Application Performance Prediction in Autonomic Systems

Shobhana Kirtane
Clemson University
100 McAdams Hall
Clemson, S.C. 29634-0974
skirtan@cs.clemson.edu

Jim Martin
Clemson University
100 McAdams Hall
Clemson, S.C. 29634-0974
jim.martin@cs.clemson.edu

ABSTRACT
An autonomic system is an intelligent system that is capable of self-configuration, self-healing, and self-management. Application performance prediction is a powerful tool that can be used in an autonomic system. Predicting application performance based on current or anticipated conditions provides fine-grained information that increases the chances that the autonomic manager makes correct decisions. In this paper, we report on the design and implementation of a system that can be used by an autonomic manager to predict the response times of transaction-oriented applications. Preliminary results suggest that our method leads to an average prediction error of less than 15% over a range of network and server loads.

Categories and Subject Descriptors
C.4 [**Computer Systems Organization – Performance of Systems**]: *Modeling techniques.*

General Terms
Measurement, performance

Keywords
Autonomic systems, intelligent systems, application prediction

1. INTRODUCTION
Corporate IT professionals struggle with complex and evolving information systems on a daily basis. Autonomic computing addresses the complexity of today's IT environment by facilitating the development of self-configuring, self-healing and self-managing systems [9,15]. An autonomic system, as illustrated in Figure 1, relies on system information from monitors that is analyzed by an autonomic manager, possibly resulting in system adjustments that are necessary to achieve service level objectives.

ACM SE'06, March, 10-12, 2006, Melbourne, Florida, USA

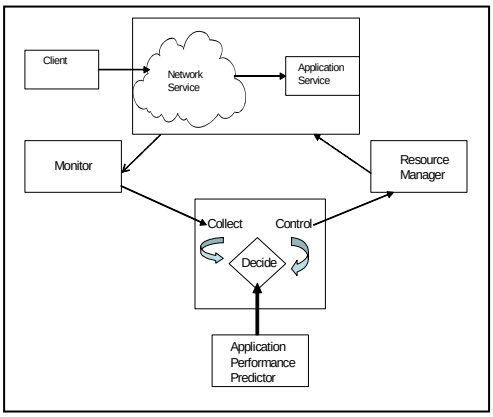

Figure 1. Autonomic control loop

Autonomic computing and web services are synergistic. Not surprisingly much of the autonomic standards work is being driven by the Internet community. The current web services framework, defined by the Web Services Description Language (WSDL), the Universal Discovery, Description and Integration (UDDI) service registry, and the Simple Object Access Protocol (SOAP), supports e-business on the Internet [26,27,28,29,30]. Methods for service level agreement (SLA) management in a web services environment have been proposed [6,13,24,14]. A standard interface for managing web services has also been proposed [31]. The motivations for our research are based on the observation that there has been tremendous progress made in defining interfaces for autonomic systems but there has been very little research in developing theory that can be applied to solving the decision algorithms that are at the heart of autonomic systems. We believe that application prediction is a powerful method that addresses this gap.

In this paper, we report on the design and implementation of a service management system that can predict the response times of transaction-oriented web applications. For brevity, we focus on the prediction method and leave the details of how an autonomic manager might use predictive capabilities for a future paper. The remainder of the paper is organized as follows. In section 2 we present related work. In section 3 we overview the service management system architecture. In section 4 we discuss the network and server delay models that we have developed. Section 5 describes our experimental methods. Section 6 presents the results of an analysis of the system. In

the last section we summarize conclusions and identify future work.

2. RELATED WORK

It is well known that correlating component-level threshold violations with system-level service level objective violations is difficult [19]. Our research addresses this problem by engaging predictive models that map easily obtainable performance measures into anticipated application performance. This capability has tremendous application in autonomic systems that involve proactive system management, service level objective management and adaptive systems.

Software performance engineering utilizes prediction techniques to ensure that a software system will meet performance requirements before deployment [1]. Prediction is commonly used to design and plan networks [22]. Prediction has also been used in computer networks for admission control or load balancing functions [25]. Recent work in self-managing systems is similar in spirit to our research [4,20]. They used queuing theory to model system behavior and subsequently to adapt system configuration to obtain desired behavior. Our work differs in that we predict application performance rather than aggregate system performance. Although the authors in [5] did not use prediction, their work has similar motivations to ours in that they seek to develop enabling technology for autonomy and self-management.

3. SYSTEM ARCHITECTURE

Figure 2 illustrates the system architecture. For demonstration purposes, we assume that the prediction service is engaged by a client that intends to invoke the desired service but before doing so asks the service manager for the predicted performance. The service manager interacts with the system and based on current conditions computes a performance prediction which it returns to the client.

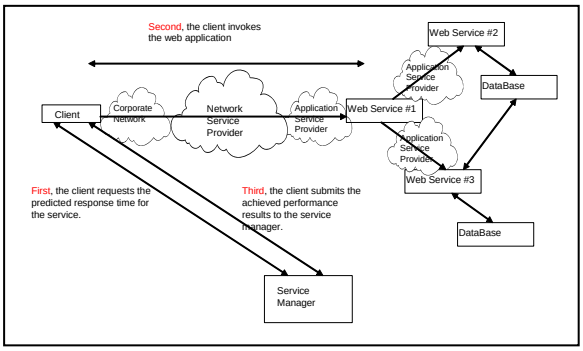

Figure 2. System architecture

We focus on transaction-oriented applications that are best assessed using response times. The simplest service is an HTTP GET operation from a web server. We refer to this as a single level transaction. A more complex application is a web service that involves multiple servers (as illustrated by Figure 2). We refer to the service accessed by the client as the target web service and all additional services that are invoked by the target web service as embedded services.

The performance of a web application is best defined during the requirements and design phases. For example, a web application might be organized in sequential processing stages beginning with the verification of client request parameters, followed by a series of network requests to embedded servers (e.g., database requests, subsequent web application requests), and then a computation section that processes the data, and finally a stage that sends data or results back to the client. In our system, we assume that all web applications, including embedded services, contain the necessary information to be able to compute an estimate of how long the application will take to execute. The approach that we use to model web service performance is similar to that described in [19] where a transaction delay consists of the sum of delays caused by embedded services in addition to delay incurred locally. The long term direction of our research is to develop programming language support that automates the application performance description and also to develop the necessary support services that provide low level system performance information used by the application models. For the preliminary work reported in this paper, we have developed a simple web server performance model that includes local system performance monitors and does not involve embedded services.

Each application must provide a set of interfaces. First, a performance request interface allows a caller to learn of the parameters supported by the applications prediction model. For example, the performance request interface associated with a simple web server might simply require the size of the web object the caller anticipates requesting. Second, the application must also provide a standard interface by which a management entity (referred to as the service manager) can request a performance prediction.

4. PREDICTION MODELS

The response time prediction for a single level transaction involves two models: a model of how the network impacts the transaction and a model of how server delays impact the transaction. This approach assumes that both components of delay are independent of each other. We rely on the method of critical path analysis to justify this assumption. Critical path analysis is a method to identify an application's performance [16]. The method makes use of the observation that only some component activities in a distributed system impact overall response time. In [3] the authors apply the method to a Web server under various levels of load. The results suggest that high server delays can contribute up to 80% of the overall file transfer time for small files. For larger files, the impact of the network tends to dominate regardless of server load. As illustrated in Figure 3, for HTTP transactions, the majority of server delay occurs between the arrival of the last packet containing the HTTP GET and the first data packet sent in response by the server. This delay does not impact TCP's RTT estimate nor will it cause packet loss. Based on this result (which we confirmed in our measurements) we model single level transactions by adding the server delay to the network delay.

Once the service manager receives a prediction request from a client, it invokes the target service's method to obtain a server

delay estimate using application model parameters included in the client's request. If the target service involves embedded transactions, each will cause an additional, recursive request to the service manager. Once the lowest level embedded service prediction requests have been completed, the results propagate back to the target service who returns the server delay request to the service manager. The service manager then accounts for the impact of the network on the application and returns the final predicted response time to the client.

4.1 Network Delay Model

There have been a significant number of analytic models of TCP developed over the past 10 years. One of the earliest efforts modeled TCP's saw-tooth sending rate behavior as a function of *1/sqrt(p)* where *p* is the loss rate [18]. More sophisticated models use different packet loss models (from highly bursty to uncorrelated) [10,11,21] and add TCP timeout, connection startup and teardown effects [7]. Based on our earlier work, we observed that each model is best suited to specific environments[17]. The prediction error of each model increases in environments that violate assumptions that have been made, especially assumptions involving packet loss.

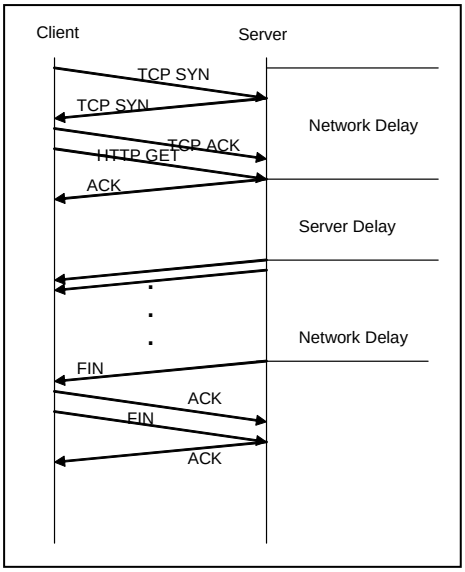

Figure 3. Critical path analysis of a TCP Cx

In prior work we developed and validated a web response time (WRT) model of TCP transactions[17]. In [17] we provided empirical evidence suggesting that the WRT model performs better than other proposed models over a broader range of network scenarios. The model is a stochastic model of TCP that includes the startup time, the impact of slow start, the contribution from steady-state data transfer and the connection teardown. The parameters to the model are as follows:

response time = f(firstPacketLossRate, correlatedLossRate, RTT, dataSize)

The *firstPacketLossRate* (referred to as *p*) is the frequency that blocks of loss occur. The *correlatedLossRate* (referred to as *p'*) is the probability of consecutive losses following a first loss event. The other parameters are the path RTT and the amount of data to be transferred from the server to the client. If the loss rate and RTT parameters accurately reflect stationary network conditions between the client and server, we found that the predicted response time will be within 15% of the actual response time over a range of network conditions.

This model parameter's can be obtained non-invasively. The RTT is based on a simple ping. We have observed in our experiments that a ping-based RTT is very similar to the RTT observed by TCP connections operating over the same path as the ping monitor. We developed a client-server UDP application that is used to estimate *p* and *p'* non-invasively. The UDP tool client sends a burst of packets to the server who observes the number of bursts with loss and the frequency of bursts with consecutive losses. This tool implements the 'three counter technique' to compute correlated packet loss [10,11]. For every probe packet successfully sent, the success counter is incremented by 1. The *pdropped* counter is incremented by 1 for each loss event. For n successive packets lost the *p* dropped counter is incremented by n-1. From the values of these three counters we compute the values of *p* and *p'* as follows.

$$p = \frac{pdropped}{success}$$

$$p' = \frac{p\_dropped}{pdropped + p\_dropped}$$

The application estimates the values of *p* and *p'* in the downstream direction. For the experiments reported in this paper, the client is configured to send a burst of 4 packets every two seconds. Refer to [17] for a complete description of the model.

4.2 Server Delay Model

In response to a prediction request from the service manager, the application (whether it is the target service or an embedded service) must compute its anticipated delay based on the parameters provided in the prediction request. For the preliminary work reported in this paper, we have developed and implemented a web server delay model based in part on [8]. We estimate web server delay, T_{server} , as follows:

$$T_{server} = T_{QueueDelay} + T_{CPUDelay} + (1 - H)T_{Disk}$$

$T_{QueueDelay}$ *:* The waiting time from when the request reaches the web server until when the first packet of data is sent. This will depend on current server load but is independent of the size of data associated with the request. We estimate this delay by running an application monitor on the server machine. For a web server (or a simple web service) we run a sockets program that periodically issues an HTTP GET for a small object from the web server operating on the localhost. The waiting time is obtained by taking the difference between the minimum response time and the average of the response time over the last 5 minutes.

$T_{CPUDelay}$: The amount of CPU overhead associated with a request. For an HTTP request, the processing time is estimated to be very small (microseconds).

H: The cache hit rate. The web object might be in the drive's cache, the file system's cache or the web server's cache. We assume a pessimistic 20% aggregate hit rate.

T_{Disk} : The time it takes to transfer the data from disk into memory. Since the performance of modern drives are very comparable, we use the specifications for a commonly used disk drive[1]. We assume an average positioning delay of 13.1 milliseconds and a transfer delay based on a transfer rate of 150MBytes/second.

5. VALIDATION

To validate our system we conducted two sets of experiments. Both sets utilized the network testbed illustrated in Figure 4. The network testbed consists of 2 private networks connected through a router. The machine labeled Dummynet Router is a Dell Optiplex GX260 with 512Mbytes of RAM and is equipped with a single Pentium processor running at 2.4 GHz. It runs FreeBSD 4.8 and uses dummynet[23] to control path latency and bandwidth. For these studies, dummynet was configured as two unidirectional pipes with a bandwidth of 10Mbps and a latency of 30ms. The capacity of the dummynet queues was set to 40 packets each. Both private networks operate at 1Gbps.

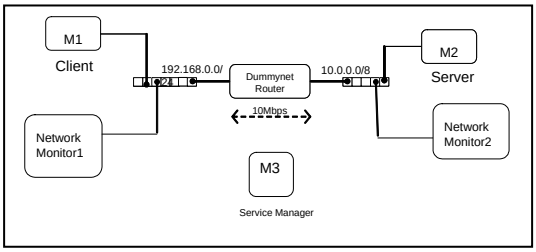

Figure 4. Network testbed

The M1 and M2 machines are Dell Precision 450 workstations equipped with dual 2.4GHz Xeon processors and 1GBytes and 2GBytes of RAM respectively. The M3 machine is a Dell Optiplex GX260 with 800MBytes of RAM. The machines labeled Network Monitor1 and Network Monitor2 are Dell Optiplex GX1p with 128MBytes RAM. All machines run Redhat 8.0 with kernel version 2.4.18-14 compiled for symmetric multiprocessing support. The client is located at M1 and the server is located at M2. IBM WebSphere Studio Application Developer Version 5.1.2 for Linux and IBM WebSphere Application Server Version 5.1 are installed on M1, M2, and M3.

[1] The 80 GB Western Digital Caviar SE 7200 RPM, 8 MB Cache, Serial ATA Hard Drive has an average access time (including seek, settle and latency random delays) of 13.1milliseconds. See http://store.westerndigital.com/product.asp?sku=2521615.

We used the Surge (Scalable URL Reference Generator) [2] to generate background traffic. The location of the Surge clients depends on the experiment but in all experiments the Surge clients interact with the Apache Web Server (version 2.0.4) operating on machine M2. The TCP socket buffer sizes were set to 64Kbytes on all machines. The client is a Unix sockets program that periodically requests a particular Web object from the Web server on machine M2. The standard ping program is run between the Network Monitor1 and Network Monitor2 machines. The ping results are used only for the visualization of WAN performance presented in this paper. The network performance data required by the network prediction component is collected by the network monitor programs which also run on these machines. This program is a Unix sockets program that collects the RTT and loss data as required by the prediction server, periodically submitting the results to the service manager.

We conducted two experiments which we refer to as Set1 and Set2. The objective of the experiments was to validate the network and server delay models. To achieve this goal we used the simplest type of server, a Web server. The experiments consist of a client test program requesting a web object of varying size from the server. Prior to issuing the HTTP GET, the client requests a transaction prediction from the service manager. In the Set1 experiment, in addition to varying the web object size we also vary the congestion level over the emulated WAN link. The objective of the Set1 experiment is to validate the prediction models when the majority of the delay is caused by the network. The WAN utilization is controlled by varying the number of Surge clients that run on machine M1. The Set2 experiment is similar except that there is no competing traffic over the WAN. Instead, a varying number of Surge clients run on Network Monitor2 resulting in a range of server utilization levels.

The experimental procedure for both Set1 and Set2 is as follows. For a particular run the number of Surge clients and the size of the Web object that is being transferred from the server to the client are specified. The number of Surge clients range from 20 to 80. The size of the Web objects varies from 10Kbytes to 80Kbytes in 10Kbyte increments. A run begins by starting the Surge clients and letting them run for 5 minutes and then the network monitors are started. After 5 more minutes, the client test program is started, repeatedly pulls the configured Web object, and after 15 minutes the test is halted. The mean of the client response time samples represents the actual response time. During the run, network statistics are sent to the service manager which computes the predicted response time. The mean of predicted response times represents the predicted response time for the run.

6. SIMULATION AND RESULTS

In this section we present empirical evidence that demonstrates and validates our models using HTTP transactions.

6.1 Set1 Analysis

Figures 5 and 6 illustrate the WAN performance for the Set1 experiment. A ping loss monitor ran between the Network Monitor1 and Network Monitor2 machines. The loss rate varied

from .1% to 6%. The utilization of the WAN varied from 40% to 100%.

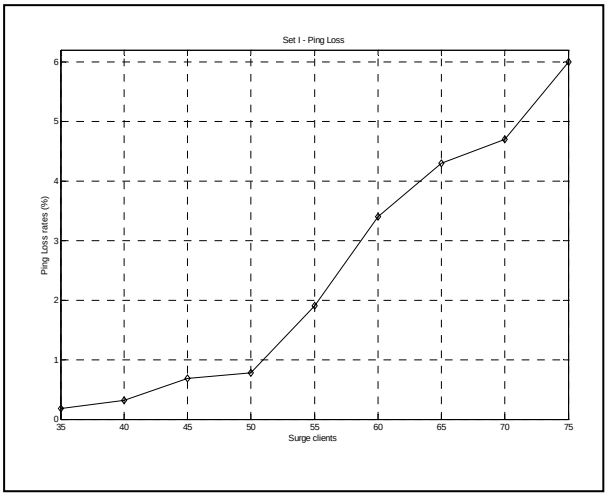

Figure 5. Loss rates for Set1 experiment

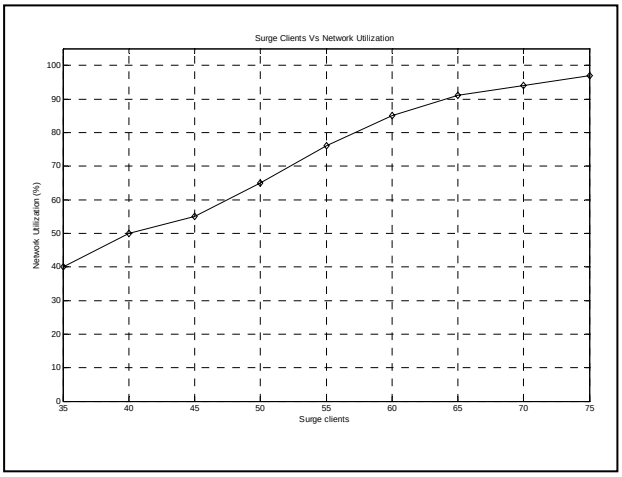

Figure 6. Bottleneck link utilization for Set1

Figure 7 compares the results of the predicted response time to the actual response time for runs configured to transfer 10Kbytes, 30Kbytes, 60Kbytes and 80Kbytes of data. The error associated with the prediction ranged from 0 to 35%. The average error was less than 15% which is similar to what we found in our previous results. We observe that the prediction error is lowest for the runs where the loss rate was in the range of .5% - 2%. Outside of this range, TCP implementation or configuration issues which are not considered by the model cause the error to approach 15%.

6.2 Set2 Analysis

In the Set2 experiment, we vary the number of Surge clients running at Network Monitor2 to control the load at the server (machine M2). Figure 8 plots the actual and predicted response times for 4 different Web object sizes as the load on the server changes. The X-axis represents the server load and is the average of the first load average number available in /proc/loadavg sampled by a monitor program that executes during a run. The load average number gives the number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over 1 minute.

Server Load	Server Delay Contribution
20	5.1%
30	6.1%
40	20%
50	35%
60	62.5%
70	76%
80	83.5%

Table 1: Server delay contribution for 10KByte Web object runs

Figure 8 shows that the prediction error ranged from near 0 to less than 25% (less than 15% on average). Table 1 shows the contribution of the server delay for each predicted response time in the Set2 10KByte graph. At the highest load level, the network delay represented only 16.5% to the overall response time. The server delay prediction at this load level was accurate to within 1%.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented the design and validation of a web application prediction method that is intended to be used by an autonomic system. The prediction capability targets web applications that are modeled as a simple service consisting of the transmission of a request, the processing of the request, and the transmission of result data back to the client. More complex web applications are modeled by requiring applications to recursively request predictions for embedded services from the service manager. We plan on extending the architecture to support multiple service managers to address scalability limitations caused by a single service manager and organizational boundary issues.

Although still under development, we have demonstrated the approach using a simple web server application. Our results suggest that the system is able to predict the response time of client GET requests to within 15% of the actual response time over a range of network and server load settings. The next step on this project is to model more complex applications. We plan on releasing our code as an open source project. The goal is to establish an open source development effort to further develop the prediction framework thereby facilitating the development of new performance models by the research community.

8. REFERENCES

[1] S. Balsamo, A. DiMarco, P. Inverardi, M. Simeoni, "Model-Based Performance Prediction in Software

Development", IEEE Transactions on Software Engineering, vol. 30, no.5, pp295-310, May 2004.

[2] P. Barford, M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", In Proceedings of Performance '98, ACM SIGMETRICS'98, pp151-160, July 1998.

[3] P. Barford, M. Crovella, "Critical Path Analysis of TCP Transactions", SIGCOMM 2000, pp 127-138, Jan 2000.

[4] M. Bennani, D. Menasce, "Assessing the Robustness of Self-Managing Computer Systems under Highly Variable Workloads", Proceedings of the International Conference on Autonomic Computing (ICAC'04), 2004.

[5] D. Breitgand, E. Henis, O. Shehory, "Automated and Adaptive Threshold Setting: Enabling Technology for Autonomy and Self-Management", Proceedings of the International Conference on Autonomic Computing (ICAC'05), 2005.

[6] M. Buco, et. Al., "Managing eBusiness on Demand SLA Contracts in Business Terms Using the Cross-SLA Execution Manager SAM", The Sixth International Symposium on Autonomous Decentrazlized Systems (ISADS03), April 2003.

[7] N.Cardwell, S.Savage and T.Anderson , "Modeling TCP Latency," in Proceedings of IEEE INFOCOM, Tel Aviv, Israel, March 2000

[8] R. Doyle, J. Chase, O. Asad, W.Jin, A. Vahdat, "Model-Based Resource Provisioning in a Web Service Utility", Fourth Usenix Symposium on Internet Technologies and Systems", USITS'03, March, 2003.

[9] A. Ganek, T. Corbi, "The Dawning of the Autonomic Computing Era", IBM Systems Journal, Vol. 42, No. 1, pp5-18.

[10] M. Goyal, R. Guerin, and R. Rajan, "Predicting TCP throughput from non-invasive data", Technical Report, University of Pennsylvania, Nov 2001.

[11] M. Goyal, R. Guerin and R. Rajan, "Predicting TCP throughput from non-invasive network sampling", IEEE Infocom 2002.

[12] A. Iyengar, M. Squillante, L. Zhang, "Analysis and Characterization of Large-scale Web Server Access Patterns and Performance", WWW2004.

[13] A. Keller, G. Kar, H. Ludwig, A. Dan, J. Hellerstein, "Managing Dynamic Services: A Contract Based Approach to a Conceptual Architecture", Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS2002), pp 513-528, April 2002.

[14] A. Keller, H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", IBM Research Report, RC22456, May 2002.

[15] J. Kephart, D. Chess, "The Vision of Autonomic Computing", IEEE Software, pp 41-50, January 2003.

[16] K. Lockyer, Introduction to Critical Path Analysis, Pitman Publishing Company, New York, NY, 1964.

[17] J. Martin, H. Karlapudi, "Web Application Performance Prediction", Proceedings of the IASTED International Conference on Communication and Computer Networks, (Boston, MA, Nov, 2004), pp. 281-286.

[18] M. Mathis, J. Semke and J. Mahdavi, "The Macroscopic Behavior of TCP Congestion Avoidance Algorithm", in Computer Communications Review, vol. 27, number 3, July 1997.

[19] D. Menasce, "Composing Web Services: A QoS View", IEEE Internet Computing, Nov/Dec 2004.

[20] D. Menasce, M. Bennani, "On the Use of Performance Models to Design Self-Managing Computer Systems", Proceedings of Computer Measurement Group Conference, Dallas TX, December 2003.

[21] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation", in SIGCOMM '98, September 1998.

[22] K. Papagiannaki, N. Taft, Z. Zhang, C. Diot, "Long-Term Forecasting of Internet Backbone Traffic: Observations and Initial Models", IEEE Infocom 2003.

[23] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," ACM Computer Communication Review, vol. 27, no. 1 , January 1997

[24] A. Sahai, A. Durante, V. Machiraju, "Automated SLA Monitoring for Web Services", IEEE/IFIP DSOM 2002, Montreal, Canada, Oct. 2002.

[25] Y. Shu, Z. Wang, O. Yang, "Prediction-based Admission Control Using FARIMA Models", Proceedings of IEEE ICC'00, June 2000.

[26] SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation, M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Nielsen, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part1-20030624/).

[27] SOAP Version 1.2 Part 2: Adjuncts, W3C Recommendation, M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Nielsen, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part2-20030624/).

[28] UDDI Version 3.02 API Specification. Universal Description, Discovery and Integration, uddi.org, June 2001.

[29] UDDI Version 3.02, Universal Description, Discovery and Integration, October 2004, http://uddi.org/pubs/uddi_v3.htm.

[30] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Working Draft, R. Chinnici, M. Gudgin, J-J. Moreau, J. Schlimmer, S. Weerawarana, 10 November 2003.

[31] Web Services Distributed Management, Oasis, 2004, available at http://docs.oasis-open.org/wsdm/2004/12-/wsdm-muws-part1-1.0.pdf

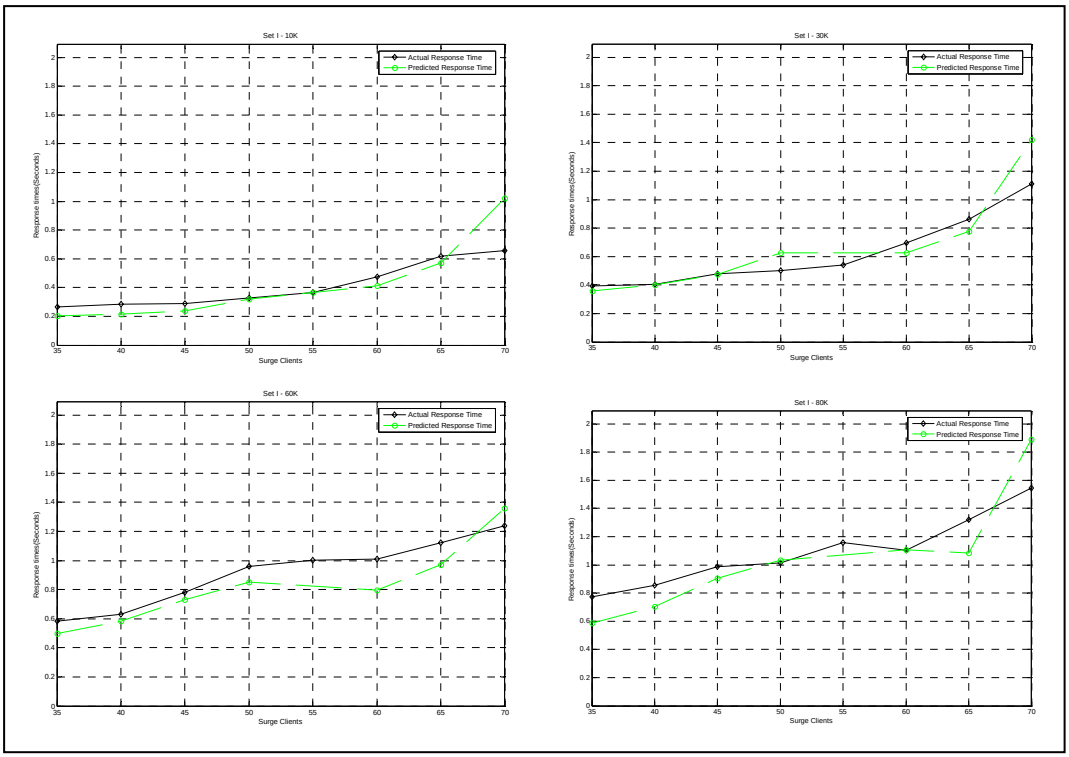

Figure 7. Set1 experiment results

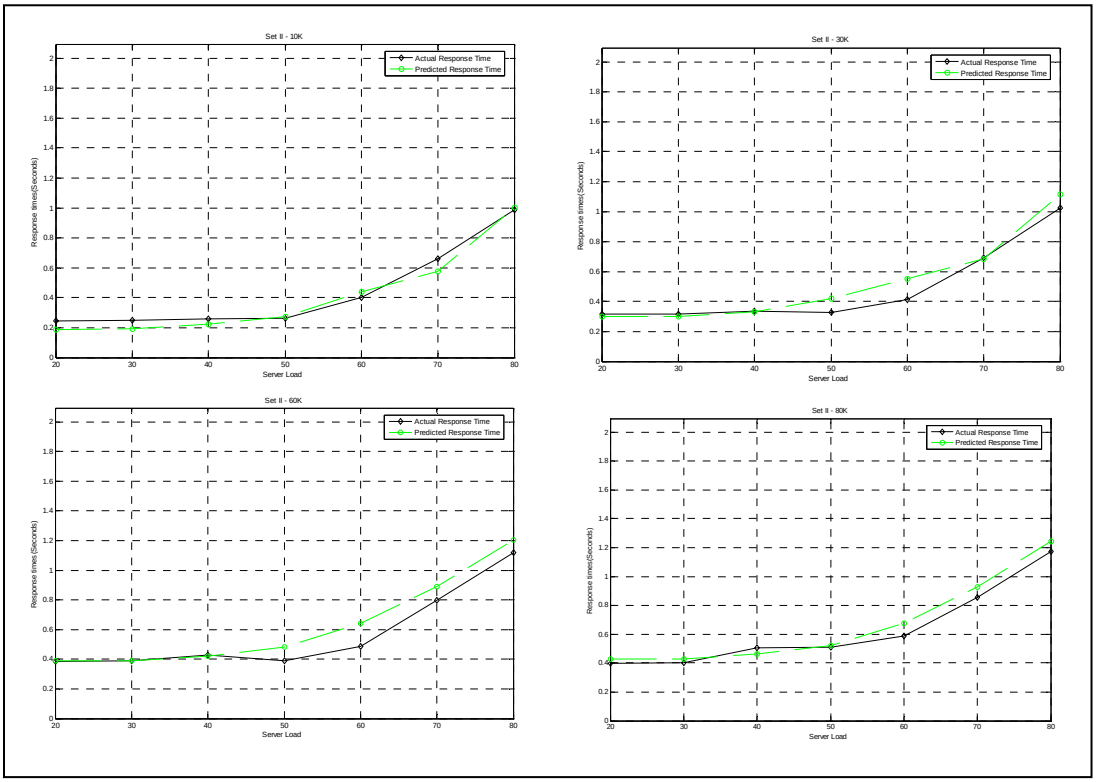

Figure 8. Set2 experiment results