RC22961 (W0311-012) November 3, 2003
Computer Science

IBM Research Report

AMBIENCE: Automatic Model Building using IferENCE

Zhen Liu, Cathy H. Xia, Petar Momcilovic, Li Zhang
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
{zhen1, cathyx, petar, zhangli}@us.ibm.com

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

This page intentionally left blank.

Abstract

Performance Modeling has become increasingly important in the design, engineering and optimization of Information Technology (IT) infrastructures and applications. A modeling approach is particularly efficient in providing architects and engineers with qualitative and quantitative insights about the system under consideration. However, the modeling work itself is time consuming and requires a good knowledge of not only the system, but also modeling techniques. In this paper, we present a tool, AMBIENCE, aimed at automating the process of performance modeling and optimization of IT infrastructure and applications. AMBIENCE incorporates an innovative approach that integrates high-level queueing network models with advanced inference techniques. This approach uses system measurement data and end-to-end reponse time measurement data as input, and infers the queueing parameters of the system. In addition, AMBIENCE features a suite of powerful performance engineering functions, including predicting performance, optimizing existing IT infrastructure, and suggesting cost-effective architecture designs through deployment and operations. AMBIENCE has been successfully used to analyze performance, quality of service and optimal design for a range of e-business infrastructures.

1 Introduction

With the rapid advances in Internet technology, e-commerce is becoming a mature business strategy. The concept of Quality of Service (QoS) is working its way to the front-line of e-business commitments and requirements as it plays an important role in Internet applications, services and price negotiations. In the growing competitive marketplace, Information Technology (IT) administrators and planners are under constant pressure to have definitive answers for the following questions: How many users can the system support with the current infrastructure? What level of service quality is being delivered for each service? How fast can the site architecture be scaled up, or down? What components should be upgraded? Is there enough capacity available to support the expected business opportunities and future growth? To solve these problems, e-businesses have an urgent need to have a formal process for planning and budgeting, for performance prediction, and for service-level agreement provisioning. Such a process requires a fundamental understanding of key issues such as capacity, scalability, availability, reliability of an e-business infrastructure. Our goal in this study is to investigate these important issues so as to enable IT managers to quantify, control and plan, efficiently and proactively, the utilization of the available capacity in their e-business infrastructures.

The characterization of an e-business infrastructure is quite complex, given the variety of system architectures, numerous emerging applications with different functions, and the vast diversity in user behaviors. Therefore it is a challenging task to assess the site's capability of delivering end-to-end performance assurance across the entire enterprise, inclusive of all IT infrastructure components and applications. A commonly used approach is to directly measure the performance of the site, either in the production system using real workloads, or on a test system using synthetic workloads. Measuring performance directly on the production system is risky since

Having a testing site is a very expensive practice. The efforts are enormous to simply duplicate the production system and make sure all the interconnected software and hardware components are functioning properly. In addition, since the workload space is continuous, it is simply impossible to test across all different load combinations.

An alternative approach to assess the performance is through performance modeling. Performance modeling has become increasingly important in the design, engineering and optimization of Information Technology (IT) infrastructures and applications. A modeling approach is particularly efficient in providing architects and engineers with qualitative and quantitative insights about the system under consideration.

There are two streams of traditional performance modeling methods used in the literature. One method is based on inference, which includes neural networks [2], learning theory or statistical inference techniques [4]. Such methods are commonly based on linear models, hence they are weak in capturing non-linear system behaviors. In addition, as for any statistics-based approach, there are issues with being able to collect data that reflects the whole range of operation of the system.

Performance modeling based on queueing networks, however, would resolve the above problem easily. The primary advantage of a queueing model is that capture the fundamental relationship between performance and capacity. The idea of using queueing networks to model Web systems has been suggested in, e.g. [8, 12]. The approach is based on building analytical models by representing each device in the system as a queue. However, it requires the knowledge of the service demands of each type of request for each device. In real systems, such service demands can be very difficult to measure since they not include the queueing delay that might occur. Not only does this require expertise in all the devices, it involves the collection of a massive amount of data to determine the service demands of each type of request at each device, given the large number of devices and diversity of request types. Practically one can not afford to build such a detailed low-level queueing network model.

In this paper, we propose an alternative, unique and innovative approach that integrates queueing network models with inference techniques. We use high-level queueing network models to capture major resource and delay effects, which provides good traceability between the performance measures and the system architecture. The parameters of the model will however be obtained through advanced inference techniques. Contrary to the previous queueing modeling work which demands the measurement of detailed service time information at every device for every request, our approach only requires as input the most common and inexpensive measurements such as the system throughput, utilization of the servers, and end-to-end response

times. It combines the strength of both queueing network and inference models, yet it is much more flexible.

AMBIENCE (Automatic Model Building using InferENCE), is an automated Web performance modeling tool using the above approach. Equipped with a user-friendly graphical user (GUI) interface, the tool is end-to-end, self-tuning, and flexible. The tool also features a suite of powerful performance engineering functions, ranging from predicting performance, optimizing existing IT infrastructure, to suggesting cost-effective architecture design through deployment and operations. With these functions, AMBIENCE helps greatly in providing insights of the capabilities of a Web site, and in better understanding the trade-offs between scalability, QoS, capacity cost, operations risk, etc. It can be used as a powerful tool for IT architects and planners to quantify, control, and plan capacity utilization and expansion.

There have been studies in the networking and queueing literature on related, but different inference problems. For example, problems in the networking area [1, 14] are to send probes and use the results to infer the network characteristics. Our inference problem is different as the input and output data are reversed. This tool is based on work in our previous paper [19].

The rest of the paper is organized as follows. The general methodology is presented in Section 2. Our implementation of the method in a tool is then described in Section 3. In Section 4, we present a detailed example and numerical results on modeling using AMBIENCE. Some concluding remarks are given in Section 5.

2 Methodology Overview

Building a successful e-Business performance model is a complicated process. Given the complexity of the e-business infrastructure, it is even more challenging to automate this process. Our approach is based on the use of high level queueing network models and inference methods. The general methodology is shown in Figure 1. There are three key modules which are the building blocks of the tool, each includes different methods that make the automation of the modeling process possible. In the following subsections, we shall explain separately the detailed methodology related to these three modules. 1) Configuration - Workload Characterization; 2) Model building; 3) Analysis functions.

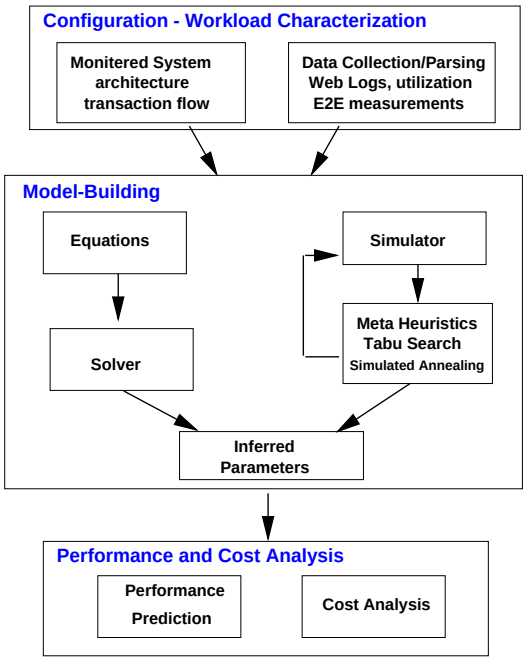

Figure 1: Block diagram of automated objective methodology for AMBIENCE

2.1 Configuration and Workload Characterization

Two initial steps are (i) understanding the system configuration and (ii) characterizing the peak workload.

2.1.1 Configuration: architecture and the key transaction flow.

Many e-business sites have a multi-tiered, geographically distributed architecture. Often servers are clustered to handle different applications or traffic from different geographical locations. Within each cluster, there are multiple tiers of machines, with each tier handling a particular set of functions. For example, front-end Web servers handle the serving of requests for static pages; the back-end database server(s) handles database related business transactions. Although there are many similarities across different Web sites, the architecture and configuration of each system are often tailored to the specific needs of a site and thus vary from business to business.

Furthermore, different applications may be running on various components depending on the type of business. Transaction flow charts therefore play an important role in providing a high-level description of how each transaction is traversing what

set of components, and how each resource is consumed by what set of transactions, etc. Such information is fundamental for extracting the underlying queueing network model.

2.1.2 Workload Characterization.

Characterizing the e-business workload is not an easy task. First, it consists of a large set of different transactions. Consider for example a typical enterprise online shopping scenario. It contains authentication transactions such as login, business transactions such as browsing the catalog, searching for products, adding items to a shopping cart, proceeding to check out, etc. Each of these transactions uses the site's resources differently. Transactions such as browsing may only involve the front-end application server to fetch some static pages which is relatively inexpensive, while other transactions such as searching or checking out may involve composition of a dynamic page or multiple queries to the database that require a large amount of processing time. In addition, user navigational patterns vary dramatically from person to person. Some users may spend all their time browsing and searching, while some frequent buyers may directly jump in for buying.

AMBIENCE incorporates a parsing and profiling engine that characterizes workload automatically by analyzing server logs. The logs contain the event records of all HTTP requests entering the server. Each record includes the client IP address, the arrival time of the request, the requested page or object, the size of the page or object, some status codes, etc. The tool processes the raw logs to obtain this information and clusters the HTTP requests into groups according to their behavior, in terms of page types (e.g. static v.s. dynamic), rate at which transactions are taking place, and the time required to process the request, etc. This analysis produces average views over a relatively long period of time (hours) and covers the majority (over 90%) of the traffic to the site. The clustering process is not trivial and a complete and accurate description of the process is beyond the scope of this paper. More details on the profiling and clustering algorithms can be referred to [10, 11].

2.2 Model Building: Queueing Networks and Inference

In order to ensure the feasibility of the modeling framework yet still capture the fundamentals of the complex e-business infrastructure, we require the model to be neither too detailed nor too general, with an acceptable level of accuracy and robustness, and to rely on a controllable number of parameters. We therefore use the multi-class queueing network model. This form of model captures major resource and

delay effects and provides good traceability between the performance measures and the system architecture. We treat the different transactions as different classes of jobs since they would make different resource demands on different components. Each resource component that incurs non-negligible delays will be modeled a *black box*. We call it a black box because of the lack of detailed service time information. Depending on whether the multiple job classes are processed sequentially or in parallel, we may need to use different service disciplines at different boxes. Consider the front-end Web servers, for example, that are primarily serving static pages and semi-static content in parallel. Each front-end server can be modeled as a generic server serving multiple classes of jobs according to the processor sharing (PS) [1] discipline.

The next challenge is to obtain the parameters of the service demands for each class at each generic server in the queueing network model. An ideal solution to capture these parameter values is to have direct measurement of resource consumption of each class at each device, a very costly practice. We therefore have to rely on performance metrics that are measurable and relatively inexpensive, such as the end-to-end response times, and the CPU load information of some or all servers. End-to-end response times, defined as the time from initiation of a request from the user, until the time that the user receives a response, can be measured easily. In fact, companies often set specific targets for these end-to-end delays so as to guarantee a satisfactory user experience. CPU load information is another metric used for Web service performance monitoring.

We provide a general inference methodology which allows one to infer these service time parameters using the most common measurements such as the system throughput, utilization of the servers, and end-to-end response times. Due to the stochastic nature of the system, it is hard to pin-point which parameter set is the best. We therefore define that a set of parameters is 'optimal' if the resulting performance is 'closest' to the measured performance, where the distance between two different performance metrics can be general, e.g. the weighted average across different components or classes.

Our inference methodology is two fold. If the underlying queueing network model has closed-form performance expressions (or good analytic approximations), we can then formulate its performance as a function of the unknown service demand parameters and use solver or other optimization tools to obtain the optimal parameter settings. Detailed discussions using this approach can be referred to [19]. On the

[1] Processor sharing is a limiting approximation to time sharing in which the quantum length tends to zero. Hence, if there are n jobs in the system, they each simultaneously receive $1/n$ of the resource.

other hand, if closed-form expressions or analytic approximations are not available, we rely on a set of meta-heuristic search methods including various tabu search algorithms and simulated annealing to search for the optimal set of parameters. More details on these algorithms can be referred to [3] and [6].

The main novel idea in the proposed methodology is that it integrates queueing network models with advanced inference techniques. This approach has a number of attractive features and advantages compared to other approaches in the literature. The first is that it utilizes the strength of the queueing network model which helps capture the fundamental relationship between performance and capacity. Secondly, by relying on inference techniques based on observed utilization and end-to-end performance data, we sidestep the problems of the traditional queueing network modeling approach. Lastly, the required input data measurements are end-to-end, the process of model building can easily be automated.

2.3 Performance Prediction and Cost Analysis

Once we have established a valid performance model, it can be used to provide quantitative solutions to predict performance, optimize existing IT infrastructure, and to suggest cost-effective architecture design through deployment and operations.

To give a simple example, to predict the performance under a different workload, (say the predicted load with expected future business growth), we can either plug the input load into analytical queueing formula or feed it into the simulator. Based on the output performance metrics and resource utilization metrics, we then know what level of service quality can be delivered for each service with the current infrastructure, and where the bottleneck of the system is, etc. Similarly, the inferred model can be used to understand the scalability and analyze a series of what-if scenarios. For example, what if we downscale or upscale the system architecture? what if the browse-to-buy ratio changes? what if the service level requirement increases. . . ?

In today's e-business environment, it is important for IT planners to associate economic factors with capacity. We provide cost analysis functions that quantify cost-performance trade-offs. From the total cost of providing the e-business infrastructure and the capacity utilization, we can compute the cost per transaction, cost of each service, and the cost to support each customer. We further rely on optimization methods to search for the optimal design by confirming the IT budgets.

3 The AMBIENCE Tool

The capabilities discussed in section 2 have been implemented in a tool called AMBIENCE. The fundamental goal in the AMBIENCE system is to simplify and automate the process so that it is easy-to-use, flexible and robust. Figure 1 can also be viewed as the guideline diagram for the software architecture of the tool implementation.

In AMBIENCE, an XML based specification describes all of the above aspects of modeling activities. We have developed a Java and XML based toolkit with a user-friendly GUI interface and a automated model building engine.

AMBIENCE follows systematically five key steps in the modeling process:
1) Initialization 2) Model Building 3) Validation 4) Prediction and 5) Cost Optimization. In the following discussions, we reveal the key features of each component in more detail. Some snapshots of the user interface are shown in Figure 2.

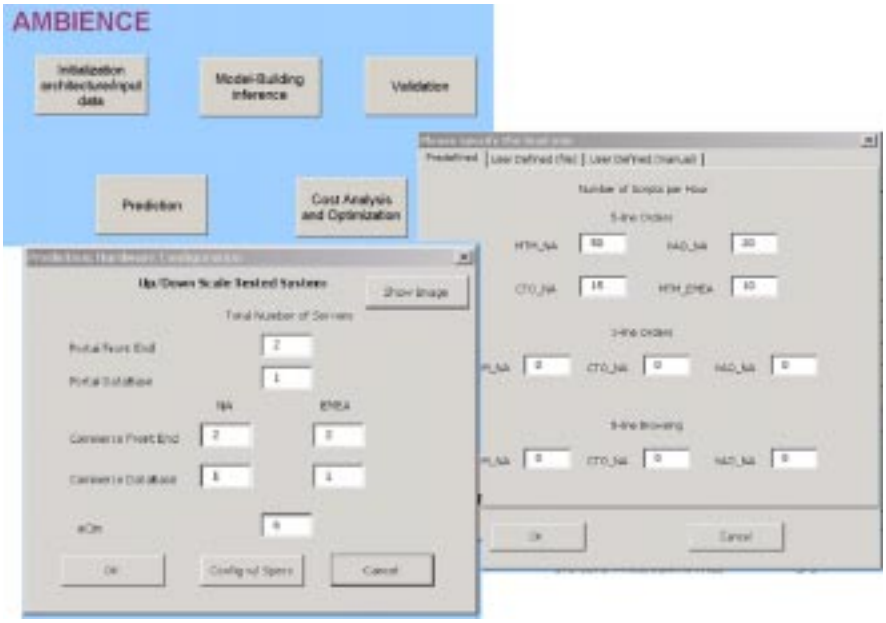

Figure 2: The AMBIENCE User Interface

1) Initialization (architecture and data input): Since the system architecture varies from business to business, we would like the AMBIENCE tool to be applicable across different configurations. Furthermore, as a capacity planning tool, it should be flexible enough to adapt to changes in system architecture and designs. Our solution is to have a hands-on GUI interface where users can drag and draw

various server boxes and define the traversing path for each transaction crossing these various boxes. The input architecture and transaction flow are then stored in an XML file for future use. The data collection process is straight-forward. Depending on whether the workload profile is defined or not, a parsing and profiling engine may be invoked to first profile the workload.

2) Model Building and Inference: The tool first constructs a high-level queueing network model based on the input architecture and transaction flow. The model parameters are then obtained via the inference engine. The inference engine has two main parts, an analytical solver, and a simulator combined with a meta heuristics search engine. Based on the underlying model, the tool selects automatically which inference method to use.

3) Calibration and validation of the model: once the model is built with inferred parameters, it is very important to show that it is indeed valid. A model is valid if the performance metrics calculated by the model (such as response times, resource utilization and throughput) match the actual system measurements within a certain acceptable margin of error. One can use part of the measurement data for training and deriving the model and part of data for validation.

4) Performance Prediction. The interface for performance prediction allows the user to reconfigure the system (upscale/downscale, or upgrade certain boxes), and to specify a different workload. Besides prediction, this module also contains a suite of functions for scaling analysis, capacity analysis, and bottleneck identification.

5) Cost Analysis and Optimization. This module allows users to specify their cost analysis objective, budget constraints, service level target, etc. It then formulates and solves the optimization problems automatically and provides the most cost-effective design of the system architecture.

4 Applying AMBIENCE: A Case Study

In this section, we present a case study using AMBIENCE based on the testing of an unnamed commercial Web site.

4.1 The Testing Environment

We experimented with AMBIENCE in modeling a testing environment for an unnamed commercial Web site. A copy of the Web site was set up in an isolated testing environment, where the basic architecture is illustrated in Figure 3. The site contains two separate clusters of servers. The portal cluster is primarily responsible for authentication-related requests; while business-related requests are processed separately by the cluster of business transaction servers. Within each cluster, there are two tiers, front-end servers and back-end database server(s). Incoming requests to the site are routed to a collection of front-end servers by a load balancing router, such as IBM's enterprise network dispatcher (eND) [5] or similar products from other vendors. The eND routes requests in a weighted round robin fashion, so as to evenly distribute the load to the front-end servers. The processed pages are usually returned directly to the clients without going back through the eND.

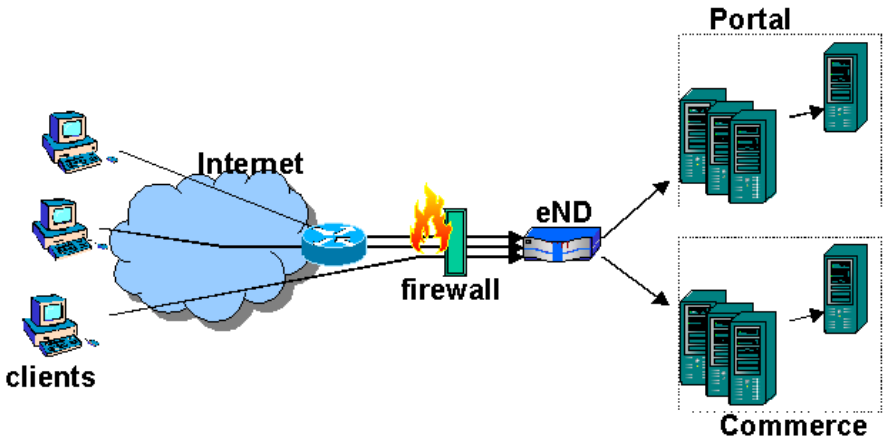

Figure 3: An Example of e-Business Site Architecture

Scripts were used to generate several classes of transactions in certain proportions, which can be easily customized. A number of load drivers (mimicking clients all over the world) were set up running these scripts so as to drive the load consistently at a desired level. We say that the system has a load of m user sessions if the total number of concurrent users in the system is m. This is realized by initiating m user sessions (usually the starting times are somewhat spread out) by the load drivers. Each session follows a sequence of page requests prespecified in the script. Once all requests in the sequence are executed, the user enters a sleep mode for a given amount of time (think time) and then starts a new cycle to execute the script again.

We used tools to measure the throughput of the front-end and back-end servers. The Web load drives collect the per-class response time statistics including their mean and variance. Aggregate utilization measures were obtained from a different set of tools on both sets of servers.

4.2 Queueing Network Model and Mean Value Analysis

The system described above can be modeled as a closed queueing network, shown in Figure 4. Using the high-level modeling method proposed in section 2, we treat each server in the Web service architecture as a service station with a multiple number of *generic servers* (or CPU's). To capture the multi-threaded processing in the computing environment, we shall assume these generic servers are serving jobs according the processor-sharing (PS) discipline.

Since the delays incurred at the firewall and network are also non-negligible, such delay effects are modeled as delay servers. We further assume that the firewall and network have constant delays in order to focus on the performance of the Web servers. In fact, this is often true in an ideal testing environment, where the testing system often resides within the same local network as a "load runner" (or, synthetic-load generator). We use another delay center to capture user think time (or idle time) between two consecutive transactions. Note that each delay center can be viewed as an infinite server queue, with each server serving at a constant rate.

We define the system *performance with m concurrent user sessions* to be the steady-state performance of the closed queueing network under the condition that m users are circulating in the network, each following a sequence of requests again and again. We treat the fraction of resource demand at each layer by each request type as a decision variable. Note that not all requests require service at the back-end server; from the modeling perspective, this corresponds to a zero service time when such a request goes through the back-end server.

We first observe that the closed queueing network given by Figure 4 is symmetric. This is because all the delay centers (eND, firewall, thinktime, and network) can be considered as IS (Infinite Server) centers, while all the Web server boxes are modeled as PS centers (i.e. under the processor sharing discipline). Therefore, the stationary distribution of this symmetric closed queueing network must have a product form (refer to, e.g. [7, 18]). The mean value (MVA) technique for product-form networks developed by Reiser and Lavenberg [13] can then be applied for computing the average delay of the system recursively from the m concurrent users to $m+1$ concurrent users.

Consider the case when there are m users in the closed queueing system. We shall

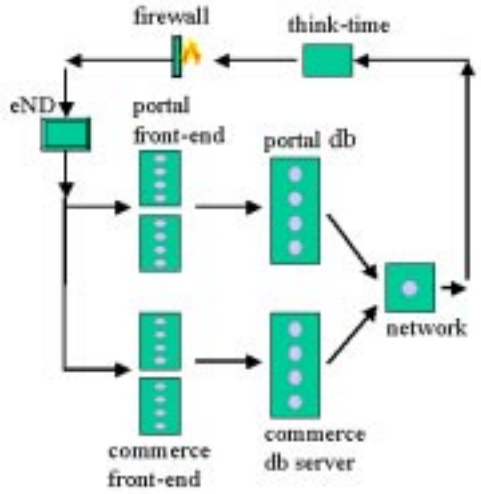

Figure 4: An Example of Queueing Network Model

use the following notations,

$$
\begin{aligned}
I &:= \text{number of service stations;} \\
y_i &:= \text{number of times that a user visits station } i; \\
S_i &:= \text{mean service time of requests at station } i; \\
\lambda_i(m) &:= \text{arrival rate for requests to station } i; \\
R_i(m) &:= \text{mean response time of requests at station } i; \\
L_i(m) &:= \text{average number of requests at station } i; \\
\rho_i(m) &:= \text{utilization level of service station } i; \\
T(m) &:= \text{total response time of a single user going through one cycle.}
\end{aligned}
$$

Based on the above definitions, the system throughput then equals $\frac{1}{T(m)}$. In addition, we have the following equations.

$$T(m) = \sum_{i=1}^{I} y_i \cdot R_i(m), \tag{1}$$

$$\lambda_i(m) = m \cdot y_i / T(m), \tag{2}$$

$$\tag{3}$$

From Little's Law, we have,

$$L_i(m) = \lambda_i(m) R_i(m), \quad \text{for } i = 0, 1, \ldots, I. \tag{4}$$

The *arrival theorem*, which is the foundation of the mean value technique of [13], states that the number of customers "seen" by a job upon arrival at a station is equal to the mean queue length at the station when the network has *one less job*. That is,

$$R_i(m+1) = S_i \cdot [\ 1 + L_i(m)\ ], \quad i = 1, \ldots, I. \tag{5}$$

Therefore, suppose we know the all the service demands S_i, by recursively making use of equations (5), (4), (1), and (2), with the initial conditions $L_i(0) = 0$, $i = 1, \ldots, I$, we can compute the mean values $L_i(m)$, $R_i(m)$ and $T(m)$ for all $i = 1, \ldots, I$ from the m user scenarios to $m+1$ user scenarios , and so on. Also, observe that the computational complexity of the algorithm is simply $O(Im)$.

It remains to infer the service demand parameters. As stated earlier, inference can be done in a number of different ways using different criteria. In this example, we choose to use an optimization approach with the objective

$$\min |Measured\ RT - Predicted\ RT|.$$

Observe that the total response time is a nonlinear function of the service demand parameters, based on the relationship between response time and service time at different queues (e.g. formula for PS queues will be different from IS queues). This nonlinear optimization problem is then solved to obtain the per-class service-time requirements.

4.3 Results

The above modeling, queueing analysis and inference methods have been axperimented on a set of real measurement data collected from the testing environment. Figure 5 and Figure 6 show the corresponding results.

In Figure 5, we have obtained the model parameters based on the measurements of a single testing point, namely, the point when the system has 25 concurrent user sessions. The input measurement data at this single testing point includes the end-to-end response times of all requests and utilization numbers at the Web server boxes. Based on the inferred model, we then plot the total mean response time as a function of the total number of concurrent user sessions, which is shown in the left plot of Figure 5. The middle and right plots of Figure 5 show the corresponding resource utilization levels at different layers of the Web service architecture.

Although the model was built only based on a single point (load with 25 user scenario), the results are quite impressive when comparing the predicted results with the measured performance at other load conditions, as shown in Figure 6. In fact, the

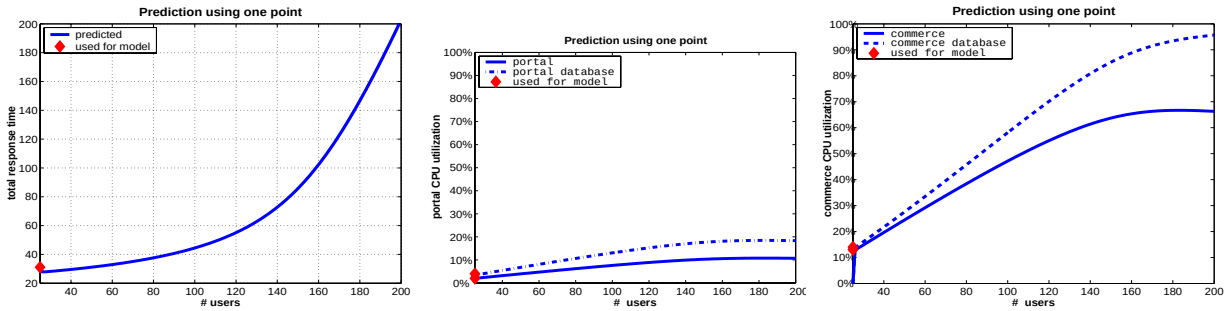

Figure 5: Prediction results with model built using one point. **Left:** Total response times as a function of number of user scenarios; **Middle:** Front-end CPU utilization; **Right:** Back-end CPU utilization.

accuracy can be improved further if the inference was based upon a few more points (load scenarios).

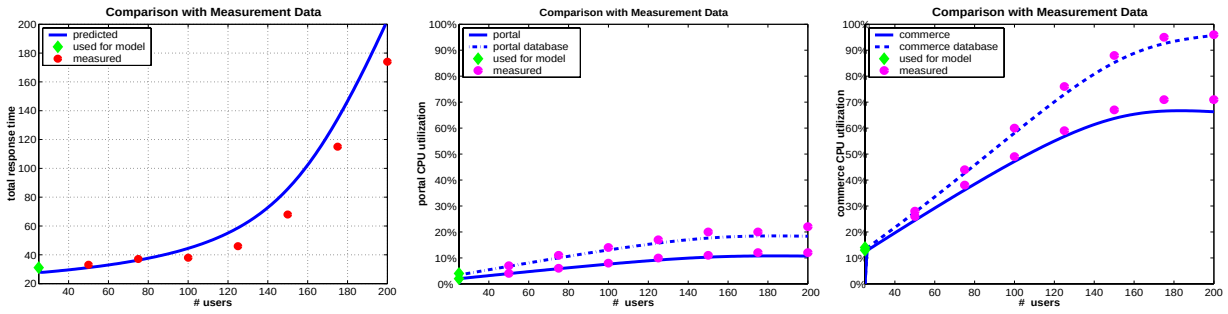

Figure 6: Comparison with measurement data. **Left:** Total response times as a function of number of user scenarios; **Middle:** Front-end CPU utilization; **Right:** Back-end CPU utilization.

We observe that the mean response time curve first stays relatively flat; it then blows up quickly and almost exponentially after the 'knee' when the total number of users is around $140 - 160$. This sharp performance decline indicates that somewhere in the system there is shortage of capacity. Meanwhile, from the server utilization curves, we see that first the system scales up nicely in a linear fashion; then, when the number of users reaches the level of $140 - 160$, the linear scalability starts to break down. In fact, it is the Commerce DB server that has become the bottleneck since

its utilization is ramping up quickly close to 100%.

We can further focus on the bottleneck server to conduct a scaling analysis. Figure 7 shows that by increasing the bottleneck server capacity, the response time of the problematic transaction drops quickly. As the cost of adding more server capacity increases linearly, the performance improvement does not react in the same linear fashion. We therefore need to make a trade-off between the economic costs and choose the right level of capacity expansion based on the desired service level.

Figure 7: Scaling Analysis of the Bottleneck Server

In fact, as we double the bottleneck server capacity, the system performance may be bounded by a new bottleneck server. One could repeat the above scaling analysis and search for the next place to upgrade the capacity. With the AMBIENCE cost optimization tool, this searching process is automated. Figure 8 shows an example that,sssssssss once the budget constraint and service level requirement are specified, the tool will then produce the optimal configuration through the cost optimization module.

5 Conclusion

We have described in this paper a unique and innovative tool that automates the performance modeling process for a complex Web service infrastructure. Equipped with a user-friendly GUI interface, the tool is end-to-end, self-tuning, and flexible. It adapts to any site configuration and environment, and requires as input a minimal amount of commonly available measurement data on the configured system. The tool also features a suite of powerful performance engineering functions, ranging from predicting performance, optimizing existing IT infrastructure, to suggesting cost-effective architecture design through deployment and operations. We presented a

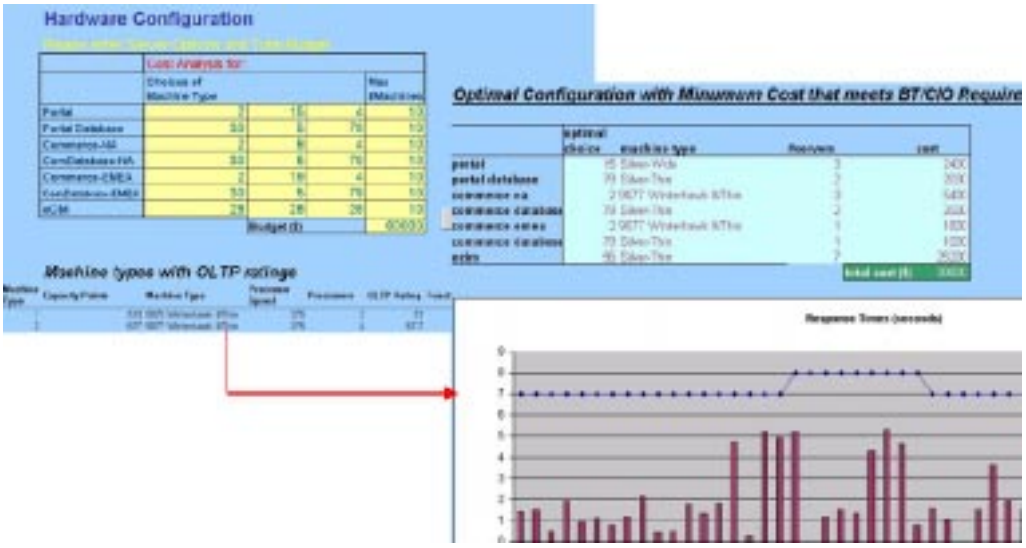

Figure 8: Cost Optimization: The most Cost-effective Design

case study with real data to demonstrate the power of the tool for capacity planning, SLA provisioning and other related issues.

Our analysis of Web traffic request patterns [15, 9, 17] demonstrates complex characteristics that include dependencies, periodicity, burstiness, high variability, and shifts in regimes. These complex traffic patterns can have a significant impact on the latency encountered by user requests, in terms of higher mean response times (several orders of magnitude in some cases), higher response-time variance, and heavy-tailed response time distributions [16]. This in turn can have a significant impact on the capacity requirements of e-business sites in order to provide an acceptable level of performance and high availability. This is on-going research and we would like to further incorporate these considerations into AMBIENCE to further enhance the tool.

References

[1] S. Alouf, P. Nain and D. Towsley. Inferring network characteristics via moment-based estimators. In *Proceedings of the IEEE Infocom 2001 Conference*, April 2001.

[2] C. Bishop. *Neural Networks for Pattern Recognition*. Morgan Kaufmann, San Mateo, 1995.

[3] F. Glover and M. Laguna. *Tabu Search*, Kluwer, Boston, 1997.

[4] M. Goldszmidt, D. Palma and B. Sabata. On the quantification of e-business capacity. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, p.235-244, October 2001, Tampa, Florida.

[5] G. Hunt, G. Goldszmidt, R. King, and R. Mukherjee. Network dispatcher: A connection router for scalable internet services. In *Proceedings of the 7th International World Wide Web Conference*, April, 1998.

[6] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation. In *Operations Research*, vol. 37, p.865-892, 1989.

[7] F.P. Kelly. *Reversibility and Stochastic Networks*. Wiley, Chichester, 1994.

[8] L. Kleinrock. *Queueing Systems Volume II: Computer Appplications.* John Wiley and Sons, 1976.

[9] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. Traffic model and performance evaluation of Web servers. *Performance Evaluation*, 46:77–100, October 2001.

[10] Z. Liu, M.S. Squillante, C.H. Xia, and L. Zhang. Preliminary analysis of various SurfAid customers. Technical report, IBM Research Division, July 2000. Revised, December 2000.

[11] Z. Liu, M.S. Squillante, C.H. Xia, S. Yu and L. Zhang. Web Traffic Profiling, Clustering and Classification for Commercial Web Sites. *The 10th International Conference on Telecommunication Systems, Modeling and Analysis (ICTSM10)*, 2002.

[12] D. Menasce and V. Almeida. *Capacity Planning for Web Performance*. Prentice hall, 1998.

[13] M, Reiser and S.S. Lavenberg, Mean-value analysis of closed multichain queueing networks, In *J. ACM*, vol. 27, p.313-322, 1980.

[14] V. Sharma, R. Mazumdar. Estimating traffic parameters in queueing systems with local information. *Performance Evaluation*, 32:217-230, 1998.

[15] M.S. Squillante, D.D. Yao, and L. Zhang. Web Traffic Modeling and Server Performance Analysis. In *Proceedings of the 38th IEEE Conference on Decision and Control (CDC)*, p.4432-4439, 1999.

[16] C.H. Xia and Z. Liu. Queueing systems with long-range dependent input process and subexponential service times. To appear in *ACM SIGMETRICS*, 2003.

[17] C.H. Xia, Z. Liu,, M.S. Squillante, L. Zhang and N. Malouch. Analysis of performance impact of drill-down techniques for Web traffic models. Submitted.

[18] R.W. Wolff. *Stochastic Modeling and the Theory of Queues*. Pretice Hall, 1989.

[19] L. Zhang, C.H.Xia, M.S. Squillante and W.N. Mills III. Workload Service Requirements Analysis: A Queueing Network Optimization Approach, In *10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02)*.