Contrôle des applications fondé sur la qualité de service pour les plate-formes logicielles dématérialisées (Cloud)

Ge Li

UNIVERSITÉ GRENOBLE ALPES

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE-ALPES

Spécialité : **STIC Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Ge LI

Thèse dirigée par **Patrice MOREAUX**
codirigée par **Frédéric POURRAZ**

préparée au sein du **Laboratoire d'Informatique, Systèmes, Traitement de l'Information et de la Connaissance (LISTIC)**
dans **l'École Doctorale Sciences et Ingénierie des Systèmes de l'Environnement et des Organisations (SISEO)**

Contrôle des applications fondé sur la qualité de service pour les plate-formes logicielles dématérialisées (Cloud)

Thèse soutenue publiquement le **21/07/2015**,
devant le jury composé de :

Mme. Frédérique Laforest
Professeur, Université Jean Monnet, Rapporteur
M. Jean-Marc Menaud
Professeur, École des Mines de Nantes, Rapporteur
Mme. Tatiana Aubonnet
Maître de conférences, CNAM Paris, Examinateur
M. Bruno Dillenseger
Docteur, Orange FranceTélécom R&D, Examinateur
M. Patrice Moreaux
Professeur, Université Savoie Mont Blanc, Directeur de thèse
M. Frédéric Pourraz
Maître de conférences, Université Savoie Mont Blanc, Co-Directeur de thèse

Acknowledgement

I really enjoy my three years scientific research experience in such a beautiful environment. I gain not only knowledge, but also friendship here.

I would like to thank Prof. Frédérique Laforest and Prof. Jean-Marc Menaud for helping me to improve my thesis.

I would like to thank Dr. Bruno Dillenseger for being part of my thesis committee and helping me patiently during the experiment.

I would like to thank Assoc.Prof. Tatiana Aubonnet for being part of my thesis committee and contributing towards the completion of this degree.

I am very grateful to my advisor Prof. Patrice Moreaux and co-advisor Assoc.Prof. Frédéric Pourraz for their patience and helpful advice throughout my research. I would like to thank Prof. Patrice Moreaux translated the "Résumé étendu" of this thesis for me.

I would thank the laboratory of LISTIC. I would like to thank my colleagues, especially Cedric Deffo-Sikounmo and Jerome Dupenloup, for their effort during the experiments.

I want to thank my dearest parents. I couldn't complete this thesis without their support.

I would like to thank my friends, especially Yajing Yan, Weiqun Liu, Li Tao, Linjuan Yan, Peng He and Zaoli Huang , and they are with me everywhere in my beautiful memory of Annecy.

Finally, I would like to thank my beloved husband Lichun Fan. Thank you for asking my phone number in English course five years ago.

Résumé : Le « Cloud computing » est un nouveau modèle de systèmes de calcul. L'infrastructure, les applications et les données sont déplacées de machines localisées sur des systèmes dématérialisés accédés sous forme de service via Internet. Le modèle « coût à l'utilisation » permet des économies de coût en modifiant la configuration à l'exécution (élasticité). L'objectif de cette thèse est de contribuer à la gestion de la Qualité de Service (QdS) des applications s'exécutant dans le Cloud. Les services Cloud prétendent fournir une flexibilité importante dans l'attribution des ressources de calcul tenant compte des variations perçues, telles qu'une fluctuation de la charge. Les capacités de variation doivent être précisément exprimées dans un contrat (le Service Level Agreement, SLA) lorsque l'application est hébergée par un fournisseur de Plateform as a Service (PaaS). Dans cette thèse, nous proposons et nous décrivons formellement le langage de description de SLA PSLA. PSLA est fondé sur WS-Agreement qui est lui-même un langage extensible de description de SLA. Des négociations préalables à la signature du SLA sont indispensables, pendant lesquelles le fournisseur de PaaS doit évaluer la faisabilité des termes du contrat. Cette évaluation, par exemple le temps de réponse, le débit maximal de requêtes servies, etc, est fondée sur une analyse du comportement de l'application déployée dans l'infrastructure cloud. Une analyse du comportement de l'application est donc nécessaire et habituellement assurée par des tests (benchmarks). Ces tests sont relativement coûteux et une étude précise de faisabilité demande en général de nombreux tests. Dans cette thèse, nous proposons une méthode d'étude de faisabilité concernant les critères de performance, à partir d'une proposition de SLA exprimée en PSLA. Cette méthode est un compromis entre la précision d'une étude exhaustive de faisabilité et les coûts de tests. Les résultats de cette étude constituent le modèle initial de la correspondance charge entrante-allocation de ressources utilisée à l'exécution. Le contrôle à l'exécution (runtime control) d'une application gère l'allocation de ressources en fonction des besoins, s'appuyant en particulier sur les capacités de passage à l'échelle (scalability) des infrastructures de cloud. Nous proposons RCSREPRO (Runtime Control method based on Schedule, REactive and PROactive methods), une méthode de contrôle à l'exécution fondée sur la planification et des contrôles réactifs et prédictifs. Les besoins d'adaptation à l'exécution sont essentiellement dus à une variation de la charge soumise à l'application, variations difficiles à estimer avant exécution et seulement grossièrement décrites dans le SLA. Il est donc nécessaire de reporter à l'exécution les décisions d'adaptation et d'y évaluer les possibles variations de charge. Comme les actions de modification des ressources attribuées peuvent prendre plusieurs minutes, RCSREPRO réalise un contrôle prédictif fondée sur l'analyse de charge et la correspondance indicateurs de performance-ressources attribuées, initialement définie via des tests. Cette correspondance est améliorée en permanence à l'exécution. En résumée, les contributions de cette thèse sont la proposition de langage PSLA pour décrire les SLA ; une proposition de méthode pour l'étude de faisabilité d'un SLA ; une proposition de méthode (RCSREPRO) de contrôle à l'exécution de l'application pour garantir le SLA. Les travaux de cette thèse s'inscrivent dans le contexte du projet FSN OpenCloudware (www.opencloudware.org) et ont été financés en partie par celui-ci.

Mots-clefs:PaaS, QoS, SLA, SLA étude de faisabilité, le provisioning des ressources, le contrôle Runtime.

Abstract: Cloud computing is a new computing model. Infrastructure, application and data are moved from local machines to internet and provided as services. Cloud users, such as application owners, can greatly save budgets from the elasticity feature, which refers to the "pay as you go" and on-demand characteristics, of cloud service. The goal of this thesis is to manage the Quality of Service (QoS) for applications running in cloud environments Cloud services provide application owners with great flexibility to assign "suitable" amount of resources according to the changing needs, for example caused by fluctuating request rate. "Suitable" or not needs to be clearly documented in Service Level Agreements (SLA) if this resource demanding task is hosted in a third party, such as a Platform as a Service (PaaS) provider. In this thesis, we propose and formally describe PSLA, which is a SLA description language for PaaS. PSLA is based on WS-Agreement, which is extendable and widely accepted as a SLA description language. Before signing the SLA contract, negotiations are unavoidable. During negotiations, the PaaS provider needs to evaluate if the SLA drafts are feasible or not. These evaluations are based on the analysis of the behavior of the application deployed in the cloud infrastructure, for instance throughput of served requests, response time, etc. Therefore, application dependent analysis, such as benchmark, is needed. Benchmarks are relatively costly and precise feasibility study usually imply large amount of benchmarks. In this thesis, we propose a benchmark based SLA feasibility study method to evaluate whether or not a SLA expressed in PSLA, including QoS targets, resource constraints, cost constraints and workload constraints can be achieved. This method makes tradeoff between the accuracy of a SLA feasibility study and benchmark costs. The intermediate of this benchmark based feasibility study process will be used as the workload-resource mapping model of our runtime control method. When application is running in a cloud infrastructure, the scalability feature of cloud infrastructures allows us to allocate and release resources according to changing needs. These resource provisioning activities are named runtime control. We propose the Runtime Control method based on Schedule, REactive and PROactive methods (RCSREPRO). Changing needs are mainly caused by the fluctuating workload for majority of the applications running in the cloud. The detailed workload information, for example the request arrival rates at scheduled points in time, is difficult to be known before running the application. Moreover, workload information listed in PSLA is too rough to give a fitted resource provisioning schedule before runtime. Therefore, runtime control decisions are needed to be performed in real time. Since resource provisioning actions usually require several minutes, RCSREPRO performs a proactive runtime control which means that it predicts future needs and assign resources in advance to have them ready when they are needed.Hence, prediction of the workload and workload-resource mapping are two problems involved in proactive runtime control. The workload-resource mapping model, which is initially derived from benchmarks in SLA feasibility study is continuously improved in a feedback way at runtime, increasing the accuracy of the control.

To sum up, we contribute with three aspects to the QoS management of application running in the cloud: creation of PSLA, a PaaS level SLA description language; proposal of a benchmark based SLA feasibility study method; proposal of a runtime control method, RCSREPRO, to ensure the SLA when the application is running. The work described in this thesis is motivated and funded by the FSN OpenCloudware project (www.opencloudware.org).

Keywords:PaaS, QoS, SLA, SLA feasibility study, Resource provisioning, Runtime control.

Contents

List of Figures

List of Tables

Chapitre 1

Résumé étendu

1.1 Introduction

1.1.1 Cloud computing

Le *cloud computing* est un nouveau modèle d'architecture informatique. L'infrastructure, les plateformes, l'application et les données sont déplacées de la machine locale à l'Internet et sont utilisées sous forme de services. Avec le cloud computing, les utilisateurs peuvent utiliser ces ressources simplement, de la même manière qu'on utilise les services d'eau et d'électricité, sans se soucier de leur maintenance. De plus, les utilisateurs de Cloud peuvent espérer des réductions de coût en payant uniquement ce qu'ils utilisent effectivement.

Services dans le Cloud

Les services de cloud peuvent être classés en *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) et *Software as a Service* (SaaS). Les IaaSs fournissent à leurs clients des machines virtuelles, de l'espace de stockage et des connectivités réseau. Un IaaS fournit également des services tels que l'équilibrage de charge, les pare-feux. Le client peut déployer et exécuter une variété de logiciels sur la machine virtuelle, y compris l'environnement d'exécution. Les PaaSs fournissent la plate-forme de développement de logiciels sous la fomre de service au client. Un PaaS comprend habituellement un système d'exploitation, un environnement d'exécution pour les serveurs d'application, les outils liés aux langages de programmation, les système de gestion de base de données et les outils de gestion via le WEB. Les clients peuvent déployer et gérer leurs propres applications sur une telle plate-forme. Ils ont un comtrôle complet sur leurs applications, mais ils ne peuvent pas contrôler ni gérer l'infrastructure sous-jacente. L'expression SaaS désigne la mise à disposition de logiciels via internet. Les clients utilisent les logiciels *via* le WEB au lieu de l'acheter et l'installer sur leurs systèmes informatiques. Un client de SaaS peut configurer l'application, mais il ne peut pas gérer l'infrastructure cloud ni la plate-forme sous-jacente sur laquelle s'exécute l'application.

Gestion du Cloud

Comparativement aux centres de données (*Data centers*), le plus grand avantage du cloud computing est certainement la simplification de la gestion des systèmes mis en jeu. La gestion est un point fondamental de l'exploitation des IaaS, PaaS et SaaS. Elle doit fournir des informations

sur les systèmes dans le Cloud et permettre de prendre des décisions pertinentes pour leur fonctionnement. De nombreuses études ont été menées à partir de plusieurs points de vue concernant la gestion des systèmes dans le Cloud. Dans cette thèse, nous nous concentrons sur la gestion des services. Par conséquent, les problèmes d'approvisionnement des ressources, ceux de la virtualisation des ressources, de la migration de VM, etc. ne sont pas abordés dans notre travail. Plus précisément, nous nous concentrons sur la gestion des contrats de niveaux de services (*Service Level Agreement*, SLA) au niveau PaaS ; d'une part en étudiant l'expression formelle des SLAs ; d'autre part en proposant une méthode pour l'étude de la faisabilité d'un SLA lors de sa négociation, puis en contrôlant l'application à l'exécution, en liaison avec son SLA.

1.1.2 Le projet OpenCloudware

Le travail décrit dans cette thèse a été réalisé dans le contexte du projet OpenCloudware [ocw]. OpenCloudware est un projet qui vise à développer des modules logiciels pour construire un PaaS multi-IaaS. En tant que tel, il fournit une plate-forme de développement (construction, déploiement) et d'exécution (maintenance, modification, qualité de service) pour les applications dans le Cloud. Les différents fonctionnalités pour réaliser ces opérations sont fournies en ligne grâce à un portail de services. OpenCloudware est un des projets retenus du programme « développement de l'économie numérique », opéré par le Fonds national pour la Société Numérique (FSN). Couvrant la période de janvier 2012 à septembre 2015, ce projet de recherche et développement regroupe 18 partenaires industriels en académiques, dont l'Université Savoie Mont Blanc et son laboratoire LISTIC dans lequel ce travail de thèse a été réalisé.

1.1.3 Contrats de niveau de service

Un SLA est une partie des accords conclus entre un consommateur de services et un fournisseur de services. Il est utilisé pour décrire de manière structuré les éléments de qualité de service (*Quality of Service*, QoS) contractualisés entre les deux parties. Il permet de définir précisément les responsabilités des parties contractantes y compris les actions à réaliser par chaque partie dans différentes conditions. Habituellement, les services offerts par le fournisseur dans le SLA sont quantitativement et de manière exhaustive définis par une série de paramètres de qualité de service avec des plages de valeurs. Non seulement le niveau de qualité de service doit être garanti, mais aussi des « récompenses » et des « sanctions » sont également spécifiées dans le SLA selon que le contrat est respecté ou non. La gestion des systèmes informatiques dans le Cloud, y compris gestion des SLA, est essentiellement réalisée par des outils automatiques. Il est donc nécessaire de disposer d'une représentation des SLAs utilisable par une machine, par exemple en *eXtended Markup Language* (XML). L'utilisation des SLA n'est pas restreinte à la qualité de service. Par exemple, les systèmes de facturation peuvent utiliser les paramètres de récompense et sanction pour moduler le prix à payer par le consommateur de services. Dans la pratique, actuellement, la plupart des fournisseurs de services dans le Cloud énoncent un SLA sous forme textuelle et informelle. Par exemple, le service de cloud public Google App Engine [goo15] et Amazon Elastic Compute Cloud (EC2) [ec213] publient leurs SLAs en ligne et sous une forme générique, non spécifique à un client donné. Aujourd'hui, avec le développement des applications structurées en composition de services (*Service Oriented Architecture*, SOA) et du Cloud, de plus en plus de services sont construits ou fondées sur d'autres services fonctionnant dans le Cloud. Ces services peuvent de plus, être offerts par différents fournis-

seurs. Cela impose une définition claire, complète et formelle des interactions et des relations entre ces services, et donc l'utilisation des mécanismes de gestion de SLAs. Dans notre travail, nous développons notre proposition de SLA de niveau PaaS fondée sur WS-Agreement.

1.1.4 Contrôle à l'exécution

Pendant la négociation du SLA, le fournisseur de PaaS doit évaluer si le SLA en cours de négociation est réalisable ou non. Ces évaluations sont basées sur une analyse dépendant de l'application, par exemple une campagne de test (*benchmark*). Les résultats de ces tests constituent la base de la correspondance entre la charge de travail (*workload*) soumise à l'application et sa configuration, correspondance utilisée par notre méthode de contrôle de l'exécution. Lorsque l'application est en cours d'exécution dans une infrastructure Cloud, la fonctionnalité de variabilité des infrastructures de Cloud nous permet d'allouer ou de libérer des ressources selon l'évolution des besoins. Ces activités d'attribution/retrait de ressources sont nommées contrôle de l'exécution. Comme l'évolution des besoins est principalement causée par la variation de la charge de travail pour la majorité des applications exécutées dans le Cloud, des décisions relatives au contrôle d'exécution sont nécessaires.

Il y a essentiellement deux niveaux de contrôle de l'exécution. D'une part, on alloue des ressources sous la forme de machines virtuelles à l'application et cette allocation est motivée par les objectifs de performance de l'application définis par le SLA de haut niveau [VTM09]. D'autre part,on place le placement des machines virtuelles sur les hôtes physiques est une conséquence des politiques de gestion des centres de données relatives aux coûts de gestion [VTM09]. Dans cette thèse, nous traitons du premier type de contrôle.

Dans les systèmes informatiques, les performance d'un système sont difficiles à évaluer à partir des paramètres de configuration. Les méthodes de tests (*Benchmark*) sont considérées comme un moyen efficace d'évaluation des indicateurs de performance spécifiques d'un système informatisé. Un système de test est composé du système à tester (*System Under Test*, SUT), d'une entité d'injection de la charge de travail, d'un mécanisme de collecte des performances et d'une analyse des résultats. Les Benchmarks doivent être quantifiés et comparables. Par exemple, le niveau d'utilisation du(des) processeur(s) (*Central Processing Unit(s)*, CPU) de calcul en flottant, la bande passante utilisée pour les accès réseau aux données peuvent aider les utilisateurs à juger si chaque type de CPU peut exécuter l'application dans des conditions satisfaisantes. Les méthodes de benchmarking sont largement utilisées pour évaluer les performance des CPU, mais elles peuvent également être utilisées pour les systèmes logiciels. Les benchmarks orientés SLA complète le benchmarking traditionnel en guidant la démarche de test par les contraintes explicitées dans le SLA (chagre de travail, performances attendues). On espère ainsi respecter le SLA en se fondant de véritables comportements de l'application.

CLIF [Dil09, Pro] est un système logiciel de benchmarking, fondé sur le modèle à composants Fractal et écrit en Java. Environnement de benchmarking, il est spécialement conçu pour s'adapter à tout type de système à tester en termes de ressources comme en termes de méthodes d'utilisation du logiciel du SUT. On trouvera dans [Dil09] une présentation détaillée de l'architecture de CLIF. SelfBench [Ben12] est un outil de test automatique destiné à déterminer les capacités maximales de traitement d'un SUT. Il est fondé sur CLIF et utilise ses composants fondamentaux (sondes (*blade*), injecteurs, mécanismes de configuration). Mais SelfBench contrôle pendant son fonctionnement le comportement du SUT et augmente de manière contrôlée la charge soumise au SUT, jusqu'à apparition d'un phénomène de saturation du SUT. On obtient

ainsi les performances maximales que peut assurer le SUT.

1.2 PSLA, un langage de description pour les PaaS

Le prestataire de niveau SaaS s'engage avec le fournisseur de PaaS sur un SLA de niveau PaaS (PaaS-SLA), niveau qui correspond à nos travaux. Du point de vue du fournisseur de PaaS, le SLA de niveau PaaS est donc fondamental pour qu'il puisse assurer sa fonction, en particulier en ce qi concerne la capacité d'adaptation à la charge de l'application, appelée élasticité.

PSLA [LPM14] est un langage de description de PaaS-SLA fondé sur WS-Agreement [ACD+07]. WS-Agreement est lui-même un modèle extensible de description de SLA. Après avoir passé en revue nombre de propositions, il nous semble que WS-Agreement constitue une base adaptée aux besoins particuliers d'un PaaS-SLA. À notre connaissance, nous sommes les premiers à proposer un langage de description de PaaS-SLA effectivement utilisable par des outils logiciels.

1.2.1 Description sémantique de PSLA

Un PaaS-SLA comprend deux types d'informations : des informations générales sur la SLA nommées Métadonnées SLA (*Metadata*) et une description détaillée de la QoS désignées par le terme comme Critères de QoS (*QoS criteria*). Dans la suite, nous utiliserons l'application Springoo pour illustrer notre propos. C'est une application à trois étages (*tiers*), employée comme cas d'utilisation dans le projet OpenCloudware. Le premier tiers de Springoo est un serveur Apache, le second tiers est un serveur d'application JavaEE Jonas et le troisième est un serveur de bases de données MySQL. Nous pouvons instancier plusieurs serveurs d'applications Jonas. Ce tiers possède donc la capacité de changement d'échelle horizontale (*horizontal scaling*). Nous pouvons changer les ressources attribuées au troisième tiers (CPU ou mémoire centrale) qui dispose ainsi de la capacité de changement d'échelle vertical (*vertical scaling*). Nous donnons ci-dessous une description des clauses utilisées dans PLSA.

Metadonnées de SLA

Les métadonnées SLA définissent les informations de base du SLA. Ces informations sont valides pour le SLA dans son ensemble. Les métadonnées SLA comprennent

- Agreement Name : c'est l'identification (unique) de l'accord, par exemple "Springoo".
- Agreement Initiator : c'est la partie qui commence la demande de création de l'accord. Par exemple, l'initiateur de cet accord est Développeur Springoo.
- Agreement Responder : c'est la partie qui répond à la demande de création d'accord. Par exemple, le répondant est OpenCloudware.
- Service provider : c'est la partie qui met à disposition les services concernés par l'accord. L'autre partie est le consommateur de services. Par exemple, le prestataire de services est également OpenCloudware.
- Agreement Expiration Time : c'est la date à partir de laquelle l'accord n'est plus valide. Toutes les périodes de validité définies dans les QoS Criteria devraient être des sous-ensembles de la période de temps définie par cette date.

Critères de qualité de service (QoS)

L'un des objectifs importants d'un SLA est de donner des garanties concernant les QoS délivrés par le fournisseur de service. Nous détaillons ici les clauses correspondantes, en expliquant les droits et obligations de chaque partie.

Valid Period : spécifie la période de temps pendant laquelle ce critère de QoS critère sera pris en compte. Elle peut être soit continue ou composée d'intervalles temporels réguliers. On la décrit par une heure de début, une heure de fin et un intervalle de temps facultatif.

Valid Workload : décrit pour quels types de charge de l'application le critère de QoS sera appliqué. Elle est décrite par : la taille des données, la constitution des données et la variation des données.

Service Level Objective : c'est l'élément central du critère de QoS. Il exprime ce qu'est exactement l'objectif de qualité de service en quantifiant la plage de valeurs de la métrique de qualité de service correspondante. Comme il est souvent difficile pour le consommateur lui-même, de spécifier une valeur correspondant à une satisfaction ou une insatisfaction, nous utilserons les notions de Confiance (*Trust*) et de valeur floue (*Fuzziness*) pour permettre la description d'un seuil flou.

Metric Description : pour un critère de QoS, un Objectif de niveau de service (*Service Level Objectif*, SLO) quantifie ce qui est contractualisé par une plage de valeurs de la métrique. Les paramètres du SLA sont décrits pour assurer qu'il n'y a pas d'ambiguïté sur la métrique entre les parties (voir la section 5.2.2). La description comporte une unité, un type de valeur, une source de valeurs et le mode de mise à jour de la valeur.

Business Values : ce sont des informations de type commercial qui différencient un SLA d'un autre, comme par exemple le niveau de pénalité ou de récompense. Ces valeurs seront utilisées dans le système de facturation pour calculer le coût du service.

Check Plan : le système de facturation ayant besoin de vérifier la qualité du service et le service lui-même tels qu'énoncés dans le SLA, cette valeur indique quand cette vérification doit être effectuée.

Priority : le prestataire de services ne dispose pas toujours de suffisamment de ressources pour assurer tous les SLO en même temps. On précise donc la priorité relative de chaque SLO.

1.2.2 Fonctionnalités de PSLA

PSLA est un langage de de PaaS-SLA, donc conçu pour décrire le contrat entre le fournisseur de PaaS et le « propriétaire » de l'application. Le fournisseur de PaaS prend en charge l'approvisionnement dynamique des ressources nécessité par la fluctuation de la charge de travail. Habituellement, les propriétaires d'applications ont des objectifs antagonistes comme la disponibilité de ressources ou un coût maximum pour maintenir la qualité de service. Par conséquent, l'application ne pourra probablement pas fonctionner si la charge de travail n'est pas limitée. Les métriques doivent être décrites sans ambiguïté pour que les objectifs de QoS qui doivent être garantis par le fournisseur de PaaS, puissent être clairement définis. Inspiré par CSLA [KL+12], qui définit une limite floue pour la plage de QoS acceptable pour la gestion effective de l'application, nous introduisons également la notion de plage de valeurs floue dans PSLA.

Charge élastique

L'élasticité à un instant donné peut être exprimée par un vecteur à quatre dimensions (s, c, t) où :

— s, la taille des données, désigne la quantité totale de données pendant une unité de temps donnée.
— c, la composition de charge de travail, donne la proportion de chaque type de charge qui forment la charge de travail. La composition de charge de travail lui-même peut être décrit par un vecteur et chaque dimension peut être définie par les ressources requises par une quantité unitaire de la charge. Les détails sont décrits ci-dessous.
— t, le temps, désigne le moment où la charge de travail est observée.

Différents services ont besoin de divers types de ressources et en quantités différentes. Du point de vue des fournisseurs de PaaS, les besoins en ressources sont le point essentiel pour assurer la qualité de service garanti. Nous classons donc les charges en fonction de la quantité de chaque type de ressources exigées sur chaque composant (nous supposons que le système logiciel est à base de composants). Comme il existe plusieurs types de ressources (toujours en quantité limitée), par exemple la mémoire centrale, nous pouvons résumer les métriques de ressources disponibles et leurs utilisations par deux vecteurs. Dans nos travaux, nous nous limitons à trois ressources (CPU, RAM, NET) (NET pour *Network*). Cette limitation pourra être levée dans de futurs travaux. Notons que la charge de travail étant composée de plusieurs charges élémentaires, différents ratios de charges peuvent conduire à des répartitions différentes des ressources nécessaires.

Métriques

Pour modéliser les métriques (ou mesures), il nous faut d'abord déterminer les plus significatives pour le niveau PaaS. Nous en déduisons ensuite une abstraction et nous définissons une structure capable d'exprimer les propriétés nécessaires de ces métriques. Pour les PaaS-SLA, pour autant que nous le sachions, il n'existe pas de modèle de métrique. Dans [AAA+10], certains des indicateurs de performance couramment utilisés, y compris le débit, la fiabilité, l'équilibrage de charge, la durabilité, l'élasticité, la linéarité, l'agilité, l'automatisation et le temps de réponse du service sont répertoriés. Certaines métriques du PaaS-SLA dérivent de métriques du niveau SaaS, comme le temps de réponse, que nous prenons en considération.

Nous considérons d'abord les propriétés suivantes :

— Unit : unité de mesure de cette métrique. Les unités communément utilisées, comme la seconde, sont intégrése dans PSLA, mais on dispose aussi du mécanisme d'extension pour des unités définies par l'utilisateur.
— Value type : le type de la valeur de la métrique, par exemple chaîne, entier ou décimal.
— Value source : la source d'acquisition de cette métrique. PSLA permet d'utiliser un identifiant de ressource uniformisé (*Uniform Resource Identifier*, URI) pour décrire l'emplacement de la valeur de la métrique.
— Value update mode : la façon dont la nouvelle valeur de la métrique est obtenue à partir de sa source.

Plage de valeurs floue

Nous utilisons les expressions de logique élémentaire pour décrire la plage de valeurs. Nous définissons des opérateurs logiques, les opérateurs de comparaison et les valeurs limites. Comme il est difficile de définir des valeurs limites exactes concernant le fonctionnement réel, nous utilisons les notions de «flou» (*fuzzyness, fuzzy*) et de «confiance» (*confidence*) comme dans [KL+12]. Le terme *confidence* est cependant rebaptisé *trust* («confiance») pour le distinguer du terme statistique confidence.

— Opérateurs logiques : $\cap$, $\cup$ et $\neg$;
— Opérateurs de comparaison : $<$, $>$, $\leq$, $\geq$ et $=$;
— Metric : à quelle métrique ces plages de valeurs s'appliquent ;
— Valeurs limites : bornes des plages de valeurs ;
— Trust : quel pourcentage minimal de valeurs de la mesure doivent être dans la plage de valeurs décrite ;
— Fuzziness : le degré acceptable de déviation d'une valeur par rapport aux limites.

1.3 Étude de faisabilité fondée sur le benchmarking

L'utilisation efficace d'un SLA établi pour répondre aux besoins du client (initialement énoncés au début de la négociation) suppose que l'on peut satisfaire ce SLA à l'exécution. L'étude de faisabilité du SLA évalue si la cible de QoS peut être atteinte avec des ressources précisées avec une charge de travail spécifiée. L'étude de faisabilité du SLA sera appliquée à plusieurs reprises au cours du processus de négociation du SLA. La modélisation du taux d'arrivées des requêtes de service et de la demande adaptée de ressources pour satisfaire la cible de QoS sont les fondements de l'évaluation de capacité (*Capacity planing*). Cette modélisation peut être obtenue soit par un modèle de simulation, soit par un modèle analytique. Puisque nous voulons concevoir une méthode *non intrusive* par rapport à l'application, notre modèle de comportement de l'application sera basé sur les résultats de benchmarks réalisés dans l'environnement PaaS cible, en utilisant des mesures, disponibles également à l'exécution en production. Nous proposons donc pour le gestionnaire de PaaS, une méthode d'étude de faisabilité de SLA, indépendante de l'application et efficace, fondée sur le benchmarking.

Dans le cadre de cette thèse, nous considérons des applications dont l'architecture est conforme au modèle *Virtualized Componentized Application Architecture* (VCAA). Ce modèle définit une application à bas de composants regroupés dans des étages (*stage*). Les interactions entre composants (appel et réponse de service entre composants) sont synthétisés en liaisons entre étages. Un étage lui-même est instancié en machines virtuelles (*Virtual Machine*, VM) toutes identiques pour un étage donné. Rappelons qu'une VM est une entité informatique en exécution, associant une image disque (un fichier) contenant le système de fichiers d'une « machine » ayant déjà été exécutée et prête à continuer son exécution, et un ensemble de ressources virtuelles qui seront allouées lors de l'exécution de la VM. Nous appelons modèle virtuel de ressources physiques (*flavor*) cet ensemble de ressources. On trouvera dans la section 4.2.2 une description détaillé du modèle VCAA.

1.3.1 Description du problème

Une étude de faisabilité de SLA vise à évaluer si une proposition de SLA est acceptable ou non. L'évaluation prendra en compte toutes les contraintes définies dans le SLA proposé et vérifiera si la proposition est cohérente.

Parmi les difficultés de ce type d'étude, le temps nécessaire (et/ou disponible !) et le coût de l'étude sont des contraintes très fortes. Par exemple, le taux maximal d'arrivées des requêtes peu nécessiter une quantité très importante de ressources, ce qui entraîne un coût élevé pour disposer, par exemple, de centaines de machines virtuelles. Nous avons donc développé un processus d'étude de faisabilité afin de réduire autant que possible le coût d'étude.

Comme décrit dans la section précédente, en PSLA, la partie centrale est GuaranteeTerm. Dans GuaranteeTerm, la partie QualifyingCondition contient la définition des objectifs de QoS. Les ServiceLevelObjectives définissent les (niveaux des) objectifs de QoSe tels que par exemple : temps de réponse < 3 s. BusinessValueList donne des informations pour la facturation comme les contraintes de coût. Dans le contexte de PSLA, l'objectif de l'étude de faisabilité est d'évaluer si, avec les contraintes de ressources définies dans QualifyingCondition et les contraintes de coût définies dans BusinessValueList, la plage de charge de travail définie dans QualifyingCondition peut être servie avec les exigences de QoS définies dans ServiceLevelObjective.

1.3.2 Dimensionnement des machines virtuelles

Un support matériel de machine virtuelle (*flavor*) possède de nombreux types de ressources. Nous considérons principalement les ressources CPU et mémoire. Par conséquent, la définition de la notion de *plus puissante* flavor est un problème multicritère. Nous pouvons seulement dire qu'une flavor a est plus puissante qu'une autre b lorsque les deux quantités de ressources, CPU et mémoire, sont plus grandes pour a que pour b. Si les deux grandeurs (CPU, mémoire) ne sont pas ordonnées de la même manière, il est impossible de dire *a priori*, que la flavor a est plus puissante que b car les performances de la flavor dépendent du type d'utilisation des ressources par l'étage donné de l'application. Nous introduisons donc un mécanisme de comparaison de flavors par étage d'application (étape FLAVOR).

Comparaison des flavors par étage

La comparaison de flavors doit être effectuée séparément pour chaque étage. On note MFC (*Maximal Flavor Capability*), la limite supérieure du taux d'arrivées global des requêtes si l'on utilise une flavor donnée pour un étage donné. MFC peut être considérée comme une mesure de la flavor. Nous disons que pour un étage donné, une flavor est plus petite (ou égale) à une autre lorsque sa MFC est plus petite (ou égale). Les valeurs de MFC peuvent être déterminées par des benchmarks. Toutefois, le calcul de certains MFCs sans benchmark, permettrait de réduire le coût de l'étude de faisabilité du SLA. L'obtention de ces MFCs calculés est réalisée selon le niveau des ressources et du MFC obtenu par benchmark avec des flavors dont les ressources sont utilisées au maximum (VM ou flavors dites saturées). Il est nécessaire d'obtenir la flavor saturée pour effectuer une déduction crédible de MFC parce que notre calcul utilisera le rapport de ressources minimal comme rapport du taux d'arrivées maximal. Toutefois, si le rapport de ressources minimal correspond à une ressource non saturée, il ne suffit pas de dire que le taux d'arrivées maximal sera plus petit. Par exemple, la flavor A a 2 vCPU et 4 GO mémoire. 80

% de vCPU mais seulement 10 % de la mémoire sont utilisés au cours du benchmark , ce qui signifie probablement que 0,5 GO de mémoire sont suffisants en prenant 80 % comme seuil d'utilisation maximal de la mémoire. La flavor B a 4 vCPU et 2 GO de mémoire. Le rapport de ressources minimal est de 0,5 (mémoire). Toutefois, 2 GO de mémoire sont probablement suffisants pour une flavor dont les 4 vCPU soient saturés puisque seulement 0,5 G de mémoire sont suffisants pour 2 vCPU. Par conséquent, B devrait être en mesure de servir un plus grand taux d'arrivées et son MFC doit être plus grand aussi. C'est la raison pour laquelle on essaie de sélectionner une flavor provoquant la saturation de toutes les ressources pendant le processus de benchmarking (étape 3 ci-dessous).

Après avoir trouvé les flavors appropriées, nous pouvons comparer les flavors et dire si une flavor est plus grande (ou plus petite) qu'une autre. Nous considérons le rapport de ressources minimal entre une flavor et une flavor testée saturée. Plus faible est la valeur de ce rapport, plus la flavor sera petite. Si le rapport est plus petit que 1, la flavor est considérée comme plus petite que la flavor saturée testée. Si le rapport est plus grand que 1, la flavor est considérée comme plus grand que la flavor saturée testée. La flavor avec le plus grand rapport minimal est considérée comme la plus grande flavor (pour un étage donné).

Toutes les flavors disponibles ne peuvent pas être saturées. Même la plus grande flavor peut ne pas être saturée pendant les benchmarks. Le MFC de la plus grande flavor non saturée sera utilisée pour déduire les MFC des petites flavors. Toutefois, le rapport utilisé pendant le calcul reste celui obtenu par comparaison avec le MFC de flavors saturées obtenus par benchmarking. Par conséquent, la situation mentionnée ci-dessus ne pose aucun problème.

Comparaison de flavors indépendamment de l'étage

La comparaison de flavors indépendamment des étages est une comparaison d'un point de vue purement matériel. Le niveau des ressources mémoire allouée par unité de CPU pour $flavor_s^i$ est $Ratio_s^i = \frac{memory_s^i}{cpu_s^i}$. Nous définissons l'écart de configuration entre $flavor_s^i$ et la flavor sans ressources CPU ni mémoire par $Distance_s^i(0,0) = \sqrt{{memory_s^i}^2 + {cpu_s^i}^2}$. Lorsque deux flavors ont le même $Ratio_s^i$, $Distance_s^i(0,0)$ peut être utilisé pour indiquer la taille de flavors et on pose $Distance_s^i(mem, cpu) = \sqrt{(memory_s^i - mem)^2 + (cpu_s^i - cpu)^2}$. $Distance_s^i(mem, cpu)$ est l'écart de configuration entre $flavor_s^i$ et une flavor avec $vCPU$ cpus et mem GO de ressources mémoire.

Plus $Distance_s^i(mem, cpu)$ est petit, plus proches sont $flavor_s^i$ et la flavor avec mem octets de mémoire et cpu de ressource CPU. Lorsque $Distance_s^i(mem, cpu) = 0$, $flavor_s^i$ a la même configuration que la flavor avec mem octets de mémoire et cpu de ressource CPU. $Distance_s^i(0,0)$ est une description quantitative des différences de taille entre $flavor_s^i$ et la flavor avec une configuration de mémoire et 0 0 CPU. $Distance_s^i(0,0)$ peut être considérée comme une description de la taille de $flavor_s^i$. Nous disons que $flavor_s^i$ est plus petite qu'une autre flavor B lorsque $Distance_s^i(0,0)$ est plus petit que celle de B. Pour autant, $Distance_s^i(0,0)$ ne suffit pas pour décrire la capacité de traitement de $flavor_s^i$.

Par exemple, la flavor avec 10 vCPU et une mémoire de 1 GO a la même valeur de la distance que la flavor avec 1 vCPU et 10 GO de mémoire mais les capacités de ces deux flavors peuvent être très différentes. Lorsque $Ratio_s^i$ est plus élevé pour une flavor A que pour une flavor B, , nous pouvons dire que A attribue plus de ressources mémoire à chaque unité de CPU. Nous pouvons également dire que deux flavors avec le même rapport attribuent la même

quantité de ressources mémoire pour chaque unité de CPU. Dans ce cas, la flavor A avec une plus grande valeur de $Distance$ qu'une flavor B est plus puissante que B. Une flavor peut être choisie comme la configuration de ressource équilibrée.

Le terme « Configuration de ressources équilibrée » signifie que les différentes ressources ne sont pas sensiblement supérieures ou inférieures à une autre. Aucune configuration standard ne peut être considéré comme équilibrée. De plus, la quantité de CPU et la quantité de mémoire ont différentes unités de mesure. Par conséquent, il est également difficile de spécifier une valeur de ratio pour représenter l'état de « configuration de ressources équilibrée ». Nous définissons la flavor avec une valeur médiane de ce ratio parmi toutes les flavors disponibles pour un étage s comme la configuration de la ressource la plus équilibrée pour s. Lorsque l'on calcule la valeur médiane, au lieu de prendre plusieurs même valeur du rapport en considération, une valeur du rapport est compté une seule fois pour éviter que de nombreux ratios déséquilibrés mènent à valeur médiane biaisée.

$Ratio_s^i$ et $Distance_s^i(0,0)$ décrivent la proportion de l'allocation de ressources et la quantité de ressources attribuées à $flavor_s^i$. $Ratio_s^i$ et $Distance_s^i(0,0)$ synthétisent la configuration des différentes ressourceset sont utilisés avant que la comparaison de flavors pour un étage donné soit disponible. Ils ne suffisent pas pour définir la flavor la plus puissante et nous ne pouvons pas décrire quantitativement la puissance intrinsèque d'une flavor pour un étage. Après benchmarking, nous pouvons décrire quantitativement par MFC, la capacité d'une flavor pour chaque étage.

1.3.3 Capacité maximal d'une flavor

La capacité maximale d'une flavor (*Maximum Flavor Capability*, MFC) est au cœur de cette méthode d'étude de faisabilité de SLA fondée sur les benchmarks. MFC_s^f est la limite supérieure (en requêtes par seconde) de la vitesse des arrivées de requêtes globales lorsque l'on utilise une flavor f en étage s, assurant que la QoS voulue est atteinte. On note $MFC_s^f(n)$ la limite supérieure du taux des arrivées globales pour n instances de la flavor f en étage s. On a $MFC_s^f(n) = n * MFC_s^f$ et $MFC_s^f = MFC_s^f(n)/n$

MFC dépend généralement de l'étage considéré : une flavor utilisable sur deux étages distincts produira probablement des taux d'arrivées maximaux différents puisque la même demande exige des ressources différentes sur les deux étages. MFC_s^i change lorsque les exigences de QoS changent. Par exemple, toutes choses égales par ailleurs, passer d'un temps de réponse maximal de 3 secondes à 3 minutes permet probablement d'obtenir un taux d'arrivées plus élevé. Par conséquent, une valeur de MFC_s^i est également liée à une périoide de temps pendant laquelle le SLA est stable. Cependant, puisque nous déterminons les MFC à SLA et charge fixés, nous les mentionnons pas dans la suite.

Si l'on utilise n instances d'une flavor i pour un étage s, la limite supérieure du taux d'arrivées maximal sera n fois MFC_s^i. Les MFC_s^i de tous les étages ainsi que le nombre d'instances utilisées à chaque étage permettent de déterminer la limite supérieure du taux d'arrivées maximal de l'ensemble du système. Ce taux est fonction de l'étage qui constitue le goulet d'étranglement. On peut considérer que le(s) MFC(s) de cet(ces) étage(s) sont les taux maximaux du système dans la configuration donnée. Ainsi, nous prenons comme limite supérieure du taux d'arrivées globales du système, la plus petite valeur de taux d'arrivées maximaux de ces étages.

1.3.4 Méthode d'étude la faisabilité d'un SLA

Nous avons proposé une méthode SLA de l'étude de faisabilité fondée sur les benchmarks. La description détaillée de notre méthode est donnée en section 6.3 et en section 6.4. Les analyses de type « Que se passe-t-il si ? » (*what-if analysis*) répondent à la question de savoir si la cible de QoS peut être atteinte avec les valeurs de charge de travail, de ressources allouées et de contraintes de coût énoncés dans le SLA. Les benchmarks permettent une analyse du comportement de l'application, par exemple déterminer le débit des requêtes servies et leur temps de réponse. Notre méthode de l'étude de faisabilité de SLA par benchmarking réalise un compromis entre la précision et les coûts en limitant le nombre de tests pour établir une correspondance charge-configuration. Cette correspondance sera utilisée dans notre méthode de contrôle à l'exécution.

1.4 RCSREPRO - contrôle à l'exécution

Le but du contrôle à l'exécution est de satisfaire le SLA pendant l'exécution tout en évitant autant que possible de sous-employer des ressources. Notre approche du contrôle est donc guidée par le SLA. Notre méthode est générique, mais nous expliquerons le problème du contrôle à l'exécution en nous basant sur PSLA. Après la phase d'étude de faisabilité du SLA, les ressources disponibles sont en mesure de servir la charge de travail prévue avec une QoS définie et pour un coût précisé. Pour garantir le SLA à l'exécution, le plus sûr moyen est de toujours déployer le maximum de ressources. Cependant, cette stratégie alloue bien entendu en général plus de ressources que nécessaire à l'application.

Nous proposons RCSREPRO (*Runtime Control method based on Schedule, REactive and PROactive methods*), une méthode de contrôle à l'exécution fondée sur l'ordonnancement, les méthodes de contrôle à l'exécution réactives et proactives. RCSREPRO nous permet de bénéficier de la capacité de variation des infrastructures de cloud et d'allouer et de libérer automatiquement des ressources aux applications selon l'évolution des besoins. En plus d'ordonnancer la mise à disposition de ressources en fonction de la charge de travail et des informations figurant dans le SLA avant l'exécution, RCSREPRO prévoit également les fluctuations de la charge de travail afin de mettre à disposition de manière proactive les ressources qui seront nécessaires. RCSREPRO effectue une prédiction de la charge de travail en utilisant l'outil WCF 7.3.2 de prévision dynamique basée sur les séries chronologiques. MFC, le modèle de correspondance charge de travail- ressources attribuées, qui est utilisé dans RCSREPRO, est initialement construit pendant l'étude de faisabilité du SLA et amélioré en permanence par suivi de l'exécution afin d'affiner la qualité du contrôle.

1.4.1 Défis

L'allocation de ressources adaptées en fonction du contexte et d'un SLA est un travail difficile. Nous identifions quatre défis : la fluctuation de la charge de travail fournie à l'application, le contrôle des dépassements de charge, l'adéquation d'une configuration de ressources à une charge donnée pour une application donnée, la variabilité de l'environnement d'exécution à configuration fixée.

Dans la plupart des applications placées dans le Cloud, la charge soumise varie dans des proportions importantes, soit de manière prévisible, soit de manière imprévisible. Ce dernier

cas nécessite qu'un contrôle à l'exécution soit mis en place, tenant compte de cette variabilité.

Même si le SLA prévoit des limites supérieures aux taux d'arrivées des différents types de requêtes soumises à l'application, il est souhaitable de contrôler cette charge entrante afin de ne pas saturer le système. Des procédés relevant des méthodes de contrôle d'admission sont donc nécessaires.

Notre travail se voulant générique, nous devons définir des mécanismes de contrôle applicables quelque soit l'application placée dans le Cloud (mais conforme au schéma VCAA). Ces mécanismes s'appuieront sur des résultats de benchmarks de l'application mais seront indépendants de l'application considérée.

Enfin, même à charge fixée, l'application peut réagir de manière variable en raison des fluctuations de son environnement. Par exemple, des VMs de l'application peuvent être en exécution sur un hôte dont l'hyperviseur est équipé de fonctions fines d'allocation de ressources. Ainsi, la présence de VMs très consommatrices de ressource sur cet hôte peut ammener l'hyperviseur à réduire les ressources normalement attribuées aux VMs de notre application. Un mécanisme de contrôle réactif doit donc être mis en place, puisque ces variations sont exogènes pour l'application.

À partir d'un SLA, on peut déterminer les périodes temporelles dans chacune desquelles les SLO et leurs valeurs sont stables. RCSREPRO fonctionne sur une période stable de SLOs. Lorsque l'on passe d'une période à l'autre, les différents éléments utilisés par RCSREPRO doivent être adaptés. Si le contexte de la nouvelle période n'a jamais été rencontré, cela exigera de reconstruire ces éléments et donc, potentiellement de réaliser de nouveaux benchmarks par exemple. Le processus d'amélioration continu utilisé dans RCSREPRO devra aussi être réinitialisé.

1.4.2 Échelles temporelles

Afin d'assurer la meilleure précision possible au système de contrôle à l'exécution, les modifications de configuration doivent être réalisées dès que possible lorsqu'une condition de contrôle change. Les métriques ou indicateurs de performance sont généralement fournis par des systèmes de surveillance et les échelles de temps de délivrance de ces informations sont variables, pouvant aller de quelques secondes quelques minutes. Par exemple, les indicateurs de performance dans Amazon-EC2 peuvent être obtenus toutes les minutes *via* le service AWS CloudWatch [Ser15]. Par ailleurs, une VM prend habituellement au moins 5 minutes pour démarrer et le système d'exploitation peut également avoir besoin de plusieurs minutes pour s'adapter à la nouvelle configuration des ressources. L'intervalle de temps entre deux actions d'adaptation devrait donc être au moins le temps total de démarrage des VMs et de la disponibilité d'une nouvelle configuration des ressources. Il est aussi nécessaire d'éviter de trop fréquentes actions d'adaptation des ressources pour laisser le système tirer partie de ces actions.

Les méthodes de facturation des VMs varient d'un fournisseur à l'autre (voir par exemple [LBMAL12] pour un raffinement de la demande de VM selon les modes de facturation, ou [Qin15] pour une facturation à la seconde). Dans notre méthode, nous supposons que l'unité temporelle de facturation pour l'utilisation d'une VM est inférieur au temps de démarrage d'une VM.

La dimension proactive de notre méthode de contrôle à l'exécution est fondée sur la classification et la prévision de la charge par des méthodes d'analyse de séries temporelles appliquées à la charge soumise au système. Nous utilisons pour cela le système logiciel de prévision dyna-

mique de charge WCF (*Workload Classification and Forecasting*) [HHKA14]. Nous renvoyons le lecteur à la section 7.2.2 pour une présentation détaillée de WCF. Une connaissance *a priori* du comportement de la charge, donc des comprtements des utilisateurs de l'application, permet d'accélérer l'efficacité de WCF, mais cette connaissance n'est pas disponible, WCF pourra produire des prévisions fiables après une période plus ou moins longue d'observation de la charge.

Au final, nous avons donc trois échelles de temps en jeu dans le contrôle à l'exécution. La plus grande est définie par les expressions Guarantee terms du SLA. L'échelle intermédiaire correspond aux fréquences des prévisions (système WCF). La plus petite échelle est associée à l'intervalle de temps nécessaire pour effectuer des actions de reconfiguration des ressources.

1.4.3 Architecture système du contrôle à l'exécution

Notre système de contrôle à l'exécution est un système informatique autonome autogéré conforme à l'architecture MAPE-K de boucle de régulation MAPE-K [C$^+$06]. Cette architecture est rappelée en détails dans la section 4.3.1. Nous présentons ici un résumé de notre architecture dont on trouvera les détails en section 7.3.

La surveillance (*monitoring*) du système est assurée à la fois par la prise de mesures sur les indicateurs de bout en bout de l'application définis dans le SLA et les indicateurs d'utilisation des ressources donnés par le système d'exécution des VMs.

La partie analyse assure plusieurs fonctions. D'une manière réactive, elle compare les indicateurs internes d'exécution (usages de ressources) à l'aide d'un système de comparaison à des seuils. Les données de charge sont utilisées par le composant WCF pour prédire la situation de la charge de travail dans deux intervalles de temps dont la durée correspond au temps minimal pour une action sur l'allocation de ressources. L'ensemble des informations est aussi utilisé pour affiner le modèle de correspondance charge-configuration qui constitue le coeur de la base de connaissances (*knowledge*) de notre boucle MAPE-K.

La partie planification calcule les actions d'adaptation dans les trois modes : contrôle fondé sur un ordonnancement déduit du SLA (modifications planifiées des SLOs), contrôle réactif et contrôle proactif. La partie réactive agit en fait de manière indirecte en adaptant la correspondance charge-ressources pendant le fonctionnement. La partie proactive est elle assurée par la prévision de charge. Les résultats de la planification sont traduits par des artefacts techniques qui dépendent de l'environnement sous-jacent au PaaS. Par exemple, dans le projet OpenCloudware, on définit une nouvelle configuration dans un fichier texte conforme à la syntaxe OVF++ (voir par exemple [SEDP$^+$13]) qui est transmis à l'outil d'exécution de reconfiguration VAMP qui lui-même dialogue avec les IaaS en jeu.

1.5 Expérimentations

1.5.1 Buts des expérimentations

Nous avons expliqué que notre méthode comporte une partie proactive fondée sur la prévision de charge. Dans les expérimentations que nous réalisons, nous supposons que la précision des prédictions fournies par le système WCF est suffisamment bonne pour que l'on puisse considérer qu'elles décrivent exactement le futur de la charge. Ceci nous a permis de ne pas (encore) déployer WCF pour nos expérimentations. Nous nous concentrons donc dans ces expérimenta-

tions sur la vérification de la pertinence du modèle MFC de correspondance charge-ressources allouées et l'amélioration continue de ce modèle pendant l'exécution.

1.5.2 Application synthétique

Nous avons développé une application, nommée « Rubberband » (élastique), pour réaliser nos expérimentations. Rubberband est architecturé selon le modèle VCAA et une requête de service définit la charge de travail demandée à chaque étage tant en nature (CPU, mémoire, etc.) qu'en intensité. Chaque étage n'assure qu'un service. On peut donc dire que RubberBand constitue une synthèse d'une véritable application conforme au modèle VCAA, les différents appels de service de l'étage s à l'étage s' étant regroupés dans l'unique appel de s à s' dans RubberBand.

1.5.3 Environnement d'expérimentation

Nous utilisons deux infrastructures d'expérimentation : une interne au LISTIC, l'autre, la plateforme OpenCloudware, disponible au sein des infrastructures Bull de Grenoble. Nos expérimentations actuelles ont été réalisées uniquement sur la plateforme du LISTIC et seront portées sur la plateforme OpenCloudware prochainement. La plateforme LISTIC est fondée sur OpenStack et implantée sur un (et bientôt deux) nœud(s) de calcul, installés dans les locaux du centre de calcul MUST de l'Université Savoie Mont Blanc à Annecy le Vieux. OpenStack [Jac12] est une plateforme IaaS open source fonctionnant sur un système d'exploitation Linux, complété par différents services. Nous utilisons en particulier le système de surveillance Ceilometer [Wik15a].

Concernant le benchmarking, nous utilisons le système open source (OW2) CLIF et son extension de recherche automatique de saturation SelfBench. On se rapportera au chapitre 3 pour une présentation détaillée de ces outils.

1.5.4 Conception des expérimentations

Notre expérimentation comporte trois séries d'expériences. La première série permet d'obtenir la valeur réelle des MFCs de chaque étage en utilisant SelfBench. Dans la deuxième série, nous employons CLIF pour vérifier que notre modèle de correspondance charge de travail - ressources fonctionne correctement en supposant que nos valeurs de MFC sont précises. Enfin, dans la dernière série, nous cherchons à vérifier que les MFC, qui sont au cœur de notre modèle de connaissances, peut être améliorée avec la partie réactive de notre méthode de contrôle à l'exécution.

Nous définissons un RubberBand simple pour la première série. Rubberband est composé de deux étages connectés séquentiellement. Chaque étape peut avoir une ou plusieurs machines virtuelles, et dispose donc d'un équilibreur de charge (*load balancer*). Chaque étage dispose au minimum d'une VM et d'une seule flavor. La charge est constituée d'un calcul plus ou moins intensif sur chaque étage.

Nous obtenons les MFC à travers un ensemble de tests avec Selfbench. Pour obtenir un MFC, nous avons besoin d'effectuer un test SelfBench, donc deux tests SelfBench seront nécessaires pour notre RubberBand. Dans ce test, l'application doit avoir le nombre minimal d'instances sur l'étage étudié et « suffisamment » d'instances sur les autres étages pour s'assurer

que les ressources sur l'étage étudié peuvent être pleinement utilisées. Nous obtenons au final MFC_1 et MFC_2.

Dans la deuxième série, nous vérifions que les ressources allouées sont suffisantes en utilisant les valeurs MFC_1 et MFC_2. Pour ce faire, nous utilisons CLIF 3 fois. Chaque test injecte une charge à taux d'arrivée constant et pour une configuration de RubberBand optimisée fondée sur MFC_1 et MFC_2. Les trois taux d'arrivées de requêtes sont définis par un premier inférieur au minimum de MFC_1 et MFC_2, un deuxième entre MFC_1 et MFC_2 et un dernier plus grand que le maximum de MFC_1 et MFC_2.

Avec la dernière série d'expérimentations, nous vérifions que les valeurs initiales (peut être imprécises) des MFCs peuvent être améliorées avec notre méthode RCSREPRO lorsque la prédiction de la charge est exacte. Nous utilisons pour cela les données de surveillance fournies par ceilometer. Nous modifions les valeurs de MFCs par nos observations issues de Ceilometer et nous observons le comportement du système de contrôle. On se reportera à la section 7.3.3 pour les détails. Nous vérifions ainsi que les valeurs imprécises des MFCs peuvent être améliorées avec RCSREPRO si MFC'_1 et MFC'_2 sont « proches » de MFC_1 et MFC_2.

1.6 Conclusion et travaux futurs

1.6.1 Conclusion

Dans cette thèse, nous nous sommes concentrés sur les problèmes de gestion de la QoS au niveau PaaS avant et pendant l'exécution. Nous avons introduit un langage, PSLA, de définition de SLA et de sa gestion après avoir rappelé la nécessité d'un tel moyen d'expression des propriétés des services et de leurs qualités. Ce langage correspond aux contrats entre fournisseurs du PaaS et clients du PaaS. PSLA est fondé sur WS-Agreement et emploie les technologies XML pour permettre une exploitation machine-à-machine (*machine to machine*) des contrats par les contractants.

Le benchmarking (tests) fournit une analyse du comportement des applications, par exemple débit de requêtes servies, temps de réponse d'une requête. Des tests permettent donc de savoir si les différents SLOs d'un SLA peuvent être satisfaits.

Dans cette thèse, nous proposons une méthode d'analyse de faisabilité de SLA fondée sur les benchmarks. Notre méthode réalise un compromis entre des tests exhaustifs, coûteux et une précision insuffisante résultant d'une faible couverture de tests. Nous en déduisons un modèle de correspondance charge de travail soumise - ressources allouées à l'application utilisé dans notre méthode de contrôle à l'exécution.

le contrôle à l'exécution vise à attribuer de façon appropriée des ressources aux applications tout en satisfaisant les contraintes définis dans le SLA. Les méthodes actuelles sont essentiellement de nature réactive où à base de modèles mathématiques de l'application et de son support d'exécution. Par ailleurs, la planification *a priori* ne peut seule assurer une adaptation à l'exécution. Nous proposons donc dans ce travail la méthode de contrôle à l'exécution RCSREPRO. RCSREPRO est une méthode fondée sur l'ordonnacement et des méthodes de contrôle à l'exécution réactives et proactives. RCSREPRO prend en compte les variations planifiées définies dans le SLA et gère les variations de charge dans le cadre du SLA de manière proactive en s'appuyant sur des prévisions, à base de séries temporelles, de cette variation. Les MFCs constituent le modèle de correspondance charge de travail - ressources utilisées dans RCSREPRO. Ils sont

initialement dérivés de benchmarks réalisés à travers une étude de faisabilité de SLA et sont améliorés en permanence (rétro-action) par observation du système en focntionnement.

1.6.2 Travaux futurs

Dans notre proposition actuelle, nous supposons que les requêtes à l'application sont homogènes, *i.e.* induisent une charge de travail de mêmes caractéristiques sur les différents étages de l'application. Une des premières extensions de nos travaux consistera à étendre notre méthode à une charge composite. Cette extension nécessite d'abord de définir une caractérisation des différents types de charge. Ceci étant réalisé, il s'agira d'étudier de quelle manière la notion de MFC peut être étendue pour prendre en compte les types de charges définis précédemment.

Chapter 2

Introduction

2.1 Cloud computing

2.1.1 The emergence of cloud computing

With the development of multi-core computer and parallel technique, the high performance computers in data center can provide much more services. Tasks, such as scientific computing, search engine, multimedia processing, require a lot of CPU, Memory, network or database resources. These tasks can be running in data centers to guarantee the performance. Clustering computing is a clustering system based network computing pattern. Clustering system is composed of independent computers connected by high speed networks and managed together by resource management software to provide an unified access interface and hide the underground running machines to end users. When too many task requirements come, limited clustered resources are intensively competed among tasks. The system may be overloaded and the tasks need longer time to finish. Grid computing [FK03] integrates geographically distributed computing resources to process large scale computing requirements. Resources are shared. Therefore, the duration of big task processing can be reduced and the one who contribute their resources can also acquire economic benefits. Users are motivated to provide their idle resource, but the dynamic and heterogeneity of the grid environment make it difficult to manage. Moreover, grid computing can't be widely used because it is designed for a specific kind of task. Cloud computing provides resources as a service. Infrastructures, platforms and applications are all treated as resources. With cloud computing, users can use these resources simply as using water and electricity services without caring about maintaining the resources.

2.1.2 Definition of cloud computing

Cloud computing is a new computing model. Infrastructure, application and data are moved from local machine to internet and provided as services. Cloud users greatly save budgets by buying only what they need. Cloud computing is not defined uniformly. [G$^+$09] summarized the cloud computing definition from 21 cloud experts. Klems thinks that the infrastructure which can increase or decrease resource allocation within several minutes or even several seconds to avoid over provisioning or under provisioning should be named as cloud. Many other researchers think that this is only a necessary but not a sufficient condition for cloud. From business aspect, Martin et al. [Mar10] think that cloud services should be "pay as you go" in the

scope of internet to extend the scale of computing resources. [BYV08] thinks that the biggest issue for cloud is the user friendliness as well as ability of delivery SLA guaranteed service. [VRMCL08] summarized different cloud computing definition as "Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs." Cloud should be defined from several aspects. [SJ12] summaries cloud characteristics from functional, non-functional and economic aspects. Functional feature include virtualization and elasticity. It can hide the complicated underground technical details and custom hardware and software configurations to dynamically adapt to the requirements of changing workload. Non-functional characteristics, such as flexibility, allow users to monitor and maintain several dynamic changing service environments to provide simplified interfaces and QoS guaranteed services. Economic characters, such as pay as you go, allow users to dynamically require and release computing, storage and network resources according to actual demand. In this way, the start-up cost and management cost can be greatly reduced.

2.1.3 Cloud service architecture

Stakeholders in cloud services

To understand the cloud service architecture, the stakeholders involved in cloud service should be clearly defined. [RJKG11] classified cloud service stakeholders from business point of view as cloud provider, enterprise user and end user. Cloud provider provides infrastructure, software development platform or software applications to enterprise users and end users. Enterprise users use the service interface provided by cloud provider to deploy and manage the application. End user is the one who uses the application service directly. Cloud provider provides IaaS, PaaS and SaaS web service interfaces based on standard protocol, such as SOAP and REST, and Service Oriented Architecture (SOA) [KBS05].

SaaS, IaaS, PaaS

According to the type of services, cloud services can be classified as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) as depicted in figure 2.1.

IaaS provide their clients virtual machines, storage space and network connectivity. It also provides services like load balancing and firewalls. The clients deploy and run a variety of software on the virtual machine, including software environment, such as operating system and application. The clients of IaaS get all of these services of infrastructure through internet. Amazon EC2, IBM Blue Cloud and Cisco UCS are three of the most representative IaaS products. Amazon EC2 provides different kinds of computing resources in the form of virtual machines. Its performance or stability has been met in enterprise-class needs. It provides API and Web management interface for users. IBM Blue Cloud is the first and technically advanced enterprise-class cloud computing solutions. It integrates enterprise's existing infrastructure by virtualization and automated management technology into cloud computing business centers. It manages hardware resources and software resources in a unified way. In other words, management, distribution, deployment, monitoring and backup are organized and centralized. Cisco

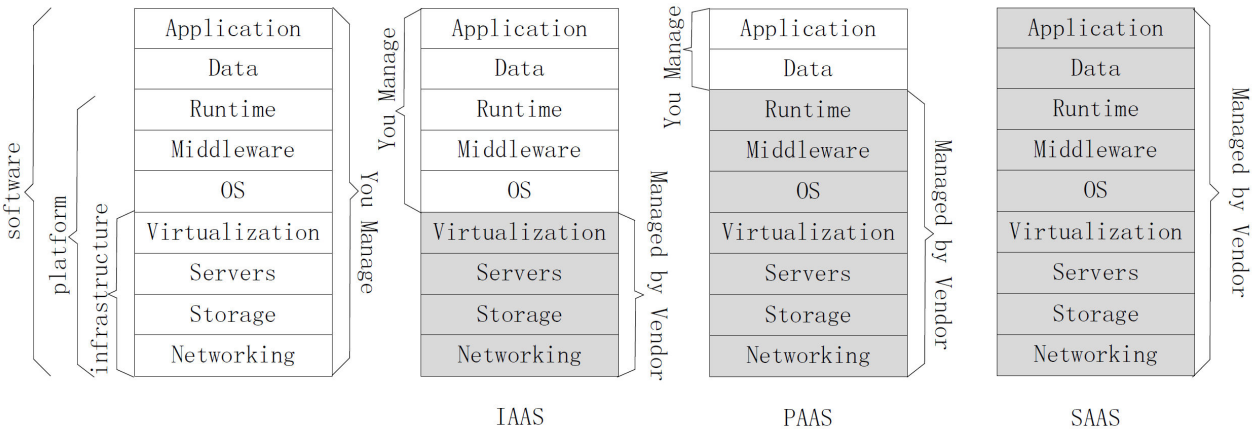

Figure 2.1. Three kinds of cloud services. [iaa15].

UCS allows Users to install VMware vSphere (cloud computing virtualization operating system provided by VMware) on UCS to support thousands virtual machines. Enterprises can use Cisco UCS to quickly build a cloud in local data center.

PaaS provides the software development platform as a service to the client. PaaSs speed up the SaaS development. PaaS usually includes operating system, runtime environment for the programming language, database and web server. Clients can deploy and run their own application on this platform. Clients have the full control of their application but they can not control and manage the underlying infrastructure. For example, they can deploy and run their own applications but they don't know where the application is running. PaaS provides a software engineering platform, including Software Development Kit (SDK for short), test environment and deployment environment to the client, so that the client can develop and deploy applications easily. Clients do not need to consider about the operations and maintenance on servers, operating systems, network and storage resources. The clients of PaaS are mainly developers. PaaS is the latest cloud service of three layers. The first PaaS platform was born in 2007. It is Salesforce's Force.com. Clients can use comprehensive development tools and frameworks to develop applications easily. Applications can be deployed directly into Salesforce infrastructure. By using its development environment and robust infrastructure, companies can deliver robust, reliable, and scalable online applications. In April 2008, Google launched the Google App Engine. It also supports Web applications. Google App Engine provides Google's infrastructure to deploy applications. It also provides a set of development tools and SDK to help application development. Windows Azure Platform is offered by Microsoft. It is running on Microsoft data center. It has scalable cloud operating system, data storage and networking. Windows Azure SDK provides a set of development, deployment and management tools and APIs for Windows Azure cloud services.

There are many technologies involved in PaaS. The main technology, includes REST, Parallel processing (MapReduce), multi-tenancy technology and general-purpose distributed memory caching system (Memcached). Representational State Transfer (short for REST) is a technology providing services to the client conveniently. Multi-tenancy allows a single application instance to be used by multiple clients with good isolation and security. Multi-tenancy can effectively reduce the cost of buying and maintaining applications. Parallel processing is a kind

of technology to process large amounts of data. Google's MapReduce is a representative work of using huge clusters to do parallel processing. Application server is also optimized to satisfy cloud computing, such as the Jetty application server for Google App Engine. Distributed cache is a kind of technology for reducing the load on the server and speeding up the response speed effectively. The most famous example is the distributed cache Memcached. Application server and distributed cache is essential for most PaaS. REST technology is usually used to provide an external interface. Multi-tenancy is mainly used behind SaaS application. Parallel processing technology is usually provided as a service.

SaaS(Software as a Service): SaaS means providing software through internet. Clients hire the software based on web instead of buying the software. SaaS providers install and run application in the cloud, and SaaS clients use web browser or other client to use the application. SaaS client can only configure the application, but can not manage the cloud infrastructure and platform underlying the application.

Public, private and hybrid cloud

Cloud can be classified into private cloud, public cloud and hybrid cloud architectures according to the deployment.

Public cloud is the most popular cloud computing model at this moment. It provides a cloud service opened to the public. It can support a large number of requests with low cost. Public cloud providers provide their clients with a variety of resources, including physical infrastructure, applications, software runtime environment and security, management, deployment and maintenance of these resources. When clients are using IT resources, they only need to pay resources and services they use. Though they share resources with other public cloud clients, they do not need to know about other clients or how the cloud platform is structured. Cloud provider is the only one who can control the physical facilities and make sure of the non-functional requirements like safety and reliability. Many big IT companies, like Amazon's AWS, Microsoft Windows Azure, Google's Google Apps and Google App Engine, provide their own public cloud services.

Public cloud has many advantages. Firstly, it serve for large scale of workload, so it makes the optimization of resource allocation possible. It may result in reducing operating costs of each workload. Secondly, using public cloud requires no upfront investment, and client can save money for managing and maintaining the IT resources. With the development of public cloud itself, the price of public cloud service will be lower and lower. Thirdly, compared to the computing resources required by one client, the public cloud provider can provide resources without limitation. So the client can be satisfied by the ability of the cloud in almost every case. Finally, public cloud can support most popular operating systems and hundreds of thousands of applications. Public cloud has also some shortcomings, for example, clients may worry about the data security since the data are not stored locally.

Although public cloud represents the general trend, for many enterprises, it is difficult to fully benefit from public clouds in a short time because of the restrictions and conditions inside the enterprises. Private cloud help these clients to benefit from the cloud technology. Private cloud service only used inside the enterprise. It works behind the firewall and can not be achieved by people outside. The IT staff in the enterprise is taking responsibility for the data, security and quality of service. Private cloud infrastructures are more dynamic and flexible than the traditional enterprise data centers. Building private cloud is a good way to reduce the com-

plexity of IT infrastructure. There are two kinds of private cloud products. One is provided by IBM and its partners such as IBM Blue Cloud. The other is provided by VCE (VMware, Cisco and EMC) such as Cisco UCS. Private clouds are mainly running inside the data centre of the enterprise. They are maintained by the IT department of the enterprise. These two aspects make the data more safe, the SLA is easily guaranteed, can make a good use of the existing hardware resources, the bygone applications can be easily adapted to the cloud, and IT department can better integrate the existing IT management processes with cloud. But the costs and expenses of building a private cloud is much higher than using public cloud, especially if the enterprise buys a private cloud solution directly from a big company. To maintain a private cloud, it is necessary to operate a professional cloud computing team which also costs a lot.

Hybrid cloud means combining public and private clouds together to keep the client's privacy and low the costs of building the cloud. By using hybrid cloud, enterprise can put the important or high security level parts of applications in their private infrastructure while others in the public clouds to decrease costs. Though hybrid cloud is not as popular as public cloud and private cloud, there are still some hybrid cloud products, like Amazon VPC (Virtual Private Cloud) which accesses some Amazon EC2 computing power behind the firewall but isolated with private computing power. VMware vCloud is also an example of hybrid cloud product. By using hybrid cloud, enterprises can own the privacy close to private cloud and the costs close to public cloud. It can quickly access to a large amount of computing resources in the public cloud when it is necessary. But there haven't a lot of hybrid cloud products to chose, and the operations of hybrid cloud are much more complicated. After all, privacy is not as good as private cloud and the cost is higher than public cloud.

2.1.4 Cloud Management

Compared to data centers, the biggest advantage of cloud computing is the management. Cloud management is the fundamental of operating IaaS, PaaS and SaaS. It provides information to learn about the cloud and to make decision in the cloud. It provides the technology to manage and maintain the cloud environment. It gives a safe and easily used interface of the cloud services to the cloud clients.

Although cloud services use standard protocols, such as SOAP and REST, as their web services interface, it is still a challenge of cooperating several services and providing effective service management. In literature [DTB10], [MKF10], [TAPB10], [LTCC12] propose hierarchical and modular cloud services management methods. The entire service architecture is composed of a three-layers structure. The top-layer is user interface. The middle layer is the service management. The bottom layer is the resource supply.

Resource supply layer provides service management layer with customized virtual resources above the physical resources through virtualization technology which can be classified as full virtualization technology and paravirtualization technology [LUC+05] according to the different levels of hardware virtualization. Multiple client servers (or say virtual machines) can be started in one single physical server with different operating systems by using virtualization technology. Virtual machines which share the same physical machine may have interlaced peak and idle periods so that resource utilization of the physical server can be increased. When multiple virtual machines on a single server have high load at the same time, part of the virtual machines can be migrated to another server to ensure virtual machine performance and avoid business interruption. On the contrary, when the load is too low, virtual machines can be

migrated to shut down the server and reduce energy consumption.

Service management layer is the most important layer of cloud service management systems. Service management layer mainly contains user management, image management, resource management, and security management. In the user management part, there are user authentication, access control, request management and billing management. In EC2, user identity authentication is done by X. 509 standard certificate mode [Qui]. Access control means authority management of reading and writing of resources. Billing management calculates the cost based on the user's resource usage monitoring information. Image management, including image database management, image creation and image deployment, aims at facilitating the deployment of complex applications to the cloud environment. Resource management includes resources demanding, load balancing, monitoring statistics and fault detection. Security management includes security auditing, access authorization and so on. Service management layer needs to choose the appropriate resource for requests. It also should be able to increase and decrease the size of resource pool dynamically to adapt to the changing workload and ensure Service Level Agreement (SLA).

The user interface layer consists mainly of service registration, service discovery, service composition, service access and service monitoring. Service registration means that cloud service developers can register to and use the service interfaces provided by service management layer to design and deploy an application in the resource supply layer. Service discovery helps a user to find an available services according to its needs. Service composition helps to custom the service process. Service access helps users to submit data to obtain the corresponding service through service interface. Service monitoring helps to monitor the status of the request execution.

In the literature, extensive discussions have been performed from several points of view. We will focus on the service management perspective. Hence, resource supply layer and user interface layer problems, such as resource virtualization, VM migration and service registration, are out of the scope. More precisely speaking, we concentrate on the SLA management of PaaS including formally expressing SLA, SLA feasibility study method during SLA negotiation and runtime SLA enforcement.

2.2 OpenCloudware

The work described in this thesis is in the context of OpenCloudware project [ocw]. OpenCloudware is a project which aims at developing software modules to build a multi-IaaS based PaaS. OpenCloudware provides a development platform for application. This platform includes tools to manage complex virtual appliances during the whole life cycle. OpenCloudware itself is composed by components. The services are provided online through a service portal as a SaaS. With the help of OpenCloudware, clients can build, generate and operate enterprise distributed applications. OpenCloudware is supported by the French FSN, from January 2012 to September 2015. It is a co-funded collaborative R&D project. 18 partners involve in the project.

2.3 Contributions

Cloud computing is a new computing model. Infrastructure, application and data are moved from local machines to internet and provided as services. Cloud users, such as application own-

ers, can greatly save budgets from the elasticity feature, which refers to the "pay as you go" and on-demand characteristics, of cloud service. The goal of this thesis is to manage the Quality of Service (QoS) for applications running in cloud environments. Cloud services provide application owners with great flexibility to assign "suitable" amount of resources according to the changing needs, for example caused by fluctuating request rate. "Suitable" or not needs to be clearly documented in Service Level Agreements (SLA) if this resource demanding task is hosted in a third party, such as a Platform as a Service (PaaS) provider. In this thesis, we propose and formally describe PSLA, which is a SLA description language for PaaS. PSLA is based on WS-Agreement, which is extendable and widely accepted as a SLA description language. Before signing the SLA contract, negotiations are unavoidable. During negotiations, the PaaS provider needs to evaluate if the SLA drafts are feasible or not. These evaluations are based on the analysis of the behavior of the application deployed in the cloud infrastructure, for instance throughput of served requests, response time, etc. Therefore, application dependent analysis, such as benchmark, is needed. Benchmarks are relatively costly and precise feasibility study usually implies large amount of benchmarks. In this thesis, we propose a benchmark based SLA feasibility study method to evaluate whether or not a SLA expressed in PSLA, including QoS targets, resource constraints, cost constraints and workload constraints can be achieved. This method makes tradeoff between the accuracy of a SLA feasibility study and benchmark costs. The intermediate of this benchmark based feasibility study process will be used as the workload-resource mapping model of our runtime control method. When application is running in a cloud infrastructure, the scalability feature of cloud infrastructures allows us to allocate and release resources according to changing needs. These resource provisioning activities are named runtime control. We propose the Runtime Control method based on Schedule, REactive and PROactive methods (RCSREPRO). Changing needs are mainly caused by the fluctuating workload for majority of the applications running in the cloud. The detailed workload information, for example the request arrival rates at scheduled points in time, is difficult to be known before running the application. Moreover, workload information listed in PSLA is too rough to give a fitted resource provisioning schedule before runtime. Therefore, runtime control decisions are needed to be performed in real time. Since resource provisioning actions usually require several minutes, RCSREPRO performs a proactive runtime control which means that it predicts future needs and assign resources in advance to have them ready when they are needed.Hence, prediction of the workload and workload-resource mapping are two problems involved in proactive runtime control. The workload-resource mapping model, which is initially derived from benchmarks in SLA feasibility study is continuously improved in a feedback way at runtime, increasing the accuracy of the control.

To sum up, we contribute with three aspects to the QoS management of application running in the cloud: creation of PSLA, a PaaS level SLA description language; proposal of a benchmark based SLA feasibility study method; proposal of a runtime control method, RCSREPRO, to ensure the SLA when the application is running. The work described in this thesis is motivated and funded by the FSN OpenCloudware project [ocw].

2.4 Organisation of the document

This document is structured as follows. SLA management related issues will be introduced in section 3. Researches around SLA management will be discussed in section 3.3. Standardized

SLA descriptions will be introduced in section 3.4.

The Virtualized Componentized Application Architecture will be described in section 4.2. Runtime control methods will be introduced in section 4.3. Existing proactive and reactive runtime control methods will be introduced in section 4.4. Our runtime control method is based on benchmarking. Benchmarking will be introduced in section 4.5.

The requirements of a PaaS level SLA will be discussed in section 5.2. The syntactic expression of PSLA will be given in section 5.3. This chapter will be concluded in section 5.4.

The context of SLA feasibility study will be introduced in section 6.2. The benchmarking based activities described in section 6.3.1, 6.3.2, 6.3.3, 6.3.4 and 6.3.5, which devote to explore Maximum Flavor Capability modeling will be introduced in section 6.3. Based on the Maximum Flavor Capability model explored in section 6.3, the constraints described in section 6.2.1 can be evaluated and the evaluation results will lead to reject or accept of the SLA contract. These constraints evaluation processes will be introduced in section 6.4. To make our benchmark based SLA feasibility study method understandable, we will give an example in section 6.5. This chapter will be concluded in section 6.6.

The challenges, time granularities and runtime control system architecture based on MAPE-K will be discussed in section 7.2. RCSREPRO, our runtime control method, will be described in section 7.3. This chapter will be concluded in section 7.4.

Chapter 3

State of the art- SLA management

3.1 Introduction

The goal of this thesis is to manage the Quality of Service (QoS) for applications running in cloud environment cloud services provide application owners with great flexibility to assign resources according to the changing needs, for example caused by fluctuating request rate. QoS requirements need to be clearly documented in Service Level Agreements (SLA) if this resource demanding task is hosted in a third party, such as a Platform as a Service (PaaS) provider. In this chapter, SLA management related issues will be introduced in chapter 3. Researches around SLA management will be discussed in section 3.3. Standardized SLA descriptions will be introduced in section 3.4.

3.2 SLA management

As the developed research of utility computing, grid computing and parallel computing, cloud computing has been the main form of network services because its virtualized shared resource, on-demand and pay-per-use features attract a large number of users. In the context of cloud computing, applications and data move from the local machines to the cloud. Users can run their business appropriately through web interface without buying a lot of infrastructure resources or maintain the software stack. Accordingly, cloud service provider can obtain economic benefits from their customer. Cloud services have been widely adopted in the fields of bank, e-commerce and academic because of the flexibility, scalability and cost-effectiveness. Cloud services are easy to use and clear paid, but due to the complexity, such as uncertainty of physical location of cloud resources or performance variation, cloud service quality can be different. Therefore, the guarantee of Quality of Service (QoS) in the cloud becomes one of the core concern when adopting cloud services. Cloud services such as Amazon EC2 usually promise the available rate and corresponding service credit percentage during the service [ec213]. Service Level Agreement (SLA) is one of the agreements between service consumer and service provider. It is used to structurally describe QoS related rights and obligations between contracted parties. Cloud computing service consumers and service providers will negotiate the SLA terms before signing the SLA and start the service. During the service, cloud service providers will provide the service under SLA. Usually, a SLA draft with service quality restrictions such as the deadline (deadline), budget (budget) and punishment (penalty rate) will be proposed to service provider.

If the SLA draft is received, a formal SLA contract is achieved between the cloud service consumer and the cloud service provider. Both cloud service consumer and cloud service provider should fulfil the terms listed in SLA. Or else, penalty will be caused because of SLA violation besides the adverse impact on credibility. For service providers, the best way to obtain the maximum economic benefits is accepting the maximum services requirements while improving customer service and satisfaction by meeting the SLA. In this way, cloud service providers can obtain high service reputation [Ver04]. Therefore, how to determine the maximum achievable SLA and how to formally and effectively describe a SLA are two important issues for effective QoS management.

In this section, SLA management related issues will be introduced. The definition of SLA is given in section 3.2.1. SLA lifecycle will be explained in section 3.2.2. SLA negotiation process will be introduced in section 3.2.3.

3.2.1 SLA definition

SLA is a negotiated agreement between service consumers and service providers to define QoS specific issues. SLA is an important way for guaranteeing the QoS of network services. In a SLA for network services, network resources are abstracted into measurable parameters, such as network latency, bandwidth, and response time. The promise about QoS between the network service consumer and the network service provider are ensured by guaranteeing these parameters within the negotiated range during the whole network service process. The QoS is managed by establishing an interactive protocol between service providers and service consumers. The protocol can also be used to the effective maintenance of all types of services and management of resources, thus desirable QoS is guaranteed [LWK+10], [MNM+07].

SLA has already been widely adopted in many kinds of web services before the spring up of cloud computing. For example, WSLA [LKD+03] proposed by IBM and WS-Agreement [ACD+07] proposed by OGF (Open Grid Forum). According to [SMJ00], an instant SLA should include the following 6 aspects:

- Parties is the role of consumers and providers in the cloud service market.
- Parameter is the metrics used to measure the cloud service. The parameters can be decided by negotiating between service provider and the service consumer.
- Service Level Objective (SLO) defines the quality of service promised by service providers. Usually, each service should include at least one SLO. In each SLO, there are some parameters. Moreover, each SLO should have a period of validity and a set of clauses to describe and measure the details of work flow.
- Monitoring monitors and evaluates the QoS parameters defined in SLA. The service provider and the service consumer should be notified when a SLA violation happens.
- Violation process means that when service provider can't satisfy the service consumer with what they promised in the SLA, SLA pre-defined actions will be taken.
- Life Cycle explains the changing state of SLA with the evolution of service.

The measurement of service level and the monitoring of service quality should be based on corresponding service parameters. The parameters considered when signing SLA between service

provider and service consumer should be customized according to different services. Cloud service parameters can be designed from 4 aspects: service aspect, technical aspect, business aspect and QoS aspect. Service aspect describe the detailed information about services provided to service consumer including the negotiated service content and the service level. For example, number of instances needed for running a job, the amount of jobs submitted by users, the load of one single instance, the duration of running each instance, the time constraints of running each instance, the budget of each instance, the budget of running each job, the type of instance and the price of each kind of instance. Technical aspect describes the measuring unit of QoS parameters and some technical support facilities. For example, the CPU power and Memory size of VM, booting up time of VM, the scale of long term and short term of data storage, the maximum and minimum number of VMs to serve one single user, the duration needed to increase or decrease n VMs, the runtime scaling ability of the data center, the maximum number of VMs that can be configured on one single physical server and the geographic location for data storage. Business aspect describes some business information and rules related to provided services including SLA violation handling and charging scheme: for example, penalty and billing. QoS parameters can be Availability, Response time, Security, Reliability, Usability, Privacy, Customizability, Scalability, System throughout, Networking, Recovery, Latency. The determination of QoS parameters is made based on service aspect, technical aspect and business aspect. The QoS monitoring informations are important for evaluating and improving the quality of services guaranteed through SLA contract.

3.2.2 SLA lifecycle

In fact, organizations have different definitions of SLA life cycle . For [RWQ+08], the SLA life cycle is divided into six stages as discovering provider, defining SLA, agreeing on SLO terms, monitoring SLA, terminating and enforcing penalty. In the stage of discovery provider, service consumers search for service providers which meet certain properties and select a provider as final service provider. In Phase of defining SLA, SLA is defined according to certain criteria. For example, WS-Agreement [ACD+07] is a SLA standard developed by OGF which specifies how to define the SLA, including the specifications like what terms should be in SLA and in what format terms should be expressed in SLA. In WS-Agreement, SLA can be classified as template and agreement in the form of XML documents. In the stage of agreeing on SLO terms, all the service level objectives (SLO) and corresponding penalty terms will be considered and finally fixed after several rounds of negotiation. In monitoring SLA stage, a trusted third party monitor is needed in the service delivery process to record the provided services as a basis for judging contravention. When the service is completed, SLA expires or service provider is no longer able to provide desirable services, SLA goes into the SLA terminated state. If SLA violation occurred, SLA enters the state of enforcing penalties and punishments will be executed. WSLA [LKD+03] is a SLA standard developed by IBM. IBM defines SLA lifecycle as five SLA lifecycle states including SLA identified, SLA requested, SLA inactive, SLA active and SLA terminated [IBM]. SLA enters SLA identified state when a customer requires the definition of SLA. During SLA requested state, the specific content of the SLA will be determined. SLA requester needs to agree on, reject or propose modifications to the SLA. SLA negotiations will continue during SLA inactive status until all the negotiations completed which means SLA enters SLA active state. SLA terminated state indicates that the service is completed or terminated according to the violation related terms defined in SLA.

3.2.3 SLA negotiation

There are many negotiation methods. Negotiation methods can be classified into three categories: game theory based methods, rule-based approaches and heuristic-based approaches [GLDK03], [LLD+03], [FSJ98], [Sim02]. [GLDK03] proposes an auto-negotiation and decision-making scheme based on all parties' benefits and rules. For grid computing, [LLD+03] uses the time-dependent model and resource-dependent model in negotiation decision function [FSJ98] to achieve an acceptable SLA collection for both grid resource buyer and sellers. In this acceptable SLA collection, an optimized SLA for both parties is chosen as the final SLA. However, the impact of market trends is not taken into consideration during negotiation.

[Sim02] proposes a market-oriented negotiation strategy, using market information to calculate the suitable propositions for both service consumers and service providers. Market informations and negotiation conditions are used during the negotiation process to adjust the SLA terms flexibly. Game theory has been applied by researchers to solve SLA negotiation and resource management problems involved in cloud. [WVZX10] applies game theory to the resource supply and demand modelling problems in cloud computing. [SCW+10] proposes an algorithm to maximize the economic benefits of both cloud service providers and cloud service consumers to overcome the shortcomings of the existing mechanisms. [ALIZ10] proposes a distributed negotiation mechanism for cloud computing environment. [Sim10] points out that in the context of multi-cloud, service markets among cloud services providers, cloud service consumers and cloud service intermediary require a more complex negotiation mechanism to achieve dynamic SLA negotiation. In WSLA, SLA negotiation is achieved by WSLA document interaction. In most cases, the internet service provider's WSLA document defines most of the content, the user simply agrees to provide some relevant information. Usually, the service provider defines most of the content in a WSLA document, and the service consumer simply provides some relevant information and agrees on the SLA document. More flexibly, the service provider defines SLA parameters and the service consumer defines SLO and SLA violation handling terms. Service consumers and service providers can also work together to define the SLA parameters, SLO and violation handling terms. In any case, negotiations are starting from a WSLA template, and finally generate a WSLA document. WSLA negotiation process can be a face to face communication between both parties, or else, it can be done online. Negotiation protocol can be any general electronic document negotiation protocol [Sta98].

In WS-Agreement, the agreement negotiation is started from an agreement template [ADK+04]. The agreement template is sent to the agreement consumer by the agreement initiator. Comparing to the real SLA agreement, the structure of the agreement template is associated with agreement creation constraints such as acceptable value range of variables in SLO. Then, the service consumer creates the initial version of the SLA according to the WS-Agreement template, and sends it back to the service provider. The service provider does feasibility study on the offer and makes the decision of acceptance or rejection. Feasibility study is based on the amount and type of resources which can be used, the cost constraints from service consumer, the QoS requirements, the target workload and so on. The feasibility study method, as an important action of SLA negotiation process, is one of the main goal of this thesis.

3.3 Review of SLA management studies in cloud computing

Various entities such as service providers, service consumer and end-users are involved in cloud services. Different entities have different objectives. SLA sets performance goals, privacy and protection constraints. SLA also defines actions that need to be taken to guarantee the QoS attributes. SLA includes obligations such as penalties and necessary informations for service provider. With the aforementioned information defined in SLA, dynamic and efficient resource provisioning can be performed for ensuring QoS of changing application demands.

A lot of researches take SLA specifications and term languages into consideration. 4CaaSt project [C+] creates a PaaS Cloud platform [GGJGT+12]. This PaaS platform introduces a description language to define dependencies among services, elasticity and multi-tenancy management rules and necessary informations about application to achieve the mapping from the application level to the resource level which is essential for elasticity management and SLA enforcement at runtime. With the business model simulation tool related to eMarketplace, complex pricing and business models for any type of cloud service (even composite cloud services offered by different providers) can be established. With business resolution feature [MGV11], experience of end users and customers can be used to propose better business offers which better meet the demands of service consumers.

CloudScale project [Pro12] aims at supporting scalable service engineering. CloudScale supports analysing, predicting and resolving scalability problems service providers' perspective [BSL+13]. CloudScale provides a language named as ScaleDL (Scalability Description Language) to describe the scalability requirements of a service. ScaleDL can be used to specify informations like application usage and cost, application deployment to support scalability analyses of services. CloudScale can discover resource related causes of potential SLA violations based on the source code of the service. Scalability problem solutions are given and what-if analysis are done to find out the best solutions such as changing the implementation of the source code.

SLA@SOI [pro11] aims at developing an open-source framework for managing the entire service stack, including negotiating, provisioning, monitoring and adaptation of SLAs. It provides SLA(T), as described in [KTK10], a SLA model to describe functional and non-functional requirements of a service. SLA(T) is based on vocabularies like QoS metrics or constraints and described as abstract syntax. To instantiate a SLA based on SLA(T), concrete syntactic format such as XML, OWL, or human-readable formats will be needed. SLA@SOI implements a SLA negotiations framework to negotiate across business, software, and infrastructure tiers. The framework contains a negotiation protocol to make interaction behaviours maintainable, readable and machine readable. It also contains a protocol engine and a negotiation protocol to execute the negotiation protocol for the customer and the provider interacting stateful. SLA@SOI provides a three-layered dynamically configurable monitoring framework to manage monitored metrics defined in SLAs, to map high level SLAs to monitorable low level metrics of monitoring system and also to manage intrusive and non-intrusive reasoners which can analyse SLA terms and perform monitoring actions.

CONTRAIL [CCD+12] aims at providing an elastic PaaS over a federation of IaaSs. With CONTRAIL, small IaaS providers can join a Cloud Federation of bigger players to develop there business QoS, SLA management, security, interoperability and scalability issues can be handled uniformly. CONTRAIL extended the SLA model proposed by the project SLA@SOI[pro11] to express virtual resources in standard OVF form, to express guarantees

terms and association with related OVF. Guaranteed Actions in SLA specify the automatic scaling which requires more resources when cross the predefined thresholds. CONTRAIL SLA specification can describe both generic parameters for any resource and specific parameters for specific resources. CONTRAIL SLA specification includes Quality of Protection terms to express data locality, protection, replication, etc., which can be linked with different pricing models. User negotiates a SLA with CONTRAIL, and CONTRAIL looks for the best solution by negotiating with IaaS providers on behalf of the users and selects the best SLA offer according to user criteria. CONTRAIL also coordinates the runtime interaction within the cloud federation to minimize SLA violations.

ETICS [LSCD+10] aims at supporting the provision of composite services based on atomic services from different providers. The SLA template includes entities identification, service description, business aspects and technical aspects like QoS parameters. A hierarchical SLA is designed to finally define SLA for end-to-end, QoS-enabled and network-guaranteed application service. Interconnections among applications, network providers and end-users are described by static SLAs at the atomic service layer [LSCD+10]. ETICS identifies and realizes different SLA composition paradigms. SLA composition can either be centralized as a unique entity or distributed. ETICS also supports different business or charging models.

OPTIMIS [FHT+12] aims at supporting and optimizing the whole life cycle of service provision, operation, delivery and use of organizations who want to use cloud services to extend their services and applications automatically. OPTIMIS provides a service manifest [FHT+12] which can be used to describe the requirements for running applications in IaaS. Both functional and non-functional requirements can be specified for each component of the application. The manifest can include the VM images with data constraints and protection, elasticity description, legal terms and other standard contractual clauses.

PrestoPRIME [Ter10] helps management of long-term preservation of audio-visual digital media objects, program and collections. The preservation is done by a "service provider" which can be either the same as producer and consumer or not. The interactions between preservation service and producers and consumers are defined in SLAs. The SLA framework includes metrics such as availability of services, storage occupation, sets QoS terms such as threshold on the SIP ingestion time, constraints such as maximum number of simultaneous users, clarifies pricing terms and also penalty terms. Automatic monitoring system are used by the service provider to support the capacity management system to maintain the QoS defined in SLA.

Q-ImPrESS [C+10] aims at helping the design decisions and forecasting performance, reliability and maintainability through quality impact analysis and simulation. Service Architecture Meta Model (SAMM) [BBB+08] is proposed to define service attributes, parameters like data volume, configuration, execution environment and the dependencies between parameters. Q-ImPrESS develops a trade-off analysis framework to estimate performance metrics which take propagation effects across systems into consideration. In this way, potential SLA violations can be forecasted and optimized design can be performed in an early stage [KSB+11].

SERSCIS [SCHM+12] aims at providing an infrastructure for dynamically managing changing situations, risks of an interconnected IT system. SERSCIS architecture consists of Application layer, Management layer, and Decision Support layer. The Application Layer refers to enterprise applications, service-oriented workflows, computation, storage, networks, and sensors. The Management Layer provides autonomic and predictive management of SLA terms. With QoS metrics and Key Performance Indicators, the Decision Support Layer provides operational decision support tools and analytic.

3.4 SLA standards

Nowadays, with the development of SOA and cloud market, more and more services are constructed or based on other services. These services may be offered by different providers. Interactions and relationships among services need to be clearly, completely and formally described as input of SLA management of composed services.

In this section, standardized SLA descriptions will be introduced. Existing SLA description languages will be introduced in section 3.4.1. We develop our PaaS level SLA description language based on WS-Agreement which will be introduced in section 3.4.2.

3.4.1 SLA specification standards literature review

There are many standardization proposals related to QoS requirements description, for example Web Service Offering Language [TPP02] (WSOL), Web Service Modelling Ontology [RKL+05](WSMO), Web Service Management layer [CV03](WSML) and WS-Policy [BCH+03] proposed by the World Wide Web Consortium (W3C). But none of them can be used as a SLA description language.

SLA description language design is a core concern of many SLA related works. A lot of researches has been done to commit to SLA standardizing. Web Service Agreement [ACD+07] (WS-Agreement) from Open Grid forum (OGF) and Web Service Level Agreement language and framework (WSLA) [LKD+03] from IBM are two widely adopted SLA description language for web services.

WSLA can not be extended and no further implementation and maintenance exist. Therefore, WSLA is not a good choice for expressing SLA specific to PaaS. The works in WSLA is further developed in WS-Agreement.

WS-Agreement [ACD+07] is based on existing standards such as web service description language (WSDL) and the web service resource framework (WSRF) which also comes from OGF. SLAs are finally expressed as XML documents which have been validated according to a XML schema document based on WS-Agreement specification. WS-Agreement can be extended. The extended SLA should also be validated according to a XML-Schema which is extended according to the WS-Agreement specification.

SLAng [LSE03] uses OMG's MetaObject Facility (MOF) in its SLA description language to achieve a certain level of language independence. SLAng is difficult to use in the context of cloud computing with domain-specific QoS constraints, since SLAng is designed for electronic services.

CC-Pi [BM07] proposes a domain independent SLA description language without basic SLA elements such as contract parties. Thus, it is difficult to use CC-Pi without its attached negotiation process. Rule-based Service Level Agreement (RBSLA) [Pas05] is based on rules and adopts RuleML [BTW01] to express SLA.

SLA@SOI provides a SLA model [KF11] which is representation language independent by offering an abstract syntax description instead of a specific language such as XML, which means abstract SLA description language proposed in SLA@SOI needs to be implemented in a specific language schema, then extended to our cloud context.

CSLA [KL+12] is specially designed for cloud computing. CSLA consults well known SLA languages like [LKD+03] [ACD+07] [LSE03]. However, at this moment, CSLA still

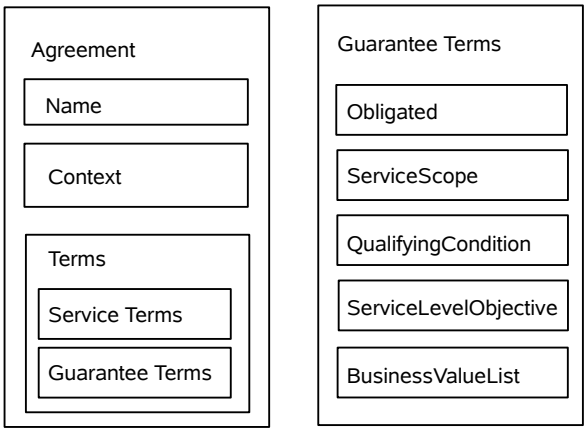

Figure 3.1. WS-Agreement Architecture

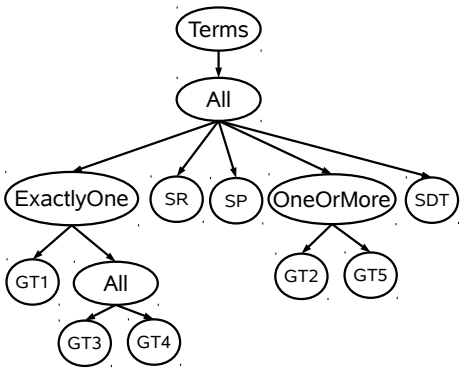

Figure 3.2. An example of term tree in WS-Agreement

needs a lot of completion of basic clauses before using. “Confidence” and “Fuzziness” are first introduced in CSLA to express the ambiguous of the boundary value of target range of QoS metrics.

3.4.2 WS-Agreement

As mentioned above, WS-Agreement Specification [ACD+07] is a better choice for formally specifying QoS requirements as a SLA.

A SLA based on WS-Agreement is a XML document which describes mainly the non-functional properties of a web service, but functional description can also be found to refer to the services guaranteed by this SLA. The structure of WS-Agreement based SLA document has been clearly defined in [ACD+07]. In this section, a shorter summary and introduction of WS-Agreement will be given.

As demonstrated in figure 3.1, besides a human-understandable Name of the SLA, a WS-Agreement based SLA should contain agreement Context and the Agreement Terms.

As depicted in Figure 3.5, Agreement Context describes SLA metadata including AgreementInitiator, AgreementResponder, ServiceProvider, ExpirationTime, TemplateID and TemplateName. Additional metadata can also be specified in Agreement Contexts to express domain specific needs. AgreementInitiator means the agreement creation proponent. Agreemen-

Figure 3.3. Business Value List structure.

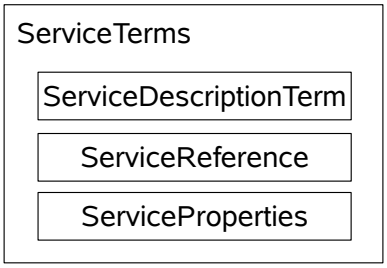

Figure 3.4. Service Term structure.

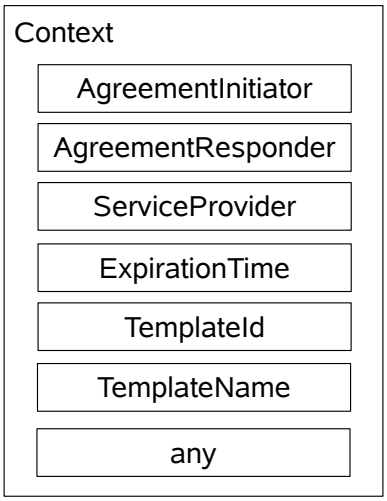

Figure 3.5. Context structure.

tResponder means the agreement proposal respondent. ServiceProvider identifies the service which is provided by AgreementInitiator or AgreementResponder. ExpirationTime express the SLA lifetime, which means during which period of time, the agreement is valid, by specifying the time point when this SLA is no longer valid. SLA template is the mould for creating a SLA offer accordingly. TemplateID and TemplateName specify which template the SLA contract or offer is based on.

Agreement Terms part is the most important part of the SLA. In Agreement Terms, both functional and non-functional requirements are specified. Correspondingly, there are two kinds of Agreement Terms: Service Description Terms and Guarantee Terms.

Logical operators, such as AND, OR and XOR which correspond to All, ExactlyOne and OneOrMore , can be used to logically group terms to recursively construct all terms together as a TermTree. Figure 3.2 is an example of TermTree.

Service Terms, as depicted in Figure 3.4, describe services which guarantee terms are applied to. There can be several services guaranteed under one SLA contract. Service Terms can be further expressed as service description terms, service reference terms and service property terms to specify different aspects of a service.

Service Description Term provides functional description of a service that will be delivered under an agreement, such as the information required from service consumer part. The service description term may contain three parts, including a unique name of the Service Description Term, ServiceName attribute to identify a service delivered under the agreement and a domain-specific functionality description of the offer.

ServiceReference contains a domain-specific reference which points to an existing service, including the name of the service reference and the ServiceName which identifies related functional services. This reference should be understandable for both parties.

ServiceProperties specify domain-specific aspects of a service that can be used to express guarantee terms. This property can be used to query the properties of the delivered services. To be more specific, properties means measurable metrics related to a service and will be used in expressing service level objectives (SLOs), such as response time. Besides the name of the service properties and the ServiceName which identifies related functional services, detailed variable definitions will be expressed in the form of Variable and wrapped up in Variable Set. To clearly define guarantee terms, metrics such as availability and response time must be clearly defined. Variable in ServiceProperties are the place to declare these metrics. Variable should be defined from three aspects, including location, Name and Metric. Location can refer to any field defined in service terms by using XML based domain-specific service description language such as XPATH to clarify this variable can be used with in which services. Metric can also be used when Location of the variable is not sufficient. A name-space, a local name and the type of the metric must be defined for the metric. And a unique string Name should be defined for each variable for using the variable easier later.

Guarantee terms [DKLR04] describe the quantified detailed QoS requirements between involved parties specified in Agreement Context. These guaranteed services are already described in Service Description Terms. Guarantee terms will be further used by SLA management systems to monitor and enforce the reached SLA. As demonstrated in figure 3.1, in each Guarantee Term, there are five parts involved, including obligated, Service Scope, Qualifying Condition, Service Level Objective and Business Value List.

Obligated specify which party is responsible for the current Guarantee Term. It can also be the case that service consumer should take responsibility so that service provider can provide

their services.

Service Scope specifies one of the services delivered under this Guarantee Term. There can be several Service Scope inside one guarantee term which means several services have overlapped service level objectives so that they can be written into and guaranteed by the same Guarantee Term. One Service Scope should contains a ServiceName and a XML reference which refers to a part of service.

Qualifying Conditions specifies conditions which should be met for current guarantee term coming into force. Time and constraints on the service consumer are two commonly used examples of Qualifying Conditions. For example, response times of querying services are lower than 3 seconds can be assured only if the arrival rate of the queries is lower than 1000 requests per second from 9:00 to 17:00 of each day. Qualifying Condition can be composed of several constraints. Meeting the Qualifying Conditions means that each constraint needs to be met.

Service Level Objective expresses the quantified QoS requirements, such as average response time, availability, in the form of target for a Key Performance Indicator (KPI) or CustomServiceLevel which can be used to express domain specific expression beyond WS-Agreement specification. Service Level Objective can be composed of several QoS requirements over different variables. Satisfying one Service Level Objective means meeting all of these requirements. And all of these requirements should share the same obligated, Service Scope, Qualifying Condition and Business Value List. For example, requirements with different level of importance should not be combined in one Service Level Objective.

Business Value List, as depicted in Figure 3.3, includes business values related to current Service Level Objectives from different business perspective. For example, the importance of this Service Level Objective to the non obligated part of this Guarantee Term. Or what should be done if the Service Level Objective is not satisfied. For example, Penalty is used to specify compensation when the Service Level Objective is not satisfied. Usually, penalty will be set higher by the non obligated party if it is more important. In this way, it is more important for the obligated party to meet this Service Level Objective.

3.5 Conclusion

In this chapter, we introduced SLA standards and SLA management. SLA, as a contract which specifies the service and QoS criterion, is indispensability for service oriented business. PaaS providers, which take charge of application's QoS, need to consider the on-demand resources and dynamic resource requirements and describe these characters in their PaaS level SLA. One PaaS level SLA contract should be customized for each contracted application since QoS is an application dependent topic. However, existing SLA description languages are not suitable for PaaS level SLA. Therefore, in chapter 5 of this thesis, we will propose PSLA which is a generic PaaS level SLA based on WS-Agreement skeleton. PSLA will be implemented in XML to make PSLA based SLA contract machine readable for automated SLA management.

Chapter 4

State of the art-Runtime control

4.1 Introduction

During SLA negotiation, the PaaS provider needs to evaluate if the SLA drafts are feasible or not. These evaluations are based on application dependent analysis, such as benchmark. These benchmarks provide workload-resource mapping model of our runtime control method. When application is running in a cloud infrastructure, the scalability feature of cloud infrastructures allows us to allocate and release resources according to changing needs. These resource provisioning activities are named runtime control. Changing needs are mainly caused by the fluctuating workload for majority of the applications running in the cloud. Runtime control decisions are needed to be performed in real time. In this chapter, runtime control will be discussed. This chapter is arranged as follows. The Virtualized Componentized Application Architecture will be described in section 4.2. Runtime control methods will be introduced in section 4.3. Existing proactive and reactive runtime control methods will be introduced in section 4.4. Our runtime control method is based on benchmarking. Benchmarking will be introduced in section 4.5.

4.2 Virtualized Componentized Application Architecture schema

Computing architectures which reduce the connection between hardware and application from both design level and runtime level, for example, container based and Google App Engine applications, can benefit from cloud services. Among several architecture paradigms, this thesis only focuses on Virtualized Componentized Application Architecture.

In this section, the Virtualized Componentized Application Architecture will be described. Component based architecture will be explained in section 4.2.1. Virtualized Componentized Application Architecture will be introduced in section 4.2.2.

4.2.1 Component based architecture

[MBNR68] proposed the concept of component based architecture for reusing software to build complex software system quickly and easily. With the increasing efforts from industry leaders such as Microsoft COM Component Object Model [WK94] CCM CORBA Component Model [SK00] and Sun EJB Enterprise Java Beans [DeM02], component based architecture is widely accepted from 90s.

The thinking of developing software based on components is not entirely new. Traditionally, the composition of complex software systems always start from defining subsystem modules such as the lower-level modules, classes, procedures, etc.

Software development reuse mechanism has been adopted for many years. Although the Object-oriented model has already been widely used to solve the problems of software reusing from the developers' perspective, component based architecture allow roles like administrator, architect and integrator taking the software reuse problem into consideration. Component-based architecture has further increased the possibility of software reusing. Using component technology to develop large-scale software systems is faster and more cost effective.

Component oriented software engineering is described in [Szy02]. A component is a unit of composition whose interfaces and contextual dependencies are specified as explicit contracts [Szy02].

The Fractal Model is a language and technical independent component model [BCS04] developed by France Telecom R & D and INRIA. Fractal is one project of free software project consortium OW2 (formerly ObjectWeb) which aims at reducing the costs of software systems development, deployment and maintenance. Fractal can be adopted in all kinds of software system, such as operating systems, middle-ware platforms and graphical user interfaces. Fractal can be realized in languages such as Java, C, .NET and Python.

Conventionally, a Fractal component is a processing unit with a heavily encapsulated state. A Fractal component interacts with the outside through customer interfaces and server interfaces. An interface does not specify an abstract type definition as in object-oriented languages, but an access point (typed and identified by name) on a component.

In Fractal, the external interface and the subcomponent interface are connected through an internal interface. It makes it possible to define composite component. Fractal provides a recursive component model instead of a single hierarchy because a subcomponent can belong to several parent components. This sharing of components was specifically designed to represent the shared resource, such as operating systems.

The component content is isolated by a membrane, whose function is to control the interactions at the external interfaces and provide observation and component reconfiguration capabilities. This membrane is controlled by control interfaces. Fractal model specification mentions some control interfaces such as naming, establishment or abolition of a connection between a client interface and a server interface, access to the content of a composite, lifecycle management and access to component state. Thus, the Fractal model was clearly specified to enable observation and hot reconfiguration of a system, including the addition, removal or replacement of a component. Moreover, a Fractal component is not only an element of design but also an element at runtime.

Componentized Application Architecture (CAA) is extracted from the design level information, such as application architecture, owned by application developer. CAA contains necessary information for deploying an application into the cloud infrastructure. To be more precise, CAA contains the hardware and software requirements of running each component, such as CPU compute power, available memory, network bandwidth required for supporting the communication between two components, the software stack, including required operating system, which are used for constructing the runtime environments for the application. In Figure 4.1, component C1 and component C2 both require services from component C4, but through different interfaces provided by component C4. Component C3 requires services both from component C4 and component C7. Application owner can shield system implementation details by only

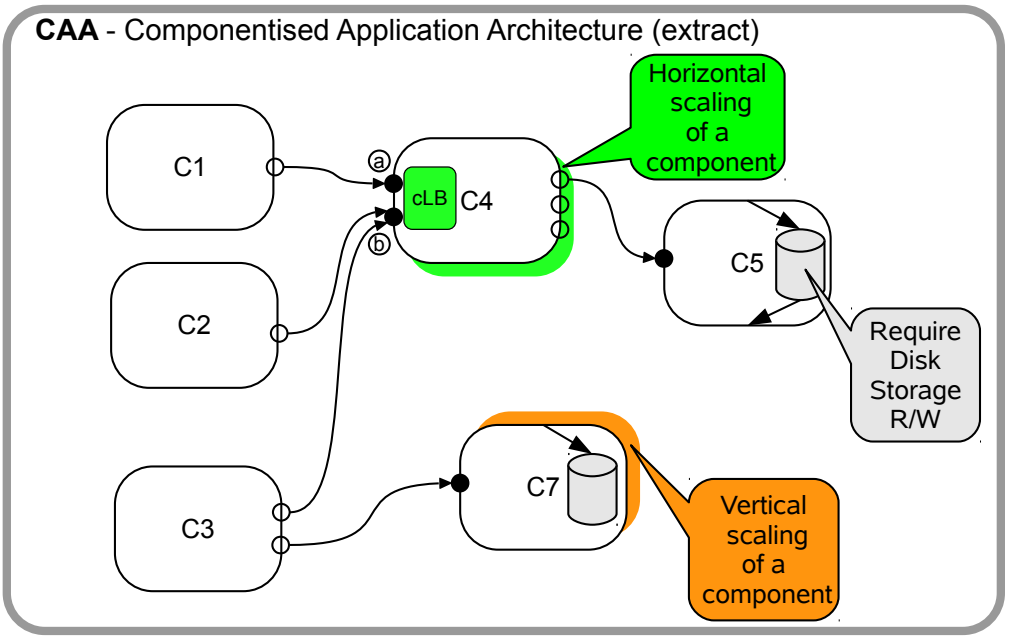

Figure 4.1. Componentized Application Architecture schema.

provide the layout's requirements of its components. According to Figure 4.1, component C1 and Component C2 should be installed in the same VM, component C3 and Component C7 should be installed in the same VM and component C4 should be installed in one VM alone. These deployment requirements can come from application owner's understanding of the system. Component C4 is only allowed to do horizontal scaling at runtime, while component C7 is only allowed to do vertical scaling. Horizontal scaling is allowed to be used for one component, the change of resource allocation can be performed by increasing or decreasing the number of duplications of the component. When vertical scaling can be used for one component, means that the change of resource allocated to the unique copy of this component can be done by increasing or decreasing its allocated resources. Since component C4 allows horizontal scaling, a load balancer is needed for routing workload among duplications of component C4.

4.2.2 Virtualized Componentized application architecture

Figure 4.1 is only an abstract description of the deployment requirements of an application. Several practical deployment schemas can be deducted from CAA described in figure 4.1. Figure 4.2 and Figure 4.3 depict typical practical deployments. These practical deployments in a virtualized environment such as cloud environment is named as Virtualized Componentized Application Architecture(VCAA).

In Figure 4.2, three different ways of deploying CAA described in 4.1 are depicted. Components C1, C2, C3, C4 and C7 are deployed together or separately accordingly. The differences among these three VCAA conditions are the ways of realizing vertical scaling for component C7 and horizontal scaling for component C4. Vertical scaling can be performed by two ways. One way is increasing or decreasing the allocated resource in component level, as depicted in condition 1 and condition 2 of Figure 4.2, which means the size of VM is kept and VM changes

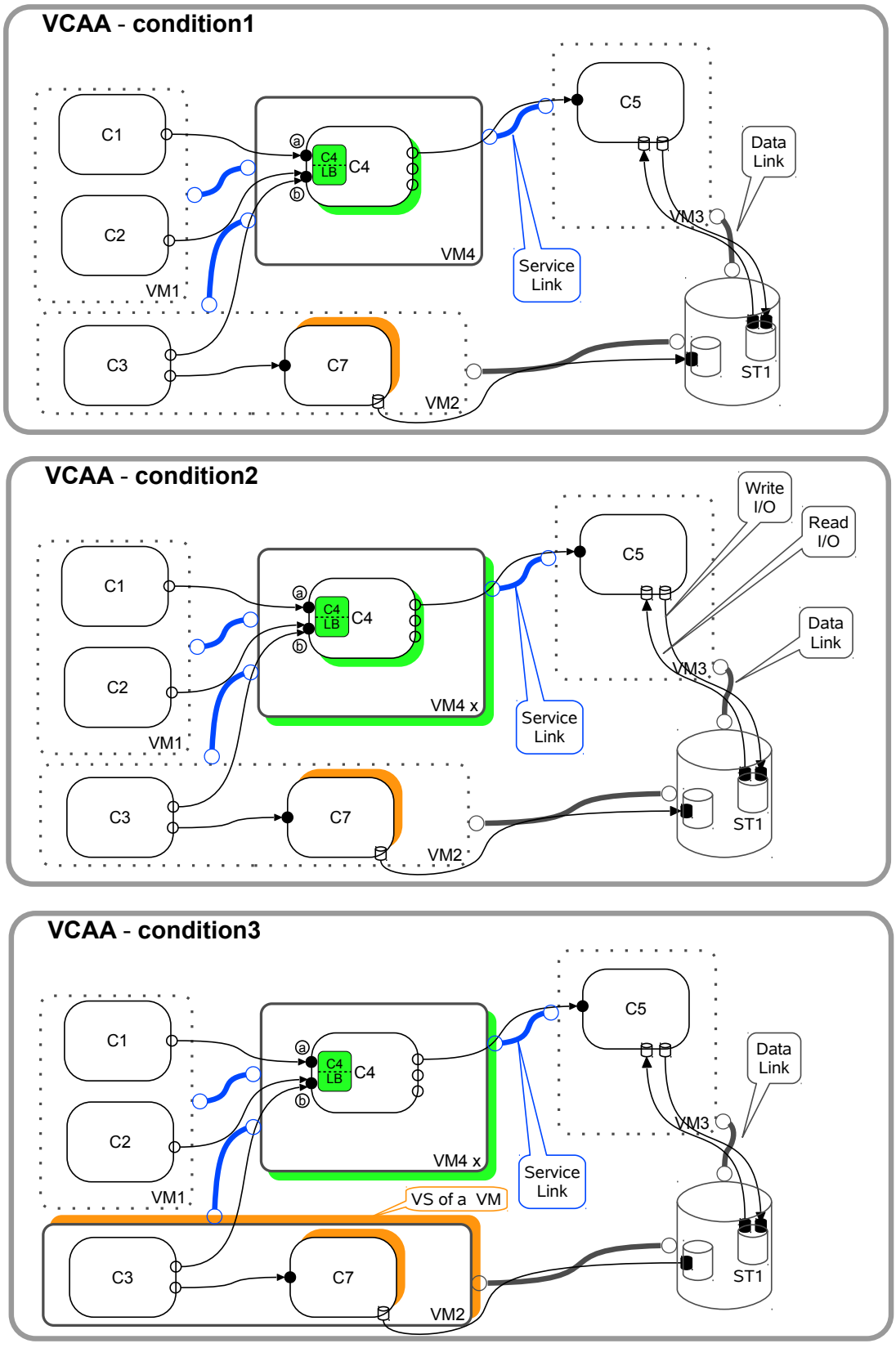

Figure 4.2. Three possible Virtualized Componentized Application Architectures.

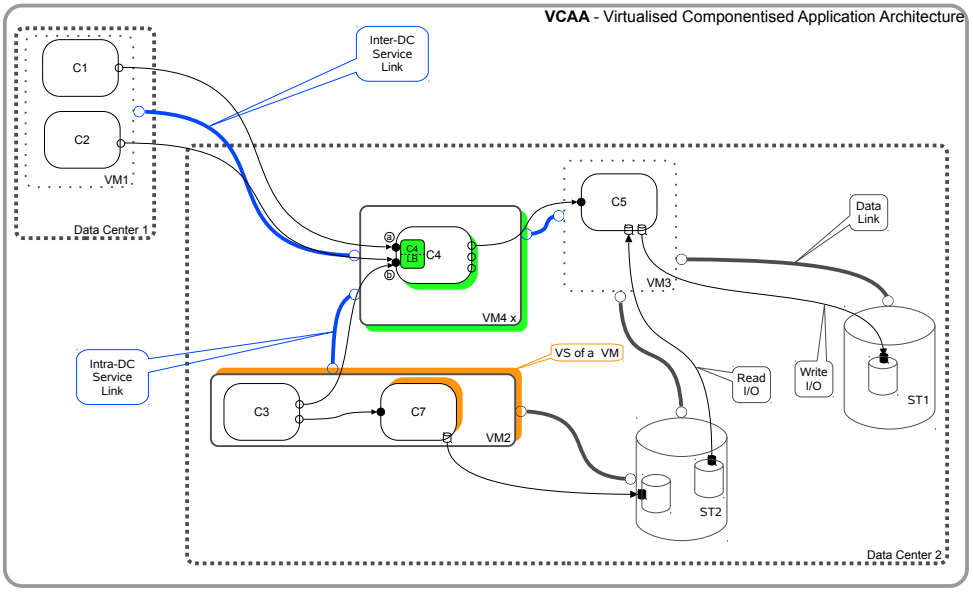

Figure 4.3. Multi-IaaS Virtualized Componentized Application Architecture example.

resource assigned to the component. Another way is that resource reallocation is performed in VM level, as depicted in condition 3 of Figure 4.2, which means the size of VM changes in order to increase or decrease resources allocated to the component. It can be the case when VM do not have enough resources to assign to the component.

Horizontal scaling can be performed in two ways. One way is that increasing or decreasing the number of component duplications in component level, as depicted in condition 1 in Figure 4.2, which means there are still one VM but the number of component duplications changes to have variational available resources for component C4. Another way is that horizontal scaling is performed in VM level, as depicted in condition 2 and condition 3 of Figure 4.2, which means VMs are copied together when duplicating components. In this way, workload is distributed by a VM level load balancer among identical VMs. It is worth mentioning that components of an application are not necessarily deployed in the same IaaS. As depicted in Figure 4.3, the VM carrying component C1 and C2 can locate alone in geographically different IaaS. This is a practical scenario, for example web server can locate closer to the end user to improve the speed of browsing articles of a e-commercial web site. If components are deployed in a geographically same IaaS, connection bandwidth and cost usually is out of question. However, bandwidth and cost can be a challenge if connections are through external network exactly the case described in Figure 4.1.

In this thesis, only VM level horizontal scaling and VM level vertical scaling are considered. We also make the hypothesis that all the VMs are of the same size when adopting horizontal scaling.

As mentioned before, adding or removing servers and changing server sizes are two ways of increasing and decreasing system capacity. Adding or removing servers refer as horizontal scaling. Another name of horizontal scaling is scaling out/ in. Changing server sizes refer as vertical scaling. Another name of vertical scaling is scaling up/ down.

Vertical scaling means changing the allocated amount of resource of a server. Vertical scaling can be achieved by reconfiguring the server or by replacing the server with a new server

of different resource allocation. Aforementioned resources can be any kinds of resources, such as CPUs and memory. Cloud environment provides the possibility of adopting vertical scaling based on the virtualization technology. As an infrastructure based on virtualization technology, servers' resource allocation can be reconfigured without power off, and it leads to significant reduction of vertical scaling time compared to traditional IT environment. Vertical scaling can be used to simplify the changing of resources shared among application modules. Vertical scaling is suitable for changing the size of the database.

Horizontal scaling means adding or removing servers to the system to handle the changing demand. When the number of user requests for an application increases, additional server resources can be launched to adapt to the increasing workload. When resources are no longer needed which means there are underutilized servers, resources can be released by terminating underutilized servers. Usually, in a runtime control problem, we only considering same size of servers for horizontal scaling. It is also the case for this thesis.

Before the emerging of cloud computing, vertical scaling is the first choice of serving an increasing workload. Horizontal scaling is performed on individual part of an application only if it reaches the vertical scaling limitation. In the context of cloud computing, horizontal scaling is the overwhelming majority of the scaling choice. "Vertical scaling" can not be supported because most of the operating systems can not change usable CPU or memory without restart the operating system [VRMB11].

4.3 Runtime control

Runtime control refers to the ability of increasing and decreasing system capacity. There are two levels of runtime control. One is allocating resources in the form of virtual machines to application and this allocation is driven by application's performance goals defined by the high level SLA [VTM09]. Another is placing virtual machines to the physical hosts and this placement is driven by the data center's management policies related to management costs [VTM09]. This thesis focuses on the former one.

To ensure that the allocated resources are able to meet QoS requirements defined in SLA while optimizing the use of resources. Runtime control is a necessary management process to prepare suitable IT resources for satisfying business needs.

In this section, runtime control methods will be introduced. MAPE-K based control system will be introduced in section 4.3.1. The classification of runtime control methods will be discussed in section 4.3.2. The schema of schedule based runtime control, proactive runtime control and reactive runtime control will be discussed in section 4.3.3.

4.3.1 MAPE-K

To provide a guaranteed cloud service, only clearly defined SLA is far from enough. Another goal of this thesis is to make scheduling decisions based on VMs at runtime by analysis the informations collected from monitoring system. MAPE-K [C+06] can be adopted to perform runtime control. As depicted in Figure 4.4, there are five parts involved in a MAPE-K based runtime control system, including Monitor, Analyze, Plan, Execute and Knowledge. MAPE-K is a control loop. Each part of MAPE-K is as follows.

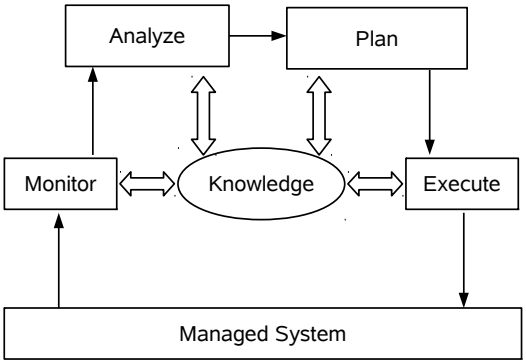

Figure 4.4. MAPE-K

- Monitor gathers informations from managed system. These informations are passed to Analyze.

- Analyze analyses and process the informations from Monitor based on the knowledges from Knowledge. Analyze will generate preliminary events and pass them to the Plan.

- Plan processes the preliminary events coming from Analyze and generates final execution decisions based on knowledges from Knowledge. The final execution decisions are passed to Execute.

- Execute takes the final execution decision generated by planning and implementing it.

- Knowledge stores the strategy and the historical data of handling all kinds of situations. It is the foundation for Analyze and Plan. Knowledge can either be inherent knowledge or acquisitive knowledge. Inherent knowledges are the strategies known in advance and configured into Knowledge before runtime. Acquisitive knowledges are acquired during running the system. Acquisitive knowledges are the results of adapting to the managed system.

4.3.2 Classification

Generally speaking, there are two ways of increasing and decreasing system capacity, which are adding or removing servers and changing server sizes. We can increase or decrease the allocated resource either according to a predefined schedule or according to the changing demands at runtime.

Schedule based runtime control, which changes allocated resource according to a predefined schedule, takes priori knowledges, statistical data and the periodical pattern of the workload into consideration to configure resource allocation manually before runtime. The ability of configuring resource allocation before runtime is interesting even in the context of cloud computing. It is because big IaaS providers, like AMAZON AWS, provide reserved VMs with a much lower price than on-demand VMs.

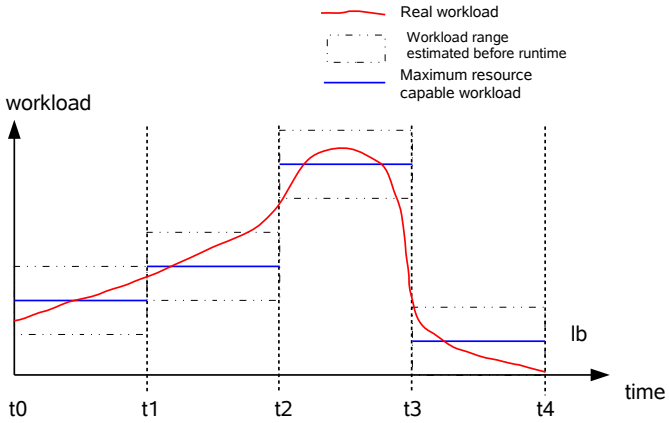

Figure 4.5. Schedule based runtime control.

The weak point of schedule based runtime control is obviously that the system cannot adapt to the unexpected changes of workload. Compared to the traditional IT environment and benefiting from the on-demand feature of cloud services, schedule based runtime control in cloud environment does not mean to prepare VMs for maximum workload, have idle resources during most of the time and undergo bad system performance when workload is more heavier than expected.

Changing allocated resource according to the changing demands at runtime is named as runtime scaling. Runtime scaling is based elasticity, the most attracting feature of cloud computing. Runtime scaling is also named as auto scaling in literatures [MLH10], [BBB+11]. With the ability of runtime scaling, cloud based systems can allocate suitable amount of resources for current workload, both reducing the amount of idle resources and be prepared for serving the peak workload.

Compared to schedule based runtime control, runtime scaling benefits a lot from the "pay as you go" feature of cloud services which means paying only for the resources you use. Researches in the runtime control domain is mainly around runtime scaling.

4.3.3 Runtime control schema

We adopt runtime control based on both schedule based runtime control and runtime scaling. The schema of schedule based runtime control will be introduced in section 4.3.3. The schema of proactive runtime control and reactive runtime control will be introduced in section 4.3.3.

Schedule based runtime control schema

When schedule based runtime control methods are adopted, a certain degree of workload knowledge is required to make the resource allocation schedule. More specifically, the more workload knowledge we have, the more accurate runtime resource allocation will be.

As depicted in Figure 4.5, the workload feature of requests arrival rate range is known before runtime. In the case when the real requests arrival rate measured during runtime fits the predicted workload feature. As a result, balanced amount of resources are allocated to serve the

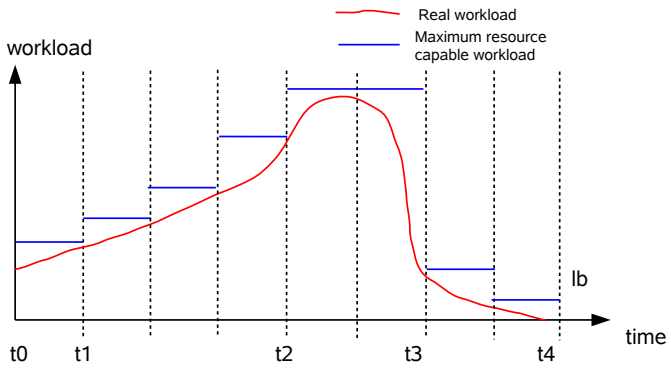

Figure 4.6. Reactive runtime scaling.

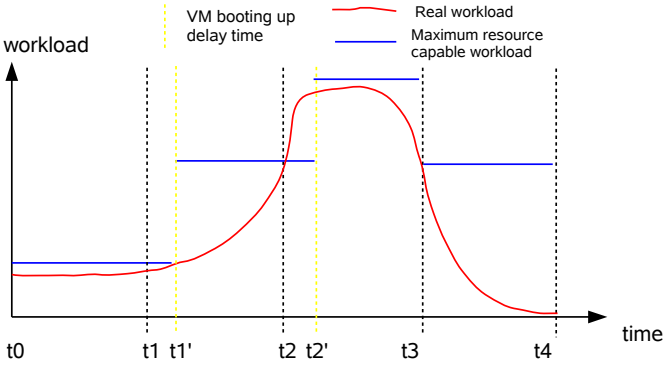

Figure 4.7. Weak point of reactive runtime scaling.

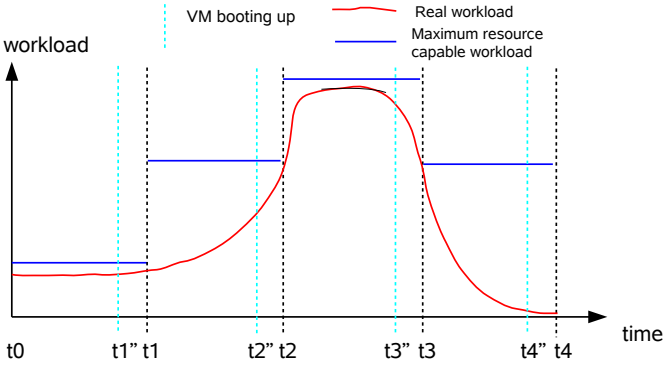

Figure 4.8. Proactive runtime scaling.

runtime workload with acceptable performance.

Since resource is allocated only according to the predefined schedule, the system performance will be challenged if workload does not behave as expected. For this reason, the ability of predicting workload behaviour precisely and meticulously before runtime is very important for schedule based runtime control methods. If workload can be well predicted before runtime, schedule based runtime control method can have acceptable performance.

Moreover, only understanding workload well is not enough for arranging suitable amount of resources before runtime. The mapping from workload to resource allocation is an important step for runtime control.

Runtime scaling schema

Runtime scaling, also named as auto-scaling method can be classified as reactive runtime scaling and proactive runtime scaling.

As depicted in Figure 4.6, reactive runtime scaling will not predict the future system performance or changing demands. Reactive runtime scaling methods only act when the current system is under a predefined situation. As depicted in Figure 4.7, to instantiate a new VM, it may take up to 15 minutes. If the system is suffering a sudden burst of traffic, this will be too long and the system can enter an even odious situation which means serious SLA violation. Therefore, pure reactive runtime scaling methods are not enough. Proactive runtime scaling methods are needed to prepare resources and have them ready when they are needed to cope with the fluctuating resource demands on time.

Proactive runtime scaling methods, as depicted in Figure 4.8, predict the future resource demands caused by fluctuating end user demands or system maintaining and reconfigure resource allocation in advance. In this way, resources increase or resources decrease can be performed ahead of the real needs.

4.4 Runtime scaling literature review

Existing runtime scaling methods are mainly based on model. In this section, proactive and reactive runtime control methods will be introduced. The theoretical models which have been used in runtime scaling methods will be briefly introduced in section 4.4.1. Existing runtime control methods will be introduced in section 4.4.2. We will focus on time series analysis based proactive runtime control methods. Time series based workload forecasting methods will be introduced in section 4.4.3.

4.4.1 Methods used in runtime scaling

In the literature, many techniques are used. For example, queuing theory [BFF90], control theory [Åst12], reinforcement learning [Bar98], time-series analysis [Ham94] and rule-based system [HR85]. Techniques will be firstly introduced in this section. Literature will be discussed in next part.

Rule-based method is the simplest but most widely used runtime scaling method. Famous cloud providers like Amazon AWS and RightScale provided rule-based runtime scaling methods. The typical rule-based runtime scaling method defines a couple of pre-defined rules on a set of target metrics for a group of VMs so that under rule-based conditions, certain actions

will be triggered. To be more precise, one rule for describing when and how to scaling up and another for scaling down. For example, when the average RAM usage of application server group is above 80%, the scaling action “add a new VM" for application server group should be performed.

Threshold based runtime scaling method is based on a set of rules. Each rule is expressed in the form of "if... then... ".

In " then ", scaling actions are defined. Theoretically, scaling action can be either horizontal scaling or vertical scaling. In practice, vertical scaling is not supported by most of the IaaS providers. A cooling down time is set for each scaling action to tell after executing current scaling action, after how long time corresponding preconditions should not be able to trigger this scaling action. This cooling down time is designed to avoid the situation that scaling actions are continuously triggered before previous scaling action contributes on the performance metrics. It’s because the boot up time of a VM takes several minutes.

In "if ", the preconditions to execute the corresponding scaling action are defined. Preconditions are usually a logical expression composed of one or more comparison expressions Each comparison expression defines the threshold of metric. For one metric, at least a couple of thresholds should be defined. One for upper bounder of the acceptable range of the metric value. Another one for lower bounder. For each precondition, a time value is set to tell how long the logic expression must be met to trigger the corresponding scaling actions. This time value is designed to avoid the situation that the threshold based runtime control system is too sensitive to the threshold.

The metrics used in the comparison expression can be different for different applications. For different applications and different server of applications, the key metrics for application performances are variational. Key metric is a resource utilization indicator which instructs that application system is under pressure of increasing allocated resource to guarantee the QoS or decreasing resource allocation to eliminate unnecessary cost. Potential metrics can be related to storage capacity, server processing power, RAM capacity, Network bandwidth , Database transactions per second and storage input/output operations per second [GoG10].

These metrics can be collected and computed periodically by monitoring system at runtime. For example, AWS Auto Scaling service makes the scaling up or scaling down decisions according to the performance metrics provided by CloudWatch which is a monitoring tool provided by AWS. CloudWatch collects performance data, such as CPU utilization, from AWS resources like instances every 5 minutes (performance data can also be collected every 1 minute with extra charges). These performance data can be used in defining the Auto-Scaling strategy. For example, if average CPU utilization is more than 70% , then start one more instance.

Reinforcement learning (RL) technical can help system to automate understanding, learning and making decision without any priori knowledge. RL can be used to solve runtime scaling problems by capturing performance informations about application and infrastructure from monitoring system. Typically, RL has an agent to interact with its environment and learn to establish its theory for decision making based on the rewards or punishments from environment. Agent should always try to maximize their rewards and minimize their punishment. In the context of runtime scaling, a scaling engine performs as the agent to make the decision of increasing or decreasing the allocated amount of resource. The aforementioned environment is an application deployed in the cloud infrastructure. The metrics such as metrics collected by monitoring system or the coming workload of the application environment are provided to the scaling engine as the state. It will be used as the input of scaling engine. The application envi-

ronment also provides the scaling engine a reward or punishment for scaling engine to optimize its decision to maximize the reward and minimize the punishment.

Traditionally, queuing theory is widely used to model application for analysing application performance, for example the inter-arrival time, the average number of requests, the average waiting time. Typically, when request arrives at the system, it joins the system and waits in the queue until it is processed by one of the servers. There can be several or only one servers in the queuing model which is not necessary the same as the real application system.

The aforementioned typical three tier application can be modelled by using queuing theory in two different way. One way is using one single queue for the load balancer and several VMs as servers to process requests. The other way is using several queues in one complicated queuing model. This complicated queuing model uses one queue to depict each tier of the application and one or more VMs are depicted together as one server in the model.

The inconstant coming requests and changing ability of servers are the big problems of adopting queuing theory for our problem. For example, the amount of allocated resources for one tier or the application can be changed dynamically at runtime. These will leads to different number of servers in the queuing model. Therefore, the queuing model needs to be calculated for all possibility of deployment for each workload condition in advance or periodically recalculated at runtime which makes queuing theory difficult to use in practice for runtime scaling problem.

Control theory has also been used to solve runtime scaling problems. The simplest control system is open loop control system. Open loop controller only computes the input of the target system based on the current state of the system by using the model of the system but do not have a look at the outputs of the target system and not to speak of correcting the model with the undesirable output as what feedback controller did. An even better developed control system is feed-forward controller. Based on feedback controller, it predicts the potential error of the output of target system and adjusts the output of controller, which is also the input of the target system, to avoid errors in advance. Fuzzy controller which means a control system based on a fuzzy model can also be used for runtime scaling problems. Fuzzy models can help to locate the required resource for workload according to predefined rules in the form of fuzzy sets. These rules have one precondition and corresponding consequence. The fuzzy model should be defined before runtime which makes fuzzy controller based runtime scaling method not well adapted to dynamic workloads.

Time-series represent a series of data recorded according to one specific time interval, for example the average response time of the whole application calculated at every one minute. Time-series analysis means the method which has been intensely used to represent and analyse the change of a measurement with time. For runtime scaling problem, time-series, such as average memory usage or average requests arrival rate periodically collected by monitoring system, can be analysed to predict the future data. This prediction will be further used by proactive runtime scaling method to make the scaling decision. The repeated patterns hidden in time-series can be dig based on the past observations, and then be used to predict the future workload or metrics.

4.4.2 Runtime scaling literature reviews

Rule-based runtime scaling methods are reactive methods because they won't anticipate and actions are only triggered when something already happened. Rule-based runtime scaling method

is very simple and easy to understand and use. So, the Rule-based methods are the most extensively used method in cloud industry. IaaS provider Amazon EC2 is the most representative one. Amazon EC2 allows its users to set rules to autocratically manage the allocated resources referred to an application to adapt to the changing demands and guarantee the SLA at the same time. However, selecting the suitable metrics and designing the logical combination are not easy. Moreover, in [DRM+10], the tuning of the thresholds is important for avoiding oscillations in the system. Cool-down periods which will have no scaling actions can also be set to avoid oscillations. All of these require both deep understanding of the application and abundant experience of these tasks. [HMC+12] proposed a method with four thresholds instead of two on each set of metrics. They are the upper threshold, the secondary upper threshold, the lower threshold and the secondary lower threshold. With these four thresholds, more sophisticated rules can be defined to perform more complicated runtime controls. In particular, most authors and cloud providers use only two thresholds per performance metric. As mentioned before, the choice of metrics to be used in rules is a technical task. Usually, one or two performance metrics will be used in the rule and average CPU load, average response time and request arrival rate are frequently the choice. [HMC+12] chose to set the metrics in a more complicated way. Different kinds of metrics and combinations of metrics of compute type, storage type and network type can be used together. For example, increasing the bandwidth of the network link resource when both the response time and the network link load are high. RightScale [rig07] provides an innovative rule-based runtime scaling method [Rig15]. It introduced democratic voting into the scaling decision making progress. The scaling up or down decision can only be made when most of the VMs want to scaling up or down. Cool-down period mechanism is also adopted and 15 minutes of cool-down time is suggested because it usually take from 5 minutes to 10 minutes to start a new VM. If cool-down time is too short, scaling up decisions can be continuously made while new VMs are still booting. Based on RightScale's rule-based voting system, [CMK11] extended their own previous work [CMKS09] which adopted active sessions. In work [CMK11], scaling one VM up decisions are made only when all VMs have session amount bigger than the upper threshold, scaling one VM down decisions are made when at least one VM have a session amount lower than threshold and all other VMs are lower than the threshold, and the VM without session will be released. [SGLI11] proposed a strategy tree to evaluate and chose different rules. These rules are set for different workloads so that rules can be switched at runtime to serve different workloads. This method is designed to solve the problem of rule-based runtime scaling that is both typical rule based runtime scaling and rule-based voting system [KSJB09] require extra effort of choosing performance metrics and tuning thresholds. [KSJB09] takes the charge unit of VMs into consideration and proposes " smart kill" which means only release a VM when the charged unit is finished even if the VM is no longer needed to guarantee the SLA. There are also considerations of using dynamic thresholds. [BB10] proposes to monitor all the CPU utilization of VMs on one host node and determine the frequently range of CPU utilization based on probability distribution to set the upper and lower thresholds at runtime. Time-series forecasting can be combined with reactive techniques. [IDCJ11] uses rule-based runtime scaling for scaling up decisions but make scaling down decisions based on regression approach.

PID controller is widely used in the literature. For example, in work [LBCP09] and [LBC10], integral controller is used to increase or decrease the number of VMs according to the average CPU usage. In [PH09], PI controller is used to control the allocation of resources to batch jobs. [PH09] proposed a model to evaluate the progress of resource provisioning

of a job. [AETE12] proposed a method to determine the dynamic gain parameters for scaling down decisions based on workload. [XZF+07] used fuzzy controller on business tier and works in [WXZ+11] performed on the database tier. Both of them can not adapt their method for highly dynamic workloads. Also [LZ10] adopted neural fuzzy controller to automatically construct fuzzy controller.

Reinforcement learning is seen as a good choice for runtime scaling problem, but there are still several weakness need to be solved. For example it takes very long time to learn and establish the decision model which means the affect of the runtime scaling can be very poor for an unacceptable long time at the beginning. Besides, when too many metrics are used to choose the suitable action to perform, the speed of decision making can be extremely low because there are too many states. These two problems are the main consideration for researches who adopted reinforcement learning for runtime scaling in their works [DRM+10], [DKM+11] [BHD13], [TJDB06], [RBX+09].

Queuing theory is used to model the performance of applications. [TJDB06] model one tier of the application as a queuing model of n servers which serve requests in a round-robin pattern. The average response time is modelled by using a parallel M/M/1 queuing model. The authors also propose to model the application as n independent parallel closed networks of one server and 1/n of the customers on each to deal with the condition of finite number of customer. [VPR07] models each application tier server as a M/GI/1/PS queue. They mentioned that the arrival process can be depicted as a Poisson process. Multi-tier applications can also be addressed using queuing networks, either considering a queue per server [USC+08], or just a queue per tier [ZCS07]. The first approach is adopted by [USC+08] use a G/G/1 queues for each server to depict multi-tier application. The peak workload is estimated by using this queuing model to determine and prepare servers for each tier. These may leads to great low resource utilization. [ZCS07] uses one queue to model each tier of the application. Closed queuing model is used by them for the finite number of users.

Time-series analysis based runtime scaling methods can be seen as proactive methods because they use the past data to predict future data. Many time-series techniques are used for prediction purpose. Moving average is used in [PH09], [LBCP09] to remove noise from time-series. Moving average usually can't be used to achieve good result [GGW10], [HLY12]. [MWY+10] takes both current and history data into consideration by using quadratic exponential smoothing on real workload and achieve high accurate of prediction.

Auto-regression is adopted in [KSJB09], [CHL+08], [GGW10], [CGS03], [KGM10]. [KSJB09] uses auto-regression with order 1. [RDG11] predicts workload based on latest three data and using a second order ARMA. Then, the response time will be estimated based on the prediction of workload. Finally the response time and costs will be used to make the best runtime scaling decision.

History window can be used with neural network [IKLL12] or multiple linear regression equation [KSJB09], [BGS+09], [IKLL12].

[KSJB09] mentions that the model can be too sensitive to the workload because of the size of sample of window. Regressing over changing size of windows and using the mean of all predictions are proposed to avoid these. [IKLL12] proposes predicting every 12-minute because of the typically 5-15 minutes of VM boot up time. As mentioned before, repeated patterns recognizing is another goal of using time-series for runtime scaling problem. [GGW10] compares methods and identifies CPU, memory, network and I/O resource usage pattens by using FFT.

4.4.3 Time series based workload forecasting methods

Our runtime control method will be based on the mapping knowledge from workload to configuration. Among aforementioned methods, time series analysis is the most promising one for proactive runtime control. Time series can be used on workload to perform forecasting. Therefore, we focus on time series analysis methods.

A time series is a sequence of data points, typically consisting of successive measurements made over a time interval [Ham94]. A time series can be decomposed into trend, season and noise [HK07], [HKOS08], [SS10]. Trend usually can be described by linear monotonic increasing or decreasing functions. Breaks may be existing due to external factors and be inferred according to the durations of historic trends. Season describes the repeated patterns in the time series. Noise describes the random unpredictable influence of the fast transforming factors. Applying smoothing techniques in an appropriate degree can be helpful to reduce the noise to improve the accuracy of forecasting as well as to control the information loss within an acceptable level.

Time series analysis comprises methods for analysing time series data in order to extract meaningful statistics and other characteristics of the data. Time series forecasting is the use of a model to predict future values based on previously observed values [wik15b]. In [HK07] [HKOS08] [SS10], time series forecasting approaches are discussed in detail. Here we briefly describe the conditions of usage, merits and demerits.

The listed time series forecasting strategies can deal with different objectives, data characteristics.

Naive forecast methods assume that the next observed value has the highest probability to be the same as the most recently observation. Naive forecast method requires only one historical data to be applied.

Moving average method uses the data included in a sliding window to do forecasting. The naive forecast method can be seen as a special case of moving average method with a sliding window size of one. Taking the average of the observation values in the sliding window can help to remove noise in a certain degree.

Simple exponential smoothing method takes the weighted moving average value of the historical observations in the sliding window. The more recent values are assignned with higher factors while the remote ones are with lower factors. Simple exponential smoothing method can better deal with the influence of trend and seasonal than moving average method because of the weight factors. However, simple exponential smoothing still set the predicted value back from the change introduced by trends and seasonal. In another word, simple exponential smoothing method can not take trend and seasonal into consideration. Cubic smoothing splines [HKPB05] can be applied in unidimensional data and the trend of the time series can be linearly extimated. Cubic smoothing splines [HKPB05] method can better predict trends than simple exponential smoothing method but still can't detect the seasonal pattern of time series.

Croston's method [SH05] is suitable for time series containing zero values. Croston's method [SH05] decomposes the observed time series observation data as two time series, one without zero values and another records the durations of zero valued. Then, these two time series are processed seperately by applying simple exponential smoothing method.

Extended exponential smoothing [HKOS08] uses state space approach and bases on simple exponential smoothing. The trend and season can be detected and selectively compounded in the final forecast result.

tBATS [DLHS11] develops the extended exponential smoothing method through trigonometric representation to better detect seasonal pattern.

Auto-Regressive Integrated Moving Averages (ARIMA) [Box76] has seven parameters to define the trend, noise and seasonality. ARIMA model can accept zero or positive integers as parameters. The selection of parameters can be achieved through space limitation and intelligent model space traversion by unit-root tests and Akiake's information criterion (AIC). The parameter selection has been automated in R forecast package. The ARIMA (1,0,1) model is one of the ARIMA models. ARIMA (1,0,1) assumes the time series is stationary. It applies auto-regressive and moving average.

The computational complexity of these time series forecasting strategies are discrepant. A detailed comparision of these methods are provided in [HHKA14]. [HHKA14] provides Workload Classification and Forecasting (WCF) to select the most applicative time series forecasting method automatically at runtime. In this thesis, we will adopt WCF to perform the forecasting. WCF will be introduced and discussed in chapter 7.

4.5 Benchmarking

We will propose a runtime control method based on benchmark. We use benchmark results to explore knowledges and workload forecasting as a mean to proactive adapt the configuration. In this section, we will introduce benchmarking and tools used by us.

The general benchmarking approach will be introduced in section 4.5.1. We adopt benchmarking to achieve application knowledges and manage SLA accordingly. Our SLA oriented benchmarking will be discussed in section 4.5.2. We adopt CLIF and SelfBench as our benchmark tools. They will be introduced in section 4.5.3.

4.5.1 Benchmarking definition

The performance of a system is difficult to be judged from the configuration parameters in computing technology. Benchmark is seen as an effective method of evaluating specific performance metrics of an object by running a set of tests. One benchmarking is composed of system under test, workload injection, performance monitoring and benchmark result analysis. Benchmarking should be quantificational and comparable. For example, the benchmark of CPU float point operation performance and the bandwidth and delay of data access can help users to judge whether each kind of CPU can run the application with proper computing performance and operating capacity. Benchmarking is a widely used method for judging the performance of CPU, but it can also be used in software systems. For example, the benchmark on Atomicity, Consistency, Isolation, Durability (ACID), query time and online transaction processing (OLTP) of database management systems (DBMS), can help users to choose the most suitable database. Benchmark is useful for the application developer to understand their application and optimize software or hardware environments to advance the performance of their target application. Application benchmark can measure the performance of the application better by running the real workload or customized workload on the real application to achieve an achievable and comparable result. There are many challenges when adopting a benchmark based approach. In the context of cloud computing, large scale of computing resources can be available. Running a benchmark on real large scale system can be costly and not practical. Cost effective pre-defined

workload and extrapolation based methods are required. It's not easy to define a cost effective workload for small scale system benchmarking to achieve a set of comparable results which can be directly used to decide which system performs better. It's also difficult to understand and explain the benchmark results and extrapolate the performances on large scale system. The design of benchmarking should also follow scientific method such as variable control, small sample size and repeatable results [Cas06].

4.5.2 SLA oriented benchmarking

SLA oriented benchmarking run different workload described in SLA on target application and observe the response of the application system according to SLA. SLA oriented benchmarking process is necessary comparing to traditional benchmarking which aims at providing a score of the system to generally mark the system performance. SLA oriented benchmarking focus on the evaluation of the ability of satisfying customers' requirements by taking the QoS requirements specified in SLA into consideration. By doing these, the benchmark results can be more persuasive for customer to judge the performance of the system. The performance of a web application can be affected by the software environment, hardware environment, operating system, network bandwidth and workload. The performance of a web application can be analysed from different perspectives. For example, an application with good performance means rapid response and no access denying from end users' perspective, while throughput and resource utilization from application owners' perspective. The performance of an web application is evaluated from several models, for example, workload model, performance model and cost model. Workload model describes the characteristics of system resource requirements and system resource utilization when performing different types of requests on the system. Performance models are used to predict response time, resource utilization and system throughput. Cost model describes the affection of software, hardware, network and supporting system operation costs. A series of performance metrics can be used in performance model, for example, response time, throughput, resource utilization, concurrent users, HTTP transactions per second and session per second, network statistics, resource request queue length.

- Response time is waiting time from the moment end user request is sent from the client to the moment the client received the response from the server. It is usually measured by time, for example seconds per request or milliseconds per request. Usually, response time is inversely to free system resource and in direct proportion to workload.

  Figure 4.9 illustrates the relationship between workload and response time. As depicted in figure 4.9, with the increasing of workload, the response time increases slowly and almost linearly until one or more resources are widely consumed and intensely competed. In figure 4.9, the sudden increasing of response time is usually because at least one of the resources utilizations are already beyond the maximum resource utilization thresholds. For example, one web server uses a fixed number of threads to process requests. When the number of concurrent requests exceeds the number of threads affordable by the web server, any coming request will be added into the request queue and wait to be processed. The time spend in request queue will also be calculated as part of response time.

  To better understanding the meaning of response time in the context of web application, we can cut response time into small pieces. These small pieces of time can be classified as network waiting time and application waiting time. Network waiting time means the

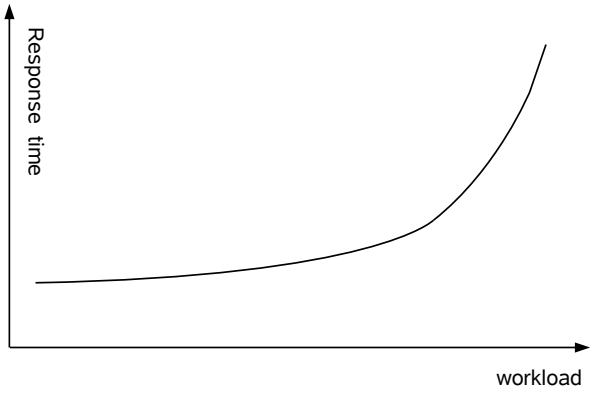

Figure 4.9. The relationship between workload and response time

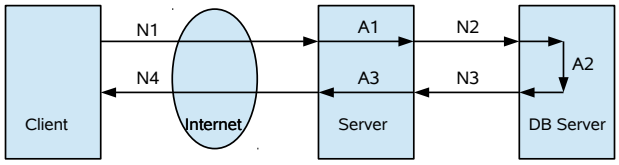

Figure 4.10. The response time composition in typical web application

time data is transformed from one server to another. Application waiting time means the time for data staying inside one server and being processed.

As depicted in figure 4.10, response time = (Nl+N2+N3+N4)+(A1+A2+A3). Nx represents network waiting time. Ax represents application waiting time. Usually, response time is highly depending on N1 and N4 which are decided by the way of client connected to servers. Methods, such as installing web application closed to their clients, are used to reduce the time spent on data transforming on internet. Network waiting time N2 and N3 usually depend on the capability of network card interface of servers. It is difficult to analyse and decrease the application waiting time A1, A2 and A3 because of the complicity of software. Especially when several components of the application works together to one single request.

- Throughput means the number of requests processed by the system during a specific time unit. The measuring unit is usually requests/second or pages/second. Throughput is an important concept during the whole lifecycle of an application. For example, during capacity planning phase, throughput is the key metric for deciding the hardware configuration of web application. Besides, throughput plays an important role in recognizing performance bottleneck and performance improvement.

  Figure 4.11 illustrates the relationship between throughput and workload. At the beginning, the throughput increases proportionally to the workload. However, due to the limitation of system resources, throughput can not be indefinitely increasing. It will gradually reach the maximum throughput , and then the whole system throughput will decrease as the workload increases. This maximum throughput depicts the maximum number of con-

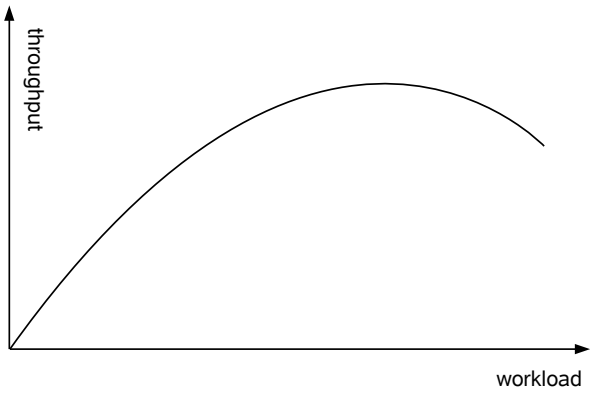

Figure 4.11. The relation between throughput and workload.

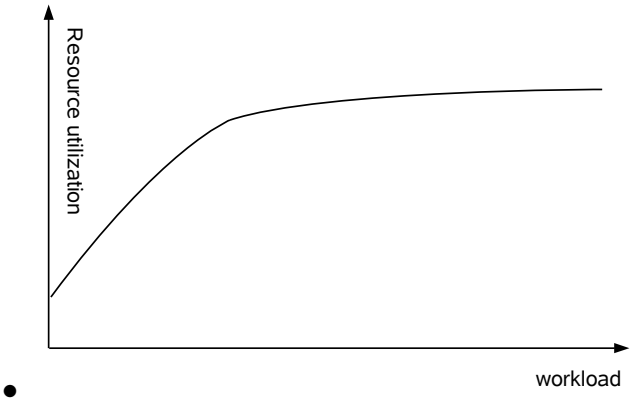

Figure 4.12. The relation between resource utilization and workload.

current user requests can be served by the system. Throughput and response time can be seen as two different perspective of the same problem. Generally, throughput will be smaller if response time is longer. It is not right to ignore waiting time when measuring throughput because the waiting time usually suddenly increases before reaching the maximum throughput. Maximum throughput will probably appear on a workload point where the response time is not acceptable.

Resource utilization refers to the utilization degree of different resources of the system, such as CPU, memory, network bandwidth. It is usually expressed as the used percentage of the maximum usable amount of resources. Figure 4.12 shows the relationship between resource utilization and workload.

Generally speaking, resource utilization is proportional to the workload until workload reaches a certain number from where the resource utilization will remain at a constant level as the workload continues growing. This constant level of resource utilization is seen as the maximum availability of this resource. When resource utilization is maintained at a constant value of 100%, it indicates that the resource has become the bottleneck of the system. System throughput can be increased and response time can be decreased by prompting the capacity of this resource. In order to locate system bottlenecks, a long and arduous performance benchmarking process is unavoidable to check all

suspicious resource by adding more this kind of resources and observing the improvement of performance.

- The number of concurrent users is the number of user sessions on a particular site defined within a given time. When the number of concurrent users increases, system resource utilization will also increase.

- The number of requests per second and Number of sessions per second. The number of requests per second refers to the number of GET or POST requests per second reaches the Web sites . Number of sessions per second is the number of users per second to reach and visit the web site. It is closely related to performance.

- Network traffic statistics should also be monitored to determine the appropriate network bandwidth. Typically, network bandwidth can be the bottleneck of the application when network utilization is more than 40%.

4.5.3 Benchmarking tools

To benchmark the application, we use SelfBench services which are provided as services under OpenCloudware project. SelfBench [Ben12] provides a virtualized and self-scalable load injection system to automatically detect the supportable upper bounder arrival rate of the system. SelfBench is based on CLIF. CLIF [Dil09] is designed for generating predefined traffic on a system to measure QoS target and observe the computing resources usage at the same time. CLIF can be used for deployment, remote control, monitored measurements collection of its distributed load injectors and probes. Acceptable request response time at the injector side and resource utilization upper bounder (e.g. RAM utilization = 80%) at the probe side is configured as the termination conditions of detection. System maximum capable arrival rate and corresponding resource utilization can be achieved by applying SelfBench. In other words, after doing SelfBench on system under test, we can get maximum arrival rate for current system under test and resource utilization for each instance.

CLIF, a benchmarking environment

CLIF [Dil09] is a software based on Fractal component and written in Java. CLIF [Pro] can be used to charge injection and performance measurement. With its software framework approach, it is especially designed to adapt to any kind of system under test in terms of invocation protocols and resources to observe. It also offers different user interfaces such as Separate GUI, Eclipse plug-ins, command line and continuous integration server Jenkins.

Figure 4.13 shows the execution principles of CLIF test. CLIF is a distributed Fractal application to deploy and supervise injection components and resource consumption measurement through the network.

The factory-type CLIF servers are used by injector or sensor components. These enable the instantiation of remote components. When the test is started, the injectors begin to send requests to the system under test and produce reports with response time and type of request.

In parallel, the probes measure the specific resources utilization, such as CPU utilization, memory utilization, and then produce measurement reports.

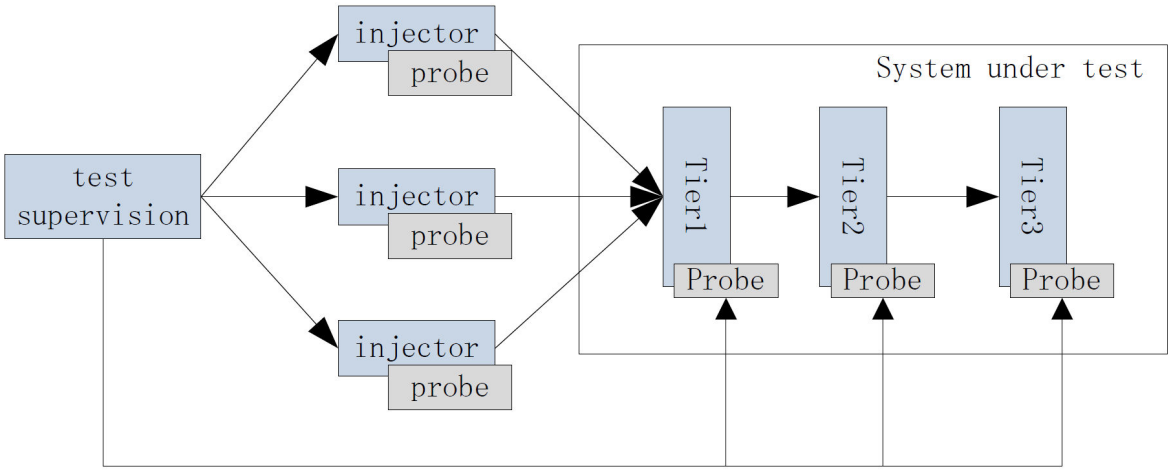

Figure 4.13. CLIF Architecture [Pro].

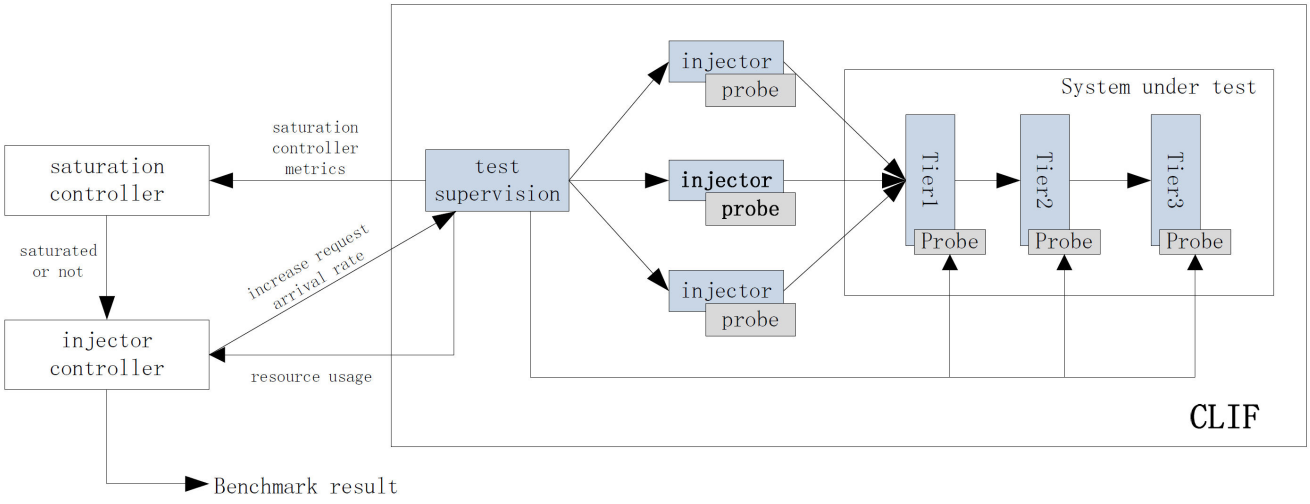

Figure 4.14. SelfBench architecture [Ben12].

All these measurement reports will be stored in a storage component for the analytical component to produce test report. These measurements will also be available after running a collection operation at the end of the test. Compared to centralized storage component, measurements are stored on each injector and probe to avoid interrupting the performance of system under test. Instead of transporting and storing all the measurements in centralized storage component in real time, distributed measurements storage allows CLIF to support High traffic, such as millions of virtual users, thousand injectors and probes.

CLIF provides periodical statistics of measured metrics on injectors and probes offer, such as average response time and average CPU utilization. This feature allows to monitor the changing of metrics in real time and in an intuitive way.

Figure 4.13 describes the component architecture of CLIF. There are storage, analyser and blades. Blades can be either injectors or sensors. All the sensors, injectors and storage are connected to and under the control of one management component named as supervisor. Detailed explanation of the architecture can be found in [Dil09].

SelfBench

SelfBench [Ben12] is based on CLIF without modification. Blades, the fundamental of CLIF, provides CLIF with the ability of deploying and controlling parallel computing components. With blades, measurements can be observed and controlling can be performed at runtime which means before retrieving the results by "collecting" at the end of execution.

SelfBench benefits from the standard features of CLIF including injectors, probes, configuration and starters to compose SelfBench deployment.

Compared to CLIF, the different part for SelfBench are an optimization controller including the injection controllers and saturation controllers.

Figure 4.14 demonstrates the SelfBench architecture. The Supervisor component, the same as in CLIF, controls and supervises the whole platform of test.

Informations such as the state of the test, the state of the injectors and probes and runtime periodical statistics measurements can be found.

The activity of the probes and injectors can be controlled and reconfigured at runtime. All the informations are centralized and achievable.

The injection control component and the saturation detection component implement two separate control loops. They specify a unique customer interface of type of Test Control. Test Control type allows the controllers to control, monitor and reconfigure all the deployed probes used by the saturation controller.

Optimization component is also Fractal controllerType. It's connected to the supervisor component. It can be used to start the configurators and starters with well defined parameters.

4.6 Conclusion

Benchmarking provides cloud application dependent analysis of the application behavior, for instance throughput of served requests, response time. Through benchmark, the feasibility of a SLA can be known. In other words, the question that whether QoS target can be satisfied with the workload, resource and cost constraints stated in SLA can be answered. We will describe our benchmark based SLA feasibility study method in chapter 6. Benchmarks are relatively costly and precise feasibility study usually imply large amount of benchmarks. Our benchmark based

SLA feasibility study method makes tradeoff between accuracy and costs. Limited amount of benchmarks are performed to achieve a workload-resource mapping model. This mapping model, as an intermediate of this benchmark based feasibility study process, will also be used in our runtime control method which will be introduced in chapter 7.

Runtime control aims at provisioning resources for running applications to satisfy the terms defined in SLA. Existing runtime control methods are in a schedule, reactive or proactive way, based on rule, control theory, queuing theory, reinforcement learning or time series. Schedule based runtime control methods can not adapt to fluctuating workload. However, reactive runtime control methods require great efforts and knowledges about the running application behaviour from application owners, while proactive runtime control methods have strong constraints which block these methods from applying in practice. Therefore, we will propose a runtime control method which releases the burden of application owners through a set of benchmarking in chapter 7. The results of benchmarks will also provide an initial proactive runtime control model to make our method more practical. This initial proactive runtime control model will be continuously improved in a reactive way with the accumulation of application behaviour knowledges to make our runtime control method better adapting to the application. Our runtime control method is based on schedule, reactive and proactive runtime control methods.

Chapter 5

PSLA: SLA description language for PaaS

5.1 Introduction

Most cloud services are provided in the way of "Pay as you go". PaaS is one kind of these services. PaaS is a special kind of SaaS, and it is based on IaaS. IaaS providers loan virtual resources, like virtual machine, to their clients. SaaS providers loan software usage rights to their clients through internet. PaaS providers run their business by loaning a platform for whole SaaS application lifecycle, including development, testing, deployment, runtime maintaining [MG11]. It is widely accepted that SaaS providers save investments on purchasing and maintaining hardware by using IaaSs. But activities and difficulties involved in SaaS application lifecycle are quite different from past. So PaaS is designed to lower the sill of using cloud infrastructure and popularize the cloud. By using PaaS, SaaS providers can ignore the differences among IaaSs, accelerate SaaS application development and acquire elasticity of SaaS.

SLA is a part of the agreements between service consumer and service provider. It is used to structurally describe QoS target. SLA is needed for several reasons. The responsibilities of contracting parties should be clearly defined in SLA. For example the actions carried out by each part in different conditions. Usually, in SLA, the services promised by provider are quantitatively and comprehensively defined by a series of QoS metrics with value ranges. Not only the level of QoS should be guaranteed but also the corresponding rewards and penalties will also be specified in SLA for the fulfilling or violating of promise. Most of the management involved in cloud computing, including SLA management, relies on automatic tools. So structured and machine readable SLA definition, e.g. in XML, is necessary so that the SLA can be used as input file of SLA management system. Other management systems in the cloud may also benefit from it. For example, billing systems can use penalty and reward attached to each QoS promise to charge to service consumer. In reality, most cloud service providers offer SLA in text, even informative way. For example, public cloud service Google App Engine [goo15] and Amazon Elastic Compute Cloud (EC2) [ec213] publish their uniform and non custom SLA online.

In this chapter, we introduce PSLA [LPM14], a PaaS level SLA description language. PSLA is based on WS-Agreement [ACD+07]. WS-Agreement is an extendable SLA skeleton. We consider the particular needs in PaaS level SLA and design PSLA properly based on WS-Agreement skeleton. PSLA is well structured and practical. To our best knowledge, we are the first one who propose a commonly usable and machine readable PaaS level SLA description language.

In this chapter, PSLA will be explained with more details compared to our previous publication [LPM14]. The rest of this chapter is arranged as follows. The specific problems of PaaS are analysed in section 5.2. In section 5.2, the semantic expression of PSLA is also summarized. In section 5.3, a detailed description of the syntactic PSLA is made. Finally, we conclude this chapter in section 5.4.

5.2 PaaS SLA requirements

If SaaS provider authorize PaaS provider to deploy and run application, the SaaS level SLA should be guaranteed by PaaS provider. So SaaS level SLA will be expressed in corresponding PaaS level SLA with appropriate constraints. Because the end users' behaviour pattern can be formed based on SaaS level contracts or end user statistic which probably are business secrets. So the workload is elastic and the elasticity is unpredictable for PaaS provider. PaaS provider can only deliver under a certain workload. This is expressed as constraints and they are obligatory.

In this section, the requirements of a PaaS level SLA will be discussed in section 5.2. The semantic description of PSLA will be given in section 5.2.1. The features of PaaS level SLA will be given in section 5.2.2.

5.2.1 PSLA semantic description

In this part, we list and explain semantic clause involved in PaaS level SLA. PaaS level SLA includes two kinds of informations. They are general informations about the SLA named as "SLA Metadata" and detailed QoS target descriptions of the SLA named as "QoS Criterion". "Springoo", a three tier application, is used in OpenCloudware project as an example to demonstrate its QoS management. The first tier of "Springoo" runs Apache server, the second runs Jonas server and the third tier runs Mysql server. We can instantiate several application servers to do horizontal scaling on the second tier. We can change the allocated virtual CPU or memory to do vertical scaling on third tier.

SLA Metadata

SLA Metadata defines the basic informations of the SLA. These informations are constant within one SLA. SLA metadata includes:

- Agreement Name

  Agreement Name is the identification of the agreement. It is a unique ID or name. e.g. Name of the agreement is “Springoo”.

- Agreement Initiator

  Agreement Initiator is the party who starts the agreement creation request. For example, the initiator of this agreement is "Springoo Developer".

- Agreement Responder Agreement Responder is the party who responds to the agreement creation request. For example, the responder is OpenCloudware.

- Service provider

  Service provider is the party who will provide services. The contrary party is service consumer. e.g. Service provider is also OpenCloudware.

- Agreement Expiration Time

  Agreement Expiration Time is the time point that current agreement is no longer valid. All the "Valid Period"s defined in "QoS Criterion" should be subsets of it. E.g. Agreement will be expired at 01/09/2015 00:00:00.

QoS Criterion

One important purpose of creating SLA is giving an insurance to the QoS delivered by service provider. In this part, the detailed clauses are described. The rights and obligations of each party are defined.

- Valid Period:

  Valid Period specifies the period of time that this QoS Criterion will be taken into effect. Valid Period can either be continuous or regularly discrete 5.2.2. This is achieved by a start time, a end time and a optional time Interval.

- Valid Workload:

  Valid Workload describes that the QoS Criterion will be taken into effect for what kinds of workload acting on SaaS application. The workload can be described from three aspects: data size, data composition and data variation.

- Service Level Objective:

  Service Level Objective (SLO) is the core part of QoS Criterion. It expresses what exactly the QoS target is by quantifying the acceptable QoS metric value range. Sometimes, it is even difficult for service consumers themselves to specify a value as an exactly threshold between satisfactory and unsatisfactory. So, inspired by "Confidence" and "fuzziness" proposed by CSLA, we use "Trust" and "Fuzziness" to allow the fuzzy description of threshold 5.2.2.

- Metric Description:

  In QoS Criterion, each SLO is quantified by QoS metric value scopes. The metrics are described to make sure that there is no ambiguity about the metric among parties 5.2.2. This is achieved by a unit, a value Type, value Source and value update mode.

- Business Values:

  Different SLOs are usually reflected in different business values, like penalty and reward. These values will be used in billing system to calculate the cost of service. Relevant penalty will play a part when the Service Level Objective is not met.

- Check Plan:

  Billing system needs to check the QoS and service promised in SLA to do the calculation. The time when this action should be take is specified in SLA.

- Priority:

  Sometimes the service provider doesn't have enough resources to meet all SLO at the same time, so it is necessary to specify priority for each SLO. So that the service provider can make trade-off.

5.2.2 PaaS SLA Features

PSLA is designed for the contract between PaaS provider and application owner. PaaS provider takes charge of the dynamic resource provisioning caused by fluctuating workloads. Usually, application owners will have constraints such as available resources or maximum cost for maintaining the QoS. Therefore, the application probably could not run application under unlimited workload. Metrics should be described without ambiguity so that the QoS, which need to be guaranteed by PaaS provider, can be clearly defined. Inspired by CSLA [KL$^+$12], giving a fuzzy border to the acceptable QoS range can be helpful for managing the application in a cost effective way, we also introduce the fuzzy value range into PSLA.

Elastic workloads

The elasticity of a given time can be expressed by a three-dimensional vector (s, c, t):

- s, Data size, means the total amount of data during a given unit of time.
- c, Workload composition, means the proportion of each kind of load which form the workload. Workload composition itself can be described by a multi-dimensional vector and each dimension can be defined by the resources required by a unit amount of the load. The details are described in the following content.
- t, Time, means the time when the workload happened.

Different services need different amount and various kinds of resources. From the PaaS providers' point of view, resource needs is the fundamental gist to ensure the promised QoS. So we classify the loads according to the amount of each kind of resources demanded on each component(We consider the component based software system). We use matrix $L^i_{M\times N} = \left\{l^i_{m\times n} | l^i_{m\times n} \geq 0, 0 \leq m < M, 0 \leq n < N\right\}$ Where $L^i_{M\times N}$ represents the load i. $l^i_{m\times n}$ represents the amount of $resources_n$ required on $component_m$ by $load_i$. M represents the number of components in software system. N represents the number of kinds of resources. Since there are limited kinds of resources, like memory usage rate, we can summary commonly used resource metrics and use a fix single dimension vector here. Now, we describe it as $(CPU, RAM, Network)$ preliminary, and these will be lucubrated in our future work.

Workload is composed by several loads. Different ratios of loads can lead to different distributions of total resources demanding.

Metric

To model PaaS level metric, the metrics that are interesting should be find out first. Then, we abstract and define a structure to express all necessary metric properties.

For PaaS level SLA, as far as we know, there is no existing metric model. In [AAA+10], some of the commonly used performance metrics, including throughput, reliability, load balancing, durability, elasticity, linearity, agility, automation and customer service response times, are listed. Since SaaS level QoS metrics may partly be transferred to PaaS level SLA, we take SaaS QoS metrics, like response time, into consideration . We consider the following properties initially:

- Unit: The measurement unit of this metric. Common used units, like second, are built-in in PSLA, but there is also mechanism for user defined unit.
- Value type: Which type the value of the metric should be, for example string, integer or decimal.
- Value source: Where the value of this metric is acquired from. PSLA allows to use a URI to describe the location of metric value.
- Value update mode: The way new metric value is retrieved from the value source.

Fuzzy Value Range

We use elementary logical expression to describe value range. We define logical operators, comparison operators and boundary values. However, it is difficult to give a fine line as boundary in the real world. So we adopt "fuzziness" and "confidence" as in [KL+12] to make the boundary more practical. "confidence" is renamed as "trust" to distinguish statistical term.

- Logical operators: Including $\cap$, $\cup$ and $\neg$.
- Comparison operators: Including$<$, $>$, $\leq$, $\geq$ and $=$.
- Metric: Value range acts on which metric.
- Boundary values: the boundary of range.
- Trust: At least how many percentage of the metric value should be within the value range described by logical operators, comparison operators and boundary values.
- Fuzziness: The acceptable degree of deviation of metric values which are out of bounds.

5.3 PSLA syntactic expression

```
<wsag:AgreementOffer ...>
  <wsag:Name>Springoo</wsag:Name>
  <wsag:Context>...</wsag:Context>
  <wsag:Terms>...</wsag:Terms>
</wsag:AgreementOffer>
```

Listing 5.1. WS-Agreement based PSLA structure example.

In this section, we will demonstrate PSLA's expression power by introducing whole syntactic structure of PSLA. As demonstrated in listing 5.1, AgreementOffer is comprised by Name, Context and Terms. This section is focus on AgreementOffer but not AgreementTemplate. AgreementOffer delivers the same informations as AgreementTemplate only lack Constraint to express when the offer can be accepted as formal AgreementOffer. As mentioned before, PSLA is based on WS-Agreement skeleton. PSLA extends WS-Agreement from five aspects,

including Context, Service Description Term, Service Reference, Qualifying Condition and CustomServiceLevel.

In this section, the syntactic expression of PSLA will be given. We present here the syntactic expression of Agreement Context and the syntactic expression of Agreement Terms. The detailed explanations of the syntactic expression of Service Term will be described in section 5.3.3. The syntactic expression of Guarantee Term will be described in section 5.3.4.

5.3.1 Agreement Context

```
<element name="ContextExtension" type="ocw:ContextExtensionType"/>
<complexType name="ContextExtensionType">
  <sequence>
    <element name="OVFLocation" type="anyURI"/>
  </sequence>
</complexType>
```

Listing 5.2. Context extension in PSLA.

In WS-Agreement, Context is the place to describe SLA Metadata, including Agreement Name, Agreement Initiator, Agreement Responder, Service provider and Agreement Expiration Time, of the AgreementOffer. Context is designed to be the place for expressing various metadatas about the agreement. Pre-mentioned informations may not be enough and WS-Agreement is designed to be extendable. As demonstrated in LIST 5.2, in PSLA, Context is extended to express the "OVFLocation", which means the location of VCAA file. In the context of OpenCloudware, OVF++ file, which describes the VCAA, is adopted.

```
<wsag:Context>
  <wsag:AgreementInitiator>Springoo Developer</wsag:AgreementInitiator>
  <wsag:AgreementResponder>OpenCloudware</wsag:AgreementResponder>
  <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
  <wsag:ExpirationTime>2015-01-01T00:00:00</wsag:ExpirationTime>
  <ocw:ContextExtension>
    <ocw:OVFLocation>
    http://www.polytech.univ-savoie.fr/opencloudware/v2/models/springoo.ovf
    </ocw:OVFLocation>
  </ocw:ContextExtension>
</wsag:Context>
```

Listing 5.3. PSLA Context example

Listing 5.3 gives an example of PSLA agreement Context. In this example, the agreement is firstly proposed by Springoo Developer. OpenCloudware is the one who responds to the PSLA proposal. OpenCloudware is also the one who provides the services which are guaranteed under this PSLA agreement. This agreement will be effective until 0 o'clock of 01/09/2015. The VCAA description of the guaranteed application can be found through the URL of "http://www.polytech.univ-savoie.fr/opencloudware/v2/models/springoo.ovf".

5.3.2 Agreement Terms

Terms is the core part of AgreementOffer. Services and QoS promised by service provider are described in Terms. Most of our works are inside Terms. The syntactic expression of Service

Term and Guarantee Term will be described in section 5.3.2. The structure of Agreement Term will be described in section 5.3.2.

Classification

There are two kinds of terms in WS-Agreement. They are ServiceTerm (see section 5.3.3) and GuaranteeTerm (see section 5.3.4).

Structure

All terms are organised together by All, ExactlyOne and OneOrMore as a TermTree. LIST 5.4 is an example of TermTree. In this example, 3 ServiceTerms, including 2 ServiceReferences and 1 ServiceDescriptionTerm, and 1 GuaranteeTerm are organized together by "All" which means all of these four Terms should be taken into effect at the same time.

```
<wsag:Terms>
  <wsag:All>
    <wsag:ServiceReference Name="Counting" ServiceName="elasticity">...</
    wsag:ServiceReference>
    <wsag:ServiceReference Name="Monitoring" ServiceName="elasticity">...</
    wsag:ServiceReference>
    <wsag:ServiceDescriptionTerm Name="Springoo Description Terms"
    ServiceName="elasticity">...</wsag:ServiceDescriptionTerm>
    <wsag:GuaranteeTerm Name="Request Response Time" Obligated="
    ServiceProvider">...</wsag:GuaranteeTerm>
  </wsag:All>
</wsag:Terms>
```

Listing 5.4. One Terms example according to WS-Agreement.

5.3.3 Service Terms

ServiceTerm mainly contains functional descriptions. There are three kinds of ServiceTerm in WS-Ageement. They are ServiceDescriptionTerm, ServiceReference and ServiceProperties. ServiceDescriptionTerm and ServiceReference have the same syntax expression and they can be extended. ServiceDescriptionTerm is designed for domain specific description, and we have extended it to express "Metric Description". In PSLA, ServiceReference has been extended to describe a service provided by PaaS provider. One or more URIs are used to refer to the entry point of service. In WS-Agreement, ServiceProperties is used to packet all the QoS metrics as Variable which will be used in GuaranteeTerm. In this section, the syntactic expression of Service Term will be described in section 5.3.3. The syntactic expression of Service Description Term will be described in section 5.3.3. The syntactic expression of Service Reference will be described in section 5.3.3. The syntactic expression of Service Properties will be described in section 5.3.3.

```
<complexType name="NamedElement">
  <attribute name="Name" type="ID" use="required"/>
</complexType>
```

Listing 5.5. Shared type NamedElement definition in PSLA.

NamedElement is a shared type defined in PSLA. As shown in LIST 5.5, an extension based on NamedElement must be tagged by a Name which is unique in the whole PSLA document. In PSLA, three types of elements are extended based on NamedElement, including InputdataType, ResourceType and MetricType.

Service Description Terms

```
<element name="ServiceDescriptionTermExtension" type="ocw:
    ServiceDescriptionTermExtensionType"/>
<complexType name="ServiceDescriptionTermExtensionType">
  <choice maxOccurs="unbounded">
    <element name="Metric" type="ocw:MetricType"/>
    <element name="InputData" type="ocw:InputDataType"/>
    <element name="Resource" type="ocw:ResourceType"/>
  </choice>
</complexType>
```

Listing 5.6. ServiceDescriptionTerm extension definition in PSLA.

In WS-Agreement, ServiceDescriptionTerms is designed to express domain specific service descriptions. ServiceDescriptionTerms can be extended for customer needs. In PSLA, ServiceDescriptionTerms, is extended as listing 5.6 to express Metric, InputData and Resource. They will be further used in Guarantee Terms to express QoS responsibilities and obligations unambiguously and clearly. Listing 5.7 is an example of ServiceDescriptionTerm in PSLA. The requests,"Atomic_Request_A" and "Atomic_Request_B", composing the workload are described as "InputData" of ServiceDescriptionTerm extension. computing resource, such as Jonas, Apache and MySQL, and storage resources, such as AppDiskID1, AppDiskID2 and AppDiskID3, are described as "Resource" of ServiceDescriptionTerm extension. All the metrics which will be used in corresponding GuaranteeTerms 5.3.4 will be described here. For example, "RequestsThroughput", "Variation" and "Instructions" are the metircs which will be used to express the workload constraints in QualifyingCondition of GuaranteeTerms 5.3.4. "Memory", "NetworkThroughput" and "Storage" are the metrics which will be used to express the scalability constraints in QualifyingCondition of GuaranteeTerms 5.3.4. "ResponseTime" and "Availability" are the metrics which will be used to express the scalability constraints in QoS requirements of GuaranteeTerms 5.3.4. It can also be the case that there is no Metric, no InputData or no Resource.

```
<wsag:ServiceDescriptionTerm Name="Springoo Description Terms"
    wsag:ServiceName="elasticity">
  <ocw:ServiceDescriptionTermExtension>
    <ocw:Metric Name="RequestsThroughput">...</ocw:Metric>
    <ocw:Metric Name="Variation">...</ocw:Metric>
    <ocw:Metric Name="Instructions">...</ocw:Metric>
    <ocw:Metric Name="Memory">...</ocw:Metric>
    <ocw:Metric Name="NetworkThroughput">...</ocw:Metric>
    <ocw:Metric Name="Storage">...</ocw:Metric>
    <ocw:Metric Name="ResponseTime">...</ocw:Metric>
    <ocw:Metric Name="Availability">...</ocw:Metric>
    <ocw:InputData Name="Atomic_Request_A">...</ocw:InputData>
    <ocw:InputData Name="Atomic_Request_B">...</ocw:InputData>
    <ocw:Resource Name="Jonas">...</ocw:Resource>
    <ocw:Resource Name="Apache">...</ocw:Resource>
```

```
    <ocw:Resource Name="MySQL">...</ocw:Resource>
    <ocw:Resource Name="appDiskId1">...</ocw:Resource>
    <ocw:Resource Name="appDiskId2">...</ocw:Resource>
    <ocw:Resource Name="appDiskId3">...</ocw:Resource>
  </ocw:ServiceDescriptionTermExtension>
</wsag:ServiceDescriptionTerm>
```

Listing 5.7. ServiceDescriptionTerm PSLA extension example.

1. wsag:ServiceDescriptionTerm
   /ocw:ServiceDescriptionTermExtension/ocw:InputData

   InputData is the place to define the basic element of workload which will be suffered by the virtualized application deployed under current PSLA contract. As shown in listing 5.8, in the definition of the InputData, the detailed description of the basic element of workloads will be given, including Scenario, Description and a unique name. It can also be extended.

```
<complexType name="InputDataType">
  <complexContent>
    <extension base="ocw:NamedElement">
      <sequence>
        <element name="Scenario" type="anyURI"/>
        <element name="Description" minOccurs="0" type="string"/>
        <any minOccurs="0" namespace="#any" processContents="strict"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Listing 5.8. InputData definition in ServiceDescriptionTerms extension of PSLA.

```
<ocw:InputData Name="Atomic_Request_A">
  <ocw:Scenario>
  http://www.polytech.univ-savoie.fr/opencloudware/v2/models/springoo.
    xis
  </ocw:Scenario>
</ocw:InputData>
```

Listing 5.9. One InputData example in PSLA.

   (a) wsag:ServiceDescriptionTerm
   /ocw:ServiceDescriptionTermExtension/ocw:InputData/@ocw:Name

   Attribute "Name" is an unique name based on NamedElement. The "NamedElement" based "Name" of InputData should be unique in the whole agreement. For example, in listing 5.9, "Atomic_Request_A" could be adopted as the name of an element only once in the whole PSLA document.

   (b) wsag:ServiceDescriptionTerm
   /ocw:ServiceDescriptionTermExtension/ocw:InputData/ocw:Scenario

   Scenario is a URI based external description. For example, in listing 5.9, Scenario can be a CLIF scenario file in the context of OpenCloudware.

(c) wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:InputData/ocw:Description

The string based Description is a human readable text description of the input data. This description is optional.

(d) wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:InputData/{any}

To make PSLA adaptable to a variety of application workloads, InputData is designed to be extendable.

2. wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Resource

```
<complexType name="ResourceType">
  <complexContent>
    <extension base="ocw:NamedElement">
      <sequence>
        <element name="XPath" type="string"/>
        <any minOccurs="0" namespace="\#any" processContents="strict
    "/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Listing 5.10. Resource definition IN ServiceDescriptionTerm of PSLA.

```
<ocw:Resource Name="Jonas">
  <ocw:XPath>
  /ovf:Envelope/ovf:References/pm:DynamicImage[ovf:id="appDisk1"]
  </ocw:XPath>
</ocw:Resource>
```

Listing 5.11. One Resource example in PSLA.

"Resource" of ServiceDescriptionTerms is the place to define referred virtualized resources provided by IaaS providers. Resources can be VMs, network resources, data storage and so on. As shown in listing 5.10, each Resource should have following informations:

(a) wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Resource/@ocw:Name

Similar as the "Name" of "InputData" defined in ServiceDescriptionTerms extension of PSLA, attribute "Name" is based on NamedElement which is a shared type defined in PSLA. For example, in listing 5.11, "Jonas" should be unique in the whole agreement.

(b) wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Resource/ocw:XPath

As mentioned before, OVF++ file is adopted by OpenCloudware project to describe the VCAA of the application which is managed under corresponding PSLA contract. "XPath" gives the location of the resource description, for example the XPath of a specific tag in an OVF++ file.

(c) wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Resource/{any}

Similar as "InputData" definition, "Resource" is also designed to be extendable to express all kinds of resource.

3. wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Metric

As defined in listing 5.12, each Metric should have the informations about metric unit, metric value type and metric value generation mode tagged as Unit, Type and ValueGenerateBy correspondingly. Detailed description of each element will be specified below.

```
<complexType name="MetricType">
  <complexContent>
    <extension base="ocw:NamedElement">
      <sequence>
        <element name="Unit" type="ocw:UnitType"/>
        <element name="Type" type="ocw:Type"/>
        <element name="ValueGenerateBy" type="ocw:ValueGenerateByType
    "/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Listing 5.12. Metric definition of ServiceDescriptionTerms extension in PSLA.

```
<ocw:Metric Name="Variation">
  <ocw:Unit>...</ocw:Unit>
  <ocw:Type>integer</ocw:Type>
  <ocw:ValueGenerateBy>...</ocw:ValueGenerateBy>
</ocw:Metric>
```

Listing 5.13. Metric example of PSLA.

(a) wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Metric/@ocw:Name

Similarly as the "Name" of "InputData" and "Resource", Metrics defined in ServiceDescriptionTerm will be later used in GuaranteeTerms 5.3.4 by referring the unique name based on NamedElement which is a shared type defined in PSLA.

(b) wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Metric/ocw:Unit

```
<complexType name="UnitType">
  <choice>
    <element name="BuiltInUnit">
      <simpleType>
        <restriction base="string">
          <enumeration value="second"/>
          <enumeration value="minute"/>
          <enumeration value="hour"/>
          <enumeration value="percent"/>
```

```
            <enumeration value="number"/>
            <enumeration value="Euro"/>
            <enumeration value="USD"/>
            <enumeration value="Pound"/>
            <enumeration value="requests/s"/>
            <enumeration value="MIPS"/>
            <enumeration value="Mo"/>
            <enumeration value="Go"/>
            <enumeration value="To"/>
            <enumeration value="Mb/s"/>
            <enumeration value="Gb/s"/>
          </restriction>
        </simpleType>
      </element>
      <element name="UserDefinedUnit" type="UserDefinedUnitType"/>
    </choice>
</complexType>
```

Listing 5.14. Unit definition in Metric definition of ServiceDescriptionTerms extension in PSLA.

"Unit" describes the unit of measurement of the metric value. "Unit" is the place to define how to understand the string based metric value, for example with the same metric value "3", "3 USD" related to money and "3 Mb/s" may related to data transforming speed. "Unit" can either be created according to customers' needs, which is named as UserDefinedUnit, or directly adopt the predefined units which is defined in BuiltinUnit.

(a) wsag:ServiceDescriptionTerm /ocw:ServiceDescriptionTermExtension/ocw:Metric/ocw:Unit/ocw:BuiltinUnit

As shown in listing 5.14, BuiltinUnit includes "second", "minute", "hour", "percent", "number", "Euro", "USD", "Pound", "requests/s", "MIPS", "Mo", "Go", "To", "Mb/s" and "Gb/s".

```
<ocw:Unit>
  <ocw:BuiltInUnit>Mo</ocw:BuiltInUnit>
</ocw:Unit>
```

Listing 5.15. One Unit example in PSLA.

In the example of listing 5.15, the predefined unit "Mo" is adopted.

(b) wsag:ServiceDescriptionTerm /ocw:ServiceDescriptionTermExtension/ocw:Metric/ocw:Unit/ocw:UserDefineUnit

As shown in listing 5.16, if none of the predefined units described in BuiltinUnit is suitable for describing the unit of measurement of the metric value, PSLA also provides a mechanism for customizing Unit according to UserDefinedUnit.

```
<complexType name="UserDefinedUnitType">
  <sequence>
    <element name="Description" type="string"/>
  </sequence>
  <attribute name="Name" type="string" use="required"/>
</complexType>
```

Listing 5.16. UserDefinedUnit definition for Unit of Metric in ServiceDescriptionTerms extension of PSLA.

To create a UserDefinedUnit, besides a string based name, a human readable unambiguous description of the unit of measurement of metric value should be given.

```
<ocw:UserDefinedUnit Name="requests/s/min">
  <ocw:Description>
  The variation of the number of requests per second during one
    minute.
  </ocw:Description>
</ocw:UserDefinedUnit>
```

Listing 5.17. One UserDefinedUnit example in PSLA.

In the example of listing 5.17, an UserDefinedUnit which name is "requests/s/min" is defined by describing meaning of this unit is "The variation of the number of requests per second during one minute.".

(c) wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Metric/ocw:Type

```
<simpleType name="Type">
    <restriction base="string">
        <enumeration value="boolean"/>
        <enumeration value="decimal"/>
        <enumeration value="float"/>
        <enumeration value="double"/>
        <enumeration value="short"/>
        <enumeration value="integer"/>
        <enumeration value="long"/>
        <enumeration value="negativeInteger"/>
        <enumeration value="positiveInteger"/>
        <enumeration value="unsignedLong"/>
        <enumeration value="unsignedInteger"/>
        <enumeration value="unsignedShort"/>
    </restriction>
</simpleType>
```

Listing 5.18. Type definition of Metric in ServiceDescriptionTerms of PSLA.

"Type" defines the numeric data type of the metric value. As defined in listing 5.18, PSLA provides 12 kinds of numeric data types, including "boolean", "decimal", "float", "double", "short", "integer", "long", "negativeInteger", "positiveInteger", "unsignedLong", "unsignedInteger" and "unsignedShort".

(d) wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Metric/ocw:ValueGenerateBy

"ValueGenerateBy" is the place to define how the metric value is retrieved. As shown in listing 5.19, the generating mode should be defined by restricting this metric value coming from which service and which operation on this service. The schedule of extracting the metric value is also defined.

```
<complexType name="ValueGenerateByType">
  <sequence>
    <element name="ServiceReferenceName" type="string"/>
```

```
    <element minOccurs="0" name="ServiceReferenceOperation" type="string
    "/>
    <element name="Schedule" type="duration"/>
  </sequence>
</complexType>
```

Listing 5.19. ValueGenerateBy definition in Metric of ServiceDescriptionTerm extension in PSLA.

```
<ocw:ValueGenerateBy>
  <ocw:ServiceReferenceName>Counting</ocw:ServiceReferenceName>
  <ocw:ServiceReferenceOperation>getVariation</
    ocw:ServiceReferenceOperation>
  <ocw:Schedule>PT1H</ocw:Schedule>
</ocw:ValueGenerateBy>
```

Listing 5.20. One ValueGenerateBy example in PSLA.

-i- wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Metric/ocw:ValueGenerateBy
/ocw:ServiceReferenceName

ServiceReferenceName represent a service already defined in ServiceReference 5.3.3 of ServiceTerm. ServiceReferenceName should not only be a string based name but also be unique. For example, in listing 5.20, "counting" represents only one service defined in ServiceReference.

-ii- wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Metric/ocw:ValueGenerateBy
/ocw:ServiceReferenceOperation

ServiceReferenceOperation describe that the metric is achieved exactly by which operation of the service which is referred by ServiceReferenceName. In the example of listing 5.20, the metric is generated by the operation of "getVariation" of service "Counting".

-iii- wsag:ServiceDescriptionTerm
/ocw:ServiceDescriptionTermExtension/ocw:Metric/ocw:ValueGenerateBy
/ocw:Schedule

Schedule describes the time granularity of gathering metric value from the operation of the service. In the example of listing 5.20, the metric value is collected every one hour.

Service Reference

```
<element name="ServiceReferenceExtension" type="ocw:
    ServiceReferenceExtensionType"/>
<complexType name="ServiceReferenceExtensionType">
  <sequence>
    <element name="WSDLLocation" type="anyURI"/>
    <element maxOccurs="unbounded" name="Operation" type="string"/>
    <any minOccurs="0" namespace="#any" processContents="strict"/>
  </sequence>
</complexType>
```

Listing 5.21. ServiceReference extension definition in PSLA.

In WS-Agreement, "ServiceReference" is designed to be the place to define domain specific reference to existing services. "ServiceReference" can refer to any kind of endpoint of the service as long as the reference can be understood by both party of the agreement. In PSLA, "ServiceReference" is extended as listing 5.21 to refer to an external Web Services Description Language (WSDL) file of the service and operations delivered by the service. The WSDL is an XML-based interface definition language that is used for describing the functionality offered by a web service [WIK15c].

1. wsag:ServiceReference
   /ocw:ServiceReferenceExtension/ocw:WSDLLocation

   "WSDLLocation" gives the location of the WSDL file of the service. In the example of listing 5.22, "http://www.opencloudware.org/Monitoring.wsdl" is the URL of WSDL file of monitoring service.

2. wsag:ServiceReference
   /ocw:ServiceReferenceExtension/ocw:Operation

   "Operation" lists all the related operations which will be mentioned or guaranteed in this agreement. For example, in listing 5.22, "getAllocatedInstructions", "getAllocated-Memory", "getAllocatedNetworkThroughput", "getAllocatedStorage", "getAvailability" and "getResponseTime" are listed for service "monitoring".

3. wsag:ServiceReference
   /ocw:ServiceReferenceExtension/{any}

   In PSLA, "ServiceReference" is still designed to be extendable to express all kinds of service references.

```
<wsag:ServiceReference Name="Monitoring" ServiceName="elasticity">
  <ocw:ServiceReferenceExtension>
    <ocw:WSDLLocation>http://www.opencloudware.org/Monitoring.xsdl</
   ocw:WSDLLocation>
    <ocw:Operation>getAllocatedInstructions</ocw:Operation>
    <ocw:Operation>getAllocatedMemory</ocw:Operation>
    <ocw:Operation>getAllocatedNetworkThroughput</ocw:Operation>
    <ocw:Operation>getAllocatedStorage</ocw:Operation>
    <ocw:Operation>getAvailability</ocw:Operation>
    <ocw:Operation>getResponseTime</ocw:Operation>
  </ocw:ServiceReferenceExtension>
</wsag:ServiceReference>
```

Listing 5.22. One ServiceReferenceExtension example of PSLA.

Service Properties

In WS-Agreement, ServiceProperties is designed to be the place for defining measurable and exposed properties of a service. The properties will be further used to express QoS requirements

in GuaranteeTerms 5.3.4. Service metrics, such as response time, should be described here according to WS-Agreement. However, it is not the mechanism for PSLA. In PSLA, metrics are expressed in "ServiceDescriptionTerms" because it provides flexible extension mechanism compared to “ServiceProperties”. In this way, "Metric" definition can be with more detailed information in PSLA.

5.3.4 Guarantee Terms

GuaranteeTerm 5.3.4 is designed for non-functional part of the agreement, including preconditions which are related to elasticity of workload, QoS target and related business clause.

In this section, the syntactic expression of Guarantee Term will be described. The structure of Guarantee Term will be described in section 5.3.4. The syntactic expression of Qualifying Condition will be described in section 5.3.4. The syntactic expression of Service Level Objectives will be described in section 5.3.4. The syntactic expression of Business Value List will be described in section 5.3.4.

Structure

As the listed example listing 5.23 demonstrated, the GuaranteeTerm is composed of ServiceScope, QualifyingCondition, ServiceLevelObjectives and BusinessValueList. QualifyingCondition and CustomServiceLevel are extended in PSLA.

```
<wsag:GuaranteeTerm Name="Request Response Time" Obligated="ServiceProvider
    ">
  <wsag:ServiceScope ServiceName="elasticity"/>
  <wsag:QualifyingCondition>
    <ocw:QualifyingConditionExtension>...</ocw:QualifyingConditionExtension
    >
  </wsag:QualifyingCondition>
  <wsag:ServiceLevelObjective>
    <wsag:CustomServiceLevel>
      <ocw:CustomServiceLevelExtension>...</ocw:CustomServiceLevelExtension
    >
    </wsag:CustomServiceLevel>
  </wsag:ServiceLevelObjective>
  <wsag:BusinessValueList>...</wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

Listing 5.23. GuaranteeTerm structure example in PSLA.

- ServiceScope indicates this GuaranteeTerm is about which services.
- QualifyingCondition declares this GuaranteeTerm becomes effective under which premises. QualifyingCondition usually constraints for the one who takes advantages from the GuaranteeTerm. QualifyingCondition is also open to be extended.
- ServiceLevelObjectives is open to be extended. We extend it to express the quantify expression of the requirement on QoS metrics.

- BusinessValueList describes the importance of this GuaranteeTerm from many aspects. BusinessValueList describes "Check Plan", "Priority" and "Business Values", including "Penalty" and "Reward". It is extendable.

  These informations can be helpful for PaaS provider to make a choice when the resources are not enough to meet all GuaranteeTerms. For example, to be helpful when making choices among GuaranteeTerms. "Check Plan" is expressed in Penalty and Reward in BusinessValueList.

We can see that WS-Agreement skeleton provides the commonly used clauses of web service based agreement. We extend it to meet domain specific needs. In PSLA, we extend WS-Agreement by considering the characteristic of PaaS level service and QoS mentioned before. Metric is expressed in SeviceDescriptionTerm extension. Elasticity workload is expressed in QualifyingCondition extension and ServiceLevelObjectives uses fuzzy value range to express the uncertain boundary of QoS target.

Qualifying Condition

In WS-Agreement, "QualifyingCondition" is designed to be the place of expressing a precondition under which a GuaranteeTerm should be taken into effect. "QualifyingCondition" is extendable.

In PSLA, "QualifyingCondition" is extended to describe under which conditions the obligate part of this GuaranteeTerm should ensure the QoS requirements. In another word, this GuaranteeTerm should be taken into consideration during runtime by the obligate part.

```
<element name="QualifyingConditionExtension" type="ocw:
    QualifyingConditionExtensionType"/>
<complexType name="QualifyingConditionExtensionType">
  <sequence>
    <element name="Periods" type="ocw:PeriodsType"/>
    <element name="Workloads" type="ocw:WorkloadsType"/>
    <element name="ScalabilityPoints" type="ocw:ScalabilityPointsType"/>
    <element minOccurs="0" name="AffinityConstraints" type="ocw:
    AffinityConstraintsType"/>
  </sequence>
</complexType>
```

Listing 5.24. QualifyingCondition definition in PSLA.

As shown in listing 5.24, there are four kinds of QualifyingConditions. They are "Periods", "Workloads", "ScalabilityPoints" and "AffinityConstraints".

1. wsag:QualifyingCondition
   /ocw:QualifyingConditionExtension/ocw:Periods

```
<complexType name="PeriodsType">
    <sequence>
            <element name="Period" type="ocw:PeriodType" maxOccurs="
    unbounded"/>
    </sequence>
</complexType>
<complexType name="PeriodType">
    <sequence>
```

```
            <element name="Start" type="dateTime"/>
            <element name="End" type="dateTime"/>
            <element name="Interspace" type="ocw:InterspaceType"
    minOccurs="0"/>
     </sequence>
</complexType>
<complexType name="InterspaceType">
     <sequence>
            <element name="TimeInterval" type="duration"/>
            <element name="Step" type="duration"/>
     </sequence>
</complexType>
```

Listing 5.25. Periods definition in QualifyingCondition extension of PSLA.

"Periods" is composed of a set of period which is dedicate to describe a time period. Each period is described by a start time point, and end time point and a time interval. Each time interval is defined through two time duration. They are "TimeInterval" and "Step". "TimeInterval" is the time duration of which the GuaranteeTerm should be effective. "Step" is the time duration between two "TimeInterval". "Step" presents the time which the current GuaranteeTerm should be not taken into effects. Listing 5.26 is an example of period defined according to PSLA. The period is from 01/01/2014 00:00:00 to 01/01/2015 00:00:00, everyday from 00:00:00 to 07:00:00. Value PT7H of TimeInterval means from the start time, valid time will last for 7 hours. Value PT17H of Step means there will always be a 17 hours time break between two TimeInterval.

```
<ocw:Periods>
  <ocw:Period>
    <ocw:Start>2014-01-01T00:00:00</ocw:Start>
    <ocw:End>2015-01-01T00:00:00</ocw:End>
    <ocw:InterSpace>
      <ocw:TimeInterval>PT7H</ocw:TimeInterval>
      <ocw:Step>PT17H</ocw:Step>
    <ocw:InterSpace>
  </ocw:Period>
</ocw:Periods>
```

Listing 5.26. One Period example of PSLA.

2. wsag:QualifyingCondition
/ocw:QualifyingConditionExtension/ocw:Workloads

```
<complexType name="WorkloadsType">
  <sequence>
    <element name="Workload" type="ocw:WorkloadType" maxOccurs="
    unbounded"/>
  </sequence>
</complexType>

<complexType name="WorkloadType">
  <sequence>
    <element name="InputDataName" type="IDREF"/>
    <element name="MetricName" type="IDREF"/>
    <element name="LowerBound" type="positiveInteger"/>
```

```
    <element name="UpperBound" type="positiveInteger"/>
    <element name="Average" type="positiveInteger"/>
    <element minOccurs="0" name="Variation" type="ocw:VariationType"/>
  </sequence>
</complexType>

<complexType name="VariationType">
  <sequence>
    <element name="MetricName" type="IDREF"/>
    <element name="Threshold" type="positiveInteger"/>
  </sequence>
</complexType>
```

Listing 5.27. Workloads definition in QualifyingCondition of PSLA.

As described in listing 5.27, Workloads is composed of a set of workloads. Each workload represents the composition of one kind of InputData which is defined in ServiceDescriptionTerms extension. For each workload, the range of the workload, the average value of the workload and the variation of the workload are defined. Workload is typically described by the metric of Arrival Rate. For example, the lower-bounder of requests arrival rate, upper-bounder of requests arrival rate, average requests arrival rate and variation condition of the requests arrival rate. The variation of the workload is defined by giving a positive integer as the threshold of the variation metric value. When the retrieved variation metric value is bigger than this threshold value, the current QualifyingCondition is no longer satisfied.

Listing 5.28 gives an example of workloads in QualifyingCondition of PSLA. In this example, workload is composed of two kinds of requests, Atomic_Request_A and Atomic_Request_B Atomic_Request_A's requests arrival rate is between 400 requests/s and 900 requests/s. The average requests arrival rate of Atomic_Request_A should be lower than 700 requests/s. The increasing of requests arrival rate should be lower than 10 requests/s for each hour. Atomic_Request_B is defined similarly.

```
<ocw:Workloads>
  <ocw:Workload>
    <ocw:InputDataName>Atomic_Request_A</ocw:InputDataName>
    <ocw:MetricName>RequestsThroughput</ocw:MetricName>
    <ocw:LowerBound>400</ocw:LowerBound>
    <ocw:UpperBound>900</ocw:UpperBound>
    <ocw:Average>700</ocw:Average>
    <ocw:Variation>
      <ocw:MetricName>Variation</ocw:MetricName>
      <ocw:Threshold>10</ocw:Threshold>
    </ocw:Variation>
  </ocw:Workload>
  <ocw:Workload>
    <ocw:InputDataName>Atomic_Request_B</ocw:InputDataName>
    <ocw:MetricName>RequestsThroughput</ocw:MetricName>
    <ocw:LowerBound>100</ocw:LowerBound>
    <ocw:UpperBound>500</ocw:UpperBound>
    <ocw:Average>300</ocw:Average>
    <ocw:Variation>
      <ocw:MetricName>Variation</ocw:MetricName>
      <ocw:Threshold>10</ocw:Threshold>
```

```
    </ocw:Variation>
  </ocw:Workload>
</ocw:Workloads>
```

Listing 5.28. One Workloads example of PSLA.

3. wsag:QualifyingCondition
   /ocw:QualifyingConditionExtension/ocw:ScalabilityPoints

```
<complexType name="ScalabilityPointsType">
    <choice maxOccurs="unbounded">
            <element name="ComputeScalabilityPoint" type="ocw:
    ComputeScalabilityPointType"/>
            <element name="StorageScalabilityPoint" type="ocw:
    StorageScalabilityPointType"/>
    </choice>
</complexType>
```

Listing 5.29. ScalabilityPoints definition in QualifyingCondition of PSLA.

"ScalabilityPoints" is the place to describe what scaling actions can be performed in order to ensure this GuaranteeTerm. As described in listing 5.29, scaling actions can either be performed on computing resources or storage resources. e.g. Bandwidth between Compute component and DataNode is 100 M/s. Memory size of NameNode is 4G. Upper-bound of memory usage of NameNode is 75 %.

   (a) wsag:QualifyingCondition
   /ocw:QualifyingConditionExtension/ocw:ScalabilityPoints
   /ocw:ComputeScalabilityPoint

```
<complexType name="ComputeScalabilityPointType">
    <sequence>
            <element name="ResourceName" type="IDREF"/>
            <element minOccurs="0" name="HorizontalScaling" type="
    ocw:HorizontalScalingType"/>
            <element minOccurs="0" name="ComputeVerticalScaling"
    type="ocw:ComputeVerticalScalingType"/>
    </sequence>
</complexType>
```

Listing 5.30. ComputeScalabilityPoint definition in ScalabilityPoints extension of PSLA.

```
<ocw:ComputeScalabilityPoint>
  <ocw:ResourceName>Apache</ocw:ResourceName>
  <ocw:HorizontalScaling>...</ocw:HorizontalScaling>
  <ocw:ComputeVerticalScaling>...</ocw:ComputeVerticalScaling>
</ocw:ComputeScalabilityPoint>
```

Listing 5.31. ComputeScalabilityPoint example in PSLA.

According to ComputeScalabilityPoint definition listing 5.30, to define a "ComputeScalabilityPoint", the computing resources and allowed scaling actions, including horizontal scaling and vertical scaling, can both be defined. For example, as listing 5.31, both horizontal scaling and vertical scaling actions are defined and can be performed on Apache server.

(b) wsag:QualifyingCondition
/ocw:QualifyingConditionExtension/ocw:ScalabilityPoints
/ocw:StorageScalabilityPoint

```
<complexType name="StorageScalabilityPointType">
    <sequence>
            <element name="ResourceName" type="IDREF"/>
            <element minOccurs="0" name="HorizontalScaling" type="
   ocw:HorizontalScalingType"/>
            <element minOccurs="0" name="StorageVerticalScaling"
   type="ocw:StorageVerticalScalingType"/>
    </sequence>
</complexType>
```

Listing 5.32. StorageScalabilityPoint definition in ScalabilityPoints in QualifyingCondition extension of PSLA.

```
<ocw:StorageScalabilityPoint>
  <ocw:ResourceName>appDiskID2</ocw:ResourceName>
  <ocw:HorizontalScaling>...</ocw:HorizontalScaling>
  <ocw:StorageVerticalScaling>...</ocw:StorageVerticalScaling>
</ocw:StorageScalabilityPoint>
```

Listing 5.33. StorageScalabilityPoint example of PSLA.

As defined in listing 5.32, similar to the definition of "ComputeScalabilityPoint", the storage resources and corresponding scaling actions can be defined. As in the example of listing 5.33, there are two kinds of scaling actions, horizontal scaling and vertical scaling, can be performed on storage of appDiskID2.

(c) wsag:QualifyingCondition
/ocw:QualifyingConditionExtension/ocw:ScalabilityPoints
/ocw:ComputeScalabilityPoint|ocw:StorageScalabilityPoint
/ocw:HorizontalScaling

Horizontal scaling for both computing resources and storage resources are defined in the same way. The definition of HorizontalScaling is given in listing 5.34.

```
<complexType name="HorizontalScalingType">
    <sequence>
            <element name="LowerBound" type="positiveInteger"/>
            <element name="UpperBound" type="ocw:UpperBoundType"/>
    </sequence>
</complexType>
<simpleType name="UpperBoundType">
    <union memberTypes="positiveInteger">
        <simpleType>
            <restriction base="string">
                <enumeration value="unbounded"/>
            </restriction>
        </simpleType>
    </union>
</simpleType>
```

Listing 5.34. HorizontalScaling definition in ScalabilityPoints extension of PSLA.

In listing 5.34, one positive integer is given as the lower bound of resource duplication. Another positive integer or "unbounded" is given as the upper-bounder of resource duplication.

```
<ocw:HorizontalScaling>
  <ocw:LowerBound>1</ocw:LowerBound>
  <ocw:UpperBound>unbounded</ocw:UpperBound>
</ocw:HorizontalScaling>
```

Listing 5.35. HorizontalScaling example in PSLA.

(d) wsag:QualifyingCondition
/ocw:QualifyingConditionExtension/ocw:ScalabilityPoints
/ocw:ComputeScalabilityPoint/ocw:ComputeVerticalScaling

```
<complexType name="ComputeVerticalScalingType">
    <sequence>
        <element maxOccurs="unbounded" name="Set" type="ocw:
    ComputeSetType"/>
    </sequence>
</complexType>
<complexType name="ComputeSetType">
    <sequence>
        <element name="CPU" type="ocw:LowerBoundType"/>
        <element name="RAM" type="ocw:LowerBoundType"/>
        <element maxOccurs="unbounded" minOccurs="0" name="
    NetworkThroughput" type="ocw:NetworkThroughputType"/>
    </sequence>
</complexType>
<complexType name="LowerBoundType">
    <sequence>
            <element name="MetricName" type="IDREF"/>
            <element name="LowerBound" type="positiveInteger"/>
    </sequence>
</complexType>
<complexType name="NetworkThroughputType">
    <sequence>
            <element name="MetricName" type="IDREF"/>
            <element name="UploadSpeed" type="positiveInteger"/>
            <element name="DownloadSpeed" type="positiveInteger"/>
            <element name="ResourceName" type="IDREF"/>
    </sequence>
</complexType>
```

Listing 5.36. ComputeVerticalScaling definition in ScalabilityPoints extension of PSLA.

Vertical scaling for computing resources and storage resources are defined differently. As defined in listing 5.36, ComputeVerticalScaling is composed of several sets of VM configuration descriptions. In each VM configuration, the requirements of CPU, memory and network bandwidth should be described.

The CPU or RAM requirements are defined according to LowerBoundType which has a Metric and a positive integer to describe the suitable value of the metric. Both the "UploadSpeed" and "DownloadSpeed" of the network are defined as the network configuration requirements for vertical scaling of computing resources.

To unambiguously express network throughput requirements, related resources should be defined in ServiceDescriptionTerm, and be referenced.

```
<ocw:ComputeVerticalScaling>
  <ocw:Set>
    <ocw:CPU>
      <ocw:MetricName>Instructions</ocw:MetricName>
      <ocw:LowerBound>20</ocw:LowerBound>
    </ocw:CPU>
    <ocw:RAM>
      <ocw:MetricName>Memory</ocw:MetricName>
      <ocw:LowerBound>1024</ocw:LowerBound>
    </ocw:RAM>
    <ocw:NetworkThroughput>
      <ocw:MetricName>NetworkThroughput</ocw:MetricName>
      <ocw:UploadSpeed>10</ocw:UploadSpeed>
      <ocw:DownloadSpeed>10</ocw:DownloadSpeed>
      <ocw:ResourceName>appDiskId2</ocw:ResourceName>
    </ocw:NetworkThroughput>
  </ocw:Set>
  <ocw:Set>
    <ocw:CPU>
      <ocw:MetricName>Instructions</ocw:MetricName>
      <ocw:LowerBound>40</ocw:LowerBound>
    </ocw:CPU>
    <ocw:RAM>
      <ocw:MetricName>Memory</ocw:MetricName>
      <ocw:LowerBound>2048</ocw:LowerBound>
    </ocw:RAM>
    <ocw:NetworkThroughput>
      <ocw:MetricName>NetworkThroughput</ocw:MetricName>
      <ocw:UploadSpeed>10</ocw:UploadSpeed>
      <ocw:DownloadSpeed>10</ocw:DownloadSpeed>
      <ocw:ResourceName>appDiskId2</ocw:ResourceName>
    </ocw:NetworkThroughput>
  </ocw:Set>
</ocw:ComputeVerticalScaling>
```

Listing 5.37. ComputeVerticalScaling example in PSLA.

(e) wsag:QualifyingCondition
/ocw:QualifyingConditionExtension/ocw:ScalabilityPoints
/ocw:StorageScalabilityPoint/ocw:StorageVerticalScaling

```
<complexType name="StorageVerticalScalingType">
    <sequence>
        <element maxOccurs="unbounded" name="Set" type="ocw:
    StorageSetType"/>
    </sequence>
</complexType>
<complexType name="StorageSetType">
    <sequence>
        <element name="Size" type="ocw:LowerBoundType"/>
        <element name="DiskThroughput" type="ocw:
    DiskThroughputType"/>
    </sequence>
```

```
</complexType>
<complexType name="DiskThroughputType">
    <sequence>
            <element name="MetricName" type="IDREF"/>
            <element name="ReadSpeed" type="positiveInteger"/>
            <element name="WriteSpeed" type="positiveInteger"/>
    </sequence>
</complexType>
```

Listing 5.38. StorageVerticalScaling definition in ScalabilityPoints extension of PSLA.

As defined in listing 5.38, StorageVerticalScaling is composed of several sets of storage configuration descriptions. In each Storage configuration description, the storage size and the disk throughput requirements should be given. Disk throughput includes both read speed and write speed of the storage. The storage size requirement is defined according to LowerBoundType which has a metric and a positive integer to describe the suitable value of the metric. Both the "ReadSpeed" and "WriteSpeed" of the Disk throughput are defined for storage vertical scaling. Listing 5.39 gives an example of expressing the vertical scaling can be performed.

```
<ocw:StorageVerticalScaling>
  <ocw:Set>
    <ocw:Size>
      <ocw:MetricName>Storage</ocw:MetricName>
      <ocw:LowerBound>1</ocw:LowerBound>
    </ocw:Size>
    <ocw:DiskThroughput>
      <ocw:MetricName>NetworkThroughput</ocw:MetricName>
      <ocw:ReadSpeed>6</ocw:ReadSpeed>
      <ocw:WriteSpeed>6</ocw:WriteSpeed>
    </ocw:DiskThroughput>
  </ocw:Set>
  <ocw:Set>
    <ocw:Size>
      <ocw:MetricName>Storage</ocw:MetricName>
      <ocw:LowerBound>2</ocw:LowerBound>
    </ocw:Size>
    <ocw:DiskThroughput>
      <ocw:MetricName>NetworkThroughput</ocw:MetricName>
      <ocw:ReadSpeed>6</ocw:ReadSpeed>
      <ocw:WriteSpeed>6</ocw:WriteSpeed>
    </ocw:DiskThroughput>
  </ocw:Set>
</ocw:StorageVerticalScaling>
```

Listing 5.39. Storage Vertical Scaling example in PSLA.

4. wsag:QualifyingCondition
   /ocw:QualifyingConditionExtension/ocw:AffinityConstraints

```
<complexType name="AffinityConstraintsType">
    <sequence>
        <element name="AffinityConstraint" type="ocw:
    AffinityConstraintType" maxOccurs="unbounded"/>
```

```
    </sequence>
</complexType>
<complexType name="AffinityConstraintType">
    <sequence>
            <element maxOccurs="unbounded" name="ResourceName" type="
    IDREF"/>
            <element maxOccurs="unbounded" name="Constraint" type="ocw
    :ConstraintType"/>
    </sequence>
</complexType>
<simpleType name="ConstraintType">
    <restriction base="string">
            <enumeration value="Host Affinity"/>
            <enumeration value="Host Anti-Affinity"/>
            <enumeration value="Rack Affinity"/>
            <enumeration value="Rack Anti-Affinity"/>
            <enumeration value="Data Center Affinity"/>
            <enumeration value="Data Center Anti-Affinity"/>
            <enumeration value="Lonely"/>
    </restriction>
</simpleType>
```

Listing 5.40. AffinityConstraints definition in QualifyingCondition extension of PSLA.

"AffinityConstraints" is the place to define the deployment location requirements. As defined in listing 5.40, in one "AffinityConstraints", one or more location constraints can be set on one ore more resources. In PSLA, 7 kinds on location constraints are provided. They are "Host Affinity", "Host Anti-Affinity", "Rack Affinity", "Rack Anti-Affinity", "Data Center Affinity" and "Data Center Anti-Affinity".

- "Host Affinity" means relevant resources should be deployed in the same host.
- "Host Anti-Affinity" means relevant resources should be deployed in the different host.
- "Rack Affinity" means relevant resources should be deployed in the same rack.
- "Rack Anti-Affinity" means relevant resources should be deployed in different rack.
- "Data Center Affinity" means relevant resources should be deployed in the same data center.
- "Data Center Anti-Affinity" means relevant resources should be deployed in different data center.

```
<ocw:AffinityConstraints>
  <ocw:AffinityConstraint>
    <ocw:ResourceName>Apache</ocw:ResourceName>
    <ocw:Constraint>Host Affinity</ocw:Constraint>
  </ocw:AffinityConstraint>
  <ocw:AffinityConstraint>
    <ocw:ResourceName>Jonas</ocw:ResourceName>
    <ocw:Constraint>Data Center Affinity</ocw:Constraint>
  </ocw:AffinityConstraint>
  <ocw:AffinityConstraint>
    <ocw:ResourceName>Apache</ocw:ResourceName>
```

```
    <ocw:ResourceName>Jonas</ocw:ResourceName>
    <ocw:Constraint>Data Center Affinity</ocw:Constraint>
    <ocw:Constraint>Host Anti-Affinity</ocw:Constraint>
  </ocw:AffinityConstraint>
  <ocw:AffinityConstraint>
    <ocw:ResourceName>MySQL</ocw:ResourceName>
    <ocw:Constraint>Lonely</ocw:Constraint>
  </ocw:AffinityConstraint>
</ocw:AffinityConstraints>
```

Listing 5.41. AffinityConstraints example in PSLA.

QualifyingCondition is mainly used to express the condition that this GuaranteeTerm will become effective under which condition. If we put all the QualifyingConditions in this AgreementOffer together, it will be a panorama of the elasticity of workload 5.2.2. We extend QualifyingCondition to describe "Valid Period" and "Valid Workload". "Valid Period" includes "Start Time", "End Time" and "Time Interval". "Valid Workload" includes "Data Size", "Data Composition" and "Location".

Service Level Objectives

```
<element name="CustomServiceLevelExtension" type="ocw:
    CustomServiceLevelExtensionType"/>
<complexType name="CustomServiceLevelExtensionType">
    <group ref="ocw:ExpressionGroup"></group>
</complexType>
<group name="ExpressionGroup">
    <choice>
        <group ref="ocw:LogicExpressionGroup"/>
        <group ref="ocw:ComparisonExpressionGroup"/>
    </choice>
</group>
```

Listing 5.42. CustomServiceLevel extension definition in PSLA.

In WS-Agreement, "CustomServiceLevel" can be extended to express customised SLO. In PSLA, we benifit from this extension mechanism to express SLO more precisely. As defined in listing 5.42 in PSLA, the expression of SLO is based on logic expression and comparison expression.

1. Logic Expression and Comparison Expression

```
<group name="LogicExpressionGroup">
  <choice>
    <element name="And" type="ocw:BinaryLogicOperatorType"/>
    <element name="Or" type="ocw:BinaryLogicOperatorType"/>
    <element name="Not" type="ocw:UnaryLogicOperatorType"/>
  </choice>
</group>
<group name="ComparisonExpressionGroup">
  <choice>
    <element name="Greater" type="ocw:PredicateType"/>
    <element name="Less" type="ocw:PredicateType"/>
```

```
    <element name="Equal" type="ocw:PredicateType"/>
    <element name="GreaterOrEqual" type="ocw:PredicateType"/>
    <element name="LessOrEqual" type="ocw:PredicateType"/>
  </choice>
</group>
<complexType name="UnaryLogicOperatorType">
  <group ref="ocw:ExpressionGroup"/>
</complexType>

<complexType name="BinaryLogicOperatorType">
  <group maxOccurs="2" minOccurs="2" ref="ocw:ExpressionGroup"/>
</complexType>
```

Listing 5.43. LogicExpressionGroup and ComparisonExpressionGroup definition in PSLA.

Listing 5.43 gives the definitions of logic expression and comparison expression used in SLO of PSLA. According to listing 5.43, three kinds of logical operators, including "And", "Or" and "Not", are supported in "PSLA". Five kinds of comparison operators, including "Greater", "Less", "Equal", "GreaterOrEqual" and "LessOrEqual", are supported in "PSLA".

```
<wsag:ServiceLevelObjective>
  <CustomServiceLevel>
    <ocw:CustomServiceLevelExtension>
      <ocw:And>
        <ocw:LessOrEqual>
          <ocw:MetricName>ResponseTime</ocw:MetricName>
          <ocw:Threshold>...</ocw:Threshold>
        </ocw:LessOrEqual>
        <ocw:GreaterOrEqual>
          <ocw:MetricName>Availability</ocw:MetricName>
          <ocw:Threshold>...</ocw:Threshold>
        </ocw:GreaterOrEqual>
      </ocw:And>
    </ocw:CustomServiceLevelExtension>
  </CustomServiceLevel>
</wsag:ServiceLevelObjective>
```

Listing 5.44. ServiceLevelObjective example of PSLA.

Listing 5.44 is an example of ServiceLevelObjective in PSLA. This SLO expresses the requirement of ResponseTime should be lower than or equals to a given threshold and Availability should be bigger than or equals to a specific threshold.

2. Fuzzy threshold expression

```
<complexType name="PredicateType">
    <sequence>
        <element name="MetricName" type="IDREF"/>
        <element name="Threshold" type="ocw:ThresholdType"/>
    </sequence>
</complexType>
<complexType name="ThresholdType">
    <sequence>
        <element name="Value" type="float"/>
```

```
        <element name="Confidence" type="ocw:ConfidenceType" minOccurs
    ="0"/>
    </sequence>
</complexType>
<complexType name="ConfidenceType">
    <sequence>
        <element name="Trust" type="ocw:Trust"/>
        <element name="Fuzziness" type="ocw:Fuzziness"/>
    </sequence>
</complexType>
<simpleType name="Trust">
    <restriction base="float">
        <maxInclusive value="1.00"/>
        <minInclusive value="0.00"/>
    </restriction>
</simpleType>
<simpleType name="Fuzziness">
    <restriction base="float">
        <maxInclusive value="1.00"/>
        <minInclusive value="0.00"/>
    </restriction>
</simpleType>
```

Listing 5.45. PredictionType definition in PSLA.

"ComparisonExpression" is of "PredicateType". As defined in listing 5.45, besides the comparison operator implied by the name of element, each comparison expression should also contain the related metrics and corresponding threshold. The threshold is composed of a float value and the confidence of this threshold value.

Confidence represents the stringency of the threshold value. The stringency is depicted from two aspects. They are "Trust" and "Fuzziness".

"Trust" and "Fuzziness" are both float value between 0.00 and 1.00. "Trust" represents how many percent of the value of comparison expression should be within the range specified by threshold. "Fuzziness" represents how many percent of the deviation between real metric value and threshold if the real metric value is not within the range specified by threshold.

When "Trust" is 1.00 or "Fuzziness" is 0.00, it means no deviation is allowed which means judging the SLO violation strictly according to comparison expression.

When "Trust" is 0.00, it means any metric value will be accepted and no SLO violation will be raised.

However, when "Fuzziness" is 1.00, it means the acceptable metric value range is "Greater" or "GreaterOrEqual" to 0 or "Less" or "LessOrEqual" to two times of threshold value.

```
<ocw:LessOrEqual>
  <ocw:MetricName>ResponseTime</ocw:MetricName>
  <ocw:Threshold>
    <ocw:Value>1.0</ocw:Value>
    <ocw:Confidence>
      <ocw:Trust>0.9</ocw:Trust>
      <ocw:Fuzziness>0.5</ocw:Fuzziness>
    </ocw:Confidence>
```

```
  </ocw:Threshold>
</ocw:LessOrEqual>
```

Listing 5.46. ComparisonExpression example of PSLA.

Listing 5.46 gives an example of ComparisonExpression in PSLA. In this example, at least 90% of the ResponseTime should be lower or equals to 1.5.

Business Value List

```
<wsag:BusinessValueList>
    <wsag:Importance>2</wsag:Importance>
    <wsag:Penalty>
        <wsag:AssessmentInterval>
            <wsag:TimeInterval>PT1H</wsag:TimeInterval>
        </wsag:AssessmentInterval>
        <wsag:ValueUnit>EURO</wsag:ValueUnit>
        <wsag:ValueExpression>2</wsag:ValueExpression>
    </wsag:Penalty>
    <wsag:Reward>
        <wsag:AssessmentInterval>
            <wsag:TimeInterval>PT1H</wsag:TimeInterval>
        </wsag:AssessmentInterval>
        <wsag:ValueUnit>EURO</wsag:ValueUnit>
        <wsag:ValueExpression>3</wsag:ValueExpression>
    </wsag:Reward>
</wsag:BusinessValueList>
```

Listing 5.47. Business Value List example in PSLA.

In WS-Agreement, BusinessValueList is defigned to be the place for expressing business related terms. Listing 5.47 gives an example of BusinessValueList in PSLA. In this example, if it is abide by OpenCloudware, 3 USD will be gained. If it is violated, 2 USD will be charged. This term should be checked once per hour. The priority of this term is 2.

5.4 Conclusion

SLA, as a contract which specifies the service and QoS criterion, is indispensable for service oriented business. In PaaS, the QoS of SaaS and elasticity of workload are cooperated by PaaS provider. So it is necessary to consider the elasticity and SaaS QoS target characters in PaaS level SLA. In this chapter, we propose PSLA. It is a generic PaaS level SLA structured language based on WS-Agreement skeleton. PSLA is implemented in XML. PSLA is adopted by OpenCloudware [ocw] project which aims at building a PaaS platform to manage application running over multiple IaaSs during lifecycle. The XML schema of PSLA is available. The XML based PSLA example described in this chapter is transformed through our XSLT based tool and displayed in "http://www.polytech.univ-savoie.fr/opencloudware/".

Chapter 6

Benchmarking Based SLA Feasibility study

6.1 Introduction

Service Level Agreements (SLA) are contracts established between Software as a Service (SaaS) providers and Platform as a Service (PaaS) providers. SLA is related to properties of the services running in the platform. We propose a generic method to evaluate SaaS provider proposed Quality of Service (QoS) targets, such as response time or maximal throughput, can be guaranteed with constraints on workload, resources and cost or not. These "what-if" evaluations are based on small scale benchmarking and ability of the application to be scaled.

SaaS providers rely on PaaS features to benefit from the " pay as you go" paradigm corresponding to pay only " adapted" resources usage at runtime while satisfying a set of constraints related to QoS, workload and cost. Elasticity is the property of the platform to meet all these requirements at a given variation rate. SLA are used to describe the responsibilities of contracting parties (SaaS provider and PaaS provider). Existing SLA in cloud industry is not customizable, machine readable or formally constructed which limit the automated QoS management. SLA management involves two stages: establishing a SLA before runtime, and fulfilling the SLA at runtime. Establishing SLA is usually done through an automated negotiation process which consists several turns of SLA feasibility study. It involves benchmarking and/or modelling the system (application and runtime support) to define the levels of QoS accepted by the SaaS "client" and offered by the PaaS provider and the constraints that both parties will fulfil. Benchmarking is an approach of evaluating system performance before runtime. It is necessary to derive an universal unified benchmark processes to collect information for automating the SLA feasibility process and exploring runtime SLA oriented resource management strategy. The limited knowledge about application behaviour, non-linearity and the willing of reducing SLA evaluation cost makes this problem challenging. Creating SLA before runtime means formally construct the SLA contract through SLA feasibility studies. A successful SLA management means the constructed SLA can meet the client's original needs (initially proposed at the beginning of the negotiation) at the most extent, and the SLA can be fulfilled during runtime. SLA feasibility study evaluates whether the QoS target can be fulfilled with permitted resources underling the qualified workload or not. SLA feasibility study will be repeatedly applied during the whole negotiation process. Modelling arrival rate and resource demands when satisfying QoS target are foundations of capacity evaluation. This modelling can either

be achieved through simulation modelling or analytic modelling. Since we want to design a non intrusive method with respect to the application, our model will be based on results of benchmarks running in the target PaaS context, using measures also available at runtime. In this chapter, we propose an application independent, cost effective, benchmarking based, SLA feasibility study method from the PaaS provider's perspective.

In this chapter, our SLA feasibility study method is developed and formally described compared to our previous publication [LPM15]. The rest of the chapter is as follows. The context of SLA feasibility study will be introduced in section 6.2. The benchmarking based activities described in section 6.3.1, 6.3.2, 6.3.3, 6.3.4 and 6.3.5, which devote to explore Maximum Flavor Capability modeling will be introduced in section 6.3. Based on the Maximum Flavor Capability model explored in section 6.3, the constraints described in section 6.2.1 can be evaluated and the evaluation results will lead to reject or accept of the SLA contract. These constraints evaluation processes will be introduced in section 6.4. To make our benchmark based SLA feasibility study method understandable, we will give an example in section 6.5. This chapter will be concluded in section 6.6.

6.2 SLA feasibility study context

SLA feasibility study is based on the evaluations of elasticity constraints described in PSLA. In this section, we discuss constraints clarified in PSLA terms, then we give the example.

In this section, the important concepts will be introduced and discussed. The goals and challenges of PaaS level SLA feasibility study will be explained. What kind of problems are involved in a SLA feasibility study will be explained in section 6.2.1. The evaluation of size of flavor will be discussed in section 6.2.2. The stage dependent evaluation of size of flavor will be discussed in section 6.2.2. The stage independent evaluation of size of flavor will be discussed in section 6.2.2. Maximum Flavor Capability, which is the core concept of our SLA feasibility study methods, will be explained and discussed in section 6.2.3.

6.2.1 Problem description

SLA is a contract between service provider and service consumer to formally describe the rights and obligations between two contract parties to satisfy the SLA. SLA feasibility study aims at evaluating one proposed SLA offer is acceptable or not. The evaluation will take all constraints defined in SLA offer into consideration and check whether there are conflicts existing in the proposed SLA offer.

The difficulties of SLA feasibility study is how we can do feasibility study with a low cost to save both money and time. Thinking about that, the highest arrival rate of the workload can be served with huge amount of resources. It is costly to do benchmark on such a large scale VMs. So, we developed a feasibility study processes to reduce the cost as much as possible.

As described in previous section, in PSLA, the core part is GuaranteeTerm. In GuaranteeTerm, QualifyingCondition is the place where preconditions of QoS targets are defined. ServiceLevelObjectives is the place to define QoS targets such as response time<3 s. BusinessValueList gives billing informations such as cost constraints. In the context of PSLA, the goal of SLA feasibility study is to evaluate, with the resource constraints defined in QualifyingCondition and cost constraints defined in BusinessValueList, if the possible workload range defined

in QualifyingCondition can be served with QoS requirements defined in ServiceLevelObjective or not.

Here we will list some formally described terms which can be found in SLA. These terms will be used in rest of this chapter to precisely express our benchmark based SLA feasibility study method and to make our method visual.

- $STAGE = \{s \mid s\ is\ the\ index\ of\ stages\ of\ a\ VCAA\}$
  - $STAGE$ is the set of stages of a VCAA.
  - s is the index of the stage of a VCAA.
- Number of instances defined in SLA
  - $MaxNoI_s^f$: the maximum number of instances allowed according to SLA when adopting flavor f on stage s. For the same stage, different flavors could have different $MaxNoI_s^f$. For example, the maximum cost for maintain the QoS is given in SLA and the bigger the flavor is, the more expensive the tariff is. When adopting the bigger flavors, the available maximum number of instances is smaller than the others.
  - $MinNoI_s$: the minimum number of instances allowed according to SLA for stage s. We consider that the constraints of minimum number of instances is probably a constraints from technical point of view. Therefore, we assume that the minimum number of instances when adopting different flavors on one stage are the same.
- Available flavors defined in SLA
  - $FLAVOR_s$: the set of available flavors for stage s according to SLA.
  - $card(FLAVOR_s)$: the number of available flavors for stage s according to SLA.
  - $flavor_s^i = (memory_s^i, cpu_s^i) i = 1, 2, ..., card(FLAVOR_s)$
  - $flavor_s^i$ is the ith available flavor for stage s.
  - $flavor_s^i$ has a configuration of $memory_s^i$ gigabyte of memory and cpu_s^i vCPU.

6.2.2 Size of flavor

One flavor has many different kinds of resources. We mainly consider CPU and memory resources. Therefore, which flavor is bigger is a multiple criteria problems. In another word, we can only say one flavor is bigger (or smaller) than another when both CPU and memory resources are more (or smaller) than the other flavor. Otherwise, for example, CPU resources is more than the other one, while memory resource is less than the other one, it is difficult to say one flavor is more powerful than another and the performances of adopting different flavors are depending on which resources are demanded more. Therefore, one flavor is bigger than another is an stage dependent assertion. However, before enough benchmarks are performed, no stage dependent comparision of flavors can be performed. Flavors can only be compared according to their configurations. Therefore, we also introduce a stage independent flavor comparision mechanism to support the process of exploring stage dependent FLAVOR comparision. It is worth mentioning that the performance can be different because of hardware architecture. As a simplification of the problem, we ignore these performance differences caused by hardware architecture and compare flavors from the IaaS consumer's perspective.

Stage dependent flavor comparision

The comparision of flavors should be performed separately for each stage. MFC, the upper-bound of capable request arrival rate when adopting one instance of this flavor on current stage, can be seen as a measurement of the flavor. We say that for a specific stage, one flavor is smaller (or bigger) than another when its MFC is smaller(or bigger). MFC values are supposed to be achieved through benchmarks. However, in order to save the cost of SLA feasibility study, only limited amount of benchmarks could be performed and some of the MFCs should be deduced from other MFCs achieved through benchmark. The deduction is performed according to the configured amount of resources and the benchmarked MFC with saturated flavor instances. It is necessary to get the benchmarked flavor saturated to achieve believable MFC deduction. It is because that our deduction will take the minimum ratio of resources as the ratio of maximum capable arrival rate. However, if the minimum ratio of resources is on the unsaturated resource, it is not sufficient to say that the maximum capable arrival rate will be smaller. For example, $flavor_a$ has 2 vCPU and 4 G memory. 80 % of vCPU but only 10% of memory are used during benchmark which means probably 0.5 G memory is enough by taking 80% memory utilization threshold into consideration. $flavor_b$ has 4 vCPU and 2 G memory. The minimum resource ratio is 0.5 because of memory. However, 2G memory probably is enough for this flavor to get the 4 vCPU saturated on this stage since only 0.5 G memory is enough for 2 vCPU. Therefore, $flavor_b$ should be able to serve bigger request arrival rate and its MFC should be bigger too. That is the reason of trying to select one suitable flavor for each stage which can get every resources saturated during the benchmark process for MFC involved in step 3.

After found the suitable flavors during step 3, we can compare flavors and say one flavor is bigger (or smaller) than another one. We will take the minimum ratio of resources between one flavor and the benchmarked and saturated one. For one stage, the lower the minimum ratio value is, the smaller the flavor is . If the minimum ratio value is smaller than 1, the flavor is seen as a flavor smaller than the saturated and benchmarked one. If the minimum ratio value is bigger than 1, the flavor is seen as a flavor bigger than the saturated and benchmarked one. The flavor with the biggest minimum ratio value is seen as the biggest flavor for this stage.

Not every available flavor can be saturated on stages. Also the biggest flavor may not be able to be saturated during the benchmark of step 5. The unsaturated benchmarked MFC of biggest flavor will also be used to deduce the smaller flavors. However the ratio used during deduction is still the one achieved by comparing with the saturated benchmarked MFC of flavors during step 3. Therefore, the aforementioned situation is not a problem.

Stage independent flavor comparision

This section will introduce Stage independent comparision of flavors, which are the comparision of flavors from a pure hardware point of view. We firstly list the important formally expressions with will be used in this section.

- Ratio
  - $Ratio_s^i = \frac{memory_s^i}{cpu_s^i}$
  - $Ratio_s^i$ is the amount of memory resources allocated per CPU unit for $flavor_s^i$.
- Distance

- $Distance_s^i(0,0) = \sqrt{{memory_s^i}^2 + {cpu_s^i}^2}$
  - $Distance_s^i(0,0)$ is the configuration gap between $flavor_s^i$ and flavor with no CPU and memory resources.
  - When two flavors have the same $Ratio_s^i$ values, $Distance_s^i(0,0)$ can be used to indicate the size of flavors.
- $Distance_s^i(mem, cpu) = \sqrt{(memory_s^i - mem)^2 + (cpu_s^i - cpu)^2}$
  - $Distance_s^i(mem, cpu)$ is the configuration gap between $flavor_s^i$ and flavor with cpu vCPU and mem gigabyte of memory resources.

Stage independent comparision of flavor compares flavors from a pure hardware configuration point of view. We introduce $Distance_s^i(mem, cpu)$ as a quantitative description of the size differences between $flavor_s^i$ and the flavor with a configuration of "mem" memory and "cpu" CPU. The smaller $Distance_s^i(mem, cpu)$ is, the more similar $flavor_s^i$ and the flavor with "mem" memory and "cpu" CPU are. When $Distance_s^i(mem, cpu)$ is 0, $flavor_s^i$ has the same configuration as the flavor with "mem" memory and "cpu" CPU.

$Distance_s^i(0,0)$ is a quantitative description of the size differences between $flavor_s^i$ and the flavor with a configuration of 0 memory and 0 CPU. $Distance_s^i(0,0)$ can be seen as a description of the size of $flavor_s^i$. We say $flavor_s^i$ is smaller than another when the $Distance_s^i(0,0)$ is smaller.

$Distance_s^i(0,0)$ is not sufficient to describe the ability of $flavor_s^i$. For example, the flavor with 10 vCPU and 1 G memory has the same Distance value as the flavor with 1 vCPU and 10 G memory but the ability of these two flavor may totally different. Therefore, $Ratio_s^i$ is introduced as a description of the resource configuration proportion of $flavor_s^i$.

$Ratio_s^i = \frac{memory_s^i}{cpu_s^i}$

When $Ratio_s^i$ is higher than another one, we can say one flavor allocated more memory resources for each unit of CPU. We can also say that two flavors have been allocated the same amount of memory resources for each CPU unit when two flavors have the same ratio value. In this case, the flavor with bigger Distance value is more powerful than another flavor.

$Ratio_s^i$ can be used as the criteria of judging the allocated amount of memory per CPU unit is high, medium or low. One flavor configuration can be chosen as the Balanced resource configuration. "Balanced resource configuration" means that one allocated resource, for example. CPU or memory, is not markedly more or less than another one. There is no standard configuration which can be seen as balanced. The amount of CPU and amount of memory have different units of measurement. Therefore, it is also difficult to specify one Ratio value to represent the state of "balanced resource configuration" . We take the flavor with median value of this ratio among all flavors available for one stage as the most balanced resource configuration for this stage. When we calculate the median value, instead of taking several same ratio value into consideration, one ratio value is counted only once to avoid that many unbalanced ratios lead to biased median value.

$Ratio_s^i$ together with $Distance_s^i(0,0)$ describe the resource allocation proportion and allocated amount of resources for $flavor_s^i$. $Ratio_s^i$ and $Distance_s^i(0,0)$ merge the configuration description of different resources together. $Ratio_s^i$ and $Distance_s^i(0,0)$ are used before the stage dependent comparision of flavors are available. $Ratio_s^i$ and $Distance_s^i(0,0)$ are not sufficient to say that which flavor is more powerful and we can not quantitatively describe how

powerful one flavor is for this stage. After a series of benchmark, we can quantitatively describe the ability of one flavor for each stage by MFC.

6.2.3 Maximum Flavor Capability

This section will introduce Maximum Flavor Capability (MFC), which is the core of this benchmark based SLA feasibility study method. We firstly list the important formally expressions with will be used in this section.

- MFC_s^f: the upper-bound of capable request arrival rate for one instance of flavor f on stage s. Capable arrival rate means that when this amount of requests are loaded on this instance, the QoS target still can be ensured.
- $MFC_s^f(n)$: the upper-bound of capable request arrival rate for n instances of flavor f on stage s.
- $MFC_s^f(n) = n * MFC_s^f$
- $MFC_s^f = MFC_s^f(n)/n$

Maximum Flavor Capability (MFC_s^i) is the upper-bound of capable arrival rate when adopting one instance of a specific flavor i on a specific stage s. MFC_s^i describes the ability for serving requests of flavor i used on stage s. MFC_s^i represents how many requests can be served when using flavor i on stage s. The unit of MFC_s^i is requests/s. The number of requests is calculated at the entrance of the application. The requests are not the one comes from one stage to another but the one comes from external to the entrance of the whole application.

MFC is not a concept which can be meaningful by only attached to one flavor, instead, stage should also be specified. For example, when one flavor can be used on two different stages, the capable arrival rate could be probably different since serving the same request may require different kinds and amount of resources on two different stages.

MFC_s^i will change when the QoS requirements changes. For example, for the same kind and amount of requests, the required amount of resources are different when the maximum request response time promotes from 3 seconds to 3 minutes. Therefore, one MFC_s^i value should be also related to a stable SLA duration. Since our discussion focus on solving the problems during which the QoS requirements and workload compositions has no change, we will not specify a stable SLA duration when we mention MFC_s^i.

When adopting n instances of a specific flavor i on a specific stage s, the upper-bound of capable arrival rate will be n times of MFC_s^i. MFC_s^i of each stages together with the number of instances used on each stage can determine the upper-bound of capable arrival rate of the whole system. The capable arrival rate of the system is depending on the bottle neck stage. In another word, the upper-bound of capable arrival rate of the bottle neck stages can be seen as the capable arrival rate upper-bound of the application running under that resource configuration. Therefore, we take the smallest value of capable arrival rate upper-bound among stages as the upper-bound of capable request arrival rate of the whole system.

6.3 Benchmark based MFC modeling

Our SLA feasibility study method is based on SelfBench 4.5.3. Here we list the formal expression of part of the informations of a VCAA configuration 4.2.

- Number of instances adopted in one configuration.
  - $NOI = \{noi_s \mid s\ in\ STAGE,\ noi_s\ is\ the\ number\ of\ instance\ of\ stage\ s\ \}$
  - NOI is the set of number of instances for each stage for a specific configuration.
  - noi_s is the number of instances used on stage s.
- Flavors adopted in one configuration.
  - $FLAVOR = \{flavor_s \mid s\ in\ STAGE,\ flavor_s\ is\ the\ flavor\ adopted\ by\ stage\ s\ \}$
  - $FLAVOR$ is the set of flavors adopted on each stage for a specific configuration.
  - $flavor_s$ is the flavor used on stage s.
- One configuration.
  - $Configuration(NOI, FLAVOR)$: the resource configuration of a VCAA. The flavors used on each stage is specified in the set $FLAVOR$. The number of instances of flavors on each stages are specified in the set NOI.
- The configuration and results of one SelfBench.
  - $(ReUtil_s^{cpu}, ReUtil_s^{memory}, ARmax) = SelfBench(Configuration(NOI, FLAVOR))$
  - Run SelfBench on the VCAA with a configuration of $Configuration(NOI, FLAVOR)$.
  - The SelfBench results include the CPU resource utilization $ReUtil_s^{cpu}$ on each stage s, the memory resource utilization $ReUtil_s^{memory}$ on each stage s and the maximum capable arrival rate $ARmax$.

The goal of Step1 and Step2 is to find the biggest most tailored flavor for each stage. One flavor is better tailored than another for one stage if the resource utilizations of CPU and memory are more balanced than another flavor. We consider that the balanced level of CPU utilization and memory utilization is determined on the configured amount of CPU and memory resources. We take the ratio between CPU amount and memory amount to represent the resource proportion of one flavor. For different stages, the most tailored flavor should be different. Step1 and Step2 will do benchmark to determine the starting point of searching the potential best tailored flavor based on used amount of CPU and memory. We try to set the starting point as big as possible so that a bigger MFC of a bigger flavor can be achieved during benchmark. Therefore more MFCs of smaller flavors can be achieved by linear estimation during the Step4. Step3 will positioning the best tailored flavors for serving step by step according to the used amount of CPU and memory and the resource utilization upper-bound. The achievable arrival rates will be approximately the same among all benchmarks during Step3, including the one with best tailored flavor and minimum number of instances, since the positioning process will try to guarantee the used amount of resources and resource utilization.

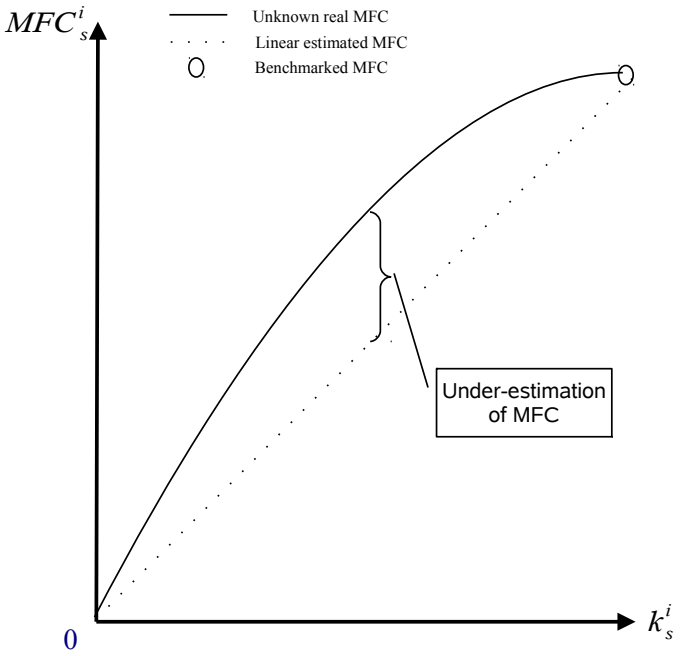

Figure 6.1. The situation that our SLA feasibility study method is applicable.

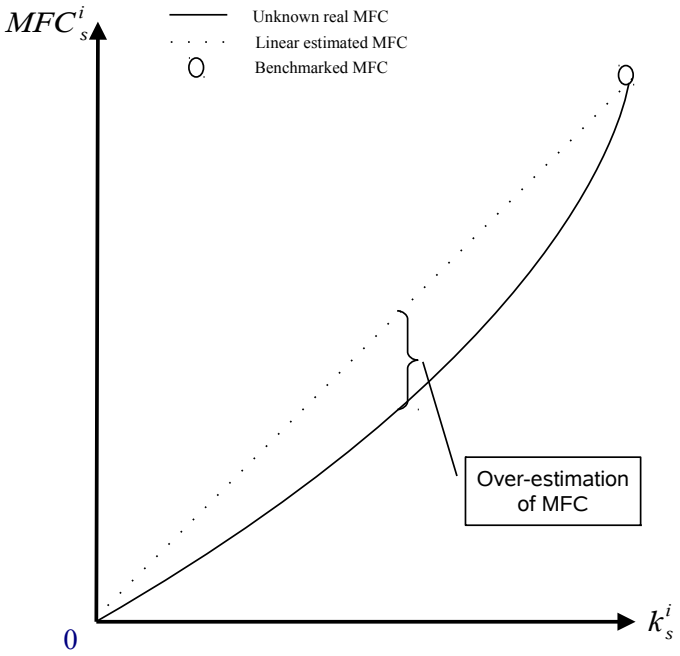

Figure 6.2. The situation that our SLA feasibility study method is NOT applicable.

Step4 will calculate the MFC for flavors smaller than or equal to the best tailored flavors during the last benchmark of Step3. This calculation is simply linear estimation with an acceptable level of under-estimation. The premise of this linear estimation, as demonstrated in figure 6.1 is that with the increase of the size of flavors, while the size of the flavor is no bigger than the benchmarked one, the increment of upper-bound of capable request arrival rate will be slower. The diversity of applications makes this premise may not be suitable for any case. On the contrary, for example, as demonstrated in figure 6.2, the growth rate of one application may increase with the increasing of flavor size. In this case, linear estimation will lead to over estimation of MFC which will further lead to under provisioning and SLA violation. Therefore, our SLA feasibility study method is not applicative. Or else, the maximum capable arrival rate may have no obvious relationship with flavor size. In this case, deducing MFC of one flavor based on benchmarking result of another flavor is risky and suspect so that our SLA feasibility study method is not adaptive. Our SLA feasibility study method do have constraints on the application. In another word, before adopting our SLA feasibility study method, it is necessary to have basic knowledge of the application to say that the growth rate of maximum capable arrival rate of this application will decrease(but still positive) with the increasing of the size of flavor.

Step5 will perform one benchmark for each stage which existing flavors bigger than the one used during the last benchmark of Step3. It is because that linear estimation can only be used to deduce the MFC of smaller flavors according to benchmarked MFC of bigger flavor, not vice versa. At least the biggest flavor's MFC need be achieved through benchmark. During previous 4 steps, benchmarked MFCs are those flavors used during the last benchmark of Step3. In order to limit the cost of SLA feasibility study process, only the minimum number of instances are used since there is no reasonable evidences for configuring the number of instances for a target request arrival rate during the benchmark and there is even no idea about what request arrival rate can be set as a target to perform consciously benchmark. However, after previous four steps, the over estimated MFC of biggest flavor is achievable and each stages can allocate resources accordingly to set up an environment for benchmarking the biggest MFC for each stages. Then, the benchmarked biggest MFC together with the benchmarked MFC during Step3 can later be used to estimate MFCs of undiscovered flavors.

6.3.1 Step1: find the smallest median configuration flavor

The goal of first two steps is to find one flavor, which has balanced resource configuration ratio and as big as possible amount of resources, as the start point of searching the best tailored flavor when serving a certain amount of request arrival rate for each stage.

In Step1, a benchmark is performed on the smallest flavors with Balanced resource configuration which means neither CPU nor memory is obviously more or less than another resource. Since we only want to check the ratio of used resource for each stages, these flavors should be as small as possible so that the benchmark process can be stopped quickly and the benchmark cost of step1 could be smaller.

As demonstrated in figure 6.3, for each stage s, we firstly calculate the $Ratio_s^i$ and $Distance_s^i(0,0)$ value for every available flavors. Then, we compare different $Ratio_s^i$ values and take the median value as $MedianRatioValue_s$ for current stage s. There may exist several flavors with the Ratio value of $MedianRatioValue_s$. Finally, for each stage, we choose the flavor with the smallest $Distance_s^i(0,0)$ among the flavors with a Ratio value of $MedianRatioValue_s$. These flavors are chosen to be used with the minimum number of in-

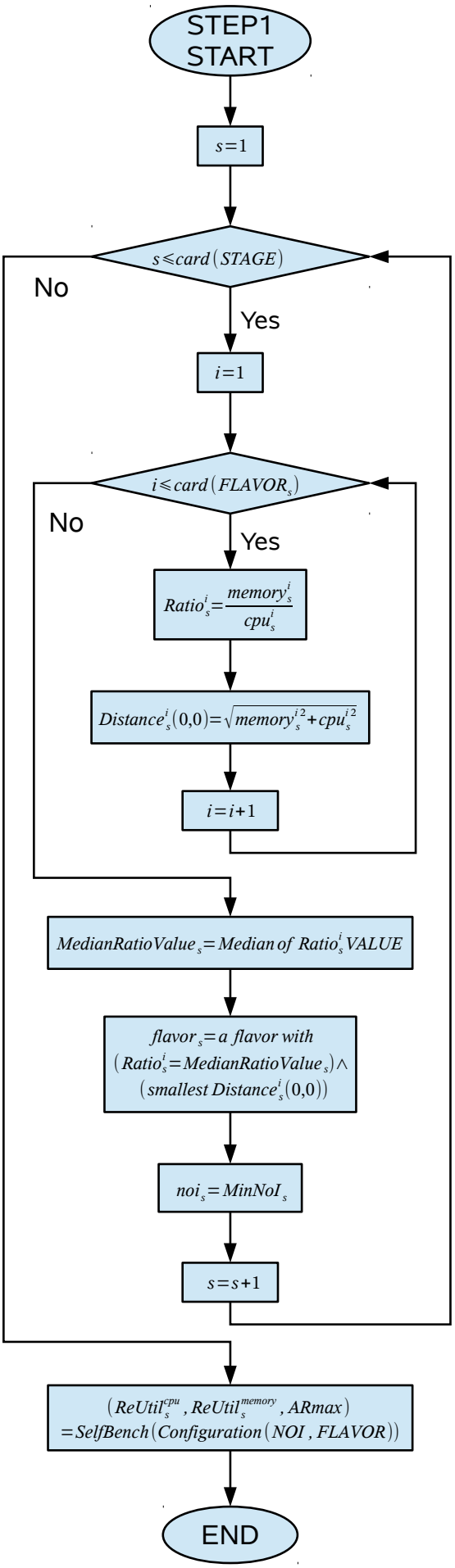

Figure 6.3. Step1: find the smallest median configuration flavor.

stances $MinNoI_s$ for SelfBench. The SelfBench gives the CPU resource utilization $ReUtil_s^{cpu}$ on each stage, the memory resource utilization $ReUtil_s^{memory}$ on each stage and the maximum capable arrival rate ARmax. These benchmark results will be used later in step2 to determine the start point of exploring the most tailored flavor for each stage.

6.3.2 Step2: find the biggest optimal configuration flavor

In the SelfBench results of Step1, probably the resource utilization of each resources on each stages are not in the same level. For example, for one stage, the CPU may already saturated while the memory utilization is quite low. It indicates that the Ratio value of chosen flavor is not optimized. In Step2, we will try to find the optimized Ratio for flavors.

As demonstrated in figure 6.4, for each stage s, we will check the CPU resource utilization $ReUtil_s^{cpu}$ and the memory resource utilization $ReUtil_s^{memory}$ to calculate how many CPU resources and memory resources are used on each stages by simply multiplying resource utilization and corresponding resource amount together. The ratio of these used resources is seen as the $OptimizedRatio_s$ for stage s. Then, the flavor which has biggest $Distance_s^i$ among the flavors with a $Ratio_s^i$ value of $OptimizedRatio_s$ will be adopted to perform next SelfBench. The number of instances for each stage is still the $MinNoI_s$. This configuration is taken as the start point of exploring the best tailored flavors during Step3. This exploring process is consist of several SelfBenchs. The first SelfBench is performed at the end of Step2 and the results of this SelfBench will be used as the input of Step3.

6.3.3 Step3: find the most tailored flavor

The most tailored flavor for each stage could be different because the resource requirements for running each stage are not necessary the same. Benchmarks on the most tailored flavors provides the possibility of fully using the capability of one flavor on the stage. The maximum capable arrival rate can be reached when adopting instances of the most tailored flavors during SelfBench processes.

Fully using all resources during the final SelfBench of step3 makes the linear estimation of speculated MFCs more accurate. To be more precise, in our SLA feasibility study method, under estimation(no over estimation) of MFCs exist when one MFC is achieved not through benchmark. The higher resource utilizations are, the smaller the under estimation level is.

As demonstrated in figure 6.5, for each stage s, the needed amount of CPU or memory resources will be calculated again if the resource utilization is less than 80%. The needed amount of resources consider how many resource should be allocated when resource utilization is 80%. It is computed by dividing the used amount of resources by 80%. The used amount of resource is computed by multiplying resource utilization from SelfBench and allocated resources. For each stage, these needed amount of resources cpu and mem are used as a optimized configuration of flavor. This optimized flavor probably are not existing according to SLA. Therefore, we will use the flavor with the smallest $Distance_s^i(mem, cpu)$ value among the flavors closer to the flavor with no CPU and memory than current one.

If one stage finds a new flavor, one more SelfBench will be performed. The SelfBench results will be calculated as mentioned above and the whole process will circulate until no new configuration is derived. The last SelfBench results will be used to calculate and speculate

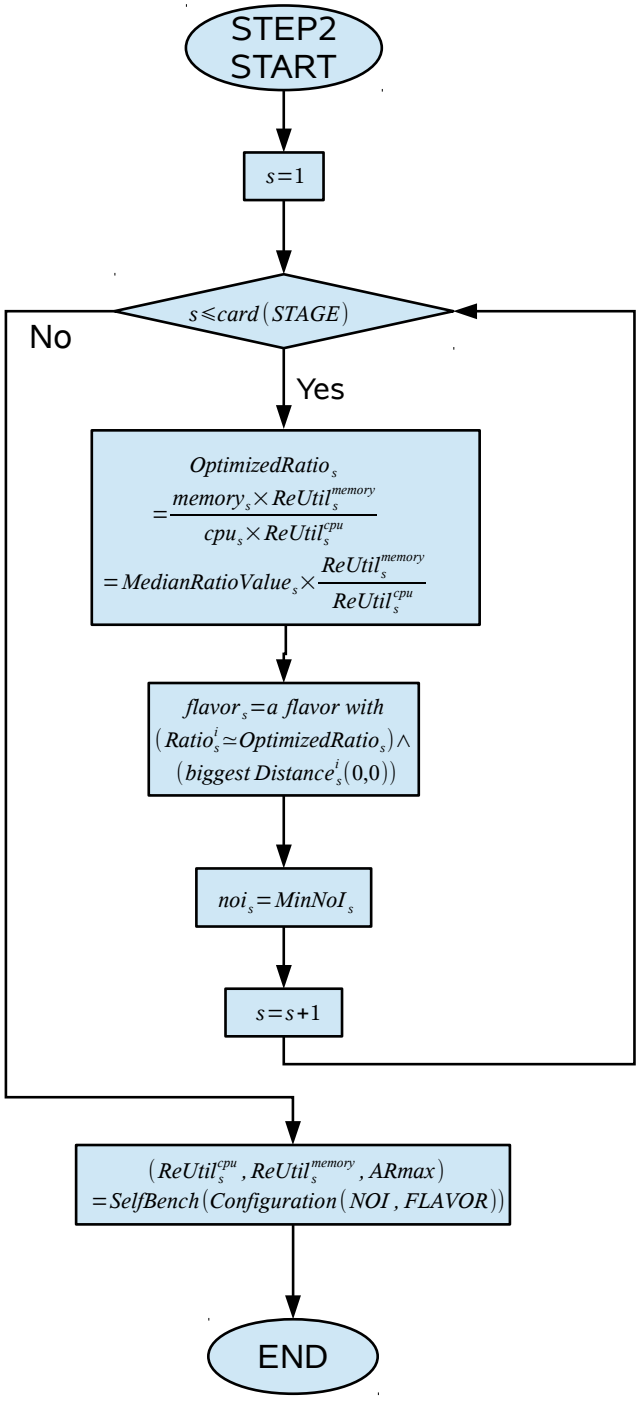

Figure 6.4. Step2: find the biggest optimal configuration flavor.

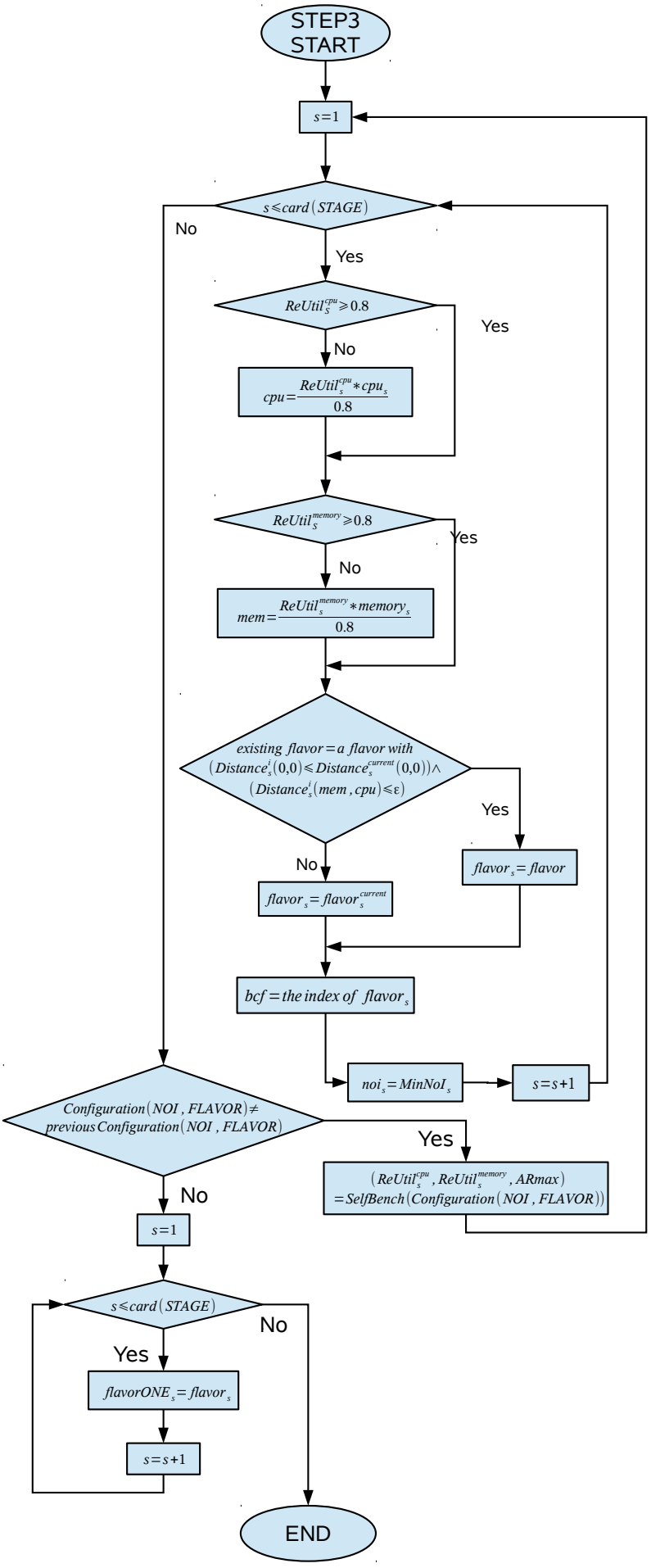

Figure 6.5. Step3: find the most tailored flavor.

MFCs in step4 and step5. The flavors used in the last SelfBench is recorded in $flavorONE_s$ and later used in step5.

6.3.4 Step4: calculate MFCs of flavors no bigger than most tailored flavor

In step4, the MFC of flavors used in the last SelfBench of step3 will be calculated. One flavor on each stage achieve its MFC value. This MFC value and its flavor resource configuration are used to speculate the MFC of some of the other flavors.

As demonstrated in figure 6.6, for each stage s, its available flavors allocated resources are compared with the resource configuration of the last benchmarked flavor. For one flavor $flavor_s^i$, the minimum ratio of these resources is marked as k_s^i. k_s^i represents that $flavor_s^i$ is how many times more powerful than the last benchmarked flavor on one stage. k_s^i can be smaller than, equal to or be bigger than 1. When k_s^i equals to 1, it means $flavor_s^i$ is as powerful as the last benchmarked flavor but not necessary has the same configuration of resources. When k_s^i is smaller than 1, it means that $flavor_s^i$ is less powerful than the last benchmarked flavor. Since it is reasonable to assume that the MFC of a flavor with 0 vCPU and 0 gigabyte of memory is 0, the less powerful flavors' MFC can be linearly estimated according to figure 6.1. The linear estimation is according to $MFC_s^i = (MFC_s) * k_s^i$. When k_s^i is bigger than 1, it means that $flavor_s^i$ is more powerful than the last benchmarked flavor. However, linear estimation based on $MFC_s^i = (MFC_s) * k_s^i$ will lead to over estimation of MFC which is not acceptable for our SLA feasibility study. For $flavor_s^i$ with k_s^i bigger than 1, more SelfBenchs are needed. The SelfBench processes will be performed in step5. As described in section 6.2.2, the linear extrapolation of MFC should be based on the benchmark result on saturated resources. However, it is possible that the last SelfBench in Step3 still can't get every flavor on every stage saturated. Therefore, the linear estimation based on the benchmark result on this unsaturated stage, will be under estimated too. To avoid these under estimation, we can perform more SelfBenchs on the configurations with more instances on saturated stages. However, we still take the unsaturated benchmark result and accept the under estimation of MFC to reduce the cost of SLA feasibility study. These under estimation of MFC can be improved during runtime control as described in chapter runtime control 7.

6.3.5 Step5: calculate MFCs of flavors bigger than most tailored flavor

For the more powerful $flavor_s^i$, linear estimation based on $MFC_s^i = (MFC_s) * k_s^i$ will lead to over estimation of MFC_s^i. Over estimation of MFC_s^i is not acceptable, but it can be a reference for establishing the benchmark for bigger MFC_s^i. In order to achieve all the bigger MFC values through minimum amount of SelfBenchs in step5, the biggest $flavor_s^{Kmax}$, which has the biggest k_s^i value, will be benchmarked. Therefore, rest of the flavors can speculate their MFC_s^i linearly.

As demonstrated in figure 6.7, for each stage s, if there is existing $flavor_s^i$ with k_s^i bigger than 1, one more SelfBench is needed for this stage s. This SelfBench will be performed to fully use the ability of the biggest flavor on this stage. The resource configuration will be using the minimum number of instances of the biggest flavor on current stage s, while other stages will use the flavor used during the last SelfBench of step3. The number of instances for other stages will be enough for serving the maximum requests arrival rate of the configuration on stage s. This maximum requests arrival rate is unknown yet, but the upper-bound of this

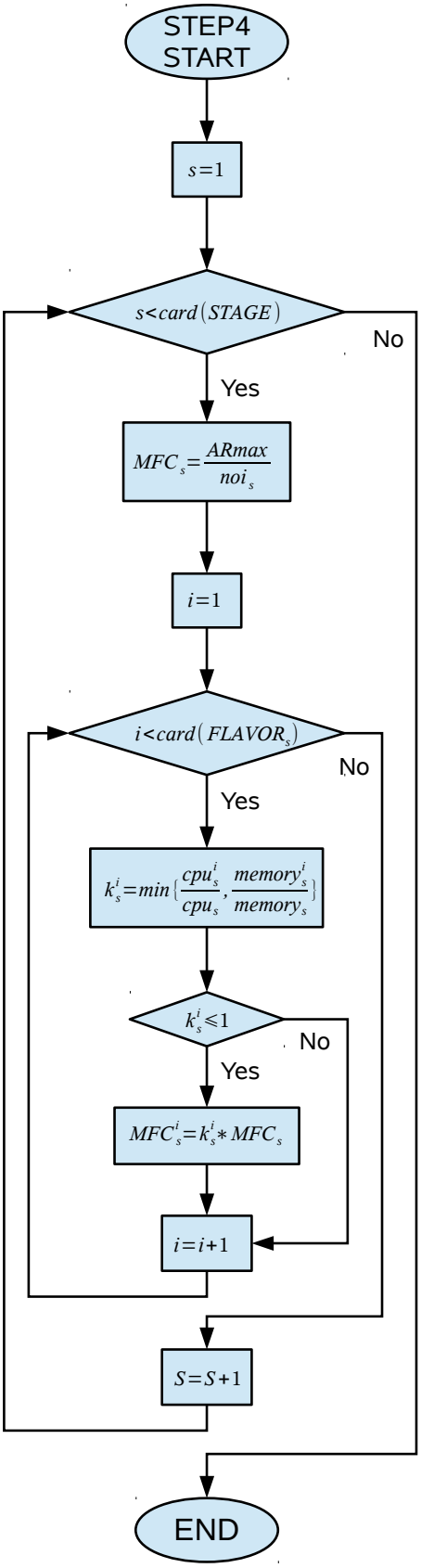

Figure 6.6. Step4: calculate MFCs of flavors no bigger than most tailored flavor.

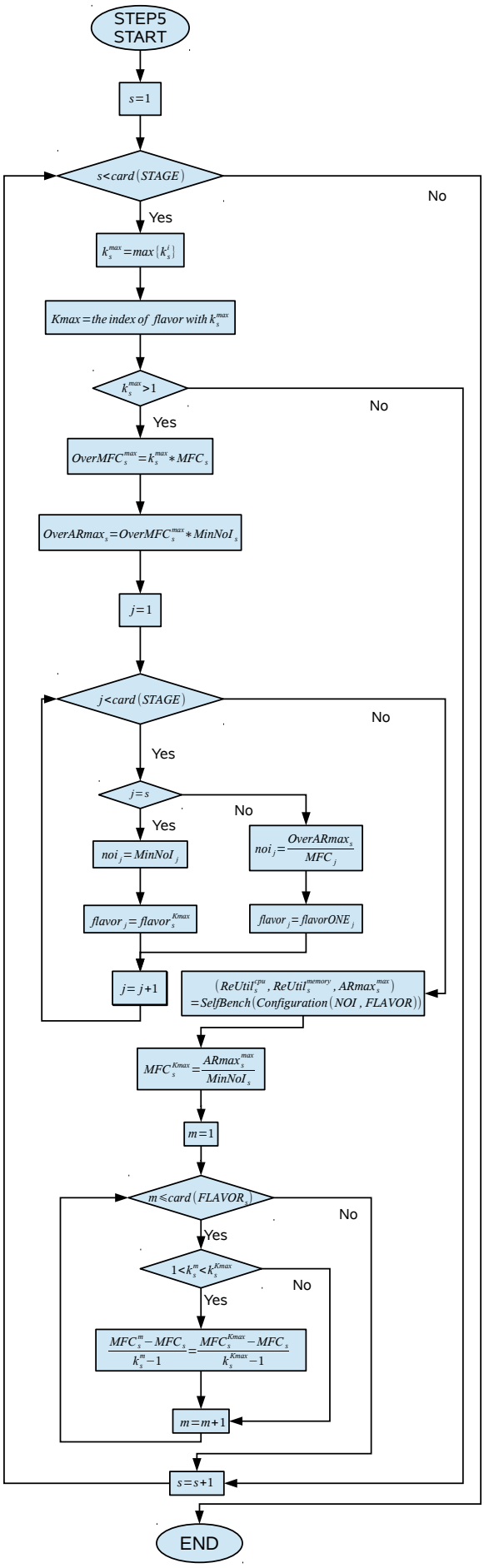

Figure 6.7. Step5: calculate MFCs of flavors bigger than most tailored flavor.

value is known by over estimation based on $OverARmax_s = k_s^{Kmax} * MFC_s * MinNoI_s$. Therefore the number of instances of other stage j will be $OverARmax_s/MFC_j$. According to this SelfBench results, the rest of MFCs can be calculated and deduced. The MFC_s^{Kmax} is derived through this SelfBench. The other flavors which have k_s^m bigger than 1 but smaller than k_s^{Kmax} will be linearly estimated according to $MFC_s^i = ((MFC_s^{Kmax} - MFC_s)/(k_s^{Kmax} - 1)) * (k_s^m - 1) + MFC_s$. Up to now, all MFC_s^i are derived. These MFC_s^i will be later used in step6 to evaluate the resource constraints and cost constraints are reasonable or not.

6.4 SLA constraints evaluation

The goal of SLA feasibility study is to evaluate whether there are conflicts of constraints in the SLA.

We evaluate these constraints based on the MFC_s^is which are benchmarked and derived through aforementioned process. The evaluation of the limited amount of instances of permitted flavors will be introduced in Resource constraints evaluation. The evaluation of the budget will be introduced later in Cost constraints evaluation. QoS targets are satisfied during the SLA feasibility study process since QoS targets are taken as reference during the whole benchmark process. Therefore, QoS targets are not evaluated in speciality.

In this section, resource constraints evaluation will be introduced in section 6.4.1. Cost constraints evaluation will be introduced in section 6.4.2.

6.4.1 Resource constraints evaluation

Resource constraints evaluation checks that the target workload can be served or not by regarding to the QoS target with the limited amount of instances of permitted flavors. The maximum capable workload is depending on the maximum capable request arrival rate according to the resource constraints on the finest bottleneck stage.

As demonstrated in figure 6.8, for each stages, the capable request arrival rate when adopting each flavors are calculated by $MFC_s^i(MaxNoI_s^i) = MFC_s^i * MaxNoI_s^i$ and the biggest one are recorded as the maximum capable request arrival rate $MaxMFC_s(n)$ for this stage s. The smallest $MaxMFC_s(n)$ value among stages is taken as the maximum capable request arrival rate $SLAResourceARmax$ of the whole application according to the resources constraints defined in SLA. $SLAWorkloadARmax$ is the maximum request arrival rate defined in the workload constraints of SLA. With the resource constraints defined in SLA, the maximum capable request arrival rate is at most $SLAResourceARmax$. If $SLAResourceARmax < SLAWorkloadARmax$, it means that the target upper-bound of request arrival rate is too high for the permitted flavors and available amount of instances. The SLA proposal should be rejected because the permitted resource is not enough for serving the target workload. If $SLAResourceARmax > SLAWorkloadARmax$, it means that the target upper-bound of request arrival rate can be served with the permitted flavors and available amount of instances. The SLA proposal passes the resource constraints evaluation.

6.4.2 Cost constraints evaluation

Cost constraints evaluation checks that the target workload can be served within the budget or not. To be more precise, the cost of serving the maximum request arrival rate defined in

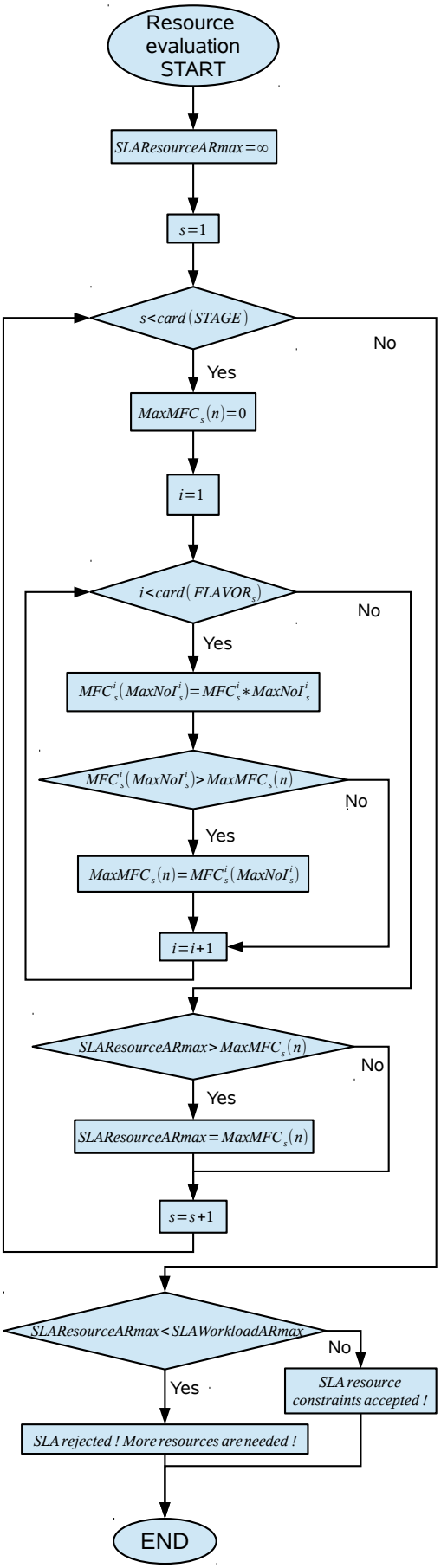

Figure 6.8. Resource constraints evaluation based on MFC_s^i.

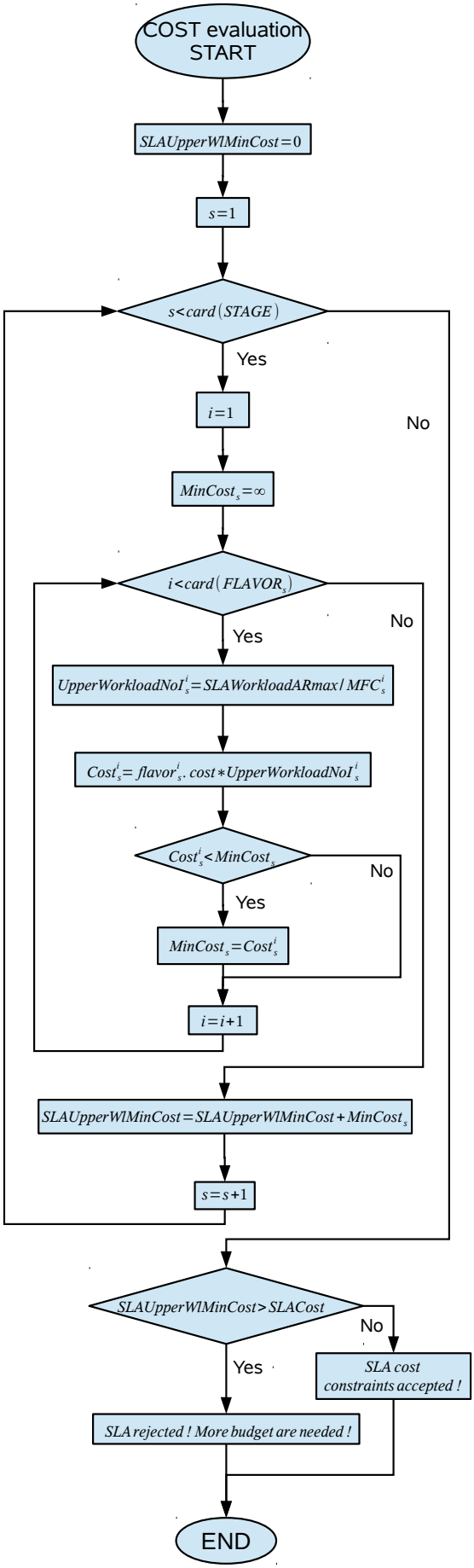

Figure 6.9. Cost constraints evaluation based on MFC_s^i.

workload constraints will be calculated and compared with the permitted cost for maintaining the QoS. As demonstrated in figure 6.9, for each stage s, the need amount of instances $UpperWorkloadNoI_s^i$ for serving the maximum request arrival rate $SLAWorkloadARmax$ are calculated by dividing MFC_s^i for adopting each permitted flavors. If the number of instances $UpperWorkloadNoI_s^i$ are available according to resource constraints, $UpperWorkloadNoI_s^i$ together with the cost of this flavor $flavor_s^i.cost$ give the cost $Cost_s^i$ for serving the maximum request arrival rate with relevent flavor on this stage. The minimum $Cost_s^i$ will be taken as the minimum cost $MinCost_s$ for serving the maximum request arrival rate $SLAWorkloadARmax$ on stage s. The minimum cost $MinCost_s$ for each stages are sumed up as the minimum cost $SLAUpperWlMinCost$ of the whole application for serving the maximum request arrival rate $SLAWorkloadARmax$. If this whole cost $SLAUpperWlMinCost$ is higher than the cost upper-bound $SLACost$ defined in SLA, this SLA proposal should be rejected because of the cost constraint evaluation.Otherwise, the cost constraints evaluation is passed.

6.5 Scenario

To make our benchmark based SLA feasibility study method understandable, we will give an example. The PSLA based SLA example will be described in section 6.5.1. The Benchmark based MFC modelling example will be given and explained in section 6.5.2. The SLA evaluations example based on MFC derived in section 6.5.2 will be given and explained in section 6.5.3.

6.5.1 PSLA based SLA example

Besides the QoS target, such as response time < 3s, there are many different constraints defined in one SLA offer, including resource constraints, workload constraints, QoS constraints and cost constraints. A PSLA based SLA example will be described in this section so that we can visually know what kind of problem we are facing. The QoS target example will be given in section 6.5.1. The workload constraints example will be given in section 6.5.1. The resource constraints example will be given in section 6.5.1. The cost constraints example will be given in section 6.5.1.

QoS target

QoS target quantifies the acceptable QoS metric "Value Range", QoS target is expressed in core part of QoS Criterion, "Service Level Objective(SLO)", by quantifying the acceptable QoS metric "Value Range". e.g. Response time $\leq$ 13 seconds.

Workload constraints

Workload constraints define the capable request arrival rate with allowed resources and budget. Workload constraints is expressed in "QualifyingCondition" of "GuaranteeTerm", including "Data Size", "Data Composition", e.g. it is composed of only "atomicrequestA" with a maximum arrival rate of 600 requests/s. However, workload composed of several different kinds of requests bring in difficulties of calibration through benchmarks. In order to simplify the problem for the first step, we assume that the workload is composed of the same kind of requests which require the same kind and amount of resources.

$FLAVOR_1$	f4	f5	f6
cpu_1^i	6	4	2
$memory_1^i$	3	2	1
cost	3	2	1

Table 6.1. Description of available flavors for $Stage_1$.

$FLAVOR_2$	f1	f2	f3	f4
cpu_2^i	12	6	3	6
$memory_2^i$	3	2	1	3
cost	6	4	2	3

Table 6.2. Description of available flavors for $Stage_2$.

Resource constraints

Resource constraints are defined in the "QualifyingCondition" of a "GuaranteeTerm". Resource constraints have the information of which flavors can be used on each stages and the range of acceptable number of instances when adopting one specific flavor. Resource constraints indicate the kinds of allowed scaling actions and its limits. Scaling actions for virtualized application are at the virtual machine level. Two kinds of scaling, "horizontal scaling up/down" and "vertical scaling up/down", can be applied for resource allocation at runtime.

As demonstrated in table 6.1, 6.2 and 6.3, flavor f4, f5 and f6 can be used on $Stage_1$, flavor f1, f2, f3 and f4 can be used on $Stage_2$ and flavor f1, f2, f3, f4, f5, f6, f7, f8 and f9 can be used on $Stage_3$. The memory and CPU configuration of each flavors are listed. The price of using one instance of each flavor is also given. The measurement units for cpu_s^i, $memory_s^i$ and cost are respectively vCPU, gigabyte and Euro/s. For example, flavor f4 has 6 vCPU, 3G of memory and using one instance of flavor f4 will be charged 3 Euro per second.

We indicate the minimum (in table 6.5) and maximum (in table 6.4) number of instances ($MaxNoI_s^i$ and $MinNoI_s^i$) for flavors and stages which can be used for scaling actions. Both $MaxNoI_s^i$ and $MinNoI_s^i$ are constraints of available resources. $MaxNoI_s^i$ could be a constraints depending on the available instances in the resource pool. It can be the case that the available amount of instances of different flavors are different. Therefore, $MaxNoI_s^i$ can be different for different flavors on different stages. For example, as demonstrated in table 6.4, $Stage_1$ could use at most 2 instances of flavor f4, $Stage_2$ could use at most 4 instances of flavor f1 and $Stage_3$ could use at most 2 instances of flavor f1.

It seems that the minimum number of instances when adopting one flavor on a stage is not a necessary information since a too low used amount of resources is not a problem to be worried. However, $MinNoI_s$ could be set for technical or functional reasons. Therefore, we specify the minimum number of instance $MinNoI_s$ for each stage. For example, as demonstrated in

$FLAVOR_3$	f1	f2	f3	f4	f5	f6	f7	f8	f9
cpu_3^i	12	6	3	6	4	2	5	3	1
$memory_3^i$	3	2	1	3	2	1	10	6	1
cost	6	4	2	3	2	1	5	3	1

Table 6.3. Description of available flavors for $Stage_3$.

$MaxNoI_s^i$	f1	f2	f3	f4	f5	f6	f7	f8	f9
$Stage_1$				2	3	4			
$Stage_2$	4	6	8	5					
$Stage_3$	2	3	4	2	3	4	2	3	4

Table 6.4. Resource constraints in SLA: maximum number of instances $MaxNoI_s^i$.

Stage	$MinNoI_s$
$Stage_1$	1
$Stage_2$	2
$Stage_3$	1

Table 6.5. Resource constraints in SLA: minimum number of instances $MinNoI_s$.

table 6.5, for any flavors, $Stage_1$ should have at least 1 instance, $Stage_2$ should have at least 2 instances and $Stage_3$ should have at least 1 instance.

Cost constraints

Cost constraints is defined either in BusinessValueList or as a QoS target in Service Level Objective. Cost constraints restrict the acceptable cost for maintaining the QoS. Cost constraints can also be transformed into permitted resources provided with pricing model which is beyond the scope of this thesis. Cost constraints tell how much money can be spent per time unit, e.g. cost $<$ 20 Euro/s.

6.5.2 Benchmark based MFC modelling

In Step1, as demonstrated in figure 6.3, a series of calculation will be performed. Firstly, $Ratio_s^i$and $Distance_s^i$ for each flavors and stages will be calculated according to $Ratio_s^i = \frac{memory_s^i}{cpu_s^i}$ and $Distance_s^i(0,0) = \sqrt{memory_s^{i\,2} + cpu_s^{i\,2}}$. In our scenario, the $Ratio_s^i$and $Distance_s^i$ calculation results for $Stage_1$, $Stage_2$ and $Stage_3$ are listed respectively in table 6.6, table 6.7 and table 6.8.

As demonstrated in figure 6.3, based on the calculated $Ratio_s^i$ and $Distance_s^i(0,0)$ listed in table 6.6, 6.7 and 6.8, the smallest flavor(flavor has smallest $Distance_s^i(0,0)$ value) among median values of $Ratio_s^i$ is adopted for each $Stage_s$. In this scenario, the median ratio value for $Stage_1$, $Stage_2$ and $Stage_3$ are 1/2, 1/3 and 1/2. The smallest flavors with these median ratio value for $Stage_1$, $Stage_2$ and $Stage_3$ are f6, f3 and f6. The amount of instances for $Stage_1$, $Stage_2$ and $Stage_3$ are the minimum number of instances defined in SLA, which are 1 instance, 2 instances and 1 instance, as demonstrated in table 6.5. The benchmark results is demonstrated in table 6.9.

$FLAVOR_1$	f4	f5	f6
$Ratio_1^i$	1/2	1/2	1/2
$Distance_1^i(0,0)$	6.7	4.5	2.2

Table 6.6. $Ratio_1^i$ and $Distance_1^i$ calculated in Step1 for $Stage_1$.

$FLAVOR_2$	f1	f2	f3	f4
$Ratio_2^i$	1/4	1/3	1/3	1/2
$Distance_2^i(0,0)$	12.4	6.3	3.2	6.7

Table 6.7. $Ratio_2^i$ and $Distance_2^i$ calculated in Step1 for $Stage_2$.

$FLAVOR_3$	f1	f2	f3	f4	f5	f6	f7	f8	f9
$Ratio_3^i$	1/4	1/3	1/3	1/2	1/2	1/2	2	2	1
$Distance_3^i(0,0)$	12.4	6.3	3.2	6.7	4.5	2.2	11.2	6.7	1.4

Table 6.8. $Ratio_3^i$ and $Distance_3^i$ calculated in Step1 for $Stage_3$.

According to the algorithm of step2, which is demonstrated in figure 6.4, the memory and CPU resource utilization ($ReUtil_s^{memory}$ and $ReUtil_s^{cpu}$) for each stage $Stage_s$ together with the allocated amount of memory $memory_s$ and CPU cpu_s, the used amount of resources $UsedRe_s^{cpu}$ and $UsedRe_s^{memory}$ can be calculated. An optimized resource ratio value $OptimizedRatio_s$ is derived from the used amount of resources $UsedRe_s^{cpu}$ and $UsedRe_s^{memory}$. This $OptimizedRatio_s$ will be used as the guideline for choosing the flavors for next SelfBench. The flavor with biggest $Distance_2^i(0,0)$ will be chosen and a bigger flavor' MFC will be get through benchmark.

Taking $Stage_2$ for example, $ReUtil_2^{memory}$ is 40% and $ReUtil_s^{cpu}$) is 30%. The allocated amount of memory $memory_s$ is 3G and CPU cpu_s is 12. The used amount of CPU $UsedRe_s^{cpu}$ is 1.2 and $UsedRe_s^{memory}$ is 3.6. The optimized resource ratio $OptimizedRatio_2$ is $1.2/3.6 = 1/3$. The flavor chosen for the SelfBench at the end of Step2 should be approximate to $OptimizedRatio_2 = 1/3$. According to table 6.12, flavor f2 and f3 both have a $Ratio_2^i$ of 1/3. Flavor f2 is chosen for $Stage_2$ in the next SelfBench because f2 has a bigger $Distance_2^i(0,0)$ value than f3. Similarly, flavor f4 is chosen for $Stage_1$ and flavor f7 is chosen for $Stage_3$.

Table 6.10 presents the SelfBench results of Step2. In our scenario, the SelfBench result $ARmax$ is 200. The memory utilization $ReUtil_s^{memory}$ and CPU utilization $ReUtil_s^{cpu}$ together with allocated amount of memory $memory_s$ and CPU cpu_s indicate that the optimized amount of memory mem and CPU cpu. The determination of mem and cpu takes 80% of resource utilization threshold into consideration. mem and cpu can be seen as the optimized configuration of flavor for serving request arrival rate $ARmax = 200$.

Taking $Stage_3$ for example, The memory utilization $ReUtil_s^{memory}$ is 40% and CPU utilization $ReUtil_s^{cpu}$ is also 40%. The allocated amount of memory $memory_s$ is 10G and CPU cpu_s is 5. The resource utilization threshold for CPU and memory are both 80%. The optimized amount of memory mem is 5 gigabyte and CPU cpu is 5/2.

The SelfBench results of step2 will be used in step3 as the starting point of searching the configuration for getting the benchmarked MFC. Taking $Stage_3$ for example, as demonstrated in table 6.13, the minimum $Distance_3^i(5, 5/2)$ value exceed the configurable threshold value $\varepsilon = 1$. Therefore, the flavor for $Stage_3$ in next configuration of SelfBench is the same as the

	$memory_s$	cpu_s	$ReUtil_s^{memory}$	$ReUtil_s^{cpu}$	$UsedRe_s^{memory}$	$UsedRe_s^{cpu}$	$OptimizedRatio_s$
$Stage_1$	1	2	80%	80%	0.8	1.6	1/2
$Stage_2$	1	3	70%	70%	0.7	2.1	1/3
$Stage_3$	1	2	80%	20%	0.8	0.4	2

Table 6.9. Extrapolation of $OptimizedRatio_s$ based on SelfBench results of step1.

	$memory_s$	cpu_s	$ReUtil_s^{memory}$	$ReUtil_s^{cpu}$	mem	cpu	$flavor_s^{current}$	$flavor_s$
$Stage_1$	3	6	30%	30%	9/8	9/4	f4	f6
$Stage_2$	2	6	80%	80%	2	6	f2	f2
$Stage_3$	10	5	40%	40%	5	5/2	f7	f8

Table 6.10. Step2 SelfBench on one instance of f4 on $Stage_1$, two instances of f2 on $Stage_2$ and one instance of f7 on $Stage_3$.$\varepsilon = 1.5$.ARmax= 200.

$FLAVOR_1$	f4	f5	f6
$Distance_1^i(9/8, 9/4)$	4.2	2	0.28
$Distance_1^i(0, 0)$	6.7	4.5	2.2

Table 6.11. Better tailored flavor exploring for $Stage_1$ in Step3.

current one which is flavor f6.

Similarly, referring to table 6.11 and table 6.12, flavor for $Stage_1$ is still flavor f6 and flavor for $Stage_2$ is still flavor f2. According to algorithm described in figure 6.5, the exploring of better tailored flavors will be stopped because there is no change of flavors. Otherwise, the SelfBench will be continuous and the exploring will be performed from all over again, until there is no change of flavors on each stages.

At the end of Step3, one flavor for each stages will be achieved its MFC_s^i through benchmark. These benchmarked MFC_s^i will be used to extrapolate the MFC_s^i of smaller flavors. The benchmarks for bigger flavors in step5 will be established based on these benchmarked MFC_s^i too. The linear estimation factor is k_s^i. k_s^i is derived by comparing each flavors with the saturated flavor for each $Stage_s$ separately. As described in figure 6.6, k_s^i is the minimum ratio between the ratio of CPU and the ratio of memory. The ratio of CPU is derived by comparing the configured amount of CPU to the saturated flavor used in the last SelfBench in Step3. The ratio of memory is derived in the same way. As demonstrated in table 6.14, taking flavor f1 used on stage $Stage_2$ for example, $flavor_2^1$ has 12 vCPU and 3 gigabyte of memory. The flavor used on stage $Stage_2$ in the last SelfBench of Step3 is flavor f2, which has 6 vCPU and 2 gigabyte of memory. Therefore, the ratio of CPU is 2 and the ratio of memory is 1.5. k_2^1, the linear factor of $flavor_2^1$, is 1.5. Other linear factors k_s^i are calculated in the same way and are listed in table 6.14.

The linear factor k_s^i will be used in step4 to get the MFC for flavors has a k_s^i lower than or equal to 1. Taking flavor f3 on stage $Stage_2$ for example, k_2^3 is 0.5 and the benchmarked MFC is 100, therefore $MFC_2^3 = k_2^3 * 100$ is 50. Flavor f6 for stage $Stage_1$, flavor f2, f3, f4 for stage $Stage_2$ and flavor f3, f6, f9 for stage $Stage_3$ have a k_s^i lower than or equal to 1. Their MFC_s^i can be derived similarly and are listed in table 6.15.

Flavor f4 and f5 for stage $Stage_1$, flavor f1 for stage $Stage_2$ and flavor f1, f2, f4, f5, f7, f8 for stage $Stage_3$ have k_s^i bigger than 1.

As depicted in figure 6.10, their MFC_s^i can not be derived only based on previous bench-

$FLAVOR_2$	f1	f2	f3	f4
$Distance_2^i(2, 6)$	6.1	0	3.2	1
$Distance_2^i(0, 0)$	12.4	6.3	3.2	6.7

Table 6.12. Better tailored flavor exploring for $Stage_2$ in Step3.

$FLAVOR_3$	f1	f2	f3	f4	f5	f6	f7	f8	f9
$Distance_3^i(5,5/2)$	9.7	4.6	4	4	3.4	4	5.6	1.1	4.3
$Distance_3^i(0,0)$	12.4	6.3	3.2	6.7	4.5	2.2	11.2	6.7	1.4

Table 6.13. Better tailored flavor exploring for $Stage_3$ in Step3.

k_s^i	f1	f2	f3	f4	f5	f6	f7	f8	f9
$Stage_1$				3	2	1			
$Stage_2$	1.5	1	0.5	1					
$Stage_3$	1/2	1/3	1/6	1/2	1/3	1/6	5/3	1	1/6

Table 6.14. Linear extrapolation factor k_s^i in step4.

MFC_s^i	f1	f2	f3	f4	f5	f6	f7	f8	f9
$Stage_1$				?	?	200			
$Stage_2$	?	100	50	100					
$Stage_3$	100	67	33	100	67	33	?	200	33

Table 6.15. Step4: $MFC_1 = 200$,$MFC_2 = 100$,$MFC_3 = 200$.

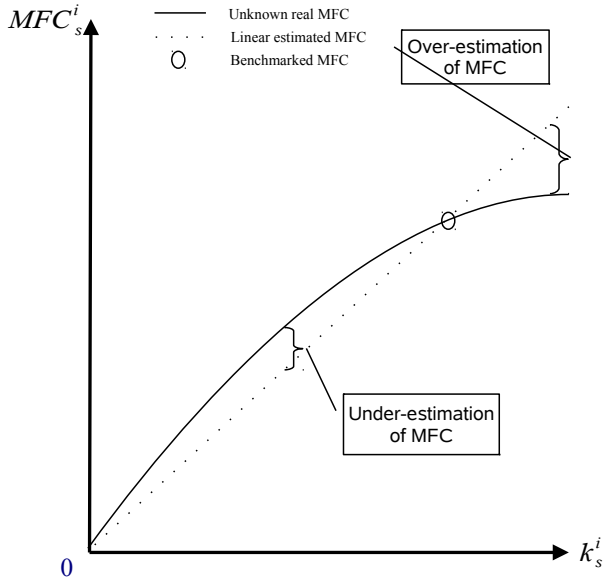

Figure 6.10. Linear estimation leads to over-estimation and under-estimation.

MFC_s^i	f1	f2	f3	f4	f5	f6	f7	f8	f9
$Stage_1$				500	350	200			

Table 6.16. Step5: SelfBench on 1 instance of f4 on $Stage_1$, 6 instances of f2 on $Stage_2$ and 3 instances of f6 on $Stage_3$.$\varepsilon = 1$.ARmax= 500. Therefore, $MFC_1^4 = 500$.

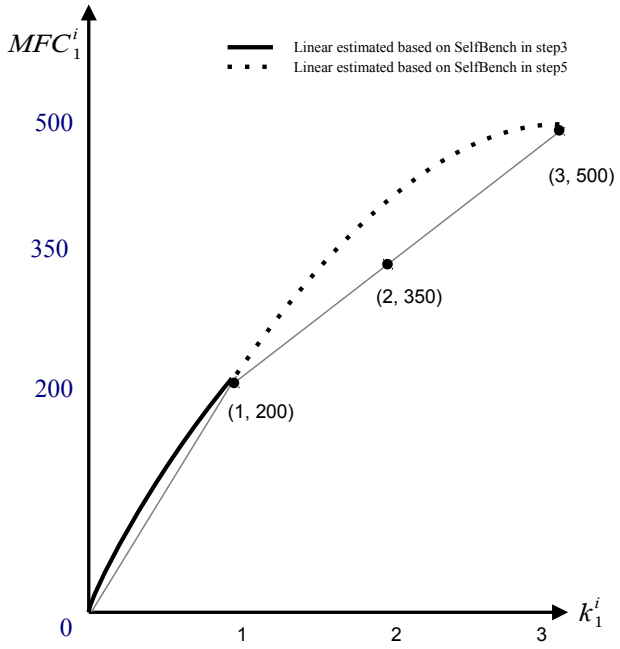

Figure 6.11. Linear estimation for $Stage_1$

marks. At least the benchmarks fully using the flavor with biggest linear factor k_s^i are needed to extrapolate rest of the MFC_s^i. The linear estimation based on factor k_s^i and previous SelfBench results will lead to over estimation of MFC_s^i of flavors with k_s^i bigger than 1. However, these over estimated MFC_s^i give us the upper-bound of real MFC_s^i and we can establish new SelfBench accordingly.

Taking stage $Stage_1$ for example, flavor f4 has the biggest k_1^i which is 3. Therefore, the biggest flavor is f4 and the MFC_1^4 is at most $3 * 200 = 600$. To get flavor f4 saturated on stage $Stage_1$, other stages $Stage_2$ and $Stage_3$ only need to configure enough resources for serving 600 request/s. In step5, we will perform SelfBench on 1 instance of flavor f4 on $Stage_1$, 6 instances of flavor f2 on $Stage_2$ and 3 instances of flavor f6 on $Stage_3$. The SelfBench result show that the maximum capable arrival rate is 500 request per second. Therefore, $MFC_1^4 = 500$.

As demonstrated in figure 6.11, MFC_1^5 can be linear estimated according to the algorithm depicted in figure 6.7. After this SelfBench in step5, all MFC_1^i for stage $Stage_1$ are known as listed in table 6.16.

Flavor f1 for $Stage_2$ and flavor f4 for stage $Stage_3$ can be benchmarked similarly. As demonstrated in figure 6.12 and 6.13, the MFC_s^i for the rest of flavors on each stage can be derived as listed in table 6.17 and table 6.18.

Until now, as summarized and listed in table 6.19, we have the all the MFC_s^i for all stages and possible flavors. The resource constraints evaluation and cost constraint evaluation depicted

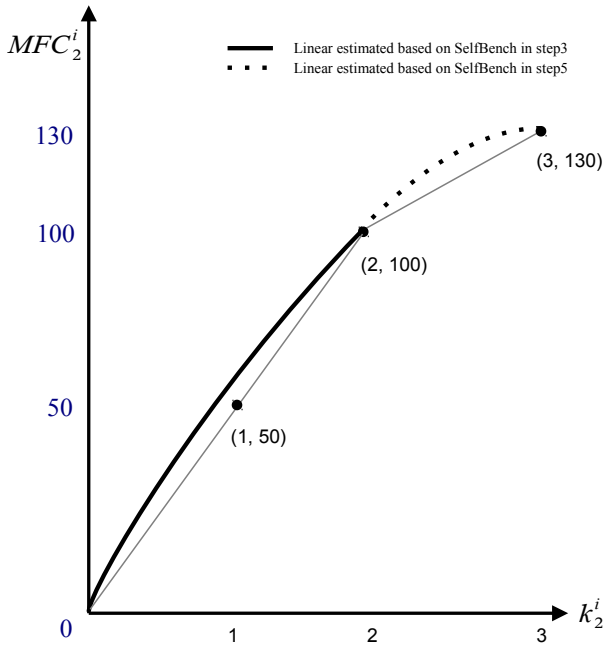

Figure 6.12. Linear estimation for $Stage_2$

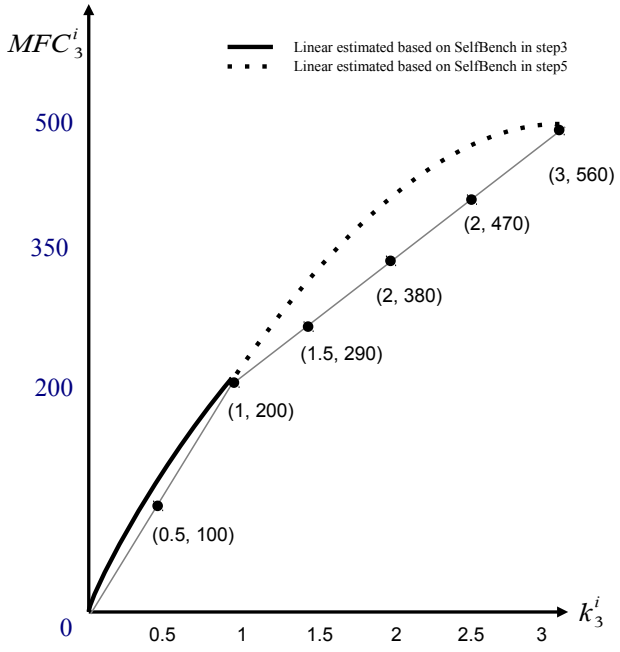

Figure 6.13. Linear estimation for $Stage_3$

MFC_s^i	f1	f2	f3	f4	f5	f6	f7	f8	f9
$Stage_2$	130	100	50	100					

Table 6.17. Step5: SelfBench on 2 instances of f6 on $Stage_1$, 2 instances of f1 on $Stage_2$ and 2 instances of f6 on $Stage_3$.$\varepsilon = 1$.ARmax= 260. Therefore, $MFC_2^1 = 130$.

MFC_s^i	f1	f2	f3	f4	f5	f6	f7	f8	f9
$Stage_3$	100	67	33	100	67	33	270	200	33

Table 6.18. Step5: SelfBench on 2 instances of f6 on $Stage_1$, 4 instances of f2 on $Stage_2$ and 1 instance of f7 on $Stage_3$.$\varepsilon = 1.5$.ARmax= 270. Therefore, $MFC_3^7 = 270$.

MFC_s^i	f1	f2	f3	f4	f5	f6	f7	f8	f9
$Stage_1$				500	350	200			
$Stage_2$	130	100	50	100					
$Stage_3$	100	67	33	100	67	33	270	200	33

Table 6.19. Final MFC_s^i

in figure 6.8 and figure 6.9 can be performed according to table 6.19.

6.5.3 MFC based SLA evaluation

In this section, the SLA evaluations example based on MFC derived in section 6.5.2 will be given and explained. Resources constraints evaluation example will be given in section 6.5.3. Cost constraints evaluation example will be given in section 6.5.3.

Resources constraints evaluation

Now, we have the MFC_s^i for each flavors and each stages. As mentioned before, The maximum arrival rate can be served under the resource constraints is actually depending on the weakest bottleneck stage. According to algorithm depicted in figure 6.8, we take the minimum value of $MFC_s^i(MaxNoI_s^i)$ among stages as the maximum capable arrival rate according to resource constraints in the SLA.

For example, flavor f4, f5 and f6 can be used on stage $Stage_1$. When using the maximum amount of flavor f4, which is 2 instances, on stage $Stage_1$, the whole application could serve at most $2 * 500 = 1000$ requests/s. Similarly, the application could serve at most 1050 requests/s when using flavor f5 and 800 requests/s when using f6. Therefore, the maximum capable arrival rate for stage $Stage_1$ is 1050 requests/s when adopting maximum amount of flavor f5. Similarly, the maximum capable arrival rate for stage $Stage_2$ is 600 requests/s when adopting maximum amount of flavor f2. The maximum capable arrival rate for stage $Stage_3$ is 600 requests/s when adopting maximum amount of flavor f8. Therefore, $SLAResourceARmax$, the maximum capable arrival rate under the resource constraints defined in SLA, is 600 requests/s. From the resource constraint evaluation process, we can also see that the resources allowed for

$MFC_s^i(MaxNoI_s^i)$	f1	f2	f3	f4	f5	f6	f7	f8	f9	$MaxMFC_s(n)$
$Stage_1$				1000	1050	800				1050
$Stage_2$	520	600	400	500						600
$Stage_3$	200	201	132	200	201	132	540	600	132	600
$SLAResourceARmax$										600

Table 6.20. SLA resource constraints evaluation based on $MFC_s^i(MaxNoI_s^i)$

$UpperWorkloadNoI_s^i$	f1	f2	f3	f4	f5	f6	f7	f8	f9
$Stage_1$				2	2	3			
$Stage_2$		x	6	x	x				
$Stage_3$		x	x	x	x	x	x	3	x

Table 6.21. Needed amount of instances to serve $SLAWorkloadARmax$.

$Cost_s^i$	f1	f2	f3	f4	f5	f6	f7	f8	f9	$MinCost_s$
$Stage_1$				6	4	3				3
$Stage_2$	x	24	x	x						24
$Stage_3$	x	x	x	x	x	x	x	9	x	9
$SLAcost$										36

Table 6.22. Cost for serving $SLAWorkloadARmax$.

stage $Stage_2$ and $Stage_3$ makes stage $Stage_2$ and $Stage_3$ be the bottleneck which block the application fully uses available resources on stages $Stage_1$ to serve higher request arrival rate.

Obviously, $SLAWorkloadARmax$, the maximum arrival rate defined in the SLA should not be higher than $SLAResourceARmax$, which is 600 requests/s in our scenario. Otherwise, this SLA proposal should be rejected.

Cost constraints evaluation

For cost constraints, we also take the maximum arrival rate into consideration. As described in figure 6.9, With MFC_s^i, we can know how many instances are needed on each stage. The minimum cost on each stage can be get with the cost of each flavors $flavor_s^i.cost$.

Taking $Stage_1$ for example, to serve a request arrival rate of $SLAWorkloadARmax = 600$, 2 instances of flavor f4, 2 instances of flavor f5 or 3 instances of flavor f6 are needed according to MFC_s^i listed in table 6.19. For $Stage_2$ and $Stage_3$, $UpperWorkloadNoI_s^i$, the needed amount of instances for each flavors can be derived similarly as table 6.21. When adopting one flavor, it is possible that the needed amount of instance of this flavor is more than SLA allowed. In this case, only SLA allowed number of instances can be taken into account.

$UpperWorkloadNoI_s^i$ listed in table 6.21 together with $flavor_s^i.cost$ in table 6.1, table 6.2 and table 6.3, we can know the $Cost_s^i$, which is the cost for adopting $flavor_s^i$ to serve the $SLAWorkloadARmax = 600$.

According to algorithm 6.9, the cost for each stages with each flavors are shown in table 6.22. The minimum cost to serve $SLAWorkloadARmax$ is the sum of the minimum $Cost_s^i$ for each stages. In this scenario, it is 3+24+9 = 36 (euro/s).

The sum of minimum cost should be no bigger than maximum cost defined as cost constraints in SLA. Otherwise, the SLA proposal should be rejected for cost constraints. Therefore the SLA proposal should be rejected because maintaining the QoS of serving 600 requests/s need at least 36 euro/s which is higher than SLA defined 20 euro/s.

6.6 Conclusion

We proposed a SLA feasibility study method based on benchmarking. The "what-if" evaluations answer the question that whether QoS target can be satisfied with the workload, resource and cost constraints stated in SLA. Benchmarking provides cloud application dependent analysis of the application behavior, for instance throughput of served requests, response time. Benchmarks are relatively costly and precise feasibility study usually imply large amount of benchmarks. In this chapter, our benchmark based SLA feasibility study method makes tradeoff between accuracy and costs. Limited amount of benchmarks are performed to achieve a workload-resource mapping model. This mapping model, as an intermediate of this benchmark based feasibility study process, will be used in our runtime control method which will be introduced in chapter 7.

Chapter 7

RCSREPRO: Runtime Control method based on Schedule, REactive and PROactive methods

7.1 Introduction

Cloud services provide their service consumers with the possibility of running their applications or tasks under optimized resource configuration. Over-provisioning, which means allocating more resources than actual needs, will lead to spend money to no avail. On the contrary, under-provisioning, which makes matter worse, means allocated resource is not sufficient to run the application with an acceptable level of QoS. The resource demand of an application is dynamic. The dynamic resource demand makes the optimized resource configuration changing. Choosing an optimized resource configuration from plentiful candidates according to VCAA and SLA is very challenging. This work can be performed either manually or automatically. In order to apply the runtime control on large scale application, it is important to automate the runtime control process.

We propose Runtime Control method based on Schedule, REactive and PROactive methods (RCSREPRO). RCSREPRO aims at automatically allocating suitable amount of resources at runtime. RCSREPRO is based on a continual improved workload resource mapping model which makes the runtime control more and more accurate with the cumulation of runtime workload and application knowledges.

This chapter is arranged as follows. The challenges, time granularities and runtime control system architecture based on MAPE-K will be discussed in section 7.2. RCSREPRO, our runtime control method, will be described in section 7.3. This chapter will be concluded in section 7.4.

7.2 Problem description

The goal of runtime control is satisfying SLA at runtime while avoiding over-provisioning as much as possible. The fundamental of runtime control is the SLA. Our method is generic, but we explain the runtime control problem based on PSLA in this section.

After the SLA feasibility study phase, the available resource and cost are able to serve the

possible workload with target QoS. To guarantee the SLA at runtime, the safest way is always deploying the maximal amount of resources. However, this strategy stand a good chance of vast over-provisioning.

In this section, the challenges of runtime control will be described in section 7.2.1. The time granularities of runtime control will be discussed in section 7.2.2. MAPE-K based runtime control system architecture will be described in section 7.2.3.

7.2.1 Challenges

The optimal runtime control method should always allocate just right resources to real needs which means the allocated resource can serve the runtime workload with QoS target and the resources are adequately used.

To allocate resources according to actual circumstances is a challenging work. The workload is always changing because of the fluctuating end user requests. The workload characteristic for a given time period is uncertain so that it is difficult to arrange suitable amount of resources before runtime. Runtime control decisions and activities need to be performed at real time according to the workload situation. The new resource allocation needs a period of time to take into effect. Therefore, the accuracy of workload prediction is crucial for arranging right amount of resources in advance. The diversity of workloads may need different resource configuration.

Since there are cost constraints and resources constraints which are evaluated for serving the target workload with required QoS defined in SLA, it stands a good chance that the workload beyond upper-bound of SLA request arrival rate can not be served. Admission control is necessary at the entrance of the application to deal with the workload beyond the upper-bound defined in SLA. For example, if the upper-bound of SLA request arrival rate is 1000 requests/s, when real request arrival rate is 2000 requests/s, only half of the requests will be served and rest of the requests will be discarded.

The mapping from workload to underneath resource configuration is a per application task. The experience of runtime control is application dependent, in another word, there is no universal way of mapping resources for any kinds of applications. This application dependent mapping knowledge can be either achieved through either analysing application implementation detail or accumulating of application performance under variance workload circumstances. However, it is possible that the implementation detail is not achievable for the party who performs runtime control. The behaviour of application can also be unknown because the property knowledge is deficient for a new application. Therefore, benchmark can be an effective mean of obtaining application behaviours under different workload.

The cloud environment can also be not stable because of the resource competition among VMs located on the same host. The unstable cloud environment based on virtualization technology makes the real usable resources lower than the declared amount of resources. It is more feasible to observe than predict from the IaaS consumer's point of view. Based on this fact, reactive runtime control is adoptable.

Above mentioned aspects, such as workload prediction, resource mapping and reactive resource allocation, will be described. To describe RCSREPRO method clearly and concisely, the following method description focus on a time duration which has no change of SLA terms. In other words, the QoS targets keeps still which means the method description is suitable of ensuring one GuaranteeTerm or GuaranteeTerms with same ServiceLevelObjectives. As the time goes on, GuaranteeTerms may shift from one to another and QoS target may changes. In

this case, the method needs to be reconfigured and the continual improvement process needs to be started from all over again. For example the MFC model should be changed because the reference QoS of benchmark activities for establishing the initial MFC model are changed.

In this chapter, MFC refers to the MFC value for adopting one instance on one stage. The MFC model is initially the same as the MFC table achieved after applying SLA feasibility study.

7.2.2 Time granularities

Our runtime control method aims at provisioning resources as meticulous as possible. To achieve this goal, scaling actions should be performed every-time when a QoS metric or a performance indicator is available so that the scaling decisions can be made correspondingly. These metrics or indicators are usually provided by monitoring systems. Depending on the time granularity of monitoring system, these values can be available within a very short time duration. For example, every 1 minute, performance metrics can be observed through AWS CloudWatch service [Ser15] with an extra fee. However, one VM usually takes at least 5 minutes to boot up. The application system can also need several minutes to adapt to the new resource configuration. So, the time interval between two scaling actions should be at least the total time of booting up VMs and adapting to new resource allocation. It's to avoid the scaling actions are too frequently and new resource configuration can not achieve the goal of scaling actions.

In some IaaSs, VMs are charged hourly. Due to this fact, " smart kill" is proposed by some researchers [LBMAL12]. "Smart kill" means never stopping a VM before its end of a charge unit. "Smart kill" is considerable in practice when adopting IaaS services provided by providers like AWS at this moment. However, there are already existing IaaS providers who charge their resources by second [Qin15] and these services show superiority on saving maintenance costs.

In our method, we assume that resources billing time unit is smaller than the VM boot up time to make our method appropriate for advanced billing system.

Therefore we don't need to take "Smart kill" problem into consideration. Our scaling action time interval is at least the total time of booting up VMs and adapting to new resource allocation. In RightScale's technical report, this sum up time is usually between 10 and 15 minutes.

Our runtime control method is a proactive method based on Workload Classification and Forecasting (WCF) [HHKA14]. WCF can select the most accurate forecasting method from several candidate time series analysis models dynamically. Before each scaling action, WCF will be performed to predict the future maximal request arrival rate of the workload. To make sure the time needed by WCF to perform forecasting is stable under 1 minute, the historical data used to forecast time interval should be at most 200 data. However, most of the time series analysis methods require at least 3 periods data to detect seasonal characteristic. So, at most $200 \div 3 \approx 66$ data per period can be collected. Depending on the nature of the workload, one period could last for various time durations. For different length of period, the minimum time interval between two data is different. The longer period is, the bigger data time interval will be. Human behaviour are the fundamental of changing workload for applications which interact with human users. In this case, one period can last for one day, one week, one month or even one year. We take one day period for example to explain the minimum data time interval required to get forecasting result with in 1 minute. 66 data per 1 day period means time interval between two data should be at least 21.8 minutes. WCF requires that the forecasting can be performed at least when one new data is observed. That is to say, the smallest time interval between two

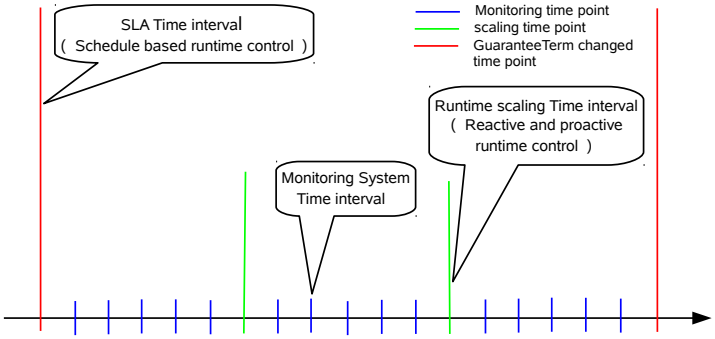

Figure 7.1. Runtime control time intervals.

forecasting should be at least the time interval between two observed data. Since forecasting needs to be performed before each scaling action, the time interval for scaling actions should be at least the minimal required forecasting time interval. In this day period example, the scaling time interval should be at least 21.8 minutes. However, if scaling actions perform at each time that an observed data is available for workload forecasting, the effect of scaling actions may not be able to be seen because the time between two scaling actions is too short. Therefore, the time interval for scaling actions should also be bigger than or equals to the total time of booting up VMs plus adapting to new resource allocation. This is similar to the idea of "Cooling down" time introduced in section 4.4.2. The "Cooling down" time can be various. We takes a 15 minutes "Cooling down" for example. Therefore, the scaling time interval is at least 21.8 minutes in this example. In the other word, every 21.8 minutes, the resource allocation will be evaluated and the scaling actions will be performed in this one day period example.

However, when period is remarkable higher than one day, the scaling time interval cloud be too long to forecast and reallocate resource to ensure SLA effectively. In this case, more fine granularity level of runtime control need to be performed. For example we can do workload prediction between two WCF based forecasting. However, this part of work will be a consideration of our future work. In this thesis, we take the widely applicable one day period and the 21.8 minutes scaling time interval for example to explain our method.

Figure 7.1 describes 3 different time scales. Biggest time scale is defined according to Guarantee terms of SLA. Middle size time scale is defined according to the forecasting frequency. Small size time scale is defined according to the time interval required for performing scaling actions.

7.2.3 Runtime control system architecture

Runtime control system is an autonomic computing system which has the ability of self-managing. The runtime control system follows the MAPE-K 4.3.1 control loop which was firstly introduced by IBM. In this part, the runtime control system architecture will be introduced. The introduction of runtime control system architecture is organized by Monitor, Analyze, Plan, Execute and Knowledge.

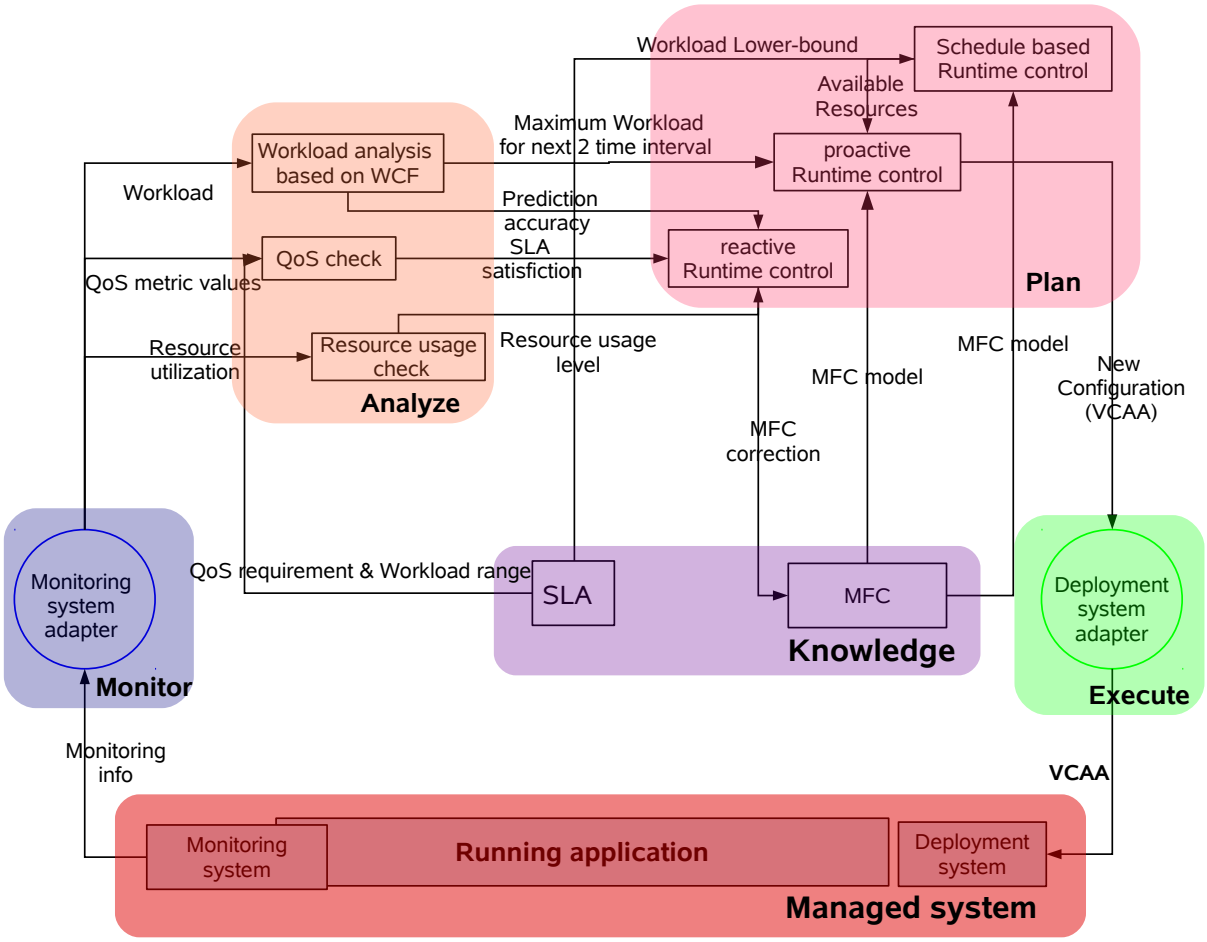

Figure 7.2. Runtime control system architecture.

Monitor

As demonstrated in figure 7.2, the part in blue background is Monitor. Monitor gathers informations from managed resources. For runtime control system, monitoring services is the source of Monitor. resource utilizations informations may be collected from monitoring services, be analysed and later be used by reactive runtime controller to adjust the allocated amount of resources. QoS metrics, such as response time, can also be collected through monitoring services and be used by proactive runtime controllers to judge the current configuration can serve workload with satisfactory QoS or not. The mapping model from workload to underneath workload requirement is continuously improved according to this judgement. The workload situation, typically request arrival rate, will be collected and used as the input of the workload forecasting component which is based on WCF in our case. Monitor component provides necessary informations about the underlying resources.

Analyze

As demonstrated in figure 7.2, the part in orange background is Analyze. Analyze receives the observation values from Monitor and perform analysis. Resource utilizations are compared with the predefined experience based threshold. Resource allocation message will be passed to reactive runtime controller of Plan. Workload request arrival rates will be analysed by WCF component to predict the workload situation of the nearest two time interval. The workload prediction results will be further used by proactive runtime controller of Plan to arrange suitable amount of resources. The QoS target metrics will be analysed according to both SLA and resource utilization to give the judgement that with allocated resources are sufficient without redundancy or not. This judgement will be further used by reactive runtime controller to improve the workloads and resources mapping model. With the accumulation of runtime workload and resource allocation observations, this mapping model will be more and more accuracy. Therefore, the accuracy of runtime control system will be continuously improved.

Plan

As demonstrated in figure 7.2, the part in pink background is Plan. Plan takes the results of the Analyze part and makes the decision, which is resource provisioning decision, as the input of Execute. Execution plans are made also based on the understanding of the characteristics of application. There are three components in Plan including schedule based runtime controller, reactive runtime controller and proactive runtime controller. These three controllers correspond to three kinds of runtime control methods adopted in RCSREPRO. Schedule based runtime controller performs resource reservation according to workload defined in SLA Reactive runtime controller update the workload-resource mapping model which will be used by proactive runtime controller. Proactive runtime controller makes the final resource provisioning decisions based on workload predicted by WCF and workload-resource mapping model continuously updated by reactive runtime controller.

The scaling decisions from all three controllers are aggregated and expressed formally as the input of Execute. Proactive runtime controller takes the workload prediction informations as the input and check at the MFC model to get the possible resource configuration. Reactive runtime controller improves the proactive runtime control decision by taking how busy the current system is into consideration. The conclusion of system is busy or not is given by Analyze.

When detected resource utilization is beyond the threshold, it means that the system is busy and under high level of pressure to serve requests with required QoS. resource utilization threshold is regarded as the indicator that the allocated resource well-matched the real need or not. The configuration of resource utilization thresholds are empirical tasks. In RCSREPRO, resource utilization thresholds are configurable and predefined. If the system is busy, certain proportion of extra resources will be assigned indirectly through the correction of MFC model and as a consequence of proactive runtime control. The schedule based runtime controller is only in action at the very beginning of a stable SLA period. Schedule based runtime control is typically seen as a manually task before runtime. Resource reservation is according to the lower-bound of workload request arrival rate defined in SLA. Based on the assumption of only same flavor can be used on one stage, the schedule based runtime controller will only be used on a specific stage where only one flavor is available.

Execute

As demonstrated in figure 7.2, the part in green background is Execute. Execute takes the formally described Plan result as the input. This formally described Plan result includes information about the new resource configuration of the application. Execute takes charge of realizing the decision of Plan. Execute is the one who deals with the underling infrastructure. For example, in the context of the OpenCloudware project, this result is described in OVF++ file [SEDP+13], which contains the VCAA informations, and VAMP reconfigures the underneath resource allocation of the application.

Knowledge

As demonstrated in figure 7.2, the part in purple background is Knowledge. All three runtime controllers made their scaling decisions based on the understanding of workload and application runtime characteristics which are named knowledge. The accuracy of RCSREPRO relies on the precision of the knowledge. The knowledges are more close to the effective situation, the more accurate RCSREPRO will be. Knowledge takes charge of organising, managing and continuously improving the knowledge.

There are two kinds of knowledge including the mapping model from workload to resource configurations and SLA constraints.

Mapping model from workload to resource configurations is expressed as MFC. MFC is the foundation and intermediates of SLA feasibility study. MFC is the core of knowledge. MFC is further used in RCSREPRO to allocate resources according to workload request arrival rate accordingly. The initial version of MFC is based on under-estimation of the capability of resources. MFC is more and more close to the real capable workload request arrival rate as a consequence of updating the mapping model with runtime observations.

The schedule based controller takes the lower-bound of request arrival rate defined in SLA. However, it is possible that application owner may set an extremely small lower-bound to make sure that the QoS can be guaranteed for a wide range of workload situations. One extreme situation is setting 0 requests/s as the lower-bound. In this case, schedule based controller is useless.

QoS targets, which is defined in SLA, are the yardstick of runtime control. QoS targets is used by the reactive controller to judge the allocated resource is sufficient or not. The judgement will be further be used to update the MFC model.

7.3 RCSREPRO runtime control method description

Our RCSREPRO runtime control method explores what the optimized resource configuration is for current application under different workload. These benchmarks have been performed during SLA negotiation as described in section 6.3. With the benchmarked and continuing improved knowledge, RCSREPRO can decide when to reallocate on which resources and the required amount of resources. As the name suggested, our runtime control methods are based on schedule, reactive and proactive based runtime control methods. Schedule based runtime control is performed at the beginning of each SLA Time interval as described in figure 7.1. Reactive and proactive based runtime control methods are performed at the beginning of each Runtime scaling Time interval as described in figure 7.1. At the beginning of each Runtime scaling Time interval, reactive method is firstly performed to correct the workload resource mapping model, then the proactive runtime control method will allocate resources according to the modified mapping model.

In this section, schedule based runtime control part of RCSREPRO will be described in section 7.3.1. Proactive runtime control part of RCSREPRO will be described in section 7.3.2. MFC based Mapping model can be continuously improved in RCSREPRO. This improvement is in a reactive way. The MFC improvement method will be introduced in section 7.3.3.

7.3.1 Schedule based runtime control

Schedule based runtime control is performed for one SLA stable duration, which means the attached SLA has no change during the time between two schedule based runtime control activities. Each schedule based runtime control will be validated at the beginning of a SLA period.

Schedule based runtime control method arranges computing resources according to SLA requirements and workload situations before runtime. When the workload characteristic can be known before runtime, schedule based runtime control method has remarkable advantage. It is because the public IaaS providers usually provide lower price to buy in advance resources. Schedule based runtime control method typically decides and allocates the right amount of resources based on the experience and knowledge about the application. However, this knowledge is difficult to be rich enough for performing accurate schedule based runtime control. More over, when there is unknown changes in the system, the application can not adapt to unexpected situations. Therefore, in our runtime control method RCSREPRO, we will adopt schedule based runtime control method to provision the minimal required amount of resources because of the lower buy in advance price. The fluctuating workload and unexpected situations will be processed by reactive and proactive part of RCSREPRO which will be explained later in this chapter. Comparing to manage application with proactive and reactive runtime control method, schedule based runtime control requires enough knowledges to reserve suitable amount of resources. Allocating right amount of resources to maintain QoS requires two kinds of knowledges including workload knowledge and the resource allocation knowledges.

When applying schedule based runtime control, the workload knowledges may be not sufficient to give a more accurate minimal workload request arrival rate. Therefore, we arrange resources according to the lower-bound of workload defined in SLA.

Resource allocation knowledges are expressed as MFC model in RCSREPRO. MFC is continuously developed to improve the accuracy of resource allocation. However, when applying schedule based runtime control, the initial MFC derived from SLA feasibility study is not yet

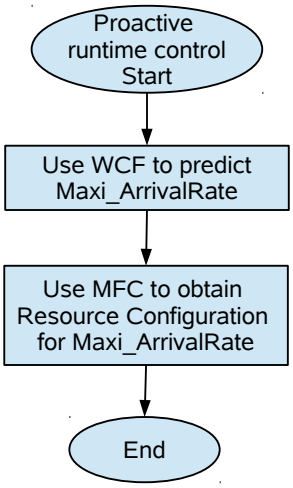

Figure 7.3. Proactive runtime control.

accurate enough to reserve resources accordingly. We have the assumption that only the VMs of same flavor can be adopted for one stage at the same time to perform the horizontal scaling. Time-shared resource reservation among several stages may be an intelligent but complicated strategy. To simplify the schedule based controller, we make resource reservation for each stage separately. Once one resource is reserved for a stage for a reservation duration, this resource is better be used during reservation duration to fully use the paid resources. Therefore, schedule based runtime control can not be available for one stage when several flavors can be chosen from. As explained before, when apply schedule based runtime control, it should have one available flavor on this stage and the MFC are all achieved through benchmark. Therefore, when schedule based runtime control is considerable for one stage, the MFC model for this stage is achieved through benchmark and they are seen as accurate enough.

7.3.2 Proactive runtime control method in RCSREPRO

As explained in figure 7.3, proactive runtime control method includes two steps. Firstly forecasting the future workload, then arrange suitable amount of resources for the predicted workload. In this section, workload forecasting method of RCSREPRO in section 7.3.2. The problems involved in resource provisioning according to forecasted workload will be introduced in section 7.3.2. The workload-resource mapping model based on MFC will be introduced in section 7.3.2.

Workload forecasting

Trend and season characteristics are seen as the inherent nature of time series. Different applications probably have different workload characteristics. For a specific application, the workload could also be evolving. Therefore, choosing the right model is not an once for all work when applying time series analysis and forecasting methods. Trend and season analysis result are crucial for the accuracy of time series forecasting. Trend and season are not necessary to be constant during the whole time series process in practice. In other words, as a dynamic stochas-

tic process, the trend and season are changing over time and one time series forecasting model probably can not be applied to the whole process. One specific model can only be applied within a period of time to achieve better accuracy. In order to automate runtime control process, it is necessary to be able to select the appropriate time series forecasting model at runtime automatically.

WCF [HHKA14] is one of the system to automate the selection of time series forecasting model based on a decision tree according to the current available historical data, the requirements of forecasting, the characteristic of time series forecasting models and the accuracy of forecasted result.

In [HHKA14], the experiment results show that WCF, as a sophisticated approach, can achieve better accuracy than applying one single approach such as extended exponential smoothing on real world workload. Therefore, we adopt WCF in our runtime control approach to continuously forecast workload request arrival rate for our proactive part of runtime control.

In WCF [HHKA14], 13 important characteristics of the workload are introduced to classify different kinds of workload with an acceptable calculation cost. WCF takes 9 existing time series forecasting methods into consideration. WCF summarizes the requirements, applicable situations, advantages and disadvantages of these 9 time series 4.4.3 forecasting methods and uses these characteristics to narrow the scope of choosing the suitable method online. WCF developed a decision tree based approach to automatically select an appropriate time series forecasting model from all nine candidates.

The selection of a suitable forecasting method is performed periodically. The selection is done according to characteristics of time series, such as the length of available historical data in time series, the customizable forecasting goals, such as the acceptable maximal computation time, and forecasting accuracy evaluation based on Mean Absolute Scaled Error (MASE). MASE is one of the metrics to evaluate the accuracy of forecasted result and its observations [HK06]. MASE is calculated according to formula 7.3.2.

$$MASE = mean\left(\left|\frac{forecastValue_t - observedValue_t}{\frac{1}{n-1}\sum_{i=2}^{n}|observedValue_i - observedValue_{i-1}|}\right|\right) \tag{7.1}$$

WCF forecasts according to several potential time series forecasting methods while the accuracy is calculated by MASE in parallel. One forecasting method is selected according to the MASE value as a feedback mechanism. Therefore the accuracy of WCF time series forecasting is improved compared to each method.

From forecasting to provisioning

When performing resource scaling actions, scaling up and scaling down actions take remarkable different amount of time to take the new resource configuration into effect. As mentioned before, scaling up actions usually may take a period of time to boot up VMs and get new configuration to perform stable. However, scaling down actions impact the system resource utilization and performance almost as soon as the scaling down command is given. The resource allocation strategy should take these differences of time into consideration.

In our runtime control method, we will allocate resources according to the maximal resource requirement for the time interval from now to two times of scaling time interval. The considerations of doing so will be explained by comparing three different resource allocation strategies

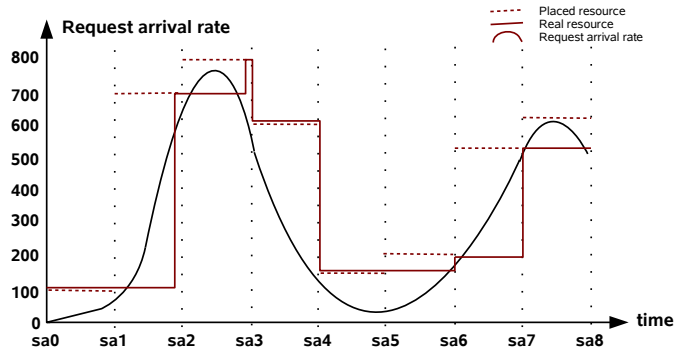

Figure 7.4. Provisioning resources for next one scaling time interval.

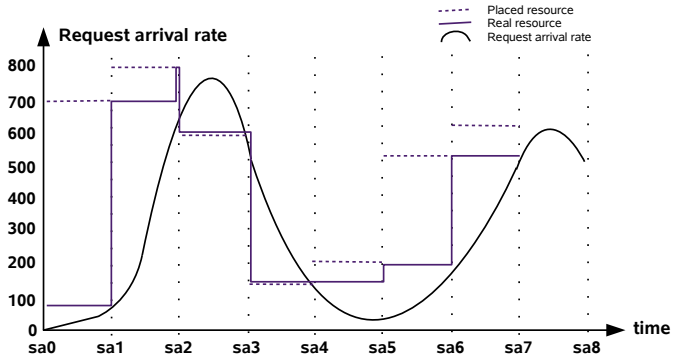

Figure 7.5. Provisioning resources for the scaling time interval after next.

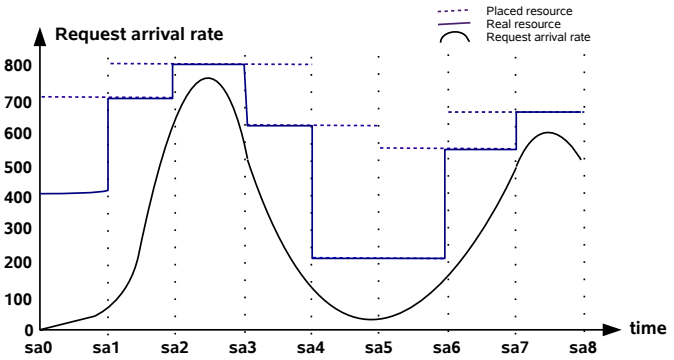

Figure 7.6. Provisioning resources for next two scaling time interval.

demonstrated in figure 7.4, 7.6 and 7.5. The resource provisioning situation can be disparate for different size of scaling time interval. To clearly explain the problem, we take a special example, which is the scaling up actions take almost the whole scaling time interval to be ready for the workload and the scaling down action can be immediately ready. In figure 7.4, 7.5 and 7.6, three different resource allocation strategies are demonstrated. In these figures, the X-axes are time, the Y-axes are workload request arrival rates, the curves are the workload request arrival rates, the dotted lines are the planned amount of resources according to capable request arrival rate of workload and the continuous lines are the real allocated amount of resources according to capable request arrival rate of workload.

Figure 7.4 demonstrates the resource provisioning situation when allocating resources according to the maximal request arrival rate of time interval from now to one scaling time interval later. When allocated amount of resources need to be decreased, the real available amount of resources reach planned amount resources immediately. So resource over-provisioning and under-provisioning will not be caused by resource deallocation. When allocated amount of resources need to be increased, the real available amount of resources are not able to reach planned amount resources. Increased amount of resources will be ready one step later than it's needed. So this strategy may lead to under-provisioning.

Figure 7.5 demonstrates the resource provisioning situation when allocate resources according to the maximal request arrival rate of time interval from one scaling time interval later to two times of scaling time interval later. When allocated amount of resources need to be increased, the real available amount of resources will reach planned amount resources just when the resource is needed. So resource over-provisioning and under-provisioning will not be caused. When allocated amount of resources need to be decreased, the real available amount of resources will be deallocated immediately even if the resources are still needed in current time interval. Therefore, decreased amount of resources will be one step before it's needed. So this strategy may lead to under-provisioning.

In figure 7.4, we can see that allocating resources according to the maximal request arrival rate of time interval from now to one scaling time interval later performs well when resources need to be deallocated but can lead to under-provisioning when the resources need to be increased. In figure 7.5, we can see that allocating resources according to the maximal request arrival rate of time interval from one scaling time interval later to two scaling time interval later performs well when resources need to be increased but can lead to under-provisioning when deallocation. There is one of the methods to combine the benefits of both. Resources should be increased according to workload condition from one scaling time interval later to two times of scaling time interval later. Resources should be decreased according to workload condition from now to one scaling time interval later. However, at a specific scaling time point, resource allocating and deallocating decisions can be in conflict. For example, when request arrival rate is low from now to one scaling time interval later and workload is high from one scaling time interval later to two times of scaling time interval later. Resources should be allocated according to the highest workload to avoid under-provisioning caused by long booting up and system adapting time. As demonstrated in figure 7.6, in our runtime control method, at each scaling time point sa_i, resources are allocated according to the maximal possible request arrival rate for next two scaling time interval that is time interval $[sa_i, sa_{i+2}]$.

It takes stably under 1 minute to forecast workload based on WCF. Therefore, each forecasting is performed at 1 minute before each scaling time point sa_i when the request arrival rate of the workload at time point sa_i is not yet available. At 1 minute before each scaling time

point sa_i, actually the workload condition for time interval $[sa_i - 1minute, sa_{i+3} - 1minute]$ is forecasted to include the workload prediction for time interval $[sa_i, sa_{i+2}]$ when scaling time interval is not significantly bigger than 1 minute. When scaling time interval is bigger enough, we still forecast the workload condition for time interval $[sa_i - 1minute, sa_{i+2} - 1minute]$ to approximately seen as the workload condition of time interval $[sa_i, sa_{i+2}]$.

MFC based resource mapping

RCSREPRO is a runtime control method based on MFC. The initial MFC model is established through benchmark before runtime. Benchmark is interesting for runtime control because the resource demanded by applications can't be summarized as a general knowledge. In other words, application behaviours under different workloads and different resource configurations depend on characteristics of the application itself. Applications can be diverse. When implementation details of the application is not available and the performance of the application is unknown as a freshly developed application, benchmark is a practical way of understanding the application behaviours under different workloads and its resource consumption. In this way, the runtime control system can determine the suitable condition for scaling actions to ensure the continuing satisfaction of SLA.

During SLA feasibility study, we achieved a table of MFC for one instance of each possible flavor on each stage. This MFC table tells us the maximal capable request arrival rate for adopting one instance on one specific stage. As explained in previous chapter 6.2.3, it is reasonable to assume that the MFC of adopting "n" instances of the same flavor on one stage equals to "n" times of MFC of one single instance of same flavor on the same stage. The maximal capable request arrival rate of the whole deployment of the application is the minimal multiple instance MFC among stages.

In a PSLA contract, the maximal number of instances and the minimal number of instances are also defined. At runtime, when we want to assign resources for a predicted maximal request arrival rate, we can search all possible deployments according to single instance MFC table and number of instances constraints, and then select one deployment. The selection can follow cost effective, balanced allocation of resources or fast scaling speed. Cost effective means to always choose the cheapest deployment. Balanced allocation of resources means the gap between maximal capable request arrival rates of each stage are small so that there is no obvious bottleneck in the deployment. In this way, the resources are not wasted. Fast scaling speed deployment means the time needed to deploy the new configuration is small. Which deployment to be selected is a multiple criteria decision making problem. The developed mechanism for solving this problem will be a part of our future work and is out of the discussion of this thesis.

Here we assume that fast scaling speed has the highest priority and horizontal scaling is faster than vertical scaling. That is to say we usually prefer the deployment which uses the same flavor as current one on each stage as long as the number of instances constraints are still satisfied. The priority of balanced allocation of resources principle comes secondly. That is to say, when the needed amount of instances is more than the $MaxNoI_s^i$ 6.5.3 defined in SLA for current flavor, the configuration which has the smallest waste of capable arrival rate on each stages will be adopted. The priority of cost effective comes last. That is to say, when several configurations have the same amount of wasted capability, one of them which has smallest cost will be adopted.

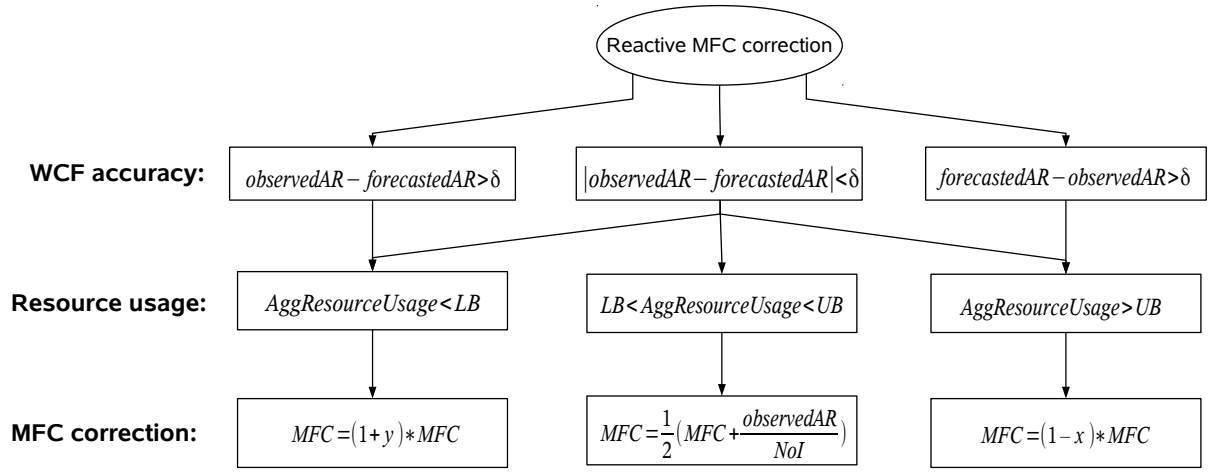

Figure 7.7. MFC model improvement based on reactive runtime control.

7.3.3 Mapping model improvement based on reactive runtime control in RCSREPRO

One primal method of runtime control is creating a load balancing server cluster and manually adding or removing servers. It is a typical practice when adopting IaaS. Nowadays, most IaaS providers provide PaaS level services to automate this process. These services are named as Auto-scaling services. Auto-Scaling services aims at automatically allocating and deallocating resources to ensure the application performance as well as avoid over-much. Auto-Scaling services can be helpful for lowering maintenance cost. When workload of the application significantly changes during runtime, Auto-Scaling service, which is a typical reactive runtime control method, is the universal choice in the industry.

Our runtime control method continuously improves the accuracy of MFC, which is our workload resource mapping model, based on reactive runtime control. Classical reactive runtime control is a mechanism based on threshold. In this subsection, how and why we adapt classical threshold based runtime scaling method in RCSREPRO will be introduced.

Why reactive runtime control in RCSREPRO

Theoretically, only proactive runtime control method should be able to allocate and deallocate right amount of resource to guarantee SLA and save cost as much as possible. This could be the truth when the prediction of the request arrival rate of the workload is extremely precise, the mapping model from workloads to underneath resources is perfectly fit the physical truth from the very beginning, the cloud environment keeps still for example VMs can achieve fixed amount of resources even during the resource competition among VMs located in the same host. Since none of these hypothesis can be easily realized in practice, we complete proactive runtime control method with reactive runtime control method. Reactive method is adopted to evaluate the effect of runtime control during the previous scaling time interval.

Comparing to the classical threshold based reactive runtime control method, we will not adopt the system cool down time. It is not necessary to set system cool down time because we already define the time duration between two scaling actions is at least the sum up of system

cool down time and system performance stable under the new configuration.

The continuous beyond threshold duration will not be adopted in initial version of RCSREPRO. It is not necessary to set the continuous beyond threshold duration to trigger the scaling action.

Compared to classical threshold based runtime control method, which simply estimates the future system usage amount will be the same as current situation, our method does not need this continuous beyond threshold duration to make the prediction reliable and to avoid the runtime control method too sensitive to the resource usage condition. Resources are assigned mainly according to root cause which is workload condition while performance metrics and threshold helps adjust the MFC model. MFC is improved step by step and system will not be too sensitive to the modification as long as the parameter x percent and y percent are carefully tuned. For example, defining x percent and y percent through benchmarks. The allocated resource for a specific workload request arrival rate is more and more fitted with the development of the MFC model.

Accuracy of workload forecasting

As demonstrated in figure 7.7, the WCF prediction is accurate or not based on the comparison of predicted request arrival rate ($forecastedAR$) and observed one ($observedAR$). If the differences between $forecastedAR$ and $observedAR$ is smaller than δ, the WCF prediction is seen as accurate. Otherwise, WCF prediction is not accurate. If WCF works with an acceptable accuracy, the resource utilization represent the matching degree of required resources and allocated resources and a series of MFC improvement activities can be performed for any resource usage situation. On the contrary, when the WCF prediction result is not accurate, MFC model can only be improved under certain circumstances.

Aggregated resource utilization and stable observations

When horizontal scaling is performed on one stage, more than one instances can be used on this stage. As mentioned before, the cloud environment is not stable and the real available resources for each instance of the same stage could be different. Therefore, it's possible that not all of these instances can reach the high level resource utilization at the same time. When only a handful of instances can reach the high level resource utilization, MFC can not be updated. It may caused by unstable cloud environment or the application hasn't been adapted to the new configuration.

Only the stable observations can be used to update the MFC. We define a stable observation as at least z percent of the instances on current stage have a resource utilization within a specific resource usage range. In figure 7.7, $AggResourceUsage$ is an aggregated value of resource utilization on all instances of one stage. $AggResourceUsage$ means at least z percent of the instances on this stage should satisfy the condition. Resource usage range is defined through lower-bound (LB) or upper-bound (UB). For example, $AggResourceUsage < LB$ means that at least z percent of the resource utilization is smaller than the lower-bound. A stable observation is defined through $AggResourceUsage$, LB and UB. z of $AggResourceUsage$, LB and UB are configurable parameters. For example, one stable observation $AggResourceUsage < 80(z = 90)$ means that at least 90% of the instances on current stage can reach resource utilization higher than 80%. When stable observations on one or more stages are achieved, MFC can be updated.

MFC value correction

In our runtime control method, the mapping from workload to resource configuration are mainly based on MFC. As mentioned before, the initial MFC is achieved through benchmark and linear under-estimation during the SLA feasibility study. The accuracy level of the MFC can influence the result of runtime control. Limited time and resources can be used during benchmark and our SLA feasibility study method itself can introduce wide range of under estimation on the capability of flavors. Initial MFCs achieved through SLA feasibility study need to be improved as a mapping model of runtime control method. With the accumulation of the runtime performance metrics and system behaviours observations, MFC values may be improved. Resource usage can be used to correct MFC model.

All stages of the application may not be able to reach the high level resource utilization at the same time. It's because there may existing one or more bottlenecks in the current configuration. The bottleneck stages can reach the high level resource utilization while the other stages may have lower resource utilization. The stages with acceptable resource utilization can be updated with new MFC based on observed request arrival rate. The stages with too high or too low level resource utilization will update its MFC according to two configurable parameters.

Even if our SLA feasibility study method is designed to eliminate the possibility of over-estimation of MFC, we still design the MFC development part of our runtime control method to be able to deal with MFC over-estimation because MFC could be over-corrected during the process of improving an under-estimated MFC.

- MFC correction when WCF forecasting is accurate

  As demonstrated in figure 7.7, if the resource utilization $AggResourceUsage$ exceeds the upper-bound threshold UB, for example CPU usage>80%, it means that there was an under-provisioning of resources. Never the less, a high level of resource utilization also means that there are still intense competitions of resources and probably certain amount of requests are still waiting in the queue to be served. Since these historical requests still need to be served during the coming scaling time interval and occupy the resources allocated for new requests, more resources will be allocated during current scaling action indirectly by decreasing MFC value. The incremental resources compared to the original planning is depending on the configuration of parameter x percent. The bigger x percent is, the larger increment of assigned resources per workload unit will be and vice versa. It is possible that current scaling action chooses another flavor than previous one. Therefore the increment of resources will not appear and under-provisioning of resources will trigger the above process again.

  If the resource utilization $AggResourceUsage$ is below the lower-bound threshold LB, for example CPU usage<50%, it means that there is over-provisioning of resources. Resources are wasted during the past time interval. Due to the feature of services, not used services can not be kept for future use. Resource utilization below the lower-bound threshold happens when predicated workload is much higher than real condition or the model used to do workload resource mapping under estimated the capability of resources. As long as the workload prediction accuracy is acceptable, we can get the assertion that MFC mapping model under-estimated the capability of this flavor for this stage. So, we will correct the MFC value by increasing y percent. y percent is configurable as x percent and higher y percent value will lead to substantially decreasing of resource allocation.

When resource utilization $AggResourceUsage$ is within an acceptable range (within $[LB, UB]$) and workload prediction is accurate, the observed workload and resource configuration can also be used to correct the MFC model. In our runtime control method, when resource utilization is in high level (but still within an acceptable range, for example 50% - 80%), we assume that the current flavor fully uses its capability on current stage. Then, we will use the current served request arrival rate to update the MFC of current flavor and stage. To be more specific, the new MFC value will be the mean value of old MFC value and the average request arrival rate on each instance. We will not always have request arrival rate monitoring informations for each instance. Our method is based on minimal requirements on request arrival rate monitoring informations which is the request arrival rate at the entrance of the application.

- MFC correction when WCF forecasting is not accurate

  The "number of instances" NoI is always assigned according to the predicted workload ($forecastedAR$). As demonstrated in figure 7.7, if $AggResourceUsage$ is too high, it indicates that the allocated resource is not enough because the real workload is heavier than expected or the MFC is over-estimated. If the $observedAR$ is lower enough than $forecastedAR$, the MFC value is too big to allocate enough instances accordingly. Therefore, MFC value should be decreased. If the $observedAR$ is bigger enough than $forecastedAR$, the accuracy of MFC value depending on the degree of $AggResourceUsage$ and differences level between $observedAR$ and $forecastedAR$. Therefore, no certain assertion can be made in this situation.

  Similarly, when $AggResourceUsage$ is too low and $observedAR$ is bigger enough than $forecastedAR$, it indicates that MFC value needs to be increased. When the $forecastedAR$ is not accurate, real request arrival rate $observedAR$ is either bigger or smaller than $forecastedAR$. However, the current MFC model can be either under-estimated or over-estimated. Therefore, the $AggResourceUsage$ could still be within the acceptable range. In this case, MFC value can also be updated by taking the mean value of current MFC and the average request arrival rate afforded by one instance.

7.4 Conclusion

In this chapter, we propose RCSREPRO, a Runtime Control method based on schedule, reactive and proactive runtime control methods. RCSREPRO allows us to benefit the scalability feature of cloud infrastructures and to automatically allocate and release resources according to changing needs. Besides making a resource provisioning schedule according to workload information listed in SLA before runtime, RCSREPRO also forecasts the fluctuating workload to achieve the detailed workload information at runtime and provision resources accordingly in a proactive way to have resources ready when they are needed. RCSREPRO predicts future workload by using WCF which is a dynamic workload forecasting tool based on time series. MFC, the workload-resource mapping model used in RCSREPRO, is initially derived from benchmarks in SLA feasibility study described in section 6.3 and continuously improved in a feedback way at runtime to increase the accuracy of the control.

Chapter 8

Experiments

8.1 Goals of the experiment

As described in chapter 7, our runtime control method RCSREPRO is based on both proactive runtime control and reactive runtime control. In the proactive runtime control part of our method, there are two important parts, workload prediction and workload-resource mapping. We continuously improve our workload resource mapping model in the reactive runtime control part of our method. The workload prediction relies on WCF. We trust that WCF can predict the workload with a high level of accuracy and we assume that the accuracy is 100%. So we will not install WCF in our experiment. We will only check our workload-resource mapping model part of our method. To be more precise, the goal of our experiment is to prove that MFC can be used as a workload-resource mapping model and MFC can be improved according to our runtime control method.

8.1.1 Abstract application

As described in chapter 4, several components can be deployed in the same VM and we considering scaling activities in VM level. To perform benchmarks in our experiment, it is necessary to have detailed knowledge about the system under test. To be more precise, we need to know how does one request use resources of the system under test.

We developed "Rubberband", an elastic application, to have more information and control on the system under test. "Rubberband" follows the Service Oriented Architecture. When one request is injected into "Rubberband", each component will be loaded (or not) according to a configurable request. Compared to the other component based application, one VM has only one component of "Rubberband" deployed on it. Therefore, "Rubberband" is actually an abstract of stage.

Rubberband request structure

Since "Rubberband" is an abstract of stage, request of "Rubberband" consists informations that which stages to load, the sequence of loading stages and the level of resource consuming. These informations are expressed in a recursive way.

```
<Request> = <Stage_Name><Commands>{Subsequent_Request_Order}{
    Subsequent_Requests}
```

Listing 8.1. Request structure of Rubberband.

<Stage_Name> indicates which stage should be loaded through the IP address of load balancer.

<Commands> is composed of several <Command>. One <Command> indicates one kind of resource to load on, such as CPU, memory, I/O or network as while as the level of loading this resource. The level of loading resource is expressed through one integer number from 0 to unlimited. Taking CPU for example, this integer indicates how many times of a factor calculation will be performed to charge the CPU. This factorial calculation is done through for loop and the used amount of RAM will not increase. When the integer is 0, no factorial calculation will be performed. When the integer is 100, the factorial calculation will be performed 100 times and the request response time will be longer. The CPU utilization can change with the dynamic request arrival rate.

{Subsequent_Request_Order} indicates the time sequence of executing the subsequent requests. There are two kinds of order, sequential and parallel. Sequential means executing the requests listed in {Subsequent_Requests} one after another. Parallel means executing the requests listed in {Subsequent_Requests} all at the same time.

{Subsequent_Requests} is composed of several <Request>. Each <Request> listed in {Subsequent_Requests} still have the structure defined for current request. Therefore, <Request> is defined in a recursive way and the end of one request definition is a <request> without {Subsequent_Requests}.

Rubberband architecture

"Rubberband" is a software which can be constructed freely. "Rubberband" can be constructed as a sequential application like typical 3 tier web application. "Rubberband" can also be constructed as a no circuit diagram which means the sequence of loading stages can be both sequential and parallel. This makes "Rubberband" can be used to test our method under complicated application environment than just sequential one. Figure 8.1 is an example of "Rubberband" application architecture. In this example, there are six stages, from stage0 to stage5. Stage0, stage4 and stage5 have only one VM on each stage. Stage2 and stage3 have 2 VMs on each stage. There are 3 VMs on stage1. One request can load one or several stages but not necessary from stage0. When there are more than one VM in one stage, a load balancer will be needed, for example LB1, LB2 and LB3. Load balancers will distribute requests equally among VMs belonging to the same stage.

8.2 Experiment environment

8.2.1 Cloud environment based on OpenStack

We build our cloud infrastructure based on OpenStack [Jac12]. OpenStack is a software stack which aims at managing large amount of compute, storage and network resources and providing virtualized resources. We install OpenStack on one single node. Ceilometer [Wik15a] aims at collecting physical and virtual resource utilization in cloud environment.

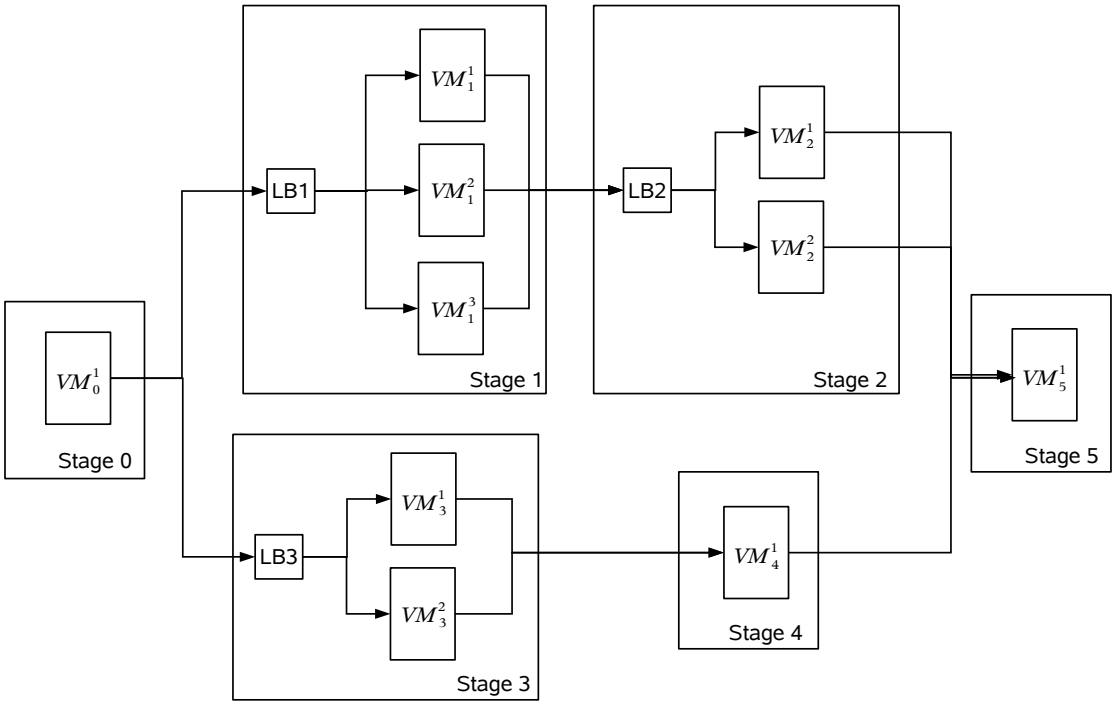

Figure 8.1. One example of Rubberband architecture.

8.2.2 Benchmark environment

We use CLIF and SelfBench to perform our benchmarks. For example, as depicted in figure 8.2, we will use SelfBench to benchmark the Rubberband with two stages. There is one VM on each stage. To perform SelfBench, one probe is installed on each VM of Rubberband. Besides the VMs used to deploy Rubberband, we also installed one injector VM and one Benchmark controller. The injector injects requests to Rubberband. Both injector and probes on Rubberband are under the control of Benchmark Controller which takes charge of the whole benchmark process. We also install Rubberband controller to automate the process of deploying Rubberband and the configuring part of the CLIF based benchmarking environment.

8.3 Experiment design

Our experiment consists of three steps. Through the first step, we will get the real MFC value of each stage through "SelfBench". In the second step, we will do "CLIF" benchmark to prove that our workload-resource mapping model works correctly as long as our MFC values are accurate. In the last step, we will prove MFC, which is the core of our workload-resource mapping model, can be improved with the reactive part of our runtime control method.

8.3.1 Rubberband and Request construct

We construct a simple Rubberband to test our method for the first step. Our Rubberband application consists of two stages connected sequentially. Each stage can have one or more VMs, therefore there is one load balancer for each stage. The minimum number of instance is 1.

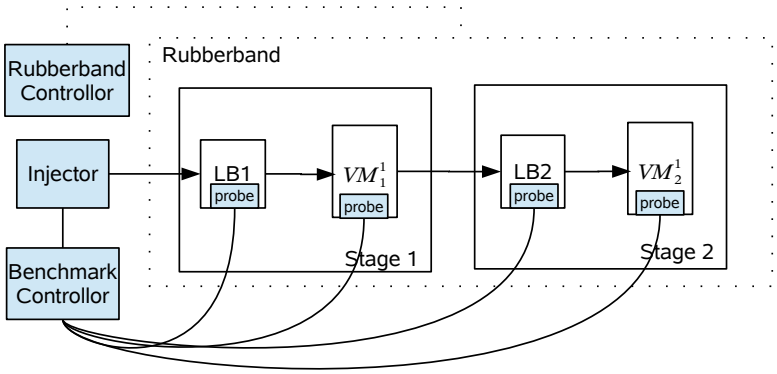

Figure 8.2. The deployment architecture for performing SelfBench on Rubberband.

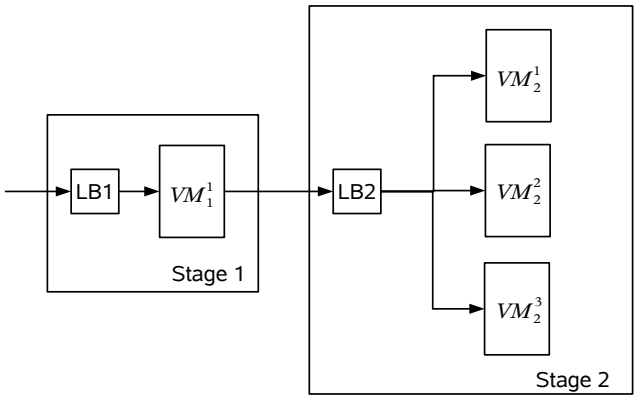

Figure 8.3. The first Rubberband under SelfBench test.

We have only one flavor for each stage. We have only one kind of request. Our request load two stages sequentially. One request will first load stage1 by performing one time of factorial calculation $2000!$. Then the second stage will be loaded by performing three times of factorial calculation $2000!$.

8.3.2 What is the real MFC?

We will get the real MFC through a set of Selfbench benchmark. To get one real MFC, we need to perform one SelfBench. This SelfBench should have the minimum number of instance on the MFC related stage and plenty instances on other stages to make sure that the resource on MFC related stage can be fully used.

Since we have two stages, two times of SelfBench will be needed. As demonstrated in figure 8.3, to fully use the resource of stage1, 3 instances are needed on stage2 so that stage1 can get saturated before stage2 and the SelfBench can be seen as real MFC_1. Similarly, as demonstrated in figure 8.4, only one VM is needed on stage1 to get the real MFC_2 through SelfBench.

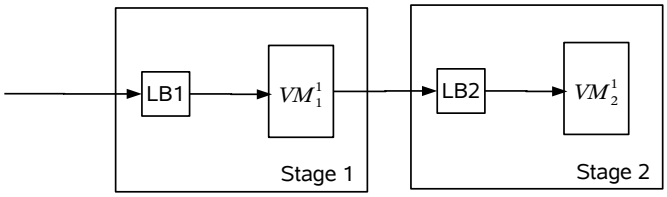

Figure 8.4. The second Rubberband under SelfBench test.

8.3.3 Whether or not our workload-resource mapping model works correctly?

The most accurate MFCs(MFC_1 and MFC_2) have been achieved through benchmark based on SelfBench. In this step, we will check that right amount of resources could be allocated with the accurate MFC, MFC_1 and MFC_2. We use CLIF to perform 3 benchmarks. Each benchmark will inject workload with constant request arrival rate in to its optimized Rubberband deployment. These deployments are defined according to the real MFC values(MFC_1 and MFC_2). These three different request arrival rate value are defined according to MFC_1 and MFC_2. Therefore, we can prove that our workload-resource mapping model works appropriately under request arrival rate lower than the minimum value of MFC_1 and MFC_2, bigger than the maximum value of MFC_1 and MFC_2 and between MFC_1 and MFC_2. Finally, it is proved that right amount of resources can be allocated according to our MFC based mapping model, if the resource utilizations and request response time are both within an acceptable range during these three CLIF benchmark.

8.3.4 Whether or not MFC can be improved by RCSREPRO?

The initial Workload-resource mapping model is derived from benchmark. In practice, this initial MFC could be not accurate enough. Therefore, our runtime control method can improve the initial MFC value in a reactive way. In this step, we will check that the initial imprecise MFC value can be improved with our runtime control method RCSREPRO when the workload prediction is accurate.

As explained in chapter 7, the MFC values are improved in a reactive way based on monitoring information. We use the monitoring information provided by Ceilometer. We initially set $MFC_1' < MFC_1$ and $MFC_2' > MFC_2$. Then, we allocate resources according to MFC_1' and MFC_2'. We check the resource utilization of VMs on each stage. We set x and y, the parameters to improve MFC, both as 10% which means the MFC value will be increased (or decreased) by 10% when the average resource utilization is lower (or higher) than the lower-bound (upper-bound). MFC values are improved according to reactive part of RCSREPRO described in section 7.3.3. Finally, it is proved that imprecise MFC value can be improved with RCSREPRO if MFC_1' closes to MFC_1 and MFC_2' closes to MFC_2.

Chapter 9

Conclusions and future works

Cloud, as a booming form of IT infrastructure, facilitates the provisioning of IT resources. IT resources are moved from local to internet environment. Cloud clients deploy their application and store their data in remote data center. Cloud services provide their clients with large scale of computing services, storage services or network resources. These resources are elastic and customizable. Cloud clients rent and use these resources in a "pay as you go" way which greatly reduce the IT investment risk and management cost. Cloud computing has aroused wide attention among academia and industry. Researches around QoS management, SLA, PaaS and runtime control have attracted more and more attention.

9.1 Conclusions

In this thesis, we focus on the QoS management problems for PaaS. To be more specific, the goal of this thesis is to manage the QoS of application running in cloud infrastructure.

In chapter 3, we introduced SLA standards and SLA management. SLA, as a contract which specify the service and QoS criterion, is indispensability for service oriented business. PaaS providers, which take charge of application's QoS, need to consider the on-demand resources and dynamic resource requirements and describe these characters in their PaaS level SLA. One PaaS level SLA contract should be customize for each contracted application since QoS is an application dependent topic. However, existing SLA description languages are not suitable for PaaS level SLA. Therefore, in chapter 5 of this thesis, we proposed PSLA which is a generic PaaS level SLA based on WS-Agreement skeleton. PSLA was implemented in XML to make PSLA based SLA contract machine readable for automated SLA management.

Benchmarking provides cloud application dependent analysis of the application behaviour, for instance throughput of served requests, response time. Through benchmark, the feasibility of a SLA can be known. In other words, the question that whether QoS target can be satisfied with the workload, resource and cost constraints stated in SLA can be answered. We described our benchmark based SLA feasibility study method in chapter 6. Benchmarks are relatively costly and precise feasibility study usually imply large amount of benchmarks. Our benchmark based SLA feasibility study method makes tradeoff between accuracy and costs. Limited amount of benchmarks are performed to achieve a workload-resource mapping model. This mapping model, as an intermediate of this benchmark based feasibility study process, is used in our runtime control method which is introduced in chapter 7.

As described in chapter 4, runtime control aims at provisioning resources for running applications to satisfy the terms defined in SLA. Existing runtime control methods are in a schedule, reactive or proactive way, based on rule, control theory, queuing theory, reinforcement learning or time series. Schedule based runtime control methods can not adapt to fluctuating workload. However, reactive runtime control methods require great efforts and knowledges about the running application behaviour from application owners, while proactive runtime control methods have strong constraints which block these methods from applying in practice. Therefore, we proposed RCSREPRO, a runtime control method, to release the burden of application owners through a set of benchmarking in chapter 7. The results of benchmarks provide an initial proactive runtime control model to make our method more practical.

RCSREPRO is a Runtime Control method based on schedule, reactive and proactive runtime control methods. RCSREPRO allows us to benefit the scalability feature of cloud infrastructures and to automatically allocate and release resources according to changing needs. Besides making a resource provisioning schedule according to workload information listed in SLA before runtime, RCSREPRO also forecasts the fluctuating workload to achieve the detailed workload information at runtime and provision resources accordingly in a proactive way to have resources ready when they are needed. RCSREPRO predicts future workload by using WCF 7.3.2 which is a dynamic workload forecasting tool based on time series. MFC, the workload-resource mapping model used in RCSREPRO, is initially derived from benchmarks in SLA feasibility study described in section 6.3 and continuously improved in a feedback way at runtime 7.3.3 to increase the accuracy of the control.

9.2 Future works

In PSLA, the workload composed of multiple kinds of requests can be defined through the definition of several kind of request. For the definition of one kind of request, both request arrival rate and the variation of request arrival rate should be defined. However, understanding the characteristic of an application which serves many kinds of requests through benchmark is a challenging task. Each kind of requests probably requires different kinds and amount of resources. When multiple kinds of requests are mixed and contend for resources together, the QoS and system behaviours affected by multiple factors such as the composition of requests. The workload-resource mapping model for multiple kinds of requests should takes these factors into consideration. In our current workload-resource mapping model, only request arrival rate is considered. Our SLA feasibility study method and runtime control method RCSREPRO can't be applied on workload consisted of multiple request arrival rate. In the next phases of our research, we will improve our SLA feasibility study method and runtime control method RCSREPRO to adapt to workload consisted of multiple requests and also introduce benchmarks about the variation of request arrival rate.

List of publications

Workshop

- Ge Li, Frédéric Pourraz and Patrice Moreaux, PSLA: a PaaS level SLA description language, in Cloud Engineering(IC2E), 2014 IEEE International Conference on, page 452-457. IEEE, 2014.

International conferences

- Ge Li, Frédéric Pourraz and Patrice Moreaux, A benchmark based SLA feasibility study method for platform as a service. CLOUD COMPUTING 2015, page 175, 2015.

Appendix A

PSLA XML Schema

PSLA is expressed in Extensible Markup Language (XML). A XML schema language is indispensable to stipulate the PSLA contracts. We choose XML Schema Definition (XSD) to express these rules. The XML Schema is listed in this appendix.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema elementFormDefault="qualified"
    targetNamespace="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement"
    xmlns="http://www.w3.org/2001/XMLSchema">
  <!-- LISTIC extensions -->

  <!-- Context extension -->
  <include schemaLocation="./listic_ocw_ContextExtension.xsd" />

  <!-- ServiceDescriptionTerm extension -->
  <include schemaLocation="./listic_ocw_ServiceDescriptionTermExtension.xsd
    " />

  <!-- ServiceReference extension -->
  <include schemaLocation="./listic_ocw_ServiceReferenceExtension.xsd" />

  <!-- QualifyingCondition extension -->
  <include schemaLocation="./listic_ocw_QualifyingConditionExtension.xsd"
    />

  <!-- CustomServiceLevel extension -->
  <include schemaLocation="./listic_ocw_CustomServiceLevelExtension.xsd" />
</schema>

<?xml version="1.0" encoding="UTF-8"?>
<schema elementFormDefault="qualified"
    targetNamespace="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement">

  <!-- LISTIC Context extension -->

  <element name="ContextExtension" type="ocw:ContextExtensionType" />
```

```
  <complexType name="ContextExtensionType">
    <sequence>
      <element name="OVFLocation" type="anyURI"/>
    </sequence>
  </complexType>
</schema>

<?xml version="1.0" encoding="UTF-8"?>
<schema elementFormDefault="qualified"
    targetNamespace="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement">

  <!-- LISTIC shared types -->
  <include schemaLocation="./listic_ocw_SharedTypes.xsd" />

  <!-- LISTIC ServiceDescriptionTerm extension -->
  <element name="ServiceDescriptionTermExtension" type="ocw:
    ServiceDescriptionTermExtensionType" />

  <complexType name="ServiceDescriptionTermExtensionType">
    <choice maxOccurs="unbounded">
      <element name="Metric" type="ocw:MetricType" />
      <element name="InputData" type="ocw:InputDataType" />
      <element name="Resource" type="ocw:ResourceType"/>
    </choice>
  </complexType>

  <!-- Metric -->
  <complexType name="MetricType">
    <complexContent>
      <extension base="ocw:NamedElement">
        <sequence>
          <element name="Unit" type="ocw:UnitType"/>
          <element name="Type" type="ocw:Type" />
          <element name="ValueGenerateBy" type="ocw:ValueGenerateByType" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="UnitType">
    <choice>
      <element name="BuiltInUnit">
        <simpleType>
          <restriction base="string">
            <enumeration value="second" />
            <enumeration value="minute" />
            <enumeration value="hour" />
            <enumeration value="percent" />
            <enumeration value="number" />
            <enumeration value="Euro" />
            <enumeration value="USD" />
```

```
            <enumeration value="Pound" />
            <enumeration value="requests/s"/>
            <enumeration value="MIPS"/>
            <enumeration value="Mo"/>
            <enumeration value="Go"/>
            <enumeration value="To"/>
            <enumeration value="Mb/s"/>
            <enumeration value="Gb/s"/>
          </restriction>
        </simpleType>
      </element>
      <element name="UserDefinedUnit">
        <complexType>
          <sequence>
            <element name="Description" type="string" />
          </sequence>
          <attribute name="Name" type="string" use="required"/>
        </complexType>
      </element>
    </choice>
  </complexType>

  <simpleType name="Type">
    <restriction base="string">
      <enumeration value="boolean" />
      <enumeration value="decimal" />
      <enumeration value="float" />
      <enumeration value="double" />
      <enumeration value="short" />
      <enumeration value="integer" />
      <enumeration value="long" />
      <enumeration value="negativeInteger" />
      <enumeration value="positiveInteger" />
      <enumeration value="unsignedLong" />
      <enumeration value="unsignedInteger" />
      <enumeration value="unsignedShort" />
    </restriction>
  </simpleType>

  <complexType name="ValueGenerateByType">
    <sequence>
      <!-- Should be an IDREF, but WS-Agreement ServiceReference's Name
    attribute is define as a string and not an ID -->
      <element name="ServiceReferenceName" type="string" />
      <element minOccurs="0" name="ServiceReferenceOperation" type="string"
    />
      <element name="Schedule" type="duration" />
    </sequence>
  </complexType>

  <!-- InputData -->
  <complexType name="InputDataType">
    <complexContent>
      <extension base="ocw:NamedElement">
        <sequence>
          <!-- A scenario where the request is described - a Clif scenario
```

```
    for example -->
          <element name="Scenario" type="anyURI" />
          <element name="Description" minOccurs="0" type="string" />
          <!-- Can be extended -->
          <any minOccurs="0" namespace="#any" processContents="strict" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Resource -->
  <complexType name="ResourceType">
    <complexContent>
      <extension base="ocw:NamedElement">
        <sequence>
          <element name="XPath" type="string" />
          <!-- Can be extended -->
          <any minOccurs="0" namespace="#any" processContents="strict" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>

<?xml version="1.0" encoding="UTF-8"?>
<schema elementFormDefault="qualified"
    targetNamespace="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement">

  <!-- LISTIC ServiceReference extension -->
  <element name="ServiceReferenceExtension" type="ocw:
    ServiceReferenceExtensionType" />

  <complexType name="ServiceReferenceExtensionType">
    <sequence>
      <element name="WSDLLocation" type="anyURI" />
      <element maxOccurs="unbounded" name="Operation" type="string" />
      <!-- Can be extended -->
      <any minOccurs="0" namespace="#any" processContents="strict" />
    </sequence>
  </complexType>
</schema>

<?xml version="1.0" encoding="UTF-8"?>
<schema elementFormDefault="qualified"
    targetNamespace="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement">

  <!-- LISTIC QualifyingCondition extension -->
  <element name="QualifyingConditionExtension" type="ocw:
```

```
  QualifyingConditionExtensionType" />

<complexType name="QualifyingConditionExtensionType">
  <sequence>
    <element name="Periods" type="ocw:PeriodsType" />
    <element name="Workloads" type="ocw:WorkloadsType" />
    <element name="ScalabilityPoints" type="ocw:ScalabilityPointsType" />
    <element minOccurs="0" name="AffinityConstraints" type="ocw:
  AffinityConstraintsType" />
  </sequence>
</complexType>

<!-- Periods -->
<complexType name="PeriodsType">
  <sequence>
    <element name="Period" type="ocw:PeriodType" maxOccurs="unbounded" />
  </sequence>
</complexType>

<complexType name="PeriodType">
  <sequence>
    <element name="Start" type="dateTime" />
    <element name="End" type="dateTime" />
    <element name="Interspace" type="ocw:InterspaceType" minOccurs="0" />
  </sequence>
</complexType>

<complexType name="InterspaceType">
  <sequence>
    <element name="TimeInterval" type="duration" />
    <!-- The duration between 2 times interval-->
    <element name="Step" type="duration" />
  </sequence>
</complexType>

<!-- Workloads -->
<complexType name="WorkloadsType">
  <sequence>
    <element name="Workload" type="ocw:WorkloadType" maxOccurs="unbounded
  " />
  </sequence>
</complexType>

<complexType name="WorkloadType">
  <sequence>
    <element name="InputDataName" type="IDREF" />
    <element name="MetricName" type="IDREF" />
    <element name="LowerBound" type="positiveInteger"/>
    <element name="UpperBound" type="positiveInteger"/>
    <element name="Average" type="positiveInteger" />
    <element minOccurs="0" name="Variation" type="ocw:VariationType" />
  </sequence>
</complexType>

<complexType name="VariationType">
  <sequence>
```

```
    <element name="MetricName" type="IDREF" />
    <element name="Threshold" type="positiveInteger"/>
  </sequence>
</complexType>

<!-- ScalabilityPoints -->
<complexType name="ScalabilityPointsType">
  <choice maxOccurs="unbounded">
    <element name="ComputeScalabilityPoint" type="ocw:
  ComputeScalabilityPointType" />
    <element name="StorageScalabilityPoint" type="ocw:
  StorageScalabilityPointType" />
  </choice>
</complexType>

<complexType name="ComputeScalabilityPointType">
  <sequence>
    <element name="ResourceName" type="IDREF" />
    <element minOccurs="0" name="HorizontalScaling" type="ocw:
  HorizontalScalingType" />
    <element minOccurs="0" name="ComputeVerticalScaling" type="ocw:
  ComputeVerticalScalingType" />
  </sequence>
</complexType>

<complexType name="StorageScalabilityPointType">
  <sequence>
    <element name="ResourceName" type="IDREF" />
    <element minOccurs="0" name="HorizontalScaling" type="ocw:
  HorizontalScalingType" />
    <element minOccurs="0" name="StorageVerticalScaling" type="ocw:
  StorageVerticalScalingType" />
  </sequence>
</complexType>

<complexType name="HorizontalScalingType">
  <sequence>
    <element name="LowerBound" type="positiveInteger"/>
    <element name="UpperBound" type="ocw:UpperBoundType"/>
  </sequence>
</complexType>

<simpleType name="UpperBoundType">
  <union memberTypes="positiveInteger">
    <simpleType>
      <restriction base="string">
        <enumeration value="unbounded"/>
      </restriction>
    </simpleType>
  </union>
</simpleType>

<complexType name="ComputeVerticalScalingType">
  <sequence>
    <element maxOccurs="unbounded" name="Set" type="ocw:ComputeSetType"
  />
```

```
  </sequence>
</complexType>

<complexType name="ComputeSetType">
  <sequence>
    <element name="CPU" type="ocw:LowerBoundType" />
    <element name="RAM" type="ocw:LowerBoundType" />
    <element maxOccurs="unbounded" minOccurs="0" name="NetworkThroughput"
   type="ocw:NetworkThroughputType" />
  </sequence>
</complexType>

<complexType name="LowerBoundType">
  <sequence>
    <element name="MetricName" type="IDREF" />
    <element name="LowerBound" type="positiveInteger"/>
  </sequence>
</complexType>

<complexType name="NetworkThroughputType">
  <sequence>
    <element name="MetricName" type="IDREF" />
    <element name="UploadSpeed" type="positiveInteger"/>
    <element name="DownloadSpeed" type="positiveInteger"/>
    <element name="ResourceName" type="IDREF" />
  </sequence>
</complexType>

<complexType name="StorageVerticalScalingType">
  <sequence>
    <element maxOccurs="unbounded" name="Set" type="ocw:StorageSetType"
 />
  </sequence>
</complexType>

<complexType name="StorageSetType">
  <sequence>
    <element name="Size" type="ocw:LowerBoundType" />
    <element name="DiskThroughput" type="ocw:DiskThroughputType" />
  </sequence>
</complexType>

<complexType name="DiskThroughputType">
  <sequence>
    <element name="MetricName" type="IDREF" />
    <element name="ReadSpeed" type="positiveInteger"/>
    <element name="WriteSpeed" type="positiveInteger"/>
  </sequence>
</complexType>

<!-- AffinityConstraints -->
<complexType name="AffinityConstraintsType">
  <sequence>
    <element name="AffinityConstraint" type="ocw:AffinityConstraintType"
 maxOccurs="unbounded" />
  </sequence>
```

```
  </complexType>

  <complexType name="AffinityConstraintType">
    <sequence>
      <element maxOccurs="unbounded" name="ResourceName" type="IDREF" />
      <element maxOccurs="unbounded" name="Constraint" type="ocw:
    ConstraintType" />
    </sequence>
  </complexType>

  <simpleType name="ConstraintType">
    <restriction base="string">
      <enumeration value="Host Affinity"/>
      <enumeration value="Host Anti-Affinity"/>
      <enumeration value="Rack Affinity"/>
      <enumeration value="Rack Anti-Affinity"/>
      <enumeration value="Data Center Affinity"/>
      <enumeration value="Data Center Anti-Affinity"/>
      <enumeration value="Lonely"/>
    </restriction>
  </simpleType>
</schema>

<?xml version="1.0" encoding="UTF-8"?>
<schema elementFormDefault="qualified"
    targetNamespace="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement">

  <!-- LISTIC shared types -->
  <include schemaLocation="./listic_ocw_SharedTypes.xsd" />

  <!-- LISTIC CustomServiceLevel extension -->
  <element name="CustomServiceLevelExtension" type="ocw:
    CustomServiceLevelExtensionType" />

  <complexType name="CustomServiceLevelExtensionType">
    <group ref="ocw:ExpressionGroup"></group>
  </complexType>

  <!-- ExpressionGroup -->
  <group name="ExpressionGroup">
    <choice>
      <group ref="ocw:LogicExpressionGroup"/>
      <group ref="ocw:ComparisonExpressionGroup"/>
    </choice>
  </group>

  <group name="LogicExpressionGroup">
    <choice>
      <element name="And" type="ocw:BinaryLogicOperatorType" />
      <element name="Or" type="ocw:BinaryLogicOperatorType" />
      <element name="Not" type="ocw:UnaryLogicOperatorType" />
    </choice>
```

```
</group>

<group name="ComparisonExpressionGroup">
  <choice>
    <element name="Greater" type="ocw:PredicateType" />
    <element name="Less" type="ocw:PredicateType" />
    <element name="Equal" type="ocw:PredicateType" />
    <element name="GreaterOrEqual" type="ocw:PredicateType" />
    <element name="LessOrEqual" type="ocw:PredicateType" />
  </choice>
</group>

<!-- Operators -->
<complexType name="UnaryLogicOperatorType">
  <group ref="ocw:ExpressionGroup"/>
</complexType>

<complexType name="BinaryLogicOperatorType">
  <group maxOccurs="2" minOccurs="2" ref="ocw:ExpressionGroup"/>
</complexType>

<!-- Predicates -->
<complexType name="PredicateType">
  <sequence>
    <element name="MetricName" type="IDREF" />
    <element name="Threshold" type="ocw:ThresholdType" />
  </sequence>
</complexType>

<complexType name="ThresholdType">
  <sequence>
    <element name="Value" type="float" />
    <element name="Confidence" type="ocw:ConfidenceType" minOccurs="0" />
  </sequence>
</complexType>

<complexType name="ConfidenceType">
  <sequence>
    <element name="Trust" type="ocw:Trust"/>
    <element name="Fuzziness" type="ocw:Fuzziness"/>
  </sequence>
</complexType>

<simpleType name="Trust">
  <restriction base="float">
    <maxInclusive value="1.00" />
    <minInclusive value="0.00" />
  </restriction>
</simpleType>
<simpleType name="Fuzziness">
  <restriction base="float">
    <maxInclusive value="1.00" />
    <minInclusive value="0.00" />
  </restriction>
</simpleType>
```

```
</schema>

<?xml version="1.0" encoding="UTF-8"?>
<schema elementFormDefault="qualified"
     targetNamespace="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement"
     xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    metamodels/cloud-agreement">

  <!-- LISTIC shared types -->
  <complexType name="NamedElement">
    <attribute name="Name" type="ID" use="required" />
  </complexType>
</schema>
```

Listing A.1. XML Schema of PSLA contract.

Appendix B

XSL based PSLA contract display

XSL is the logogram for eXtensible Stylesheet Language. XSL can be used to display XML based information in a human readable way. Therefore, a XSL based tool is developed to display the PSLA contract in a friendly way. The XSL code is listed in this appendix.

```
<!-- @author frederic.pourraz@univ-savoie.fr -->
<xsl:stylesheet xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware
    /v2/metamodels/cloud-agreement" xmlns:wsag="http://schemas.ggf.org/graap
    /2007/03/ws-agreement" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="2.0">

<xsl:import href="./listic_ocw_ContextExtension.xsl"/>
<xsl:import href="./listic_ocw_ServiceDescriptionTermExtension.xsl"/>
<xsl:import href="./listic_ocw_ServiceReferenceExtension.xsl"/>
<xsl:import href="./listic_ocw_QualifyingConditionExtension.xsl"/>
<xsl:import href="./listic_ocw_CustomServiceLevelExtension.xsl"/>

<xsl:template match="wsag:AgreementOffer">
<html lang="fr">
<head>
<meta charset="utf-8"/>
<meta content="frederic.pourraz@univ-savoie.fr" name="author"/>
<meta content="initial-scale=1.0, maximum-scale=1.0, user-scalable=no,
    width=device-width" name="viewport"/>
<link href="http://www.polytech.univ-savoie.fr/opencloudware/media/image/
    favicon.ico" rel="shortcut icon" type="image/x-icon"/>
<link href="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    transformations/media/style/all.css" media="all" rel="stylesheet" type="
    text/css"/>
<link href="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    transformations/media/style/768.css" media="(max-width:768px)" rel="
    stylesheet" type="text/css"/>
<link href="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    transformations/media/style/640.css" media="(max-width:640px)" rel="
    stylesheet" type="text/css"/>
<title>
<xsl:value-of select="wsag:Name"/>
</title>
<script src="http://www.polytech.univ-savoie.fr/opencloudware/v2/
    transformations/media/script/colorify.js" type="text/javascript"/>
<script src="http://www.polytech.univ-savoie.fr/opencloudware/v2/
```

```
    transformations/media/script/variables.js" type="text/javascript"/>
</head>
<body>
<xsl:apply-templates select="wsag:Name"/>
<xsl:apply-templates select="wsag:Context"/>
<xsl:apply-templates select="wsag:Terms"/>
</body>
</html>
</xsl:template>

<xsl:template match="wsag:Name">
<header>
<h1>
<xsl:value-of select="."/>
</h1>
</header>
</xsl:template>

<!-- Context -->
<xsl:template match="wsag:Context">
<section id="Context" onclick="colorify(event, this);">
<header>
<h1>Context</h1>
</header>
<ul>
<li>
<span class="name">Agreement Initiator</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="wsag:AgreementInitiator"/>
</span>
</li>
</ul>
<ul>
<li>
<span class="name">Agreement Responder</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="wsag:AgreementResponder"/>
</span>
</li>
</ul>
<ul>
<li>
<span class="name">Service Provider</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="wsag:ServiceProvider"/>
</span>
</li>
</ul>
<ul>
<li>
<span class="name">Expiration Time</span>
<span class="separator">:</span>
<span class="value">
```

```
<xsl:value-of select="wsag:ExpirationTime"/>
</span>
</li>
</ul>
<xsl:apply-templates select="ocw:ContextExtension"/>
</section>
</xsl:template>

<!-- Terms -->
<xsl:template match="wsag:Terms">
<section id="Terms" onclick="colorify(event, this);">
<header>
<h1>Terms</h1>
</header>
<xsl:apply-templates select="wsag:All"/>
</section>
</xsl:template>

<!-- TermCompositor -->
<xsl:template name="TermCompositor">
<xsl:apply-templates select="wsag:All"/>
<xsl:apply-templates select="wsag:ExactlyOne"/>
<xsl:apply-templates select="wsag:OneOrMore"/>
<xsl:apply-templates select="wsag:ServiceProperties"/>
<xsl:apply-templates select="wsag:GuaranteeTerm"/>
<div class="space"/>
<xsl:apply-templates select="wsag:ServiceDescriptionTerm"/>
<div class="space"/>
<xsl:apply-templates select="wsag:ServiceReference"/>
<div class="space"/>
</xsl:template>

<!-- Operators -->
<xsl:template match="wsag:All">
<section class="TermCompositorOperator" onclick="colorify(event, this);">
<header>
<h1>All</h1>
</header>
<xsl:call-template name="TermCompositor"/>
</section>
</xsl:template>
<xsl:template match="wsag:ExactlyOne">
<section class="TermCompositorOperator" onclick="colorify(event, this);">
<header>
<h1>Exactly One</h1>
</header>
<xsl:call-template name="TermCompositor"/>
</section>
</xsl:template>
<xsl:template match="wsag:OneOrMore">
<section class="TermCompositorOperator" onclick="colorify(event, this);">
<header>
<h1>One or More</h1>
</header>
<xsl:call-template name="TermCompositor"/>
</section>
```

```
</xsl:template>

<!-- ServiceDescriptionTerm -->
<xsl:template match="wsag:ServiceDescriptionTerm">
<section onclick="colorify(event, this);">
<h1>ServiceDescriptionTerm</h1>
<ul>
<li>
<span class="name">Name</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@Name"/>
</span>
</li>
</ul>
<ul class="serviceName">
<li>
<span class="name">Service Name</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@ServiceName"/>
</span>
</li>
</ul>
<xsl:apply-templates select="ocw:ServiceDescriptionTermExtension"/>
</section>
</xsl:template>

<!-- ServiceReference -->
<xsl:template match="wsag:ServiceReference">
<section id="{@Name}" onclick="colorify(event, this);">
<h1>ServiceReference</h1>
<ul>
<li>
<span class="name">Name</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@Name"/>
</span>
</li>
</ul>
<ul class="serviceName">
<li>
<span class="name">Service Name</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@ServiceName"/>
</span>
</li>
</ul>
<xsl:apply-templates select="ocw:ServiceReferenceExtension"/>
</section>
</xsl:template>

<!-- ServiceProperties -->
<xsl:template match="wsag:ServiceProperties">
```

```
<xsl:apply-templates select="wsag:VariableSet"/>
</xsl:template>

<!-- Variable -->
<xsl:template match="wsag:VariableSet">
<xsl:apply-templates select="wsag:Variable"/>
</xsl:template>
<xsl:template match="wsag:Variable">
<xsl:apply-templates select="wsag:Location">
<xsl:with-param name="name" select="@Name"/>
</xsl:apply-templates>
</xsl:template>
<xsl:template match="wsag:Location">
<xsl:param name="name"/>
<xsl:variable name="metric">
<xsl:value-of select="substring-before(substring-after(., 'ocw:MetricName
    ='), '"]')"/>
</xsl:variable>
<script>
addVariable("
<xsl:value-of select="$name"/>
", "
<xsl:value-of select="$metric"/>
");
</script>
</xsl:template>

<!-- GuaranteeTerm -->
<xsl:template match="wsag:GuaranteeTerm">
<section id="{@Name}" onclick="colorify(event, this);">
<h1>GuaranteeTerm</h1>
<ul>
<li>
<span class="name">Name</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@Name"/>
</span>
</li>
</ul>
<ul>
<li>
<span class="name">Obligated</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@Obligated"/>
</span>
</li>
</ul>
<div class="space"/>
<xsl:apply-templates select="wsag:ServiceScope"/>
<xsl:apply-templates select="wsag:QualifyingCondition"/>
<xsl:apply-templates select="wsag:ServiceLevelObjective"/>
<xsl:apply-templates select="wsag:BusinessValueList"/>
</section>
</xsl:template>
```

```
<!-- ServiceScope -->
<xsl:template match="wsag:ServiceScope">
<ul class="serviceName">
<li>
<span class="name">Service Name</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@ServiceName"/>
</span>
</li>
</ul>
</xsl:template>

<!-- QualifyingCondition -->
<xsl:template match="wsag:QualifyingCondition">
<section onclick="colorify(event, this);">
<h1>QualifyingCondition</h1>
<xsl:apply-templates select="ocw:QualifyingConditionExtension"/>
</section>
</xsl:template>

<!-- ServiceLevelObjective -->
<xsl:template match="wsag:ServiceLevelObjective">
<xsl:apply-templates select="wsag:CustomServiceLevel"/>
</xsl:template>
<xsl:template match="wsag:CustomServiceLevel">
<xsl:apply-templates select="ocw:CustomServiceLevelExtension"/>
</xsl:template>

<!-- BusinessValueList -->
<xsl:template match="wsag:BusinessValueList">
<section onclick="colorify(event, this);">
<h1>BusinessValueList</h1>
<xsl:apply-templates select="wsag:Importance"/>
<xsl:apply-templates select="wsag:Penalty"/>
<xsl:apply-templates select="wsag:Reward"/>
<xsl:apply-templates select="wsag:Preference"/>
</section>
</xsl:template>
<xsl:template match="wsag:Importance">
<ul>
<li>
<span class="name">Importance</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="wsag:Penalty">
<ul>
<li>
<span class="value">Penalty</span>
<xsl:apply-templates select="wsag:AssessmentInterval"/>
```

```
<xsl:apply-templates select="wsag:ValueUnit"/>
<xsl:apply-templates select="wsag:ValueExpression"/>
</li>
</ul>
</xsl:template>
<xsl:template match="wsag:Reward">
<ul>
<li>
<span class="value">Reward</span>
<xsl:apply-templates select="wsag:AssessmentInterval"/>
<xsl:apply-templates select="wsag:ValueUnit"/>
<xsl:apply-templates select="wsag:ValueExpression"/>
</li>
</ul>
</xsl:template>
<xsl:template match="wsag:AssessmentInterval">
<xsl:apply-templates select="wsag:TimeInterval"/>
<xsl:apply-templates select="wsag:Count"/>
</xsl:template>
<xsl:template match="wsag:TimeInterval">
<ul>
<li>
<span class="name">Time Interval</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="wsag:Count">
<ul>
<li>
<span class="name">Count</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="wsag:ValueUnit">
<ul>
<li>
<span class="name">Unit</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="wsag:ValueExpression">
<ul>
<li>
<span class="name">Value</span>
```

```
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="wsag:Preference">
<ul>
<li>
<span class="value">Preference</span>
<xsl:apply-templates select="wsag:ServiceTermReference"/>
<xsl:apply-templates select="wsag:Utility"/>
</li>
</ul>
</xsl:template>
<xsl:template match="wsag:ServiceTermReference">
<ul>
<li>
<span class="name">ServiceTermReference</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="wsag:Utility">
<ul>
<li>
<span class="name">Utility</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
</xsl:stylesheet>

<xsl:stylesheet xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware
    /v2/metamodels/cloud-agreement" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform" version="2.0">

<!-- ContextExtension -->
<xsl:template match="ocw:ContextExtension">
<div class="opencloudware">
<xsl:apply-templates select="ocw:OVFLocation"/>
</div>
</xsl:template>

<!-- OVFLocation -->
<xsl:template match="ocw:OVFLocation">
<ul>
<li>
<span class="name">OVF Location</span>
```

```
<span class="separator">:</span>
<span class="value">
<a href="{.}" target="_blank">
<xsl:value-of select="."/>
</a>
</span>
</li>
</ul>
</xsl:template>

</xsl:stylesheet>

<xsl:stylesheet xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware
    /v2/metamodels/cloud-agreement" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform" version="2.0">

<!-- ServiceDescriptionTermExtension -->
<xsl:template match="ocw:ServiceDescriptionTermExtension">
<div class="opencloudware">
<div class="space"/>
<xsl:apply-templates select="ocw:Metric"/>
<div class="space"/>
<xsl:apply-templates select="ocw:InputData"/>
<div class="space"/>
<xsl:apply-templates select="ocw:Resource"/>
</div>
</xsl:template>

<!-- Metric -->
<xsl:template match="ocw:Metric">
<section id="{@Name}" onclick="colorify(event, this);">
<h1>Metric</h1>
<ul>
<li>
<span class="name">Name</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@Name"/>
</span>
</li>
</ul>
<xsl:apply-templates select="ocw:Unit"/>
<xsl:apply-templates select="ocw:Type"/>
<xsl:apply-templates select="ocw:ValueGenerateBy"/>
</section>
</xsl:template>

<!-- Unit -->
<xsl:template match="ocw:Unit">
<xsl:apply-templates select="ocw:BuiltInUnit"/>
<xsl:apply-templates select="ocw:UserDefinedUnit"/>
</xsl:template>
<xsl:template match="ocw:BuiltInUnit">
<ul>
<li>
<span class="name">Built In Unit</span>
```

```
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:UserDefinedUnit">
<ul>
<li>
<span class="name">User Defined Unit</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@Name"/>
</span>
</li>
</ul>
<xsl:apply-templates select="ocw:Description"/>
</xsl:template>
<xsl:template match="ocw:Description">
<ul>
<li>
<span class="name">Description</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>

<!-- Type -->
<xsl:template match="ocw:Type">
<ul>
<li>
<span class="name">Type</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>

<!-- ValueGenerateBy -->
<xsl:template match="ocw:ValueGenerateBy">
<xsl:apply-templates select="ocw:ServiceReferenceName"/>
<xsl:apply-templates select="ocw:Schedule"/>
</xsl:template>
<xsl:template match="ocw:ServiceReferenceName">
<ul>
<li>
<span class="name">Value Generate by the Service Reference</span>
<span class="separator">:</span>
<span class="value">
<a href="#{.}" onclick="colorify(event, document.getElementById('{.}'));">
```

```
<xsl:value-of select="."/>
</a>
<xsl:apply-templates select="../ocw:ServiceReferenceOperation"/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:ServiceReferenceOperation">
(
<xsl:value-of select="."/>
)
</xsl:template>
<xsl:template match="ocw:Schedule">
<ul>
<li>
<span class="name">Schedule</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>

<!-- InputData -->
<xsl:template match="ocw:InputData">
<section id="{@Name}" onclick="colorify(event, this);">
<h1>Input Data</h1>
<ul>
<li>
<span class="name">Name</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@Name"/>
</span>
</li>
</ul>
<xsl:apply-templates select="ocw:Scenario"/>
<xsl:apply-templates select="ocw:Description"/>
</section>
</xsl:template>

<!-- Scenario -->
<xsl:template match="ocw:Scenario">
<ul>
<li>
<span class="name">Scenario</span>
<span class="separator">:</span>
<span class="value">
<a href="{.}" target="_blank">
<xsl:value-of select="."/>
</a>
</span>
</li>
</ul>
</xsl:template>
```

```
<!-- Resource -->
<xsl:template match="ocw:Resource">
<section id="{@Name}" onclick="colorify(event, this);">
<h1>Resource</h1>
<ul>
<li>
<span class="name">Name</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="@Name"/>
</span>
</li>
</ul>
<xsl:apply-templates select="ocw:XPath"/>
</section>
</xsl:template>

<!-- Scenario -->
<xsl:template match="ocw:XPath">
<ul>
<li>
<span class="name">
XPath inside the OVF document (see
<a href="#Context">OVF Location</a>
)
</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
</xsl:stylesheet>

<xsl:stylesheet xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware
    /v2/metamodels/cloud-agreement" xmlns:wsag="http://schemas.ggf.org/graap
    /2007/03/ws-agreement" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="2.0">

<!-- ServiceReferenceExtension -->
<xsl:template match="ocw:ServiceReferenceExtension">
<div class="opencloudware">
<xsl:apply-templates select="ocw:WSDLLocation"/>
<xsl:apply-templates select="ocw:Operation"/>
</div>
</xsl:template>

<!-- WSDLLocation -->
<xsl:template match="ocw:WSDLLocation">
<ul>
<li>
<span class="name">WSDL Location</span>
<span class="separator">:</span>
<span class="value">
```

```
<a href="{.}" target="_blank">
<xsl:value-of select="."/>
</a>
</span>
</li>
</ul>
</xsl:template>

<!-- Operation -->
<xsl:template match="ocw:Operation">
<ul>
<li>
<span class="name">Operation</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
</xsl:stylesheet>

<xsl:stylesheet xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware
    /v2/metamodels/cloud-agreement" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform" version="2.0">

<xsl:import href="./listic_ocw_SharedTypes.xsl"/>

<!-- QualifyingConditionExtension -->
<xsl:template match="ocw:QualifyingConditionExtension">
<div class="opencloudware">
<div class="space"/>
<xsl:apply-templates select="ocw:Periods"/>
<div class="space"/>
<xsl:apply-templates select="ocw:Workloads"/>
<div class="space"/>
<xsl:apply-templates select="ocw:ScalabilityPoints"/>
<div class="space"/>
<xsl:apply-templates select="ocw:AffinityConstraints"/>
</div>
</xsl:template>

<!-- Period -->
<xsl:template match="ocw:Periods">
<xsl:apply-templates select="ocw:Period"/>
</xsl:template>
<xsl:template match="ocw:Period">
<section onclick="colorify(event, this);">
<h1>Period</h1>
<xsl:apply-templates select="ocw:Start"/>
<xsl:apply-templates select="ocw:End"/>
<xsl:apply-templates select="ocw:Interspace"/>
</section>
</xsl:template>

<!-- Start -->
```

```
<xsl:template match="ocw:Start">
<ul>
<li>
<span class="name">Start</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>

<!-- End -->
<xsl:template match="ocw:End">
<ul>
<li>
<span class="name">End</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>

<!-- Interspace -->
<xsl:template match="ocw:Interspace">
<ul>
<li>
<span class="value">Interspace</span>
<xsl:apply-templates select="ocw:TimeInterval"/>
<xsl:apply-templates select="ocw:Step"/>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:TimeInterval">
<ul>
<li>
<span class="name">Time Interval</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:Step">
<ul>
<li>
<span class="name">Step</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
```

```
</xsl:template>

<!--  Workload  -->
<xsl:template match="ocw:Workloads">
<xsl:apply-templates select="ocw:Workload"/>
</xsl:template>
<xsl:template match="ocw:Workload">
<section onclick="colorify(event, this);">
<h1>Workload</h1>
<xsl:apply-templates select="ocw:InputDataName"/>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:LowerBound"/>
<xsl:apply-templates select="ocw:UpperBound"/>
<xsl:apply-templates select="ocw:Average"/>
<xsl:apply-templates select="ocw:Variation"/>
</section>
</xsl:template>
<xsl:template match="ocw:InputDataName">
<ul>
<li>
<span class="name">Input Data</span>
<span class="separator">:</span>
<span class="value">
<a href="#{.}" onclick="colorify(event, document.getElementById('{.}'));">
<xsl:value-of select="."/>
</a>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:LowerBound">
<ul>
<li>
<span class="name">Lower Bound</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:UpperBound">
<ul>
<li>
<span class="name">Upper Bound</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:Average">
<ul>
<li>
<span class="name">Average</span>
```

```
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:Variation">
<ul>
<li>
<span class="value">Throughput Variation</span>
<xsl:apply-templates select="ocw:MetricName"/>
<ul>
<li>
<span class="name">Threshold</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="ocw:Threshold"/>
</span>
</li>
</ul>
</li>
</ul>
</xsl:template>

<!-- ScalabilityPoints -->
<xsl:template match="ocw:ScalabilityPoints">
<xsl:apply-templates select="ocw:ComputeScalabilityPoint"/>
<div class="space"/>
<xsl:apply-templates select="ocw:StorageScalabilityPoint"/>
</xsl:template>

<!-- ComputeScalabilityPoint -->
<xsl:template match="ocw:ComputeScalabilityPoint">
<section onclick="colorify(event, this);">
<h1>Compute Scalability Point</h1>
<xsl:apply-templates select="ocw:ResourceName"/>
<div class="space"/>
<xsl:apply-templates select="ocw:HorizontalScaling"/>
<xsl:apply-templates select="ocw:ComputeVerticalScaling"/>
</section>
</xsl:template>

<!-- StorageScalabilityPoint -->
<xsl:template match="ocw:StorageScalabilityPoint">
<section onclick="colorify(event, this);">
<h1>Storage Scalability Point</h1>
<xsl:apply-templates select="ocw:ResourceName"/>
<div class="space"/>
<xsl:apply-templates select="ocw:HorizontalScaling"/>
<xsl:apply-templates select="ocw:StorageVerticalScaling"/>
</section>
</xsl:template>

<!-- ResourceName -->
<xsl:template match="ocw:ResourceName">
```

```
<ul>
<li>
<span class="name">Resource</span>
<span class="separator">:</span>
<span class="value">
<a href="#{.}" onclick="colorify(event, document.getElementById('{.}'));">
<xsl:value-of select="."/>
</a>
</span>
</li>
</ul>
</xsl:template>

<!-- HorizontalScaling -->
<xsl:template match="ocw:HorizontalScaling">
<section onclick="colorify(event, this);">
<h1>Horizontal Scaling</h1>
<xsl:apply-templates select="ocw:LowerBound"/>
<xsl:apply-templates select="ocw:UpperBound"/>
</section>
</xsl:template>

<!-- VerticalScaling -->
<xsl:template match="ocw:ComputeVerticalScaling">
<section onclick="colorify(event, this);">
<h1>Vertical Scaling</h1>
<table>
<tr>
<xsl:for-each select="ocw:Set">
<td>
<xsl:apply-templates select="ocw:CPU"/>
<xsl:apply-templates select="ocw:RAM"/>
<xsl:apply-templates select="ocw:NetworkThroughput"/>
</td>
</xsl:for-each>
</tr>
</table>
</section>
</xsl:template>
<xsl:template match="ocw:CPU">
<ul>
<li>
<span class="value">CPU</span>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:LowerBound"/>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:RAM">
<ul>
<li>
<span class="value">RAM</span>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:LowerBound"/>
</li>
</ul>
```

```
</xsl:template>
<xsl:template match="ocw:NetworkThroughput">
<ul>
<li>
<span class="value">Network Throughput</span>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:UploadSpeed"/>
<xsl:apply-templates select="ocw:DownloadSpeed"/>
<xsl:apply-templates select="ocw:ResourceName"/>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:UploadSpeed">
<ul>
<li>
<span class="name">Upload Speed</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:DownloadSpeed">
<ul>
<li>
<span class="name">Download Speed</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:StorageVerticalScaling">
<section onclick="colorify(event, this);">
<h1>Vertical Scaling</h1>
<table>
<tr>
<xsl:for-each select="ocw:Set">
<td>
<xsl:apply-templates select="ocw:Size"/>
<xsl:apply-templates select="ocw:DiskThroughput"/>
</td>
</xsl:for-each>
</tr>
</table>
</section>
</xsl:template>
<xsl:template match="ocw:Size">
<ul>
<li>
<span class="value">Size</span>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:LowerBound"/>
</li>
```

```
</ul>
</xsl:template>
<xsl:template match="ocw:DiskThroughput">
<ul>
<li>
<span class="value">Disk Throughput</span>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:ReadSpeed"/>
<xsl:apply-templates select="ocw:WriteSpeed"/>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:ReadSpeed">
<ul>
<li>
<span class="name">Read Speed</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:WriteSpeed">
<ul>
<li>
<span class="name">Write Speed</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>

<!-- AffinityConstraints -->
<xsl:template match="ocw:AffinityConstraints">
<xsl:apply-templates select="ocw:AffinityConstraint"/>
</xsl:template>
<xsl:template match="ocw:AffinityConstraint">
<section onclick="colorify(event, this);">
<h1>Affinity Constraint</h1>
<xsl:apply-templates select="ocw:ResourceName"/>
<xsl:apply-templates select="ocw:Constraint"/>
</section>
</xsl:template>

<!-- Constraint -->
<xsl:template match="ocw:Constraint">
<ul>
<li>
<span class="name">Constraint</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
```

```
</li>
</ul>
</xsl:template>
</xsl:stylesheet>

<xsl:stylesheet xmlns:ocw="http://www.polytech.univ-savoie.fr/opencloudware
    /v2/metamodels/cloud-agreement" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform" version="2.0">

<xsl:import href="./listic_ocw_SharedTypes.xsl"/>

<!-- CustomServiceLevelExtension -->
<xsl:template match="ocw:CustomServiceLevelExtension">
<div class="opencloudware">
<div class="space"/>
<section id="{@Name}" onclick="colorify(event, this);">
<h1>SLO</h1>
<xsl:call-template name="LogicExpression"/>
</section>
<div class="space"/>
</div>
</xsl:template>

<!-- LogicExpression -->
<xsl:template name="LogicExpression">
<xsl:apply-templates select="ocw:And"/>
<xsl:apply-templates select="ocw:Or"/>
<xsl:apply-templates select="ocw:Not"/>
<xsl:apply-templates select="ocw:Greater"/>
<xsl:apply-templates select="ocw:Less"/>
<xsl:apply-templates select="ocw:Equal"/>
<xsl:apply-templates select="ocw:GreaterOrEqual"/>
<xsl:apply-templates select="ocw:LessOrEqual"/>
</xsl:template>

<!-- Operators -->
<xsl:template match="ocw:And">
<section class="LogicExpressionOperator" onclick="colorify(event, this);">
<header>
<h1>And</h1>
</header>
<xsl:call-template name="LogicExpression"/>
</section>
</xsl:template>
<xsl:template match="ocw:Or">
<section class="LogicExpressionOperator" onclick="colorify(event, this);">
<header>
<h1>Or</h1>
</header>
<xsl:call-template name="LogicExpression"/>
</section>
</xsl:template>
<xsl:template match="ocw:Not">
<section class="LogicExpressionOperator" onclick="colorify(event, this);">
<header>
<h1>Not</h1>
```

```
</header>
<xsl:call-template name="LogicExpression"/>
</section>
</xsl:template>

<!-- Predicates -->
<xsl:template match="ocw:Greater">
<ul>
<li>
<span class="value">Greater</span>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:Threshold"/>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:Less">
<ul>
<li>
<span class="value">Less</span>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:Threshold"/>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:Equal">
<ul>
<li>
<span class="value">Equal</span>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:Threshold"/>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:GreaterOrEqual">
<ul>
<li>
<span class="value">Greater or Equal</span>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:Threshold"/>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:LessOrEqual">
<ul>
<li>
<span class="value">Less or Equal</span>
<xsl:apply-templates select="ocw:MetricName"/>
<xsl:apply-templates select="ocw:Threshold"/>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:Threshold">
<xsl:apply-templates select="ocw:Value"/>
<xsl:apply-templates select="ocw:Confidence"/>
</xsl:template>
<xsl:template match="ocw:Value">
```

```
<ul>
<li>
<span class="name">Threshold</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:Confidence">
<xsl:apply-templates select="ocw:Trust"/>
<xsl:apply-templates select="ocw:Fuzziness"/>
</xsl:template>
<xsl:template match="ocw:Trust">
<ul>
<li>
<span class="name">Trust</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select=". * 100"/>
%
</span>
</li>
</ul>
</xsl:template>
<xsl:template match="ocw:Fuzziness">
<ul>
<li>
<span class="name">Fuzziness</span>
<span class="separator">:</span>
<span class="value">
<xsl:value-of select="."/>
</span>
</li>
</ul>
</xsl:template>
</xsl:stylesheet>
```

Listing B.1. XSL code for displaying PSLA contract.

Bibliography

[AAA+10] Miha Ahronovitz, D Amrhein, P Anderson, A de Andrade, J Armstrong, BE Arasan, J Bartlett, R Bruklis, K Cameron, M Carlson, et al. Cloud computing use cases white paper. online, 2010.

[ACD+07] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web services agreement specification (ws-agreement). In *Open Grid Forum*, volume 128, 2007.

[ADK+04] Alain Andrieux, Asit Dan, K Keahey, Heiko Ludwig, and John Rofrano. Negotiability constraints in ws-agreement. In *Grid Resource Allocation Agreement Protocol (GRAAP) Working Group Meetings, Tech. Rep*, 2004.

[AETE12] Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 204–212. IEEE, 2012.

[ALIZ10] Bo An, Victor Lesser, David Irwin, and Michael Zink. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 981–988. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[Åst12] Karl J Åström. *Introduction to stochastic control theory*. Courier Corporation, 2012.

[Bar98] Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[BB10] Anton Beloglazov and Rajkumar Buyya. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, page 4. ACM, 2010.

[BBB+08] S Becker, L Bulej, T Bures, P Hnetynka, L Kapova, J Kofron, H Koziolek, J Kraft, R Mirandola, J Stammel, et al. Q-impress project deliverable d2. 1: service architecture meta model (samm). *Project Deliverable*, 2008.

[BBB+11] Dominique Bellenger, Jens Bertram, Andy Budina, Arne Koschel, Benjamin Pfänder, Carsten Serowy, Irina Astrova, Stella Gatziu Grivas, and Marc Schaaf. Scaling in cloud environments. *Recent Researches in Computer Science*, 2011.

[BCH+03] Don Box, Francisco Curbera, Maryann Hondo, Chris Kaler, Dave Langworthy, Anthony Nadalin, Nataraj Nagaratnam, Mark Nottingham, Claus von Riegen, and John Shewchuk. Web services policy framework (ws-policy), 2003.

[BCS04] Eric Bruneton, Thierry Coupaye, and Jean-Bernard Stefani. The fractal component model. *Draft of specification, version*, pages 2–0, 2004.

[Ben12] El Hachemi Bendahmane. *Introduction de fonctionnalités d'auto-optimisation dans une architecture de selfbenchmarking*. PhD thesis, Université de Grenoble, 2012.

[BFF90] Benjamin S Blanchard, Wolter J Fabrycky, and Wolter J Fabrycky. *Systems engineering and analysis*, volume 4. Prentice Hall Englewood Cliffs, New Jersey, 1990.

[BGS+09] Peter Bodık, Rean Griffith, Charles Sutton, Armando Fox, Michael Jordan, and David Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, pages 12–12, 2009.

[BHD13] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[BM07] Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In *Programming Languages and Systems*, pages 18–32. Springer, 2007.

[Box76] George E Box. P, and jenkins, gm, "time series analysis: Forecasting and control". *Time Series and Digital Processing*, 1976.

[BSL+13] Gunnar Brataas, Erlend Stav, Sebastian Lehrig, Steffen Becker, Goran Kopčak, and Darko Huljenic. Cloudscale: scalability management for cloud systems. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, pages 335–338. ACM, 2013.

[BTW01] Harold Boley, Said Tabet, and Gerd Wagner. Design rationale for ruleml: A markup language for semantic web rules. In *SWWS*, volume 1, pages 381–401, 2001.

[BYV08] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 5–13. Ieee, 2008.

[C+] 4CaaSt Consortium et al. 4caast project website.

[C+06] Autonomic Computing et al. An architectural blueprint for autonomic computing. *IBM White Paper*, 2006.

[C+10] Q-ImPrESS Consortium et al. The q-impress project. *Project website: http://www. q-impress. eu*, 2010.

[Cas06] Kevin Castor. Hardware testing and benchmarking methodology. online, 2006. retrieve 01.2014.

[CCD+12] Emanuele Carlini, Massimo Coppola, Patrizio Dazzi, Laura Ricci, and Giacomo Righetti. Cloud federations in contrail. In *Euro-Par 2011: Parallel Processing Workshops*, pages 159–168. Springer, 2012.

[CGS03] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *Quality of Service-IWQoS 2003*, pages 381–398. Springer, 2003.

[CHL+08] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI*, volume 8, pages 337–350, 2008.

[CMK11] Trieu C Chieu, Ajay Mohindra, and Alexei A Karve. Scalability and performance of web applications in a compute cloud. In *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on*, pages 317–323. IEEE, 2011.

[CMKS09] Trieu C Chieu, Ajay Mohindra, Alexei A Karve, and Alla Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *e-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*, pages 281–286. IEEE, 2009.

[CV03] María Agustina Cibrán and Bart Verheecke. Modularizing web services management with aop. *Orientation and Web Services*, page 14, 2003.

[DeM02] Linda G DeMichiel. Enterprise javabeans specification, version 2.1. 2002.

[Dil09] Bruno Dillenseger. Clif, a framework based on fractal for flexible, distributed load testing. *annals of telecommunications-annales des télécommunications*, 64(1-2):101–120, 2009.

[DKLR04] Asit Dan, K Keahey, H Ludwig, and J Rofrano. Guarantee terms in ws-agreement. In *Grid Resource Allocation Agreement Protocol (GRAAP) Working Group Meetings, Tech. Rep*, 2004.

[DKM+11] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, pages 67–74, 2011.

[DLHS11] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011.

[DRM+10] Xavier Dutreilh, Nicolas Rivierre, Aurélien Moreau, Jacques Malenfant, and Isis Truck. From data center resource allocation to control theory and back. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 410–417. IEEE, 2010.

[DTB10] Amir Vahid Dastjerdi, Sayed Gholam Hassan Tabatabaei, and Rajkumar Buyya. An effective architecture for automated appliance management system applying ontology-based cloud discovery. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 104–112. IEEE, 2010.

[ec213] Amazon EC2 SLA. online, 06/2013. retrieve 02/2015.

[FHT+12] Ana Juan Ferrer, Francisco HernáNdez, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, RaüL Sirvent, Jordi Guitart, Rosa M Badia, Karim Djemame, et al. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66–77, 2012.

[FK03] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[FSJ98] Peyman Faratin, Carles Sierra, and Nick R Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3):159–182, 1998.

[G+09] Jeremy Geelan et al. Twenty-one experts define cloud computing. *Cloud Computing Journal*, 4:1–5, 2009.

[GGJGT+12] Sergio Garcia-Gomez, Miguel Jimenez-Ganan, Yehia Taher, Christof Momm, Frederic Junker, Jozsef Biro, Andreas Menychtas, Vasilios Andrikopoulos, and Steve Strauch. Challenges for the comprehensive management of cloud services in a paas framework. *Scalable Computing: Practice and Experience*, 13(3), 2012.

[GGW10] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9–16. IEEE, 2010.

[GLDK03] Henner Gimpel, Heiko Ludwig, Asit Dan, and Bob Kearney. Panda: Specifying policies for automated negotiations of service contracts. In *Service-Oriented Computing-ICSOC 2003*, pages 287–302. Springer, 2003.

[GoG10] GoGrid. Gogrid whitepaper " scaling your internet business". online, 2010.

[goo15] Googleappengine sla. On line, 5 2015.

[Ham94] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994.

[HHKA14] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. Self-adaptive workload classification and forecasting for proactive resource provisioning. *Concurrency and Computation: Practice and Experience*, 26(12):2053–2078, 2014.

[HK06] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

[HK07] RJ Hyndman and Y Khandakar. Automatic time series forecasting: the forecast package for r 7, 2008. *URL http://www. jstatsoft. org/v27/i03*, 2007.

[HKOS08] Rob Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media, 2008.

[HKPB05] Rob J Hyndman, Maxwell L King, Ivet Pitrun, and Baki Billah. Local linear forecasts using cubic smoothing splines. *Australian & New Zealand Journal of Statistics*, 47(1):87–99, 2005.

[HLY12] Jinhui Huang, Chunlin Li, and Jie Yu. Resource prediction based on double exponential smoothing in cloud computing. In *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pages 2056–2060. IEEE, 2012.

[HMC+12] Masum Z Hasan, Edgar Magana, Alexander Clemm, Lew Tucker, and Sree Lakshmi D Gudreddi. Integrated and autonomic cloud resource scaling. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 1327–1334. IEEE, 2012.

[HR85] Frederick Hayes-Roth. Rule-based systems. *Communications of the ACM*, 28(9):921–932, 1985.

[iaa15] What is infrastructure as a service(iaas) definition. online, 2015.

[IBM] IBM. Ibm service level agreement lifecycle. online. retrieved 01/2015.

[IDCJ11] Waheed Iqbal, Matthew N Dailey, David Carrera, and Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, 27(6):871–879, 2011.

[IKLL12] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012.

[Jac12] Kevin Jackson. *OpenStack cloud computing cookbook*. Packt Publishing Ltd, 2012.

[KBS05] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: service-oriented architecture best practices*. Prentice Hall Professional, 2005.

[KF11] T. Kearney Keven and Torelli Francesco. *The SLA Model*, volume 2, chapter 4, pages 43–68. springer, jun 2011. Service Level Agreements for Cloud Computing.

[KGM10] Sunirmal Khatua, Anirban Ghosh, and Nandini Mukherjee. Optimizing the utilization of virtual resources in cloud environment. In *Virtual Environments Human-Computer Interfaces and Measurement Systems (VECIMS), 2010 IEEE International Conference on*, pages 82–87. IEEE, 2010.

[KL+12] Yousri Kouki, Thomas Ledoux, et al. CSLA: a language for improving cloud SLA management. In *Proceedings of the International Conference on Cloud Computing and Services Science*, Porto, Portugal, April 18-21 2012.

[KSB+11] Heiko Koziolek, Bastian Schlich, Carlos Bilich, Roland Weiss, Steffen Becker, Klaus Krogmann, Mircea Trifu, Raffaela Mirandola, and Anne Koziolek. An industrial case study on quality impact prediction for evolving service-oriented software. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 776–785. ACM, 2011.

[KSJB09] Jonathan Kupferman, Jeff Silverman, Patricio Jara, and Jeff Browne. Scaling into the cloud. *CS270-Advanced Operating Systems*, 2009.

[KTK10] Keven T Kearney, Francesco Torelli, and Constantinos Kotsokalis. Sla*: An abstract syntax for service level agreements. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 217–224. IEEE, 2010.

[LBC10] Harold C Lim, Shivnath Babu, and Jeffrey S Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, pages 1–10. ACM, 2010.

[LBCP09] Harold C Lim, Shivnath Babu, Jeffrey S Chase, and Sujay S Parekh. Automated control in cloud computing: challenges and opportunities. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, pages 13–18. ACM, 2009.

[LBMAL12] Tania Lorido-Botrán, José Miguel-Alonso, and Jose Antonio Lozano. Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, 12:2012, 2012.

[LKD+03] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P King, and Richard Franck. Web service level agreement (wsla) language specification. *IBM Corporation*, pages 815–824, 2003.

[LLD+03] Richard Lawley, Michael Luck, Keith Decker, Terry Payne, and Luc Moreau. Automated negotiation between publishers and consumers of grid notifications. *Parallel Processing Letters*, 13(04):537–548, 2003.

[LPM14] Ge Li, Frédéric Pourraz, and Patrice Moreaux. Psla: a paas level sla description language. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 452–457. IEEE, 2014.

[LPM15] Ge Li, Frédéric Pourraz, and Patrice Moreaux. A benchmarking based sla feasibility study method for platform as a service. *CLOUD COMPUTING 2015*, page 175, 2015.

[LSCD+10] Nicolas Le Sauze, Agostino Chiosi, Richard Douville, Helia Pouyllau, Häkon LONSETHAGEN, Paola Fantini, Claudio Palasciano, Antonio Cimmino, Maria Angeles CALLEJO RODRIGUEZ, Olivier Dugeon, et al. Etics: Qos-enabled interconnection for future internet services. *Future network and mobile summit*, 2010.

[LSE03] D Davide Lamanna, James Skene, and Wolfgang Emmerich. Slang: a language for service level agreements. 2003.

[LTCC12] Shou-Yu Lee, Dongyang Tang, Tingchao Chen, and WC-C Chu. A qos assurance middleware model for enterprise cloud computing. In *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*, pages 322–327. IEEE, 2012.

[LUC+05] Joshua LeVasseur, Volkmar Uhlig, Matthew Chapman, Peter Chubb, Ben Leslie, and Gernot Heiser. *Pre-virtualization: Slashing the cost of virtualization*. Citeseer, 2005.

[LWK+10] Philipp Leitner, Branimir Wetzstein, Dimka Karastoyanova, Waldemar Hummer, Schahram Dustdar, and Frank Leymann. Preventing sla violations in service compositions using aspect-based fragment substitution. In *Service-Oriented Computing*, pages 365–380. Springer, 2010.

[LZ10] Palden Lama and Xiaobo Zhou. Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 151–160. IEEE, 2010.

[Mar10] Timothy D Martin. Hey-you-get off of my cloud: Defining and protecting the metes and bounds of privacy, security, and property in cloud computing. *J. Pat. & Trademark Off. Soc'y*, 92:283, 2010.

[MBNR68] M Douglas McIlroy, JM Buxton, Peter Naur, and Brian Randell. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98. sn, 1968.

[MG11] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[MGV11] Andreas Menychtas, Anna Gatzioura, and Theodora Varvarigou. A business resolution engine for cloud marketplaces. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 462–469. IEEE, 2011.

[MKF10] Paul Marshall, Kate Keahey, and Tim Freeman. Elastic site: Using clouds to elastically extend site resources. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 43–52. IEEE Computer Society, 2010.

[MLH10] Ming Mao, Jie Li, and Marty Humphrey. Cloud auto-scaling with deadline and budget constraints. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 41–48. IEEE, 2010.

[MNM+07] Megha Mohabey, Y Narahari, Sudeep Mallick, P Suresh, and SV Subrahmanya. A combinatorial procurement auction for qos-aware web services composition. In *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*, pages 716–721. IEEE, 2007.

[MWY+10] Haibo Mi, Huaimin Wang, Gang Yin, Yangfan Zhou, Dianxi Shi, and Lin Yuan. Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 514–521. IEEE, 2010.

[ocw] Opencloudware home page. online.

[Pas05] Adrian Paschke. Rbsla a declarative rule-based service level agreement language based on ruleml. In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, volume 2, pages 308–314. IEEE, 2005.

[PH09] Sang-Min Park and Marty Humphrey. Self-tuning virtual machines for predictable escience. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 356–363. IEEE Computer Society, 2009.

[Pro] CLIF Project. The clif project ow2 web page. online.

[pro11] SLA@SOI project. Sla@soi project homepage. online, 2011.

[Pro12] CloudScale Project. Scalability management for cloud computing. online, 10 2012.

[Qin15] QingCloud. Only pay for the used. online, 2015.

[Qui] Mick Quigley. X. 509 certificates.

[RBX+09] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th international conference on Autonomic computing*, pages 137–146. ACM, 2009.

[RDG11] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 500–507. IEEE, 2011.

[rig07] Rightscale home page. online, 2007.

[Rig15] RightScale. Set up autoscaling using voting tags. Online, 02 2015.

[RJKG11] Bhaskar Prasad Rimal, Admela Jukan, Dimitrios Katsaros, and Yves Goeleven. Architectural requirements for cloud computing systems: an enterprise cloud approach. *Journal of Grid Computing*, 9(1):3–26, 2011.

[RKL+05] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web service modeling ontology. *Applied ontology*, 1(1):77–106, 2005.

[RWQ+08] Omer F Rana, Martijn Warnier, Thomas B Quillinan, Frances Brazier, and Dana Cojocarasu. Managing violations in service level agreements. In *Grid Middleware and Services*, pages 349–358. Springer, 2008.

[SCHM+12] Mike Surridge, Ajay Chakravarthy, Martin Hall-May, Xiaoyu Chen, Bassem Nasser, and Roman Nossal. Serscis: Semantic modelling of dynamic, multi-stakeholder systems. 2012.

[SCW+10] Dawei Sun, Guiran Chang, Chuan Wang, Yu Xiong, and Xingwei Wang. Efficient nash equilibrium based cloud resource allocation by using a continuous double auction. In *Computer Design and Applications (ICCDA), 2010 International Conference on*, volume 1, pages V1–94. IEEE, 2010.

[SEDP+13] Gwen Salaün, Xavier Etchevers, Noel De Palma, Fabienne Boyer, and Thierry Coupaye. Verification of a self-configuration protocol for distributed applications in the cloud. In *Assurances for Self-Adaptive Systems*, pages 60–79. Springer, 2013.

[Ser15] Amazon Web Service. Amazon cloudwatch. online, 2015.

[SGLI11] Bradley Simmons, Hamoun Ghanbari, Marin Litoiu, and Gabriel Iszlai. Managing a saas application in the cloud using paas policy sets and a strategy-tree. In *Proceedings of the 7th International Conference on Network and Services Management*, pages 343–347. International Federation for Information Processing, 2011.

[SH05] Lydia Shenstone and Rob J Hyndman. Stochastic models underlying croston's method for intermittent demand forecasting. *Journal of Forecasting*, 24(6):389–402, 2005.

[Sim02] Kwang Mong Sim. A market–driven model for designing negotiation agents. *Computational Intelligence*, 18(4):618–637, 2002.

[Sim10] Kwang Mong Sim. Towards complex negotiation for cloud economy. In *Advances in Grid and Pervasive Computing*, pages 395–406. Springer, 2010.

[SJ12] Lutz Schubert and Keith Jeffery. Advances in clouds: Research in future cloud computing. *Expert Group Report, Public version*, 1, 2012.

[SK00] Douglas C Schmidt and Fred Kuhns. An overview of the real-time corba specification. *Computer*, 33(6):56–63, 2000.

[SMJ00] Rick Sturm, Wayne Morris, and Mary Jander. *Foundations of Service Level Management*. SAMS Publishing, April 2000.

[SS10] Robert H Shumway and David S Stoffer. *Time series analysis and its applications: with R examples*. Springer Science & Business Media, 2010.

[Sta98] William Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[Szy02] Clemens Szyperski. *Component software: beyond object-oriented programming*. Pearson Education, 2002.

[TAPB10] Cláudio Teixeira, Ricardo Azevedo, Joaquim Sousa Pinto, and Tiago Batista. User provided cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 727–732. IEEE, 2010.

[Ter10] Daniel Teruggi. Prestoprime: Keeping digital content alive. *Cultural heritage on line*, pages 1000–1002, 2010.

[TJDB06] Gerald Tesauro, Nicholas K Jong, Rajarshi Das, and Mohamed N Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 65–73. IEEE, 2006.

[TPP02] Vladimir Tosic, Kruti Patel, and Bernard Pagurek. Wsol-web service offerings language. In *Web Services, E-Business, and the Semantic Web*, pages 57–67. Springer, 2002.

[USC+08] Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, Pawan Goyal, and Timothy Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 3(1):1, 2008.

[Ver04] Dinesh C Verma. Service level agreements on ip networks. *Proceedings of the IEEE*, 92(9):1382–1388, 2004.

[VPR07] Daniel Villela, Prashant Pradhan, and Dan Rubenstein. Provisioning servers in the application tier for e-commerce systems. *ACM Transactions on Internet Technology (TOIT)*, 7(1):7, 2007.

[VRMB11] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[VRMCL08] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[VTM09] Hien Nguyen Van, Frederic Dang Tran, and J-M Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Computer and Information Technology, 2009. CIT'09. Ninth IEEE International Conference on*, volume 1, pages 357–362. IEEE, 2009.

[Wik15a] OpenStack Wiki. Ceilometer, 2015.

[wik15b] wikipedia. Time series. online, 04 2015.

[WIK15c] WIKIPEDIA. Web services description language. online, 4 2015.

[WK94] Sara Williams and Charlie Kindel. The component object model: A technical overview. Technical report, Microsoft Technical Report, 1994.

[WVZX10] Guiyi Wei, Athanasios V Vasilakos, Yao Zheng, and Naixue Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, 54(2):252–269, 2010.

[WXZ+11] Lixi Wang, Jing Xu, Ming Zhao, Yicheng Tu, and Jose AB Fortes. Fuzzy modeling based resource management for virtualized database systems. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 32–42. IEEE, 2011.

[XZF+07] Jing Xu, Ming Zhao, Jose Fortes, Robert Carpenter, and Mazin Yousif. On the use of fuzzy modeling in virtualized data center management. In *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*, pages 25–25. IEEE, 2007.

[ZCS07] Qi Zhang, Ludmila Cherkasova, and Evgenia Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*, pages 27–27. IEEE, 2007.