d'une prédiction avec 10 secondes d'avance. Il faudrait donc prédire plus de valeurs et cela allongerait encore le temps d'exécution de nos scripts. Un ordinateur plus puissant que le notre qui possède 4 cœurs et 8 Go de RAM, permettrait peut-être de suffisamment accélérer l'exécution des scripts Matlab afin de pouvoir les utiliser dans notre outil de gestion dynamique et proactive des ressources CPU du conteneur S-CSCF. L'autre idée serait de réduire suffisamment l'intervalle de recherche des paramètres afin d'accélérer la phase de configuration qui est la plus lente de l'algorithme. Ces méthodes de régression sont donc fiables mais leur exécution est actuellement trop lente pour qu'elles soient utilisées avec MAA sur notre nuage informatique privé. Elles semblent cependant prometteuses pour la prédiction du CPU en temps réel.

4.6 Conclusion du chapitre

Au cours de ce chapitre, nous avons décrit la plateforme utilisée et les jeux de test construits pour tester les algorithmes proposés au chapitre 3. Tous ces tests nous permettent de conclure que les méthodes SVM analysées dans ce mémoire fournissent une meilleure prédiction du CPU consommé par le S-CSCF que l'algorithme VI.1 implémenté dans MAA. Cependant, ces méthodes ne peuvent actuellement pas être utilisées dans notre outil, car les ressources physiques de notre nuage privé ne sont pas suffisantes pour que les méthodes SVM soient assez rapides. En effet, comme nous l'avons expliqué au chapitre 3, nous avons besoin de prédire le CPU utile au conteneur S-CSCF en temps réel et avec 10 secondes d'avance, mais lors de nos tests nous avons remarqué que les méthodes SVM testées ne le permettaient pas. Avec la puissance de calcul de notre nuage privé, nous devons attendre plusieurs minutes pour obtenir 5 secondes de prédiction du CPU consommé. L'outil MAA ne sera donc, pour le moment, pas modifié et il continuera d'utiliser l'algorithme VI.1 pour prédire la consommation du S-CSCF en CPU. Bien que moins précise que les méthodes SVM, la prédiction avec MAA permet tout de même d'optimiser la consommation du CPU par le S-CSCF. En effet, le S-CSCF consomme en moyenne 46.5% du CPU disponible pour les trois premiers jeux de test proposés quand MAA est actif contre 32.8% lorsque MAA est inactif et que 4 cœurs sont en permanence disponibles pour le conteneur. Cela s'explique par le fait que MAA adapte le nombre de cœurs mis à la disposition du S-CSCF en fonction des prédictions qu'il fait du CPU consommé. Cette optimisation de l'utilisation du CPU par le conteneur S-CSCF est utile car elle permet de réduire le gaspillage des ressources CPU fait par ce même conteneur et donc par le système IMS virtualisé.

CONCLUSION

Le gaspillage est un fléau de notre société. Dans ce mémoire, nous nous sommes intéressés à diminuer ce gaspillage pour des systèmes IMS virtualisés. Pour de tels systèmes, le gaspillage vient principalement d'un excès de ressources physiques disponibles. Nous avons donc cherché à gérer de manière dynamique et proactive les ressources physiques d'un système IMS virtualisé afin de lui rendre disponibles uniquement les ressources physiques nécessaires à son bon fonctionnement.

Dans les systèmes IMS virtualisés, nous avons constaté que les conteneurs de type S-CSCF représentent un goulot d'étranglement puisqu'ils interagissent avec différents modules IMS. Ainsi, la gestion du CPU pour le S-CSCF est très critique alors que la variation de la consommation du CPU par les autres conteneurs est négligeable. La gestion du nombre de cœurs à mettre à disposition des conteneurs de type S-CSCF est donc primordiale. Elle est cependant délicate car les besoins en CPU des S-CSCF varient rapidement avec le nombre d'appels par seconde.

Dans le cadre de cette recherche, nous avons focalisé notre travail sur la gestion dynamique et proactive des ressources CPU du conteneur S-CSCF du système IMS virtualisé. L'objectif était alors de développer un outil qui permet de minimiser les ressources CPU fournies au conteneur S-CSCF tout en garantissant le minimum de violation du SLA exprimé en termes de perte de sessions SIP. Dans ce contexte, nous avons développé MAA qui permet de télémétrer le système IMS virtualisé, d'analyser les valeurs relevées, de prédire les besoins en CPU du conteneur de type S-CSCF et d'adapter ses ressources en CPU avec le minimum de perte de sessions SIP. Pour finir, dans le cas où l'outil développé ne peut pas agir pour satisfaire les besoins du système IMS virtualisé, il prévient le gestionnaire du système de la fin imminente de celui-ci. Une fin prématurée du système IMS peut-être dû à un manque de ressources CPU du nuage informatique privé pour satisfaire les besoins du S-CSCF ou à un arrêt d'un conteneur de type P-CSCF. La cause peut aussi être inconnue et dans ce cas l'outil

détecte une défaillance du système lorsqu'il observe une forte croissance du nombre de sessions SIP interrompues non volontairement.

Nos principales contributions ont été faites pour le module de prédiction de MAA. Ce module prédit la consommation en CPU du S-CSCF. Pour implémenter ce module, nous avons commencé par analyser la consommation de CPU en fonction du nombre d'appels par seconde. Durant cette analyse, nous avons remarqué que la fonction qui lie le CPU au taux d'arrivée des appels peut être caractérisée par une courbe de tendance exponentielle. Nous avons alors proposé deux algorithmes de prédiction qui calculent les paramètres d'une fonction exponentielle pour que cette dernière représente le plus fidèlement possible les dernières valeurs enregistrées du CPU consommé par le S-CSCF. Ces algorithmes permettent à MAA d'adapter le nombre de cœurs CPU alloués au S-CSCF en fonction de la prédiction réalisée. L'inconvénient du premier algorithme est qu'il est gourmand en ressources CPU de la machine hôte qui héberge le S-CSCF. Ceci est un problème car le CPU consommé par MAA pour les calculs est du CPU qui n'est pas à la disposition du conteneur en cas de besoin. L'inconvénient du deuxième algorithme est qu'il nécessite une phase de configuration qui est très critique. En effet, si un problème survient pendant cette phase alors des paramètres aberrants pourraient être enregistrés et utilisés par la suite. Si cette erreur avait des répercussions sur la qualité de service du système, le SLA pourrait être violé.

Nous avons ensuite souhaité améliorer la qualité de prédiction du CPU consommé par le S-CSCF. La conséquence positive d'une meilleure prédiction serait une diminution du nombre de sessions interrompues. En effet, avec les algorithmes précédemment proposés, si une erreur de prédiction ne permet pas de satisfaire le S-CSCF en termes de ressources CPU alors le suivi du nombre de sessions interrompues permet d'allouer des cœurs CPU au conteneur. Nous avons ajouté cette règle pour l'ajout d'un cœur à la liste de ceux utilisables par le S-CSCF, car nous avons constaté qu'une augmentation du nombre de sessions interrompues était généralement liée à un manque de ressources CPU du S-CSCF. Malgré cette règle de sécurité pour satisfaire le conteneur en ressources CPU et faire redescendre le nombre de sessions interrompues chaque seconde, il n'est pas bon d'y avoir recours car elle implique

une augmentation du nombre de sessions interrompues et cela pourrait entraîner une violation du SLA. Pour améliorer la prédiction de la consommation du CPU par le S-CSCF, nous avons proposé d'utiliser les méthodes de régressions des SVM. Bien que ces méthodes soient très utilisées dans de nombreux domaines, elles ne sont, à notre connaissance, pas encore souvent appliquées à la prédiction de la consommation de ressources physiques par un module d'un système virtualisé. Après de nombreux tests avec les méthodes SVR, LS-SVM et DLS-SVM, il nous est apparu que la meilleure méthode était dans notre cas la méthode SVR. Cependant, si nous filtrons avec un filtre de Kalman les données d'entrée fournies aux différentes méthodes pour la prédiction du CPU consommé par S-CSCF, nous constatons que la méthode DLS-SVM est globalement la plus fiable. En effet, dans deux cas sur quatre elle est la meilleure, loin devant SVR et les autres méthodes. Dans les deux autres cas, elle est légèrement devancée par la méthode SVR. La fiabilité est définie par l'erreur moyenne entre les valeurs relevées et celles prédites. Dans tous les cas, ces méthodes sont plus fiables que l'algorithme proposé précédemment pour la prédiction du CPU consommé par le S-CSCF. L'inconvénient de ces dernières est leur temps d'exécution. En effet, la puissance de calcul de notre nuage privé ne permet pas une prédiction en temps réel. Il nous est donc impossible d'utiliser ces méthodes avec notre outil MAA.

Comme travaux futurs, nous proposons de travailler sur l'optimisation du script développé pour la prédiction de la consommation du CPU par le S-CSCF avec la méthode DLS-SVM et le filtre de Kalman. Pour cela, nous pensons qu'il serait intéressant de chercher à réduire les ensembles de valeurs possibles pour les paramètres de la méthode. En effet, la phase la plus longue lors de l'application de ces méthodes des SVM est celle qui teste tous les paramètres possibles afin de déterminer ceux qui permettront d'obtenir le meilleur modèle pour la prédiction de la consommation du CPU par le S-CSCF.

ANNEXE I

L'OUTIL DE TÉLÉMÉTRIE DES RESSOURCES AU NIVEAU SERVICE

Pour télémétrer le système IMS virtualisé sur lequel nous travaillons, nous avons besoin de monitorer les ressources au niveau service. Les valeurs que nous voulons relever sont le temps de latence entre les différents composants du système IMS et le nombre de sessions SIP reçues par le système IMS. Pour faire cela, un outil adapté, présenté par G. Combs (Combs), est TShark. TShark nous permet de récupérer tous les messages échangés entre les différents composants CSCF du système IMS. Ainsi nous pouvons calculer le temps entre deux messages reçus par un module CSCF pour mesurer la latence et la fréquence des messages envoyés entre deux modules.

Le script suivant nous permet de récupérer tous les messages échangés par les composants CSCF du système IMS virtualisé.

```
#!/usr/bin/env bash
# ${1} is IP adress from container
# ${2} is name from container
YEAR=$(date +%Y);
MONTH=$(date +%m)
DAY=$(date +%d)
HOUR=$(date +%H)
MINUTE=$(date +%M)
SECONDE=$(date +%S)
echo "$YEAR:$MONTH:$DAY:$HOUR:$MINUTE:$SECONDE" >> ../result/time_${2}.txt
{
ssh root@${1} "tshark -B 80 -i eth0 -l -Tfields -Eseparator=, -e frame.time_relative -e sip.from.user -e sip.Method -e sip.Status-Line -e ip.src | sed -e 's/,,/,/' |sed -e 's/SIP\/2.0 100 trying -- your call is important to us/trying/'| sed -e 's/SIP\/2.0 180 Ringing/Ringing/'"> ../result/datas_${2}.txt
} &> /dev/null
```

Il faut ensuite parcourir le fichier de tous les messages échangés entre les modules CSCF pour relever les valeurs voulues. Les métriques relevées sont le délai d'établissement d'un appel, le temps de réponse des conteneurs de type P-CSCF et S-CSCF ainsi que le nombre de messages RINGING et BYE reçu par le système toutes les secondes. Le problème de l'outil TShark est le temps nécessaire à récupérer tous les messages échangés entre les modules

CSCF. Il faut environ trois minutes pour récupérer une seconde de simulation pour 1000 appels par seconde. Nos simulations durent en moyenne cinq minutes, débutent à 150 appels par seconde et peuvent atteindre les 1400 appels par seconde. L'outil TShark n'est donc pas utilisable dans notre cas.

ANNEXE II

DIAGRAMME DE SÉQUENCE DU MODULE DE SURVEILLANCE DU LOGICIEL DÉVELOPPÉ

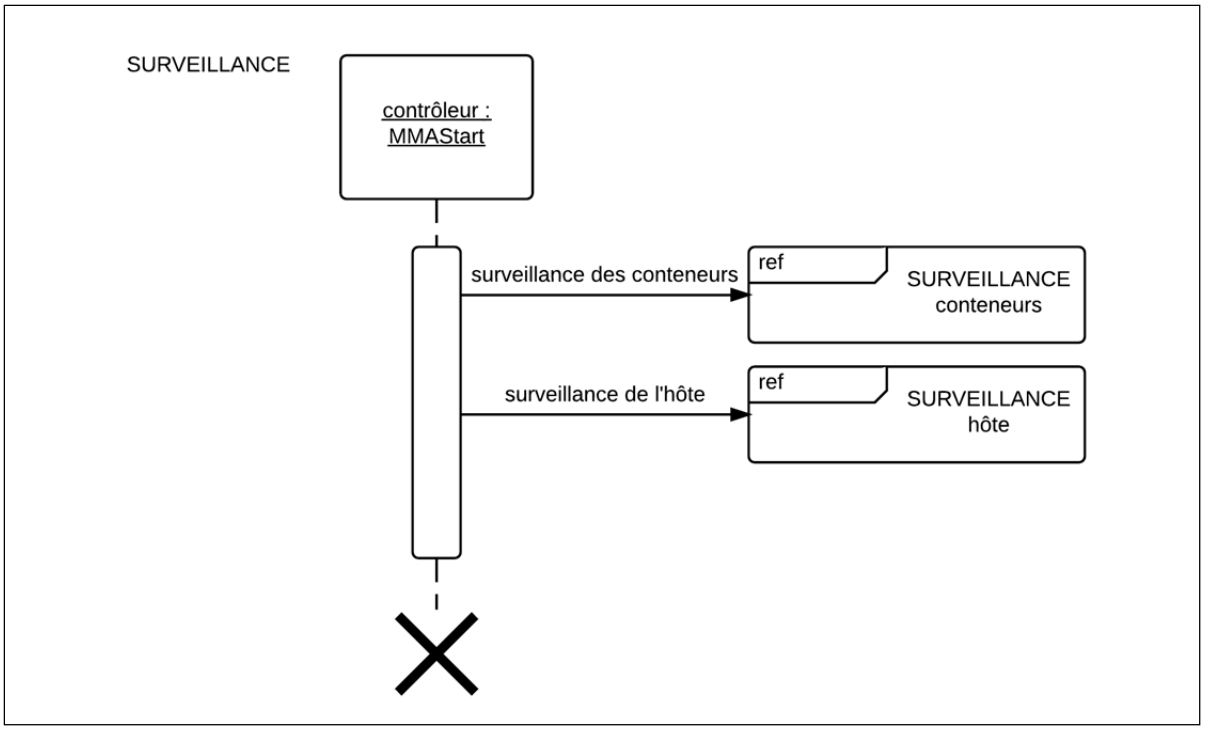

Figure A II-1 Diagramme de séquence du service de surveillance

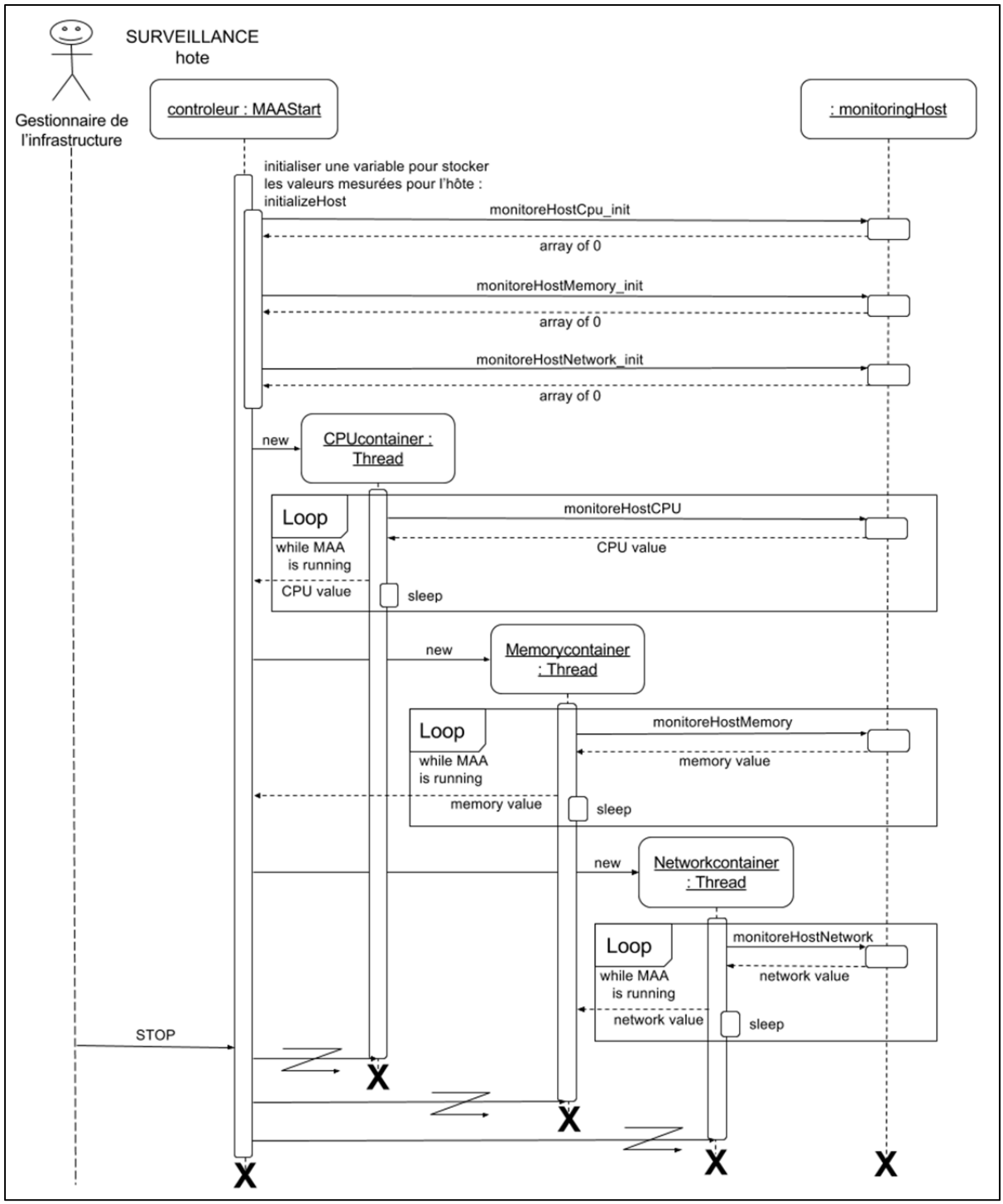

Figure A II-2 Diagramme de séquence du service de surveillance pour l'hôte

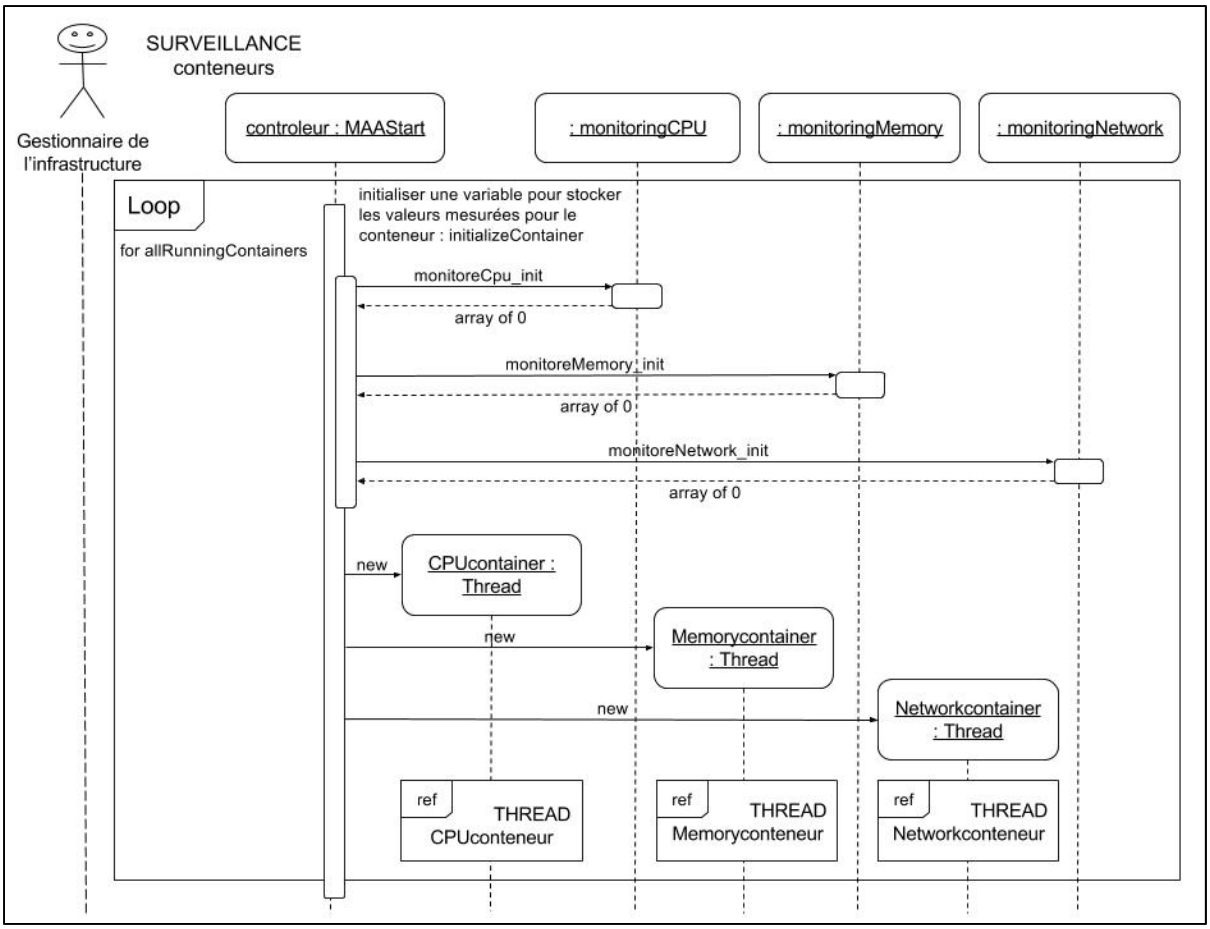

Figure A II-3 Diagramme de séquence du service de surveillance pour les conteneurs

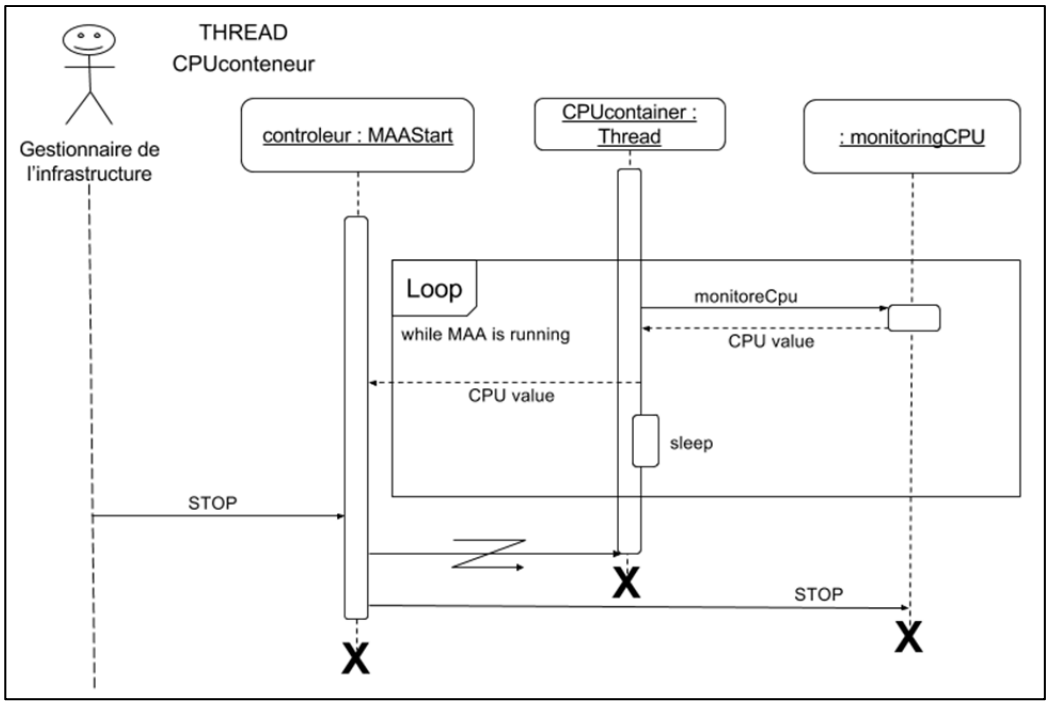

Figure A II-4 Diagramme de séquence du thread pour la surveillance du CPU des conteneurs

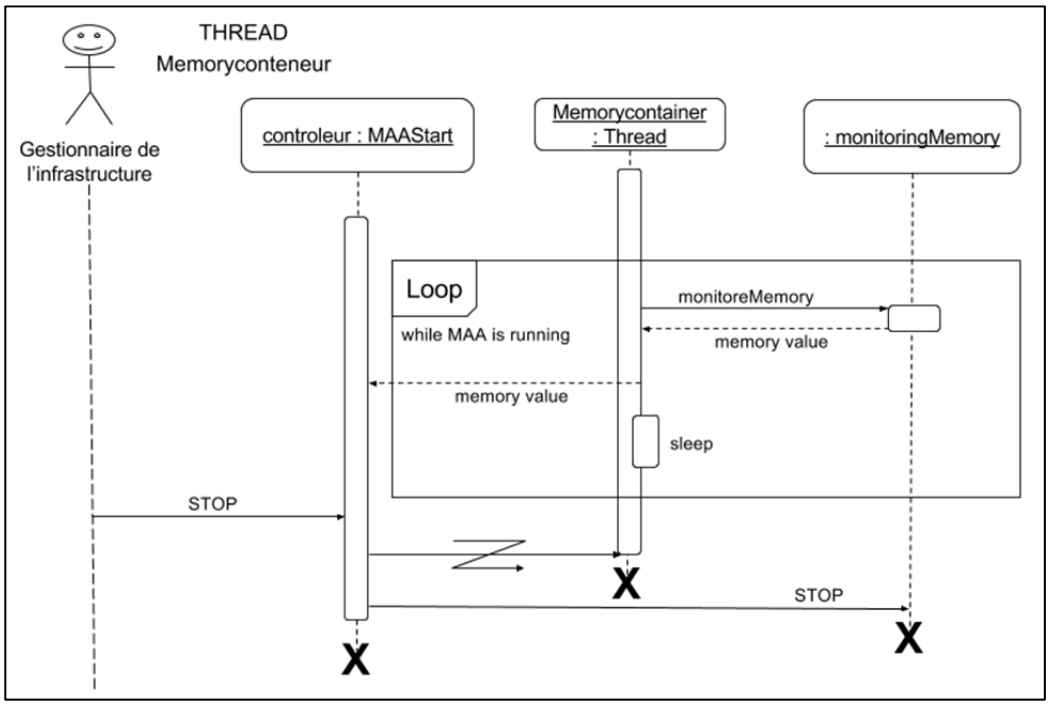

Figure A II-5 Diagramme de séquence du thread pour la surveillance de la mémoire des conteneurs

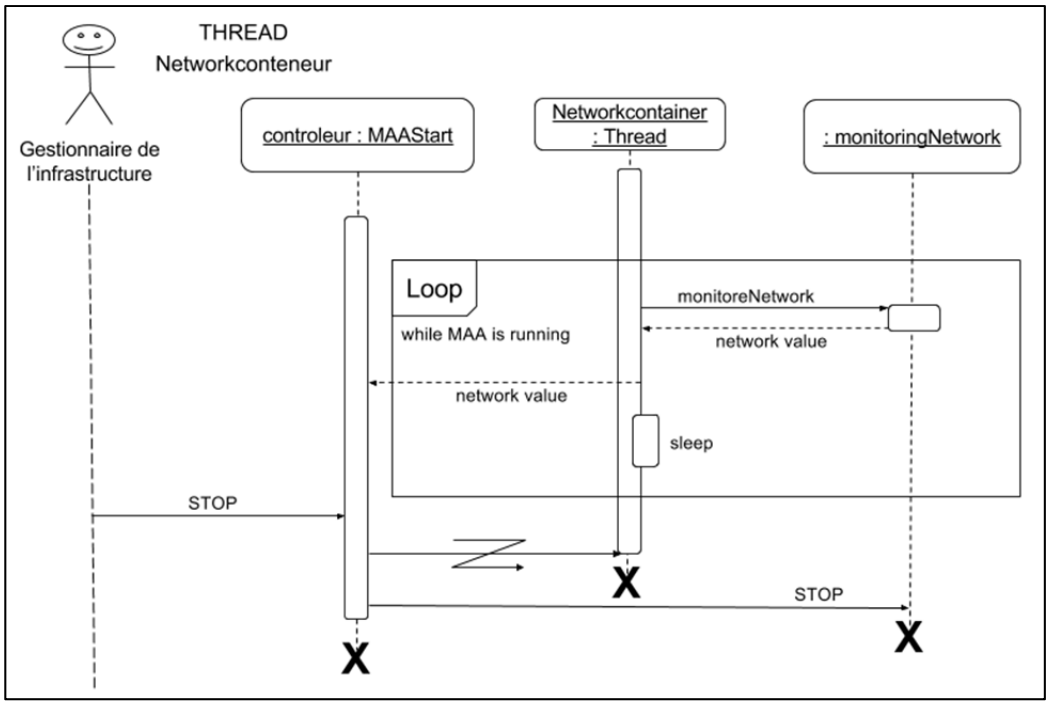

Figure A II-6 Diagramme de séquence du thread pour la surveillance de la bande passante des conteneurs

ANNEXE III

SCRIPTS POUR SUIVRE L'ÉTAT DES RESSOURCES

Télémétrer le CPU

Nous souhaitons avoir à intervalle de temps régulier le pourcentage de charge d'utilisation du CPU. L'intervalle de temps est fixé dans un fichier de configuration. Pour les conteneurs, les commandes Linux des *CGroups* nous permettent d'obtenir le temps total d'utilisation du CPU par un conteneur. Le détail de toutes les commandes possibles est disponible au chapitre 8.2.3 du guide Oracle Linux (Oracle). Pour avoir le pourcentage, nous relevons donc la charge d'utilisation du CPU à un instant t1 puis à un instant t2 et nous faisons la différence.

```
#!/bin/bash
# ${1} is the container's name
# ${2} is the time for observation
# ${3} is a computer constante
# get the cpu usage a t=0
NSCPUNOW=`lxc-cgroup -n ${1} cpuacct.usage`
# get the cpu usage a ${2} second later
sleep ${2}
NSCPUAFTER=`lxc-cgroup -n ${1} cpuacct.usage`
# calculate the percentage of cpu usage
NSDIFFERENCE=`echo "$NSCPUAFTER $NSCPUNOW" | awk '{print $1-$2}'`
TOTALTIME=$((${2}*1000000000))
RESULT=`echo "$NSDIFFERENCE $TOTALTIME" | awk '{print $1*100/$2}'`
# save result in text file
HOUR=$(date +%H)
MINUTE=$(date +%M)
SECONDE=$(date +%S)
echo ""$HOUR:$MINUTE:$SECOND", $RESULT" >> ./result/containers-${1}-cpu.csv
echo ""$HOUR:$MINUTE:$SECOND", $RESULT"
```

Pour l'hôte, Linux nous fournit une commande, *mpstat,* qui permet d'avoir pour chaque cœur de la machine physique son pourcentage d'utilisation. Il faut donc utiliser cette commande et récupérer la valeur correspondant à chaque cœur.

```
#!/bin/bash
# ${1} is the time for observation
# ${2} is a computer constante
# get the number of computer processors
NBPROCESSOR=`cat /proc/cpuinfo | grep processor | wc -l`
# to initialize the good line to get value
COUNTER=0
MAX=`echo "$NBPROCESSOR $COUNTER" | awk '{print $1+$2}'`
# Initialization of arrays
PREV_USER=()
USER=()
PREV_NICE=()
NICE=()
PREV_SYST=()
SYST=()
PREV_IDLE=()
IDLE=()
PREV_TOTAL=()
TOTAL=()

while [ $COUNTER -lt $MAX ]; do
  CPU=`cat /proc/stat | grep '^cpu'$COUNTER' '`
  PREV_USER=("${PREV_USER[@]}" "`echo $CPU | awk '{print $2}'`")
  PREV_NICE=("${PREV_NICE[@]}" "`echo $CPU | awk '{print $3}'`")
  PREV_SYST=("${PREV_SYST[@]}" "`echo $CPU | awk '{print $4}'`")
  PREV_IDLE=("${PREV_SYST[@]}" "`echo $CPU | awk '{print $5}'`")
  let SUM=${PREV_USER[$COUNTER]}+${PREV_NICE[$COUNTER]}+${PREV_SYST[$COUNTER]}
  PREV_TOTAL=("${PREV_TOTAL[@]}" "$SUM")
  let COUNTER=COUNTER+1
done

sleep ${1}

COUNTER=0
while [ $COUNTER -lt $MAX ]; do
  CPU=`cat /proc/stat | grep '^cpu'$COUNTER' '`
  USER=("${USER[@]}" "`echo $CPU | awk '{print $2}'`")
  NICE=("${NICE[@]}" "`echo $CPU | awk '{print $3}'`")
  SYST=("${SYST[@]}" "`echo $CPU | awk '{print $4}'`")
  IDLE=("${SYST[@]}" "`echo $CPU | awk '{print $5}'`")
  let SUM=${USER[$COUNTER]}+${NICE[$COUNTER]}+${SYST[$COUNTER]}
  TOTAL=("${TOTAL[@]}" "$SUM")
  let COUNTER=COUNTER+1
done

COUNTER=0
while [ $COUNTER -lt $MAX ]; do
  # Calculate the CPU usage between two checked.
  let SLEEPTIME=${2}*${1}
  let "DIFF_TOTAL=${TOTAL[$COUNTER]}-${PREV_TOTAL[$COUNTER]}"
  let "DIFF_USAGE_IDLE=100*DIFF_TOTAL/$SLEEPTIME"
  let "DIFF_USER=${USER[$COUNTER]}-${PREV_USER[$COUNTER]}"
  let "DIFF_USAGE_USER=100*$DIFF_USER/$SLEEPTIME"
  let "DIFF_SYST=${SYST[$COUNTER]}-${PREV_SYST[$COUNTER]}"
```

```
  let "DIFF_USAGE_SYST=100*$DIFF_SYST/$SLEEPTIME"

  # incremente and number to save
  let COUNTER=COUNTER+1

  # Save result in text file
  RESULT=`echo $RESULT$DIFF_USAGE_IDLE+`
  HOUR=$(date +%H)
  MINUTE=$(date +%M)
  SECOND=$(date +%S)
  echo ""$HOUR:$MINUTE:$SECOND", $DIFF_USAGE_USER" >> ./result/host-cpu-$COUNTER-usr.csv
  echo ""$HOUR:$MINUTE:$SECOND", $DIFF_USAGE_SYST" >> ./result/host-cpu-$COUNTER-sys.csv
  echo ""$HOUR:$MINUTE:$SECOND", $DIFF_USAGE_IDLE" >> ./result/host-cpu-$COUNTER.csv
done

echo ""$HOUR:$MINUTE:$SECOND", $RESULT"
```

Télémétrer la mémoire

Nous souhaitons avoir à intervalle de temps régulier l'utilisation de la mémoire par les conteneurs. Nous retrouvons au chapitre 8.2.8 du guide Oracle Linux (Oracle) la commande à utiliser. Elle nous renvoie la mémoire, en octets, utilisée par un conteneur. Il nous suffira ensuite de récupérer la capacité totale en mémoire de la machine physique pour calculer le pourcentage d'utilisation de cette dernière par le conteneur étudié.

```
#!/bin/bash
# ${1} is the container's name
# computer memory
TOTALCOMPUTER=`free -b | awk '$2 {print $2}' | awk 'NR==2'`
# memory use by the container
USEDCONTAINER=`lxc-cgroup -n ${1} memory.usage_in_bytes`
# percentage
RESULT=$USEDCONTAINER
# save result in text file
HOUR=$(date +%H)
MINUTE=$(date +%M)
SECOND=$(date +%S)
echo ""$HOUR:$MINUTE:$SECOND", $RESULT" >> ./result/containers-${1}-memory.csv
echo ""$HOUR:$MINUTE:$SECOND", $RESULT"
```

Télémétrer la bande passante

La dernière ressource de bas niveau que nous souhaitons télémétrer est le nombre d'octets lu et écrit sur le réseau par chaque conteneur et par l'hôte.

Dans les deux cas, la commande Linux *ifconfig* a été utilisée, car elle donne l'information souhaitée pour toutes les interfaces de la machine physique. Il a seulement fallu, avant

d'exécuter cette commande, récupérer le nom de l'interface pour chaque instance que nous souhaitons observer. Pour les conteneurs, une commande des *CGroups*, lxc-info, permet de récupérer de nombreuses informations sur un conteneur, dont le nom de son interface.

```
#!/bin/bash
# ${1} is the container's name
# ${2} is the time for observation
# save result in text file
HOUR=$(date +%H)
MINUTE=$(date +%M)
SECOND=$(date +%S)
# get the interface network
# monitor at t = 0
RX0=`lxc-attach -n ${1} ifconfig eth0 | grep "RX bytes" | cut -d: -f2 | awk '{ print $1 }'`
TX0=`lxc-attach -n ${1} ifconfig eth0 | grep "TX bytes" | cut -d: -f3 | awk '{ print $1 }'`
# wait TIME_MONITORING seconds
SLEEP=`echo "${2}" | awk '{print $1/2}'`
sleep $SLEEP
# monitor at t = t+TIME_MONITORING
RX1=`lxc-attach -n ${1} ifconfig eth0 | grep "RX bytes" | cut -d: -f2 | awk '{ print $1 }'`
TX1=`lxc-attach -n ${1} ifconfig eth0 | grep "TX bytes" | cut -d: -f3 | awk '{ print $1 }'`
RX=`echo "$RX1 $RX0 $SLEEP" | awk '{print ($1-$2)/$3}'`
TX=`echo "$TX1 $TX0 $SLEEP" | awk '{print ($1-$2)/$3}'`
echo ""$HOUR:$MINUTE:$SECOND", $RX" >> ./result/containers-${1}-Rx-network.csv
echo ""$HOUR:$MINUTE:$SECOND", $TX" >> ./result/containers-${1}-Tx-network.csv
echo ""$HOUR:$MINUTE:$SECOND", $RX+$TX"
```

Pour l'hôte, la première étape a été de lancer une commande permettant de trouver l'adresse IP de la machine. Ensuite, à partir de cette adresse, le nom de l'interface a pu être retrouvé. L'adresse IP a été trouvée grâce au code python suivant :

```
ipaddress = [(s.connect(('8.8.8.8', 80)), s.getsockname()[0], s.close()) for s in
                                        [socket.socket(socket.AF_INET, socket.SOCK_DGRAM)]][0][1]
```

ANNEXE IV

ARCHITECTURE GLOBALE DE L'OUTIL MAA

Chaque module présenté au chapitre 2 reporte dans un fichier log les états remarquables du système et les décisions prises pour qu'il fonctionne au mieux tout en optimisant le nombre de ressources consommées. Par exemple, le module d'analyse y inscrit les moments où il détecte un excès ou un manque de ressources CPU et le module de déploiement y spécifie le nombre et l'identifiant des cœurs virtuels qu'il attribue à chaque conteneur monitoré. Les figures A.IV.1, A. IV.2, A. IV.3, A. IV.4 et A. IV.5 illustrent le fonctionnement de l'outil, la synchronisation des modules et l'enchaînement des tâches. Pour avoir des illustrations claires, seuls les processus de surveillance, d'analyse et de déploiement utilisés pour les solutions proposées au chapitre 3 sont représentés.

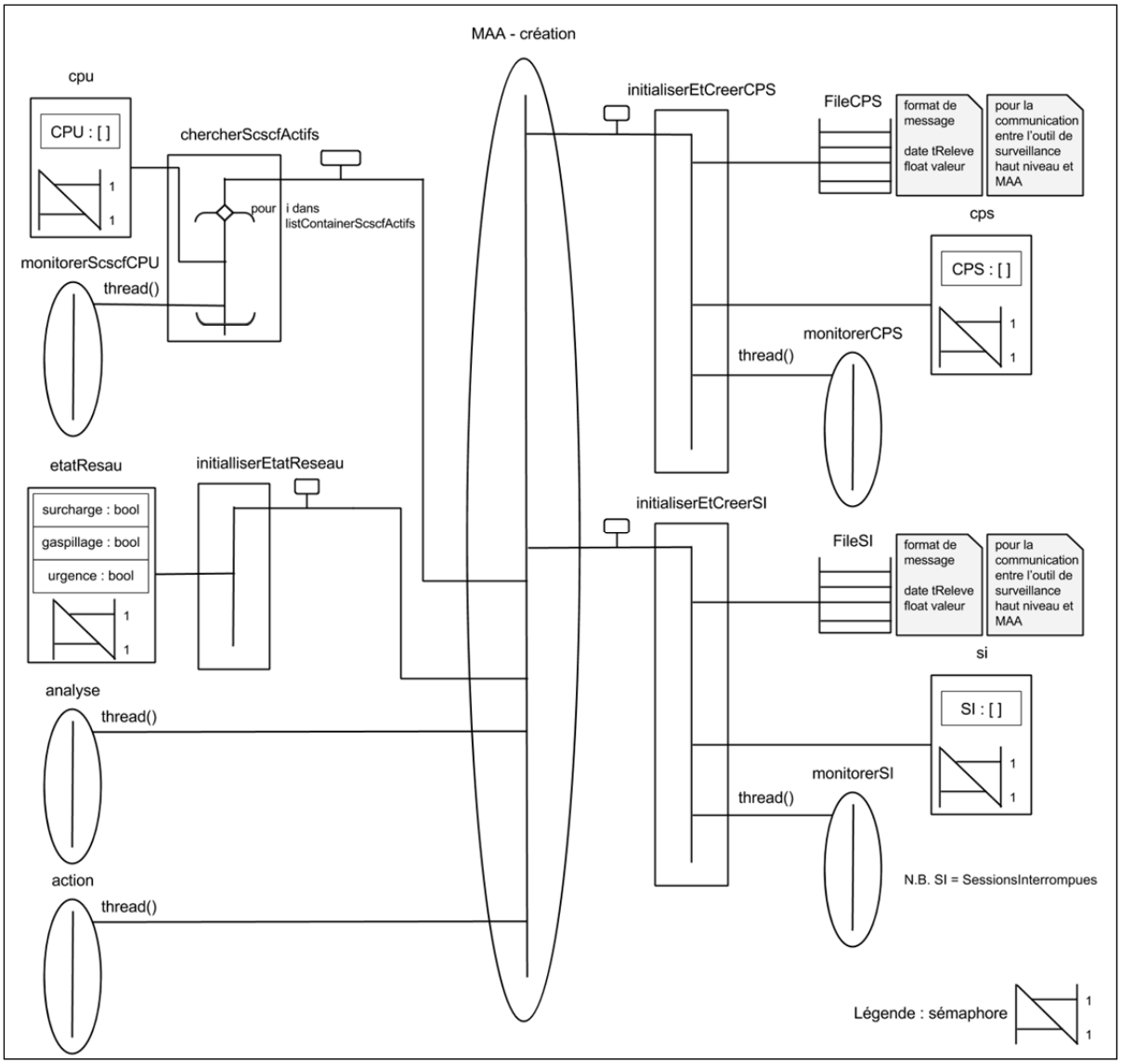

Figure A. IV.1 Initialisation de MAA

Lors de sa phase d'initialisation, représentée à la figure A. IV.1, MAA crée toutes les variables et les sémaphores nécessaires au fonctionnement de l'outil. Il initialise aussi les files de données pour communiquer avec l'outil de télémétrie haut niveau. Pour finir, MAA instancie les processus pour le monitorage, l'analyse et l'action. Ces processus s'initient à leur tour en se rattachant en écriture et lecture aux zones de mémoire partagée protégée dont ils vont avoir besoin. Quand tout est initialisé, le processus MAA sert juste à gérer l'interface

utilisateur. Lors de nos simulations, tous les processus créés ici s'endorment après l'initialisation jusqu'à ce que le système de simulation du système IMS démarre.

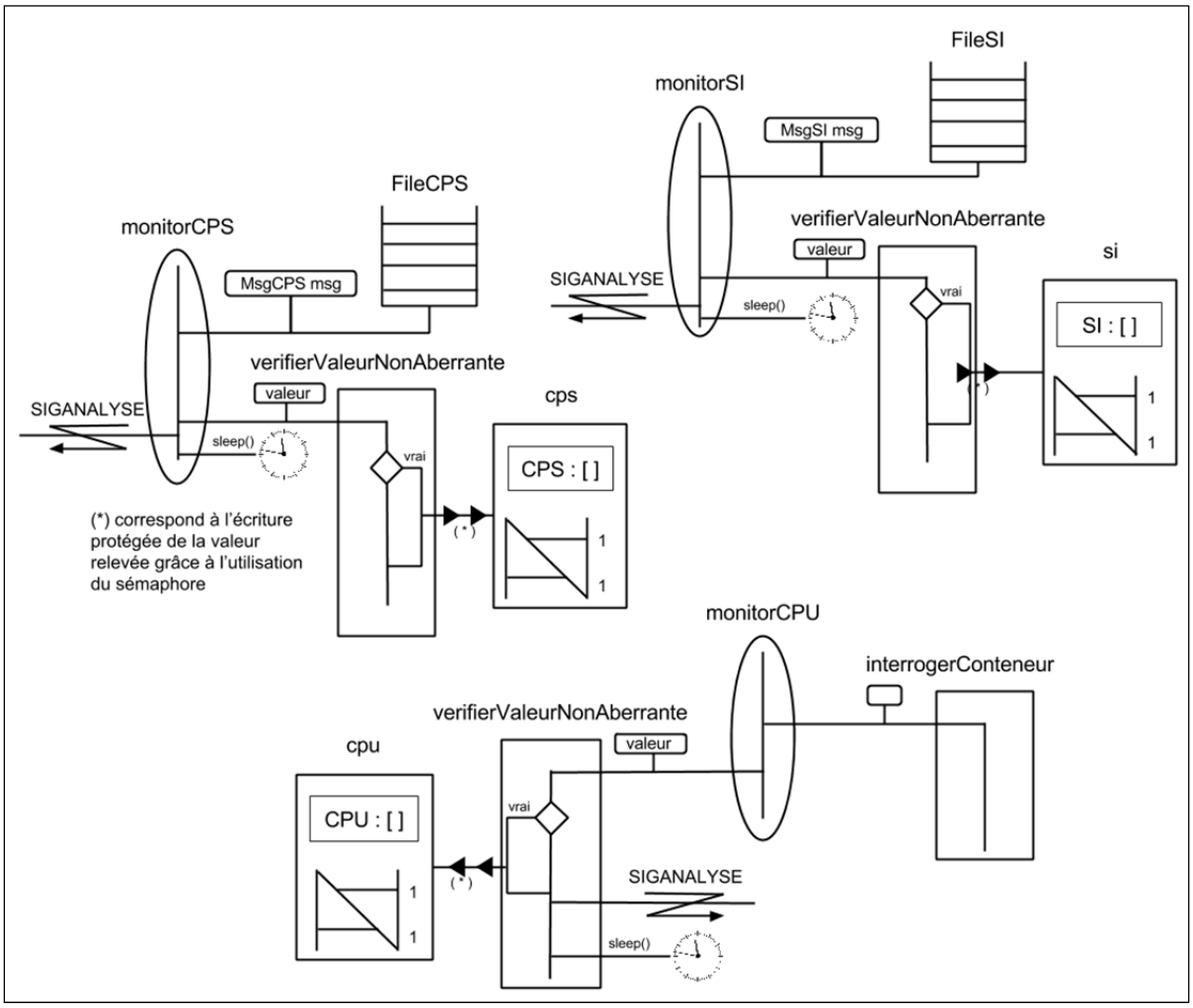

Figure A. IV.2 Phase moteur des processus de monitorage

La figure A. IV.2 regroupe la phase moteur des processus de monitorage de MAA pour la consommation du CPU par les conteneurs (monitorCPU), le nombre d'appels par seconde émis par les utilisateurs du système (monitorCPS) et le nombre de sessions interrompues de manière non souhaitée (monitorSI). Le processus de monitorage du CPU interroge le conteneur qu'il supervise pour connaître sa consommation. Les deux autres processus lisent la valeur dans les fichiers de stockage remplis par l'outil de télémétrie haut niveau. Chacun de ces processus vérifie ensuite que la valeur relevée n'est pas aberrante. Les règles pour

cette vérification sont mentionnées à la section 3.3.6. Si la valeur relevée est acceptée, ils l'enregistrent dans la variable correspondante pour que le module suivant puisse l'utiliser. Pour informer le module d'analyse d'une nouvelle valeur, les processus lèvent un *flag* ici nommé *SIGANALYSE*. Les processus dorment ensuite avant de faire une nouvelle mesure. Cette pause est possible car nous avons vu au chapitre 3 que nous n'avons besoin que d'une valeur toutes les secondes.

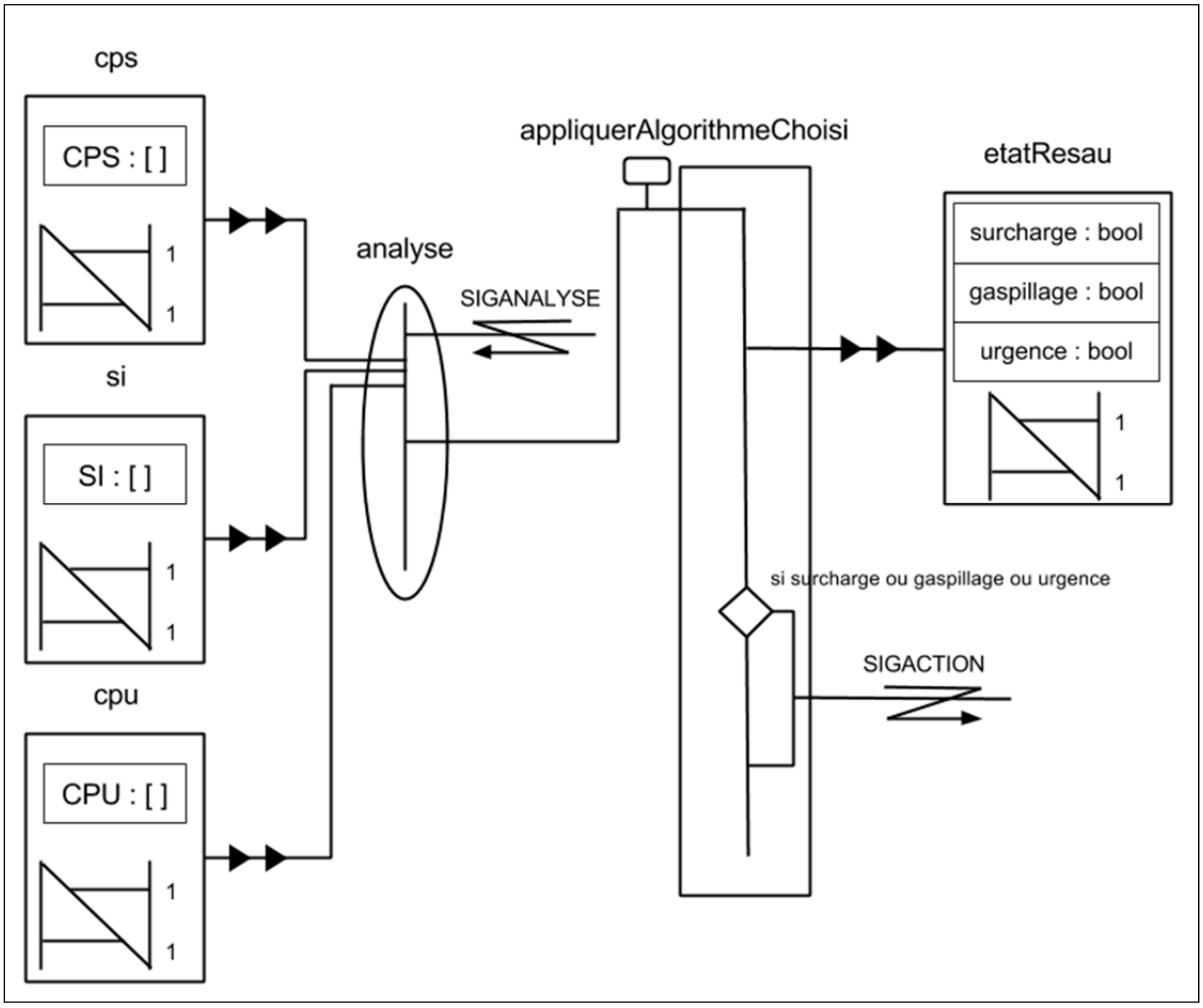

Figure A. IV.3 Phase moteur du processus d'analyse

La figure A. IV.3 représente la phase moteur du processus d'analyse. Nous voyons que lorsque le *flag SIGANALYSE* mentionné précédemment est levé, le processus d'analyse récupère dans les variables adéquates toutes les valeurs voulues. Afin d'éviter tout conflit

d'accès, ces variables sont protégées par un sémaphore pour que les processus ne puissent pas les utiliser simultanément. Le processus d'analyse applique ensuite un des algorithmes présentés au chapitre 3. Le choix de cet algorithme est défini dans le code du module. En fonction de la prédiction, le processus tient à jour les variables sur l'état du système. S'il constate que le système va manquer ou gaspiller des ressources, le processus lève le *flag SIGACTION* pour que le module de déploiement adapte les ressources allouées aux conteneurs gérés.

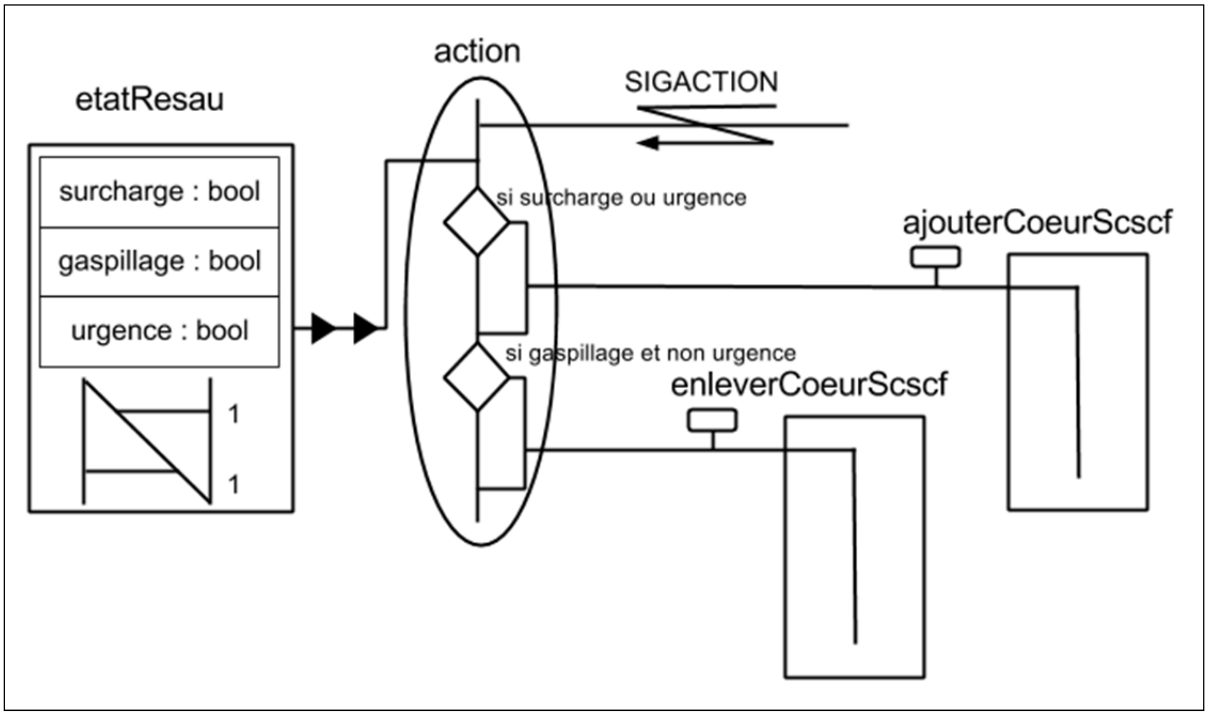

Figure A. IV.4 Phase moteur du processus d'action

La phase moteur du processus d'action, représentée à la figure A. IV.4, ajoute ou enlève un cœur virtuel d'un CPU à la liste de ceux utilisables par un conteneur. La décision est prise en fonction des variables sur l'état du système mises à jour par le module d'analyse. Ce processus n'agit que si le *flag SIGACTION* est levé.

La phase moteur des processus représente leurs tâches principales. Nous remarquons sur ces schémas les accès concourants aux variables et le système de synchronisation des processus

entre eux. Cette synchronisation est réalisée, comme représentée ci-dessus, par des *flags* qui génèrent des évènements. Un processus en attente d'un évènement est bloqué, c'est-à-dire qu'il n'exécute plus de tâche et donc qu'il ne consomme plus de ressources. C'est pour cette raison que l'enchaînement événementiel des processus, tout comme leur endormissement, a été mis en place. En effet, si les processus n'avaient qu'une boucle infinie ils consommeraient beaucoup plus de ressources CPU. Or ces dernières sont déjà très prisées par le système IMS virtualisé et doivent donc être utilisées le moins possible par l'outil de gestion développé.

Pour finir, un processus de destruction a été pensé. Comme illustré à la figure A. IV.5, ce dernier s'assure que tous les processus soient bien détruits quand l'outil est arrêté. Dans le cas contraire, nous aurions des processus zombies.

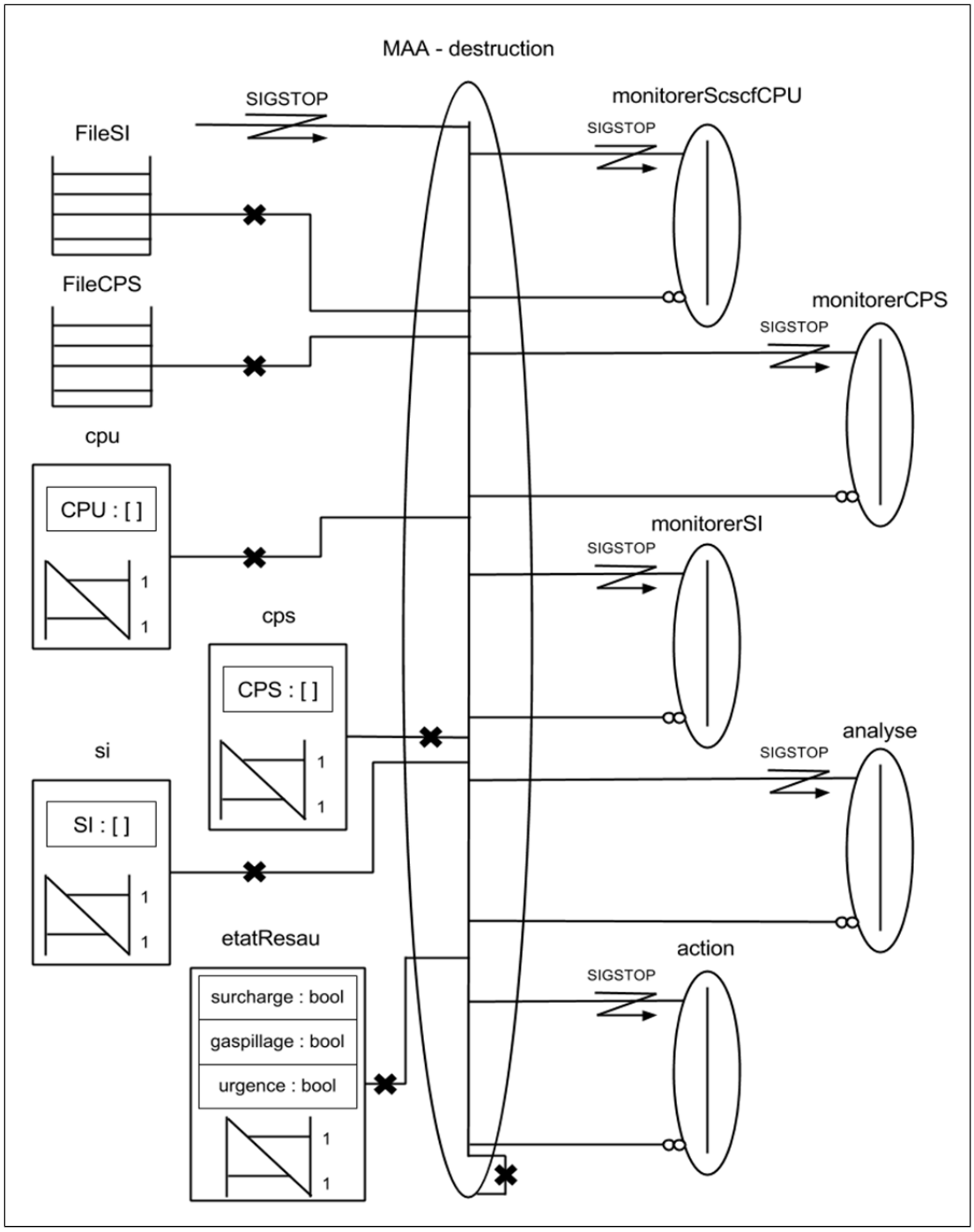

Figure A. IV.5 Destruction des processus de MAA

ANNEXE V

UTILISATION DE L'OUTIL MAA

Si l'utilisation de l'outil MAA vous intéresse, voici l'installation conseillée pour reproduire les mêmes tests. Une fois installée, il faudra aussi configurer l'outil à votre infrastructure avant de pouvoir l'utiliser. Une configuration supplémentaire est requise dans le cas où l'on veut utiliser l'algorithme avec le lambda. Il faut, en effet, créer la base de données des coefficients à utiliser avec cet algorithme.

Installation

L'outil a été développé en Python et utilise des scripts Shell. Comme expliqué dans les chapitres précédents, notre outil fonctionne sur Linux étant donné qu'il gère les conteneurs créés en clonant notre système d'exploitation et que ces conteneurs sont faits pour être gérés grâce à des commandes Unix. L'installation proposée ci-dessous sera donc exclusivement pour Linux. Dans notre cas, nous utilisons Linux dans sa version Ubuntu 14.04.

Il faut d'abord placer le dossier *maa* à l'emplacement de votre choix sur l'ordinateur qui hébergera aussi les conteneurs. Il faut ensuite donner les droits à tous les scripts de l'outil pour télémétrer les ressources sans que le mot de passe super utilisateur soit demandé. Pour cela, ouvrir un terminal et taper la commande : *sudo visudo.* Dans le fichier qui s'ouvre, sous la ligne *%sudo ALL=(ALL :ALL) ALL* ajouter les lignes suivantes :

ALL ALL=(ALL) NOPASSWD: chemin/maa/askAllContainer.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/askStateContainer.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/monitoring/cpu.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/monitoring/io.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/monitoring/memory.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/monitoring/network.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/monitoring/hostcpu.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/monitoring/hostio.sh

ALL ALL=(ALL) NOPASSWD: chemin/maa/monitoring/hostmemory.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/monitoring/hostnetwork.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/monitoring/working.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/monitoring/searchIP.sh
ALL ALL=(ALL) NOPASSWD: chemin/maa/action/scscf_core.sh

Dans les lignes ci-dessus, *chemin* correspond au chemin sur votre machine où vous avez placé le dossier *maa.* Sauvegarder le fichier et quitter.

Il faut maintenant mettre en place l'outil de télémétrie haut niveau. Sur la machine qui représente la porte d'entrée des utilisateurs sur le système IMS déployé, placez sur le bureau le dossier *monitoringmanager.* L'outil proposé ici fonctionne avec le simulateur OpenIMSCore (OpenSourceIms) où nous avons modifié le chemin de sortie du fichier log afin de récupérer les informations nécessaires à notre outil de gestion des ressources. Les modifications effectuées sont les suivantes : à la ligne 784 du fichier */ims_bench/mgr_main.cpp*, remplacer le chemin de la variable *g_LogFile* par celui du dossier *monitoringmanager* placé sur le bureau. Il faut, ensuite, donner les droits à l'outil de lancer son script sans que le mot de passe super utilisateur soit demandé. Pour cela, taper dans un terminal *sudo visudo* et ajouter sous la ligne *%sudo ALL=(ALL :ALL) ALL* la ligne suivante :. *ALL ALL=(ALL) NOPASSWD: home/username/Desktop/monitoringmanager/extract.sh.* Dans le cas d'un système sans simulateur ou d'un autre simulateur, vous devez modifier le code du fichier *monitormanager.py* afin que l'outil déployé ici renvoie à l'outil principal le nombre d'appels par seconde reçus et le nombre de sessions interrompues de manière non voulue. L'installation est ensuite terminée.

Configuration

Maintenant que l'installation est terminée, l'outil doit être configuré pour fonctionner avec votre infrastructure. Pour configurer l'outil, il faut le lancer. Pour cela, placez-vous dans le dossier *maa* et tapez la commande suivante : *python MMAstart.py.* La fenêtre illustrée à la figure A V-1 s'ouvre.

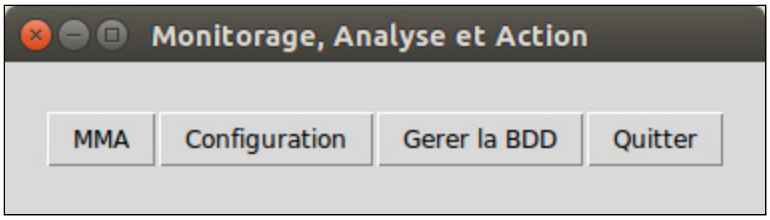

Figure A V-1 Fenêtre principale de l'outil

Il faut ensuite cliquer sur le bouton *configuration* pour accéder à la fenêtre d'édition des paramètres. En cas de problème, les paramètres sont aussi modifiables dans le fichier *config.py* situé à la racine du dossier du code source de l'outil. Dans la fenêtre d'édition des paramètres ou dans le fichier, il faut renseigner tous les champs suivant en fonction de votre système.

- TEST_BED_IP : correspond à l'adresse IP de la machine qui reçoit les demandes de sessions. Dans notre cas de test, cela correspond à la machine qui génère les appels. C'est aussi sur cette machine que vous avez placé l'outil de télémétrie haut niveau.
- TEST_BED_USER : correspond au nom de l'utilisateur associé à l'adresse IP précédemment renseignée.
- MY_IP : correspond à l'adresse IP de la machine sur laquelle est installé l'outil MAA.
- MY_USER : correspond au nom de l'utilisateur qui utilise l'outil MAA et est lié à l'adresse IP fournie au paramètre précédent.
- MY_SAVEPATH : correspond au dossier dans lequel les scripts de télémétrie de l'outil enregistreront les valeurs qu'ils auront relevées et fournies à l'outil. Il n'est pas nécessaire de le modifier.
- PATH_CPS, PATH_FAILED, PATH_TIMESTART : correspondent aux chemins des fichiers renseignés par l'outil de télémétrie haut niveau. Si vous n'avez pas modifié le chemin dans l'autre outil, il ne faut pas le modifier ici non plus.
- PATH_LOG : correspond au chemin du fichier log.
- CORE_SCSCF : correspond au numéro des cœurs virtuels de la machine physique sur laquelle les conteneurs sont déployés qui peuvent être alloués aux conteneurs S-CSCF gérés.

- TIME_ToDEPLOY_NewCORE : correspond au temps nécessaire pour qu'un cœur alloué soit totalement intégré au système. La valeur renseignée par défaut, 10, est celle trouvée au cours de notre étude et mentionnée dans les chapitres précédents. Elle peut être modifiée si vous constatez une différence entre votre système et le nôtre.
- STEP_REGISTRATION_NUMBER, STEP_BABYSLEEP_NUMBER, STEP_STIR_NUMBER, STEP_START_NUMBER : correspondent aux numéros des étapes avant le lancement de la simulation. Si vous souhaitez que l'outil démarre directement, sans attendre que le simulateur démarre la simulation, ou si vous n'avez pas besoin d'attendre, il suffit de mettre tous ces paramètres à 0.
- TIME_SLEEP : correspond au temps pendant lequel un thread de monitorage s'endort entre deux relevés. Plus ce temps est élevé et moins l'outil consomme de ressources mais moins il est fiable. Une seconde semble un bon compromis entre fiabilité et consommation des ressources et c'est le temps utilisé dans toutes nos simulations.
- TIME_MONITORING : correspond au temps pendant lequel un script s'endort lorsqu'il doit relever deux valeurs séparées par un intervalle de temps pour calculer les ressources consommées par le système. Attention, si la valeur est trop faible, le résultat sera retourné plus rapidement, mais risque d'être erroné car le système n'aura pas suffisamment varié dans l'intervalle de temps considéré.
- MAX_MESSAGE_FAILED : correspond à la valeur du pourcentage de sessions interrompues à partir de laquelle on considère que le système est dans un état critique. Au-delà de cette valeur, même si on considère que le système dispose de plus de ressources CPU que nécessaire, on ne lui enlèvera pas de cœur. Dans nos simulations, nous avions choisi 3% car notre maximum toléré avait été fixé à 5% et que le système, même quand il va bien, a un nombre de sessions interrompues qui oscille entre 0 et 3%.
- URGENCY_MESSAGE_FAILED : correspond à la valeur du pourcentage de sessions interrompues à partir de laquelle le système est dans un état d'urgence. Lorsque cette valeur est atteinte, quelles que soient les ressources utilisées par les conteneurs de type S-CSCF, un cœur leur est ajouté. Cette valeur a été fixée à 5% car

on a remarqué qu'au-delà, sans changement du système ou du jeu de test joué, le nombre de sessions ne cesse d'augmenter jusqu'à ce que le système meurt.

- MAX_SCSCF_CPU : correspond à la valeur du seuil du CPU à partir duquel on ajoute un cœur au conteneur S-CSCF. Comme expliqué précédemment nous avons fixé cette valeur à 70%.
- TOLERANCE_LINEAIRE : correspond à la valeur minimale admise pour le coefficient de la régression linéaire. Si la valeur calculée est inférieure à cette valeur, les coefficients obtenus ne sont pas gardés et ceux précédemment utilisés seront repris. La tolérance linéaire permet alors de ne pas prendre en compte des résultats aberrants qui seraient dus à une valeur relevée particulière et non représentative. Cependant, si cette valeur est trop élevée, trop de valeurs sont ignorées et la prédiction ne sera pas optimale car les valeurs seront souvent non prises en compte. En effet, le coefficient de corrélation est plus faible dans les phases de la simulation où il y a changement de pente car les points sont plus éparpillés. Dans notre cas, nous l'avons fixé à 0.6.
- TOLERANCE_EXPONENTIELLE : correspond à la valeur minimale pour le coefficient de la régression exponentielle. Si la valeur calculée est inférieure à cette valeur, les coefficients obtenus ne sont pas gardés et ceux précédemment utilisés sont repris. Pour les mêmes raisons que le paramètre précédent, la valeur choisie ne doit pas être trop faible pour ne pas garder des valeurs aberrantes mais ne doit pas non plus être trop haute pour ne pas ignorer trop de valeur comme cela est souvent le cas au moment des variations du jeu de test qui entraîne des variations sur la consommation du CPU des conteneurs de type S-CSCF. Nous l'avons fixé, dans notre cas et pour toutes les simulations présentées, à 0.3.
- SIZE_ARRAY_BIG : correspond au nombre de valeurs utilisées pour le calcul des coefficients de la régression linéaire et exponentielle. Nous l'avons fixé à 15. Ce nombre est suffisant pour avoir des coefficients proches de la réalité et permet un temps d'exécution du calcul inférieur à la demi-seconde. En effet, comme nous l'avons vu lors de la présentation de l'algorithme de décision par seuil prédictif le plus gourmand, moins on garde de la valeur et plus le temps d'exécution est rapide.

Cependant, avec trop peu de valeurs, le calcul des coefficients des régressions ne sera pas toujours fiable. Avec trop de valeurs non plus car, dans le cas où il y a eu une variation dans la pente de la métrique étudiée, nous aurons un résultat faussé par des valeurs qui n'ont plus de raisons d'être prises en compte.

- CPS_MIN : correspond au nombre d'appels par seconde minimum que peut recevoir le système. Si cette valeur est lue par l'outil de monitorage alors il considère que le système de fonctionne plus et s'arrête. Pour que l'outil ne s'arrête jamais parce que trop peu d'appels sont reçus il suffit de mettre la valeur du paramètre à 0. Dans le cas de la simulation de nos 3 jeux de test, nous l'avons fixée à 100 car le minimum est de 150 et ainsi l'enregistrement des valeurs des métriques s'arrête automatiquement.
- UP_LAMBDA : permet de déterminer la variation du cps à partir de laquelle on considère un changement de tendance de cette métrique à un instant t. Dans notre cas, nous utilisons 50 car nous savons qu'à chaque étape de notre simulation la variation du cps est de 50. Dans les simulations où nous ne savons pas quelle valeur indiquée, si la valeur est trop faible, la variation tolérée avant de considérer un changement de pente sera faible. Donc, la moindre variation de cps sera détectée comme un changement de pente. Étant donné qu'à chaque changement de pente le tableau pour prédire le CPS et le CPU est vidé, si trop de changements de pente sont détectés ils seront toujours de petites tailles et une taille trop petite ne permet pas de prédire des valeurs avec beaucoup de fiabilité car il y a trop peu de valeurs sur lesquelles se baser. En revanche, si la valeur est trop grande, un changement de pente sera détecté plus tard et la prédiction risque d'être faussée car elle s'appuiera sur des valeurs d'avant le changement de pente encore non détecté.

Une fois toutes les modifications nécessaires effectuées, le premier outil est prêt.

Dans l'outil de télémétrie haut niveau, il faut aussi adapter les paramètres en fonction des noms et adresses IP des machines de votre infrastructure. Pour cela, ouvrir le fichier *monitormanager.py* et éditer les paramètres *username*, *userip* et *userpath* situés dans les premières lignes du code. *Username* correspond au nom de l'utilisateur qui va lancer l'outil MAA sur la machine où sont déployés les conteneurs à gérer. *Userip* correspond à l'IP de

cette même machine. *Userpath* correspond au chemin du dossier dans lequel le dossier du code source de MAA a été placé.

Utilisation

L'outil peut être utilisé avec les deux algorithmes de décision par seuil prédictif. Cependant, pour utiliser le moins gourmand des deux, celui qui utilise les valeurs des coefficients de la régression exponentielle pré-calculés, il faut remplir la base de données. La base de données est automatiquement créée au lancement de MAA, mais elle est vide. Pour la remplir, depuis la fenêtre principale de MAA, cliquer sur le bouton *Gérer la BDD*. Une nouvelle fenêtre s'ouvre, la liste à gauche représente les données présentent dans la BDD. À droite, trois boutons permettent de gérer la base de données. Pour la remplir, il faut lancer une simulation et cliquer sur le bouton *Paramétrer la BDD*. Une fois la simulation terminée, il faut cliquer sur le bouton *Quitter* de la fenêtre principale pour que l'outil calcul tous les coefficients de la simulation qui vient de se terminer et les enregistre. Lorsque l'outil sera relancé, les valeurs calculées sont consultables dans la fenêtre de gestion de la base de données. Cette dernière peut être vidée en cliquant sur le bouton *Vider la BDD* ou juste les tuples jugés non nécessaire peuvent être sélectionnés et supprimés en cliquant sur le bouton *Supprimer la sélection*. Attention, toute suppression est définitive. Une fois que la base de données est remplie, les valeurs sont automatiquement utilisées lorsque l'outil est lancé en cliquant sur le bouton *MAA*. Si la base de données est vide, les algorithmes présentés à l'annexe VI seront utilisés. Dès qu'elle contient une valeur, si le lambda courant correspond à celui de valeurs enregistrées dans la base de données alors ces dernières seront automatiquement utilisées et l'algorithme présenté à l'annexe VIII sera utilisé. Pour remplir la base de données, l'algorithme VI.1 est utilisé. Tout au court de la simulation, les valeurs relevées sont aussi enregistrées dans des fichiers. Il est possible de les regrouper en un seul fichier lorsque la simulation est terminée, grâce à l'outil présent dans le dossier *maa/result/extractResults* en lançant la commande *python extractData.py*. Un fichier log est également créé par MAA et indique la détection des phases critiques du système et l'ajout ou le retrait de cœurs.

ANNEXE VI

ALGORITHMES POUR LA MÉTHODE DE PRÉDICTION AVEC UN SEUIL

Algorithme VI.1 Algorithme d'analyse par la méthode prédictive avec seuil

Analyse

Entrée : nombre d'appels par seconde : CPS
consommation du CPU du S-CSCF : CPU
nombre de sessions interrompues : SI
nombre de P-CSCF détecté à l'initialisation : nbPcscf

Sortie : booléens pour indiquer l'état du S-CSCF : SCSCF_OVERLOADED et SCSCF_UNDERLOADED
booléens pour indiquer l'état du système : SYSTEM_MANYLOST, SYSTEM_FAILED

Constante : nombre maximal de sessions interrompues tolérées : SESSION_FAILED
nombre critique de sessions interrompues : MANY_FAILED
seuil maximal pour la charge CPU d'un cœur : MAX_CPU

```
1   % initialization
2   SCSCF_OVERLOADED = SCSCF_ UNDERLOADED =
    SYSTEM_MANYLOST = SYSTEM_ FAILED = False
3   % analyser la dernière valeur relevée du nombre de sessions interrompues
4   si SI[-1] >= SESSION _FAILED
5       SYSTEM_ FAILED = True
6       return
7   sinon si SI[-1] >= MANY_FAILED
8       SYSTEM_MANYLOST = True
9   % vérifier que tous les P-CSCF soient encore actifs
10  si nombrePcscfActif() != nbPcscf
```

```
11          alerterSystemeMourant()
12          return
13      % calculer la tendance du CPS
        % 1 pour croissant, 0 pour constant, -1 pour décroissant
14      tendance = calculerTendanceCPS(CPS)
15      % calculer le nombre de cœurs actuellement alloués au S-CSCF
16      nbCoeurScscf = calculerNombreCoeurAlloue(SCSCF)
17      % si CPS croissant ou constant
18      si tendance != -1
19          preditCPS = predireCPS(CPS)
20          preditCPU = predireCPU(CPS, CPU, preditCPS)
21          si preditCPU >= MAX_CPU
22                  SCSCF_OVERLOADED = True
23      % si CPS décroissant, vérifier si le seuil MIN_CPU est atteint
24      sinon
25          % relever la dernière valeur monitorée pour le CPU du S-CSCF
26          cpu = CPU[-1]
27          % calculer le seuil minimal pour la charge CPU du conteneur S-CSCF
28          MIN_CPU = MAX_CPU*(nbCoeurScscf – 1)
29          % comparer la valeur du CPU et celle du seuil
30          if cpu < MIN_CPU
31                  SCSCF_UNDERLOADED = True
```

L'algorithme X.1 d'analyse fait appel aux fonctions calculerTendanceCPS, predireCPS et predireCPU décrites par les algorithmes VI.2, VI.4 et VI.5.

Algorithme VI.2 Algorithme de calcul de la tendance du CPS

CalculerTendanceCPS

Entrée : nombre d'appels par seconde : CPS

Sortie : entier représentant la tendance : tendance % 1 pour croissant, 0 pour constant et -1 pour décroissant

Constante : le temps moyen d'une étape de la simulation : GAP_LAMBDA

```
1   actuelCPS = CPS[-1]
2   precedentT = max( -tStart, -GAP_LAMBDA )
3   precedentCPS = CPS[precedentT]
4   c = compare(actuelCPS, precedentCPS)
5   return c
```

Avec la fonction compare défini par l'algorithme VI.3 :

Algorithme VI.3 Algorithme de comparaison de deux valeurs

Compare

Entrée : valeur à comparer : valeurC

valeur de référence : valeurR

Sortie : entier représentant la comparaison % 1 pour supérieur, 0 pour égal et -1 pour inférieur

Constante : augmentation moyenne d'une étape de la simulation : UP_LAMBDA

```
1   tolerance = UP_LAMBDA * 30/100
2   si valeurC > valeurR + tolerance
3       return 1
4   sinon si valeurC < valeurR – tolerance
5       return -1
6   sinon
7       return 0
```

La valeur de 30% utilisée dans l'algorithme VI.3 pour définir la tolérance a été fixée après de nombreuses simulations. Cette valeur ne doit pas être trop élevée afin de ne pas trop dissimuler la réalité, mais elle ne doit pas non plus être trop basse sinon la moindre fluctuation dans les valeurs relevées peut entraîner des prises de décision de l'outil MAA qui ne seront pas valides. Par exemple, lorsque le cps est dans une phase de croissance, nous pouvons tout de même avoir une valeur légèrement inférieure à la valeur précédente à cause de la fiabilité du simulateur, MAA ne doit cependant pas penser que le cps décroît.

Algorithme VI.4 Algorithme de prédiction du nombre d'appels par seconde

PredireCPS

Entrée : nombre d'appels par seconde : CPS
instant du dernier changement de tendance du CPS : tStart

Sortie : valeur prédite pour le CPS : preditCPS

Constante : nombre de secondes entre l'instant t et celui de la prédiction : TIME_TO_DEPLOY_CORE
coefficient minimum de la régression linéaire accepté : TOLERANCE_LINEAIRE

```
1   depart = max( (len(CPS) – TIME_TO_DEPLOY_CORE), (len(CPS) - tStart) )
2   analyseCPS = CPS[depart:]
3   t = range(depart, len(CPS) )
4   a, b, r = regressionLineaire(t, analyseCPS)
5   si r >= TOLERANCE_LINEAIRE
6       preditCPS = a * ( now() + TIME_TO_DEPLOY_CORE ) + b
7   sinon
8       preditCPS = precedentPreditCPS
9   return preditCPS
```

Algorithme VI.5 Algorithme de prédiction de la charge CPU

PredireCPU

Entrée : nombre d'appels par seconde : CPS

consommation du CPU du S-CSCF : CPU

valeur prédite du CPS : preditCPS

instant du dernier changement de tendance du CPS : tStart

Sortie : valeur prédite pour le CPU : preditCPU

Constante : nombre de secondes entre l'instant t et celui de la prédiction : TIME_TO_DEPLOY_CORE

coefficient minimum de la régression exponentielle accepté : TOLERANCE_EXPONENTIELLE

```
1. depart = max( (len(CPS) – TIME_TO_DEPLOY_CORE), (len(CPS) - tStart) )
2. analyseCPS = CPS[depart:]
3. analyseCPU = CPU[depart:]
4. a, b, r = regressionExponentielle(analyseCPS, analyseCPU) % détaillé en annexe VII
5. si r >= TOLERANCE_EXPONENTIELLE
6.         preditCPU = a * exp( b * preditCPS )
7. sinon
8.         preditCPU = precedentPreditCPU
9. return preditCPU
```

ANNEXE VII

ALGORITHMES DE CALCUL DES COEFFICIENTS DE LA RÉGRESSION LINÉAIRE ET EXPONENTIELLE

Régression linéaire

Pour nos algorithmes prédictifs, nous avons besoin de calculer les coefficients a et b de la régression linéaire d'équation Y = a*X+b.

Pseudo-code de l'algorithme implémenté

Algorithme VII.1 Calcul des coefficients de la régression linéaire

```
regressionLineaire(valX : [taille n], valY : [taille n])
    % calculer a et b pour la régression linéaire de valY = f(valX) et le coefficient de corrélation r
    mX = moyenne(valX)
    mY = moyenne(valY)
    cov = covariance(valX, mX, valY, mY)
    vX = covariance(valX, mX, valX, mX)
    vY = covariance(valY, mY, valY, mY)
    si VX != 0 et vY != 0
        a = cov / vX
        b = mY – a * mX
        r = cov / sqrt(vX * vY)
```

Régression exponentielle

Pour nos algorithmes prédictifs, nous avons besoin de calculer les coefficients α et β de la régression exponentielle d'équation $Y = \alpha * \exp(\beta * X)$.

Explication mathématique

$$Y = \alpha * \exp(\beta * X)$$

Transformation avec ln()

$$\ln(Y)=\ln(\alpha*\exp(\beta*X))=\ln(\alpha)+\ln(\exp(\beta*X))=\ln(\alpha)+ \beta*X$$

On pose $Y'=\ln(Y)$

$$Y'=\alpha'+\beta X$$

Régression linéaire $y=a*x+b$

$$b=\alpha' \text{ et } a=\beta$$

Avec la transformation ln()

$$\alpha'=\ln(\alpha)=b \text{ d'où } \alpha=\exp(\ln(\alpha))=\exp(\alpha')=\exp(b) \text{ et } \beta=a$$

Pseudo-code de l'algorithme implémenté

Algorithme VII.2 Calcul des coefficients de la régression exponentielle

```
regressionExponentielle(CPS : [taille n], CPU : [taille n])
    % récupérer que les lignes où l'on a une valeur dans les deux tableaux
     valX = valY = []
    pour i dans range(0;len(CPS))
        si CPS[i] not null et CPU[i] not null
            valX.append(CPS[i])
            valY.append(CPU[i])
    % calculer a et b pour la régression linéaire de valY = f(valX) et le coefficient de corrélation r
    mX = moyenne(valX)
    mY = moyenne(valY)
    cov = covariance(valX, mX, valY, mY)
    vX = covariance(valX, mX, valX, mX)
```

```
vY = covariance(valY, mY, valY, mY)
si VX != 0 et vY != 0
        a = cov / vX
        b = mY – a * mX
        r = cov / sqrt(vX * vY)
% calculer α et β pour la régression exponentielle de CPU = g(CPS)
α = exp(b)
β = a
```

Complexité de l'algorithme

Nous appelons n la taille des tableaux de données en entrée de la fonction décrite dans l'algorithme VII.2. Pour initialiser les tableaux de données valX et valY, l'algorithme effectue une première boucle sur l'ensemble des données du tableau CPS. L'initialisation de ces tableaux est donc en O(n). Pour calculer les coefficients a et b, il est nécessaire de calculer la moyenne et la variance de valX et valY ainsi que la covariance. Ces calculs nécessitent de parcourir les tableaux valX ou valY qui sont de taille n. La complexité de chaque calcul est donc en O(n). Les autres calculs effectués lors de la recherche des coefficients de la régression exponentielle sont de complexité inférieure à O(n) et sont donc négligés. L'algorithme est alors de complexité 6*O(n).

ANNEXE VIII

ALGORITHMES POUR LA MÉTHODE DE PRÉDICTION AVEC SEUIL MOINS GOURMANDE EN RESSOURCES PHYSIQUES

Algorithme VIII.1 Algorithme de prédiction du CPU pour l'analyse avec la méthode prédictive, avec seuil et base de connaissance

PredireCPU

Entrée : nombre d'appels par seconde : CPS
consommation du CPU du S-CSCF : CPU
valeur prédite du CPS : preditCPS
instant du dernier changement de tendance du CPS : tStart

Sortie : valeur prédite pour le CPU : preditCPU

Constante : nombre de secondes entre l'instant t et celui de la prédiction : TIME_TO_DEPLOY_CORE
coefficient minimum de la régression exponentielle accepté : TOLERANCE_EXPONENTIELLE

```
1. l = calculerLambda(CPS)
2. si l in BDD
3.         a, b = recupererCoefficientDansBDD(l)
4.         preditCPU = a * exp( b*preditCPS )
5. sinon
6.         predictCPU = predireCPU(CPS, CPU, preditCPS) % algorithme VI.5 pour la
   prédiction du CPU sans BDD
7. return preditCPU
```

La seule fonction supplémentaire avec cette méthode est celle du calcul de lambda décrit par l'algorithme VIII.2.

Algorithme VIII.2 Algorithme de calcul de lambda

CalculerLambda

Entrée : nombre d'appels par seconde : CPS
instant du dernier changement de tendance du CPS : tStart

Sortie : lambda

Constante : temps entre deux mesures : TIME_BETWEEN_2_MEASURE

```
1.  debut = tStart
2.  fin = -1
3.  pour i dans range( len(CPS), tStart )
4.          c = compare(CPS[i], CPS[-1]) % algorithme VI.3 de l'annexe VI
5.          si c != 0 et fin == -1
6.                  fin = i
7.          sinon si c != 0
8.                  debut = i
9.                  return (fin-debut)
10. differenceCPS = CPS[fin] – CPS[debut]
11. differenceT = (fin – debut)*TIME_BETWEEN_2_MEASURED
12. si difference == 0
13.         return 0
14. sinon
15.         return differenceCPS / differenceT
```

ANNEXE IX

CODE MATLAB DE LA MÉTHODE SVR

```
function [predictedValues,error] = svr(metric)
% prédiction du CPU consommé par le S-CSCF avec la méthode SVM-R

   %% initialisation des jeux de données pour les différentes phases (configuration, entraînement et prédiction) de la prédiction
   trainSize = round( length(metric) *4/6 );
   testSize = round( (length(metric)-trainSize) /2 );
   tuneSize = length(metric) - trainSize - testSize;

   tune_data.X = metric(1:tuneSize,1);
   tune_data.y = metric(1:tuneSize,2);

   train_data.X = metric(tuneSize+1:tuneSize+trainSize,1);
   train_data.y = metric(tuneSize+1:tuneSize+trainSize,2);

   test_data.X = metric(tuneSize+trainSize+1:tuneSize+trainSize+testSize,1);
   test_data.y = metric(tuneSize+trainSize+1:tuneSize+trainSize+testSize,2);

   %% phase de configuration (calcul des paramètres)
   optparam.t = 2;  % noyau RBF
   optparam.s = 3;  % epsilon de la méthode SVR
   [optparam.C,optparam.g,optparam.e] = svmtune(tune_data,optparam.s,optparam.t);

   %% phase d'entraînement (détermination du modèle)
   optparam.libsvm = ['-s ', num2str(optparam.s), ' -t ', num2str(optparam.t), ...
                      ' -c ', num2str(optparam.C), ' -g ', num2str(optparam.g), ...
```

```
                    ' -p ', num2str(optparam.e)];
  model = svmtrain(train_data.y, train_data.X, optparam.libsvm);

  %% phase de prédiction
  predict.X = test_data.X;
  y = ones(length(predict.X), 1);
  predict.y = svmpredict(y, predict.X, model);

  %% mise en forme des valeurs prédites calculées
  predictedValues = zeros(length(metric),1);
  predictedValues(1:tuneSize,1) = NaN;
  predictedValues(tuneSize+1:tuneSize+trainSize,1) = NaN;
  predictedValues(tuneSize+trainSize+1:tuneSize+trainSize+length(predict.y),1)
    = predict.y;

  % calcul de l'erreur quadratique moyenne (MAE) pour la prédiction calculée
  error = zeros(length(metric),1);
  error(1:tuneSize,1) = NaN;
  error(tuneSize+1:tuneSize+trainSize,1) = NaN;
  for i = 1:length(predict.y)
     j = i + tuneSize + trainSize;
     error(j) = abs(test_data.y(i) - predict.y(i));
  end
end
```

ANNEXE X

CODE MATLAB DES MÉTHODES LS-SVM ET DLS-SVM

```
function [predictedValues, error] = ls_svm(metric)
% prédiction du CPU consommé par le S-CSCF avec la méthode LS-SVM et DLS-SVM (en function du jeu de données en entrée)

  %% initialisation des jeux de données pour les différentes phases (configuration, entraînement et prédiction) de la prédiction
  trainSize = round( length(metric) *4/6 );
  testSize = round( (length(metric)-trainSize) /2 );
  tuneSize = length(metric) - trainSize - testSize;

  tune_data.X = metric(1:tuneSize,1);
  tune_data.y = metric(1:tuneSize,2);

  train_data.X = metric(tuneSize+1:tuneSize+trainSize,1);
  train_data.y = metric(tuneSize+1:tuneSize+trainSize,2);

  test_data.X = metric(tuneSize+trainSize+1:tuneSize+trainSize+testSize,1);
  test_data.y = metric(tuneSize+trainSize+1:tuneSize+trainSize+testSize,2);

  %% phase de configuration (recherche des paramètres)
  type='function estimation';
  [gam,sig2]=
          tunelssvm({tune_data.X,tune_data.y,type,[],[],'RBF_kernel'},
                                                  'simplex','leaveoneoutlssvm',{'mse'});
```

```
   %% phase d'entraînement (détermination du modèle)
   %% en entrée : les matrices de données pour la phase d'entraînement, le type de prédiction
         voulue ('function estimation' ('f') or 'classifier' ('c')) et les paramètres précédemment
         déterminés. Le noyau utilisé par défaut est RBF.
   model = trainlssvm({train_data.X, train_data.y, type, gam, sig2});

   %% phase de prédiction
   predict.X = test_data.X;
   predict.y = simlssvm(model, predict.X);

   %% mise en forme des valeurs prédites calculées
   predictedValues = zeros(length(metric),1);
   predictedValues(1:tuneSize,1) = NaN;
   predictedValues(tuneSize+1:tuneSize+trainSize,1) = NaN;
   predictedValues(tuneSize+trainSize+1:tuneSize+trainSize+length(predict.y),1)
     = predict.y;
   % calcul de l'erreur quadratique moyenne (MAE) pour la prédiction calculée
   error = zeros(length(metric),1);
   error(1:tuneSize,1) = NaN;
   error(tuneSize+1:tuneSize+trainSize,1) = NaN;
   for i = 1:length(predict.y)
      j = i + tuneSize + trainSize;
      error(j) = abs(test_data.y(i) - predict.y(i));
   end
end
```

ANNEXE XI

ALGORITHMES POUR LA MÉTHODE DE PRÉDICTION SVM

L'algorithme XI.1 nous permet d'exécuter les scripts des méthodes SVM testées pour la prédiction de la consommation du CPU par le conteneur S-CSCF du système IMS virtualisé. Il permet également de regrouper ces valeurs dans une matrice pour les comparer entre elles. Pour cela, l'algorithme récupère aussi les valeurs prédites par MAA et il calcule les erreurs de prédiction faites par toutes les méthodes testées. Pour le calcul de l'erreur, le script calcule l'erreur absolue entre la valeur prédite du CPU consommé et celle relevée lors de la simulation.

Algorithme XI.1 Algorithme d'exécution des méthodes de SVM sélectionnées

ExecutionMethodesSVM

Entrée : jeu de données : dataset

Sortie : matrice des valeurs prédites : P
matrice des erreurs de prédiction : E

Constante : taille du sous jeu de données pour la prédiction : SPLIT
délai entre deux exécutions de la boucle de prédiction : STEP

```
1   % lecture du nombre de lignes dans le jeu de données
2   nbRow = getNbRow(dataset)
3   % extraction de la colonne du temps du jeu de données
4   T = getColumnSecond(dataset)
5   % extraction de la colonne du CPU consommé du jeu de données
6   CPU = getColumnCPU(dataset)
7   % récupération de la prédiction du CPU faite par MAA
8   predictMAA = getColumnPredict(dataset)
9   % récupération de l'erreur de la prédiction du CPU consommé faite par MAA
10  errorMAA = getColumnError(dataset)
```

```
11  % initialisation de la matrice qui regroupe les prédictions du CPU consommé
    par le S-CSCF faites par les différentes méthodes SVM testées et par MAA
12  P = initializeP(predictMAA)
13  % initialisation de la matrice qui regroupe les erreurs de prédictions du CPU
    consommé faites par les différentes méthodes SVM testées et par MAA
14  E = initializeE(errorMAA)
15  % boucle d'exécution des méthodes SVM testées pour prédire le CPU
    consommé, les méthodes sont exécutées toutes les STEP secondes
16  pour i = 1 : STEP : nbRow
17      % construction du sous-jeu de test de taille SPLIT
18      subdata = buildSubData(T,CPU,SPLIT)
19      % exécution de la méthode SVR
20      pSVR, eSVR = svr(subdata)
21      % exécution de la méthode LS-SVM
22      pLSSVM, eLSSVM = ls_svm([T,CPU])
23      % exécution de la méthode DLS-SVM
24      pDLSSVM, eDLSSVM = ls_svm(subdata)
25      % ajout des valeurs prédites par les méthodes SVM à la matrice P
26      insertPredictResult(P, pSVR, pLSSVM, pDLSSVM)
27      % calcul et ajout des erreurs de prédiction des méthodes SVM
28      insertErrorResult(E, eSVR, eLSSVM, eDLSSVM)
```

ANNEXE XII

OUTIL D'AUTOMATISATION DES TESTS

Pour lancer une simulation, il faut lancer successivement et dans un ordre précis un certain nombre de processus. L'outil d'automatisation des tests permet de les lancer pour vous dans le bon ordre.

L'outil doit être exécuté à partir de l'ordinateur qui héberge le simulateur et les agents SIP car ces derniers ne peuvent pas être lancés à distance par une commande SSH. Pour que l'outil fonctionne, il faut aussi qu'il puisse communiquer en SSH avec la machine qui héberge les conteneurs et l'outil de gestion dynamique de notre réseau ainsi qu'avec la machine sur laquelle est déployé le HSS. Toutes les machines doivent être configurées pour qu'aucun mot de passe ne soit demandé lorsque le script d'automatisation lance les commandes SSH.

Dans l'ordre, l'outil d'automatisation permet le lancement des processus suivants :

1. démarrage de tous les conteneurs
2. lancement du processus de chaque conteneur
3. lancement du HSS qui va se lier à chaque conteneur actif
4. lancement du manager qui attend les agents SIP pour démarrer la simulation
5. lancement de tous les agents SIP
6. lancement de l'outil MAA pour la gestion dynamique du système IMS
7. pause de 1200 secondes parce que c'est le temps que dure nos tests
8. arrêt de notre outil MAA car il n'y a plus de valeur à relever, le test est terminé
9. arrêt du manager qui entraîne l'arrêt des agents SIP
10. arrêt du HSS
11. arrêt du processus de tous les conteneurs

12. arrêt des conteneurs afin d'être sûr que le cache de leur mémoire se vide car nous avons remarqué qu'il pouvait entraîner la fin prématurée de l'exécution d'un processus de type P-CSCF ce qui entraîne l'arrêt de notre réseau
13. exécution de l'outil d'extraction des résultats présentés à l'annexe XIII
14. suppression des fichiers de relevés afin de ne pas gaspiller trop de mémoire si on exécute le test un très grand nombre de fois
15. suppression des fichiers log du HSS, du manager et des agents SIP car la mémoire des machines sur lesquelles ils sont déployés est petite

Pour lancer le manager, il a fallu y apporter quelques modifications. Dans le fichier XML de ce dernier, il faut renseigner le champ *number_test_system* avec le nombre d'agents SIP que vous allez lancer. Ainsi en exécutant la commande *./manager -e,* il démarrera automatiquement quand le nombre d'agents spécifié sera atteint. Il faut également ajouter un paramètre pour lancer MAA afin qu'il s'exécute à son lancement dans le mode qui vous convient. Pour ce faire, ajoutez *auto* dans la ligne de commande pour exécuter l'outil pour être en mode normal de surveillance de votre système ou *config* pour être en mode de configuration afin d'initialiser la base de données avec les valeurs de lambda relevées pendant la simulation.

ANNEXE XIII

OUTIL D'EXTRACTION DES RÉSULTATS

Dans le dossier de l'outil MAA, vous pouvez trouver un dossier *result* dans lequel toutes les valeurs des métriques suivies sont sauvegardées. Dans ce dossier se trouve également un dossier *extractResults* dans lequel se trouve un outil qui permet de regrouper sur une même feuille de calcul les valeurs des métriques voulues. L'outil permet aussi d'afficher un certain nombre d'autres valeurs calculées comme l'erreur entre les valeurs lues et celles prédites.

En ouvrant le fichier *parameters.py*, vous pouvez paramétrer l'outil en renseignant le nom des fichiers de relevés des métriques que vous voulez voir apparaître dans la feuille de calcul que vous allez générer. Vous pouvez aussi choisir le nom de la feuille de calcul à créer et, dans le cas où l'information vous intéresse, vous pouvez choisir pour quelle métrique vous souhaitez calculer le coefficient de corrélation. Une fois les paramètres renseignés, il faut ouvrir un terminal pour exécuter le fichier python nommé *extractData.py*. Ce programme permet de regrouper les données des fichiers de relevés des métriques mais aussi de calculer le coefficient de corrélation entre deux métriques et l'écart entre une valeur prédite et une valeur relevée. Ci-dessous, la liste des paramètres à ajouter à la commande pour avoir l'information qui nous intéresse :

- *extract* : pour extraire toutes les valeurs des fichiers de relevés des métriques spécifiées dans *parameters.py* et les regrouper dans une feuille de calcul. Dans la feuille de calcul, une colonne est ajoutée et représente l'erreur entre la valeur mesurée et la valeur prédite pour le CPS et le CPU. Cette option renomme également le fichier log avec le même titre que la feuille de calcul suivi de l'information log.
- *balance* : pour créer un fichier qui regroupe le temps d'exécution et le pourcentage d'erreur de prédiction pour chaque feuille de calcul présente dans le dossier *result*
- *correlation* : pour générer un fichier avec le coefficient de corrélation entre les métriques spécifiées dans le fichier *parameters.py*

Plusieurs options peuvent être appelées simultanément.

BIBLIOGRAPHIE

Adhikari, Ratnadip, and RK Agrawal. 2013. 'An Introductory Study on Time Series Modeling and Forecasting', *arXiv preprint arXiv:1302.6613*.

Ahmed, Mohiuddin, ASMR Chowdhury, Mustaq Ahmed, and Md Mahmudul Hasan Rafee. 2012. 'An advanced survey on cloud computing and state-of-the-art research issues', *IJCSI International Journal of Computer Science Issues*, 9: 1694-0814.

ArcticDesign. 2014. http://linux.arcticdesign.fr/docker-et-ses-conteneurs/.

Armbrust, Michael, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. 2010. 'A view of cloud computing', *Communications of the ACM*, 53: 50-58.

Arumtec. 'Pourquoi virtualiser est GREEN'. http://www.arumtec.net/fr/ressources/faq/q5-pourquoi-virtualiser-est-green.

Avram, Abel. 2013. 'Docker: Automated and Consistent Software Deployments', *InfoQ*.

Balabin, Roman M, and Ekaterina I Lomakina. 2011. 'Support vector machine regression (SVR/LS-SVM)—an alternative to neural networks (ANN) for analytical chemistry? Comparison of nonlinear methods on near infrared (NIR) spectroscopy data', *Analyst*, 136: 1703-12.

Barroso, Luiz André, and Urs Hölzle. 2007. 'The case for energy-proportional computing', *Computer*: 33-37.

Bellavista, Paolo, Giuseppe Cardone, Antonio Corradi, and Luca Foschini. 2012. 'The Future Internet convergence of IMS and ubiquitous smart environments: An IMS-based solution for energy efficiency', *Journal of Network and Computer Applications*, 35: 1203-09.

Bjorkqvist, Mathias, Lydia Y Chen, and Walter Binder. 2012. "Opportunistic service provisioning in the cloud." In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 237-44. IEEE.

Bonnet-Bearstech, Lucas. 'État de l'art des solutions libres de virtualisation pour une petite entreprise'.

Brown. 2013. 'IMS/VoLTE/RCSe...'. http://www.brown-iposs.com/en/2g3g4g/ims.php.

Callow, Brett. 2000. 'La Virtualisation'.

Chaisiri, Sivadon, Bu-Sung Lee, and Dusit Niyato. 2012. 'Optimization of resource provisioning cost in cloud computing', *Services Computing, IEEE Transactions on*, 5: 164-77.

Chang, Chih-Chung, and Chih-Jen Lin. 2011. 'LIBSVM: A library for support vector machines', *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2: 27.

Chee, Brian JS, and Curtis Franklin Jr. 2010. *Cloud computing: technologies and strategies of the ubiquitous data center* (CRC Press).

Cherkassky. 2011. 'EE4389/8591: Support Vector Machine Regression'.

Chiosi, Margaret; Clarke, Don; Willis, Peter; Feger, James; Bugenhagen, Michael; Khan, Waqar; Fargano, Michael; Chen, Clark; Huang, Jinri; Benitez, Javier; Michel, Uwe; Damker, Herbert; Ogaki, Kenichi; Fukui, Masaki; Shimano, Katsushiro; Delisle, Dominique; Loudier, Quentin; Kolias, Christos; Guardini, Ivano; Demaria, Elena; Lopez, Diego; Ruhl, Frank; Sen, Prodip. 2012. "Network Functions Virtualisation." In. Darmstadt-Germany.

Combs, Gerald. 'tshark(1) - Linux man page'. http://linux.die.net/man/1/tshark.

De Brabanter, K, P Karsmakers, F Ojeda, C Alzate, J De Brabanter, K Pelckmans, B De Moor, J Vandewalle, and JAK Suykens. 2011. 'LS-SVMlab toolbox user's guide', *ESAT-SISTA Technical Report*: 10-146.

Dillon, Tharam, Chen Wu, and Elizabeth Chang. 2010. "Cloud computing: issues and challenges." In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, 27-33. Ieee.

Dinda, Peter A;O'Hallaron, David R. 2000. 'Host load prediction using linear models', *Cluster Computing*, 3: 265-80.

Djawida, Dib. 2010. 'Migration dynamique d'applications réparties virtualisées dans les fédérations d'infrastructures distribuées'.

Ericsson, AB. 2007. 'Introduction to IMS', *White Paper, maaliskuu*.

Fakhfakh, Malek, Omar Cherkaoui, Imen Limam Bedhiaf, and Mounir Frikha. 2009. "High availability in IMS virtualized network." In *Communications and Networking, 2009. ComNet 2009. First International Conference on*, 1-6. IEEE.

Fan, Yugang, Ping Li, and Zhihuan Song. 2006. "Dynamic least squares support vector machine." In *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, 4886-89. IEEE.

Frey, Stefan, C Reich, and C Lüthje. 2013. "Key performance indicators for cloud computing SLAs." In *The Fifth International Conference on Emerging Network Intelligence, EMERGING*, 60-64.

Gadhgadhi, Ridha. 2013. 'Openicra: vers un modèle générique de déploiement automatisé des applications dans le nuage informatique', École de technologie supérieure.

Gong, Zhenhuan, Xiaohui Gu, and John Wilkes. 2010. "Press: Predictive elastic resource scaling for cloud systems." In *Network and Service Management (CNSM), 2010 International Conference on*, 9-16. IEEE.

Gourong, Barisaux;. 2010. 'Le protocole SIP'. http://wapiti.telecom-lille1.eu/commun/ens/peda/options/st/rio/pub/exposes/exposesser2010-ttnfa2011/barisaux-gourong/SIP.html.

Grangeversanne, Yohann; Mezille, Jérôme; Isorce, Philippe. 'A Way To IMS'. http://www.ims-way.com.

Grobauer, Bernd, Tobias Walloschek, and Elmar Stöcker. 2011. 'Understanding cloud computing vulnerabilities', *Security & privacy, IEEE*, 9: 50-57.

Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. 'The WEKA data mining software: an update', *ACM SIGKDD explorations newsletter*, 11: 10-18.

Hamilton, James. 2009. "Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for internet-scale services." In *Conference on Innovative Data Systems Research (CIDR'09)(January 2009)*.

Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin. 2003. "A practical guide to support vector classification." In.

Huang, Peijie, Dashu Ye, Ziwei Fan, Peisen Huang, and Xuezhen Li. 2015. "Discriminative Model for Google Host Load Prediction with Rich Feature Set." In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, 1193-96. IEEE.

InterLinkNetworks. 2002. "Introduction to Diameter." In.

J.A.K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, J. Vandewalle. 2002. *Least Squares Support Vector Machines* (World Scientific: Singapore).

Jiang, Jing, Jie Lu, Guangquan Zhang, and Guodong Long. 2013. "Optimal cloud resource auto-scaling for web applications." In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, 58-65. IEEE.

Jiang, Yexi, Chang-Shing Perng, Tao Li, and Rong N Chang. 2013. 'Cloud analytics for capacity planning and instant vm provisioning', *Network and Service Management, IEEE Transactions on*, 10: 312-25.

Juhasz, Zoltan, Peter Kacsuk, and Dieter Kranzlmüller. 2004. *Distributed and Parallel Systems: cluster and grid computing* (Springer Science & Business Media).

Kalman, Rudolph Emil. 1960. 'A new approach to linear filtering and prediction problems', *Journal of Fluids Engineering*, 82: 35-45.

Kephart, Jeffrey O, and David M Chess. 2003. 'The vision of autonomic computing', *Computer*, 36: 41-50.

Khan, Arijit, Xifeng Yan, Shu Tao, and Nikos Anerousis. 2012. "Workload characterization and prediction in the cloud: A multiple time series approach." In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, 1287-94. IEEE.

Lee, Teck Kiong, Teck Yoong Chai, Lek Heng Ngoh, Xu Shao, Joseph Chee Ming Teo, and Luying Zhou. 2009. "An IMS-based testbed for real-time services integration and orchestration." In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, 260-66. IEEE.

Liao, Lingxia, Victor CM Leung, and Min Chen. 2014. 'Virtualizing IMS Core and Its Performance Analysis.' in, *Cloud Computing* (Springer).

Lu, Feng, Hao Pan, Xiao Lei, Xiaofei Liao, and Hai Jin. 2013. "A Virtualization-Based Cloud Infrastructure for IMS Core Network." In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, 25-32. IEEE.

'LXC Web Panel'. 2013. http://lxc-webpanel.github.io/index.html.

'Manage Engine'. http://www.manageengine.com/.

Mao, Ming, Jie Li, and Marty Humphrey. 2010. "Cloud auto-scaling with deadline and budget constraints." In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, 41-48. IEEE.

Marinos, Alexandros, and Gerard Briscoe. 2009. 'Community cloud computing.' in, *Cloud Computing* (Springer).

Mell, Peter, and Tim Grance. 2011. 'The NIST definition of cloud computing'.

Menascé, Daniel A. 2005. "Virtualization: Concepts, applications, and performance modeling." In *Int. CMG Conference*, 407-14.

Müller, Klaus-Robert, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. 2001. 'An introduction to kernel-based learning algorithms', *Neural Networks, IEEE Transactions on*, 12: 181-201.

Nemati, Hani; Singhvi, Arjun; Kara, Nadjia; El Barachi, May. 2014. "Adaptative SLA-based Elasticity Management Algorithms for a Virtualized IP Multimedia Subsystem." In.

OpenSourceIms. 'OpenIMS Core'. http://www.openimscore.org/.

OpenStack. 'Ceilometer', Accessed 3-février-2015. https://wiki.openstack.org/wiki/Ceilometer.

Oracle. *Administrator's Solutions Guide for Release 6*.

Oumina, Hanane, and Daniel Ranc. 2007. "Towards a real time charging framework for complex applications in 3GPP IP multimedia system (IMS) environment." In *Next Generation Mobile Applications, Services and Technologies, 2007. NGMAST'07. The 2007 International Conference on*, 145-50. IEEE.

Ozçelebi, Tanır, Igor Radovanovic, and Johan Lukkien. 2007. "Real-Time Resource Availability Signalling in IMS-Based Networks." In *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, 6083-86. IEEE.

Patel, Pankesh, Ajith H Ranabahu, and Amit P Sheth. 2009. 'Service level agreement in cloud computing'.

Patil, Ashwini, and HK Sawant. 2012. 'Technical Specification Group Services and System Aspects, IP Multimedia Subsystem (IMS)', *IJECCE*, 3: 234-38.

Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V., B. and Grisel and Thirion, O. and Blondel, M. and Prettenhofer, P., R. and Dubourg and Weiss, V. and Vanderplas, J. and Passos, A. and, and D. and Brucher Cournapeau, M. and Perrot, M. and Duchesnay, E. 2011. 'Scikit-learn: Machine Learning in Python', *Journal of Machine Learning Research*, 12: 2825--30.

Poikselkä, Miikka, Harri Holma, Jukka Hongisto, Juha Kallio, and Antti Toskala. 2012. *Voice over LTE (VoLTE)* (John Wiley & Sons).

Popek, Gerald J, and Robert P Goldberg. 1974. 'Formal requirements for virtualizable third generation architectures', *Communications of the ACM*, 17: 412-21.

Powers, Rob, Moises Goldszmidt, and Ira Cohen. 2005. "Short term performance forecasting in enterprise systems." In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 801-07. ACM.

Rosenblum, Mendel, and Tal Garfinkel. 2005. 'Virtual machine monitors: Current technology and future trends', *Computer*, 38: 39-47.

Roy, Nilabja, Abhishek Dubey, and Aniruddha Gokhale. 2011. "Efficient autoscaling in the cloud using predictive models for workload forecasting." In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 500-07. IEEE.

Sadiku, Matthew NO, Sarhan M Musa, and Omonowo D Momoh. 2014. 'Cloud computing: Opportunities and challenges', *Potentials, IEEE*, 33: 34-36.

Sahoo, Jyotiprakash, Subasish Mohapatra, and Radha Lath. 2010. "Virtualization: A survey on concepts, taxonomy and associated security issues." In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, 222-26. IEEE.

Sailer, Reiner, Trent Jaeger, Enriquillo Valdez, Ramon Caceres, Ronald Perez, Stefan Berger, John Linwood Griffin, and Leendert Van Doorn. 2005. "Building a MAC-based security architecture for the Xen open-source hypervisor." In *Computer security applications conference, 21st Annual*, 10 pp.-285. IEEE.

Sailer, Reiner, Enriquillo Valdez, Trent Jaeger, Ronald Perez, Leendert Van Doorn, John Linwood Griffin, Stefan Berger, Reiner Sailer, Enriquillo Valdez, and Trent Jaeger. 2005. 'sHype: Secure hypervisor approach to trusted virtualized systems', *Techn. Rep. RC23511*.

Sanou, Brahima. 2013. "Le monde en 2013 : Données et chiffres concernant les TIC." In.: ITU.

Santy, François. 2013. "La virtualisation." In.: Université libre de Bruxelles.

Schölkopf, Bernhard, and Alexander J Smola. 2002. *Learning with kernels: Support vector machines, regularization, optimization, and beyond* (MIT press).

Schrater, Paul. 2002. "Regression Part II." In.

Silvestri, Luca. 2014. 'SERVICE LEVEL PROVISIONING IN CLOUD SYSTEMS: MODELS, ALGORITHMS AND ARCHITECTURES'.

Smola, Alex, and Vladimir Vapnik. 1997. 'Support vector regression machines', *Advances in neural information processing systems*, 9: 155-61.

Sotomayor, Borja, Rubén S Montero, Ignacio M Llorente, and Ian Foster. 2009. 'Virtual infrastructure management in private and hybrid clouds', *Internet computing, IEEE*, 13: 14-22.

Studnia, Ivan, Eric Alata, Yves Deswarte, Mohamed Kaaniche, and Vincent Nicomette. 2012. "Survey of security problems in cloud computing virtual machines." In *Computer and Electronics Security Applications Rendez-vous (C&ESAR 2012). Cloud and security: threat or opportunity*, p. 61-74.

Tharan, Olivier. 2004. "Architecture réseaux." In.: Institut Pasteur.

Triki, Nizar. 2013. 'Gestion de ressources dans les infrastructures de virtualisation de réseaux', École de technologie supérieure.

Umair, Muhammad. 2013. 'Performance Evaluation and Elastic Scaling of an IP Multimedia Subsystem Implemented in a Cloud'.

Üstün, Bülent. 2003. 'A comparison of support vector machines and partial least squares regression on spectral data', *Department of Analytical Chemistry*.

VMWare. 2012. edited by virtualize-why-choose-hybrid-cloud-dg-en-full2.

Whitle, Peter. 1951. *Hypothesis testing in time series analysis* (Almqvist & Wiksells).

Ye, Jieping, and Tao Xiong. 2007. "SVM versus least squares SVM." In *International Conference on Artificial Intelligence and Statistics*, 644-51.

Yegulalp, Serdar. 2014. 'Docker on course for next big frontier: Container monitoring', Accessed 3-février-2015. http://www.infoworld.com/article/2847718/application-virtualization/container-monitoring-is-dockers-next-big-frontier.html.

Zahariev, Alexander. 2009. 'Google app engine', *Helsinki University of Technology*.

Zareian, Saeed, Rodrigo Veleda, Marin Litoiu, Mark Shtern, Hamoun Ghanbari, and Manish Garg. 2015. "K-Feed-A Data-Oriented Approach to Application Performance Management in Cloud." In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, 1045-48. IEEE.

Zhang, Qi, Lu Cheng, and Raouf Boutaba. 2010. 'Cloud computing: state-of-the-art and research challenges', *Journal of internet services and applications*, 1: 7-18.

Zhang, Yuanyuan, Wei Sun, and Yasushi Inoguchi. 2006. "CPU load predictions on the computational grid*." In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, 321-26. IEEE.

Zhiqun, Xu, Chen Duan, Hu Zhiyuan, and Sun Qunying. 2013. 'Emerging of telco cloud', *Communications, China*, 10: 79-85.