RC22501 (W0206-131) June 25, 2002
Computer Science

IBM Research Report

Service Level Driven Provisioing of Outsourced IT Systems

Dinesh Verma, Seraphin B. Calo
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

Service Level Driven Provisioning of OutSourced IT Systems

D. Verma and S. Calo
IBM T. J. Watson Research Center
PO Box 704
Yorktown Heights, NY 10598
{dverma, scalo}@us.ibm.com

Abstract:

The outsourcing of Information Technology (IT) operations is a common phenomenon in the industry, and has spurred a significant growth in the number of companies that provide hosting support for different IT operations. The current state of the art is for an IT service provider to negotiate a detailed system configuration with the enterprise customer based upon explicit resource requirements. The negotiation and provisioning of such system configurations are typically time-consuming, manual processes. We present an architecture and algorithms by which the service provider can negotiate the service provisioning in terms of performance metrics, rather than on the basis of resource configuration. Furthermore, we describe the different techniques that can be used to reduce the amount of time required for making an outsourced service operational.

1. Introduction

It is becoming common practice for many enterprises to outsource some of their IT operations to an ASP, or Application Service Provider [1]. Some examples of the IT operations outsourced in this manner include the operation of enterprise web-sites, mail servers, database servers, help-desks and intranets. IBM Global Services [2], as an example, operates the web-sites of many prominent companies all over the world.

The outsourcing of an enterprise IT operation is a process that is often unwieldy and time-consuming. The initial step in the outsourcing process is the selection of an ASP. Once the ASP is selected, the ASP and the enterprise enter into a lengthy negotiation to determine the right configuration for the system. The ASP then implements and tests the negotiated configuration of the system, a process that can take several weeks, or even months. Quite frequently, the enterprise discovers after the implementation that the configured system does not meets its desired level of performance. In these cases, the appropriate configuration has to be renegotiated and re-implemented, leading to additional delays.

One of the causes of frequent changes in system configuration is the lack of detailed IT expertise within the enterprise. In most cases, the lack of expertise and/or the expense associated with obtaining this expertise are the main motivating factors for outsourcing the IT infrastructure in the first place. This often leads to the enterprise negotiating a system configuration that is not

really suitable for its needs. The negotiation process can be reduced and simplified if the enterprise and ASP negotiate on the quality (performance, reliability and availability) of the service, rather than on the exact configuration. The task of mapping the service quality parameters to the right system configuration can then be delegated to the ASP. After the determination of the system configuration, the ASP needs to procure the different machines required for creating the system, interconnect the machines, install and configure the requisite software, and test the system for proper operation. This process is often time-consuming and laborious, and a leading cause of delays in implementation of outsourced IT systems.

In this paper, we present techniques that reduce the time required to outsource an IT service by having the ASP and enterprise negotiate on the terms of service quality rather than system configuration and by reducing the time required to implement a given system configuration. In Section 2 of the paper, we look at the typical way in which system configurations are specified, and the manner in which the Quality of Service parameters are stated. In Section 3, we discuss the algorithms that can allow the determination of a system configuration on the basis of the Quality of Service parameters. In Section 4, we discuss the different techniques that can be used to automate the provisioning of the system configuration determined as a result of the algorithms described in the previous section. Finally, we present our summary and conclusions.

2. Representing Service Levels and System Configuration

We define the system configuration of an IT service to be a description of the type of systems and software that are required in order to configure and provision the service. The system configuration will define the type of machines required for a service, the inter-connections between different machines, the software that needs to be installed on each of the systems, and the associated configuration of each software component. Once the system configuration is described in an unambiguous manner, the provider of an outsourced IT service can take steps to deploy the required set of machines and software specified in this configuration.

System configuration is usually done in a tiered manner. In this approach, the set of machines that provide a service are arranged in tiers. Each tier would typically consist of machines which are identical in configuration. Requests entering the system are distributed to one of the machines within the tier for processing. Depending on the application, the distribution may be stateless or stateful. In the stateless distribution, a request is sent to any of the machines in a tier. In the case of stateful distribution, a request from a new user is sent to any of the machines, but an affinity to the selected machine is maintained for subsequent requests from the same user. Tiered configurations are frequently used for web-based applications, where the first tier consists of caching proxies, the second tier consists of web-servers providing static content, the next tier consists of application servers providing dynamic content (e.g., execution of servlets, cgi-bin scripts or ASP pages), and the final tier often consists of a database server.

The definition of a system configuration can be captured by means of an information model such as that shown in Figure 1. The information model is captured as a UML diagram [3], where the overall system configuration consists of one or more tiers. Each tier has a set of machines, possibly including some load-balancers, is connected to another tier at the front-end, and connected to one or more tiers at the back-end. Each machine in the tier is characterized by its

operating system, its machine model number, and the set of applications that need to be installed on that machine.

The information model shown in Figure 1 would be the description of the system as installed and configured by an ASP. When the ASP actually implements the configuration, it would procure the right set of machines and software, determine the right configuration for the applications, and configure them appropriately.

The UML diagram in turn may be represented by means of an XML schema [4]. The XML schema that can be used to describe the system would consists of various tags representing the different objects, with multiple tags denoting the different attributes of the configuration specification. Each tag definition includes nested definitions of the attributes associated with the object, and objects with which it has a relationship in the UML diagram. A simple XML representation would consist of a topmost tag corresponding to the overall system, with embedded tags describing each of the tiers. The tiers in turn would include nested tags describing the machines they contain, and the machines include nested tags describing the applications that they are running.

With the configuration of the system defined using a tiered approach, as described above, the ASP is responsible for implementing the system in an appropriate manner. The performance, security and reliability of the system are then determined by the actual configuration instantiated within the (possibly shared) infrastructure maintained by the ASP in support of all its customers.

As an example, let us consider the architecture of a hypothetical web-hosting commercial site as shown in Figure 2. The site architecture consists of four tiers: a first tier consisting of a firewall; a next tier of caching web-proxies; a tier of web application servers; and, a tier of back-end databases. Let us assume that the firewall consists of a CheckPoint 3000 firewall, the caching and application tiers are each front-ended by a Cisco 5000 LoadBalancer, the caching web-proxies consist of six IBM Websphere EdgeServers running on Windows 2000, the web application server tier consists of IBM WebSphere CommerceSuite software running on four AIX servers, and the back-end database consists of a DB/2 database on an IBM-390 mainframe running z/OS. The configuration of the site is described pictorially in Figure 2. The system may further consist of a monitoring tier connected to the front-end machine which consists of two web-monitors that generate synthetic workload for the entire site for purposes of performance monitoring and reporting.

The above description of the system configuration can be captured by means of an XML document, as shown in Figure 3. The XML notation is shown for illustrative purposes only, and a more comprehensive configuration description can be obtained using the machine description specification of CIM V2.0 [5]. If an enterprise customer and a service provider obtain agreement on the configuration of the system, they can sign a contract and the service provider can follow established procedures in order to provision and install the required system.

As opposed to the configuration description, the service level objective of an application refers to the performance characteristics that are desired to be met by the application being provided. There are three main types of service level objectives that are commonly used within the industry:

- Capacity: This performance metric is characterized either by the size or throughput of the transactions/requests that the system can support while maintaining an acceptable delay.

- Latency: This performance metric is characterized by the delay incurred by a request when the load on the system is under a specific limit.

- Availability: This performance metric is characterized by the amount of time that the system is operational and able to perform at some desired level of latency.

The capacity and latency of the system depends on the application that is under consideration. As an example, outsourced mail hosting servers will be characterized in terms of the maximum number of users they can handle, and the latency experienced in handling a message of a fixed size. The units in terms of which the capacity or corresponding metrics are defined also need to be specified. Another class of information that must be supplied in defining SLOs consists of the platform and application dependencies that are required to support the customer. For example, if user data and/or application code needs to be provided by the customer (e.g., personnel directory entries, cgi bin programs), specific database or operating systems may have to be used. A UML diagram representing an SLO specification is shown in Figure 4. Formal representations of these performance objectives in specific contexts such as web-services [6] or networking services [7] can be found in the literature.

As opposed to the system configuration, the capacity, latency and availability of the system are much easier to describe and negotiate by business people who may not be well-versed in the technical details required to determine an appropriate configuration. Negotiating and offering services on the basis of the SLOs rather than the specific configuration can reduce the time involved in both the negotiation and system configuration steps of the outsourcing process. This can ease the burden of negotiating the terms of the agreement and expedite the provisioning decisions, by allowing greater leeway on the part of the ASP in determining how best to satisfy the requirements of the customer enterprise. However, the service provider would need to ensure that the system configuration implemented can meet the performance objectives specified.

As an illustrative example, the performance objectives of the system described in Figure 2 can be represented by means of an XML specification as shown in Figure 5. The XML tags used are mapped using the same attribute names as those in the object specifications. As would be obvious from the comparison of the XML descriptions shown in Figures 3 and 5, the specification of the performance objectives is much simpler than the detailed system configuration.

In the next section, we look at the typical processes used in defining a contract for outsourcing of IT services at the system configuration level, and examine some of the causes of delay in the process. We then consider the same set of processes when outsourcing IT services at the SLO level, and show how many of the basic causes are eliminated.

3. The Provisioning Process

The provisioning process for outsourcing an IT service at the configuration level is shown in Figure 6. We assume that the customer and the ASP both have separate business teams and technical teams. In most cases, the technical team at the customer is much smaller, since the desire of customers not to have to develop technical skills across broad areas of information technology is the primary reason for the outsourcing of IT services in the first place. We assume

that the customer and the service provider are the only entities in the process, and that the customer has selected the service provider.

While there are many possible ways for the outsourcing process to function, we assume that it begins with the technical team of the customer designing a system configuration that they feel is satisfactory for their needs. Since the technical team at the customer may be small or nonexistent, this step may take considerably longer than if the technical team at the service provider did the same. In some cases, the customer may outsource the design step to another consulting company (perhaps even to the consulting division of the service provider). After the design is completed, the technical team hands over the specification to the business team of the customer. The customer business team hands over the design to the business team of the service provider to get the terms and conditions of the outsourced service. The business team at the service provider goes through a feasibility checking step with the technical team at the service provider, to determine if they can support the required system configuration. This step may require changes to the supported system configuration, which needs to be renegotiated with the technical team at the customer. The renegotiation step may again be time-consuming if the services of a consultant needs to be relied upon again.

After the final system configuration is agreed upon, the service provider's technical team needs to procure the desired machines and then to install and configure the required software. Since the technical team at the service provider may need to acquire new hardware, and expertise in new software that is desired by the customer, this step may again be time-consuming. It also increases the costs incurred by the service provider, which will be reflected in a higher price for the outsourced service to the customer.

On the other hand, when the customer and service provider negotiate on performance objectives rather than the system configuration, the process could be streamlined to be similar to the one shown in Figure 7. The customer's and SP's business teams negotiate on the performance objectives of the system to be outsourced, and the technical team of the customer may provide any constraints, e.g. requirements of a specific platform they consider more secure or on which they may have a legacy application operational. The SP's technical team maps the performance objectives to an appropriate system configuration, and the business and SP teams negotiate a suitable price. The service provider's team then procures and installs the required hardware/software and configures the software appropriately.

The service provider can design the system configuration in a manner that is easiest and most cost-efficient with regard to its own infrastructure and sets of resources, subject to the technical constraints specified by the customer. The business teams from both the customer and the service provider can negotiate the performance objectives without a detailed technical understanding of the underlying system, and the negotiation as well as the provisioning process can be streamlined and make simpler.

The hard problem in the entire process lies in the mapping of system performance objectives to a system configuration that can be outsourced and managed in an efficient manner by the service provider. In the next section, we look at different algorithms that the service provider's technical team may use to map performance objectives to system configurations.

4. Translating SLOs to Configurations

While there are multiple capacity planning and design tools [8] [9] that can enable one to design a system to support a desired service level, such tools usually do not include constraints such as a service provider's ease in obtaining and procuring specific hardware, nor do they include any aspects that reflect a service provider's existing familiarity with a given software application. The delays in obtaining the required hardware for a machine, or the ease in installing existing software are soft measures that can not be easily captured in configuration planning tools, but they can have a tremendous impact on the ability of the service provider to live up to its expectations. Such tools also typically deal with parameters such as response time in a monolithic way, and do not consider that an end-to-end SLO should be expressed as the aggregation of SLOs for the different resources that comprise the system. As a result, vanilla capacity design tools are not adequate to translate SLOs into system configurations that can be easily supported by a service provider. Also, an effective way to capture the soft requirements of system familiarity, and delays are difficult to specify in a precise quantitative manner required to be used meaningfully in a capacity planning tool. At the same time, this is precisely the problem that needs to be solved in order to map SLOs into system configurations in a practical manner.

In the following subsections, we look at a number of approaches that can be used to provide this mapping, and compare the relative merits of each of them.

4.1 The Fixed Menu Solution

One effective way to ensure that SLOs can be translated into effective configurations is for the service provider to only support a limited set of system configurations. Each of these would consist of a prescribed set of hardware, software, and concomitant configuration parameters. A given configuration would fully describe the number of tiers and the composition of each of the tiers that the service provider could support. The service provider would typically have tested and verified the performance of each of these configurations. Thus, the throughput, capacity and other limits of each of the configurations would be fully known. When the technical team from the service provider needs to develop a system configuration that is sufficient to meet a given set of performance objectives, they would simply choose from the menu the configuration which is adequate to meet the desired performance constraints. Since each of the configurations is tested beforehand, it is a relatively straightforward task for the service provider to repeat the configuration when a new customer agrees to one of the predefined systems.

As an example, the service provider may have tested out 12 configurations of the commerce site following the tiered structured shown in Figure 2. The 12 sample configurations and their performance characteristics are shown in Figure 8. The configurations offer the possibility of 0, 3 or 5 caching proxies at the caching tier, and 1 or 3 application servers. They further offer the choice of using either Oracle or DB/2 as the database choices. The platforms and operating environments of each of the tiers otherwise is fixed. This results in 12 overall configurations, and the throughputs and latencies of the resulting systems are as shown in Figure 12. The service provider would typically obtain the overall system throughput and latency numbers by measurements, e.g. by running performance benchmarks against each configuration.

When a customer request is received with desired throughput and latency objectives, the service provider chooses the configuration that can provide the customer with the desired performance. If the customer has specified a preference for a specific type of database, then only the configurations corresponding to that particular choice (i.e., either Oracle or DB2) are considered.

The fixed menu solution has the advantage that all of the supported configurations are known to work, and it is relatively easy to choose from amongst them. Thus, there is no risk that the desired configuration may not work. On the other hand, the service provider needs to ensure that it has tested enough configurations so that the likelihood of meeting most of its customer's requirements in an efficient manner is high. The fixed menu has the disadvantage that it does not allow any changes to the configuration to meet the specific needs of particular customers. As an example, the latency needs of a customer may be met by having only 4 web-caching servers instead of 3 or 5, but the fixed menu solution does not permit that according to the constraints of Figure 8.

4.2 Perturbation Analysis

Perturbation analysis allows the service provider to asses the impact of changing some of the aspects of the system configuration based on the overall service objectives. As an example, the service provider may assess by means of measurements or interpolation among known configurations that increasing the number of caching web-proxies by 1 decreases the response-time by 5% on the average, and increases throughput by 4%. The number of parameters that can be meaningfully varied and the impact of each of the parameters on the configuration of the system can be ascertained by the service provider.

An example of the perturbation analysis approach corresponding to the configurations indicated in Figure 8 is shown in Figure 9. If we consider only the configurations with DB/2 databases, we can interpolate the impact of a relative change in configuration for each of the allowable base system configurations. The interpolation is done by estimating the impact of adding one more web-proxy in each of the configurations. The impact of the perturbation is effectively an estimate of the first derivative of the SLO metric with respect to the parameter being changed.

By using perturbation analysis, one can estimate the impact of having a slightly different configuration on SLO parameters such as user latency and maximum throughput. This allows slight adjustments to the known configurations of the system that work well. There is a slight risk that the configuration obtained as a result of the perturbation may not work as projected, but the risk is reasonably small. Perturbation analysis allows the service provider to support configurations that can be customized to a certain extent to meet the desired performance objectives.

4.3 Tiered Calculus

The amount of different configurations that can be created by means of perturbation analysis are relatively limited. A more flexible configuration of the overall system can be obtained by leveraging the tiered nature of the service provider, and by combining the SLO metrics that are available for each of the tiers that are available. The tiered calculus can be viewed as an adaptation of the network calculus approach [10] which has been applied to provide service guarantees in communication networks [11].

Each resource in the system configuration can be considered a component in the overall system configuration. The SLO objectives (latency, availability etc.) are a function of the load that is

offered to that system component. While the exact performance varies significantly depending on the actual load on the system, one can characterize the effect of load variation by a bounding curve that provides an overall bound on the performance of the system. The bounding curve that we use for the purpose of a tiered calculus is shown in Figure 10. The component has the characteristic that its performance is less than the threshold P as long as the load on the component is less than T. As an example, the load on a web-server can be measured in terms of connections per second and the performance can be measured in terms of the overall response time. While the exact variation of performance is a complex function of the load, determining a bounding threshold (or selecting reasonable values of P and T) is a much simpler task. As in Figure 10, the bounding curve (shown in solid lines) for the system is shown for three different actual performance curves. The bounding curve can be determined even if the performance curve of the system is not known precisely.

As an example, assume an Application Server has a minimum response time of 30 ms in executing EJB transactions. The response time increases relatively slowly as long as the load on the system does not exceed 500 requests per second, and then can increase as a function of the actual load. If the load at 500 requests per second is 35 ms, one can determine that the (T,P) value for the application server is (500req/s,35 ms). The exact performance curve may be quite variable, and much more difficult to ascertain.

In any system configuration, components are either connected in series or in parallel. Let us consider two components that are connected in series, forming a larger overall component. Let them be characterized by the tuples (T_1, P_1) and (T_2, P_2) respectively. Consider the performance of the composed component when the load on the overall system is t. The load on the individual components will be a multiple of the load on the overall system. Let the multipliers a_1 and a_2 be such that the load on the two subsystems are a_1t and a_2t respectively. The first component provides a performance bound of P_1 as long as a_1t is less than T_1, and the second component provides a performance bound of P_2 as long as a_2t is less than T_2. Therefore, the bounding threshold for the overall system would be the minimum of T_1/a_1 and T_2/a_2. The performance of the composed component would be obtained by combining the performance metrics of the two subcomponents, e.g., the response time would be the sum of the individual response times, and the availability would be the minimum of the subcomponent availabilities. Thus, for delays we can obtain the relationship that the bounding curve of the overall component would be given by:

$$(T,P) = (\min(T_1/a_1, T_2/a_2), P_1+P_2);$$

Similarly, when the components are composed together in parallel, they experience loads which are given by the fractions a_1 and a_2 respectively of the overall system load. It is relatively simple to show that the bounding curve for the composed component would be given by:

$$(T,P) = (\min(T_1/a_1, T_2/a_2), \max(P_1,P_2));$$

The generalization to more than two components in series and parallel is obvious, with N components connected in series having the overall performance bounds (for additive metrics like delay):

$$(T,P) = (\min(Ti/a_i), S_N),$$

where S_N denotes the sum of the N individual performance bounds, and the minimum is taken over the N subcomponents.

Similarly, for components in parallel:

$$(T,P) = (\min(T_i/a_i), \max(P_i)),$$

where again, the optima are taken over the N subcomponents.

For a tier of machines, assuming that all machines are homogenous and the traffic is evenly distributed, the overall metric would be given by:

$$(T,P) = (T*n, P),$$

where n is the number of machines in the tier, and T and P are the respective throughput and performance bounds for each of the machines. The performance bound for the overall system can then be determined by putting all the tiers in series.

The above calculus of bounding curves allows one to obtain bounds on system SLOs if one knows the bounds on each of the individual components in the system. From a given system SLO, and a set of feasible components, one can then enumerate the different possible configurations that would meet the desired composite SLO. This can then yield an appropriate system configuration that would provide the desired performance objectives at the lowest cost to the service provider. Once the overall system configuration is determined, the standard provisioning procedures can be used by the service provider to deliver upon its desired goals.

The tiered calculus provides for greater flexibility in designing the configuration of the overall system. However, since it determines a large number of untested configurations, the risk that an unexpected threshold/problem in overall system configuration is encountered is larger.

4.4 Optimization Techniques

Optimization techniques could be applied directly if a reasonable objective function could be formulated. Given a set of base resources and their individual performance models, the aggregate performance that attains some minimum cost subject to resource constraints constitutes a well formed optimization problem. There are analytic, heuristic, and search algorithms for solving such problems. Their appropriateness and effectiveness depend upon the characteristics of the objective function, and the characteristics of the performance functions of the individual resources themselves.

For a tiered configuration, with homogeneous resources in each tier, the problem reduces to one in which the number of resources provided at each tier are the variables, and the objective function to be minimized is a measure of the cost of providing this number of resources. The constraints are then determined by the overall performance objectives, i.e., that the series combination of the tiers, consisting of collections of parallel resources, attain more than a certain throughput at less than a certain delay. Additional constraints can be placed on the individual resources to keep them performing within a reasonable range, thus simplifying their respective delay-throughput curves (e.g., keeping them in a relatively linear or piece-wise linear portion of their operating characteristics).

Optimization techniques require the use of a performance model, and the development of such a model entails the making of simplifying assumptions in order to maintain tractability. While the approach is more elegant, it leads to an increased risk of developing a system configuration that may not meet the desired performance objectives.

A qualitative comparison of all the methods discussed above is shown in Figure 12. The horizontal axis shows the risk associated with deploying a solution found using one of the above four approaches. The fixed menu solution has virtually no risk, while the optimization approach is associated with the maximum risk. The vertical axis shows a cost metric. One curve shows the cost of wasted resources, i.e., the difference in cost between the system configuration determined by the solution and the ideal (lowest-cost) system configuration that would be required to meet the customer's expectations. Another curve on the same graph shows the cost associated with the latency incurred in deploying a solution obtained by each of the approaches. In Fixed Menu Solutions, the components of the configuration are tested, and in some cases procured in advance. Thus, it has the lowest development delay. The optimization approach has the largest delay since it may require that new configurations be built and tested from first principles. The other approaches lie in-between.

For any given service provider, the choice between the various approaches would be determined by the tradeoff between the two costs of wasted resources and latencies in deploying the solution. Depending upon the relative importance that an ASP places on these two costs, it can select the appropriate solution approach to map service level objectives to system configuration.

5. Summary and Conclusions

We have presented the problem of determining the resources and the system configuration needed to attain given performance levels for an outsourced IT service. A number of techniques for mapping from a set of performance requirements to a set of computing resources and a system configuration have been proposed. These are in support of the overall goals of allowing IT outsourcing to be specified at a higher level of abstraction, and supporting service provisioning in as automated a manner as possible. The desired requirements flow is shown in Figure 11, where business level requirements drive the specification and negotiation of Service Level Agreements which then determine global resource requirements. These are subsequently mapped onto specific configurations, as discussed in this paper.

The simplest technique for translating SLOs into system configurations is the fixed menu approach presented in section 4.1 above. Variations on this methodology are being employed today, either automatically or manually. Reference Models capturing performance information from experimental system configurations or previous customer engagements are used to guide the design of new service instances. As mentioned, however, this approach is somewhat inflexible, and provides little guidance if the new service requirements are not close to the performance objectives attained by any of the Reference Models.

In this regard, the perturbation analysis in 4.2 provides a mechanism for refining the fixed set of Reference Models in order to broaden the number of alternatives. The accuracy of such an approach would tend to decrease as the projected configuration became increasingly different from the members of the set of Reference Models, so that this methodology is also somewhat limited.

The tiered calculus of section 4.3 provides a more general and constructive manner of estimating the required configuration. Bounding curves for the global resource requirements can be obtained from bounding curves for the performance of each of the individual resources. The issue, of course, is how accurately these bounds can be obtained, and how badly the aggregate bounds

degrade as the interconnections of components become more extensive. The method is also more effective at composition than decomposition. Given an end-to-end response time goal, there may be many feasible sets of component goals that can achieve it. Enumeration and search techniques could be used to generate potential solutions that could then be evaluated for effectiveness, but the computational cost of such a procedure needs to be assessed, particularly for complex interconnections of many heterogeneous resources.

Similarly, formulating the mapping as an optimization problem also leads to issues of computational costs. In order to make the optimization more efficient, one might attempt to simplify the resource performance functions (e.g., by piece wise linear approximation), but the accuracy of the resulting solution might then not warrant even the reduced computational effort.

At this point, we tend to favor simpler and combined approaches. If the new service requirements are close to the performance objectives met by any of the Reference Models, then the known configuration is the best alternative. If not, and there is sufficient data to apply perturbation analysis, then the generation of a feasible configuration with this methodology should be attempted, but only within a reasonable extension of the ranges of parameter values (e.g., plus or minus ten percent). Failing this, the tiered calculus can be used to ascertain feasible configurations. The known Reference Model configurations might then be used to guide the enumeration and search phases, since the closer a feasible solution is to one of these (in structure if not in number of resources), the greater the confidence in its actual performance. It is not clear when the optimization approach would yield better feasible solutions, and that is one aspect that needs to be assessed.

The overall goal is not to generate exact configurations, but reasonable ones that meet the service requirements. It is envisioned that following the service provisioning process, there will be a tuning or "burn-in" phase during which the performance of the deployed configuration will be measured without incurring penalties for not meeting SLOs. During this phase, the resource allocations can then be readjusted so that all the SLOs will be met when the service goes operational. This phase is particularly important in shared infrastructure environments like eUtilities, in which many customers are supported and resources can be dynamically assigned to virtual IT configurations. The impact of a new service instance in such a system would need to be determined, since it could affect existing SLOs that were previously being met. The service provider needs to insure that all customers are receiving their contracted levels of service, or else algorithmically determine which customers get degraded service (e.g., by minimizing potential penalties).

This research is part of a broader effort to advance technologies for automated provisioning. The basic approaches have been worked through against a set of typically scenarios, and an overall system architecture has been developed. A prototype incorporating the mapping from higher level service requirements to resource configuration, and the automated deployment and tuning of outsourced Web-based services is under development.

References

[1] Tim Weiss, Service Provider Trends, Cisco World Magazine, August 2000, URL http://www.ciscoworldmagazine.com/webpapers/2000/08_telechoice.shtml.
[2] IBM Global Services, http://www-1.ibm.com/services/.
[3] M. Page-Jones, “Fundamentals of Object Oriented Design in UML”, Addison-Wesley, ISBN 020169946X, 2000.
[4]] XML Schema, Description available at http://www.w3.org/XML/Schema
[5] Common Information Model (**CIM**) Version 2.2. Specification, *Distributed Management Task Force*, June 1999.
[6] A. Keller, et al, "Managing Dynamic Services: A Contract Based Approach to a Conceptual Architecture," Proceedings of NOMS 2002, Florence, Italy, April 15-19, 2002.
[7] D. Verma, Supporting Service Level Agreements in IP Networks, Mcmillan Technical Publishing, 1999.
[8] Harry K. Jackson and Normand L. Frigon, “Fulfilling Customer Needs : A Practical Guide to Capacity Management” John Wiley & Sons, ISBN 0471180920, May 1998
[9] S. F. Lam and K. H. Chan, “Computer Capacity Planning: Theory & Practice”, Academic Press, Incorporated, ISBN: 0124344305, March 1987.
[10] R. Cruz. A calculus for network delay, part II: network analysis. IEEE Trans. Information Theory, 37:132-- 141, 1991.
[11] R. Agrawal, R.L. Cruz, C.M. Okino, R. Rajan, "A Framework for Adaptive Service Guarantees," Proceedings of Allerton Conf, Monticello, 1998. http://citeseer.nj.nec.com/agrawal98framework.html

Figures

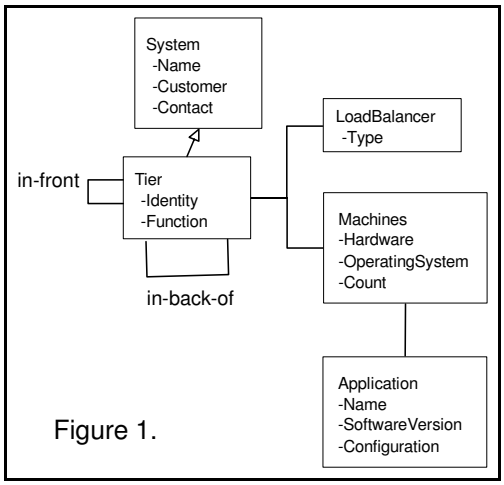


Figure 1.

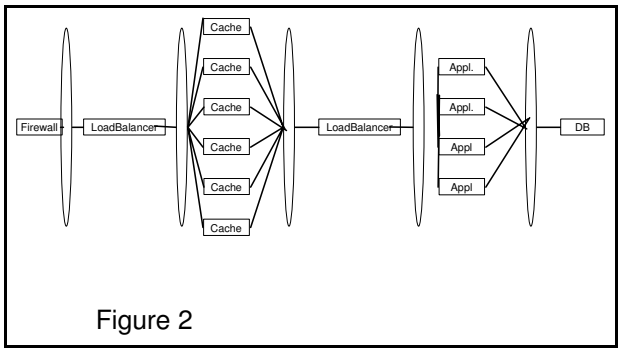
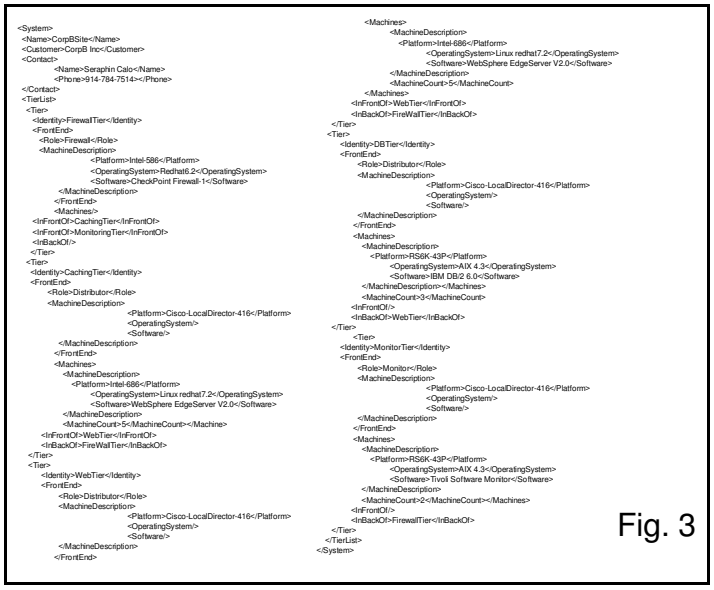


Figure 2



```
<System>
 <Name>CorpBSite</Name>
 <Customer>CorpB Inc</Customer>
 <Contact>
        <Name>Seraphin Calo</Name>
        <Phone>914-784-7514></Phone>
 </Contact>
 <TierList>
  <Tier>
   <Identity>FirewallTier</Identity>
   <FrontEnd>
    <Role>Firewall</Role>
    <MachineDescription>
                <Platform>Intel-586</Platform>
                <OperatingSystem>Redhat6.2</OperatingSystem>
                <Software>CheckPoint Firewall-1</Software>
        </MachineDescription>
       </FrontEnd>
       <Machines/>
   <InFrontOf>CachingTier</InFrontOf>
   <InFrontOf>MonitoringTier</InFrontOf>
   <InBackOf/>
  </Tier>
 <Tier>
   <Identity>CachingTier</Identity>
   <FrontEnd>
       <Role>Distributor</Role>
       <MachineDescription>
                <Platform>Cisco-LocalDirector-416</Platform>
                <OperatingSystem/>
                <Software/>
        </MachineDescription>
       </FrontEnd>
       <Machines>
         <MachineDescription>
          <Platform>Intel-686</Platform>
                <OperatingSystem>Linux redhat7.2</OperatingSystem>
                <Software>WebSphere EdgeServer V2.0</Software>
        </MachineDescription>
        <MachineCount>5</MachineCount></Machine>
   <InFrontOf>WebTier</InFrontOf>
   <InBackOf>FireWallTier</InBackOf>
 </Tier>
 <Tier>
   <Identity>WebTier</Identity>
   <FrontEnd>
       <Role>Distributor</Role>
       <MachineDescription>
                <Platform>Cisco-LocalDirector-416</Platform>
                <OperatingSystem/>
                <Software/>
        </MachineDescription>
       </FrontEnd>
       <Machines>
         <MachineDescription>
          <Platform>Intel-686</Platform>
                <OperatingSystem>Linux redhat7.2</OperatingSystem>
                <Software>WebSphere EdgeServer V2.0</Software>
         </MachineDescription>
         <MachineCount>5</MachineCount>
       </Machines>
   <InFrontOf>WebTier</InFrontOf>
   <InBackOf>FireWallTier</InBackOf>
 </Tier>
 <Tier>
   <Identity>DBTier</Identity>
   <FrontEnd>
       <Role>Distributor</Role>
       <MachineDescription>
                <Platform>Cisco-LocalDirector-416</Platform>
                <OperatingSystem/>
                <Software/>
        </MachineDescription>
       </FrontEnd>
       <Machines>
         <MachineDescription>
          <Platform>RS6K-43P</Platform>
                <OperatingSystem>AIX 4.3</OperatingSystem>
                <Software>IBM DB2 6.0</Software>
         </MachineDescription></Machines>
         <MachineCount>3</MachineCount>
   <InFrontOf/>
   <InBackOf>WebTier</InBackOf>
 </Tier>
    <Tier>
   <Identity>MonitorTier</Identity>
   <FrontEnd>
       <Role>Monitor</Role>
       <MachineDescription>
                <Platform>Cisco-LocalDirector-416</Platform>
                <OperatingSystem/>
                <Software/>
        </MachineDescription>
       </FrontEnd>
       <Machines>
         <MachineDescription>
          <Platform>RS6K-43P</Platform>
                <OperatingSystem>AIX 4.3</OperatingSystem>
                <Software>Tivoli Software Monitor</Software>
         </MachineDescription>
         <MachineCount>2</MachineCount></Machines>
   <InFrontOf/>
   <InBackOf>FirewallTier</InBackOf>
 </Tier>
 </TierList>
</System>
```

Fig. 3

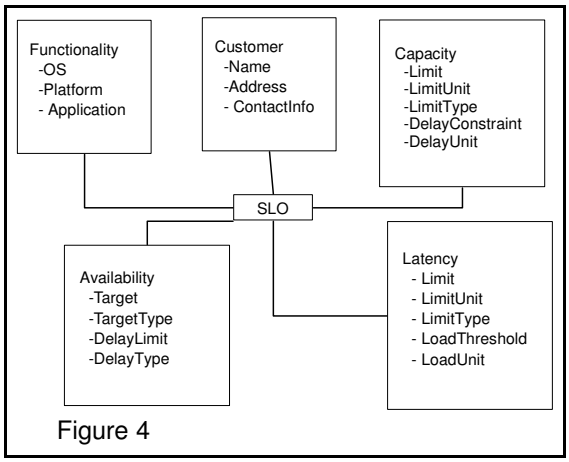


Figure 4

```
<SLODescription>
    <Functionality>
        <Platform> <AIX-43P> </Platform>
        <Middleware>WebSphere V4.0</Middleware>
        <Function>eCommerce Site </Function>
    </Functionality>
    <Customer>
        <Name>CorpB Inc</Name>
        <Contact>
            <Name>Seraphin Calo</Name>
            <Phone>914-784-7514></Phone>
        </Contact>
  </Customer>
  <Availability>
        <Target>0.999</Target>
        <TargetType>Uptime</TargetType>
        <DelayLimit>100ms</DelayLimit>
        <DelayType>Response Time</DelayType>
  </Avaialability>
  <Capacity>
        <Limit>1000</Limit>
        <LimitUnit>tranascations-per-second</LimitUnit>
        <LimitType>0.99 Percentile</LimitType>
        <DelayLimit>100ms</DelayLimit>
        <DelayType>Response Time</DelayType>
  </Capacity>
<SLODescription>
```

Figure 5

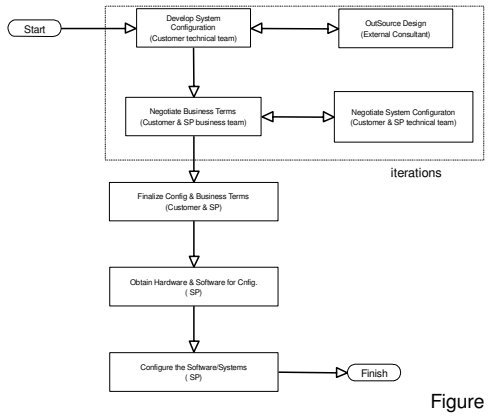


Figure 6.

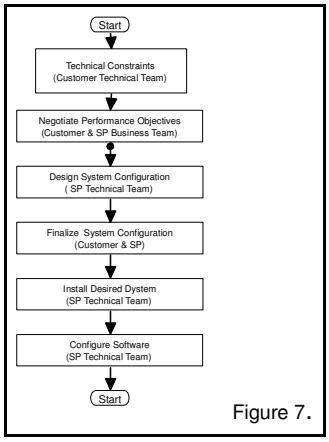


Figure 7.

Configur No.	1	2	3	4	5	6	7	8	9	10	11	12
Caching Tier (Number of Caching Proxies)	0	0	0	0	3	3	3	3	5	5	5	5
Web Tier (Number of web application-servers)	1	1	3	3	1	1	3	3	1	1	3	3
Database (Type of database)	DB/2	Orcale	DB/2	Orcale	DB/2	Orcale	DB/2	Oracle	DB/2	Orcale	DB/2	Oracle
Capacity (Maximum connections/second)	1000	1200	2000	2000	1800	1600	4500	3800	3200	3600	5000	4500
Latency (Average time for transactions)	250	300	250	300	120	100	100	85	30	30	25	20

Figure 8

Configuration Number	1	1+	3	3+	5	5+	7	7+	9	9+	11	11+
Caching Tier (Number of Caching Proxies)	0	1	0	1	3	4	3	4	5	6	5	6
Web Tier (Number of web application-servers)	1	1	3	3	1	1	3	3	1	1	3	3
Database (Type of database)	DB/2	DB/2	DB/2	DB/2	DB/2	DB/2	DB/2	DB/2	DB/2	DB/2	DB/2	DB/2
Capacity (Maximum connections/second)	1000	1040	2000	2080	1800	1870	4500	4680	3200	3330	5000	
Latency (Average time for transactions)	250	240	250	240	120	115	100	95	30	28	25	24

Figure 9

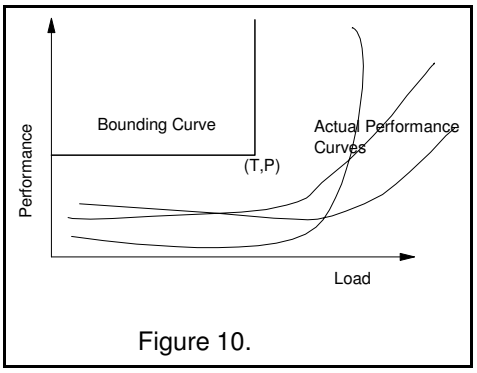


Figure 10.

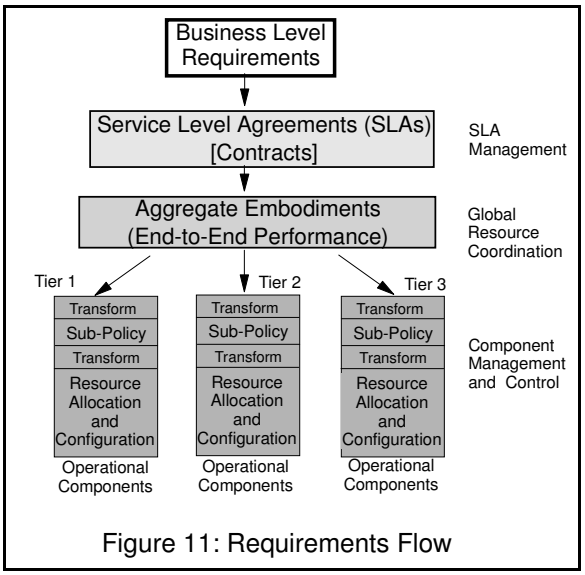


Figure 11: Requirements Flow

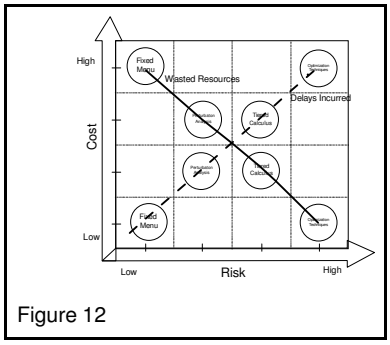


Figure 12