Une approche pour la prise de décision d'adaptation à base de logique floue pour les SaaS Composites

Rima Grati[1], **Khouloud Boukadi**[1], **Hanene Ben Abdallah**[2,1]

1. Laboratoire Miracl, Université de Sfax
Faculté des sciences économiques et de gestion de Sfax, Tunisie
(rima.grati ; Khouloud.boukadi) @gmail.com

2. King Abdulaziz University
Faculty of Computing and Information Technology, Jeddah 23218, Arabie Saoudite
hbenabdallah@kau.edu.sa

RÉSUMÉ. Le logiciel en tant que service (SaaS) constitue un modèle de prestation dans lequel les applications sont hébergées et gérées dans le cloud. Pour offrir un SaaS avec des fonctions flexibles à faible coût, le concept de SaaS composite a été introduit comme une combinaison de Cloud et de services Web. Fonctionnant dans un environnement dynamique et volatil tel que le Cloud, un SaaS composite a des paramètres QoS (Quality of Service*) très variables. Par conséquent, la surveillance et l'adaptation de SaaS composite s'avèrent d'une grande importance pour garantir les paramètres QoS définis dans le contrat SLA (*Service Level Agreement*). Dans ce contexte, cet article propose trois systèmes flous pour les SaaS auto-adaptatifs, qui visent à prévenir les violations du SLA des SaaS composites. Ces algorithmes ont le mérite d'affronter trois problèmes importants : la dépendance des paramètres de QoS (par ex, la disponibilité) des paramètres bas niveau (ex. temps de disponibilité) ; la sélection des stratégies d'adaptation du service en fonction des couches métier et infrastructure informatique (IT) ; et la minimisation de l'impact et du coût des changements. L'article présente les résultats expérimentaux de ces trois algorithmes flous, qui montrent leur efficacité.*

ABSTRACT. Software as a Service (SaaS) is increasingly being adopted as a provision model where applications are hosted in a Cloud computing environment. To offer a SaaS with flexible functions at a low cost, the concept of composite SaaS was introduced as a combination of Cloud and Web services. Operating in a dynamic and volatile environment like the Cloud, a composite SaaS has very variable Quality of Service (QoS) parameters. Consequently, monitoring and adapting composite SaaS are vital means to guarantee QoS parameters defined in the Service Level Agreement (SLA). In this context, this paper proposes three Fuzzy algorithms for self-adaptive SaaS to prevent SLA violations for composite SaaS. The proposed algorithms have the merit of confronting three important issues, namely: the dependence of QoS parameters (e.g. availability) on low level metrics (e.g. uptime, downtime); the selection of service adaptation strategies that can cover both the business and

IT infrastructure layers and have different time and computation complexity; and minimization of the cost and impact of changes. We present results of experiments that we have conducted to evaluate the work. These experimental results show the effectiveness of the fuzzy systems.

MOTS-CLÉS : Cloud, SaaS composite, adaptation, système fuzzy.

KEYWORDS: Cloud, composite SaaS, adaptation, fuzzy system.

DOI:10.3166/ISI.22.4.77-106 © 2017 Lavoisier

1. Introduction

L'adoption et la réussite des services composite (appelés aussi SaaS composite) dans un environnement Cloud sont conditionnées par une exécution correcte de ces services et un respect du SLA (*Service Level Agreement*) agréé entre le client et le fournisseur. Cependant, dans le Cloud, les services sont soumis à des variations de charge qui risquent de modifier les conditions initialement posées par les fournisseurs pour formuler leurs offres. Ainsi, il est indispensable de doter ces services par des mécanismes d'adaptation dynamique qui leur permettent de s'adapter eux même au changement de leurs contextes d'exécution. En agissant ainsi, ces services évitent aux fournisseurs de payer des remboursements monétaires dus à la violation des SLA et permettent au client de s'assurer que les services exécutés ont respecté le contrat agréé.

L'adaptation dynamique des services dans le Cloud vise à modifier le service en fonction de l'environnement dans lequel il s'exécute. Ainsi, un service auto-adaptatif évolue et modifie son propre comportement quand il n'a atteint pas ses objectifs, en se concentrant généralement sur le niveau applicatif et aussi le niveau middleware.

Etant donné qu'un SaaS composite est né d'une combinaison entre le Cloud et le service Web, il hérite les concepts proposés par ces derniers et rajoute ses propres spécificités. En effet, de la technologie service Web, il hérite les concepts liés aux relations client-fournisseur, aux services abstraits/concrets, à la composition des services, etc. Du paradigme Cloud, il hérite les concepts de couche, l'élasticité des ressources et le paiement à l'usage, etc.

Jusqu'à présent, les travaux existants focalisent principalement sur la sélection des services qui proposent des paramètres de qualité de service plus élevés, par exemple un service avec le moindre temps d'exécution ou le moindre coût, etc. et examinent uniquement les cas où les niveaux de qualité ont baissé.

Nous nous intéressons particulièrement dans ce travail de recherche au SaaS composite implémentant des processus de longue durée d'exécution. Ces processus impliquent souvent de longues simulations, des calculs à grande échelle et la manipulation massive des données. Ils requièrent, pour leurs exécutions, des puissances de calcul élevées et la disponibilité de grandes infrastructures informatiques.

Aujourd'hui, les SaaS composites exécutées dans un environnement dynamique et volatil tel que le Cloud sont sujets à des modifications fréquentes de leurs paramètres de qualité de service par rapport à ceux s'exécutant dans un environnement classique. Par conséquent, la surveillance et l'adaptation des SaaS composites est d'une grande importance pour garantir les paramètres QoS (*Quality of Service*) définis dans le SLA. C'est dans ce contexte que nous essayons dans ce travail de répondre aux questions de recherche suivantes.

1.1. Comment assurer une adaptation transverse?

Dans un environnement Cloud, la qualité d'un service offert dépend fortement de la qualité de l'infrastructure informatique, sur laquelle il s'exécute. Le SaaS composite peut donc essayer d'améliorer la qualité du service offert si des ressources suffisantes étaient disponibles. Cette considération implique qu'un SaaS doit être en mesure de comprendre son contexte d'exécution et de réagir à divers changements que ce soit au niveau des besoins de l'utilisateur (telle que la substitution d'un service avec un autre plus performant ou moins cher) et le contexte des ressources (par exemple, une ressource chargée qui nécessite une augmentation des ressources exploitées).

Ces modifications doivent être capturées, évaluées et des actions d'adaptation appropriées doivent être prises en conséquence. Plusieurs types d'adaptations peuvent être prises en compte, notamment des adaptations métier et d'autres de type infrastructure (appelée aussi adaptation IT). Bien que l'adaptation de l'infrastructure, celle de la plateforme ou celle du service (métier) poursuivent des objectifs différents et manipulent des entités distinctes, chacune a un impact sur le système et cet impact peut avoir des conséquences sur la qualité du service offert. Par exemple, il est possible que l'infrastructure libère une ressource particulière appartenant au centre de données et devra ainsi migrer toutes les plateformes/services qui s'exécutent sur cette ressource. Ce changement de localisation bien qu'il n'ait pas d'impact sur le fonctionnement des services peut tout de même dégrader la qualité du service offert. C'est la raison pour laquelle nous pensons qu'il faut voir l'adaptation comme une problématique transverse afin de coordonner l'adaptation de chacun des niveaux et même prévoir de l'adaptation entre plusieurs niveaux. Nous nous limitons dans le cadre de ce travail de recherche aux deux types d'adaptation à savoir : métier et IT.

Pour mettre en œuvre une adaptation transverse, il est nécessaire d'avoir un système d'adaptation pour chaque niveau et des interactions bien définies qui régissent les collaborations possibles entre les systèmes d'adaptation. Ces systèmes doivent veiller au respect du SLA des deux niveaux. En effet, généralement il s'agit de deux types de SLA : le SLA-SaaS et le SLA-IaaS. Le premier consiste à un contrat entre le client final et le fournisseur de SaaS. Il spécifie un ensemble d'objectifs de niveau de service SaaS en termes de performance et disponibilité. Tandis que le second concerne le contrat entre le fournisseur SaaS et le fournisseur

IaaS. Il indique un objectif de niveau de service IaaS en termes de disponibilité et de capacité des machines virtuelles.

Cependant, les résultats et les orientations dans ce domaine sont encore insuffisants. Les systèmes d'adaptation focalisent généralement sur un seul niveau à savoir le niveau métier (substitution d'un service par un autre afin de veiller au respect du SLA-SaaS) ou encore IT.

1.2. Choix de la ou des stratégie (s) d'adaptation à appliquer

En fonction des variations du contexte (contexte des ressources au niveau infrastructure ou dégradation des paramètres de qualité du niveau métier), il existe un ensemble de stratégies d'adaptation possible. Néanmoins, le choix de la stratégie d'adaptation la plus appropriée est une tâche complexe en raison du nombre de critères qui interviennent dans le processus de prise de décision (Raman *et al.*, 2009). En outre, ces stratégies peuvent couvrir à la fois le niveau métier et le niveau infrastructure simultanément et ont différents coût, complexité et temps d'exécution.

Dans la littérature, les travaux considèrent uniquement la substitution du service élémentaire lorsque la qualité du service se dégrade. Notre approche examine également le fait que l'amélioration d'un paramètre de qualité dans un certain point de l'exécution peut être considérée pour améliorer d'autres paramètres en ajustant les poids pendant la phase de sélection du service. Par ailleurs, l'adaptation de l'infrastructure doit garantir le contrat établi en prenant en considération toutes les variations de charge des ressources sur lesquelles les SaaS composites sont déployés. Des exemples d'action d'adaptation d'infrastructure peuvent être d'augmenter ou de diminuer les ressources de certaines machines ou de migrer une machine virtuelle vers une autre machine physique. Ces actions doivent être conduites pour éviter des violations du SLA et doivent être entreprises tout en considérant aussi les adaptations du niveau métier : un des fondements de base de l'adaptation transverse. En effet, pour réaliser une adaptation transverse, il est indispensable de bien étudier et analyser les variations du contexte. Cependant les informations de contexte, sont sujettes à beaucoup d'incertitude et sont souvent considérées comme difficilement modélisable. En effet, l'interprétation reste une mesure subjective et l'évaluation des variations du contexte est coûteuse et fastidieuse. Par conséquent, la conception d'un système qui peut quantifier objectivement les informations du contexte que ce soit du service ou de la machine sur laquelle il s'exécute, avec une précision raisonnable et en temps réel est devenue un besoin primordial qui constitue un challenge intéressant à résoudre. La logique floue apporte des solutions intéressantes aux problèmes déjà soulignés. En effet, la logique floue a été appliquée dans de nombreux domaines pour la résolution de problèmes et la prise de décision où des relations entre les paramètres existants sont trop compliquées à modéliser par la logique mathématique classique. La logique floue, en se basant sur la notion d'ensemble flou, permet d'une part, de modéliser l'incertitude et l'imprécision de certaine information du contexte comme par

exemple la disponibilité du service. Et d'autre part, de dresser des relations d'indépendance entre les informations de contexte et aussi entre ces informations et l'utilité de l'adaptation que ce soit au niveau métier ou IT.

Etant la multitude de ces variations ainsi que les possibles relations entre les différents éléments de contexte des deux niveaux, le besoin est fort à un ou éventuellement à plusieurs systèmes d'aide à la décision basés sur la logique floue et capables à un moment donné de décider de la stratégie d'adaptation ainsi que des actions d'adaptation à mettre en place.

1.3. Coût et impact de l'adaptation

Le coût des actions d'adaptation que ce soit du niveau métier ou celui de l'infrastructure ainsi que l'impact du changement doivent être considérés avant de décider de la meilleure action d'adaptation à appliquer. En effet, une action d'adaptation particulière possède un coût d'exécution qui peut être parfois plus élevé que le bénéfice attendu. Par exemple, les actions d'adaptation des ressources devraient soutenir les exigences de qualité des SaaS composites sans encourir des coûts supplémentaires, que ce soit pour le fournisseur Cloud ou le client. De la même manière, le coût d'une substitution de service est également un facteur important pour déterminer la pertinence de la substitution. En outre, le déclenchement des actions d'adaptation à chaque variation du contexte (par exemple une variation de la performance d'une ressource ou d'un service élémentaire) n'est pas toujours la meilleure des solutions. Ces différentes constations nous poussent à nous poser un certain nombre de questions :

– Est ce qu'une action d'adaptation est indispensable uniquement dans le niveau métier ou dans le niveau infrastructure ou dans les deux simultanément ?

– Quelle est l'utilité de ces actions d'adaptation par rapport à leurs coûts d'exécution?

1.4. Principales contributions

Afin de répondre aux différentes problématiques déjà soulignées, ce travail propose :

– Une catégorisation de contexte qui facilite les mécanismes d'auto-adaptation des services. Cette catégorisation comporte : le contexte des services élémentaires, le contexte des ressources et le contexte des SaaS composites.

– Un modèle de service composite sensible à son contexte, et en réponse aux changements pertinents, évalue l'utilité de l'adaptation et exécute le ou les actions d'adaptation afin d'éviter les violations de SLA. Ce modèle de service repose en réalité sur trois systèmes de logique floue : deux systèmes de logique floue pour le niveau métier et un système pour le niveau infrastructure. Le premier système évalue les valeurs de qualité de service élémentaires appartenant au service

composite afin d'évaluer la pertinence de l'adaptation au niveau métier (appelée utilité de l'adaptation). Le deuxième est invoqué lorsque l'utilité de l'adaptation est élevée ou moyenne et vise à sélectionner la meilleure action d'adaptation. Quant au troisième système, il guide le processus de prise de décision au niveau infrastructure notamment en terme d'allocation optimale des ressources aux machines virtuelles en fonction de leurs contextes d'exécution et les SLA-IaaS déjà agréés.

– L'implémentation et l'évaluation du modèle de service (SAV-CSaaS) dans un environnement de Cloud.

Le reste de ce document est organisé comme suit. La section 2 décrit les principaux travaux connexes. La section 3 décrit la définition du contexte pour les SaaS composites auto-adaptatifs. La section 4 présente l'architecture globale du système. La section 5 détaille les systèmes flous pour les SaaS composites auto-adaptatifs. Les résultats des expériences concernant l'application des systèmes flous sont présentés à la section 6. La section 7 conclut le document et discute de futures perspectives.

2. Travaux connexes

Pour mieux motiver et placer nos contributions, nous examinons d'abord certains travaux sur l'adaptation niveau métier. Ensuite, nous présentons les travaux les plus cités relatifs à l'adaptation niveau IT.

2.1. Adaptation métier

Dans (Pernici et Siadat, 2011), les auteurs proposent un système d'inférence flou (FIS) pour capturer une QoS globale et choisir des stratégies d'adaptation en fonction des changements de QoS. Ce système repose sur deux moteurs d'inférence Fuzzy : le moteur d'évaluation QoS et le moteur de prise de décision. Le premier est utilisé pour déduire le degré global de QoS. Le second moteur est utilisé pour la prise de décision des stratégies d'adaptation. Trois stratégies d'adaptation sont prises en considération par les auteurs : ne rien faire, renégociation et substitution. L'approche proposée permet de choisir des stratégies d'adaptation en fonction des changements de QoS.

Dans (Aschoff et Zisman, 2012), les auteurs décrivent un *framework* ProAdapt (*Proactive Adaptation*) qui prend en charge l'adaptation de composition de services déclenchées par différents types de problèmes. Ce *framework* permet de remplacer une opération de service ou un groupe d'opérations par une autre opération de service ou un groupe d'opérations dynamiquement composées. L'adaptation proactive de la composition du service se compose de trois étapes : la première étape est l'identification et la prédiction des problèmes tandis que la deuxième étape est l'analyse des problèmes déclenchés par la prédiction suivie par la troisième étape

qui est la décision des actions à prendre face aux problèmes pour exécuter les actions d'adaptation.

Dans (Avila, 2014), l'auteur présente la conception et la mise en œuvre du *framework* pour l'adaptation des services composés basée sur une logique floue. Ce cadre vise à prévenir la dégradation de la QoS et à améliorer les niveaux de QoS d'un service composite. Il utilise deux systèmes d'inférence Fuzzy qui évaluent les valeurs QoS des services composites, basées sur des données historiques afin d'identifier le besoin d'adaptation. Le *framework* se compose de six composantes : *composition engine, adaptation manager, service binder, service selector, predictor et sensors.*

2.2. Adaptation IT

Dans (Yazdanov et Fetzer, 2012), les auteurs décrivent un nouveau contrôleur en ligne pour faciliter l'approvisionnement élastique des applications en combinant l'adaptation des contraintes de ressources des machines virtuelles et la connexion dynamique de nouvelles CPU virtuelles. La mise à l'élastique des services vise à répondre aux exigences de provisionnement des services, à éviter la violation des SLA et à utiliser les ressources disponibles de manière efficace. Les auteurs utilisent deux niveaux de mise à l'échelle : échelle verticale liée aux niveaux VM et hôte et échelle horizontale liée au niveau du centre de données. Le contrôleur d'élasticité proposé utilise un modèle de prédiction pour effectuer une mise à niveau proactive des ressources. La mise à l'échelle horizontale consiste à changer la puissance de la CPU d'un noyau virtuel en étendant la puissance de la VM en branchant de nouvelles CPU à la volée. La mise à l'échelle verticale consiste à allouer des ressources avec une latence plus faible.

Dans (Maurer *et al.*, 2013), les auteurs présentent une approche pour la configuration de ressources adaptative pour la gestion de l'infrastructure Cloud. À cette fin, ils structurent hiérarchiquement toutes les actions d'adaptation possibles appelées niveaux d'escalade qui structurent toutes les possibilités de réaction en différents sous-problèmes en utilisant un modèle hiérarchique. Ensuite, sur la base de l'un de ces niveaux, les données surveillées extraites des machines virtuelles sont analysées afin de prendre des décisions sur la configuration des ressources à l'aide de la gestion des connaissances, du raisonnement basé sur des cas et des approches fondées sur des règles.

Cette approche vise à obtenir des taux bas de violation de SLA et une utilisation élevée des ressources, de sorte que le niveau des ressources allouées mais inutilisées soit aussi faible que possible. Cette approche est conçue et mise en œuvre à l'aide d'un moteur de simulation. Les simulations révèlent la faisabilité de considérer les violations des SLA, l'utilisation des ressources et le nombre de reconfigurations nécessaires.

Dans (Huber *et al.,* 2011), les auteurs présentent une boucle de contrôle générique et auto-adaptable ainsi qu'un algorithme respectif d'allocation de ressources pour les environnements virtualisés basés sur des modèles de performance en ligne. L'utilisation de tels modèles permet de prédire les effets des changements dans les charges de travail des utilisateurs ainsi que les effets des actions de reconfiguration respectives afin d'éviter les violations du SLA ou l'utilisation inefficace des ressources. Les actions de reconfiguration se concentrent sur l'ajout/suppression de CPU virtuelles et l'ajout/suppression de serveurs d'applications à un cluster de serveurs d'applications. Le modèle de boucle de contrôle se compose de quatre phases. La première phase recueille le changement de charge de travail rapporté par les clients ou par des techniques comme la prévision de charge de travail. Pour la phase d'analyse, l'utilisation de la performance du logiciel vise à prédire l'effet des changements et à décider (troisième phase) des actions à entreprendre. Au cours de la quatrième phase, les options de reconfiguration envisagées dans l'analyse sont mises en œuvre.

Les auteurs présentent une définition formelle de l'algorithme d'allocation de ressources proposé. Cet algorithme est composé de deux phases : phase de poussée et phase de traction. La première affecte des ressources supplémentaires jusqu'à ce que les exigences des clients soient satisfaites. La deuxième phase optimise l'efficacité des ressources en libérant des ressources inutilisées.

2.3. La synthèse

Dans cette section, nous avons comparé et analysé les travaux cités trouvés dans la littérature relative à l'adaptation niveau métier et IT.

L'analyse de ces travaux montre que les approches proposées dans (Avila, 2014), (Pernici et Siadat, 2011) et (Aschoff et Zisman, 2012) ont porté sur l'adaptation niveau métier alors que celles proposées dans (Yazdanov et Fetzer, 2012 ; Maurer, Brandic *et al.,* 2013 et Huber *et al.,* 2011) traitent uniquement l'adaptation IT. Nous avons remarqué qu'aucun travail n'a eu l'occasion d'aborder l'adaptation métier et IT à la fois. Cela signifie que la dépendance entre le SLA-SaaS et le SLA-IaaS n'a pas été traitée auparavant à l'exception de notre travail qui est considéré comme une première tentative visant à traiter le problème de dépendance.

La majorité des travaux présentés considèrent soit un but, une situation, soit un contexte de ressources. Les travaux de (Avila, 2014) et (Pernici et Siadat, 2011) sont conscients de la QoS seulement. Le travail de (Aschoff et Zisman, 2012) et (Maurer *et al.,* 2013) met l'accent sur les situations nécessitant une adaptation. Dans (Yazdanov et Fetzer, 2012), les auteurs ne considèrent que le contexte de la ressource. Nous avons essayé de couvrir le contexte de l'environnement de travail (le contexte du service élémentaire, le contexte de la VM, le contexte du SaaS composite) pour proposer des actions d'adaptation appropriées.

Les autres apports de notre approche sont : l'inclusion d'une évaluation de la décision d'adaptation et l'optimisation des QoS. Ces deux critères n'ont pas été pris en compte par les travaux présentés.

3. Définition du contexte pour un SaaS auto-adaptatif

L'étude de la littérature montre que la sensibilité au contexte est devenue un élément central pour la conception et la mise en place des services auto-adaptatifs. En ayant cette sensibilité au contexte, les services traditionnellement très peu informés de leur entourage deviennent plus conscient de l'environnement dans lequel ils évoluent et aussi de l'environnement du demandeur de services. Par ce biais, ils permettent une meilleure flexibilité et renforcent leur capacité à se reconfigurer de manière automatique afin de respecter les exigences du client. Nous proposons dans ce qui suit notre définition du contexte ainsi que les catégories de contexte prises en compte dans le cadre de ce travail.

3.1. Notre vision du contexte

Dans le domaine de la sensibilité au contexte, les auteurs dans la littérature n'ont pas encore proposé une définition à la fois générique et pragmatique de la notion de contexte. Malgré la variété des définitions, la formalisation de la notion de contexte reste difficile. La définition la plus répandue dans la littérature et la plus acceptée par les chercheurs est celle proposée par (Dey *et al.*, 2001). Cette dernière définit le contexte comme « toute information qui caractérise la situation d'une entité. Une entité étant une personne, un lieu ou un objet considéré comme pertinent relativement à une interaction entre un utilisateur et une application, incluant l'utilisateur et l'application eux-mêmes ». Cependant, cette définition trace un espace infini et illimité de ce qui fait partie du contexte. Le contexte comme nous l'avons déjà signalé a surtout été étudié dans le domaine des applications mobiles et de l'informatique diffuse. Nous proposons notre définition du contexte :

« Le contexte est l'ensemble des paramètres qui peuvent appartenir à l'environnement du SaaS (SaaS composite, service élémentaire, machine virtuelle) et qui influencent son comportement en définissant de nouvelles vues sur les fonctionnalités proposées par les services participants. Ces paramètres peuvent être statiques ou dynamiques (c'est-à-dire qui changent durant l'exécution du processus) ».

Cette définition complète celle proposée par (Dey *et al.*, 2001) en apportant plus de précision. En plus, elle appuie la définition de (Winograd, 2001) qui stipule que « la considération d'une information comme contexte est due à la manière dont elle est utilisée et non à ses propriétés inhérentes ».

3.2. Catégories de contexte

Étant donnée la diversité des informations composant le contexte, il est utile d'essayer de les classer par catégories pour faciliter leur utilisation. Notre étude de la littérature a révélé que plusieurs catégories de contexte ont été proposées et étudiées surtout dans le domaine de l'informatique pervasive ou ambiante où la notion de contexte prend beaucoup d'ampleur (Boukadi *et al.,* 2008 ; Xiuguo *et al.*, 2013 ; Yuanping et Feng, 2009). En se basant sur les différentes tentatives de catégorisation de contexte, nous proposons une catégorisation qui détaille les paramètres contextuels pouvant influencer le comportement du SaaS composite.

En considérant les parties prenantes dans un SaaS composite nous identifions trois catégories de base à savoir : le contexte du service élémentaire, le contexte du service composite et le contexte de la machine virtuelle. Ces différentes catégories sont décrites en détail dans ce qui suit (figure 1).

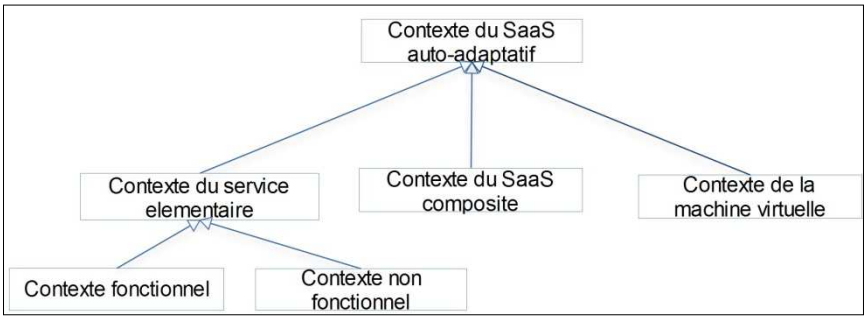

Figure 1. Catégorisation de contexte

– *Contexte du service élémentaire (elementary service related context)* décrit les services participants au SaaS composite. Il comporte deux sous-catégories de base à savoir : le contexte fonctionnel et le contexte non fonctionnel :

- Le *contexte fonctionnel* : décrit la fonctionnalité proposée par un service élémentaire à travers un ensemble d'opérations. Un service élémentaire dans un SaaS composite est un service à fine granularité ayant un fichier WSDL (*Web Service Description Language*) et publié dans le registre de service. Afin d'accomplir efficacement sa fonctionnalité de base, le contexte du service élémentaire regroupe également un ensemble de besoins en termes de ressources comme la mémoire requise, le stockage, etc.

- Le *contexte non fonctionnel* : comprend les paramètres de qualité (QoS) qu'ils soient agréés ou mesurés, la charge de travail du service ainsi que son coût de substitution. Les paramètres de qualités agréés ainsi que le coût de substitution constituent le contexte non fonctionnel statique acquis à partir du document WSLA (*Web Service Level Agreement*) (Alexander et Heiko, 2003). Tandis que les

paramètres de qualités mesurées et la charge du travail forment le contexte non fonctionnel dynamique recueilli par le module appelé SmartListner. Le Smart Listener détecte les changements dynamiques des paramètres de contexte qui surviennent au moment de l'exécution du service et qui sont susceptibles de déclencher des adaptations. En effet, le service élémentaire possède une charge de travail qui fait partie de son contexte non fonctionnel et qui influence largement ses paramètres de qualité. En général, les paramètres de qualité différencient les services élémentaires ayant la même fonctionnalité. Ainsi, les clients peuvent évaluer ces paramètres et sélectionner le service élémentaire à inclure dans leurs schémas de composition. Les paramètres QoS peuvent être quantitatifs (par exemple, temps de réponse, disponibilité, coût, etc.) ou qualitative (par exemple, réputation et sécurité). Dans le cadre de ce travail, nous nous intéressons aux paramètres de QoS quantitatifs suivants : Temps de réponse (*Response Time*), Prix (*Cost*) et Disponibilité (*Availability*).

– *Contexte de la machine virtuelle (Virtual Machine Related Context)* : dans un environnement Cloud, les machines virtuelles (VM) sont utilisées pour exécuter le SaaS composite. Chaque machine virtuelle est responsable de l'exécution d'un service élémentaire. Pour ce faire, elle repose sur ses ressources à savoir : le processeur (CPU en anglais *Central Processing Unit*), la mémoire, le stockage et/ou la bande passante. Le contexte de la VM comprend des paramètres statiques et d'autres dynamiques qui changent au cours de son exécution. Les paramètres statiques relèvent du profil de la VM comme son processeur, la taille de la mémoire et aussi l'espace de stockage qui lui a été réservé. Ces paramètres de profil sont agréés dans un contrat interne au fournisseur Cloud appelé SLA-IaaS. Quant aux paramètres dynamiques, ils concernent les fractions de CPU, de mémoire et d'espace de stockage utilisés par la machine virtuelle à un instant t.

– *Contexte du SaaS composite (composite SaaS related Context)* : tout comme le contexte du service élémentaire, le contexte du service composite distingue deux sous-catégories de contexte : le contexte fonctionnel et le contexte non fonctionnel. Le contexte fonctionnel décrit la fonctionnalité offerte par le SaaS à ses clients à travers un document WSDL. Quant au contexte non fonctionnel, il s'intéresse aux paramètres de qualité qui caractérisent un SaaS composite (agréés dans le SLA-SaaS et mesurés pendant l'exécution) ainsi qu'un ensemble de points de conformité. Ces derniers sont des points dans l'exécution de la composition où une décision d'adaptation peut être prévue. En effet, ils visent à vérifier si l'exécution des services se comporte conformément au document SLA déjà agréé. Le concept de points de conformité est similaire à celui proposé dans (Maurer *et al.,* 2013) et surtout applicable pour les processus de longue durée d'exécution. Nous avons suivi les consignes décrites dans (Maurer *et al.,* 2013) pour définir les points de conformités. Ces dernières stipulent que les points de conformité, très tôt dans l'exécution d'une composition, ont accès à peu d'informations sur l'exécution des services élémentaires, par contre le temps de réaction est élevé. Inversement, les points de conformité, définis très tard dans le schéma de composition, augmentent la

probabilité de la violation du SLA. En général, les points de conformité doivent être conçus de telle sorte que suffisamment d'informations contextuelles sur l'exécution des services est disponible pour générer des prévisions utiles.

4. Architecture d'un SaaS composite auto-adaptatif (SAV-CSaaS)

Les architectures présentées dans les travaux existants pour assurer la sensibilité au contexte accordent une très grande importance à la gestion du contexte sans toutefois présenter comment modifier le comportement d'une application pour qu'elle s'adapte au contexte. Dans notre approche d'adaptation, nous accordons une importance capitale à la fois à la gestion du contexte et à l'adaptation. La gestion de contexte permet la capture, le stockage et la dissémination du contexte. Tandis que l'adaptation, comme son nom l'indique, se préoccupe d'assurer l'adaptation du SaaS composite aux diverses changement du contexte que ce soit du niveau métier ou du niveau infrastructure. Nous définissons une architecture qui aide le SaaS composite à être conscient de son contexte, et en réponse aux changements pertinents, évalue l'utilité de l'adaptation et s'adapte en cas de besoin pour éviter les violations de SLA.

Une vue d'ensemble de l'architecture du SAV-CSaaS est illustrée dans la figure 2. Cette architecture respecte le principe de modularité (elle se compose de plusieurs modules) qui est un principe phare dans le domaine du développement des logiciels. La modularité porte sur le fait de proposer plusieurs modules relativement indépendants les uns des autres et dont chacun met en œuvre une fonctionnalité particulière afin d'assurer l'adaptabilité du service composite. Les modules dédiés à la couche métier sont : *Adaptation Manager, Context Manager, QoS Calculator, SLA-SaaS violation predictor* et *Discovery Manager*. Ceux responsables de la couche infratstructure sont : *Scaling Manager, Resource Monitor* et *SLA-IaaS violation predictor*. Ces différents modules sont décrits dans ce qui suit.

– Le module *« Adaptation Manager »* : l'objectif ultime de ce module est de maintenir la qualité de service agréé au niveau du SLA-SaaS. Pour ce faire, il se base sur les informations capturées par le *Context Manager* et invoque les systèmes de logique floue afin de décider, à chaque point de conformité, si une adaptation est nécessaire au niveau métier. En plus, il réalise l'action d'adaptation suggérée (substitution ou ne rien faire).

– Le module « *Context Manager »* : à l'encontre des gestionnaires de contexte dans les applications mobiles où ils sont définis comme des composants logiciels liés à un ou plusieurs dispositifs ou capteurs, le gestionnaire de contexte que nous proposons est un module qui se base sur d'autres modules à savoir: le *Conformity points Collector*, *Instance Context Collector* et le *Smart Listener*, afin de collecter les paramètres de contexte. Le *Conformity points Collector* et l'*Instance Context Collector* capturent les paramètres du contexte statiques relatifs aux différents services élémentaires et au service composite lui-même. Quant aux paramètres

dynamiques, ils sont surveillés et détectés par le *Smart Listener*. Outre les fonctions de capture du contexte, le *Context Manager* assure la communication avec les autres modules d'adaptation (décrits dans la suite de cette section). Il les informe des informations contextuelles et des changements éventuels du contexte.

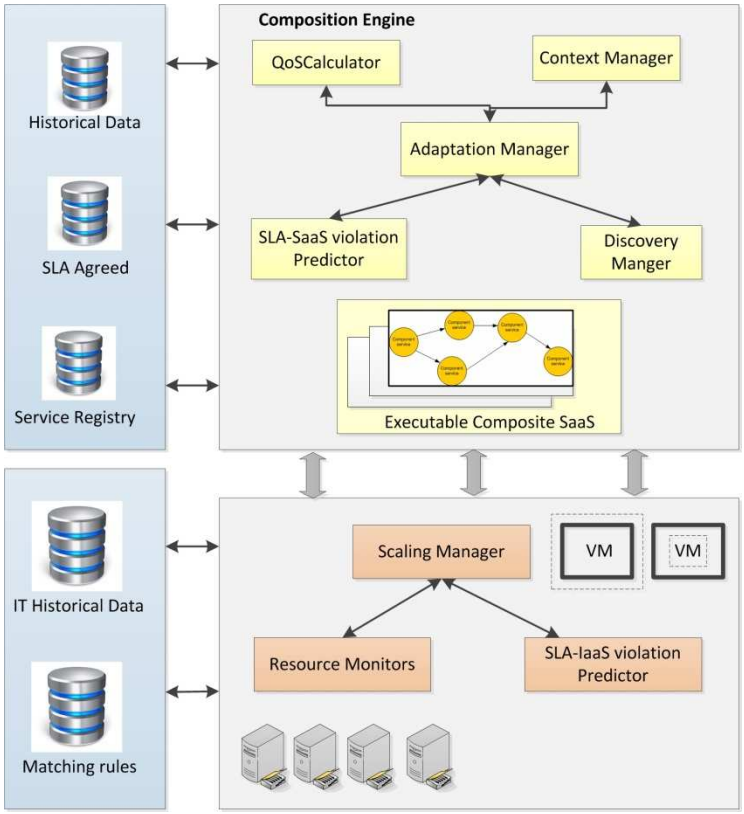

Figure 2. Architecture d'un SAV-CSaaS

– *Instance Context Collector* : sa mission est de collecter les paramètres de contexte statique d'une instance d'un SaaS composite et des services élémentaires qu'ils le composent. Des exemples de paramètres statiques sont le SLA-SaaS agréé, les paramètres de QoS des services élémentaires, etc.

– *Conformity Points Collector* : comme son nom l'indique, son rôle est de collecter les points de conformité d'une instance d'un SaaS composite, une fois sélectionné par le client. Ces points de conformité sont considérés comme les éléments déclencheurs qui font un inventaire sur la performance des services en cours d'exécution et par conséquent aident à activer les actions d'adaptation si

nécessaire *Smart Listener* : techniquement, le *Smart Listener* est l'équivalent d'un thread qui s'exécute en parallèle avec le processus BPEL sans toucher a la logique métier de ce dernier et sans intercepter ou modifier le fonctionnement du moteur d'exécution. Grâce à ses différents composants (*Catch*, *Filter* et *Storage*), il permet d'écouter et de collecter tous les événements générés par le moteur d'exécution, d'analyser et filtrer les événements relatifs aux paramètres de contexte des instances de SaaS composites entrain d'être exécutés et de les stocker comme des objets en mémoire. Le *Smart Listener* est décrit en détail dans (Grati *et al.*, 2012).

– Le module « *QoS Calculator* » : permet le calcul les paramètres QoS. Pour ce faire, il exploite les détails du QoS représentant le SLA-SaaS agréé et invoque les plugins nécessaires afin de calculer les valeurs des paramètres de QoS du service composite. Pour réaliser ce calcul, il considère les branchements utilisés pour connecter les services élémentaires ainsi que les valeurs de paramètres de QoS des services élémentaires (exprimés ou pas par des mesures de bas niveau fournis et convertis par le *Moniteur*).

– Le module « *SLA-SaaS violation predictor* » : utilise les réseaux bayésiens afin de prédire les violations du SLA-SaaS. À cette fin, il utilise les valeurs des paramètres de QoS du chemin exécuté précédant le point de conformité. Ces valeurs sont obtenues au moment de l'exécution à partir du module QoSCalculator et sont utilisées ensuite pour construire le modèle probabiliste qui peut prédire la violation du SLA agréé. La prédiction se base fondamentalement sur les valeurs de QoS de l'exécution en cours et l'exécution passé de chemins similaires stockées dans le référentiel de données historiques (*Historical Data*). Le *SLA-SaaS violation predictor* est implémenté comme celui présenté dans (Leitner, 2011).

– Le module « *Discovery Manager* » : si l'utilité de l'adaptation du niveau métier est élevée et la stratégie d'adaptation de type substitution est maintenue, ce module est invoqué. Il considère le contexte fonctionnel du service à substituer et cherche dans le référentiel des services (*Service Registry*), pour constituer une liste des services candidats. Ensuite, en se basant sur le poids suggéré par la stratégie de substitution (substitution avec poids élevé pour le coût ou temps de réponse, etc.), il sélectionne à partir de la liste des services candidats le service adéquat.

Quant aux modules dédiés à la couche infrastructure sont :

– Le module « *Resource Monitor* » : responsable de la surveillance des métriques des ressources IaaS afin de respecter les objectifs agréés dans le SLA-IaaS. Ce module, respecte la dépendance entre le monitoring des métriques bas niveaux (ressources IaaS) et le monitoring des paramètres SLA-SaaS. En effet, il est considéré comme un élément central vu qu'il mesure les données nécessaires et permet d'établir des tableaux de bord pour observer l'état du centre des données du fournisseur. Le monitor se compose de deux sous-modules, à savoir le « Collector » et le « Converter ». Le *Collector* est intégré dans les différentes ressources appartenant au fournisseur de et responsable de l'extraction des paramètres bas niveaux. Ces ressources peuvent être une machine physique, une machine virtuelle,

un appareil de stockage ou un dispositif réseau. Quant au *Converter*, il permet de convertir les métriques bas niveau en paramètres SLA selon des règles de correspondance stockées dans une base de données.

– Le module « *SLA-IaaS violation predictor* » : tout comme le module *SLA-SaaS violation predictor*, ce module exploite les techniques d'apprentissage automatiques à savoir les réseaux bayésiens afin de proposer des prédictions automatisées du risque de violation du SLA-IaaS.

– Ce module communique avec les deux autres modules à savoir : le *Smart Listner* et le *Monitor* afin de collecter les données essentielles pour la construction du réseau bayésien. En effet, ce réseau dresse les relations existantes entre la charge du service s'exécutant sur la VM, l'état des ressources de la VM (par exemple, CPU, mémoire et stockage) et les paramètres de QoS, tels le temps de réponse, la disponibilité, etc. (voir figure 3) afin de prédire le risque de violation de la SLA-IaaS.

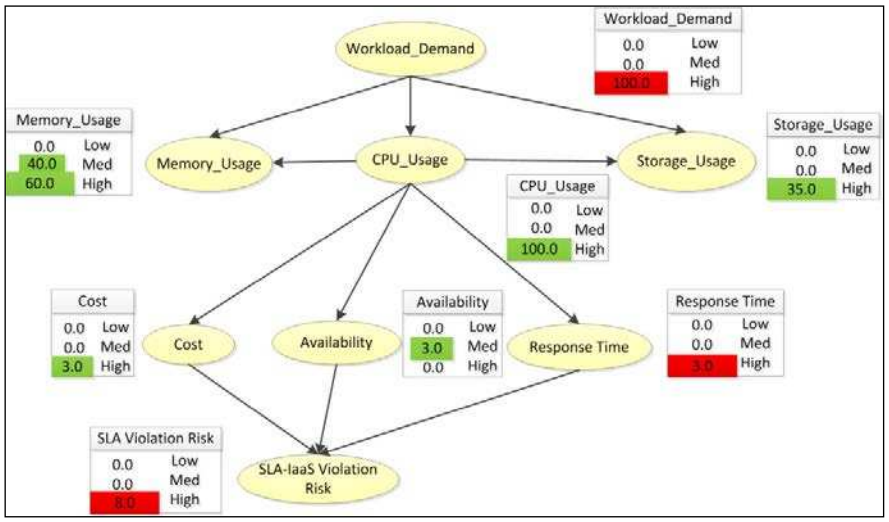

Figure 3. Modèle bayésien pour la détection de la violation du SLA-IaaS

Dans le cadre de notre travail, la construction du réseau suit les étapes déjà définis dans (Bashar, 2013). La première étape consiste à discrétiser les mesures continues en un nombre prédéfini d'états. Ensuite, la phase d'apprentissage implique l'apprentissage structurel en utilisant des algorithmes PNJ (Spirtes *et al.*, 2001) (Steck, 2001) et l'apprentissage de paramètres en utilisant l'algorithme EM (Lauritzen, 1995). Afin de vérifier si le modèle est adapté à la prédiction, une validation du réseau est réalisée à travers le processus de la validation croisée.

– Le Module « *Scaling Manager* » : exécute la stratégie d'adaptation IT suggérée par le système de logique floue du niveau infrastructure. En effet, il peut modifier les capacités des ressources de la VM comme augmenter la mémoire, l'espace de stockage ou encore réduire la mémoire ou l'espace de stockage. Le *Scaling Manager* réalise les adaptation de type *Scale Up/down* des ressources de la VM et ceci en tenant en considération les capacités maximales que la VM peut avoir de la machine physique (mémoire, stockage ou CPU) et en suivant aussi les meilleures pratiques et les simulations définies dans (Maurer *et al.*, 2013).

5. Systèmes de logique floue pour SAV-CSaaS

Comme nous l'avons déjà mentionné auparavant, SAV-CSaaS repose sur des systèmes de logique floue[1], comme des systèmes d'aide à la décision multicritères, pour les adaptations métier et infrastructure. Chacun de ces systèmes propose ses propres variables linguistiques et ses propres règles. Ces différents systèmes sont décrits dans ce qui suit.

5.1. Systèmes de logique floue du niveau métier

Les deux systèmes de logique floue du niveau métier sont illustrés dans la figure 4.

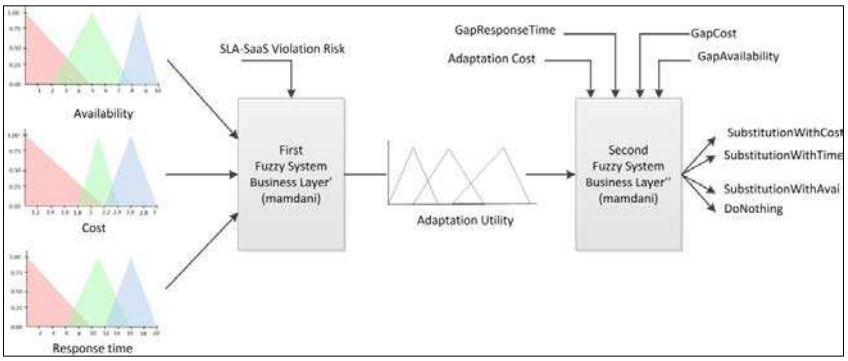

Figure 4. Les deux systèmes de logique floue du niveau métier

Le premier système flou est invoqué par l'*Adaptation Manager* à chaque point de conformité. Son objectif est d'évaluer les valeurs des paramètres de QoS du service composite afin de décider si des adaptations doivent être prises en compte au niveau métier (voir figure 5).

1. Dont les paramètres de réglage sont définis après avoir réalisé un ensemble de tests.

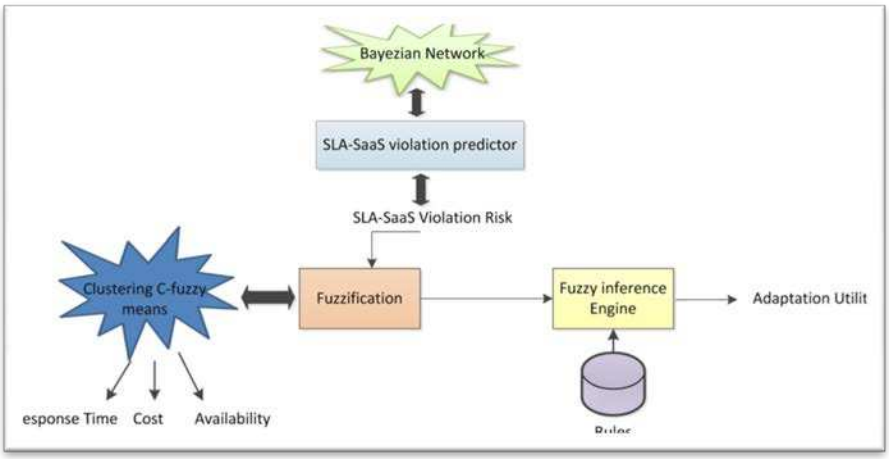

Figure 5. Premier système de logique floue du niveau métier (FS1)

Deux variables d'entrée sont prévues pour ce système, à savoir : 1) les paramètres QoS du chemin exécuté du SaaS composite tels que le temps d'exécution, le coût, etc. et 2) le risque de violation du SLA-SaaS récupéré à partir du SLA-SaaS Violation Predictor. Quant à la seule variable de sortie, elle illustre l'utilité de l'adaptation (*Adaptation Utility*). Ces variables sont exprimées selon trois termes linguistiques : faible, moyen et élevé.

Afin de regrouper les paramètres de QoS en des ensembles flous, nous utilisons l'algorithme Fuzzy C-Means (FCM) (Bezdek *et al.*, 1984). Nous estimons que ce choix est plus judicieux et moins couteux que celui de se baser sur les exécutions antérieures des services ou sur l'avis des experts (Rajni et Dahiya, 2012). En effet, dans le contexte de service Web et Cloud, les experts ne sont pas en mesure de juger facilement l'appartenance d'un paramètre de QoS à un ensemble flou.

L'algorithme Fuzzy C-Means (FCM) que nous utilisons est un algorithme de classification non supervisée flou. Issu de l'algorithme des C-moyennes (*C-means*), il introduit la notion d'ensemble flou dans la définition des classes : chaque point dans l'ensemble des données appartient à chaque cluster avec un certain degré, et tous les clusters sont caractérisés par leur centre de gravité.

Comme les autres algorithmes de classification non supervisée, il utilise un critère de minimisation des distances intra-classe et de maximisation des distances inter-classes, mais en donnant un certain degré d'appartenance à chaque classe. Cet algorithme nécessite la connaissance préalable du nombre de clusters et génère les classes par un processus itératif en minimisant une fonction objective. Ainsi, il permet d'obtenir une partition floue de chaque qualité de service grâce à un degré d'appartenance (compris entre 0 et 1) à une classe donnée. Le cluster auquel est

associé un paramètre de qualité de service est celui dont le degré d'appartenance sera le plus élevé.

À cette fin, nous définissons trois classes de paramètres de QoS à savoir : faible, moyen et élevé. Pour chaque contexte fonctionnel de service élémentaire et à chaque paramètre de qualité de service, nous appliquons le Fuzzy C-means pour organiser les services autour de trois clusters de qualité. Après l'application de l'algorithme, les centroïdes des classes $C = \{c_1, \ldots, c_c\}$ et une matrice de partition $W = w_{ij} \in [0,1]$ sont définis. Chaque élément w_{ij} indique le degré auquel le paramètre de qualité de service du service élémentaire j remplit le terme linguistique i. Les degrés d'appartenance du service aux clusters correspondants sont obtenus en minimisant itérativement la fonction objective suivante :

$$J_m(W,C,S) = \sum_{j=1}^{n}\sum_{i=1}^{c}(W_{ij})^m \left\|S_j - C_i\right\|2$$

avec

$$W_{ij} = \frac{1}{\sum_{k=1}^{c}\left(\frac{\left\|S_j - C_i\right\|}{\left\|S_j - C_k\right\|}\right)^{\frac{2}{m-1}}} \quad (1)$$

L'avantage d'utiliser l'algorithme Fuzzy C-means est que chaque valeur d'un paramètre de QoS d'un service élémentaire est classée dans au moins une classe avec un degré d'appartenance. Une fois classés, ces paramètres de qualité peuvent être évalués. Par exemple, un service élémentaire appelé « Plan assembling » pourrait être classé dans la classe « élevé » pour le coût avec un degré d'appartenance de 0,2 ; il peut être également classé dans la classe « moyen » avec un degré d'appartenance de 0,8.

Afin d'aider à la prise de décision, le système de logique floue se base sur un ensemble de règles de type if-then (voir figure 6). Nous avons défini 32 règles composées, identifiées en combinant les différents variables d'entrée du système flou et les relations possibles avec la variable de sortie à savoir l'utilité d'adaptation. Ces règles représentent le scénario qui peut avoir lieu pour la première prise de décisions à savoir si une adaptation au niveau métier est nécessaire ou une stratégie de rien faire sera maintenue.

Quant au second système de logique floue (FC2), il est invoqué quand l'utilité de l'adaptation est élevée ou moyenne et permet ainsi de retrouver la meilleure stratégie d'adaptation. Les variables d'entrée de ce système sont : l'utilité de l'adaptation, le coût de l'adaptation et l'écart entre les paramètres de QoS agréés et ceux mesurés[2].

2. Calculé en utilisant la formule suivante : $GapX(S_i) = \frac{agreed\ X(S_i) - mesured X(S_i)}{mesured\ X(S_i)}$
Avec X = coût, temps d'exécution ou disponibilité et S_i est un service élémentaire. Ensuite une fonction gaussienne est utilisée pour définir le degré d'appartenance de l'écart aux ensembles flou : élevé, moyen et faible.

La variable de sortie de ce système est la stratégie d'adaptation qui peut être : ne rien faire ou substitution avec Temps/Coût/Disponibilité. Les différents variables de ce système sont exprimées en utilisant les même termes linguistiques que le FC1 à savoir: élevé, moyen et faible.

Rule 1

IF (Adaptation Utility IS high AND Adaptation Cost IS low AND GapTime IS high AND GapCost IS low and GapAvailability IS low)
OR (Adaptation Utility IS high AND Adaptation Cost IS meduim AND GapTime IS high AND GapCost is low and GapAvailability IS low)
OR (Adaptation Utility IS high AND Adaptation Cost IS medium AND GapTime IS high AND GapCost is meduim and GapAvailability IS low)
OR (Adaptation Utility IS high AND Adaptation Cost IS low AND GapTime IS high AND GapCost is low and GapAvailability IS medium)
THEN AdaptationStrategy IS SubstitutionWithTime

Figure 6. Extrait des règles du premier système de logique floue du niveau métier

5.2. Les deux systèmes de logique floue en action

Les deux systèmes de logique floue déjà décrits sont invoqués par le gestionnaire d'adaptation (*Adaptation Manager)* conformément à l'algorithme présenté dans la figure 7. Les notations utilisées par l'algorithme sont les suivantes :

Au moment de l'exécution, tout SaaS composite, comme mentionné auparavant possède un contexte nommé SaaS-Ctxt.

Toute exécution d'un SaaS composite trace un chemin d'exécution particulier $P = \{S_1, S_2, S_3.....S_k\}$ qui décrit les services invoqués et les branchements utilisés par ces services.

Chaque service élémentaire appartenant au chemin d'exécution possède un contexte nommé e-Ctxt.

GapTime, GapCost et GapAvailability désignent respectivement l'écart observé entre le temps d'exécution, coût ou disponibilité agréé(e) et celui mesuré(e) des services élémentaires présent dans le chemin P.

La logique de fonctionnement des deux systèmes de logique floue débute par l'invocation du module SLA-SaaS Violation Predictor (ligne 6). Si le risque de violation est faible, l'algorithme est arrêté. Sinon, la liste des services participants au chemin d'exécution P est organisée en utilisant un tri descendant selon le GapTime (ligne 9) après avoir récupéré à partir du gestionnaire de contexte (Context Manager ligne) les temps d'exécution agréés et mesurés. Le service élémentaire qui a le plus d'écart entre le temps d'exécution agréé et celui mesuré est sélectionné par la suite (ligne 11). Le temps d'exécution de ce service est extrait. La même procédure est appliquée pour les autres paramètres de qualité de service (le coût et la disponibilité)

(ligne 12 à 29) pour invoquer le premier système de logique floue (FS1) qui va décider à son tour si une stratégie d'adaptation est nécessaire (ligne 31). Si oui (c'est-à-dire l'utilité de l'adaptation est élevée ou moyenne), le second système de logique floue (FS2) est invoqué pour obtenir la stratégie d'adaptation adéquate (ne rien faire ou substitution avec Temps/Coût/Disponibilité). Cette stratégie est envoyée par la suite au gestionnaire d'adaptation qui va sélectionner le service substituant en tenant en considération les recommandations du FS2 ou ne rien faire le cas échéant (ligne 43).

```
1  Input SaaS_id, CP_id;
2  Output Adapted_SaaS;

3  Path Vector <Service>=null;
4  Context SaaS_ctxt=ContextManager.getContext(SaaS_id)
5  SaaS_ctxt.getmesuredQoS();
6  SLAViolRisk=SLASaaSViolationPredictor(Path,mesuredQoS[]);
7  Path=ExtractPath(SaaS_id, CP_id);
8  If (SLAViolRisk is high or meduim)
9  {        Sort Path by GapTime Desendent
10   {
11   S_id=Path[0];
12   GapTime=getGapTime(S);
13   Context e_ctxt=ContextManager.getContext(S);
14   RT=ctxt.getMesuredRT();
15   }
16        Sort Path by GapCost desendent;
17   {
18   S_id=Path[0];
19   GapCost=getGapCost(S);
20   Context.e_ctxt=ContextManager.getContext(S);
21   Cost=e_ctxt.getmesuredCost();
22  }
23        Sort Path by GapAvail desendent
24  {
25  S_id=Path[0];
26  GapAvail=getGapAvail(S);
27  Context.e_ctxt=ContextManager.getContext(S);
28  Avail=ctxt.getmesuredAvail();
29  }
30  }
31   AdaptationUtility=FS1(SLAViolRisk, RT, Cost, Availability)
32  If AdaptationUtility is high or meduim
33 {
34  Service S_id=getSucesseurCP(CP_id);
35  Contexte e_ctxt=ContextManager.getContext(S_id);
36  SubstitutionCost=e_ctxt=getAdaptationCost();
37  Service_funct=e_ctxt.getFunctionalContext();
38  SubstitutionStrategy=FS2(GapTime, GapCost, GapAvail, AdaptationCost);
39  If (SubstitutionStrategy !=DoNothing)
40  {
41 CandidateService[]=DiscoveryManager(SubstitutionStrategy, Service_funct);
42 }
43  AdaptationManager(CandidateService[0], SaaS_id);
44 }}
```

Figure 7. Logique de fonctionnement des deux systèmes de logique floue

5.3. Système de logique floue pour la prise de décision d'adaptation niveau IT

Les SaaS composites implémentés sous la forme d'un ensemble de services Web sont très diversifiés allant de simple processus de prise de commandes clients aux processus plus sophistiqué comme les processus de la neuroscience (Zhao *et al.*, 2004), ayant une structure complexe ou les processus de d'annotation de protéines (O'Brien *et al.,* 2004). Malgré leur diversité, ces services s'accordent sur le fait que leurs qualités de service sont largement influencées par les capacités des machines virtuelles sur lesquelles ils s'exécutent. Par capacité nous sous-entendons le taux d'utilisation de la VM, la quantité de mémoire disponible et l'utilisation de CPU. Ainsi, il est fondamental de surveiller la VM, de voir la relation entre la consommation de ses ressources en fonction de la charge du service qu'elle exécute et de dimensionner par conséquent la quantité de ressources si besoin d'une manière automatique. En agissant sur la quantité de ressources de la VM, on contrôlera sa capacité de traitement. Cela a un impact direct sur les performances des traitements effectués, et donc sur la qualité du service offert.

Ce travail de recherche s'intéresse principalement à l'adaptation dynamique des changements du contexte d'exécution de la VM en adoptant la stratégie d'adaptation de type dimensionnement vertical. Rappelons que le dimensionnement permet l'ajout (*scale up*) ou la suppression (*scale down*) des ressources telles que CPU ou mémoire à la VM. Ce dimensionnement est limité par la quantité de ressource libre disponible sur le serveur physique hébergeant la machine virtuelle et est guidé dans notre travail par un système de logique floue. Ce dernier collabore avec deux autres modules à savoir : le moniteur des ressources et le module SLA-IaaS violation predictor. Le moniteur de ressources, surveille la machine virtuelle et reporte les informations sur l'utilisation de ses ressources. Il s'agit en effet de capturer le contexte dynamique de la VM à savoir : l'utilisation de la mémoire, de l'espace de stockage, etc. et ceci à chaque intervalle de temps. Le choix de l'intervalle est capital pour assurer l'efficacité et l'efficience de toute approche de monitoring. Cet intervalle doit garantir un équilibre entre la fréquence de collecte des informations pertinentes et la prestation du fournisseur de Cloud. Prévoir un court intervalle apporte de l'information contextuelle rapide sur l'état de la ressource, mais peut éventuellement dégrader la performance du service. A l'inverse, un long intervalle, risque de retarder la détection de la violation de l'SLA et par conséquent impose au fournisseur de Cloud de payer des pénalités coûteuses. Plusieurs travaux de recherche se sont attardés sur la définition d'un intervalle optimal pour la détection précoce de la violation de SLA dans le contexte du Cloud (Emeakaroha *et al.* 2012 ; Kouki *et al.*, 2014). Dans ce travail, les moniteurs de ressources, intégrés dans les machines virtuelles du fournisseur, utilisent l'intervalle de monitoring proposé dans (Emeakaroha *et al.,* 2012). Les informations ainsi collectées par les moniteurs sont envoyées au module SLA-IaaS violation predictor afin de construire le réseau bayésien; qui prédit à son tour le risque de violation du SLA-IaaS.

Ensuite, le système de logique floue (voir figure 8) est déclenché pour guider la décision de la stratégie d'adaptation IT à appliquer. Les variables d'entrée de ce système sont le contexte dynamique de la VM (utilisation de ses ressources) et le risque de violation du SLA-IaaS. Quant à la variable de sortie, elle dénote la stratégie d'adaptation IT (Ne rien faire, Scaling up_Memory, Scaling down_CPU, etc.). Ces différentes variables sont définies en utilisant les termes linguistiques à savoir : élevé, moyen et faible, et des fonctions trapézoïdales ont été définies pour mesurer les degrés d'appartenance des valeurs de ces variables aux ensembles flous. Une fois une stratégie d'adaptation décidée, le Scaling Manager dimensionne dynamiquement la ressource relative à la VM.

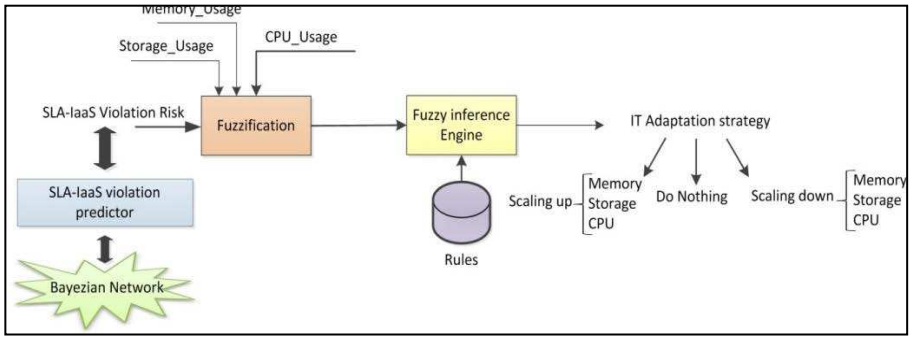

Figure 8. Système de logique floue du niveau IT

6. Evaluation expérimentale et validation

Dans cette section, nous présentons les expériences réalisées pour valider l'approche d'adaptation proposée ainsi que les résultats obtenus. D'abord, nous détaillons le cadre expérimental. Nous présentons ensuite, les expérimentations essentielles qui concernent l'adaptation du niveau métier ainsi que celles du niveau IT. Notre conclusion présente nos commentaires sur le travail conduit dans ces expérimentations.

Afin de tester l'approche SAV-CSaaS nous avons utilisé l'environnement de Cloud appelé OpenStack. Openstack est un projet open source qui offre un ensemble de logiciels permettant de déployer des infrastructures de Cloud (*Infrastructure as a Service*). La plateforme indispensable à l'exécution des services composites a été déployée sur toutes les machines virtuelles initiées pour l'expérimentation (le moteur d'exécution, les services élémentaires, le serveur d'application, le serveur de base de données, etc.).

L'ensemble des expérimentations ont été réalisé en utilisant l'étude de cas d'un processus de longue durée d'exécution adapté de (Leitner, 2011). Cette étude de cas considère un revendeur de robots sophistiqués (ACME BOT) (figure 9). Les clients s'adressent à ce revendeur pour demander un devis concernant un robot ayant une

configuration particulière. Pour se faire, le service « Plan Assembling » est invoqué afin de prévoir les pièces indispensables pour la configuration demandée. Ensuite, le service « Check Availability of Parts » intervient pour vérifier si les pièces sont en stock ou pas. Si les pièces existent ou peuvent être approvisionnées, le service « Generate Offer » se charge de générer une offre qui sera adressée par la suite au client. Ce dernier peut alors annuler ou passer la commande. Si la commande est passée, l'approvisionnement des pièces manquantes est lancée à travers le service « Order Unavailable Parts ». Une fois les pièces disponibles, le robot est prévu pour l'assemblage grâce au service « Product assembled ». Un service de contrôle de qualité permettra par la suite de vérifier la qualité du robot. En cas de conformité avec les standards de qualité, il est livré au client (service « Ship »). En parallèle à la livraison, le compte du client est crédité du montant de la commande et une facture lui sera envoyée.

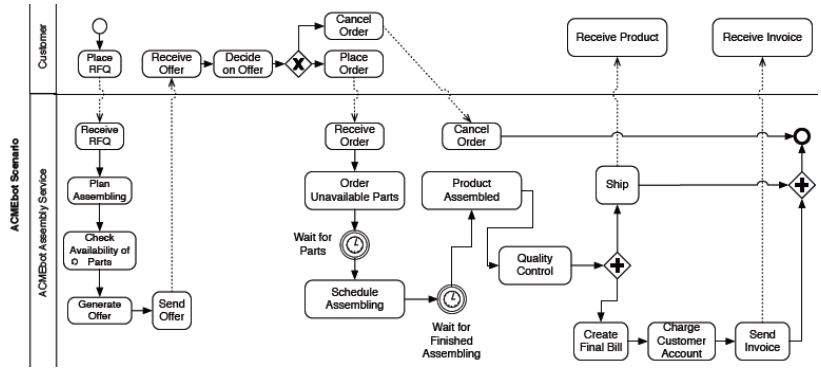

Figure 9. Scénario de motivation (Leitner 2011)

6.1. Evaluation de l'adaptation métier

Dans cette section, nous évaluons l'efficacité de l'approche d'adaptation proposée au niveau métier. Tout d'abord, nous évaluons la décision d'adaptation suggérée par le système de logique floue (FS1) en la comparant avec une adaptation classique (substitution à chaque variation du contexte). Ensuite, nous évaluons la stratégie d'adaptation proposée par le système de logique floue FS2 en la comparant avec une approche de substitution basée sur une pondération fixe.

6.1.1. Evaluation du nombre de substitution des services

Nous avons exécuté 10 fois le composite SaaS (CSaaS) en variant à chaque exécution les paramètres de contexte et en adoptant deux approches d'adaptation différentes : notre approche basée sur les systèmes de logique floue et l'approche d'adaptation classique. Une approche classique effectue une substitution de services élémentaires quand une déviation de QoS se produit. Une comparaison entre ces deux approches est illustrée dans figure 10.

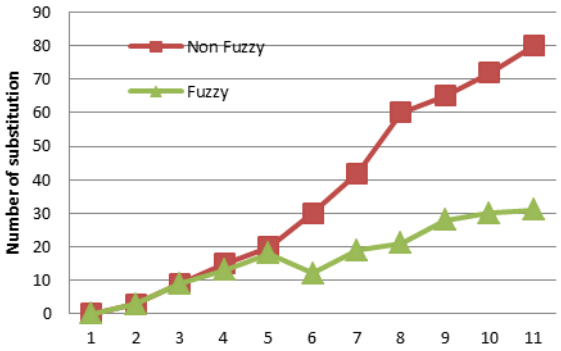

Figure 10. Evaluation du nombre de substitution selon les deux approches d'adaptation

L'analyse des résultats obtenus de la figure 10 nous permet de constater que notre approche d'adaptation réduit le nombre de substitutions de service par rapport à une adaptation classique. En effet, le nombre de substitutions augmente dans le cas d'une adaptation classique de façon exponentielle avec le nombre d'exécutions simultanées du CSaaS. Ceci peut être expliqué par le fait que notre approche vise à garantir un compromis entre la qualité globale du service composite, la pertinence de l'adaptation et le coût de substitution. En effet, dans certains cas de variations des paramètres de contexte (temps d'exécution élevé d'un service élémentaire) ; celle-ci peut être ignorée en raison du coût élevé de la substitution.

6.1.2. Evaluation du coût total de l'adaptation

La deuxième expérimentation permet de comparer le coût de notre approche d'adaptation et celui de l'approche d'adaptation classique. Comme mentionné déjà, l'approche d'adaptation classique effectue à chaque variation du contexte une substitution, dans laquelle un nouveau service est sélectionné et un nouveau contrat est établi avec un coût donné. Dans ce cas, le coût total de l'adaptation consiste au coût de substitution ainsi qu'au coût constant de la mise en place du nouveau contrat. En revanche le coût de notre approche d'adaptation dépend de la décision produite par les deux systèmes de logique floue. Si une substitution est maintenue comme décision d'adaptation, alors le coût d'adaptation total est le même que celui d'une adaptation classique. Sinon (il s'agit de ne rien faire stratégie), le coût total de l'adaptation est égale uniquement au coût relatif à cette stratégie. Il s'agit en effet du coût relatif à la prise de décision.

Comme l'illustre la figure 11 on peut constater que notre approche d'adaptation peut entraîner le même coût total d'adaptation que celui de l'approche d'adaptation classique lorsque la prise de décision est la stratégie de substitution (substitution avec temps de réponse/coût). Cependant, ce coût augmente pour l'approche d'adaptation classique par rapport à notre approche d'adaptation. Ceci est dû au fait

que dans certains cas notre approche choisit la stratégie de ne rien faire plutôt que de substituer les services défaillants.

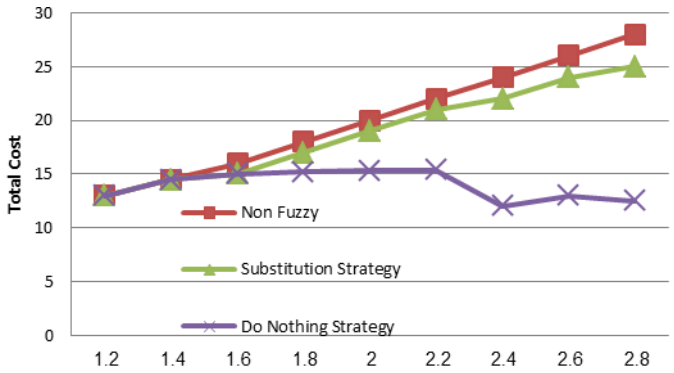

Figure 11. Evaluation du coût total d'adaptation

6.1.3. Adaptation de service basée sur des poids fixes

Dans cette série d'expérimentations nous évaluons notre approche d'adaptation en la comparant avec une approche d'adaptation basée sur une substitution à pondération fixe des paramètres de QoS ($W_1 = W_2 = W_3 = 0{,}33$ avec W_1 est le poids associé au temps de réponse, W2 est le poids pour le coût et W3 est le poids de la disponibilité).

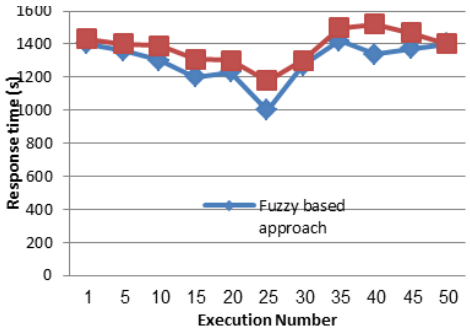

Figure 12. Comparaison selon le temps d'exécution

Le SaaS composite relatif à notre étude de cas a été exécuté 50 fois et ceci en adoptant les deux approches d'adaptation. Les figures 12 et 13 illustrent une comparaison entre l'approche proposée et l'approche de substitution à pondération fixe pour chacun des paramètres de qualité de service (temps de réponse et coût).

L'analyse de ces figures prouve que l'approche d'adaptation proposée améliore les valeurs de qualité de service globales de la composition. En effet, notre approche d'adaptation permet un temps de réponse plus réduit que celui de l'approche de substitution à pondération fixe (voir figure 12). Il s'agit en effet d'une réduction moyenne de 2,5 % et un écart-type moyen de 4,7 %. Quant à la comparaison selon le coût, elle montre que l'utilisation du système de logique floue pour l'adaptation permet une réduction moyenne de 3,7 % avec un écart type moyen de 5,2 %, par rapport à une approche de substitution à pondération fixe (voir figure 13).

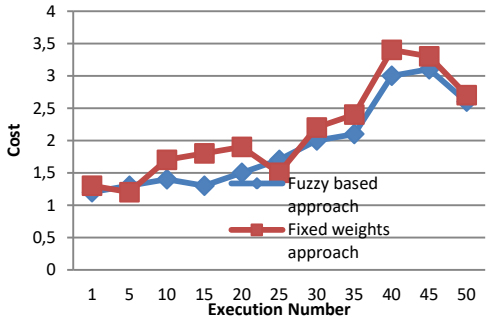

Figure 13. Comparaison selon le coût

6.2. Evaluation de l'adaptation IT

Les expérimentations réalisées au niveau de l'adaptation IT visent à évaluer d'une part, la performance du module SLA-IaaS Violation Predictor, module responsable du déclanchement de l'adaptation, et d'autre part, la pertinence des stratégies d'adaptation suggérées par le système de logique floue. Nous décrivons les configurations logicielles indispensables pour l'adaptation IT puis détaillons en les différentes expérimentations réalisées.

6.2.1. Configurations logicielles et implémentation

Les différentes machines virtuelles concernées par cette expérimentation sont contrôlées via l'interface de gestion de VMware ESX Server. Ce dernier fournit une couche de virtualisation et divise ainsi un serveur physique en plusieurs machines virtuelles sécurisées et portables qui peuvent fonctionner côte à côte sur la même machine physique (VMware 2015). Chaque machine virtuelle représente un système complet avec un processeur, une mémoire, un réseau et un stockage et elle est surveillée grâce aux moniteurs de ressources. La mise en œuvre des moniteurs de ressources se base sur le projet Monitorix (Sanfeliu 2005). Quant aux système de logique floue et le *scaling manager*, ils sont implémentés en utilisant le langage JAVA. Le *scaling manager* travaille de concert avec le VMware ESX. Ce dernier l'utilise afin de gérer les ressources partagées entre les différentes machines virtuelles hébergées sur la même machine physique.

6.2.2. Expérimentations pour tester l'efficacité de la stratégie de dimensionnement du CPU

L'objectif principal de cette expérimentation consiste à montrer l'efficacité de l'approche d'adaptation IT en examinant la capacité du système de logique floue à proposer la meilleure stratégie d'adaptation en fonction du contexte. Etant donné que nous avons examiné des processus coûteux en processeur, nous nous concentrons dans l'expérimentation sur le dimensionnement du CPU comme stratégie d'adaptation. Pour ce faire, nous considérons le service élémentaire « Plan Assembling » appartenant à l'étude de cas (ACME BOT).

Ce service est déployé sur une machine virtuelle dédiée dont les ressources sont continuellement surveillées par les moniteurs dédiés. Afin de conduire les expérimentations nous soumettons le service à des variations de charge (nous varions le nombre d'invocations du service). Les informations sur l'utilisation des ressources de la machine virtuelle ainsi que sur le risque de violation du SLA sont envoyées au système de logique floue afin de guider la décision d'adaptation.

Pour prouver l'efficacité du système de logique floue et plus précisément la stratégie de dimensionnement du CPU, nous la comparons à deux autres stratégies utilisées dans la littérature à savoir : l'allocation statique (une grande quantité de CPU est attribuée à chaque fois à la machine virtuelle) et l'allocation de CPU en période de surcharge (appelée en anglais *peak load*). L'analyse des résultats de la figure 14 montre que l'allocation de CPU fournie par le système flou en collaborant avec le *scaling manager* est conforme à la demande en CPU de la machine virtuelle (l'erreur moyenne est inférieure à 1,9 %) tout comme la stratégie l'allocation de CPU en période de surcharge. Quant à l'allocation statique, étant donnée qu'elle ne considère pas la variation de la charge de travail du service, elle peut conduire à une importante sous ou sur-utilisation de la machine virtuelle et cause ainsi une erreur moyenne d'environ 34,9 %. Cette différence dans la précision d'allocation du CPU peut affecter la performance du service.

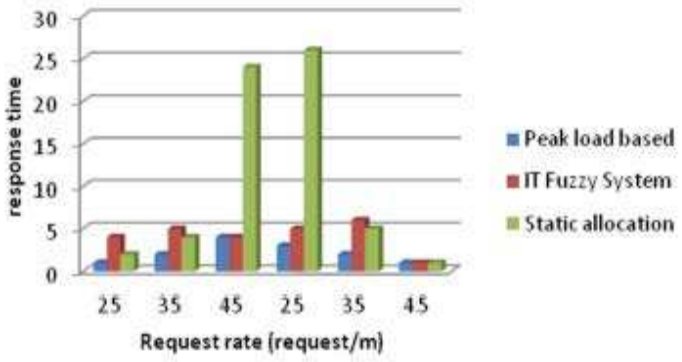

Figure 14. Comparaison des stratégies d'allocation de CPU

7. Conclusion

Un SaaS composite souvent fonctionne dans un environnement dynamique et volatil tel que le Cloud Computing et a donc des paramètres QoS très variables par rapport à ceux qui fonctionnent dans un environnement de service classique. Par conséquent, le suivi et l'adaptation de SaaS composite s'avèrent d'une grande importance pour garantir les paramètres de QoS définis dans le contrat SLA. Le travail décrit dans cet article propose :

1) Une catégorisation contextuelle qui facilite les mécanismes d'auto-adaptation du service. Cette catégorisation comprend : le contexte du service élémentaire, le contexte lié à la VM et le contexte de SaaS composite.

2) Un système qui aide les SaaS composites à prendre conscience de son contexte et, en réponse aux changements pertinents dans son contexte, évalue l'utilité de l'adaptation et s'adapte si nécessaire pour éviter la violation de SLA. Le système proposé considère deux couches : métier et IT.

3) Trois systèmes flous pour les SaaS auto-adaptatifs : deux systèmes flous dans la couche métier et un système flou pour la couche IT. Le premier évalue les valeurs QoS du chemin exécuté dans le service composite afin de décider si des actions d'adaptation appropriées doivent être prises dans la couche métier. Le deuxième est invoqué lorsque l'utilité d'adaptation est élevée ou moyenne et vise à sélectionner la meilleure stratégie d'adaptation. Le troisième système Fuzzy guide le processus de prise de décision de l'affectation des ressources aux machines virtuelles en fonction de leur demande de charge de travail dynamique ainsi que du SLA-IaaS convenu.

4) La mise en œuvre et l'évaluation de l'adaptation métier et l'adaptation de l'infrastructure à l'aide d'un environnement Cloud et d'un long processus.

Dans notre futur travail, nous prévoyons enquêter sur la qualité des prédictions. En effet, la qualité des prédictions est encore plus importante que le temps nécessaire pour générer des modèles de prédiction ou des prédictions d'exécution. En outre, nous étudierons la variation de la charge pour inclure d'autres paramètres tels que le stockage, le réseau, etc. En outre, nous devons travailler sur la quantité de CPU à l'échelle en utilisant l'exécution passée en tenant compte du coût d'échelle pour aider le fournisseur à économiser des coûts tout en essayant de satisfaire le SLA du client.

Bibliographie

Alexander K. et Heiko L. (2003). The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, vol. 11, n° 1, p. 57-81.

Aschoff R. R. et Zisman A. (2012). Proactive adaptation of service composition. *2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS).*

Avila S. D. G. (2014). Proactive Adaptation in Service Composition using a Fuzzy Logic Based Optimization Mechanism. *4th International Conference on Cloud Computing and Service Science (CLOSER'14)*, Barcelona, Spain.

Bashar A. (2013). Autonomic scaling of Cloud Computing resources using BN-based prediction models. *IEEE 2nd International Conference on Cloud Networking (CloudNet)*.

Bezdek J. C., Ehrlich R. et Full W. (1984). FCM: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, vol. 10, n° 2, p. 191-203.

Dey A. K., Abowd G. D.et Salber D. (2001). A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, vol.16, n° 2, p. 97-166.

Emeakaroha V. C., Netto M. A. S., Calheiros R. N., Brandic I., Buyya R. et De Rose C. A. F. (2012). Towards autonomic detection of SLA violations in Cloud infrastructures. *Special section: Quality of Service in Grid and Cloud Computing*, vol. 28, n° 7, p. 1017-1029.

Grati. R., Boukadi. K. et Ben Abdallah. H. (2012). An Event based approach to Extract the Run Time Execution Path of BPEL Process for Monitoring QoS in the Cloud. *World Academy of Science, Engineering and Technology (WASET)*, vol. 6, n° 10.

Huber N., Brosig F. et Kounev S. (2011). Model-based self-adaptive resource allocation in virtualized environments. *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Waikiki, Honolulu, HI, USA, ACM, p. 90-99.

Kouki Y., Jr F. A. D. O., Dupont S. et Ledoux T. (2014). A Language Support for Cloud Elasticity Management. *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid*, Chicago, United States.

Lauritzen S. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics & Data Analysis* 19.

Leitner P. (2011). *On Preventing Violations of Service Level Agreements in Composed Services Using Self-Adaptation*, Fakultät für Informatik der Technischen Universität Wien.

Maurer M., Brandic I. et Sakellariou R. (2013). Adaptive resource configuration for Cloud infrastructure management. *Future Generation Computer Systems*, vol. 29, n° 2, p. 472-487.

O'Brie A., Newhouse S. et Darlington J. (2004). Mapping of scientific workflow within the e-protein project to distributed resources. *UK e-Science All Hands Meeting, Nottingham*, UK: p. 404-409.

Pernici B. et Siadat S. H. (2011). Selection of Service Adaptation Strategies Based on Fuzzy Logic. *2011 IEEE World Congress on Services (SERVICES)*.

Rajni M. et Dahiya D. (2012). An Optimized Business Service Directory for the ESB Platform in SOA. *International Journal of Computer Networks & Communications (IJCNC)* vol. 4, p. 165-187.

Raman K., Marco P. et A. Z. (2009). Cross-layer Adaptation and Monitoring of Service-Based Applications. *2nd Workshop on Monitoring, Adaptation and Beyond (MONA+)*, Italy.

Sanfeliu J. (2005). Monitorix tool. from http://www.monitorix.org/.

Spirtes P., Glymour C. et Scheines R. (2001). *Causation, Prediction and Search*, 2nd ed. MIT Press.

Steck H. (2001). Constrained-based structural learning in Bayesian networks using finite data sets. Ph.D. dissertation, Department of Informatics,Technical University Munich.

VMware (2015). from http://www.vmware.com/products/esxi-and-esx/overview.

Winograd T. (2001). Architectures for Context. *Human Computer Interaction Journal*, vol. 6, n° 2-3, p. 401-419.

Yazdanov L. et Fetzer C. (2012). Vertical Scaling for Prioritized VMs Provisioning. *2012 Second International Conference on Cloud and Green Computing.*

Zhao Y., Wilde M., Foster I., Voeckler J., Jordan T., Quigg E. et Dobson J. (2004). Grid middleware services for virtual data discovery, composition, and integration. *2nd workshop on Middleware for grid computing*, Toronto, Ontario, Canada, p. 57-62.