Adaptive Management of Composite Services under Percentile-based Service Level Agreements

Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, and Francesco Lo Presti

Università di Roma "Tor Vergata", Viale del Politecnico 1, 00133 Roma, Italy
{cardellini,casalicchio}@ing.uniroma2.it,
{vgrassi,lopresti}@info.uniroma2.it

Abstract. We present a brokering service for the adaptive management of composite services. The goal of this broker is to dynamically adapt at runtime the composite service configuration, to fulfill the Service Level Agreements (SLAs) negotiated with different classes of requestors, despite variations of the operating environment. Differently from most of the current approaches, where the performance guarantees are characterized only in terms of bounds on average QoS metrics, we consider SLAs that also specify upper bounds on the percentile of the service response time, which are expected to better capture user perceived QoS. The adaptive composite service management is based on a service selection scheme that minimizes the service broker cost while guaranteeing the negotiated QoS to the different service classes. The optimal service selection is determined by means of a linear programming problem that can be efficiently solved. As a result, the proposed approach is scalable and lends itself to an efficient implementation.

1 Introduction

The Service Oriented Architecture (SOA) paradigm encourages the construction of new applications through the composition of loosely coupled network-accessible services offered by independent providers. One of the underlying ideas is that different providers may offer different implementations of the same functionality, differentiated by their quality of service (QoS) and cost attributes, thus allowing a prospective user to choose the services that best suit his/her needs. To state the respective expectations and obligations concerning the delivered QoS and cost, users and providers of services engage in a negotiation process that culminates in the definition of a *Service Level Agreement* (SLA) contract.

Given a SOA system consisting of a composition of services, the fulfillment of the QoS requirements stated in its SLA is a challenging task that requires the system to take complex decisions within short time periods, because of the intrinsically dynamic and unpredictable nature of the SOA operational environment. A promising way to manage effectively this task is to make the system able to

self-configure at runtime in response to changes in its operational environment. In this way, the system can timely react to environment changes (concerning for example the available resources or the type and amount of user requests), to keep the ability of fulfilling at runtime the QoS requirements stated in a SLA, thus avoiding SLA violations that could cause loss of income and reputation.

Several methodologies have been already proposed to drive the self-configuration of QoS-aware SOA systems. Most of them (*e.g.*, [1, 2, 4, 15, 16]) address this issue as a *service selection* problem: given the set of functionalities (*abstract services*) needed to compose a new added value service, the goal is to identify at runtime a set of *concrete services* (one for each abstract service) that implement them, selecting it from a pool of candidates. Each selected concrete service is then dynamically bound to the corresponding abstract service. Other methodologies [5, 9, 10] extend this idea by also considering the possibility of binding each abstract service to a set of functionally equivalent concrete services rather than a single service, coordinated according to some redundancy pattern (*e.g.*, 1-out-of-N or sequential retry), to achieve higher QoS at the expense of higher cost: in this case both the redundancy pattern and the set of equivalent concrete services must be selected at runtime.

The proposed methodologies also differ in the type of scenario they deal with: most of them (*e.g.*, [1, 2, 10, 15, 16]) deal with *single* requests for the composite service independently of each another. The goal in this case is to determine the concrete implementation of each abstract service for that request that is best suited to satisfy the requestor SLA given the current conditions of the operating environment. Others [4, 5] jointly consider the aggregate *flow* of requests. In this case, the goal is to determine how to switch different flows of requests, possibly generated by several classes of users, to the different candidate implementations as to satisfy the different SLAs.

Most of the proposed approaches for self-configurable QoS-aware SOA systems consider SLAs where the performance guarantees are specified only in terms of bounds on the expected values of the QoS metrics of interests. A potential limitation of these approaches lies in the fact that the user perceived QoS is often better expressed in terms of bounds on the percentile of the QoS metrics, as also reflected in commercial practices. For example, the Amazon SOA-based e-commerce platform [7] includes SLAs concerning the 99.9 percentile of the response time under a given peak load of service requests per second. To the best of our knowledge, only the approaches proposed in [8, 14] offer guarantees on the percentile of the response time. The results in [14], though, are limited to sequential patterns and only apply to the single request scenario, while [8] proposes a heuristic for request scheduling in a single database server which is based on the execution time prediction.

In this paper, we overcome this limitation of current methodologies for self-configurable QoS-aware SOA systems. We consider the *flow* scenario and propose a service selection scheme to drive the self-configuration of composite SOA applications, which considers SLAs that include performance guarantees on the percentiles of the QoS attributes. We present our solution from the perspective

of an application implemented as a composite service and managed by an intermediary broker. We show that the application can efficiently provide the SLAs by selecting, among the pool of available services, those services that allow it to fulfill the SLAs negotiated with the users, given the constraints defined by the SLAs settled with the providers. The selection is driven by the goal of maximizing some broker utility goal. The search for a new solution is triggered by the occurrence of events that could make no longer valid a previously calculated solution, *e.g.*, the arrival or departure of a user or a change in the set of providers. We formulate the service selection problem as an optimization problem with non-linear constraints. For the solution, we linearize the constrains. The resulting linear programming problem can thus be efficiently solved via standard techniques. Hence the proposed approach is suitable for on-line operations.

The rest of the paper is organized as follows. In Sect. 2 we provide an overview of the system architecture offering the composite service with percentile-based SLAs and outline the SLA definition. In Sect. 3 we discuss how to compute the QoS attributes of the composite service. In Sect. 4 we present the formulation of the optimization problem that is solved to determined the percentile-based service selection. Then, in Sect. 5 we present the simulation experiments to assess the effectiveness of the proposed approach. Finally, we draw some conclusions and give hints for future work in Sect. 6.

2 System Architecture

We present our approach from the perspective of an application architected as a composite service and provided by an intermediary *service broker*. The service broker offers to prospective users a composite service with a range of different service classes. It acts as a full intermediary between users and concrete services, performing a role of service provider towards the users and being in turn a requestor to the concrete services offering the operations used to implement the composite service. Its main task is to drive the adaptation of the composite service it manages to fulfill the SLAs negotiated with its users, given the SLAs it has negotiated with the concrete services. Moreover, it also aims at optimizing a given utility goal.

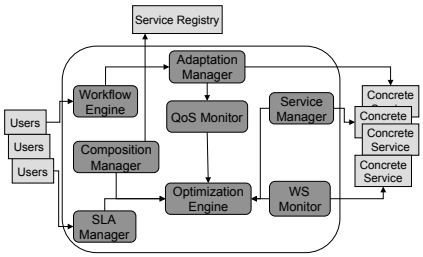


Fig. 1. Service broker high-level architecture.

Figure 1 shows the core components of the broker high-level architecture and their interaction. A detailed description of the architecture can be found in [5]; in the next, we summarize the respective tasks of the components. The *Composition Manager* describes the composite service in some suitable workflow orchestration language (e.g., BPEL [13]) and identifies the concrete services implementing the required functionalities of the abstract composition (including their SLAs).

The *Workflow Engine* is the software platform executing the business process and represents the user front-end for the composite service provisioning. For each invocation of the component services it interacts with the *Adaptation Manager*. The latter binds dynamically the request to the real endpoint that represents the concrete service, which is identified through the solution of the optimization problem presented in Sect. 4. Together, the Workflow Engine and the Adaptation Manager manage the user requests flow, once the user has been admitted to the system with an established SLA. The *Optimization Engine* is the component that solves the broker optimization problem. The parameters values for this problem are derived from the parameters of the SLAs negotiated with the composite service users and concrete services, and from a monitoring activity carried out by the QoS Monitor and the WS Monitor.

The *QoS Monitor* collects information about the performance and availability levels (specified in the SLAs) perceived by the users and offered by the concrete services. This component is also in charge to observe and compute the distribution of the response time of the composite service for each service class and to estimate its z_α value (defined in Sect. 3). The *WS Monitor* checks periodically the responsiveness of the pool of concrete services and notifies if some of them becomes unavailable. Besides maintaining up to date the parameters of the optimization problem, the QoS Monitor and WS Monitor check and notify whether some relevant change occurs in the composite service environment. This may lead to the solution of a new instance of the optimization problem which accounts for the occurred changes. Events to be tracked include the arrival/departure of a user, an observed variation in the SLA parameters of the concrete services, and the addition/removal of concrete services.

The *Service Manager* and the *SLA Manager* are mainly responsible for the SLA negotiation processes in which the broker is involved as intermediary. The former negotiates the SLAs with the concrete services. The tasks of the latter are the user SLA negotiation and registration, that is, it is in charge to add, modify, and delete SLAs and users profiles. The SLA negotiation process towards the user side includes also the admission control of new users; to this end, it involves the use of the Optimization Engine in order to evaluate the broker capability to accept the incoming user. Most of the broker components access to a storage layer (not shown in Fig. 1) to know the model parameters of the composite service operations and environment.

2.1 Composite Service

We assume that the composite service structure is defined using BPEL [13]. In this paper, we actually refer to a significant subset of the whole BPEL definition,

focusing on its structured style of modeling (rather than on its graph-based one, thus omitting to consider the use of control links). Specifically, in the definition of the workflow describing the composite service, besides the primitive `invoke` activity, which specifies the synchronous or asynchronous invocation of a Web service, we consider most of the structured activities: `sequence`, `switch`, `while`, and `pick`, whose meaning is summarized in Table 1. The percentile-based service selection proposed in this paper is not currently able to manage the `flow` structured activity, which is used for the concurrent execution of activities.

Table 1. Structured activities in BPEL.

Activity	Meaning
`sequence`	Sequential execution of activities
`switch`	Conditional execution of activities
`while`	Repeated execution of activities in a loop
`pick`	Conditional execution of activities based on external event/alarm

Figure 2 shows an example of a BPEL workflow described as a UML2 activity diagram. With the exception of the `pick` construct, this example encompasses all the structured activities listed in Table 1.

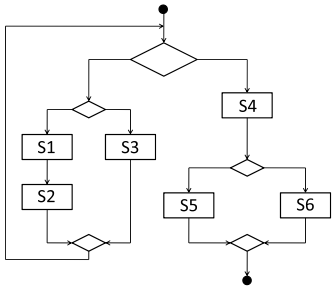


Fig. 2. An example of BPEL workflow.

The business process for the composite service defines a set of abstract services $\{S_1, \ldots, S_n\}$. Each abstract service can be instanciated with a specific concrete service $k_{ij} \in \mathcal{K}_i$, where $\mathcal{K}_i$ is the set of functionally equivalent concrete services that have been identified by the Composition Manager as candidates to implement S_i.

2.2 SLA Negotiation

The Service Manager and SLA Manager components are involved in the SLA negotiation with two counterparts: on one side the requestors of the composite

service, on the other side the providers of the concrete services. Let us first discuss the SLA settled with the latter. The QoS of each concrete service can be characterized according to various attributes of interest, such as response time, cost, reputation, availability, and throughput [6, 16]. The values of these QoS attributes are advertised by the service providers as part of their SLA. Without loss of generality, in this paper we consider the following QoS attributes for each concrete service k_{ij}:

- the response time t_{ij}, which is the interval of time elapsed from the invocation to the completion of the concrete service k_{ij};
- the cost c_{ij}, which represents the price charged for each invocation of the concrete service k_{ij};
- the log of the availability, a_{ij}, *i.e.*, the logarithm of the probability that the concrete service k_{ij} is available when invoked.

In the latter case, as in [16] we consider the *logarithm* of the availability, rather than the availability itself, in order to obtain linear expressions when composing the availability of different services.

For a given concrete service k_{ij}, the SLA established by the broker with the service provider defines the service cost (measured in money per invocation), availability, and expected response time (measured in time unit), provided the volume of requests generated by the broker does not exceed the negotiated average load. Therefore, the SLA for the concrete service k_{ij} is represented by the template $\langle t_{ij}, c_{ij}, a_{ij}, L_{ij} \rangle$, being L_{ij} the agreement on average load.

We denote by K the set of QoS classes offered by the broker. In the SLAs created with the requestors, the broker characterizes the QoS of the composite service in terms of bounds on the expected response time, quantile of the response time, expected cost, and excepted availability for each QoS class $k \in K$ (*i.e.*, $T^k_{max}, T^k_{\alpha,max}, C^k_{max}, A^k_{min}$), where $T^k_{\alpha,max}$ is a bound on the α-quantile T^k_α of the response time. Observe that while the concrete service provides only guarantees on the expected response time t_{ij}, the composite service offered by the broker provides guarantees on the tail of the response time distribution.

Each requestor has to negotiate for each QoS class the volume of requests it will generate in that class (denoted by $\Delta\gamma^k$). The SLA established by the broker with the requestor for the QoS class $k \in K$ has therefore the template $\langle T^k_{max}, T^k_{\alpha,max}, C^k_{max}, A^k_{min}, \Delta\gamma^k \rangle$.

2.3 Admission Control

Upon the arrival of a new user, the SLA Manager determines whether it can be accepted for the required class of service, without violating the SLAs of already accepted requestors. Let γ be the aggregate arrival rate of already accepted requestors (*i.e.*, $\gamma = (\gamma^1, ..., \gamma^{|K|})$ and denote by $\Delta\gamma$ the arrival rate requested by the new user for all the service classes (*i.e.*, $\Delta\gamma = (\Delta\gamma^1, ..., \Delta\gamma^{|K|})$). The SLA Manager determines whether the new requestor can be accepted by invoking the Optimization Engine and asking for a new resolution of the optimization problem

with $\gamma + \Delta\gamma$ as aggregate arrival rate. We have two possible cases. If a feasible solution to the optimization problem exists, it means that the additional requests can be satisfied - at the requested QoS - without violating the QoS of already accepted users. The new requestor can be thus accepted and the SLA finalized for the requested rate and QoS class. If, instead, a feasible solution does not exists, the broker can: 1) turn down the new requestor; 2) renegotiate the SLA with the requestor; 3) renegotiate the parameters of the SLAs with the service providers.

2.4 Service Selection Model

The Adaptation Manager determines, for each QoS class, the concrete service k_{ij} that must be used to fulfill a request for the abstract service S_i. We model this selection by associating with each abstract service S_i a service selection policy vector $\boldsymbol{x}_i = (\boldsymbol{x}_i^1, ..., \boldsymbol{x}_i^{|K|})$, where $\boldsymbol{x}_i^k = [x_{ij}^k]$ and $k_{ij} \in \mathcal{K}_i$. Each entry x_{ij}^k of $\boldsymbol{x}_i^k$ represents the probability that the class-k request will be bound to the concrete service k_{ij}. The x_{ij}^k values are determined by the Optimization Engine every time a new solution of the optimization problem is triggered by some environmental change and are then stored in the storage layer, from where they are accessed by the Adaptation Manager.

With this model, we assume that the Adaptation Manager can probabilistically bind to different concrete services the requests (belonging to a same QoS class k) for an abstract service S_i. As an example, consider the case $\mathcal{K}_i = \{k_{i1}, k_{i2}, k_{i3}\}$ and assume that the selection policy $\boldsymbol{x}_i^k$ for a given class k specifies the following values: $x_{i1}^k = x_{i2}^k = 0.3$, $x_{i3}^k = 0.4$. This strategy implies that 30% of the class-k requests for S_i are bound to k_{i1}, 30% are bound to k_{i2} while the remaining 40% are bound to k_{i3}. From this example we can see that, to get some overall QoS objective for a flow of requests of a given class, the Adaptation Manager may bind different requests to different concrete services.

3 Composite Service QoS Model

In this section, we present the QoS model for the composite service and show how to compute its QoS attributes.

Upon a composite service invocation, the broker executes a sequence of tasks as dictated by the service workflow. Each time a task S_i is invoked, the broker determines at runtime the concrete service k_{ij} to be bound to S_i and invokes it. We denote by N_i^k the number of times the task S_i is invoked by a class-k user service request.

For each class $k \in K$ offered by the broker, the overall QoS attributes, namely,

- the expected response time T^k, which is the time needed to fulfill a class-k request for the composite service;
- the α-quantile T_α^k of the response time;
- the expected execution cost C^k, which is the price to be paid to fulfill a class-k request;

– the expected availability A^k, which is the logarithm of the probability that the composite service is available for a class-k request

depend on: 1) the actual concrete service k_{ij} selected to perform each abstract service S_i, $i = 1, \ldots, n$, and 2) how the services are orchestrated.

Expected Value. To compute these quantities, let $Z_i^k(\boldsymbol{x})$ denote the QoS attribute of the abstract service S_i, $Z \in \{T, C, A\}$. We have

$$Z_i^k(\boldsymbol{x}) = \sum_{k_{ij} \in \mathcal{K}_i} x_{ij}^k z_{ij}^k$$

where z_{ij}^k, $z \in \{t, c, a\}$ is the corresponding QoS attribute offered by the concrete service $k_{ij} \in \mathcal{K}_i$ which can implement S_i.

From these quantities, we can derive closed form expressions for the QoS attributes of the composite service. Since all metrics, namely, the cost, the (logarithm of the) availability, and the response time QoS metrics are additive [6], for their expected value we readily obtain

$$Z^k(\boldsymbol{x}) = \sum_{i=1}^{n} V_i^k Z_i^k(\boldsymbol{x}) = \sum_{i=1}^{n} V_i^k \sum_{k_{ij} \in \mathcal{K}_i} x_{ij}^k z_{ij}$$

where $V_i^k = \mathrm{E}[N_i^k]$ is the expected number of times S_i is invoked for a class-k request.

Response Time α-quantile. It is not possible to find a general expression for a percentile of the response time. We assume to know - or to be able to estimate - the z_α^k-value of the distribution, *i.e.*, the α-quantile of the normalized response time, which is defined as $z_\alpha^k = \frac{T_\alpha^k - \mathrm{E}[T^k]}{\sqrt{\mathrm{Var}[T^k]}}$. Hence,

$$T_\alpha^k = \mathrm{E}[T^k] + z_\alpha^k \sqrt{\mathrm{Var}[T^k]} \tag{1}$$

i.e., we rewrite the percentile of the distribution as function of the expected value $\mathrm{E}[T^k]$, the variance $\mathrm{Var}[T^k]$, and the associated z_α^k-value. The response time variance takes the following form (the derivation of which, for its length, is omitted and can be found in [3]):

$$\mathrm{Var}[T^k] = \sum_{i=1}^{n} V_i^k \mathrm{Var}[T_i^k] + \sum_{i=1}^{n} \sum_{i'=1}^{n} \mathrm{Cov}[N_i^k N_{i'}^k] T_i^k(\boldsymbol{x}) T_{i'}^k(\boldsymbol{x}) \tag{2}$$

where

$$\mathrm{Var}[T_i^k] = \sum_{k_{ij} \in \mathcal{K}_i} x_{ij}(t_{ij}^2 + \sigma_{ij}^2) - \left(\sum_{k_{ij} \in \mathcal{K}_i} x_{ij} t_{ij} \right)^2 \tag{3}$$

is the variance of the response time of task S_i (being σ_{ij}^2 the response time variance of service k_{ij} which we also assume to estimate). Observe that the variance (2) comprises two terms: the first accounts for the variability of the response time of each task weighted by the expected number of times each task is invoked; the second term accounts for the variability of the number of tasks invocations (which are correlated), weighted by the tasks expected response time.

4 Optimization Problem

The Optimization Engine goal is to determine the service selection strategy x_{ij}^k, $i = 1, \ldots, n$, $k \in K$, $k_{ij} \in \mathcal{K}_i$ which maximizes a suitable utility function. For the sake of simplicity, here we consider the simple case that the broker wants to minimize the overall expected cost to offer the composite service, defined as $C(\boldsymbol{x}) = \frac{1}{\sum_{k \in K} \gamma^k} \sum_{k \in K} \gamma^k C^k(\boldsymbol{x})$. In general we could optimize multiple QoS attributes (which can be either mutually independent or possibly conflicting). Therefore, the optimal service selection would take the form of a multi-objective optimization which can be solved by reducing to a single objective problem using a scalarization method, *e.g.*, the Simple Additive Weighting technique.

The Optimization Engine task consists in finding the variables x_{ij}^k, $i = 1, \ldots, n$, $k \in K$, $k_{ij} \in \mathcal{K}_i$, which solve the following optimization problem:

$$\textbf{OPT : } \textbf{min } C(\boldsymbol{x})$$

$$\textbf{subject to: } T^k(\boldsymbol{x}) \leq T^k_{\max} \quad k \in K \tag{4}$$

$$C^k(\boldsymbol{x}) \leq C^k_{\max} \quad k \in K \tag{5}$$

$$A^k(\boldsymbol{x}) \geq A^k_{\min} \quad k \in K \tag{6}$$

$$\mathrm{P}[T^k > T^k_{\alpha,\max}] \leq 1 - \alpha \qquad k \in K \tag{7}$$

$$\sum_{k \in K} x_{ij}^k V_i^k \gamma^k \leq L_{ij} \quad i = 1, \ldots, n, k_{ij} \in \mathcal{K}_i \tag{8}$$

$$x_{ij}^k \geq 0 \quad k_{ij} \in \mathcal{K}_i, \sum_{k_{ij} \in \mathcal{K}_i} x_{ij}^k = 1 \quad i = 1, \ldots, n, k \in K \tag{9}$$

Equations (4)-(6) are the QoS constraints for each service class on average response time, average cost and availability, where $T^k_{\max}$, $C^k_{\max}$, and $A^k_{\min}$ are respectively the maximum response time, the maximum cost and the minimum (logarithm of the) availability that characterize the QoS class k. Equation (7) is the QoS constraint on the percentile of the response time. Equations (8) are the broker-providers SLA constraints and ensure the broker does not exceed the SLA with the service providers. Finally, equations (9) are the functional constraints. The constraints $\mathrm{P}[T^k > T^k_{\alpha,\max}] \leq 1-\alpha$ can be rewritten as $T^k_\alpha \leq T^k_{\alpha,max}$. Hence,

$$T^k_\alpha \leq T^k_{\alpha,\max} \Longleftrightarrow \mathrm{E}[T^k] + z_\alpha \sqrt{\mathrm{Var}[T^k]} \leq T^k_{\alpha,\max}$$

Thus, the constraints on the response time percentile can be rewritten as:

$$z_\alpha \left(\sum_{i=1}^{n} \sum_{i'=1}^{n} \mathrm{Cov}[N_i N_{i'}] \sum_{k_{ij} \in \mathcal{K}_i} x_{ij} t_{ij} \sum_{k_{ij} \in \mathcal{K}_{i'}} x_{i'j} t_{i'j} + \ldots \right.$$

$$\left. \sum_{i=1}^{n} V_i^k \left(\sum_{k_{ij} \in \mathcal{K}_i} x_{ij} (t_{ij}^2 + \sigma_{ij}^2) - \left(\sum_{k_{ij} \in \mathcal{K}_i} x_{ij} t_{ij} \right)^2 \right) \right)^{\frac{1}{2}} \leq$$

$$T^k_{\alpha,\max} - \sum_{i=1}^{n} V_i^k \sum_{k_{ij} \in \mathcal{K}_i} x_{ij}^k t_{ij} \quad k \in K \tag{10}$$

Constraints Linearization. Because of the constraints (10), there is no known technique to solve problem **OPT**[1]. We tackle the problem by deriving a linear program (LP) which is obtained by linearizing (10) in two steps. First of all we eliminate the square root, which is not differentiable in zero by taking the square of both side of (2). Then, we linearize the constraints by approximating both sides with the first term of the Taylor expansion around a suitable point $\boldsymbol{x}_0$, which yields (for space limitation, the computation of the first term of the Taylor expansion is omitted; the details can be found in [3]):

$$\sum_{i=1}^{n} \sum_{k_{ij} \in \mathcal{K}_i} x_{ij}^k \left[z_\alpha^2 \left(2 \sum_{i'=1}^{n} \mathrm{Cov}[N_i N_{i'}] T_{i'}^k(\boldsymbol{x}_0) t_{ij} + \right.\right.$$
$$\left.\left. V_i^k \left(t_{ij}^2 + \sigma_{ij}^2 - 2 t_{ij} T_i^k(\boldsymbol{x}_0) \right) \right) + 2 t_{ij} \left(T_{\alpha\,\max}^k - T^k(\boldsymbol{x}_0) \right) \right] \leq$$
$$z_\alpha^2 \left(\sum_{i=1}^{n} \sum_{i'}^{n} \mathrm{Cov}[N_i N_{i'}] T_{i'}^k(\boldsymbol{x}_0) T_i^k(\boldsymbol{x}_0) - \sum_{i=1}^{n} V_i^k T_i^{k2}(\boldsymbol{x}_0) \right) +$$
$$\left(T_{\alpha,\max}^k - T^k(\boldsymbol{x}_0) \right)^2 + 2 \left(T_{\alpha,\max}^k - T^k(\boldsymbol{x}_0) \right) T^k(\boldsymbol{x}_0) \tag{11}$$

By replacing (7) with (11) in **OPT** we obtain the following LP **LINOPT** which we use to determine the optimal service selection policy:

LINOPT : min $C(\boldsymbol{x})$

subject to:

$$\sum_{i=1}^{n} V_i^k \sum_{k_{ij} \in \mathcal{K}_i} x_{ij}^k t_{ij} \leq T_{\max}^k \quad k \in K \tag{12}$$

$$\sum_{i=1}^{n} V_i^k \sum_{k_{ij} \in \mathcal{K}_i} x_{ij}^k c_{ij} \leq C_{\max}^k \quad k \in K \tag{13}$$

$$\sum_{i=1}^{n} V_i^k \sum_{k_{ij} \in \mathcal{K}_i} x_{ij}^k a_{ij} \geq A_{\min}^k \quad k \in K \tag{14}$$

$$\text{percentile constraints (11)} \qquad k \in K \tag{15}$$

$$\sum_{k \in K} x_{ij}^k V_i^k \gamma^k \leq L_{ij} \quad i = 1, \ldots, n, k_{ij} \in \mathcal{K}_i \tag{16}$$

$$x_{ij}^k \geq 0 \quad k_{ij} \in \mathcal{K}_i, \sum_{k_{ij} \in \mathcal{K}_i} x_{ij}^k = 1 \quad i = 1, \ldots, n, k \in K \tag{17}$$

LINOPT is a LP problem and can be efficiently solved via standard techniques. The solution thus lends itself to both on-line and off-line operations.

The choice of the linearization point $\boldsymbol{x}_0$ is crucial to obtain good solutions, *i.e.*, solutions close to those that would have been obtained by solving **OPT**. We found that a good choice for $\boldsymbol{x}_0$ is provided by a most recent solution $\boldsymbol{x}$ itself. In case such a solution is not available, *e.g.*, when the broker is initialized, we simply do not consider the constraints (15) the first time **LINOPT** is executed.

[1] Had the constraints be convex, we could have used semidefinite programming to solve the problem. Since convexity does not hold in general, we have resorted to linearization instead.

5 Simulation Model and Experiments

In this section, we first describe the simulation model we have defined to study the effectiveness of the percentile-based adaptation policy and then present the results of simulation experiments.

5.1 Simulation Model

The broker simulation model comprises the same components of the architecture shown in Fig. 1. We consider an open system model, where new users belonging to a given service class $k \in K$ offered by the broker arrive at mean *user inter-arrival rate* Λ_k. Each class-k user is characterized by its SLA parameters defined in Sect. 2.2 and by the *contract duration* d_k. If admitted (according to the admission control mechanism explained in Sect. 2.3), the user will start generating requests to the composite service until its contract expires.

Differently from traditional Web workload, SOA workload characterization has been not deeply investigated up to now (some preliminary results are in [12]). Therefore, in our workload model we assume exponential distributions with parameters Λ_k and $1/d_k$ for the user inter-arrival time and contract duration, respectively, and a Gaussian distribution with parameters m_k and σ_k for the inter-arrival rate of requests to the composite service generated by each user. We also assume that the response times of the concrete services follow Erlang distributions with different shape parameters.

The discrete-event simulator has been implemented in C language using the CSIM package [11]. Multiple independent random number streams have been used for each stochastic model component. The experiments involved a minimum of 10,000 completed requests to the composite service; for all reported mean and percentile values the 95% confidence interval has been obtained using the run length control provided by CSIM.

5.2 Experimental Results

We illustrate the dynamic behavior of our adaptive service selection through the simple abstract workflow of Fig. 2. For the sake of simplicity we assume that two candidate concrete services (with their respective SLAs) have been identified for each abstract service, except for S_2 for which four concrete services have been identified. The respective SLAs differ in terms of cost, availability, and response time (all time values are measured in sec.). Table 2 summarizes for each concrete service k_{ij} the SLA parameters $\langle t_{ij}, c_{ij}, a_{ij} \rangle$ and the shape parameter Erl_{ij} of the Erlang distribution for the response time (for each k_{ij}, the mean value of the Erlang distribution corresponds to t_{ij}). The SLA and the Erlang shape parameters have been chosen so that for abstract service S_i, concrete service k_{i1} represents the best implementation, which at a higher cost guarantees higher availability and lower response time (in terms of mean as well as variance) with respect to concrete service k_{ij} for $j \geq 2$, which costs less but has lower availability and higher response time. For all concrete services, $L_{ij} = 10$.

Table 2. Concrete service SLA parameters and shape parameter of Erlang distribution.

Service	c_{ij}	a_{ij}	t_{ij}	Erl_{ij}
k_{11}	6	0.995	2	4
k_{12}	3	0.99	4	2
k_{21}	4.5	0.99	1	4
k_{22}	4	0.99	2	4
k_{23}	2	0.95	4	2
k_{24}	1	0.95	5	2
k_{31}	2	0.995	1	4
k_{32}	1.8	0.95	2	2

Service	c_{ij}	a_{ij}	t_{ij}	Erl_{ij}
k_{41}	1	0.995	0.5	4
k_{42}	0.8	0.99	1	2
k_{51}	2	0.99	2	4
k_{52}	1.4	0.95	4	2
k_{61}	0.5	0.99	1.8	4
k_{62}	0.4	0.9	4	2

On the user side, we assume a scenario where the broker offers the composite service with four QoS classes. The SLAs negotiated by the users are characterized by a wide range of QoS requirements as listed in Table 3, with users in class 1 having the most stringent requirements $A^1_{min} = 0.95$ and $T^1_{max} = 7$, and users in class 4 the least stringent requirements $A^4_{min} = 0.8$ and $T^4_{max} = 18$. With regard to the bound on the α-quantile $T^k_{\alpha,\max}$ of the response time, we assume that for all classes $T^k_{\alpha,\max} = \beta T^k_{\max}$ and $\alpha = 0.95$ (i.e., we consider 95-percentile of the response time). The SLA costs parameters for the four classes have been set accordingly, where class 1 has the highest cost per request and class 4 is the cheapest. The expected number of service invocations for the different classes is:

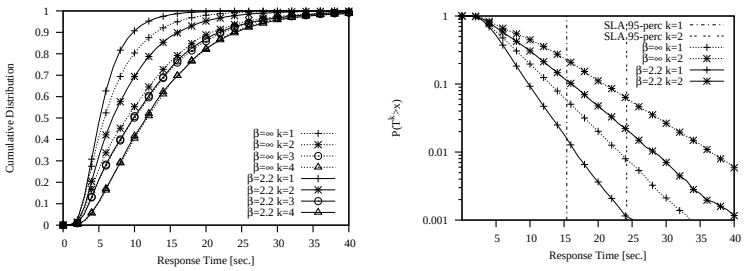
Table 3. User-side SLA parameters for each service class.

Class k	C^k	$A^k_{\min}$	$T^k_{\max}$	$T^k_{0.95,\max}$	γ^k
1	25	0.95	7	$\beta T^1_{\max}$	10
2	18	0.9	11	$\beta T^2_{\max}$	4
3	15	0.9	15	$\beta T^3_{\max}$	2
4	12	0.8	18	$\beta T^4_{\max}$	1

$V^k_1 = V^k_2 = V^k_3 = 1.5$, $V^k_4 = 1$, $k \in K$; $V^k_5 = 0.7$, $V^k_6 = 0.3$, $k \in \{1, 3, 4\}$; $V^2_5 = V^2_6 = 0.5$, that is, all classes have the same usage profile except users in class 2, who invoke S_5 and S_6 with different intensity. The values of the parameters that characterize the workload model are $d_k = 100$ and $(m_k, \sigma_k) = (3, 1)$ for each k (Λ_k, d_k, and m_k values have to be set so that $\gamma^k = \Lambda_k m_k d_k$ for Little's formula).

We compare the performance obtained by the service selection with tight bounds on the percentile of the response time with that of the service selection where only guarantees on the mean values are offered to the users of the composite service. The problem formulation of the latter case is in [4] and we denote it with $\beta = \infty$ (i.e., the tail of the response time distribution is unbounded). We initially set $\beta = 2.2$, which represents a tight bound on the 95-percentile of the response time; we then analyze the sensitivity of the response time to β.

Figure 3(a) shows the cumulative distribution of the response time of the composite service for all the service classes when the percentile-based and mean-based optimizations are used, corresponding to $\beta = 2.2$ and $\beta = \infty$ curves, respectively. We can see that the percentile-based optimization achieves a better



(a) Cumulative distribution.

(b) Complementary cumulative distribution (lin-log scale).

Fig. 3. Response time.

response time than the mean-based optimization for classes 1 and 2, which have the more stringent SLA requirements. Through Fig. 3(b) we further investigate the tendency of the response time for classes 1 and 2 by plotting its complementary cumulative distribution on a linear-logarithmic scale. The vertical lines represent the 95-percentile of the response time agreed in the SLA with the users of the composite services. We can see that the percentile-based service selection largely satisfies the 95-percentile SLA, that is only 1.6% and 2.1% of class 1 and class 2 requests respectively experience a response time greater than the 95-percentile value. The conservative behavior of the percentile-based approach is due to the constraints linearization explained in Sect. 4.

To compare in more detail the percentile-based and mean-based approaches, Fig. 4 shows how the mean and 95-percentile response times of the composite service vary over time for classes 1 and 2. The horizontal line is the agreed response time (both mean and 95-percentile values), as reported in Table 3. We observe that the mean-based approach leads to some violations of the agreed response time, while the percentile-based approach allows the broker to offer always a response time much better than that agreed.

We conducted a last set of experiments to analyze the sensitivity of the percentile-based service selection to the β parameter, which correlates in the SLA the 95-percentile to the mean of the composite service's response time (smaller values of β correspond to a tighter bound on the distribution tail). Figure 5 shows the trend of the 95-percentile response time to β for all classes (the corresponding SLA value is only shown for the most demanding classes 1 and 2). The percentile-based approach succeeds in respecting the agreed 95-percentile for all service classes and β values. The disadvantage of a tighter bound

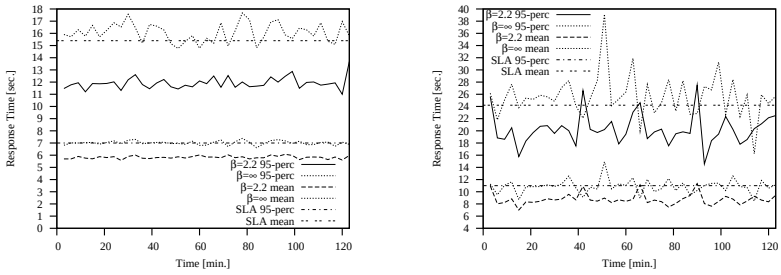


Fig. 4. 95-percentile and mean response time over time for classes 1 (left) and 2 (right).

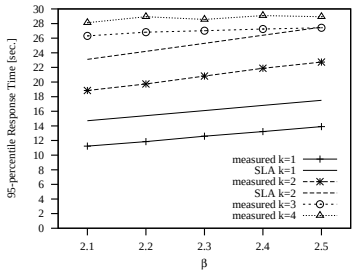


Fig. 5. Sensitivity of 95-percentile response time to β.

on the percentile (e.g., $\beta = 2.1$) is that a larger fraction of incoming contract requests are rejected.

6 Conclusions

In this paper, we have addressed the problem of selecting concrete services in a composite service offered by a brokering service which supports differentiated QoS service classes. Most of the existing approaches only consider SLAs based on bounds of the expected values of the relevant QoS metrics. A limitation of these solutions lies in the fact that the user perceived quality is often better expressed in terms of bounds on the percentile rather than the expected value of the QoS metrics. To overcome this limitation, in this paper we have considered SLAs which also specify bounds on the percentile of the response time. We have formulated the service selection problem as an optimization problem with non-linear constraints. For the solution, we have linearized the constrains. The resulting linear programming problem can be efficiently solved via standard techniques. Therefore, our approach can be used to efficiently manage the service selection in a real operating broker-based architecture, where the broker efficiency and scalability in replying to the users requests are important factors.

The model proposed in this paper provides statistical guarantees on the percentile of the response time. The results, though, only apply to the service se-

lection scenario and only consider a subset, albeit significant, of the workflows' structured activities. Our future work includes the extension of these results to the use of redundant coordination patterns, the support of the `flow` BPEL activity, and the management of long term SLAs whose users cannot get their request rejected.

Acknowledgment

Work partially supported by the Italian PRIN project D-ASAP.

References

1. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Trans. Softw. Eng. 33(6), 369–384 (June 2007)
2. Canfora, G., Di Penta, M., Esposito, R., Villani, M.: A framework for qos-aware binding and re-binding of composite web services. J. Syst. Softw. 81(10) (2008)
3. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F.: Adaptive service selection in service oriented systems under percentile-based service level agreements. Tech. Rep. RR-10.85, DISP, Univ. of Roma Tor Vergata (2010), `http://www.ce.uniroma2.it/publications/RR-10.85.pdf`
4. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F., Mirandola, R.: Flow-based service selection for web service composition supporting multiple qos classes. In: Proc. IEEE ICWS '07. pp. 743–750 (2007)
5. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F., Mirandola, R.: Qos-driven runtime adaptation of service oriented architectures. In: ACM ESEC/SIGSOFT FSE. pp. 131–140 (2009)
6. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.J.: Modeling Quality of Service for Workflows and Web Service Processes. Web Semantics J. 1(3) (2004)
7. DeCandia, G. et al.: Dynamo: Amazon's highly available key-value store. SIGOPS Oper. Syst. Rev. 41(6), 205–220 (2007)
8. Gmach, D., Krompass, S., Scholz, A., Wimmer, M., Kemper, A.: Adaptive quality of service management for enterprise services. ACM Trans. Web 2(1), 1–46 (2008)
9. Grosspietsch, K.: Optimizing the reliability of component-based n-version approches. In: Proc. IEEE IPDPS 2002 Workshops (2002)
10. Guo, H., Huai, J., Li, H., Deng, T., Li, Y., Du, Z.: Angel: Optimal configuration for high available service composition. In: Proc. IEEE ICWS '07. pp. 280–287 (2007)
11. Mesquite Software: `http://www.mesquite.com/`
12. Nagpurkar, P., Horn, W., Gopalakrishnan, U., Dubey, N., Jann, J., Pattnaik, P.: Workload characterization of selected jee-based web 2.0 applications. In: Proc. IEEE Int'l Symposium on Workload Characterization. pp. 109–118 (Sep 2008)
13. OASIS: Web Services Business Process Execution Language Version 2.0 (Jan 2007), `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
14. Xiong, K., Perros, H.: Sla-based service composition in enterprise computing. In: IEEE Int'l Workshop on Quality of Service. pp. 35–44 (2008)
15. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans. Web 1(1), 1–26 (2007)
16. Zeng, L., Benatallah, B., Dumas, M., Kalagnamam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Trans. Softw. Eng. 30(5) (2004)