In proceedings of 5th IEEE Conference on Cloud computing (CLoud 2012), 24-29 June, Waikiki, Hawaii, USA, © IEEE 2012

QoS-Driven Service Selection for Multi-Tenant SaaS

Qiang He, Jun Han, Yun Yang and John Grundy
Faulty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Australia
{qhe, jhan, yyang, jgrundy}swin.edu.au

Hai Jin
Services Computing Technology and Systems Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, China
hjin@hust.edu.cn

Abstract—**Cloud-based software applications (Software as a Service - SaaS) for multi-tenant provisioning have become a major development paradigm in Web engineering. Instead of serving a single end-user, a multi-tenant SaaS provides multiple end-users with the same functionality but with potentially different quality-of-service (QoS) values. The service selection for such a SaaS is a complex decision-making process which involves a number of stakeholders with different QoS requirements. SaaS developers need to compose services with different QoS values to meet end-users' different multi-dimensional QoS constraints for the SaaS. Furthermore, they also need to satisfy SaaS providers' optimisation goals for the SaaS, such as least resource cost and best system performance. Existing QoS-aware service selection approaches are oriented at a single tenant. They do not consider the characteristics of multi-tenant SaaS and hence are ineffective and inefficient when applied to compose multi-tenant SaaS. In this paper, we introduce a novel QoS-driven approach for helping SaaS developers select the services for composing multi-tenant SaaS, which achieves SaaS providers' optimisation goals while fulfilling the end-users' different levels of QoS constraints. The proposed approach is evaluated using an example SaaS synthetically generated based on a dataset of real-world Web services. Experimental results show that our approach significantly outperforms existing approaches in terms of both effectiveness and performance.**

Keywords-Cloud computing; SaaS; Service Composition; Quality of Service; Multi-Tenancy; Optimisation

I. Introduction

Cloud computing provides on-demand provisioning of software applications (Software as a Service - SaaS) [1], which are built on compositions of services that are locally or remotely accessed by an application engine (e.g., a BPEL engine [2]). In order to offer cost-effective solutions to multiple end-users, a SaaS must achieve *multi-tenancy*, i.e., the ability to satisfy multiple end-users in parallel based on a single application instance [3]. These end-users (stakeholders of the SaaS), although require the same functionality, usually have (and often different) multi-dimensional QoS constraints for the SaaS, e.g., response time, throughput and availability. In addition, the SaaS provider (also a stakeholder) often has its own optimisation goal for the SaaS, e.g. least resource cost, best performance or maximised revenues [4]. Moreover, in a cloud environment, the candidate services available (both in-house and external) for composing SaaS often differ in their QoS values. To compose a multi-tenant SaaS, the developer needs to, from the available candidate services, select appropriate services that meet all stakeholders' requirements for the SaaS, including end-users' QoS constraints and SaaS provider's optimisation goal.

The management of a multi-tenant SaaS involves work on different levels, including data-centre level, infrastructure level and application level [5]. In this research we focus on QoS management for SaaS on the application level. The QoS delivered to an end-user can be guaranteed by creating an execution plan using the services with the "right" QoS values. Existing approaches to QoS guarantee and optimisation [6-9] target single-end-user systems - they optimise the quality of a SaaS for only a single end-user. However, a multi-tenant SaaS needs to serve multiple end-users that have potentially different QoS constraints. A possible solution is to adopt one of the existing single end-user oriented optimisation approaches to optimise the QoS for the end-users one by one. However, it is very difficult for such optimisation approaches to achieve the SaaS provider's optimisation goal - while the QoS delivered to individual end-users are locally optimal the overall quality of the SaaS might not be optimal.

According to [10], there has been a more than 130% increase in the number of published Web services from October 2006 to October 2007. The statistics published by webservices.seekda.com, a Web service search engine, also indicate an exponential growth in the number of published Web services. In addition, the proliferation of cloud computing is promoting this growth trend, further increasing the number of available services for composing SaaS [8]. In such a large-scale scenario, applying existing single-tenant-oriented service selection approaches for composing multi-tenant SaaS is very computationally expensive.

In order to address the above issues, this research aims to provide novel techniques and tools to support service selection for multi-tenant SaaS. This work is motivated by the needs to:

- Help SaaS developers capture and model end-users' multi-dimensional QoS constraints for the SaaS.
- Help SaaS developers model SaaS providers' various optimisation goals.

- Help SaaS developers select the services for SaaS that meet all their stakeholders' QoS requirements.
- Help SaaS developers find near-optimal solutions for multi-tenant SaaS efficiently in large-scale scenarios.

This paper introduces a novel QoS-driven approach, named MSSOptimiser (Multi-tenant SaaS Optimiser), for effective and efficient service selection for multi-tenant SaaS. It takes as input the functional business process specification of the SaaS, the stakeholders' QoS requirements and the QoS information of the available candidate services, and generates as output the execution plans for the end-users that are to be executed by the SaaS and meet all stakeholders' QoS requirements. The novelty of MSSOptimiser lies in its full support for composing a multi-tenant SaaS that meets different QoS requirements of multiple stakeholders. The major contributions of this paper include:

- We effectively capture and model the differences in end-users' multi-dimensional QoS constraints, in SaaS providers' optimisation goals and in the quality of the candidate services. This is achieved by modelling the problem of service selection for a multi-tenant SaaS as a constraint optimisation problem (COP).
- For extremely large-scale scenarios where finding an optimal solution to the SaaS optimisation problem is too computationally expensive, we introduce a greedy algorithm to help developers efficiently find a near-optimal solution.
- To evaluate MSSOptimiser, we conduct extensive experiments using a published real-world Web service dataset which contains QoS information about over 2500 real-world Web services. The experiments have shown that MSSOptimiser significantly outperforms existing approaches in terms of both effectiveness and performance.

The rest of this paper is organised as follows: Section II presents an example to motive this research. Section III introduces related work. Section IV introduces the compositional quality model for SaaS quality evaluation. Section V presents the mechanisms for SaaS optimisation based on the compositional quality model. Section VI demonstrates the effectiveness and performance of MSSOptimiser. Section VII concludes the paper and outlines future work.

II. MOTIVATING EXAMPLE

Figure 1 shows an example of multi-tenant SaaS that finds the best used-car offers. Its functionality is represented as a business process that includes five tasks (t_1, …, t_5). This example originates from Alrifai et al. [8] and is adapted to the characteristics of this research. The SaaS serves multiple used-car dealers by processing their requests. The dealers submit requests to the SaaS, specifying the criteria for selecting the cars for their customers (e.g., brand, type and model). In response to each request, the SaaS returns a list of used cars with a loan offer and an insurance quote for each car on the list. These dealers, although requesting the same functionality of finding the best used-car offers, usually have different multi-dimensional constraints for the quality of the

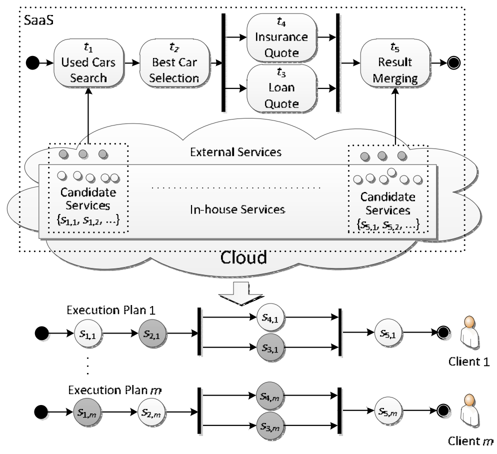

Figure. 1. Multi-tenant SaaS.

SaaS. For instance, one dealer may require a very fast response time despite a high price for such a high-performance SaaS, while another dealer is primarily concerned about the throughput of the SaaS. The SaaS provider, on the other hand, also has its own optimisation goal for the SaaS. Two examples of such goals are 1) to minimise the cost of resource usage for the SaaS, i.e., the total cost of the services selected to serve the end-users, and 2) to maximise the overall SaaS performance regardless of the cost of resource usage.

For each car dealer, an execution plan needs to be created by selecting one service from each of the five sets of functionally equivalent in-house (denoted by light grey circles) and external (denoted by dark grey circles) candidate services. The goal of composing this SaaS is to make sure that the execution plans collectively achieve the SaaS provider's optimisation goal, while separately fulfilling respective dealers' QoS constraints. Similarly to other research efforts [6-9, 11], in this research, we assume that alternative functionally equivalent services are available and can be categorised into different service classes based on their functionalities.

III. RELATED WORK

In recent years, the problem of QoS-aware service selection for composing software applications has received a lot of attention from many researchers. To name a few, in [6, 12], Zeng et al. present AgFlow, a middleware platform that enables QoS-driven composition of Web services. Integer Programming (IP) is used to compute the optimal plan for composite service executions from several execution paths represented by Directed Acyclic Graph (DAG). Following the work in [6, 12], in [7], Ardagna and Pernici formulate the QoS-aware service selection problem as a Mixed Integer Linear Programming (MILP) problem and adopt loops peeling for optimisation. When a feasible solution does not exist, a QoS negotiation algorithm is suggested to enlarge the solution space of the optimisation problem. Berbner et al. [13] design a Web service-based workflow engine named WSQoSX, which aims at optimising QoS-aware service composition under heavy load in real-time. A heuristic algorithm is proposed. Firstly, linear programming is used to relax the mixed integer programming formulation of the service composition problem constructed by Zeng et al. in [6, 12]. Then a backtracking algorithm is used to construct a feasible solution based on the result of the relaxed integer problem. Alrifai et al. [8, 9] adopt a heuristic distributed method to find the best Web services that meet local QoS constraints generated by decomposing global QoS constraints using integer linear programming. The work in [11] models QoS-aware service selection as a 0-1 knapsack problem as well as a multi-constraint optimal path problem. Yu et al. present heuristic algorithms to find near-optimal solutions in polynomial time. For different compositional structures, e.g. sequence, parallel, branches and loops, different algorithms are proposed. In [14], Wang et al. find that optimal solutions can be found in polynomial time for some specially structured service compositions. The common and critical limitation of these existing approaches when applied in cloud computing is that they only support single-tenant SaaS - they try to optimise the QoS for only one end-user. These approaches can be adopted to create execution plans for multiple end-users one after another. The result is that, although the created execution plans can locally fulfil the QoS constraints of corresponding end-users, the overall quality of the SaaS is usually sub-optimal, i.e., the SaaS provider's optimisation goal for the SaaS is not achieved. Furthermore, applying these approaches to compose multi-tenant SaaS is very computationally expensive in large-scale scenarios.

To address the above issues, this paper presents MSSOptimiser, an approach that supports service selection for multi-tenant SaaS. MSSOptimiser can help SaaS developers efficiently select appropriate services to compose an optimal SaaS that fulfils the QoS constraints of multiple end-users.

IV. COMPOSITIONAL QUALITY MODEL

In this section, we present the compositional structures for representing the business processes of SaaS, the quality and utility evaluation methods for SaaS.

A. *Compositional Structures*

Compositional structures describe the order in which the tasks are implemented in the business process of a SaaS. In this research, we consider four types of basic compositional structures, i.e., *sequence*, *branch*, *loop* and *parallel* [7, 15], which are included in BPMN [16] and addressed by BPEL [2] - the de facto standards for specifying service-oriented business processes.

- **Sequence.** In a sequence structure, the services are executed one by one.
- **Branch.** In a branch structure, only one branch is selected for execution. For every set of branches $\{b_1, \dots, b_n\}$, the execution probability distribution $\{p_{b_1}, \dots, p_{b_n}\}$ ($0 \le p_{b_i} \le 1$, $\sum_{i=1}^{n} p_{b_i} = 1.0$) is specified, where p_{b_i}, i=1, …, n, is the probability that b_i is selected for execution.
- **Loop.** In a loop structure, the loop is executed for n ($n \ge 0$) times. For every loop, the probability distribution $\{p_0, \dots, p_{MNI}\}$, ($0 \le p_i \le 1$, $\sum_{i=0}^{MNI} p_i = 1.0$) is specified, where p_i, i=0, …, *MNI*, is the probability that the loop iterates for i times and *MNI* is the expected Maximum Number of Iterations for the loop.
- **Parallel.** In a parallel structure, all the branches are executed at the same time.

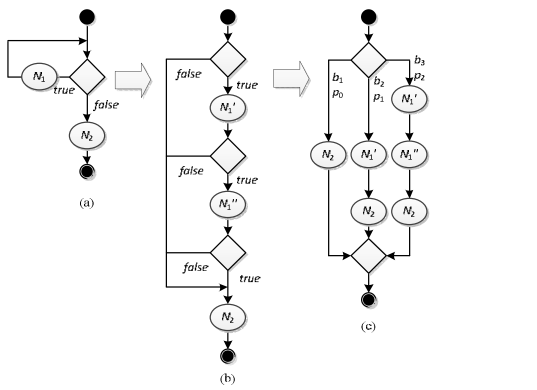

Figure 2. The loop peeling process.

p_{b_i}, p_i and *MNI* can be evaluated based on the SaaS's past executions or can be specified by the SaaS developer [6, 7]. In this research, we assume that for every loop, the *MNI* is determined. Otherwise, if an upper bound for the number of iterations for a loop does not exist, the QoS delivered by the execution plan that contains the loop cannot be calculated since the loop can iterate infinitely.

B. *Quality Evaluation*

The execution plan created for an end-user must meet the end-user's multi-dimensional QoS constraints. Thus, we need to evaluate the QoS delivered by an execution plan, considering all its execution paths. An execution path is a set of services forming a sequential path from the initial service to the final service of an execution plan. Take execution plan 1 in Figure 1 for example, there are two execution paths: $ep_1 = s_{1,1}\text{-}s_{2,1}\text{-}s_{3,1}\text{-}s_{5,1}$ and $ep_2 = s_{1,1}\text{-}s_{2,1}\text{-}s_{4,1}\text{-}s_{5,1}$. For a SaaS $\mathbb{S}$, let epl_k (k=1, 2, …) be its execution plans, $ep_{k,i}$ (i=1, 2, …) be epl_k's execution paths, $s_{k,i,j}$ (j=1, 2, …) be $ep_{k,i}$'s constituent services, the QoS delivered by the execution plans and the overall quality of the SaaS can be evaluated using the aggregation functions presented in Table 1 [6]. For example, the cost of execution plan 1 in Figure 1 can be calculated by $cost(epl_1) = \sum_{i=1}^{5} s_{i,1} \cdot q_{price}$ and the cost of the SaaS (given m=3) can be calculated by $cost(\mathbb{S}) = \sum_{i=1}^{3} cost(epl^i)$.

In this paper, examples are based on a subset of QoS parameters, including price (or cost) and response time, which also have been the basis for QoS consideration in other approaches [6-8]. Other QoS parameters can be generalised as added dimensions in the quality model.

Numerical QoS parameters can be divided into two categories: positive and negative QoS parameters. A positive QoS parameter is a QoS parameter whose evaluation will increase as its value increases, e.g., availability and throughput. A QoS parameter is a negative QoS parameter whose evaluation will decrease as its value increases, e.g., price (or cost) and response time. To accommodate non-numerical QoS parameters (e.g., reputation) that are expressed by a rating selected from {very high, high,

TABLE I. QoS AGGREGATION FUNCTIONS

Quality Parameter	Aggregation Function
Cost	$cost(epl_k) = \sum_{s_{k,i,j} \in epl_k} s_{k,i,j} \cdot q_{price}$ $cost(\mathbb{S}) = \sum_{epl_k \in \mathbb{S}} cost(epl_k)$
Response Time	$resTime(epl_k) = \max_{ep_i \in epl_k} (\sum_{s_i^j \in ep_i} s_i^j \cdot q_{resTime})$ $resTime(\mathbb{S}) = \text{average}_{epl_k \in \mathbb{S}} (resTime(epl_k))$
Availability	$availability(epl_k) = \prod_{s_i \in epl_k} s_i \cdot q_{availability}$ $availability(\mathbb{S}) = \text{average}_{epl_k \in \mathbb{S}} (availability(epl_k))$
Throughput	$throughput(epl_k) = \min_{ep_i \in epl} (\min_{s_i^j \in ep_i} (s_i^j \cdot q_{throughput}))$ $throughput(\mathbb{S}) = \sum_{epl_k \in \mathbb{S}} throughput(epl_k)$

medium, low, very low}, the approach proposed by Mumtaz et al. in [17] is adopted. Based on a pre-defined semantics-based hierarchical structure of all possible values of a non-numerical QoS parameter, each level of the hierarchy is associated with a numerical value. In this way, the utility of the QoS parameter can be calculated and the QoS parameter can be termed negative or positive. If the levels that are more preferable to end-users are assigned with higher values, the QoS parameter is treated as a positive QoS parameter, and vice versa.

In the remainder of this paper, we use negative QoS parameters and omit mostly repetitive yet similar introduction for positive QoS parameters.

C. Utility Evaluation

Functionally equivalent services usually differ in multiple QoS parameters. For example, the *Insurance Quote* service provided by a service provider may have lower response time but require higher price than the *Insurance Quote* service provided by another service provider. Selecting from these services by their QoS characteristics is a multi-attribute decision making problem. For the purpose of ranking and sorting the services in a same service class, a method is needed to evaluate a given service based on its multiple QoS parameters. In this research, we use the method adopted in [6-9] for service utility evaluation.

Given the utility of the services, denoted by $u(s_i)$, the utility of an execution path ep_j composed by n ($n \geq 1$) services $s_1, \ldots, s_n$, can be calculated by:

$$u(ep_j) = \sum_{i=1}^{n} u(s_i) \tag{1}$$

Let ef_i be the execution frequency (i.e., execution probability) of an execution path ep_i in an execution plan epl_j. The utility calculation of epl_j must consider all its execution paths according to their execution frequencies:

$$u(epl_j) = \sum_{i=1}^{n} ef_i \cdot u(ep_i) \tag{2}$$

The overall utility of the SaaS, denoted by $u(\mathbb{S})$, that consists of m ($m \geq 1$) execution plans can be calculated by:

$$u(\mathbb{S}) = \sum_{i=1}^{m} w_i \cdot u(epl_i) \tag{3}$$

where w_i is the SaaS provider's preference and priority for the i^{th} end-user, $w_i \in [0,1]$ and $\sum_{i=1}^{m} w_i = 1$. For example, given m=3, the utility of the SaaS presented in Figure 1 is calculated by: $u(\mathbb{S}) = w_1 \cdot u(epl_1) + w_2 \cdot u(epl_2) + w_3 \cdot u(epl_3)$.

V. SYSTEM OPTIMISATION

A multi-tenant SaaS requires that all end-users' multi-dimensional QoS constraints for the SaaS are fulfilled. Thus, for each end-user, an execution plan needs to be created that fulfils the end-user's QoS constraints. Meanwhile, all the execution plans must collectively achieve the SaaS provider's optimisation goal. In this section, we present the optimisation formulation and techniques that support service selection for such a multi-tenant SaaS.

A. Determining Execution Plans

Suppose the business process of a SaaS $\mathbb{S}$ consists of n ($n \geq 1$) tasks. Assume there are n service classes S_i, i=1, …, n, each containing r ($r \geq 1$) available services $s_{i,j}$, j=1, …, r, that provide the same functionality but potentially differ in t QoS parameters q_p, p=1, …, t, The problem of service selection for $\mathbb{S}$ that serves m ($m \geq 1$) end-users is a constraint optimisation problem that aims at finding the services for creating m execution plans epl_k, k=1, …, m, that fulfil corresponding end-users' t-dimensional QoS constraints $c_{k,p}$, k=1, …, m, p=1, …, t, while achieving the SaaS provider's optimisation goal *objective*($\mathbb{S}$). In this research, we assume that $r \geq m$, i.e., there are enough candidate services in each service class to be selected exclusively for creating an execution plan for each end-user. If $r<m$, more candidate services need to be discovered through sources of services, e.g., public UDDI registry and service search engine.

To model end-users' different multi-dimensional QoS constraints, we first model the service selection for a multi-tenant SaaS as a constraint satisfaction problem (CSP), which consists of a finite set of variables $X=\{x_1, \ldots, x_n\}$, with respective domains $D=\{D_1, \ldots, D_n\}$ listing the possible values for each variable, and a set of constraints $C=\{c_1, \ldots, c_t\}$ over X. A solution to a CSP is an assignment of a value to each variable from its domain such that every constraint is satisfied. The CSP model of the above problem can be formally expressed as follows.

For m end-users, there are $m \times n \times r$ 0-1 variables $X_{k,i,j}$ (k=1, 1, …, m, i=1, …, n, j=1, …, r and $D_{k,i,j}=\{0, 1\}$), $X_{k,i,j}$ being 1 if the j^{th} candidate service in the i^{th} service class is selected to create the execution plan for end-user k, 0 otherwise. The constraints for the CSP model are:

$$\sum_{j=1}^{r} X_{k,i,j} = 1 \quad \forall k \in [1,m] \;\; i \in [1,n] \tag{4}$$

$$\sum_{k=1}^{m} X_{k,i,j} = 1 \quad \forall i \in [1,n] \;\; j \in [1,r] \tag{5}$$

$$\sum_{k=1}^{m} \sum_{j=1}^{r} X_{k,i,j} = m \quad \forall i \in [1,n] \tag{6}$$

$$epl_k.q_p < c_{k,p} \quad \forall k \in [1,m],\; p \in [1,t] \tag{7}$$

where $epl_k.q_p$ is the p^{th} QoS parameter of the execution plan for end-user k and can be obtained by applying the QoS aggregation functions presented in Table 1.

Constraint family (4) guarantees that only one candidate service is selected in each class for one end-user. Constraint family (5) guarantees that one service can only be selected for one end-user. Constraint family (6) ensures that a total of m candidate services are selected in each service class. Constraint family (7) ensures that each end-user's t-dimensional QoS constraints are fulfilled by the corresponding execution plan.

Solving the above CSP can generate a solution consisting of m execution plans that fulfil the corresponding end-users' QoS constraints. Such a solution is called a *feasible solution*. Very often, there are many feasible solutions that differ in their QoS values, e.g., overall SaaS utility and overall cost. Now we seek to achieve the SaaS provider's optimisation goal for the SaaS. Given an objective function that represents the SaaS provider's optimisation goal, the CSP is turned into a constraint optimization problem (COP). In a COP, each solution generated by solving the CSP is associated with a ranking value for the objective function. The solution with the optimal ranking value is the *optimal solution* to the COP.

System providers' optimisation goals can be various, which can be represented using different objective functions. In this research, we use the objective functions for two typical optimisation goals as examples: 1) to maximise the overall SaaS utility; and 2) to minimise the overall cost of resource usage for the SaaS.

- **Maximising the overall SaaS utility.** This optimisation goal is to maximise the overall utility of the SaaS. The objective function that captures this optimisation goal is as follows:

$$objective(\mathbb{S})\text{: maximise } \sum_{k=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{r}\sum_{p=1}^{t} w_{k,p} \times u(s_{i,j}.q_p) \times X_{k,i,j} \quad (8)$$

  where $w_{k,p}$ is the weight that represents the kth end-user's preference and priority for the pth QoS parameter of the SaaS.

- **Minimising the overall cost of resource usage**. This optimisation goal is to minimise the total price of the selected services. The objective function that captures this optimisation goal is as follows:

$$objective(\mathbb{S})\text{: minimise } \sum_{k=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{r} s_{i,j}.price \times X_{k,i,j} \quad (9)$$

  where $s_{i,j}.price$ is the price of the jth candidate service in the ith service class.

This COP can be solved by applying Integer Programming techniques [6] (or the Mixed Integer Programming technique [7, 8] if decimal variables are involved). Based on the results from solving the COP, execution plans can be created for the end-users, which individually meet their QoS constraints and collectively achieve the SaaS provider's optimisation goal.

B. *Find Near-Optimal Solutions*

In service selection for multi-tenant SaaS, given n service classes, each containing r candidate services, there are $(P_r^m)^n$ possible combinations of services for a SaaS that serves m end-users. It is practically tractable only when the number of candidate services is small. The pay-per-use business model driven by cloud computing and SaaS enables service providers to offer their services to end-users with different QoS values and prices. As a result, the number of candidate services available for SaaS is expected to grow dramatically in the foreseeable future. In such scenarios, using Integer Programming to find the optimal solution to the COP described in Section V.A can be very computationally expensive. The skyline technique [18] can be applied to reduce the search space of the COP [8]. However, after pruning the non-skyline services, the number of remaining candidate services may still be too large for the SaaS optimisation problem to be solved efficiently. The possible reasons are twofold. First, the QoS parameters of the services are often anti-correlated. Second, in practice, the number of skyline services increases rapidly with the dimensionality of their QoS, i.e., the number of their QoS parameters [19].

In such cases, MSSOptimiser provides a greedy algorithm to help the SaaS developer find a near-optimal solution to the SaaS optimisation problem efficiently. The greedy algorithm always selects the most representative candidate services that are more likely to be part of the solution that achieves the optimisation goal. To serve m end-users, m services need to be finally selected from each service class to create m execution plans. Thus, the greedy algorithm starts with selecting the first batch of m most representative candidate services from each service class. Then, the selected representative candidate services are inserted into the search space of the COP (see Section V.A) and the COP is solved. If no solution can be found using these candidate services, another batch of the m most representative candidate services are selected from each service class and added to the search space of the COP. This process is repeated until a solution is found or until it is determined that a solution cannot be found, i.e., all the skyline services have been considered and yet no solution can be found.

The criterion for selecting the representative candidate services is dependent on the SaaS provider's optimisation goal. For example, if the optimisation goal is to minimise the overall cost of resource usage, the greedy algorithm will give preferences to the candidate services with the lowest prices. If the optimisation goal is to maximise the overall SaaS utility, the greedy algorithm will always, from the remaining candidate services, select the ones with the highest utility values. However, the end-users of the SaaS usually have different priorities and preferences for different QoS parameters. Thus, a utility function is needed for evaluating the average utility of a given service across all end-users. Suppose there are m end-users, we average the weights that represent individual end-users' preferences and priorities for different QoS parameters to calculate an average weight for each of the t QoS parameters considered in the calculation of the average utility:

$$w_{ave,p} = \frac{1}{m} \cdot \sum_{i=1}^{m} w_{i,p} \text{ , } p=1, \ldots, t \quad (10)$$

where $w_{i,p} \in [0, 1]$, $\sum_{i=1}^{m} w_{i,p} = 1$ and $w_{i,p}$ is the weight that represents the i^{th} end-user's preference and priority for the k^{th} QoS parameter.

Then, the average utility of a given service $s_{i,j}$ with t QoS parameters across all end-users is calculated as:

$$u_{ave}(s_{i,j}) = \sum_{p=1}^{t} w_{ave,p} \times u(s_{i,j}.q_p) \tag{11}$$

C. Integrated Optimisation Methods

Combining the techniques introduced above, including integer programing (Section V.A), skyline computation (Section V.B) and greedy algorithm (Section V.B), MSSOptimiser provides three different SaaS optimisation methods: 1) *Exact-Global*. This optimisation method creates execution plans for all end-users in one COP model, considering all the services in each service class as candidate services. This method is suitable for small-scale scenarios where the skyline computation and the greedy algorithm are unnecessary. 2) *Skyline-Global*. This optimisation method creates execution plans for all end-users in one COP model, giving preferences to the skyline services in each service class as candidate services. This method is suitable for large-scale scenarios where the optimisation problem can be solved efficiently after pruning the non-skyline services. 3) *Greedy-Global*. This optimisation method creates execution plans for all end-users in one COP model, using the greedy algorithm and giving preferences to the most representative candidate services in each service class as candidate services. This method is suitable for extremely large-scale scenarios where the optimisation problem still cannot be solved efficiently after pruning the non-skyline services.

VI. EXPERIMENTS

This section presents the experimental evaluation of our approach, focusing on the comparison with existing optimisation approaches in terms of effectiveness (measured by the success rate of finding an optimisation solution) and performance (measured by the computation time taken to find an optimisation solution).

A. Prototype Implementation

We have implemented MSSOptimiser in Java using JDK 1.6.0 and Eclipse Java EE IDE. The prototype includes three main modules: a skyline operator, a greedy representative candidate services selector and an integer programming problem solver. For solving the COPs, we used CPLEX v12.2, a commercial solver developed by IBM. By integrating the modules, the prototype realised the three SaaS optimisation methods presented in Section V, i.e., Exact-Global, Skyline-Global and Greed-Global. Given the functional specification of the business process of a SaaS, the quality information about the candidate services, a set of end-users' quality constraints and an optimisation objective, the prototype, using the selected optimisation method, generates the execution plans that separately fulfil the end-users' QoS constraints and collectively achieve the optimisation objective.

B. Experimental Setup

We evaluated MSSOptimiser using the prototype and a publicly available Web service dataset QWS [10], which comprises measurements of nine QoS parameters of over 2500 real-world Web services. The information about the services was collected from public UDDI registries, search engines and service portals. Their QoS values were measured using commercial benchmark tools. For large scenarios that involved more than 2500 services, we created extra services based on QWS. In the experiments, we considered a SaaS whose business process consists of five tasks, as presented in Figure 1. Accordingly, we randomly partitioned the services in QWS into five categories as if they were the five classes of candidate services corresponding to the five tasks. We added a randomly generated price to each candidate service as an additional QoS parameter. End-users' QoS constraints were randomly generated. To compare our approach with existing approaches, we implemented the optimisation approaches presented in [7] and [8]. Specifically, we compared our SaaS optimisation methods with the following methods using average results from 100 instances for each set of experiments:

In the experiments, we utilised the example SaaS presented in Figure 1. Accordingly, we randomly partitioned the services in QWS into five categories as if they were the five classes of candidate services corresponding to the five tasks. End-users' QoS constraints were randomly generated according to normal distributions from intervals whose lower and upper bounders are determined using the worst and best QoS values of all candidate services in each service class. To compare our approach with existing approaches, we implemented the optimisation approaches presented in [7] and [8] as they are also based on integer programming techniques - as far as we understand the most popular and representative techniques adopted in research on QoS-aware service compositions. Specifically, we compared our SaaS optimisation methods with the following methods using average results from 100 instances for each set of experiments: 1) *Skyline-Local*. This optimisation method adopts the approach presented in [8], which creates execution plans for the end-users one by one in different COP models, giving preferences to the skyline services in each service class as candidate services. 2) *Exact-Local*. This optimisation method adopts the approach presented in [7], which creates execution plans for the end-users one by one in different COP models, considering all the services in each service class as candidate services. 3) *Greedy-Local*. This optimisation method adopts a greedy algorithm similar to the method presented in [8], which creates execution plans for the end-users one by one in different COP models, giving preferences to the most representative candidate services in each service class as candidate services.

The experiments were conducted on a machine with AMD Athlon(tm) X4 640 3.00GHz CPU and 8 GB RAM, running Windows 7 x64 Ultimate.

C. Experimental Results

In this section, we present the experimental results and compare our global methods with their corresponding local methods, i.e., Exact-Local vs. Exact Global, Skyline-Local vs. Skyline-Global and Greedy-Local vs. Greedy-Global.

We first compare the effectiveness of the six optimisation methods by their success rates of SaaS optimisation, i.e., the percentage of scenarios where a solution could be found that met all stakeholders' QoS requirements. In this series of experiments, we fixed the number of candidate services per class at 100 and changed the number of end-users from 10 to 60 at steps of 10. As illustrated in Figure 3, the global methods significantly outperform the local methods. The global methods maintain a very high level of success rate (above 90%) across all scenarios while the success rate obtained by the local methods decrease quickly from 100% to 0% as the number of end-users increases from 10 to 60. The consistency of the success rates obtained by the approaches in the global approaches family or the local approaches family demonstrates that the skyline technique and the greedy algorithm do not negatively impact the effectiveness of the optimisation methods.

We also compared the SaaS utility obtained (when a solution was found) by the six optimisation methods. The SaaS utility, which is cumulative, increases as the number of end-users increases (see formula (3)). Thus, we use the utility per end-user, calculated by $u(\mathbb{S})/m$, where m is the number of end-users, as the measurement for the evaluation in this set of experiments. As presented in Figure 4, the global methods beat the local methods. In particular, Exact-Global and Skyline-Global yielded the highest utility per end-user while the Greedy-Local and the Skyline-Local methods yielded the lowest and second lowest utility per end-user. However, the *utility optimalities* of the two greedy methods, evaluated by $u_{greedy}/u_{optimal}$, where u_{greedy} and $u_{optimal}$ are the utility per end-user obtained by the greedy methods and the optimal methods respectively, are all above 95% in all scenarios when a solution can be found. In Figures 3 and 4 (as well as Figure 5 (b)), the data for the local optimisation methods in scenarios where the number of end-users exceeds 50 are missing because in these scenarios no solution could

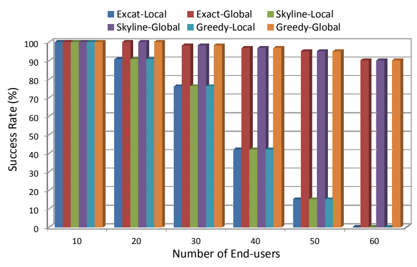

Figure 3. Successful rate vs. number of end-users.

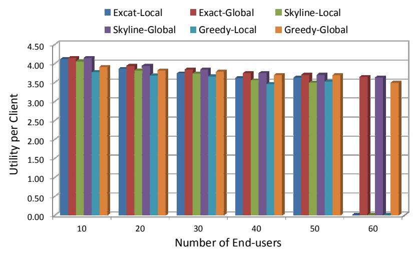

Figure 4. System utility vs. number of end-users.

be found using local optimisation methods.

The optimisation of service selection for a multi-tenant SaaS comes at a price - the computational overhead measured by computation time. In large-scale scenarios, the computational overhead of the optimisation methods is a very important concern to SaaS developers. To compare the computational overhead of the six different optimisation methods, we conducted a series of experiments in different scenarios, where the scales vary in three different aspects that affect the computational overhead of the optimisation methods:

- the number of QoS constraints, which determines the number of each end-user's QoS constraints for the COP;
- the number of end-users, which determines the number of sets of end-users' QoS constraints for the COP; and
- the number of candidate services per service class, which determines the size of the original search space of the COP.

As presented in Figures 5 (a), (b) and (c), the global methods significantly beat corresponding local methods in most cases. In particular, Greedy-Global clearly outperforms all other methods across all scenarios. Skyline-Global, showing a performance similar to Greedy-Local, beats the remaining three methods. Thus, Greedy-Global is the best option if performance is the priority, while Skyline-Global is the best option if the SaaS being optimised to the fullest extent is desirable.

In Figure 5 (a), as the number of QoS constraints exceeds three, Skyline-Local and Skyline-Global start to significantly outperform Exact-Local and Exact-Global respectively because the number of skyline services starts to grow rapidly. However, as the number of QoS constraints continues to increase, the outperformance margins start to decrease. The reason is that the number of skyline services in each service class is approaching the totasl number of candidate services in the service class, giving Skyline-Local and Skyline-Global

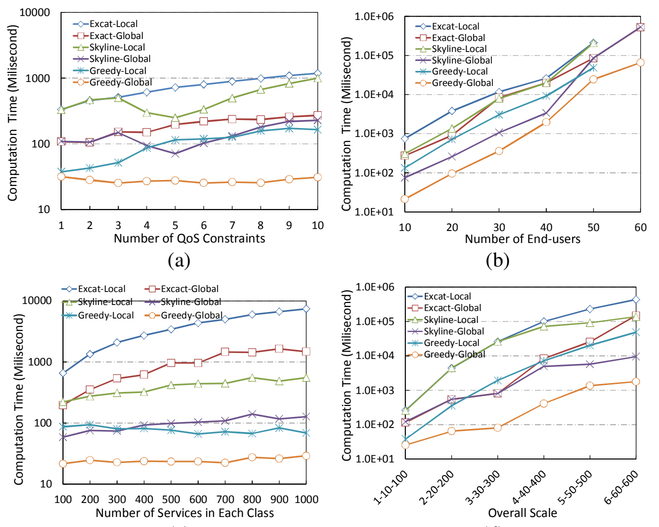

Figure 5. Computation time: (a) vs. number of QoS constraints (b) vs. number of end-users (c) vs. number of services in each class (d) vs. overall scale (number of QoS constraints – number of end-users – number of candidate services per class).

less advantage over Exact-Local and Exact-Global.

Figure 5 (b) shows that Skyline-Global outperforms Exact-Global until the number of end-users reaches 50. In the scenarios with 50 and 60 end-users, the number of skyline services is not enough for creating enough execution plans for the end-users. In such cases, Skyline-Global had to use all the services in each class as candidate services. The situation of Skyline Local vs. Exact Local is the same.

Figure 5 (c) shows that the greedy methods perform clearly better (especially Greedy-Global) than other methods in scenarios where the number of candidate services per class is significantly larger than the number of end-users (10 in this series of experiments).

In order to compare the computational overhead of the six optimisation methods against the three aspects combined, i.e., the numbers of QoS parameters, end-users and candidate services per class, we conducted a set of experiments where the scale of the scenarios varies in all the above three aspects. The results are presented in Figure 5 (d). Again, Greedy-Global outperforms all other methods remarkably. Skyline-Global, showing the second best performance, starts to significantly outperform the other four methods as the number of QoS constraints exceeds 3. The results from this series of experiments show that Greedy-Global and Skyline-Global are still the best two options.

VII. CONCLUSIONS

In this paper, we have proposed MSSOptimiser, a QoS-driven approach which supports the service selection for multi-tenant cloud-based software applications (Software as a Service - SaaS). Using optimisation techniques, particularly Integer Programming, it helps SaaS developers determine the optimal services for a multi-tenant SaaS that meet different stakeholders' QoS requirements, including the optimisation goal of the SaaS provider and the different levels of QoS constraints of different end-users. In large-scale scenarios where the SaaS optimisation problem is computationally expensive, MSSOptimiser provides a greedy algorithm to find a near-optimal solution efficiently. We have evaluated MSSOptimiser using an example SaaS synthetically generated based on a large real-world Web services dataset, and compared the effectiveness and performance of the proposed approach to existing approaches. The evaluation has shown that the proposed SaaS optimisation methods outperform existing methods significantly in terms of both effectiveness and performance. In particular, Skyline-Global showed the best effectiveness at very reasonable computational overhead and Greedy-Global showed remarkably high performance with less than 5% sacrifice in utility optimality.

In future work, we plan to apply the proposed approach to realise SaaS re-optimisation for runtime SaaS adaptation. In addition, we intend to investigate the scalability of the proposed approach in scenarios where the number of end-users is very large.

ACKNOWLEDGMENTS

This work is funded by the Australian Research Council in collaboration with CA Labs.

REFERENCES

[1] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's Inside the Cloud? An Architectural Map of the Cloud Landscape," in *1st International Workshop on Software Engineering Challenges for Cloud Computing (ICSE CLOUD2009)*, Washington, DC, USA, 2009, pp. 23-31.

[2] OASIS. (2007). *Web Services Business Process Execution Language Version 2.0*. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[3] F. Chong and G. Carraro, "Architecture Strategies for Catching the Long Tail," *MSDN Library,* 2006.

[4] D. Ardagna, B. Panicucci, and M. Passacantando, "A Game Theoretic Formulation of the Service Provisioning Problem in Cloud Systems," in *20th International Conference on World Wide Web*, Hyderabad, India, 2011, pp. 177-186.

[5] J. Fiaidhi, I. Bojanova, J. Zhang, and L.-J. Zhang, "Enforcing Multitenancy for Cloud Computing Environments," *IT Professional* vol. 14, pp. 16-18, 2012.

[6] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering,* vol. 30, pp. 311-327, 2004.

[7] D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Transactions on Software Engineering,* vol. 33, pp. 369-384, 2007.

[8] M. Alrifai, D. Skoutas, and T. Risse, "Selecting Skyline Services for QoS-based Web Service Composition," in *19th International Conference on World Wide Web (WWW2010)*, Raleigh, North Carolina, USA, 2010, pp. 11-20.

[9] M. Alrifai and T. Risse, "Combining Global Optimization with Local Selection for Efficient QoS-Aware Service Composition," in *18th International Conference on World Wide Web (WWW2009)*, Madrid, Spain, 2009, pp. 881-890.

[10] E. Al-Masri and Q. H. Mahmoud, "Investigating Web Services on the World Wide Web," in *17th International Conference on World Wide Web (WWW2008)*, 2008, pp. 795-804.

[11] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Transactions on the Web,* vol. 1, 2007.

[12] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality Driven Web Services Composition," in *12th International Conference on World Wide Web (WWW2003)*, Budapest, Hungary, 2003, pp. 411-421.

[13] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Heuristics for QoS-aware Web Service Composition," in *IEEE International Conference on Web Services (ICWS2006)*, Chicago, Illinois, USA, 2006, pp. 72-82.

[14] J. Wang, J. Wang, B. Chen, and N. Gu, "Minimum Cost Service Composition in Service Overlay Networks," *World Wide Web,* vol. 14, pp. 75-103, 2011.

[15] W.-L. Wang, D. Pan, and M.-H. Chen, "Architecture-based Software Reliability Modeling," *Journal of Systems and Software,* vol. 79, pp. 132-146, 2006.

[16] Object Management Group. (2011). *Business Process Model And Notation (BPMN) Version 2.0*. Available: http://www.omg.org/spec/BPMN/2.0/PDF/

[17] S. Mumtaz, A. Villazon, and T. Fahringer, "Grid Allocation and Reservation - Grid Capacity Planning with Negotiation-based Advance Reservation for Optimized QoS," in *ACM/IEEE Conference on High Performance Networking and Computing (SC2006)*, Tampa, FL, USA, 2006.

[18] S. Börzsönyi, D. Kossmann, and K. Stocker, "The Skyline Operator," in *International Conference on Data Engineering (ICDE2001)*, Washington, DC, USA, 2001, pp. 421-430.

[19] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, "Finding k-Dominant Skylines in High Dimensional Space," in *ACM SIGMOD International Conference on Management of Data (SIGMOD2006)*, Chicago, Illinois, USA, 2006, pp. 503-514.