Utility-based QoS Brokering in Service Oriented Architectures

Daniel A. Menascé
Department of Computer Science, MS 4A5
George Mason University
Fairfax, VA 22030, USA
menasce@cs.gmu.edu

Vinod Dubey
The Volgenau School of IT & Engineering
George Mason University
Fairfax, VA 22030, USA
vdubey@gmu.edu

Abstract

Quality of Service (QoS) is an important consideration in the dynamic service selection in the context of Service Oriented Architectures. This paper extends previous work on QoS brokering for SOAs by designing, implementing, and experimentally evaluating a service selection QoS broker that maximizes a utility function for service consumers. Utility functions allow stakeholders to ascribe a value to the usefulness of a system as a function of several attributes such as response time, throughput, and availability. This work assumes that consumers of services provide to a QoS broker their utility functions and their cost constraints on the requested services. Service providers register with the broker by providing service demands for each of the resources used by the services provided and cost functions for each of the services. Consumers request services from the QoS broker, which selects a service provider that maximizes the consumer's utility function subject to its cost constraint. The QoS broker uses analytic queuing models to predict the QoS values of the various services that could be selected under varying workload conditions. The broker and services were implemented using a J2EE/Weblogic platform and experiments were conducted to evaluate the broker's efficacy. Results showed that the broker adequately adapts its selection of service providers according to cost constraints.

1. Introduction

Service Oriented Architectures (SOA) enable a multitude of service providers (SP) to provide loosely coupled and interoperable services at different Quality of Service (QoS) and cost levels in a number of service domains. This provides a unique opportunity for businesses to dynamically select services that better meet their business and QoS needs in a cost-effective manner. However, the real value of dynamic service selection can only be realized if the service provider selected on the fly meets client's QoS requirements/SLAs under varying workload conditions. Also, the selected service providers need to optimize a global utility for their clients under cost constraints. Clients may want to negotiate for certain QoS commitments from the participating service providers and the service providers need to ensure that they keep the client's commitments at runtime and not be affected by operational factors such as workload variations and server failures.

Menascé et. al. [9] devised a framework for QoS management in SOAs where a QoS Broker(QB) provides QoS negotiation and resource reservation mechanisms on behalf of service providers and ensures that the QoS commitments made on their behalf are honored during runtime. The framework allows the clients to send QoS requests and negotiate with the QB for a QoS commitment on behalf of a service provider. At the end of a successful QoS negotiation, the client receives from the broker a digitally signed token representing a QoS commitment on behalf of a service provider. The client uses that token to send requests directly to the service provider. The service provider provides services while keeping the commitments made by the broker on its behalf by performing admission control. Decisions on QoS commitments are made by the broker with the help of an analytical performance model.

The work in [9] assumed that the client specifies the service provider it wants to use. In this paper, we discuss an extension to the QoS framework of [9] in which the QoS broker uses utility functions and cost functions to perform service provider selection. With the help of predictive analytical queuing network performance models, the QoS broker identifies the service provider that optimizes a global utility for a client under a cost constraint. We have also addressed the performance and availability issues of the QoS Broker and extended it to provide a flexible and loosely coupled integration scheme. Additionally, we have provided a demonstration of the ideas presented in this paper by developing components and services using J2EE platforms and by conducting experiments. The results presented in the paper show that the QoS broker adapts to the cost constraints

and selects service providers based on cost and utility.

The rest of the paper is organized as follows: Section 2 discusses the extended QoS Broker Architecture. Section 3 discusses the QoS Broker Interaction Protocol. The next section highligts the performance model used. Section 5 specifies the utility functions used and shows how to evaluate global utility of a service provider for a client. Section 6 describes the service selection algorithm and section 7 describes the results of the experiments carried out to validate the ideas presented in this paper. Section 8 discusses related work and section 9 presents some concluding remarks.

2 Extended QoS Broker Architecture

The QoS Broker architecture described in this paper extends the architecture discussed in [9] in which QoS brokering was used for resource reservation and to guarantee QoS commitments. In the present paper, the QoS Broker (QB) selects a service provider, from among participating service providers, that maximizes a global utility for a client under a cost constraint. Additionally, our architecture supports a flexible and loosely coupled integration scheme and serves as a mediator to route requests and responses between service providers and consumers. As in [9], the broker ensures that none of the previous commitments made on behalf of the SPs are violated.

The extended QB architecture (see Fig. 1) consists of Registration Services, QoS Negotiation Services, Business Services, QoS Request Evaluator, Performance Model Solver, Utility Function Evaluator, repositories to store SLA/QoS commitments and service providers' metadata including a service demand matrix (see below), and a QB Service Bus.

The Registration Service allows SPs to submit service descriptions (e.g., WSDL files), a service demand matrix, and cost functions to the QB. The *service demand* matrix indicates the amount, in time units, required of each resource—hardware and software resources—to process requests for each of the services provided by a SP [11]. The QB uses this information, along with an analytical peformance model and the client's utility function, to evaluate a QoS request and identify an optimal SP. The cost vs. response time function is assumed to be monotonically decreasing.

The QoS Negotiation Service is used to evaluate a client's request, negotiate for a QoS commitment, and identify a service provider that maximizes the client's global utility (described in Section 5). A request contains the service type, its concurrency level, a utility function, and a cost constraint. The QoS Negotiation Service delegates the request to the QoS Request Evaluator that uses the Performance Model Solver and the Utility Function Evaluator to evaluate the service request against each service provider.

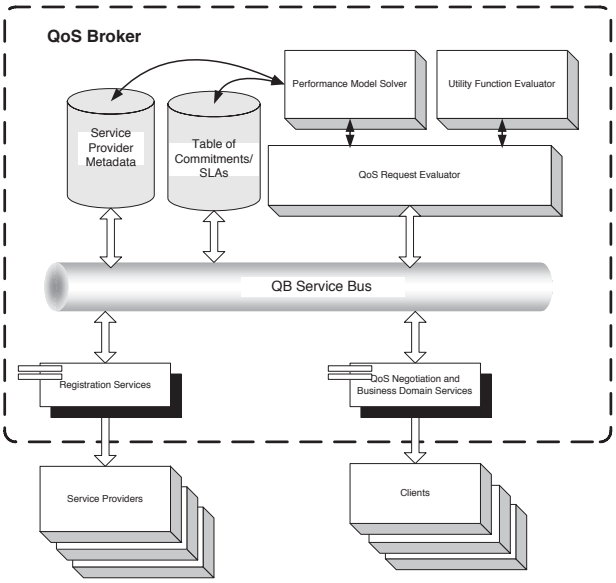

Figure 1. Extended QoS Broker Architecture

The Performance Model Solver implements a multi-class closed-QN model, as discussed in more detail in [11, 12], to predict QoS metrics such as response time and throughput of participating service providers taking into account their current workload intensities and commitments. Each class in the model corresponds to an already committed session plus one for the new service request being considered. The Performance Model Solver computes response time (R), throughput (X), and probability of rejection (P_{rej}) for each committed session as well as for the new request. Using the computed values of R and X for each class corresponding to the committed sessions and new request, the QB via the QoS Request Evaluator ensures that the previous commitments made on behalf of SPs are not violated. Using the output of the model (i.e., computed values of R, X, and P_{rej}), the QB evaluates the global utility function with the help of the Utility Function Evaluator. The values of the global utility function are computed and compared against each service provider and the service provider that maximizes the global utility for the client under a cost constraint is chosen to provide the requested service to the client.

The QB Service Bus serves as a gateway to the QoS Broker and provides a mediation layer. It also supports a loosely coupled integration scheme and avoids the need of a tightly coupled point-to-point integration between clients and servicer providers. The next section discusses the QoS interaction protocol used by the QB.

Another extension of the QoS Broker architecture is the concept of Super QoS Broker (SQB) that provides brokerage services for service providers to locate a QB that can

offer the best QoS brokering services for a cost. The SQB provides two major services: Registration Service and Discovery Service. The Registration Service is used by the QBs to register themselves with the SQB by providing their cost model and QoS level. The Discovery Service is used by SPs to locate a QB that offers the best QoS level for a given cost. There could be several QBs with different QoS levels and cost models. A SQB can support one or more QBs. A QB can manage one or more service providers providing one or more services that can be consumed by one or more service consumers.

3 QoS Broker Interaction Protocol

Figure 2 shows a sequence diagram depicting the interaction between the SQB, the QB, the SPs, and the clients.

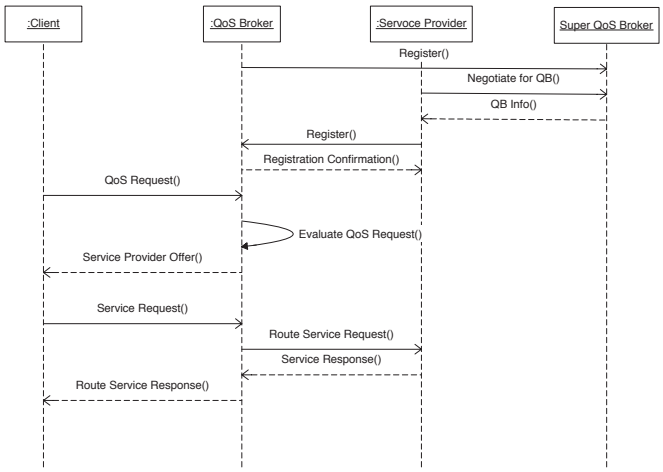

Figure 2. QoS Broker Interaction Diagram

1. QBs register their brokerage services, cost models, and QoS levels with the SQB.
2. SPs negotiate with the SQB for a QB that meets their QoS requirements and cost constraints.
3. SPs register their services, service descriptions, service demand matrix, and cost models with a QB
4. A service consumer (client) starts a session by sending to the QB a QoS request indicating the requested service type, the client's utility function, its cost constraint, and the concurrency level (i.e., maximum number of simultaneous requests to be sent to the service provider for that type of service).
5. The QB identifies a service provider that maximizes the global utility for the client under the given cost and sends to the client, on behalf of the selected SP, a QoS offer composed of a digitally signed token. The token consists of a) the id of the selected service provider, b) a QoS commitment id which indicates the position in a Table of Commitments (ToC) maintained by the QB of the newly accepted request, c) a session id, d) the id of the service within the service provider, e) the QoS offer, which consists of the offered response time, throughput, and concurrency level, and f) an expiration date and time for the offer. When evaluating a request, the QB takes into account the current commitments and workload conditions on the service providers.
6. If the client accepts the QoS offer, the QB saves it in the ToC. The client is now able to submit requests within the session. The requests contain the token provided to the client by the QB. Since the token contains all information required to identify the service and QoS commitment, it is easy to design scalable stateless services.
7. Contrary to the approach described in [9], the QB routes requests and responses between the service providers and the clients and performs any required data transformation.

4 Performance Model

As in [9], the QB uses a predictive analytic performance model to predict the response time and throughput of service providers for the current workload conditions. The response time of a service request depends on the amount of contention for software resources (e.g., software threads) and for hardware resources (e.g., processors and storage devices). We use a performance model called SQN-HQN (for Software Queueing Network - Hardware Queuing Network) described in detail by Menascé in [12]. This model is an iterative use of two queuing network models, one for the software queuing network and another for the hardware queuing network. Details about analytic queuing networks are outside the scope of this paper. The reader is referred to [11].

5 Utility Functions

Utility functions have been used for achieving self-optimization in distributed autonomic systems [7, 14]. Bennani and Menascé [4] used utility functions along with analytical queuing network models to dynamically allocate servers amongst application environments (AE) to optimize the global utility of AEs. Utility is a dimensionless quantity

and measures the usefulness of a system to a stakeholder in terms of a set of attributes. Bennani and Menascé (2005) [4] used response time and throughput as attributes. The utility functions used in this paper for response time and throughput are similar to those in [4]. We introduced a utility function for the probability that requests are rejected. Utility functions different from the ones adopted here could be used provided that they are monotonically decreasing for response time and probability of rejection and monotonically increasing for throughput. The monotonicity assumption is not a severe restriction since it corresponds to rational user expectations. For example, one would expect a user to see less utility in a system as its response time increases than the other way around.

The utility function, $U_R(r)$, for response time is :

$$U_R(r) = \frac{K_R \times e^{-r+\beta_R}}{1 + e^{-r+\beta_R}} \quad (1)$$

where r is the response time, β_R is the response time SLA, and K_R is a scaling factor computed so that $U_R(0) = 100$. Thus, $K_R = 100\ (1 + e^{\beta_R})/e^{\beta_R}$. The response time r is a function of the workload intensity and the capacity of the servers used by a SP. An example of $U_R(r)$ is shown in Fig. 3 for $\beta_R = 4$ sec, 2 sec, and 1 sec, respectively.

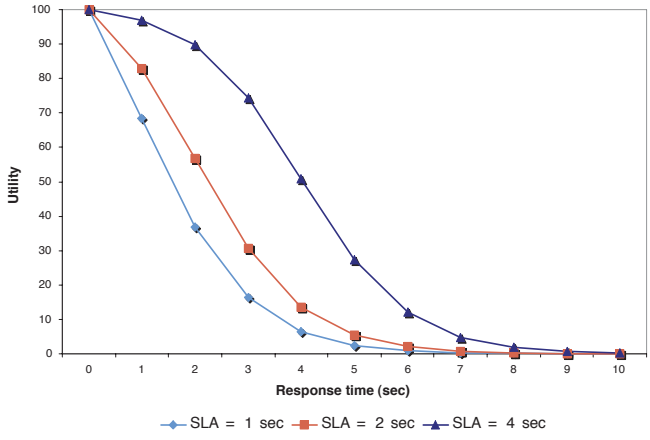

Figure 3. Utility function vs. response time

The utility function, $U_X(x)$, for the throughput is:

$$U_X(x) = K_X \times \left(\frac{1}{1 + e^{-x+\beta_X}} - \frac{1}{1 + e^{\beta_X}} \right) \quad (2)$$

where x is the throughput, β_X, is the throughput SLA, and K_X is a scaling factor computed so that $\lim_{x\to\infty} U_X(x) = 100$. Thus, $K_X = 100\ (1 + e^{\beta_X})/e^{\beta_X}$. The throughput x is a function of the workload intensity and the capacity of the servers used by the SP. Figure 4 shows an example of $U_X(x)$ for $\beta_X = 1$ tps, 2 tps, and 4 tps, respectively.

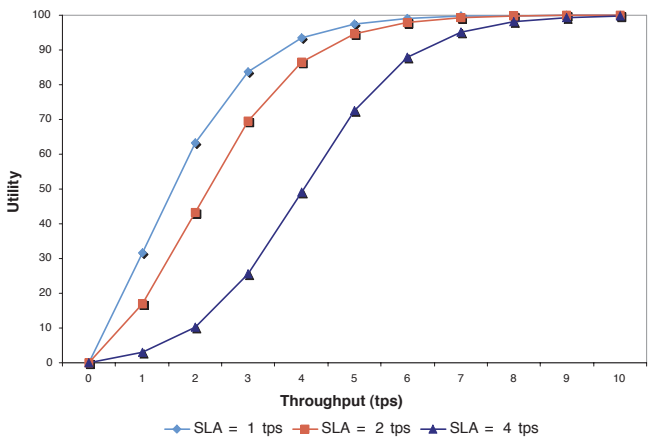

Figure 4. Utility function vs. throughput

We also consider the utility function $U_P(p)$ for the probability, p, that requests are rejected by an SP. This utility function has its maximum value of 100 when $p = 0$ and decreases with p:

$$U_P(p) = \frac{1 - p}{\frac{1}{100} + \beta_P \times p}. \quad (3)$$

In Eq. (3), β_P is the SLA for p. Figure 5 shows an example of $U_P(p)$ for $\beta_P = 0.05, 0.1$, and 0.2, respectively.

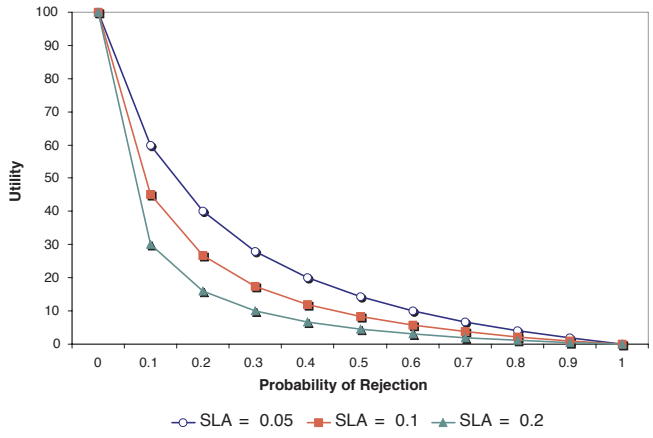

Figure 5. Utility vs. the probability of rejection

A global utility U_g is a function of the individual utility functions $U_R(r)$, $U_X(x)$, and $U_P(p)$:

$$U_g = f(U_R(r), U_X(x), U_P(p)). \quad (4)$$

A possible expression for U_g is:

$$U_g = w_R \times U_R(r) + w_X \times U_X(x) + w_P \times U_P(p) \quad (5)$$

where w_R, w_X, and w_P represent importance weights assigned to each of the three metrics. Each weight is a number in the range (0,1) and the sum of all weights is equal to one.

6 Service Provider Selection

In order for the QB to identify a service provider that maximizes the global utility U_g for a client, SPs register their services with the QB and provide their (invertible) cost functions, and service demand matrices. We first describe the service provider selection approach for a single metric: response time. We assume that the response time cost functions are monotonically decreasing functions of the response time. Let $C_i(r), i = 1, \ldots, N$ be the cost function for a given service i. A client sends a QoS request to the QB and provides its utility functions $U_R(r)$ along with a cost constraint $C_{\max}$.

Given the client's cost constraint $C_{\max}$ and SPs' cost functions $C_i(r)$, the QB estimates the best response times that the client might get from different SPs under $C_{\max}$. These response time estimates are the intersections of the SP's cost functions and the client's cost constraint (see Fig. 6). Let the response time for which $C_i(r)$ and $C_{\max}$ intersect be denoted by $R_i^{\min}$ and called minimum cost-constrained response time (MCRT). Consider the example illustrated by Fig. 6 with three cost functions $C_1(r), C_2(r)$, and $C_3(r)$. The MCRTs are denoted by R_1, R_2, and R_3, respectively. Let $R_1 < R_2 < R_3$ without loss of generality.

The algorithm to select a SP that maximizes the client's utility under C_{max} is given below.

Step 1 The QB, using analytical performance models, obtains response time estimates R_i^{est} from each service provider SP_i taking into account the SPs' current workload intensities.

Step 2 For each i such that $R_i^{\text{est}} < R_i^{\min}$, do $R_i^{\text{est}} = R_i^{\min}$. This is done in order to avoid violating the cost constraint since $R_i^{\min}$ is the minimum possible response time that does not violate the cost constraint.

Step 3 Select the SP j such that $R_j^{\text{est}} = \min_i\{R_i^{\text{est}}\}$ and save R_j^{est} in the Table of Commitments (ToC). This step assumes that the utility function $U_R(r)$ is monotonically decreasing so the smallest response time provides the largest possible value for the utility.

We now describe a service provider selection algorithm for two QoS metrics: response time and throughput. Let $Cost = f(r, x)$ be a service provider's cost function of the response time r and throughput x. We use the following general procedure to select an SP:

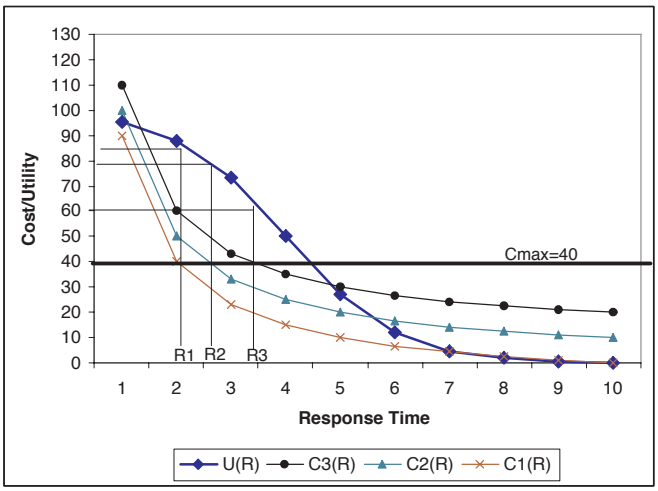

Figure 6. Utility and Cost Functions of Response Time

Step 1 Obtain $(R_i^{\text{est}}, X_i^{\text{est}})$ for each SP_i using the analytical performance model.

Step 2 Compute $Cost_i = f(R_i^{\text{est}}, X_i^{\text{est}})$ for each SP_i

Step 3 Eliminate SP$_i$ such that $Cost_i > C_{max}$

Step 4 For each remaining SP j compute the global utility function as

$$U_g^j = f(U_R(R_j^{\text{est}}), U_X(X_j^{\text{est}})) \quad (6)$$

Step 5 Select the SP such that U_g^j is maximum.

Extending the above algorithm to more QoS metrics is straightforward.

7 Experimental Evaluation

To validate the ideas presented in this paper, we designed and developed a working prototype of the QoS Broker to solve a QoS problem in the travel-agency domain. The QB provides QoS brokerage services for service consumers on behalf of service providers. In our example, an online travel agent uses airline reservation service providers to provide travel services to customers. There are three service providers—SP1, SP2, and SP3—providing the same type of airline reservation Web service at different QoS and cost levels. The Web service of each SP supports two operations: 1) checkFlightAvailability and 2) makeReservation. The experiment was conducted using the Weblogic

8.1 Web/Application server running on a Windows XP platform with 2 GB RAM and a 1.5GHz Intel Pentium processor. The Web services were deployed on a Weblogic Web/Application server that hosts Web services as Web applications and uses the Servlet multi-threaded model with five threads per Web service.

The service providers register their services and respective service demands and the cost functions with the QB. Table 1 shows the service demands, D_{CPU} and $D_{\mathrm{I/O}}$ for the CPU and I/O, respectively, for each SP. As can be seen from the table, SP1 is the fastest and most expensive SP and SP3 is the slowest and cheapest. The table also shows the cost function as a function of the response time r for each service provider. When making a QoS request, the client sends its utility function (see below) representing an SLA of 3 sec and cost constraint to the QB.

$$U_R(r) = \frac{105 \times e^{-r+3}}{1 + e^{-r+3}}. \quad (7)$$

Upon receiving a QoS request for a service from a client, the QB identifies the service provider that maximizes the global utility for the client under the given cost as described in section 6.

Table 1. Service Demands (in sec) and Cost Functions

	SP1	SP2	SP3
D_{CPU}	0.0679	0.1763	0.3044
$D_{\mathrm{I/O}}$	0.0174	0.0252	0.0338
$\mathrm{Cost}(r)$	$20/r + 4$	$20/r + 2$	$20/r$

The workload used for the experiment is composed of random requests for checking flight availability or booking a flight based on a relational database of 2,400 city-pair and date-time options.

In this experiment, we assume that the clients have access to the WSDL of the services provided by the QB. Alternatively, the QB could publish its services with a UDDI compliant service registry. Service providers providing airline reservation services may search the service registry and discover brokerage service interfaces offered by the QB. Additionally, the client (the travel agency) could also search the service registry and access the business service proxies that the client uses to bind and invoke requests.

The experimental configuration consists of 20 consecutive sessions running on a client machine separate from that of the SPs. Each session runs 30 concurrent client threads and each thread executes 9 requests in sequence. These requests could either be for checking flight availability or for booking flights. Thus, 5,400 requests are submitted by all sessions during the experiment. Average response times (R_s) are computed for each session and subsequently utilities are computed using $U_R(R_s)$.

Figure 7 shows the average response time per request for each of the 20 sessions for different values of the cost constraint $C_{\max}$: 10, 30, 50, 100, and 150. The figure also shows 95% confidence intervals for the averages. The average values of the response times over all sessions for $C_{\max} = 10, 30, 50, 100$ and 150 are 7.98 sec, 5.96 sec, 4.44 sec, 2.42 sec, and 2.10 sec, respectively. It can be seen that as the cost constraint decreases, the response time increases because the QB has less flexibility of submitting requests to faster SPs because the client cannot afford it. For higher values of the cost constraint, the selection is essentially driven by the SP that has a better predicted response time.

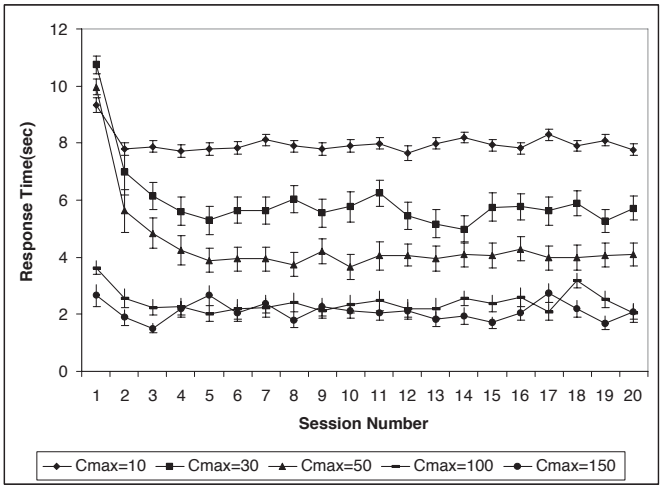

Figure 7. Average Response Time (in sec) for 20 sessions and for various values of $C_{\max}$.

Figure 8 shows the resulting utility seen by the clients based on the response time values shown in the curves of Fig. 7 for each of the 20 sessions and for various values of $C_{\max}$. The curves show, as expected, that higher values of the cost constraint increase the utility because the QB has more flexibility to select among the three SPs. When the cost is very constrained, i.e., $C_{\max} = 10$, the utility is very low.

Figure 9 shows the number of requests sent to each SP for the same different values of $C_{\max}$. Note that SP1 is the fastest and most expensive and SP3 the slowest and cheapest. The figure shows that as the cost constraint changes, the distribution of requests sent to each SP will change accordingly. For a very tight cost constraint, i.e., $C_{\max} = 10$, all requests go to SP3, the cheapest. As the cost constraint is relaxed, more requests move to SP2 and then to SP1.

The results presented in this section show that the QB is

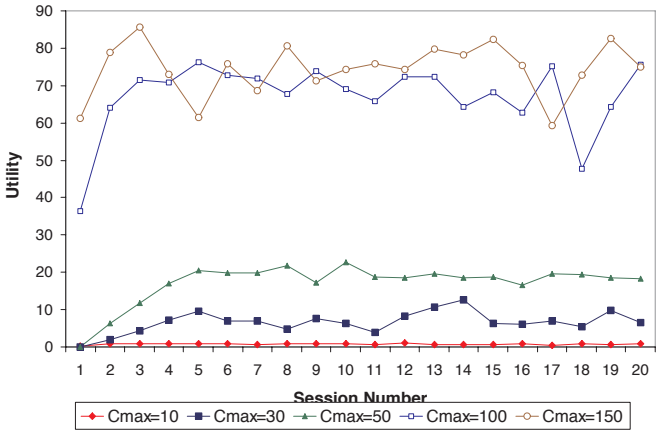

Figure 8. Utility for 20 sessions and for various values of $C_{\max}$

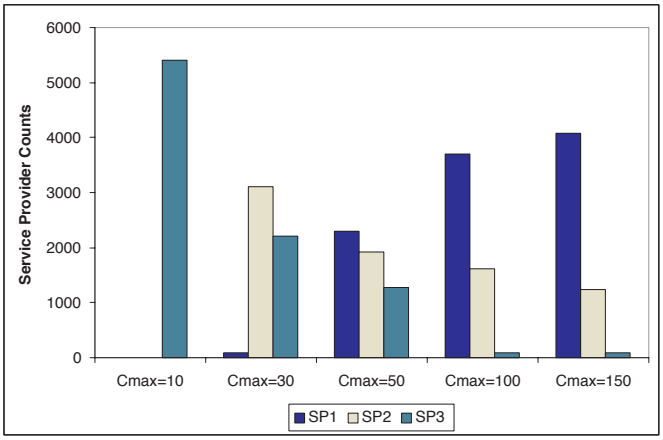

Figure 9. Number of requests sent to each SP for different values of $C_{\max}$.

able to adapt to the cost constraints when making SP selections.

8 Related Work

Besides the work by one of the authors [9], there are other research activities related to the work reported here. Serhani et. al. [13] propose broker-based verification and certification of Web services for their functional and QoS claims and thus support clients to find Web services based on their QoS requirements. The authors, however, do not demonstrate what mechanisms are in place to guarantee that the QoS claims will be met at runtime under varying load conditions and do not elaborate on how the broker is involved in QoS negotiation, monitoring, and adaptation.

Fung et. al. [6] propose a SOAP message tracking model for supporting end-to-end management of QoS in the context of WSBPEL and SLA and aims to motivate the study of QoS management in the Web Services composition framework. The idea is that the client should be able to specify QoS requirements in the SOAP message header and the service provider should enforce the client's QoS requirement. The authors do not demonstrate how SLAs are negotiated and whether QoS requirements specified by a client are really met.

Jaeger et. al. [8] discuss algorithms for the selection of the most suitable candidate services to optimize the overall QoS of a composition. To select Web services for the individual tasks in the composition, the authors calculate the QoS of a composition by aggregating the QoS of individual services based on a previous work on QoS aggregation for Web services composition using workflow patterns. In that paper, the authors identify composition patterns that represent basic structural elements of a composition like a sequence, loop, or parallel executions. They demonstrate that the pattern-wise selection algorithms that leverages the composition patterns determines the best assignment based on best possible QoS, considering each composition pattern in isolation. However, the optimal composition achieved through the composition patterns does not meet the end-to-end QoS constraint for the process.

Canfora et. al. [5] discuss genetic algorithms as an approach for solving an optimization problem for Web service composition. The approach aims to determine a set of concrete services to be bound to the abstract services composing the workflow of a composite service in such a way that the selected set meets QoS constraints established in the SLA and optimize a function of some other QoS parameters. That paper presents an interesting approach to the service composition optimization problem. The authors do not show whether the QoS guarantees of the optimized solution could be met at runtime under varying workload conditions.

Berbner et. al. [2] propose the use of heuristics to solve the QoS-aware service composition optimization problem, especially for business processes with *sequential execution*. Each task of the business process may be supported by different candidate web services. The goal is to compute an execution plan by selecting one service from each category to support the individual tasks (of the business process) in a way that optimizes an objective function under some QoS constraint. The objective function considered is the weighted sum of the overall QoS attributes. The simulated evaluation of the heuristic reveals that this heuristic is extremely fast and outperforms linear programming based solutions with regard to computation time, especially

with increasing number of candidate Web services and process tasks. That paper considers business processes with sequential execution only and not the ones with more complex flows such as loop and parallelism. Additionally, the authors do not elaborate on how they combine different QoS attributes with different units to construct the objective function used in this paper.

A number of authors have tried to solve the service composition optimization problem using linear integer programming. The focus was on workflows with sequential execution. The linear integer programming approach discussed by Zeng et. al. [16], and as confirmed by Canfora et. al. [5] and Yu and Lin [15] suggests that the approach is not scalable with increasing number of candidate services and activities in the business process. Genetic algorithms proposed by Canfora et. al. [5] and a heuristic based approach presented by Bernber [2] for service composition with sequential execution is an improvement over the linear integer programming approach.

D'Ambrogio [1] proposes a WSDL extension to desribe QoS characteristics of Web services as a metamodel transformation. Mei and Meeuwissen [3] introduce the concept of SLA negotiation space that may span multiple domains to realize end-to-end QoS requirements.

9 Concluding Remarks

This work considers service provider selection based on maximizing a utility function under cost constraints. The paper described the architecture and implementation of a QoS broker that selects service providers based on a client-provided utility function and on cost constraints also provided by the client. The broker uses a predictive analytic queuing network model to estimate the values of performance metrics such as response time and throughput for each possible service provider. Service selection is then based on maximizing a utility function under cost constraints. The selection of service providers performed by the QoS broker is optimal at session initiation time. Using the admission control mechanism described by one of the authors in [9], negotiated QoS levels are maintained throughout a session even if the workload conditions change with time. In order to validate the ideas presented in the paper, the QoS broker was implemented and experiments conducted to assess its behavior.

The performance and high-availability of the broker is absolutely critical to the continuity of operations and the success of service consumers, especially if the number of services, SPs, and consumers is large. There are a number of ways to enhance the availability and performance of the broker. One of them consists in the use of clustering [10], which allows a group of servers to work together and provide a unified front-end to the services offered by the servers in the cluster. Clustering facilitates high availability through server redundancy by replicating services across the servers. When a server in the cluster fails, another can take over without impacting the clients. Performance and scalability in a clustered environment can be achieved by adding more servers, if needed, and by using load balancing techniques.

We propose to use a cluster architecture to improve the performance and availability of the QoS Broker and use a cluster of Web/Application servers and database servers to achieve this goal. Such a fail-over technique requires persistence and replication of state information, such as the service demand matrix, SLAs, Table of Commitments, and service registration information. This information can be stored in a persistent storage such as relational database. Clustered databases and database replication techniques can be used to replicate the state information across all servers of the cluster.

References

[1] Andrea D'Ambrogio, "A Model-driven WSDL Extension for Desribing the QoS of Web Services," *Proc. 2006 IEEE International Conference on Web Services (ICWS06),* 2006.

[2] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Heuristics for QoS-aware Web Service Composition," *Proc. 2006 IEEE International Conference on Web Services (ICWS06),* 2006.

[3] R.D. van der Mei and H.B. Meeuwissen, "Modeling End-to-end Quality-of-Service for Transaction-based Services in Multi-Domain Environments," *Proc. 2006 IEEE International Conference on Web Services (ICWS05),* 2006.

[4] M.N. Bennani and D.A. Menascé, "Resource Allocation for Autonomic Data Centers Using Analytic Performance Models," *Proc. 2005 IEEE International Conference on Autonomic Computing,* Seattle, WA, June 13-16, 2005.

[5] G. Canfora, M.D. Penta, R. Esposito, and M.L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," *Genetic and Evolutionary Computation Conference (GECCO 2005),* Washington DC, USA, 2005.

[6] C.K. Fung, P.C.K. Hung, G. Wang, R.C. Linger, and G.H. Walton, "A Study of Service Composition with QoS Management," *Proc. 2005 IEEE International Conference on Web Services (ICWS05),* 2005.

[7] J. O. Kephart and R. Das, "Achieving Self-Management via Utility Functions," *IEEE Internet Computing*, vol. 11, pp. 40–48, January/February, 2007.

[8] M.C. Jaeger, G. Muhl, and S. Golze, "QoS-aware Composition of Web Services: A Look at Selection Algorithm," *Proc. 2005 IEEE International Conference on Web Services (ICWS05),* 2005.

[9] D.A. Menascé, H. Ruan, and H. Gomma, "QoS Management in Service Oriented Architectures," Performance

Evaluation Journal, North-Holland, *Elsevier Science,* Vol. 64, Nos. 7-8, August 2007, pp. 646-663.

[10] D.A. Menascé, "Trade-offs in Designing Web Clusters," *IEEE Internet Computing*, September/October 2002.

[11] D.A. Menascé, V.A.F. Almeida, and L.W. Dowdy, *Performance by Design: capacity Planning by Example*, Prentice Hall, Upper Saddle River, 2004.

[12] D.A. Menascé, "Two-level Iterative Queuing Modeling of Software Contention," *Proc. 10th IEEE Intl. Symp. Modeling, Analysis, and Simulation of Computerand Telecommunications Systems (MASCOTS'02),* October 11-16, 2002 Fort Worth, Texas, pp. 267-276.

[13] M.A. Serhani, R. Dssouli, A. Hafid, and H.A. Sahraoui, "A QoS Broker Based Architecture for Efficient Web Service Selection," *Proc. 2005 IEEE International Conference on Web Services (ICWS05),* 2005.

[14] W.E. Walsh, G. Tesauro, J.O. Kephart, and R. Das, "Utility Functions in Autonomic Systems," *Proc. Intl. Conf. on Autonomic Computing,*, New York, NY, May, 2004.

[15] T. Yu and K.J. Lin, "Service Selection Algorithms for Web Services with End-to-end QoS constraints," *Proc. 2004 IEEE International Conference on E-Commerce Technology,* 2004.

[16] L. Zeng, B. Bentallah, 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, No. 5, May 2004.