Optimizing Service Systems Based on Application-Level QoS

Qianhui Liang, *Member*, *IEEE*, Xindong Wu, *Senior Member*, *IEEE*, and Hoong Chuin Lau

Abstract—Making software systems service-oriented is becoming the practice, and an increasingly large number of service systems play important roles in today's business and industry. Currently, not enough attention has been paid to the issue of optimization of service systems. In this paper, we argue that the key elements to be considered in optimizing service systems are robustness, system orientation, and being dynamic and transparent. We present our solution to optimizing service systems based on application-level QoS management. Our solution incorporates three capabilities, i.e., 1) the ability to cater to the varying rigidities on Web service QoS in distinct application domains and of various users in a robust and heuristic manner, 2) the ability to formulate the overall system utility of a service system perceived by a particular system end user and to suggest its maximization using a utility model incorporated into a three-dimensional weighting scheme, and 3) the ability to dynamically achieve a higher perceived system utility of a service system via transparent negotiations. The calculation of the system utility encompasses a negotiation algorithm and a robust search algorithm for selecting heuristically best Web services. The effectiveness of the proposed algorithms and our solution is demonstrated by simulation experiments and our demo deployment, SSO.

Index Terms—Optimization of services systems, quality of services, service selection, composite services, system utility, negotiation, robust.

1 Introduction

Services represent a type of relationships-based interactions (activities) between at least one service provider and one service consumer to achieve a certain business goal or solution objective—[2]. Web services technology as a de facto realization of services computing and a key enabler of its disciplines has evolved into a system development logic that focuses on the effective selection and integration of distributed, heterogeneous, and autonomous system components in the form of individual Web services for building new systems. Such integrations rely on the capability of assembling individual system components over certain control and dataflows into service systems. Usually, a number of competing Web services are able to offer the functionality of a system component, and very likely, these Web services have different nonfunctional characteristics, or quality of service (QoS).

Network QoS parameters such as bandwidth, latency, jitter, and loss have long been studied by the researchers in the network area. QoS of networks pertains to the features of each network layer defined by the layered network model. QoS of Web services, on the other hand, is a set of characteristics defined specially for the particular middleware that allows software exposed as network reusable "services." To date, much work within the area of QoS of Web services has been in the context of satisfying QoS constraints and meeting QoS agreements on multiple individual QoS characteristics of Web services, such as execution time, execution cost, reliability, availability, and reputation. Such research works have facilitated service selection based on multiple individual QoS requirements. There has also been in-depth research into QoS aggregation and QoS-aware service composition. However, not enough attention has been paid to a system perspective in studying the quality of service systems and how that can contribute to optimizing service systems.

System optimization is a principle in services computing [1], [3] and can be seen as a foundation for building valid and efficient systems in the service-oriented paradigm. Most often, the quality of a service system is among the top concerns of system users, and therefore, constitutes a main objective for optimization of service systems. Techniques currently available for managing Web service QoS do not seem to provide a complete and general solution to optimizing service systems from the QoS perspective for a number of reasons. The reasons we have listed below are not meant to be exhaustive. Reason 1: Some QoS functions of service systems are best controlled at the application level from a system perspective instead of at the Web service level. For example, the flexibility on the timeliness of data delivery by the system can purely be dependent on the nature of the application domain itself. From this perspective, system optimization on service systems plays an indispensable role in engineering service systems. Reason 2: One purpose of services computing is to ease the job of the programmers, designers, and business operation managers in building software systems. An operation manager in an SOA-oriented business, when participating in the design of a service-oriented IT system, needs to know the expected performance of the system under a given resource level without micromanaging the constituent services. In view of

- Q. Liang and H.C. Lau are with the School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902. E-mail: althealiang@smu.edu.sg, hclau@smu.ed.sg.
- X. Wu is with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China, and the Department of Computer Science, University of Vermont, Burlington, VT 05405. E-mail: xwu@cs.uvm.edu.

Manuscript received 1 July 2008; revised 18 Dec. 2008; accepted 8 May 2009; published online 21 May 2009.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org and reference IEEECS Log Number TSC-2008-07-0061.
Digital Object Identifier no. 10.1109/TSC.2009.13.

1939-1374/09/$25.00 © 2009 IEEE Published by the IEEE Computer Society

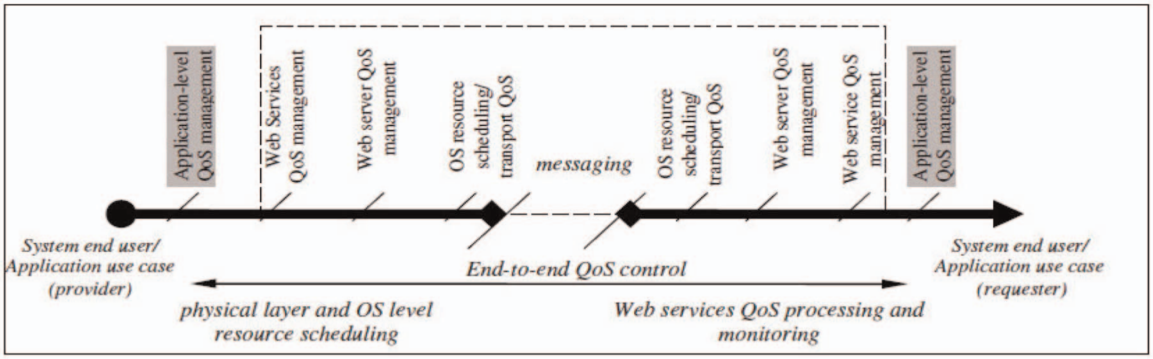

Application-level QoS management
Web Services QoS management
Web server QoS management
OS resource scheduling/ transport QoS
messaging
OS resource scheduling/ transport QoS
Web server QoS management
Web service QoS management
Application-level QoS management
System end user/ Application use case (provider)
End-to-end QoS control
System end user/ Application use case (requester)
physical layer and OS level resource scheduling
Web services QoS processing and monitoring

Fig. 1. A layered model of QoS control in service systems.

such needs, a general solution to optimizing service systems is very much in need.

We argue that there has not been a general approach proposed to system optimization of service systems. We argue that the key elements in a general approach to optimizing service systems must include at least robustness, system orientation, and being dynamic and transparent:

1. *Robustness:* The ability to cater to the varying rigidities on Web service QoS in distinct application domains and of various users using a robust solution in a heuristic manner;
2. *System orientation*: The ability to formulate the overall system utility of service systems in the system perspective of a particular system end user and suggest its maximization to that end user; and
3. *Being dynamic and transparent*: The ability to dynamically achieve higher perceived system utility of service systems (again heuristically) by adjusting QoS levels of different components of the system in a transparent way to the system end user.

In this paper, we try to contribute to the issue of optimizing service systems by presenting a general solution that accommodates the above three key elements. In particular, our contributions in this work include 1) supporting robust statements such as "If the quality of an application is better than λ with the probability of $1 - \varepsilon$, the expectation is considered fulfilled," 2) making it system-utility-enabled to take care of the distinct characteristics of measuring usefulness by the end users, and 3) encompassing dynamic and transparent negotiation to achieve the maximal usefulness of the system for its system end users.

The broad implication of the above contributions can be seen as twofold, i.e., proposing key aspects in system-level studies of service systems in the discipline of services computing and providing practitioners in the service industry with an option in improving the quality of service-based IT solutions. Our work has helped to identify, for the research community of services computing, the distinguishing system-level characteristics of software systems that are enabled by a service-oriented paradigm. Such distinguishing system-level characteristics can be used to form a basis of a bread of research focusing on the system-level study of service systems. Our work has also approached such characteristics in a way that accommodates the need and perspective of IT practitioners who are ready to adopt services computing. From the point of view of the IT industry, our work shall be useful to business operation managers who play an important role in designing, managing, engineering, and fine-tuning service-based software systems. With our work, the goal in achieving the utility of software systems can be seamlessly integrated into their service management practices. For example, in order to achieve IT automation, they can incorporate our research results into their service solution evaluation framework to fine-tune the solution they have produced for a best utility to the business.

The remainder of the paper is organized as follows: In Section 2, we detail three contributions of our proposal of an application-level QoS management for service systems and walk through a scenario to exemplify the idea of optimizing service systems. In Section 3, we describe a model of service systems in the context of optimizing service systems. Then, we elaborate the technical aspects of our solution. In Section 4, we present an algorithm for robust and system-utility-based optimization in our solution. In Section 5, we validate our solution by both simulation-based experiments and a demo deployment called SSO. In Section 6, we review the current research on WS QoS research for a comparison. We conclude the paper in Section 7.

2 Application-Level QoS Management

2.1 Distinguishing Features

QoS control in service systems can be seen as a layered architecture, from the network QoS control during message transmission, to the physical layer and OS-level resource scheduling at the host, to Web server QoS management, and finally, reaching the processing and monitoring of WS QoS. This layered model can be illustrated as the dashed line box in Fig. 1 [29]. Our observation is that for optimizing service systems, the gap between WS QoS and its utility to system end users must be filled. Our approach is to introduce an extra layer of application-level QoS management as a bridging layer. The design of this layer entails a number of considerations in terms of optimizing the QoS of service systems. Fig. 1 also shows the application-level QoS management for service systems within the layered QoS control architecture. This layer is expected to seamlessly establish the linkage of WS QoS and the perceived values or utilities by the ultimate end application use case in order to guarantee a satisfaction level of system end users. This layer is designed with the following distinguishing features:

1. *Being robust*, which is designed to cater to distinct domains and various end users with different degrees of rigidities on the QoS of the service systems. Optimization of service systems needs to accommodate the stochastic nature of Web services and entail an effort of providing a spectrum of QoS offerings with certain probabilities, while maintaining a satisfactory level to the system end users. This has not been investigated before. We claim that given uncertain QoS values, a best lower bound to the quality of a service system is a powerful measurement that can be used to both communicate with the service consumers and evaluate the entire application. In view of this, we propose a heuristic-based service selection scheme for a robust quality analysis of service-based applications. Multiple QoS attributes such as execution time, reliability, availability, and cost are represented by their means and variances. The evaluation of an application built on certain Web services is cast as an optimization problem under probabilistic constraints and solved efficiently (albeit heuristically) by a local search method that guarantees to return the best quality level ε such that the probability that the actual quality received by the end user falls below q is within a prescribed threshold ε.
2. *Being system utility enabled,* which seeks to provide a system-oriented view of a service system in terms of its overall system utility perceived by a particular end user. The system utility perceived varies across different end users. Due to the budget limit and other possible constraints, a trade-off is always unavoidable. In this case, it is important to model the preference of receiving higher quality of services on one component to that of another, for example, positioning and navigation versus video playing in the scenario (which will shortly be described in the next section). In other words, the utility increase due to quality improvement on one component is more significant than that on another. Further, for various users or various applications, often the sensitivity along different quality dimensions also varies, for example, more cost sensitive or more execution time sensitive. In this case, the utility against different quality dimensions varies accordingly. The utility model proposed by us distinguishes our solution by accommodating the preferences and constraints of the system end users. Particularly, in our solution, we aggregate measures of a QoS dimension over all service components as in some existing studies, e.g., [11], [20]. We also apply a general three-dimensional weighting scheme to enable a combination of quality measures over all quality dimensions. Further, the utility functions of all quality dimensions over all components are incorporated into the scheme to provide a perceived system utility of the system end user. The system utility is maximized using search-based techniques for the intended system end user.

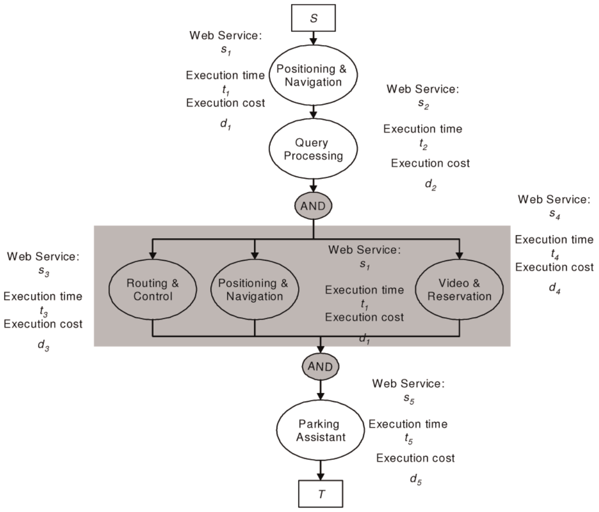


Fig. 2. Precedence graph of a mobile distributed application.

3. *Dynamic and transparent negotiation*, which, based on the first two features, provides a transparent negotiation mechanism in order to achieve a maximal overall perceived system utility. Due to the dynamisms of the dependent factors of Web services and of service providers' business strategies, Web service QoS provisions may change from time to time for multiple times of usages by the service system and even during one single service invocation. Meanwhile, in the peer-to-peer interactions among the collaborating service systems, resource fluctuations, e.g., due to transient overloads or resource failures, at the local server will affect the QoS constraints imposed on the remote services. In either case, adjustment, i.e., degradation or upgrading, of certain QoS dimensions through service offering reassignments may be necessary in order to maintain a maximum perceived utility of the system. In contract to the existing Web service negotiation research, the negotiations among multiple service implementations and/or among multiple service components in our solution are transparent to the system end users. The resource available at the local site is determined by a local heuristic and the QoS of the remote services is also determined by the remote sites. Given the local resource constraint and QoS of the remote service offerings, our QoS (re)negotiation mechanism ensures graceful QoS degradation and upgrading in the cases of local or remote resource changes during runtime.

2.2 Scenario of Service System Optimization

In this section, we use Web service execution time and execution cost as example quality attributes to motivate the robust and system-utility-enabled application-level quality control with a capability of dynamic and transparent negotiation.

Here, we put forth a mobile and distributed application that is able to answer queries of mobile device users. This application has been implemented and used for evaluating the deployment of our system optimization solution. A graph that shows its possible components and optional precedence relationships among them is given in Fig. 2.

Imagine that someone in a (partially) autonomous vehicle loaded with some mobile devices on a back street in the Central Chao Yang District neighborhood of Beijing, China, looking for an Italian restaurant. The passenger can issue a query to the vehicle like "What are the closest Italian restaurants to have original lasagna?" The application needs to answer this query in a timely fashion because the vehicle is moving and the services need to be completely delivered before the vehicle and passenger move out of the area.

This application is an example of the class of application domains whose QoS measurements are not a fixed value, which require a stochastic analysis. Any Web services used by such a service system may be executed under uncertain environmental factors, input/output factors, and platform factors of distinct servers. Since the dependent factors of these quality measures are variable, a service provider provides services at uncertain levels of QoS at different times, and therefore, demands a variable charge.

The variations on QoS dimensions of systems may or may not actually allow flexibility of the analysis depending on the nature of the applications, i.e., different rigidities on their QoS requirements. In the case of video playing, if a small percentage of the speech or images is delayed and distorted, the utility to the end users may not be affected. In terms of cost, some applications allow a flexible budget. As far as the overrun of the budget is within a certain percentage and the probability is not too high, the cost constraint is considered to be satisfied.

3 QoS-Based Optimization of Service Systems

In this section, we discuss in details of application-level QoS Management for service systems.

3.1 Model of Service Systems

A service system can be seen as a collection of service components, each requiring completion of certain tasks. These service components are organized into some structure within the service system in order to achieve a goal. A service system can thus be formalized as a collection of tuples, each of which consists of a distinct system component and its corresponding number of occurrences within the system, i.e.,

$$ss = \{\langle c_1, o_1\rangle, \ldots, \langle c_n, o_n\rangle, \langle c_{n+1}, o_{n+1}\rangle\}. \quad (1)$$

The ith instantiation of a particular service system (referred to as instantiation i of SS) is a collection of Web service assignments to each distinct system component j, i.e.,

$$ss_i = \{\langle s_{i1}, c_1\rangle, \ldots, \langle s_{ij}, c_j\rangle, \ldots\}. \quad (2)$$

If the average number of possible assignments for a component is M and the number of components is N, i can range from 1 to $M \times N$. The only constraint on assignments is that one component needs one and only one Web service assignment. There is not such a constraint that one Web service can only be assigned to one system component. In other words, the same Web service can be used multiple times in a service system.

A number of QoS attributes for Web services have been proposed in the existing research on QoS, e.g., execution time, execution price, reputation, successful execution rate, and availability. Some WS QoS attributes are tightly coupled with infrastructure and some are not.

3.2 Robust Service Selection

Most existing researches on QoS have not taken a robust approach to evaluating Web services against some or all of the quality criteria. In other words, all existing evaluations have taken a deterministic perspective when studying the quality of Web services, which is normally not useful enough when dynamic resource constraints are present. For example, the execution time of certain Web services provided by a software asset may fluctuate due to the processing load on the server that hosts the Web service, or the reliability of the Web service may only be known as a probability. Calculating only a bound value of the quality is not able to provide enough information.

Therefore, the probability issue in Web service QoS must be taken into consideration for a solution to optimizing service systems. One approach is to model the measurement of the QoS level of Web services on a particular QoS dimension as a random variable with a given mean and variance value. In this section, we propose a service selection scheme for optimizing service systems based on a robust analysis of quality of services. Multiple quality attributes including execution time, reliability, and availability may be considered from the perspective of a system end user within the overall capacity and constraints/preferences of the service system. These quality attributes are depicted as probability models with certain means and variances. Service selection is cast as an optimization problem with probabilistic characterization. This optimization problem is solved by a robust local search that returns the best quality level q such that the probability that the actual quality received by the system end user falls below q is within a given threshold ε. The approach that we present in the paper solves the problem of optimization of service systems at two levels as follows:

- What is the best lower bound on quality measurement of a service system within a given probability?
- Which set of assignments of Web services to the service components in the service system yields the best quality (even in the worst scenario) within a given probability?

Robust optimization has been applied to deliver promising results in immunizing uncertainty in optimization against infeasibility while preserving the tractability of the model. A robust local search framework can efficiently solve resource constraint scheduling problems with data uncertainty. Given a value $0 < \varepsilon \leq 1$, the robust objective value, i.e., the optimal value if we allow an ε-chance of not meeting it, assuming that certain data values are defined on bounded random variables given by $\tilde{z}$, can be computed [29].

Let

$$V(x, \tilde{z}) \quad (3)$$

denote the random variable representing the objective value for solution x under uncertainty $\tilde{z}$. The robust local search scheme is based on the definition of a robust fitness function associated with ε, shown in (4):

$$f(x, \tilde{z}, \varepsilon) = E[V(x, \tilde{z})] + \sqrt{\frac{1-\varepsilon}{\varepsilon}}\sqrt{Var[V(x, \tilde{z})]}. \quad (4)$$

Let V^* denote the solution value yielded by the local search with respect to the above fitness function. Then it can be shown that (5) is also true. A detailed description of the algorithm design and its implementation are provided in Section 5:

$$P(V(x, \tilde{z}) \leq V^*) \geq 1 - \varepsilon. \qquad (5)$$

3.3 Enabling System Utility for End Users

End users of service systems observe the quality of service systems from a system perspective that is a collective effect of the local quality values of all individual component services encompassed by the system. A system perspective of a system end user also entails a utility measure that spans across all quality dimensions of the system and relies on the preferences and constraints of the particular end user. Utility theory has been applied to model Web service requests in [22]. Here, we propose a comprehensive utility model to model the overall system utility perceived by a particular end user based on QoS measurements. We have designed a three-dimensional weighting scheme for combining quality measures over all quality dimensions and over all components. It uses the utility model for optimizing the service system. Our utility model is described below.

Using the standard multiattribute decision analysis [10], a component could have defined upon it a vector of QoS metrics q, shown as in (6),

$$q = \{q_1,\ q_2, \ldots, q_m\}, \qquad (6)$$

within which $q_1, q_2, \ldots q_m$ represent the individual QoS dimensions. Such q is also known as decision variable vector in optimization problems. For the ith instantiation of the service system, all its assignment utilities form a vector, shown in (7). A utility vector that corresponds to all QoS dimensions of the jth component in an instantiation i of a service system, i.e., $\bar{f}_j$, can be written as (8), where $f_{jm}(\cdot)$ is the mth QoS dimension of jth component. Each elemental function in this vector may or may not be dependent on each other. Based on this, a utility matrix that corresponds to the ith composite Web service instance can be written in (9):

$$U_i = (u(s_{i1})^T, u(s_{i2})^T, \ldots, u(s_{ij})^T, \ldots, u(s_{in})^T), \qquad (7)$$

$$\begin{aligned} u(s_{ij}) &= \\ \bar{f}_j &= \begin{pmatrix} f_{j1}(q_{ij1}) & \cdots & f_{jm}(q_{ijm}) & \cdots \end{pmatrix}. \end{aligned} \qquad (8)$$

In our combinational and aggregational study of application-level QoS, utility is examined in a comprehensive way by applying a three-dimensional weighting scheme. This three-dimensional scheme is based on 1) a collection of two dimensional weighting vectors, in particular, a sequence of weighting vectors W_j, as shown in (11), each of which corresponds to the weighting functions for all QoS dimensions of the jth component and 2) a weighting vector W_F, as shown in (12), which corresponds to the weights for all the components of the service system. The weighting scheme is introduced to model the overall utility V, written as (10).

These weighting vectors reflect the facts 1) that for the same set of QoS dimensions, the relative weightings of these QoS dimensions can be distinct within different subfunctions of a service system, 2) that such relative weightings can also vary with the QoS values, and 3) that the weightings of distinct components within the service system can be different:

$$\begin{pmatrix} \bar{f}_1^{\,T} & \cdots & \bar{f}_j^{\,T} & \cdots \end{pmatrix}, \qquad (9)$$

$$V = U_i \times W, \qquad (10)$$

$$U_i = \begin{pmatrix} u_{i1} \\ \cdots \\ u_{ij} \\ \cdots \end{pmatrix} = \begin{pmatrix} W_1 \times \bar{f}_1^{\,T} \\ \cdots \\ W_j \times \bar{f}_j^{\,T} \\ \cdots \end{pmatrix}, \qquad (11)$$

$$W_F = \begin{pmatrix} \dot{w}_1 & \cdots & \dot{w}_j & \cdots \end{pmatrix}. \qquad (12)$$

3.4 Transparent QoS Negotiation

WS QoS negotiation refers to the process of making trade-off on WS QoS metrics among multiple service entities in order to achieve a maximum utility possible, perceived by the system end users. Static WS QoS negotiation happens during service discovery for the purpose of feasibility assessment of a service system. Dynamic WS QoS negotiation happens when specified failure conditions are violated, e.g., when the end user's perceived utility of the service being executed has to drop at runtime. Two distinct reasons might have caused such a drop. The provider of particular Web service may experience high load and decide to drop the quality level. Alternatively, the scarce of the server resource at the end user side may also require reducing the QoS of the services of the service system.

WS QoS negotiation can take place at different levels. A pool of Web services can be configured to be managed by one single (virtual) service provider, who appears to the end user as a provider who is in charge of all the implementations within the pool. In this case, negotiation can autonomously be initiated by the virtual service provider to do QoS capacity pooling for the services within the pool. We refer to this as service pool negotiation. Service pool negotiation is switched on transparently to the user whenever sharing QoS capacity is helpful in improving the utility.

Distinct components of a service system most probably show different variation characteristics in each QoS attribute. Since the functions of each component may be of a different utility to the user, trade-offs among components are necessary before or during the service execution to maximize the overall perceived utility of the service system. This negotiation is referred to as component negotiation. Component negotiation relies on service pool negotiation. The negotiation results at the service pool level are used to analyze the best trade-off among multiple components. Component negotiation maintains its transparency to the user as far as the perceived utility has not violated the failure conditions. Negotiation may result in degradation or upgrading of the perceived utility. However, the best utility possible with the current QoS capacity is guaranteed. In this case, the end user will not get involved in the negotiation. An exception to this is that the specified constraints in one or more QoS attributes are validated. In this case, the user will be notified and participate in the negotiation.

Both service pool negotiation and component negotiation use the same negotiation algorithm. Let N denote the number of services that are involved in a service pool or a service system and that have resource allocation conflicts.

Let x_k denote the resource demand of service k and $v_k(x)$ denote the payoff of service k if the resource allocation strategy of $x = (x_1, \ldots, x_N)$ is selected. An example of payoff is the service execution time due to allocated CPU computing resources. $x_k(i)$ and $x(i)$ denote the allocation of resource(s) at the ith iteration. Let $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_k)$ denote the resource allocation if no agreement is reached. The negotiation algorithm can be described as the following.

Algorithm 1.

```
$i = 0$;
Each service submits an initial resource request $x_k(i)$;
IF the resource allocation strategy $x = x(i)$ is feasible Quit;
ELSE {
L1:
    Select one service to degrade (upgrade) its demand to the next level that achieves the minimum decrease (maximum increase) of payoff;
    IF no services can be selected {
        $x = \tilde{x}$ and Terminate;
     }
     ELSE
        $x = x(i+1)$;
     IF $x(i+1)$ is feasible Quit;
     ELSE {
        $i = i + 1$ and go to L1;
     }
}
```

4 Robust and Utility-Based Optimization

We have chosen a heuristic approach of local search to optimizing service systems mainly out of performance consideration. When service providers have limited capacity, the problem is NP-hard. To solve such an NP-hard problem, approaches like Integer Programming, which rely on an enumeration method of branch and bound, will be much more time-consuming comparing to local search. The stochastic version of the problem adds complexity on top of that and gives us more incentives to take a heuristic approach. We propose an algorithm to perform robust search for a heuristically good system. Our purpose is to hedge uncertainty in Web service offerings via a robust model (versus a conventional optimization model). The algorithm is given in Algorithm 2. Given 1) a service system composed of service components, 2) the mean and variance values of all possible Web service assignments, and 3) a value of ε, which can been seen as a level of risk, the algorithm will return a collection of Web service assignments to the components of the service system that has the best quality value, or V^*.

In essence, we perform a robust local search on the neighborhood set of assignment list. An assignment list is defined as a precedence-constraint feasible assignment of Web services to the components. The algorithm first performs some preprocessing. It finds the component assignments that are dominated by some other assignments. For example, if the Web service assignment fluctuates vastly and the average quality value is not high, i.e., with a larger variance and a smaller mean, it will not be picked to be part of the assignment list being searched for. Such assignments will be removed during the preprocessing.

The algorithm then explores different assignment lists by local moves in which one assignment is switched in the assignment list. The switch that maximizes the quality is selected as the next local move. We set the maximum number of local moves as Max_Iteration. When the algorithm stops, it outputs the best quality value and its associated assignment list for the service system with the given conditions. Local moves direct the search to approach the assignment list with the maximum V^* value. In particular, local moves must move from a feasible or a nonfeasible solution to a feasible solution that has a larger V^* value than the currently achieved best value V^*. Since the feasible solution regions may not be connected in the search space, local moves that are restrained to one feasible solution region will fail to reach feasible solutions in other unconnected feasible solutions. We, therefore, diversify the exploration of assignment lists in the feasible solutions by the local search move from a feasible solution to a nonfeasible solution.

With such considerations, we design the local move in the following way: If an assignment list is feasible, the algorithm randomly picks a component and then picks from among the available Web service assignments to swap with the current assignment to the component. If the new assignment list yields a larger V^* value, this is accepted by the algorithm as the next local move. In order to explore the entire search space, a small probability to accept an infeasible solution is introduced. In Algorithm 2, this small probability is set to be 0.01. The current minimum value at each local move is stored in V^*.

Algorithm 2.

```
L1: for each component of the composite Web service do
Remove the assignments that cannot be part of the result to be output;
Generate a composite Web service assignment list AL randomly;
IF AL is feasible, compute $V^*_{now}$; $V^* = V^*_{now}$
ELSE Go to L1;
FOR i = 1 to Max_Iteration do{
  L2: Select one component c in AL randomly;
  Randomly select another assignment $a'$ of c;
  Swap Web service assignment a of c with $a'$ and form AL$'$;
  IF AL$'$ is feasible{
       Compute $V^*_{now}$;
       IF $V^*_{now} \leq V^*$ Go to L2
       ELSE $V^* = V^*_{now}$ and AL = AL$'$
       }
   ELSE{
      p = rand(0,1);
      IF (p < 0.01) AL = AL$'$
      ELSE Go to L2;
   }
}
```

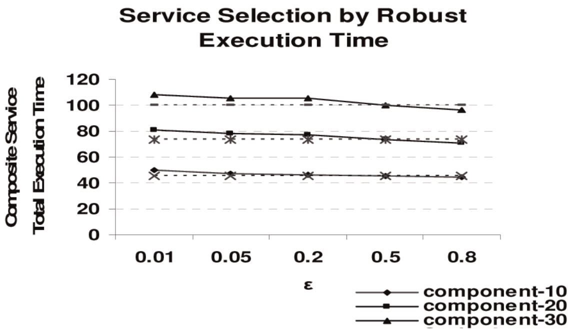


Fig. 3. Execution time (millisecond) of the optimized service systems.

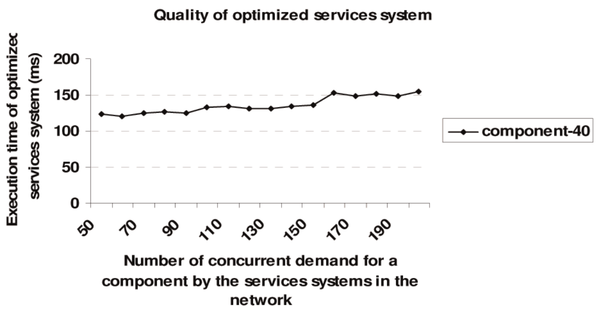


Fig. 4. Execution time (millisecond) of the optimized service systems with uniform limited capacity of providers.

5 Evaluation

5.1 Robust and System Utility Analysis

In this section, we present simulation experiments by applying Algorithm 2 to find an optimized service system against the time dimension of QoS. We run this algorithm against randomly generated service precedence graphs, each with sets of available Web service assignments for its service components. The algorithm is implemented in Java and the results have been obtained by running the algorithm on a P4 2.9 GHz computer with 1.5 GB of RAM.

Below are the details of the experiment setups. A total of 50 service precedence graphs have been generated. The numbers of distinct software function components of generated precedence graphs vary over five values, i.e., 10, 20, 30, 40, and 50. Ten different precedence graphs are generated for each number of components. For each precedence graph generated, we also randomly produce multiple Web services that can be assigned to each service component, respectively. The total numbers of service offerings vary from 50 to 150.

We have considered three schemes in terms of the relationship between the capacity of service providers and the demand of service components in the context of multiple service system on the same network. Scheme SSC, we assume that there is roughly the Same amount of demand for each component and a Similar number of Web service offerings for each component. Each service provider is Capable of handling all incoming requests on service offerings, which is usually the case in normal situations. Scheme SSL, we assume that there is roughly the same amount of demand for each component and the number of Web service offerings for each component is roughly the Same. Each service provider has a Limited capacity in handling incoming requests. But the capacity of all service providers in combination is adequate for the incoming requests. Scheme VSL, we consider that the amount of demand for each component may Vary and the number of Web service offerings for each component is roughly the Same. Each service provider has a Limited capacity in handling incoming requests. But the capacity of all service providers in combination is adequate for the incoming requests.

We have used the following five variance values: 0.1, 0.3, 0.5, 0.7, and 0.9, and the following three mean values: 3 (ms), 5 (ms), and 7 (ms), which give a total of 15 different variance and mean combinations. We have randomly chosen a variance and mean combination for a Web service offering. Max_Iteration in Algorithm 2 is set to 1,000.

The first part of the experimental study of robust search is on the search results in terms of the execution time of the optimized service systems. The results shown in Fig. 4 are the averages of those of all 10 precedence graphs with the same number of components. Three solid lines in the figure are charted against the resultant execution time of the precedence graphs with 10, 20, and 30 service components, respectively. For each type of precedence graphs, the robust execution time decreases as the values of ε increase. The higher level of risk the user of the service is willing to take, the shorter execution time the service system can possibly get. Using traditional approaches solely based on the expected values of components' execution time, we can get the resultant execution time of the service system as a fixed value over all different ε values for a given precedence graph, which are shown as dashed lines paired with the corresponding robust analysis results in Fig. 3. We can see that a fixed value is at the same level as the robust value with $\varepsilon = 0.5$.

We observe in the figure that service precedence graphs with more software function components experience longer robust execution time. This is the case when the same experiments are repeated a number of times (in our case, three times) and the mean values of the Web service assignments are picked randomly. We can also observe that service systems with a larger number of components demonstrate a steeper slope on the corresponding line.

If we compare the robust analysis with the worst-case-based analysis, which is sometimes required (e.g., for life critical applications), the robust analysis provides a quantified trade-off analysis on the upper bound execution time and the possibility of not reaching this upper bound. Such analysis results help the service user to make a decision on the desired service selection options.

In two other experiments, we fix the complexity of the problem and experiment on optimizing service systems with 40 components. The value of ε is set to 0.2 and mean is set to be 3 ms. We use scheme SSL and the number of service offerings keep unchanged for different runs of experiment. The capacity of each service providers is set to be 50 calls. Fig. 4 shows that the execution time of the optimized service system periodically jumps up as the current demand for a component increases, indicating a drop of the quality of the optimized system. For example, when there are between 50 and 100 concurrent calls from the service systems in the network, the quality of the optimized service systems maintains at a level around 124 ms. Over 100, the execution time increases to a level around 134 ms. Then it maintains at

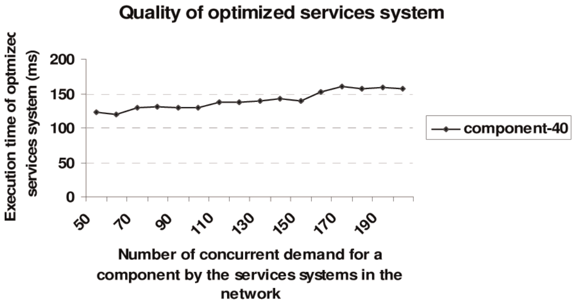


Fig. 5. Execution time (millisecond) of the optimized service systems with limited capacity (normal distribution) of providers.

around 134 ms until the number of concurrent calls reaches 200, when the execution time again increases to a level around 153 ms. This is due to the limited capacity of the service providers.

Once a threshold is reached, the quality has to be compromised because when a server is fully loaded, the assignment to this service provider is not considered feasible and another assignment (possibly with lower quality) has to be made, and thus, reduces the quality level.

We simulate the VSL scheme in a similarly way. The value of ε is set to 0.2 and mean is set to be 3 ms. In this experiment, we have assumed that service demand of various components follows a normal distribution statistically with a mean of 50 and standard deviation of 10. Fig. 5 shows that the execution time of the optimized service system jumps up as the current demand for a component increases, indicating a drop of the quality of the optimized system. But the jump up does not seem to necessarily follow a fixed periodic pattern. Generally speaking, it remains at a level for a while before it jumps up again. This is due to the limited capacity of service providers. When a provider is fully loaded, no assignment will be made to the service offerings on this provider even if they have the best quality. Since the demand follows a normal distribution, there is not a clear pattern in terms of when the execution time of the optimized system gets increased or the quality gets dropped.

Next, we have performed experiments to study the performance and sensitivity of the algorithm. For performance study, we run Algorithm 2 over precedence graphs of 30 components, each with 15 service offerings. For sensitivity study, we run Algorithm 2 over 50 precedence graphs with 10, 20, 30, 40, and 50 function components, respectively. The value of ε is set to 0.2. We use scheme SSC and the number of service offerings keep unchanged for different runs of experiment. We vary the maximum number of iterations (Max_Iteration) in the algorithm between 200 and 1,600. We measure the execution time of the algorithm and the quality of the search results (in terms of the mean execution time of the optimized service systems). The results are shown in Figs. 6 and 7.

As Fig. 6 shows, the execution time of the algorithm increases proportionally with the increase of the value of Max_Iteration. Since for each iteration, the amount of time spent on checking the neighborhood of a solution keeps the same if the number of service offerings keep the same, the time spent on searching for a solution will bear a linear relationship with the value of Max_Iteration. Fig. 7 depicts the change of solution quality when we increase the value of Max_Iteration.

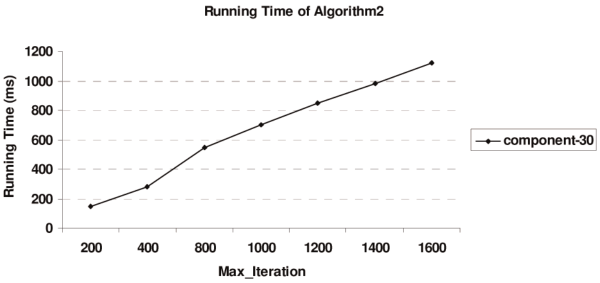


Fig. 6. Algorithm running time.

Each line in the graph presents solutions to a class of problems with certain complexity. For example, the top line corresponds to problems of optimizing service systems with 50 components and the bottom line corresponds to problems of optimizing service systems with 10 components. We see that a general downward trend with each line, which tells a smaller execution time (or an improved quality) of the solution when the value of Max_Iteration increases. This is true for problem instances with any level of complexity (i.e., service systems with from 10 through to 50 components).

We can also observe that as the value of Max_Iteration increases, the decrease on the execution time of the resultant service system more or less stops at some point. In other words, the quality of the solution no longer increases with the increase of Max_Iteration after certain point. This is generally true for problems with different complexities, although the stop happens at different stages. In the cases of 10, 20, and 30 components, this seems to happen when Max_Iteration reaches 1,000. In the case of 40 components, quality improvement stops when it reaches 1,400. In the case of 50 components, the quality seems to continue improving as it reaches 1,600. We show that Max_Iteration affects the quality of the optimization solution until a certain point, which depends on the complexity of the problem.

Finally, we measure the distance between the solutions found by our local search algorithm and the "optimal" solution. For this purpose, we run Algorithm 2 over 50 precedence graphs with 10, 20, 30, 40, and 50 function components, respectively. The value of ε is set to 0.2. The maximum number of iterations (Max_Iteration) in the algorithm is set to be 1,600. We measure the quality of the search results (in terms of the mean execution time of the optimized service systems).

We also run simulated annealing, with the maximum number of iterations set to 25,000, on the same set of

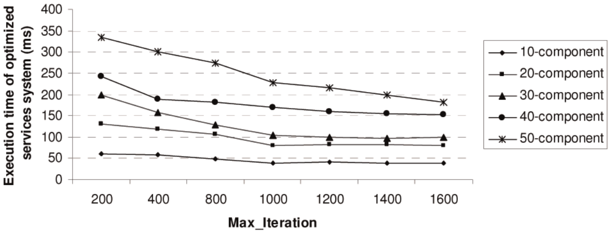


Fig. 7. Quality of resultant service system with different Max_Iteration values.

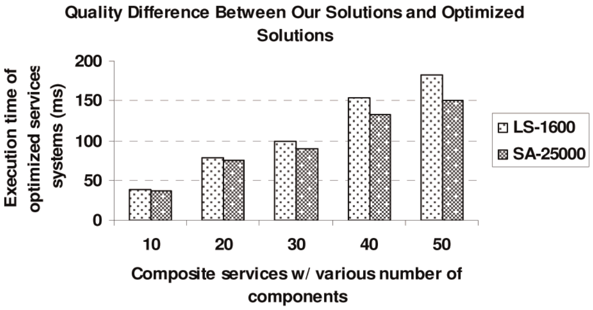


Fig. 8. Difference between LS-1600 and SA-25000.

precedence graphs. The purpose in doing this is to achieve solutions with as high as possible quality. We then compare solutions produced by our system with the ones generated by simulate annealing. The results are shown in Fig. 8.

For all experiments, we repeatedly run the algorithm on 10 different precedence graphs and take the average over the results on all the precedence graphs. In Fig. 8, LS-1600 series corresponds to the solutions generated by our system and SA-25000 series corresponds to the solutions generated by simulated annealing. In all cases, the solutions found by LS-1600 are suboptimal and worse than the solutions found by SA-25000. In other words, the time efficiency costs us the quality of the results.

We see that in the cases where the component numbers are low, the quality difference is also relatively small. For example, in the case of 10-component, the difference is only 2 percent out of 38 ms. As the number of components increases, the quality difference becomes more obvious. In the case of 50 components, the difference is 15 percent out of 183 ms. This is due to the increase of cost on searching for an "optimized" solution of a problem instance with a higher complexity. In other words, a larger number of iterations will be needed to find a solution of the same quality comparing to the cases with smaller numbers of components.

It is more important that the heuristics can be applied to reach a reasonably good-quality solution with a reasonable time complexity for a problem instance of a high complexity. In service applications that need in-time quality adjustment, which we are focusing on in this paper, it is most probably not worth improving the quality by ambitiously increasing the number of iterations after a certain point. The reason is that the increase on the quality has usually decreased to a low level with a slow speed at this point.

5.2 Transparent Negotiation

In this section, we show experimental results of utility study when the transparent QoS negotiation architecture is applied. An end application's utility is specified by the parameters of its execution model. For illustration purpose, we model execution time as the single negotiable parameter, i.e., we only show the utility perceived by the application resulting from the negotiation on execution time. The utility $u_{T,j}(\cdot)$ is then quantified as $u_{T,j}(t) = (t_j^{\max} - t_j)/(t_j^{\max} - t_j^{\min})$, where t_j is the execution time of the jth component on a particular negotiated Web service offering and $t_j^{\max}$ and $t_j^{\min}$ are the maximum and minimum execution time of the jth component among all negotiated Web service offerings, respectively. As expected, a shorter execution time corresponds to a better utility.

TABLE 1
Execution Times, Utility Levels, and Weights of the Components

Virtual provider of Component	Execution Time (ms)	Utility Level	Weight
a	*27*	*0*	*10*
	16	*0*	*10*
	12	*0.5*	*15*
	8	*1*	*20*
b	*20*	*0*	*1*
	16	*0.25*	*30*
	12	*0.5*	*50*
	8	*0.75*	*100*
	4	*1*	*150*
c	*27*	*0*	*10*
	16	*0*	*10*
	12	*0.5*	*15*
	8	*1*	*20*

We use the negotiation algorithm listed in Algorithm 1 to (re)compute the set of QoS levels for all components of the service system in order to maximize the weighted sum of their utility. Here, the overall utility is modeled as the weighted sum of the utility of each individual Web service shown as in (13), where V_i denotes the overall utility of the service system,w_j is the weight of the jth component with a particular negotiated resource, $w_j^{\max}$ is the maximum weight of the jth component mong all negotiated resources, and $u_{T,j}$ is the utility of the jth component with a particular negotiated resource. Recomputing the QoS levels may involve lowering the utilities of some individual Web services to accommodate more important components. However, the aim is to maximize the overall utility perceived by the end application.

We assume that we have components a, b, and c in a service system whose QoS levels, due to the limited resource, have to be negotiated in order to satisfy the user best. The (mean) execution times, the corresponding utility levels, and the weights of these components are shown in Table 1. We also assume that each of these components is required by this service system, at least at a degraded QoS level, and therefore, the Web service offerings used by all components are always accepted by the QoS Negotiation module. Using the utility levels and weights listed in Table 1, we illustrate the behavior of the negotiation using the negotiation heuristic of Algorithm 1. In this experiment, we kept the number of components fixed and decreased the computation power of the local server, i.e., its CPU speed, dedicated to the service system. This triggers a renegotiation with the SSO on the preferred QoS of the services of the service system. The result of renegotiation is a degraded QoS or an increase in the execution times. We then observed the corresponding decrease in component utility levels and the overall utility level. Fig. 9 shows the execution time (calculated using means) of the service system and its components resulted from negotiation versus the CPU speed. Fig. 10 plots the normalized utility levels versus the CPU speed.

As shown in Fig. 9, the mean execution time of component b keeps at the best level of 4 ms as the CPU speed decreases. It only starts degradation to the next level of 8 ms when the CPU speed drops to 20. As for components a and b, their execution time continuously increases from 8 to 12 ms, then 16 ms, and finally, 27 ms as the CPU under testing slows

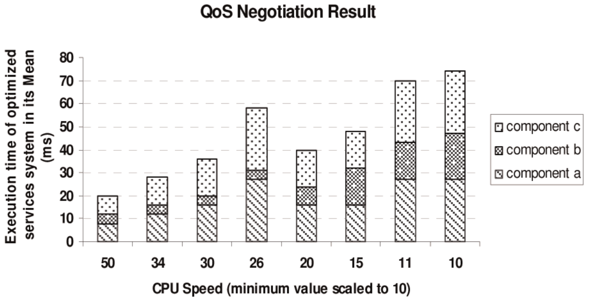


Fig. 9. Execution time (millisecond) of optimized service systems.

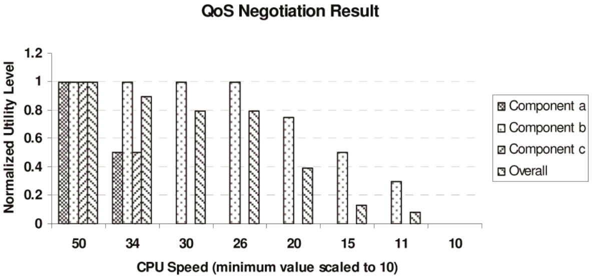


Fig. 10. Utility levels of optimized service systems.

down. We see the summation of execution times at CPU speed of 26 higher than those of two slower CPU speeds.

$$U_T = \sum_{j=a,b,c} w_j \times u_{T,j} \Big/ \sum_{j=a,b,c} w_j^{\max}. \qquad (13)$$

As shown in Fig. 10, components a and c quickly degrade to QoS level 0.5 and then 0 as soon as all better levels are no longer possible. This indicates that to keep component b fast at around 4 ms is more important than the summation of the three execution times, which is in consistency with what the utility functions tell. We have this result primarily because components a and c are less critical, so the loss in the total utility due to their degradation is not as great. Also shown in the figure is the overall utility, calculated as the weighted summation of the utility levels of individual components.

Following this result, we further examine how much the negotiation results depend on the parameter choices of utility, weight, and execution time as shown in Table 1. We intend to prove the general validity of the system independently of the specific characteristics of the deployment environment using the following experiments. We prepare parameter values with different characteristics, respectively, for utility level, weight, and execution time. As shown in Table 2, the utility levels vary from one constant value (b-4) to changing in various ways: linear (a-b-c), polynomial (b-1 and b-2), and staircase (b-3). The parameter of weight, as shown in Table 3, can become increasingly dominant as utility changes (b-1), or gradually become dominant from subdominant (b-2), or roughly remain tied with others (b-3), or stay at the same dominant level (b-4). For execution time, as shown in Table 4, one component may take obviously longer (b-2), shorter (b-1), or the same execution time than the other components. While testing the impact of each individual parameter, the other parameters are fixed to the same choice, which correspond to the gray columns in the tables.

TABLE 2
Variations of Utility Levels for Components

a-b-c	*b-1*	*b-2*	*b-3*	*b-4*
0	0	0	0	0.5
0.2	0.316	0.008	0.2	0.5
0.3	0.447	0.027	0.2	0.5
0.4	0.548	0.064	0.4	0.5
0.5	0.707	0.125	0.4	0.5
0.6	0.775	0.216	0.6	0.5
0.7	0.837	0.343	0.6	0.5
0.8	0.894	0.512	0.8	0.5
0.9	0.949	0.729	0.8	0.5
1.0	1.0	1.000	1	0.5

TABLE 3
Variations of Weights for Components

a-b-c	10	10	10	10	10	10	10	10	10	10
b-1	20	30	50	60	70	100	150	200	250	250
b-2	5	6	10	20	30	30	50	100	150	150
b-3	15	15	15	15	15	15	15	15	15	15
b-4	50	50	50	50	50	50	50	50	50	50

TABLE 4
Variations of Execution Time for Components

a-b-c	70	70	64	50	46	38	27	16	12	8
b-1	40	36	32	28	24	20	16	12	8	4
b-2	160	110	96	80	53	47	40	32	27	16

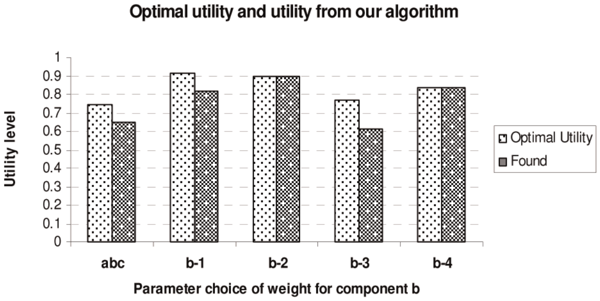


Fig. 11. Results against parameter choices of weight.

We list the solutions calculated by our system under the given varying parameter choices and compare them with the optimized solutions. The results are shown in Figs. 11, 12, and 13. We can see that in all cases, the quality of the solutions generated by our system is not seen to be affected by the parameter choices. We can measure the quality of solutions by "precision," i.e., the percentages of utilities in our solutions out of those of the optimized solutions. Although the abstract utility values are sometimes slightly different under different parameter choices, the utilities found by our system are seen to be around 90 percent of the optimized solution. (Sometimes, it does find the optimized solutions, as shown in Fig. 11.) In other words, the relative precision of the generated solutions remains stable at around 90 percent in our experiment setting.

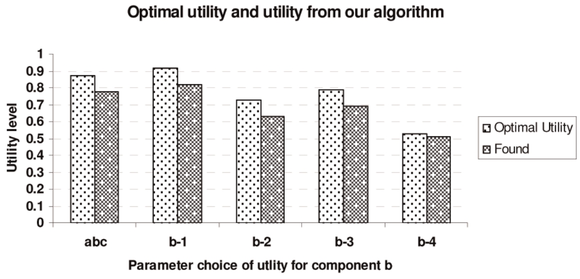


Fig. 12. Results against parameter choices of utility.

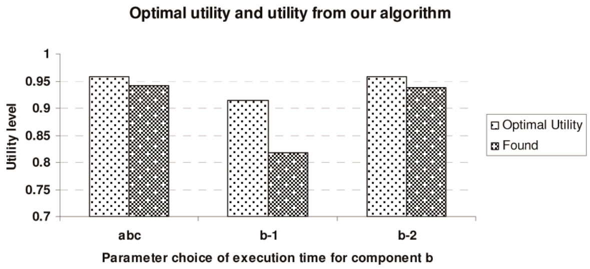


Fig. 13. Results with parameter choices of time.

5.3 Deployment In-the-Field: SSO

Our implementation of the solution is named the Service System Optimizer (SSO). It has been deployed in the field. In this section, we show results of this optimizer interacting with the demo mobile system that we have described in Section 2.2 and optimizing its overall utility through robust local search and transparent negotiation.

The SSO is deployed at an HP-Compaq desktop with Intel Pentium 4 CPU 3.00 GHz with 1.5 GB of RAM running Microsoft Windows XP Professional Version 2002 SP2.

The architecture of SSO is shown in Fig. 14. The SSO serves an intermediary layer between Web service QoSs and end users of service systems. An end user describes system utility, constraints, and preferences on functional requirements in the form of high-level objectives to the SSO, which assumes the responsibility of optimizing the performance of the service system for the end user. The transparent negotiation service is exported to the end users through QoS Negotiation APIs. End users use the APIs to specify system quality levels and the corresponding utilities to the SSO as a request to optimization. The three most important APIs are the following:

1. request(Component, QDimension, Utility), which takes a triplet consisting of a component, a quality dimension, and a utility. Using this API, the system end user submits an initial request on receiving a utility level of a particular component and over a particular quality dimension.
2. utilityFunction (Array), which takes an array of triplets of a component, a quality dimension, and a utility. Using this API, the system end user describes the utility functions to the SSO.
3. constraint (linearFunction(Components, QDimensions, Utility)), which takes a linear function over a number of components, quality dimensions, and a utility. Using this API, the end user specifies a utility constraint on component(s) over certain quality dimension(s).

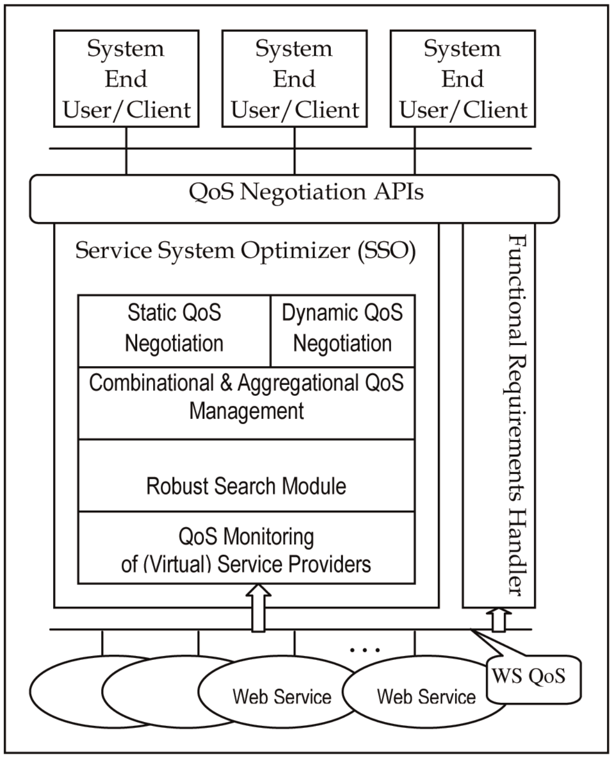


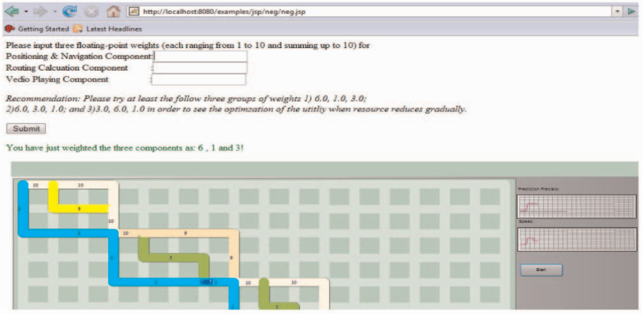
Fig. 14. Negotiation architecture.

QoS levels are monitored by the QoS monitoring module. The QoS monitoring model of (virtual) service providers supports all the other modules of SSO. Each virtual provider runs on top of a number of Web services whose QoS levels are explicitly represented and may vary dynamically. The virtual service provider is in charge of this collection of services and serves requests from a set of components by providing a single (virtual) service.

Two other important modules of the framework are Robust Search Module and Combinational and Aggregational QoS Management. They are responsible to customize the QoS allocation to the Web services used by the service system according to the rigidity of the particular application's QoS requirements. It converts the Web service QoSs to the system utility that can be perceived by the service system. The results will be used by the Static QoS Negotiation and Dynamic QoS Negotiation for application feasibility assessment, dynamic service pooling, and QoS reallocation. Dynamic QoS Negotiation handles newly arrived requests from dynamic components and QoS reallocation to existing components due to fluctuating Web service QoS provisions.

A collection of Web services, each providing a required function of the system and each having distinctive quality levels are installed at three other desktops that are connected with each other and connected with the SSO by LAN and WLAN. In our case, the service-based system is arranged to be colocated with the SSO. However, we do not exclude the possibility that the system be deployed at a site different than the SSO.

To illustrate the working of the SSO, we have developed a Web-based visualizer, which is a service-based application that is to be optimized by the framework. The visualizer is designed in such a way that we can demonstrate the



http://localhost:8080/examples/jsp/neg/neg.jsp

Getting Started Latest Headlines

Please input three floating-point weights (each ranging from 1 to 10 and summing up to 10) for
Positioning & Navigation Component:
Routing Calcuation Component :
Vedio Playing Component :

Recommendation: Please try at least the follow three groups of weights 1) 6.0, 1.0, 3.0; 2)6.0, 3.0, 1.0; and 3)3.0, 6.0, 1.0 in order to see the optimzation of the utitliy when resource reduces gradually.

Submit

You have just weighted the three components as: 6 , 1 and 3!

Fig. 15. Web-based visualizer application for optimization.

functioning of the framework through the optimization of this particular service-based system and visualizing the optimization results. Optimization results can be observed from the changing quality (and utility) on each component and each quality dimension of the system. We allow the user to specify the weights for each component. Then, the visulizer is able to show the fluctuations of the utility adjusted by the optimization framework when the services for remote navigation, control and routing, and video playing have to be negotiated and reselected. As shown in Fig. 15, on the top of the window, the user can specify the weights, and in the middle, there is a simulated map. The right area shows 1) a chart of the precision of the location calculation, which is affected by the quality of the global positioning service being used for navigation, 2) a chart of the past speed values of the vehicle, which is affected by the quality of choosing the best route, and 3) a chart showing the quality of video playing.

Figs. 16, 17, and 18 show the charts at a particular point in time for the prediction precision, the speed, and the buffering time. We can see that (due to a more stringent constraint on QoS) the QoS of the system has to degrade. Video playing first degrades due to its low impact on the overall system utility. This is followed by a degradation of route choosing. Positioning and navigation remains at the same utility for some time before degrading because it is considered the most important component in terms of the contribution to the overall system utility of the service system.

6 Related Work

The Web service model has emerged as a new application integration logic and B2B interaction framework. Web services, as a new class of programmable entities, are thus exposed to the issue of QoS. QoS for traditional Web applications has been studied within a layered architecture including transport, network, operating system, distributed system, and applications.

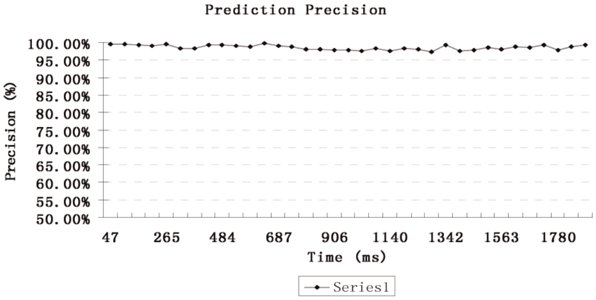


Fig. 16. Positioning and navigation quality.

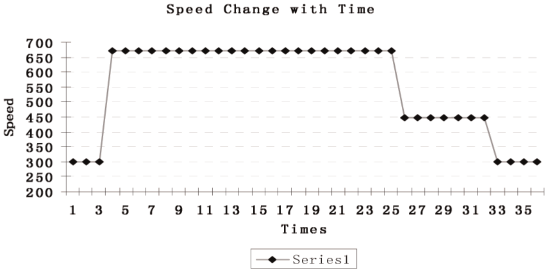


Fig. 17. Routing quality.

However, such studies are not sufficient for studying service systems. QoS of Web services has, in the past five to six years, been investigated as a separate topic in a number of research projects. A range of research issues regarding QoS provisions on Web services have been studied recently, which can be categorized as follows:

1. *Methods in processing QoS to be used for Web service selection and composition.*
2. *Models of Web service QoS characteristics and related ontologies and semantic languages.*

The majority of the development in the provision of quality of service support from Web services is to solve the issues of processing methods; and of models, ontologies, and semantic languages. Within the first category, service selection and composition are studied from the middleware perspective based on WS QoS [13], [14], [21], [30]. Researchers have also worked on static and runtime scheduling [25], [26], runtime QoS collection, monitoring and binding/rebinding [8], [15], transaction support [30], privacy and trust support [5], [7], and QoS negotiation and matching [20], [27], [6]. These works form the basis of working toward a general solution to optimizing service systems. A few research efforts have taken into consideration user satisfaction levels on Web service usage and end-to-end or application QoS management [29]. Within the second category, researchers have focused on defining models of QoS characteristics [3], [11], [31] and related ontologies and semantic languages [24], [16], and constraints of service attributes [27]. Due to space limit, the research efforts listed above are not meant to be exhaustive.

We have compared the research efforts of the two categories against a number of important elements that we have identified in approaching a general solution to service

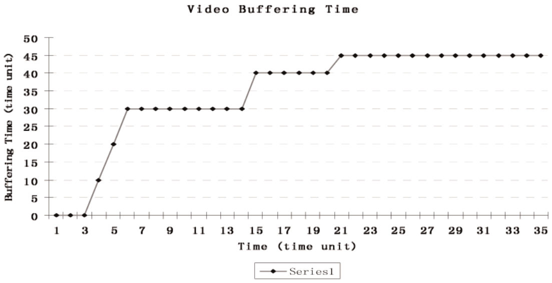


Fig. 18. Video playing quality.

TABLE 5
Comparison on Some Web Service QoS Research

Features	*Bench-marking Efforts*
Supporting Service Selection and Composition	[9][11][13] [15] [20] [21][23][25][26][27][28]
Aggregational	[11][15] [20]
System Utility & Robustness	
Negotiation	[19][21]
End-to-end	[12][23]

system optimization. The results are shown in Table 5. The following features are considered:

1. whether it supports service selection and service composition,
2. whether it counts on an aggregated QoS measure of the same QoS dimension over all components of a service system,
3. whether it reflects utility of system users and in a robust way,
4. whether it encompasses a transparent negotiation mechanism for maximizing the overall perceived system utility, and
5. whether it allows end-to-end QoS control.

If it is not obvious whether a particular feature is supported by an effort, it is not listed. Among the works that we surveyed, most of them support service selection and composition and many support aggregation. However, only a few support negotiation and end-to-end QoS. Currently, no work has taken into consideration of system utility and robustness.

We have approached the problem of optimizing service system by "application level QoS management" and we focus on the system utility perspective of an end application use case, or a system end user in the layered architecture of the Web service QoS stack. We claim that this is important because providing effective QoS control from the application level and in a system perspective has always influenced the usability of service systems. Web services have eased the job of building systems to achieve a user's business objective by translating such a problem to a problem of mapping required business activities to Web services. Due to this reason, the demand of providing controllability of QoS from end users or end applications has naturally escalated. Furthermore, with the presence of automatically discoverable, network accessible, heterogeneous, and dynamic Web service, it is not the case anymore that the level of service is always fixed—identically and uncontrollably for all end users and application use cases. Service systems need extra control on QoS that enables them to recognize and react to the environment.

We feel that our approach has largely relied on the research ideas originated from the area of Web service QoS. What we have tried in this research is to identify the core elements that are currently lacking for deriving a general solution to optimizing service systems, such as robustness, system orientation, and transparency and dynamisms. Then we propose techniques that implement these elements based on QoS management, which also constitute our solution to the problem.

7 Conclusion

In this paper, we advocate optimizing service systems and the key elements in a general solution to optimizing service systems. The approach we have taken is to propose an extra layer in the Web service QoS stack to serve as a bridge between the Web service QoS and the perceived system utility of the system end users. We have presented the distinguishing features of the application-level QoS management and the mechanisms used to facilitate searching for a heuristically good service system while achieving such features.

In this paper, we have only explored several core issues of service system optimization, given its vast and rich content. Furthermore, our solution is not exhaustive. We are trying to extend our solution to handle more complex scenarios, which exist in the real situation but are not considered in this paper. For example, we may want to factor the waiting time into our utility model for various reasons including a limited overall capacity of all service providers and the preference of end users to high quality but relatively busy servers.

Acknowledgments

The authors wish to thank the anonymous reviewers for their accurate and constructive comments, which have helped to improve the paper greatly.

References

[1] *TSC Taxonomy,* IEEE CS, http://www.computer.org/portal/pages/transactions/tsc/mc/tsc_taxonomy.html, 2008.

[2] L.J. Zhang, J. Zhang, and H. Cai, *Services Computing.* Springer and Tsinghua Univ. Press, July 2007.

[3] L.J. Zhang, "EIC Editorial: Introduction to the Knowledge Areas of Services Computing," *IEEE Trans. Services Computing,* vol. 1, no. 2, pp. 62-74, Apr.-June 2008.

[4] S. Cheng, C.K. Chang, and L.J. Zhang, "Modeling and Analysis of Performance Oriented and Revenue Based Admission Control Framework for Service Providers," *Proc. IEEE Services Computing Workshops (SCW '07),* pp. 9-16, 2007.

[5] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino, "Efficient Integration of Fine-Grained Access Control in Large-Scale Grid Services," *Proc. IEEE Int'l Conf. Services Computing (SCC '05),* pp. 77-84. 2005.

[6] Q. Liang, H. Lam, L. Narupiyakul, and P.C.K. Hung, "A Rule-Based Approach for Availability of Web Service," *Proc. IEEE Int'l Conf. Web Services (ICWS '08),* pp. 153-160, 2008.

[7] J. Kangasharju, T. Lindholm, and S. Tarkoma, "XML Security with Binary XML for Mobile Web Services," *Int'l J. Web Services Research,* vol. 5, no. 3, pp. 1-19, 2008.

[8] H. Ludwig, A. Dan, and R. Kearney, "Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreents," *Proc. Int'l Conf. Service-Oriented Computing (ICSOC '04),* pp. 65-74, 2004.

[9] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava, "A Service Creation Environment Based on End to End Composition of Web Services," *Proc. Conf. World Wide Web (WWW '05),* pp. 128-137, 2005.

[10] R.L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.* Wiley and Sons, 1976.

[11] J. Cardoso, A.P. Sheth, J.A. Miller, J. Arnold, and K.J. Kochut, "Modeling Quality of Service for Workflows and Web Service Processes," *Web Semantics J.: Science, Services Agents on the World Wide Web J.,* vol. 1, no. 3, pp. 281-308, 2004.

[12] P.C.K. Hung and H. Li, "Web Services Discovery Based on the Trade-Off Between Quality and Cost of Service: A Token-Based Approach," *ACM SIGecom Exchanges,* vol. 4, no. 2, pp. 21-31, 2003.

[13] D.A. Menasce, "Composing Web Services: A QoS View," *IEEE Internet Computing,* vol. 8, no. 6, pp. 88-90, Nov./Dec. 2004.

[14] P. Bellavista, A. Corradi, and S. Monti, "Integrating Web Services and Mobile Agent Systems," *Proc. First Int'l Workshop Services and Infrastructure for the Ubiquitous and Mobile Internet (ICDCSW '05),* pp. 283-290, 2005.

[15] G. Canfora, M.D. Penta, R. Esposito, F. Perfetto, and M.L. Villani, "Service Composition (re)Binding Driven by Application-Specific QoS. Service-Oriented Computing," *Proc. Fourth Int'l Conf. Service-Oriented Computing (ICSOC '06),* pp. 141-152, 2006.

[16] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.,* vol. 30, no. 5, pp. 311-327, May 2004.

[17] E. Giallonardo and E. Zimeo, "More Semantics in QoS Matching," *Proc. IEEE Int'l Conf. Service-Oriented Computing Applications (SOCA '07),* pp. 163-171, 2007.

[18] M. Colombo, E. Di Nitto, and M. Mauri, "SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined through Rules," *Proc. Int'l Conf. Service Oriented Computing (ICSOC '06),* pp. 191-202, 2006.

[19] E.D. Nitto, M.D. Penta, A. Gambi, G. Ripa, and M.L. Villani, "Negotiation of Service Level Agreements: An Architecture and a Search-Based Approach," *Proc. Int'l Conf. Service Oriented Computing (ICSOC '07),* pp. 295-306, 2007.

[20] G. Canfora, M.D. Penta, R. Esposito, and M.L. Villani, "A Framework for QoS-Aware Binding and Re-Binding of Composite Web Services," *J. Systems Software,* vol. 81, no. 10, pp. 1754-1769, Oct. 2008.

[21] M.C. Jaeger, G. Rojec-Goldmann, and G. Muhl, "QoS Aggregation in Web Service Compositions," *Proc. IEEE Int'l Conf. e-Technology, e-Commerce, and e-Service (EEE),* pp. 181-185, 2005.

[22] Q. Liang, J.Y. Chung, and S. Miller, "Modeling Semantics in Composite Web Service Requests by Utility Elicitation," *Knowledge and Information Systems,* vol. 13, no. 3, pp. 367-394, 2007.

[23] T. Yu and K.J. Lin, "Service Selection Algorithms for Web Services with End-to-End QoS Constraints," *J. Information Systems and e-Business Management,* vol. 3, no. 2, 2005.

[24] A. D'Ambrogio and P. Bocciarelli, "A Model-Driven Approach to Describe and Predict the Performance of Composite Services," *Proc. Sixth Int'l Workshop Software Performance,* pp. 78-89, 2007.

[25] J. Jin and K. Nahrstedt, "On Exploring Performance Optimizations in Web Service Composition," *Proc. Fifth ACM/IFIP/USENIX Int'l Conf. Middleware Table of Contents,* pp. 115-134, 2004.

[26] M. Bunruangses, W. Poompattanapong, P. Banyatnoparat, and B. Piyatamrong, "QoS Multi-Agent Applied for Grid Service Management," *Proc. 2004 Int'l Symp. Information Comm. Technologies,* pp. 74-79, 2004.

[27] Q. Liang and S. Su, "AND/OR Graph and Search Algorithm for Discovering Composite Web Services," *Int'l J. Web Services Research (IJWSR '05),* vol. 2, no. 4, pp. 46-64, 2005.

[28] Q. Liang and H. Lam, "Web Service Matching by Ontology Instance Categorization," *Proc. IEEE Int'l Conf. Services Computing,* pp. 202-209, 2008.

[29] Q. Liang, H.C. Lau, and X. Wu, "Robust Application Level QoS Management in Service Oriented Systems," *Proc. IEEE Int'l Conf. e-Business Eng.,* pp. 239-246, 2008.

[30] K. Hogg, P. Chilcott, M. Nolan, and B. Srinivasan, "An Evaluation of Web Services in the Design of a B2B Application," *Proc. 27th Australasian Conf. Computer Science,* pp. 331-340, 2004.

[31] D.F. García, J. García, J. Entrialgo, M. García, P. Valledor, R. García, and A.M. Campos, "A QoS Control Mechanism to Provide Service Differentiation and Overload Protection to Internet Scalable Servers," *IEEE Trans. Services Computing,* vol. 2, no. 1, pp. 3-16, Jan.-Mar. 2009.

Qianhui Liang received the PhD degree in computer engineering from the University of Florida. She is an assistant professor in the School of Information Systems at Singapore Management University. She is currently the program vice chair of the IEEE International Conference on Information Reuse and Integration (IRI '09), the publicity chair of the IEEE International Conference on Services Computing (SCC '09), the publicity chair of the Fifth World Congress on Services (SERVICES-II '09), the publicity chair of the IEEE 2009 International Conference on Cloud Computing (CLOUD-II '09), and the local organization cochair of the Fourth IEEE Asia-Pacific Services Computing Conference (APSCC '09). She has published in journals and conferences like *International Journal of Web Services Research* (JWSR), *Knowledge and Information Systems* (KAIS), International Conference on Services Computing (SCC), International Conference on Service-Oriented Computing (ICSOC), International Conference on Web Services (ICWS), International Conference on Information Reuse and Integration (IRI), International Conference on Enterprise Information Systems (ICEIS), etc. Her research interests are services computing (service composition, service modeling, and service ontology), cloud computing, and machine learning. She is a member of the IEEE and the ACM.

Xindong Wu received the PhD degree in artificial intelligence from the University of Edinburgh, Britain. He is currently a professor and the chair of the Computer Science Department at the University of Vermont. His research interests include data mining, knowledge-based systems, and Web information exploration. He has published more than 170 refereed papers in these areas in various journals and conferences, including the IEEE TKDE, TPAMI, ACM TOIS, DMKD, KAIS, IJCAI, AAAI, ICML, KDD, ICDM, and WWW, as well as 22 books and conference proceedings. His research has been supported by the US National Science Foundation (NSF), the US Department of Defense (DOD), the National Natural Science Foundation of China (NSFC), and the Chinese Academy of Sciences, as well as industrial companies including US West Advanced Technologies and Empact Solutions. He is the founder and current steering committee chair of the IEEE International Conference on Data Mining (ICDM), the founder and current editor-in-chief of the *Knowledge and Information Systems* (KAIS, by Springer), the founding chair (2002-2006) of the IEEE Computer Society Technical Committee on Intelligent Informatics (TCII), and a series editor of the Springer Book Series on *Advanced Information and Knowledge Processing* (AI&KP). He was the editor-in-chief of the *IEEE Transactions on Knowledge and Data Engineering* (TKDE, by the IEEE Computer Society) between 1 January 2005 and 31 December 2008, and served as the program committee chair for the 2003 IEEE International Conference on Data Mining (ICDM '03) and as program committee cochair for the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-07). He is the 2004 ACM SIGKDD Service Award winner, the 2006 IEEE ICDM Outstanding Service Award winner, and a 2005 chair professor in the Changjiang (or Yangtze River) Scholars Programme at the Hefei University of Technology appointed by the Ministry of Education of China. He has been an invited/keynote speaker at numerous international conferences including JCKBSE '08, NSF-NGDM '07, PAKDD-07, IEEE EDOC '06, IEEE ICTAI '04, IEEE/WIC/ACM WI '04/IAT '04, SEKE 2002, and PADD-97. He is a senior member of the IEEE.

Hoong Chuin Lau received the DEng degree in information engineering and science from Tokyo Institute of Technology, Japan. He is an associate professor at the School of Information Systems, Singapore Management University.