Performance Modeling of Distributed Multi-Tier Enterprise Systems

Abhishek Dubey
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN

Rajat Mehrotra
Electrical and Computer Engineering
Mississippi State University
Mississippi State, MS

Sherif Abdelwahed
Electrical and Computer Engineering
Mississippi State University
Mississippi State, MS

Asser Tantawi
IBM TJ Watson Research Center
Hawthorne, NY

1. INTRODUCTION

Management techniques for achieving self-adaptive behavior in enterprise computing systems has been recently investigated by both academia and industry. Primarily, these techniques observe application measurements and take corrective action, based on a given model, to achieve a specified Quality of Service (QoS). Examples of such approaches include task scheduling, CPU provisioning [1], and power management [2].

Effective management of enterprise systems require fine-grained decisions that balance various, often conflicting, service level agreements while adjusting to changes in the operating environment, caused by factors such as time-varying user workload and incomplete knowledge of the system operating state. Such decisions, affect how an application behaves under different workload intensities [3]. Over time, they influence system reliability, energy efficiency and customer experience. These decisions are implemented through a control system, requiring a detailed model of the system that reflects how various management choices affect the system behavior under varying environment inputs and operating conditions. An example of such a model is the layered queuing model presented in [4].

In this paper, we present the results of recent experiments aiming to model a distributed multi-tier enterprise application. Using dynamic regression and queuing modeling techniques, we have been able to obtain an approximate model structure that captures the system behavior under various operation conditions with a high degree of accuracy. In future, these models will be used to implement a distributed control structure that optimizes the system parameters for a given set of objectives [5].

This paper is organized as follows. Section 2 discusses the experimental setup. Section 3 discusses the performance management objectives, and discusses our experiments and modeling efforts. Section 4 concludes this paper.

2. EXPERIMENTAL SETUP

Computing Resources: Our experimental testbed consists of three heterogeneous nodes (machines) called *Nop01*, *Nop02* and *Nop03*. Nop01 has 2 quad-core 1.9 GHz CPU with 8 GB of RAM. Nop02 has 1 quad-core 2 GHz CPU with 4 GB of RAM. Nop03 has 2 quad-core 2 GHz CPU with 8 GB of RAM. We use the open source version of IBM's J2EE middleware, Web sphere Application Server Community Edition (WASCE) with MySQL for deploying enterprise applications. Xen hypervisor (http://www.xen.org/) was used to create and manage the resources available to a cluster of virtual machines (VMs) over the given physical machines. It allows us to control the number of virtual machines and corresponding resource budgets.

Enterprise Application: As a representative application, we have chosen to use *Daytrader* -a benchmark application used to measure the performance of multi-tier installations. It allows users to monitor their stock portfolio, inquire about stock quotes, and buy or sell stock shares. The source code also comes with a client application that drives a session-less trade scenario and measures the response time. We have modified this scenario servlet in order to shift some of the processing load of a request from the database node to the computing node. This was done to emulate a mix of business enterprise loads.

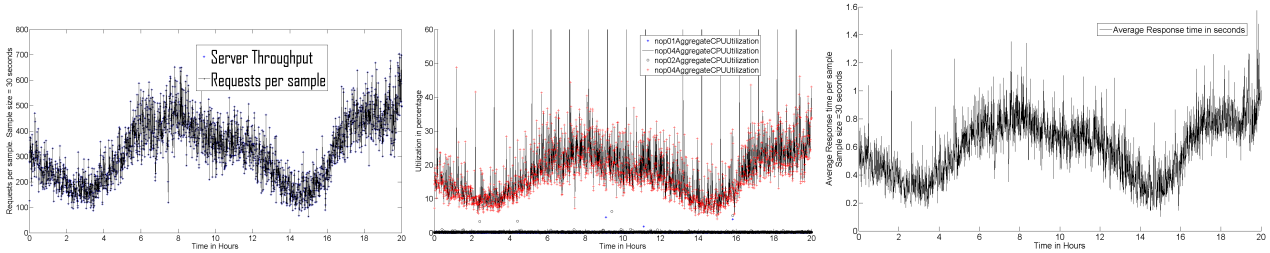
The daytrader application was deployed across multiple instances of WASCE running on different virtual machines. Multiple instances of the same application across different machines (physical or virtual) are collectively called a *Cluster*. All instances belonging to the same cluster share a common instance of MySQL server running on a separate virtual machine. All instances receive a fraction of the incoming client requests to perform load balancing. Several service classes can be implemented using Daytrader with minor changes to the source code. The differences between such classes are based on different quality of service expected for each of them.

Clients: Customer request were generated using a modified version of day trader client. This modification allowed us to generate exponentially distributed workload requests with request rate per sample specified as a look up table.

3. PERFORMANCE MODELING

Multi-tier applications are typically required to achieve a set of objectives while adjusting to changes in their operating environment. Effective operation of such systems depends directly on available resources, the configuration of computation platform, and the characteristics of the underlying operating environment. The current modeling effort aims to address the following performance management objectives.

Cluster Allocation. The objective is to determine (i) the number of instances of the application that should be in the cluster, (ii) determine the placement of all cluster instances on the machines, (iii) determine the budget of computing resources that can be consumed by a cluster instance (deployed in a virtual machine) such that response time service level agreement is satisfied. The last two items are controlled using a hypervisor such as Xen.



(a) Generated requests and the corresponding server throughput (b) Aggregate CPU utilization (c) Response time in seconds

Figure 1: Experiment 1. Sampling period=30 seconds.

Load Balancing. Determine the share of the requests served by any one of the application instance in a cluster such that the load across all instances remains uniform. This is required to ensure small variation in total response times.
Power Management. Determine the set of machines that can be either turned off or set to low power mode to ensure that the data center meets the stipulated power budgets.

3.1 Quality of Service Modeling

We have collected several experimental data in order to develop performance models that will allow us to predict the response time for a service class, given the request rate and the cluster allocation map. During these experiments, sensors across computing nodes were kept synchronized using a feedback technique described in [6].

Experiment 1: In this experiment, we setup the webserver on virtual machine Nop04 (physical machine Nop01). The database was setup on another virtual machine Nop05 (physical machine Nop02). Xen enables us to monitor the resource utilization of both virtual machines in run time. It also allows us to limit the resources available to these machines. Using Xen, we limited the maximum memory available to both virtual machines to 1000MB and limited their maximum CPU share to 25 percent of a single physical core.

Fig. 1(a) shows the generated client request rate. The average response time and average request rate was measured over a sample of 30 seconds duration. This trace is based on the customer request traces from the 1998 World Cup Soccer web site [7]. Fig. 1(b) shows the corresponding CPU utilization as measured on respective VM and physical machines. The corresponding response times are shown in 1(c). Visually, we can see the correlation between the request rate and response time.

3.1.1 *Estimating the bottleneck tier*

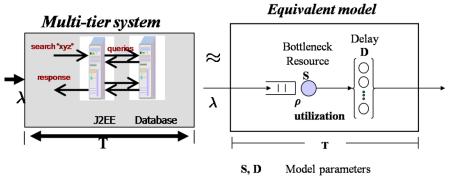


Figure 2: A queuing model for the two tier system

To determine the bottleneck resource, we used an open queuing approximation for a two tier system as shown in Fig. 2. λ is the incoming throughput of requests to an application. ρ is the utilization of the bottleneck resource. S is the average service time on the bottleneck resource. D is the average delay. T is the average response time of a request. Average waiting time for a request $W = T - S - D$. We took two different approaches for identifying the bottleneck resource for the experiment described in previous section. In the first case, we used a M/M/1 model with the assumption that CPU on the web server is the bottleneck resource. In the second case, we used a M/G/1 model with no assumption about the bottleneck resource. For both approaches, we aggregated eight data samples into one data point to smooth out the spikes in the data.

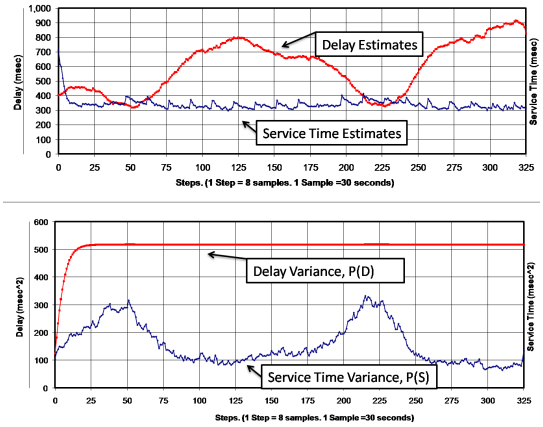


Figure 3: Estimated service and delay (msec) with variance estimate for CPU as the bottleneck.

CPU bottleneck: We used a M/M/1 queue model for nop04 (webserver) CPU as the bottleneck resource with service time (S) and a non-CPU delay (D). Thus, our state vector is [S D]. The observation vector is [T ρ]. For a given timed index of observation, k, the equations $\begin{pmatrix} \hat{S}_k \\ \hat{D}_k \end{pmatrix} = \begin{pmatrix} S_{k-1} \\ D_{k-1} \end{pmatrix} + Z(0,Q)$ and $\begin{pmatrix} T_k \\ \rho_k \end{pmatrix} = \begin{pmatrix} \frac{1}{\frac{1}{S_k} - \lambda_k} + D_k \\ \lambda_k S_k \end{pmatrix} + V(0,R)$ were used as the prediction and correction steps for an Extended Kalman Filter. Z, V are Gaussian noises with mean zero and covariances Q and R respectively.

Results shown in Fig. 3 indicate that the assumption of CPU being the bottleneck resource is invalid. This is because given a CPU utilization, the service time should have been in the range of 15 to 20 msec. Given that the CPU utilization was lightly loaded (between 10% and 30%), the waiting time was small and therefore did not capture the strong relationship between the average response time and the throughput. Hence, the delay estimate collected the re-

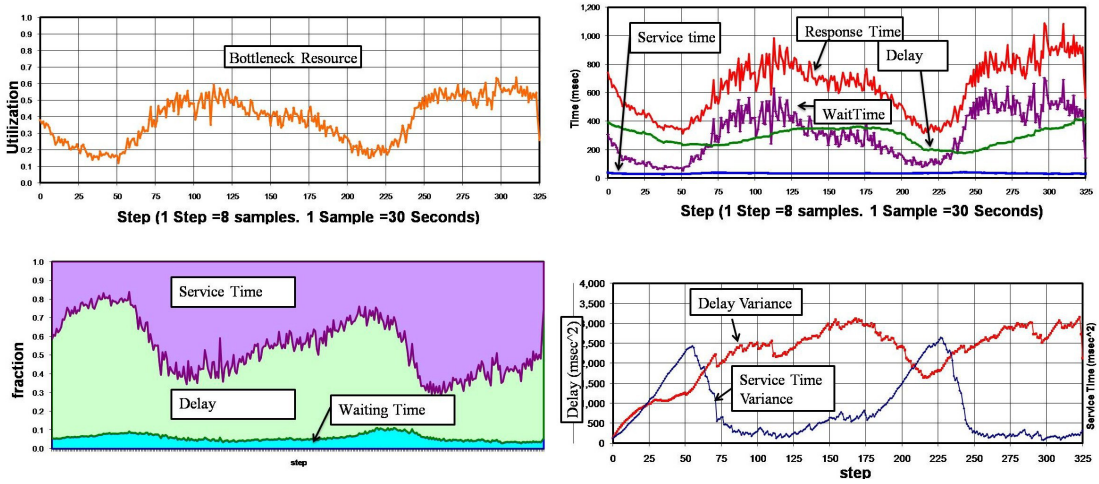


Figure 4: Estimated service and delay (msec) with variance estimate for the unknown bottleneck.

mainder of the response time (delay = response time - (service time + waiting time)). The delay estimate D varied quite a bit and had a high variance.

Unknown Bottleneck: We used a M/G/1 queue model assuming an unknown and not necessarily the CPU bottleneck. Our state vector was [S D]. The observation vector was the average response time, T. We assumed a coefficient of variation for the service time equal to 5. The predict equation for the extended Kalman filter was still the same as in the previous paragraph. However, the update equation was changed to $T = S(1 + \frac{1+C_b^2}{2(1-\rho)}\rho) + D + V(0, R)$, where C_b is the coefficient of variation for service time. Fig. 4 shows the results. The estimate of service time looks more promising compare to the previous results. The bottleneck utilization was estimated to be in the range of 15% to 60 %. Upon further investigation, the maximum number of threads available in the JAVA web server seems to be a likely candidate for being the bottleneck resource.

3.1.2 *Modeling Power Consumption*

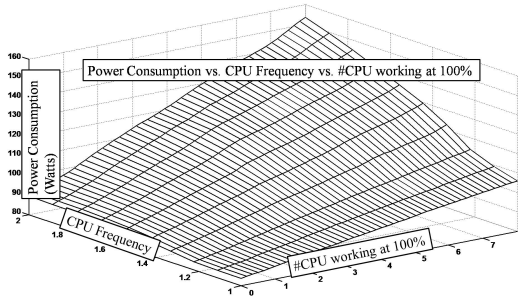


Figure 5: Power consumption on Nop03 vs CPU frequency and the no. of cores at 100% utilization.

We used a real-time watt meter to monitor the power usage through the servers, and to identify the relationship between power consumption and CPU core usage as well as CPU frequency. Fig. 5 shows the power used on one of the nodes, Nop03 (see Section 2). Based on such experiments we developed a regression model that correlates the load on the machine with the power consumed and current CPU frequency. The corresponding regression models are $power = a + b*numcore + c*freq$ and $a + b*numcore*freq$, with R^2 equal 0.91 and 0.92, respectively.

4. FUTURE WORK AND DISCUSSION

In this paper, we presented an approach for performance modeling of multi-tier enterprise applications using regression analysis of system measurements corresponding to a discretized domain of operating settings. This approach was demonstrated on a two-tier benchmark enterprise application. Experimental results shows that the generated models can accurately predict system behavior and accordingly can support a variety of model-based management techniques.

We are currently investigating the use of limited processor sharing (LPS) queues [8] for modeling web servers. In an LPS model, the server is shared equally by the jobs in service subjected to maximum limit on the number of jobs that can be in the system at a time. In future, we will integrate this system with a limited lookahead control framework [5] that can optimize the system parameters for the set of objectives specified in section 3.

Acknowledgment This work is supported in part by the NSF SOD Program, contact number CNS-0804230.

5. REFERENCES

[1] Chenyang Lu et al. Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal of Real-Time Systems*, 23(1/2):85–126, 2002.

[2] T. Simunic and S. Boyd. Managing power consumption in networks on chips. In *Proc. Design, Automation, & Test Europe (DATE)*, pages 110–116, 2002.

[3] J. Wildstrom et al. Towards self-configuring hardware for distributed computing systems. In *Proc. ICAC*, 2005.

[4] Yixin Diao et al. Modeling differentiated services of multi-tier web applications. In *MASCOTS '06: Proceedings*, pages 314–326, 2006.

[5] S. Abdelwahed et al. Online control for self-management in computing systems. In *Proc. RTAS*, pages 365–375, 2004.

[6] Compensating for timing jitter in computing systems with general-purpose operating systems. In *ISROC*, 2009.

[7] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup web site. Technical Report HPL-99-35R1, Hewlett-Packard Labs, September 1999.

[8] Jiheng Zhang and Bert Zwart. Steady state approximations of limited processor sharing queues in heavy traffic. *Queueing Syst. Theory Appl.*, 60(3-4):227–246, 2008.