A Methodology and Tool for Performance Analysis of Distributed Server Systems

Rukma Prabhu Verlekar, Varsha Apte
Department of Computer Science and Engineering
Indian Institute of Technology
IIT Bombay, Mumbai 400 076, India
rukma@cse.iitb.ac.in, varsha@cse.iitb.ac.in

ABSTRACT

We present a methodology and tool for performance analysis of distributed server systems, which allows high-level specification of the system, and generates and solves the underlying queueing network model. Our approach is different from the existing ones in that the specification captures the natural manner in which application servers are deployed on machines and machines are deployed on networks. The model does not impose any strict tiers on the server system. Multiple use case scenarios can be specified, and the tool computes measures such as end-to-end response times for each scenario while taking into account queueing delays at the hardware device, software threads and at the network. The development of the tool is ongoing, and will include detailed network protocol models as well as more flexible distributed system behavior, in the future.

Categories and Subject Descriptors: D.2.8 **Software Engineering** Metrics [performance measures, tools]

General Terms: Performance, Design

Keywords: Software performance models, queueing network models, distributed systems.

1. INTRODUCTION

Most on-line services of today are provided using heterogeneous distributed server systems. Such systems consist of a multitude of software servers, deployed on hundreds of different types of machines, which could be distributed on a LAN or a WAN. Users of such services now also expect a certain minimum performance from the transactions they perform on such systems. These requirements are often codified in the form of *service level agreements* (SLAs). Although models such as queueing networks can be used to predict whether a given service will meet its SLA, the complexity of the system is such that it is nearly impossible to do so manually. Thus, tools are required that can translate software and hardware design into performance models. The use of such tools during software development and during deployment of software onto server machines can result in significant performance improvement or cost savings.

Over the last two decades, a lot of effort has been directed towards developing methods and tools that will result in tighter coupling of performance engineering with software engineering [1]. The early efforts in this direction were focussed on queueing models that would appropriately capture the contention at software as well as hardware resources [5]. However efforts are now directed more towards automated translation of software design specifications to performance models [2, 4]. E.g. Menasce [2] presents an extremely comprehensive approach to build a performance model of a client-server system, and models such fine details as database accesses, internal program behavior along with many hardware architectural details such as resources, machines and network topology. However, the scope of the methodology seems to be limited to two or three tier architectures. In contrast, there are fairly generic tools such as SPE·ED [6] and UCM2LQN [4]. However, these do not include rich specification capability for hardware and network architecture.

In our work, we have attempted to meet the need of being able to specify software and hardware design in a simple and intuitive manner. E.g., we allow specification of types of "server machines". A number of different types of machines and also the different types of the software servers that can be deployed on these machines, can be specified. The tool then uses this information to figure out the contention on the devices of these machines. We maintain the feature, offered by most tools, of specification of multiple use case scenarios, which can include synchronous as well as asynchronous calls. We allow machines to be distributed across networks, while abstracting WANs as simple point-to-point links. The resulting model is solved analytically to derive various performance measures. We do not impose any strict layers among the software servers. Our methodology has been coded into a tool, that takes a textual input as specification and generates and solves the underlying performance model. To our knowledge, no existing tool captures the hardware, software and network details of the system in the flexible and natural manner as allowed by our tool.

The development of the tool is currently in progress. When complete, we plan to capture further details of network protocol behavior, and sophisticated queueing models for various queueing disciplines, etc. At this stage, the tool makes several simplifying assumptions regarding the net-

ICSE'06, May 20–28, 2006, Shanghai, China.

work and abstracts all resources with simple queuing models. However, we believe our approach is fresh enough to be reported at an intermediate stage of completion. In the next section, we will describe our system model, methodology and an example with preliminary results. In Section 3 we conclude with discussions and future work.

2. OUR SYSTEM MODEL

Our distributed system model consists of various software and hardware components. We list them below in a "bottom-up" manner.

1. *Hardware Devices:* At the lowest level, there are hardware devices which are parts of server machines (e.g. CPU). Each such devices can be declared, by giving each type a symbolic name which is used later for referencing it.
2. *Server Machines:* We can define different "types" of server machines, and specify how many of each type there are in the system. One machine is distinguishable from another in terms of the types and the quantity of the various hardware devices that it possesses.
3. *Network Architecture:* We allow our system to consist of several LANs that can be separated by point-to-point links. For each such point-to-point link we specify the values of transmission rates, propagation delays and the Maximum Transmission Unit (MTU). The server machines in the distributed system can be mapped onto different LANs. Multiple machines of the same type can be independently mapped to the various LANs.
4. *Software Servers:* Each of the software servers in the distributed system is specified with its corresponding set of tasks and the number of threads it possesses. A task is assumed to be a continuous piece of computation, which holds a device of its host machine, for a specified amount of time. The mapping of these software servers to the corresponding server machines and the execution times of each task of a software server on each device of the system is specified.
5. *Use Case Scenarios:* The interaction of the software servers to provide a variety of services is specified through a formalism similar to message sequence charts. Each use case scenario consists of an ordered sequence of tasks of various software servers, with certain annotations. Once a thread executes a task locally, it may make a synchronous or asynchronous call to a task of another server. We allow probabilistic branching to reflect different paths that may be taken in a single scenario. We also have a notion of a message sent when a task makes a remote call. Thus each such call is annotated with a SYNC/ASYNC tag, and a message size in bytes. The use case scenario itself is annotated with a symbolic name and its corresponding arrival rate.

2.1 The Analytical Modeling Approach

For the system model described above, there are several performance measures of interest such as the end-to-end response times of use case scenarios, or utilizations, queue lengths and response times at specific servers or devices. We calculate these measures analytically by converting the system model to a queueing network model, and employing judicious approximations wherever necessary. In this section, we describe our analytical modeling approach.

The main exercise in the analytical approach is to identify the resources of contention in the system, model these as queues, and derive the corresponding queueing model parameters from the higher level input parameters. The queues in the system above are of three main types: queues for threads, for hardware devices, and for network links. The analysis needs to derive average arrival rate, and average "service time" at each of the queues. For example, the arrival rate to a software task takes into account all the use case scenarios that invoke this task. Similarly, arrival rates to network links will depend on rates at which calls are made to tasks across the network, which depends on the specified topology. Calculation of net arrival rate to each of the hardware and the software resources is illustrated in Section 2.2.

The holding times of each of these resources must be computed carefully. For example, hardware devices are held only for the amount of time that they are actually being used. However, the holding times of threads are affected by contention at the device level and time spent in synchronous (or blocking) calls. Thus, the actual holding time of the thread that executes a task on a device includes the waiting time of this task at the device queue. Requests to the hardware devices come from different threads of different servers, and this contention is modeled by computing the overall arrival rate to a hardware device and an overall weighted average of execution times. This is used in the device queueing model to compute the waiting time at the device. The holding times of the threads (that include local execution time and time spent in waiting for synchronous calls) are in turn used to compute waiting times in the thread queues.

This approach is essentially very similar to the L layered approach proposed by Rolia and Sevcik [5]. However, we do not impose a strict layering architecture among the software entities. In that sense we have only two "layers": the software layer, and the hardware layer.

The network delay of a message is calculated as follows: we first compute the effective arrival rate and the average packet size coming to the specified network links, from all servers and machines deployed on that LAN and employ a simple queueing model to compute packet waiting times. If a message has n packets, then message delay across a link is estimated to be the sum of transmission times of all packets, propagation time of the last packet and the waiting time of the first packet. In a use case scenario, wherever calls are made across different software servers, we determine whether the call was also to a different machine. If it was, then we add the network delay thus computed to the response time of the scenario (network requests are assumed to be non-blocking).

2.2 Example

In this example, we illustrate the use of the above methodology for the calculation of performance measures of an E-mail system, which comprises of several use case scenarios such as login, read message etc.

Our example system consists of three software servers namely the Web, the Auth and the IMAP server, and two machines of type "SUN" and "HP". The HP is configured with dual CPUs and dual Disks, the SUN with four CPUs and four Disks. The two machines are deployed on two different LANs, denoted by LAN-1 and LAN-2 respectively. The Web and the IMAP server are deployed on the SUN machine whereas the Auth server is on the HP machine. The machines are connected by point_to_point links. Tables 1 and 2 show the software server parameters and the network parameters, respectively. For the sake of simplicity, we consider the Message Sequence Diagrams of only the lo-

Table 1: Software Server Parameters

Server description			Resource demands in milliseconds	
Server	**Threads**	**Tasks**	**CPU**	**Disk**
WEB	4	s_a_p	30	30
		s_a_s	80	20
		s_i	80	20
		c_h	10	20
AUTH	2	v_p	72	20
		v_s	10	20
IMAP	4	r_m	70	30
		l_m	20	70

Table 2: Network Parameters

MTU		Trans rates (Mbps)/ prop delays (millisec).		
Across LANs	512 bytes		LAN-1	LAN-2
On LAN-1	1500 bytes	LAN-1	3Mbps 2ms	3Mbps 3ms
On LAN-2	1500 bytes	LAN-2	3Mbps 4ms	3Mbps 2ms

gin and the read message scenarios (Figure 1). In the login scenario the user sends a request to the Web server with a user name and password. The thread handling this request executes task *s_a_p*, and then makes a synchronous call to the task *v_p* ("verify password") of the Auth server. If the authentication fails (probability 0.1), a reply is sent back to the Web server thread which executes task *c_h* locally and sends a reply back to the user. In the case where the authentication succeeds, the task *s_i* is executed locally, and the "list message" task (denoted as *l_m*) of the IMAP server is called synchronously. After a reply is received, task *c_h* is executed by the Web server thread locally, and this ends the server-side flow of the request. The read message scenario is very similar to this scenario.

To calculate measures such as the end-to-end response time our first step is to calculate the effective arrival rates to the software tasks. For a login arrival rate of 10 requests per second and read arrival rate of 5 requests per second, it is clear, for example, that arrival rate for task *r_m* is 4, for task *l_m* is 9, while task *c_h* is 15. This calculation is carried out for all tasks.

The next step would be to estimate the thread holding times. E.g., for the unsuccessful login scenario, the Web server thread holding time will be sum of the response time of the task *s_a_p* at the CPU and disk of the SUN machine, the time it waits till it receives a response from task *v_p* of the Auth server (which will include network delays) and again the time for the execution of task *c_h* on the devices of SUN machine. Initial estimates of waiting times at hardware devices are set to zero.

The contention at the devices is now to be modeled. The arrival rates to each device is the sum of the throughput of all the software tasks which contend for it. In this example, the total arrival rate to the CPU of the SUN machine is the sum of the throughput of the Web server tasks *s_a_p*, *s_a_s*, *c_h*, *s_i* and the throughput of IMAP server tasks namely *r_m* and *l_m*. We calculate the effective service time at the device as the weighted average of "raw" execution

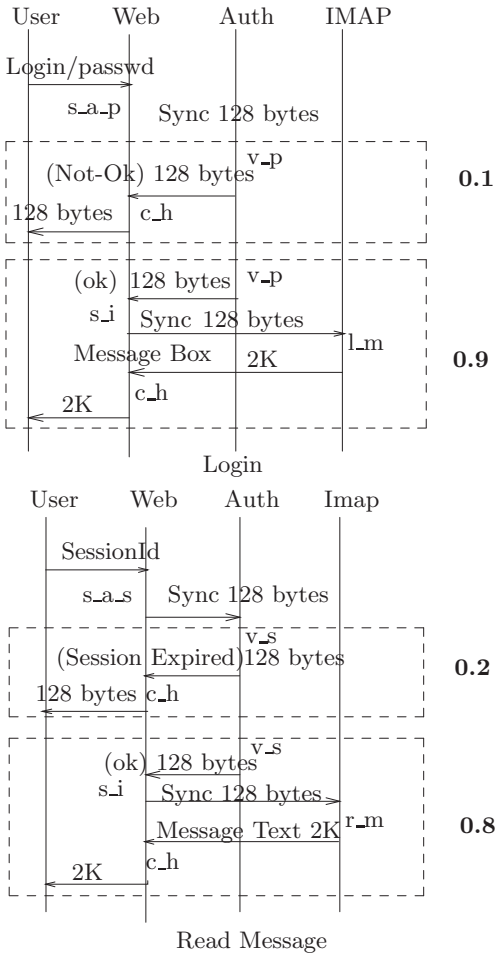

Figure 1: Login and Read Message Scenarios

times of all the above mentioned tasks and their throughput. Using the total arrival rate at each of the device and the effective service times of the device we can calculate the waiting times of each task on this device. The total response time of the task for the local computation is then the sum of the response times of the task on each of the devices to which it makes a request. For example the total response time of *s_a_p* on the SUN machine is its response time at the CPU as well as the Disk of SUN machine.

The hardware and software layer models are interdependent,and are solved iteratively until convergence is achieved. Interdependence in the model is due to the fact that arrival rates at the hardware layer are derived from the software layer throughputs and the thread holding times depend on the response times of the tasks at the hardware layer.

In case all calls are synchronous (as shown), the response time of a scenario is simply the response time at the Web-server thread queue of a request of that scenario.

In case all calls are asynchronous, the delays at each software task and at each network link can be added up according to the sequence shown in the use case scenario, to arrive at the end-to-end response time of that scenario. The details of the analytical modelling technique can be found in [7].

2.3 Results

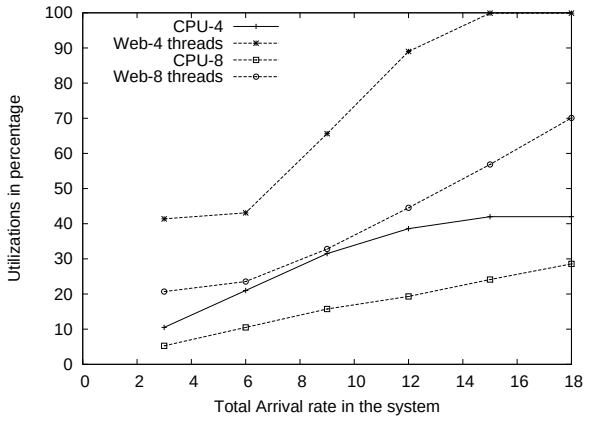

Figure 2: Hardware/Software Utilization Vs. Arrival Rate

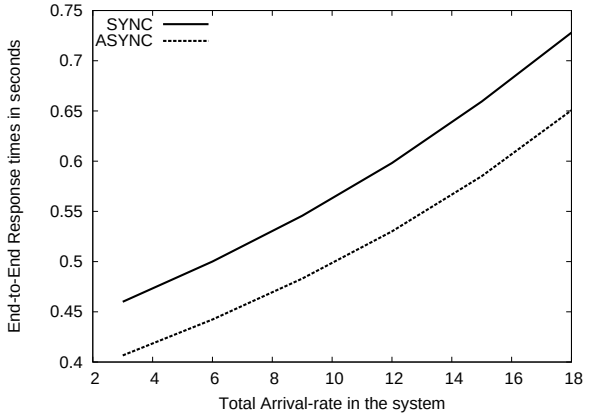

Figure 3: Response time Vs. Arrival Rate

In this section we present some of the results from our modelling tool, with the above mentioned example parameters as the input. The modelling tool implements the analytical methodology discussed in the previous section. Figure 2 shows the utilization of the Web server and of the CPUs of SUN machine (labelled as Web-4 threads, CPU-4 threads), when all calls are synchronous. The arrival rate shown is total, where the mix of login and read scenarios is assumed to be one read request per two login requests. Let us assume that we want to determine whether this system can support a total arrival rate of 15 requests per second. The Web server reaches 100% utilization at 15 requests per second, indicating that this system cannot support this load. We change the configuration of the SUN machine to have 8 CPUs, 8 Disks, and 8 Web server threads. The utilizations of the Web server and the CPUs (labelled as Web-8 threads and CPU-8, respectively) are now below 60% at 15 requests per second.

For the upgraded configuration, Figure 3 shows the average end-to-end response time of the system with the login and read scenarios, for the case where all remote calls are synchronous, vs. when all calls are asynchronous. As expected, for the same configuration, synchronous calls result in higher end-to-end response times.

3. DISCUSSION AND FUTURE WORK

As illustrated with the E-mail application example, our specification formalism is simple and emulates the way deployment decisions are made (i.e. software servers on machines). We believe that what-if scenarios can be run in a very intuitive way using our specification.

Our analytical methodology imposes no constraints on the interactions between software servers–e.g. software servers do not have to belong to any particular "layer". Any server thread can remotely call any other server thread in a synchronous as well as asynchronous manner.

We believe that because of the intuitive specification formalism, we are on the path of developing a tool that would be useful to software developers as well as data center designers. However, the tool has several drawbacks, that we hope to remove. E.g. the tool requires details of execution times by tasks on various hardware devices. We are addressing this problem by developing related tools, e.g. for automatically deriving resource consumption profiles of transactions [3]. A simulator is being developed, which takes the same input specification as the analytical modelling tool. The results of simulation will be used for the validation of our results.

In the future, we propose to develop a more robust system to model various software resources such as log files, critical sections etc, and relevant details of common network protocols such as TCP and HTTP. We also plan to develop algorithms for suggesting configurations that would meet the user specified performance constraints.

4. REFERENCES

[1] S. Balasmo, A. D. Marco, and P. Inverardi. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5), 2004.

[2] D. A. Menasce and H. Gomaa. A method for design and performance modelling of client/server systems. *IEEE transactions on software engineering*, 26(11), November 2000.

[3] B. Nagaprabhanjan and V. Apte. A tool for automated resource consumption profiling of distributed transactions. In *International Conference On Distributed Computing and Internet Technology*. Springer-Verlag, December 2005.

[4] D. Petriu and C. Woodside. Software performance models from system scenarios in use case maps. In *12th Int. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, 2002.

[5] J. Rolia and K. Sevcik. The method of layers. *IEEE transactions on software engineering*, 21(8), August 1995.

[6] C. U. Smith and L. G. Williams. Performance and scalability of distributed software architectures: An SPE approach. *Parallel and Distributed Computing Practices*, 3(4), December 2000.

[7] R. P. Verlekar and V. Apte. A methodology and tool for performance analysis of distributed server systems, CSE Department, IIT Bombay, http://www.cse.iitb.ac.in/∼rukma/papers/lqm.pdf, Feb 2006.