PerfCenter: A Performance Modeling Tool for Application Hosting Centers

Akhila Deshpande
Department of Computer
Science and Engineering
IIT Bombay
Powai, Mumbai-400076,
INDIA
uakhila@cse.iitb.ac.in *

Varsha Apte
Department of Computer
Science and Engineering
IIT Bombay
Powai, Mumbai-400076,
INDIA
varsha@cse.iitb.ac.in †

Supriya Marathe
Department of Computer
Science and Engineering
IIT Bombay
Powai, Mumbai-400076,
INDIA
supriyam@cse.iitb.ac.in

ABSTRACT

We present a tool, *PerfCenter*, which can be used for performance oriented deployment and configuration of an application in a hosting center, or a "data center". While there are a number of tools which aid in the process of performance analysis during the software development cycle, few tools are geared towards aiding a data center architect in making appropriate decisions during the deployment of an application. PerfCenter facilitates this process by allowing specification in terms that are natural to a data center architect. Thus, PerfCenter takes, as input, the number and "specs" of hosts available in a data center, the network architecture of geographically diverse data centers, the deployment of software on hosts, hosts on data centers, and the usage information of the application (scenarios, resource consumption), and provides various performance measures such as scenario response times, and resource utilizations. We describe the PerfCenter specification, and its performance analysis utilities in detail, and illustrate its use in the deployment and configuration of a Webmail application.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server,Distributed applications*; C.4 [**Performance of Systems**]: [Modeling Techniques]; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*

General Terms

Performance,Design

Keywords

Performance analysis, tool, data center

1. INTRODUCTION

Enterprise applications are typically hosted in central facilities called "application hosting centers", or more commonly, "data centers". Each time a new application is to be hosted in a data center, the data center architect must make several design choices. E.g.- does the application need additional machines? What should be the configuration of these machines? How should the application components be deployed on these machines? How should they be configured? If there are multiple data centers, connected through wide-area network (WAN) links, there would be a choice of possibly distributing some application components over the different data centers. All these decisions significantly impact the *performance* of the application and the *utilization* of the data center resources. Apart from the direct hardware costs, it is very important that data centers minimize the area as well as the energy footprint of the machines housed in them. Thus, over-provisioning, while an "easy" solution, may prove to be a costly one. At the same time, data center owners must satisfy the performance requirements of the clients whose applications they host, which may be expressed in the form of *service level agreements*. Therefore, data center architects need appropriate tools using which they can make deployment and configuration decisions which result in satisfactory usage of resources, while still meeting service level agreements.

A number of tools and methodologies have been proposed over the last decade and more, that address the basic question of how software systems perform [5]. All these techniques, in one way or the other abstract a multi-tier software application as a *queueing network*. The software processes that service requests are essentially queuing systems. Since a request has to pass through multiple such servers to be fulfilled, the natural abstraction is a queueing network. However, a key challenge is to model the several *layers* of resources that are present in distributed software systems. That is, the server resources themselves can be customers of other resources, which in turn can be customers of other resources, and so on, thus creating "layers" in the queueing network. E.g. a Web server thread is a server for the HTTP requests, as well as a customer for the CPU "queueing system".

Several approaches have been proposed to model such "layered" systems of hardware and software resources. The

early significant ones include the *Stochastic Rendezvous Network* model by Woodside et al [26] and the *Method of Layers* by Sevcik and Rolia [21]. Both techniques map the system to a layered queueing network. These approaches are notable for their *analytical* solutions of the models. The Stochastic Rendezvous Network model, also known simply as a *Layered Queueing Network (LQN)* has been particularly popular, with many extensions made to it over the years [19].

Some of these methodologies have been implemented as software tools. E.g. (LQNSolver [13]) implements the SRVN (or LQN) model. $SPE \bullet ED$ [24] is another important tool (commercially available) that promotes the "Software Performance Engineering (SPE)" process.

The work in recent years has focused immensely on *translation* of high-level specification, to the formalisms accepted by such queueing network models [19, 14, 15]. These approaches assume that a tool for the solution of the target model is available. Specifically, there has been significant amount of work in converting systems specified using the UML SPT (Schedulability, Performance and Timing) profile [18], to an queueing model, to be solved by an existing solver [16, 28, 23, 9, 6].

The existing tools and methodologies are primarily focused on the notion of making performance analysis convenient during the software *development cycle*. This explains the sharp focus on developing tools and methods that will work directly with UML, as this is the standard for modeling object-oriented software.

We feel, however that there is a need for a tool which is particularly convenient, when the potential user is a *data center architect*, whose focus would be on deployment and configuration of an application, given the resource constraints of the data center. A direct approach by which a "data center architecture" can be specified would be very convenient. In other words, we would like to specify the hardware architecture (such as how many machines are available, of what type), the network architecture (how the machines are deployed on LANs, and perhaps separated by a WAN), the software architecture (the message flows for various scenarios), and finally, the deployment and configuration details of the software and hardware. To our knowledge, none of the existing tools/methodologies focus on such specification.

We introduce an *open source, freely downloadable* software tool, called *PerfCenter* [1] which we believe is both intuitive and straightforward, and allows for convenient analysis of various deployment and configuration scenarios, which can help a data center architect. PerfCenter allows for easy and natural specification of the hardware, network and software architecture of a data center, and analyzes the performance of the application at given volumes. The specification language of PerfCenter supports various language constructs such as variables, expressions, and for and while loops so that analyzing various configurations becomes easier. PerfCenter generates and solves the underlying queuing network model using either an analytical solution or by *simulation*. The methodology of solving the PerfCenter model analytically has been presented in an earlier paper [25]. This paper focuses on the *usage* of PerfCenter: we explain the input language, and illustrate its usefulness for data center de-

sign with an example of deployment of a Web-based e-mail system. We show how the various analysis utilities of PerfCenter can be used to not only arrive at a good deployment and configuration, but to also provide insight into the why the system performs in a certain way. For this last reason (and because it is free, and modify-able), PerfCenter can also serve as a teaching tool for the subject of software performance analysis.

The rest of the paper is as follows: we review related work in this area in Section 2, introduce PerfCenter in detail in Section 3, and then illustrate the use of PerfCenter in evaluating various deployment and configuration alternatives, using an example in Section 4. We conclude the paper in Section 5.

2. RELATED WORK

A large amount of research has been done over the last decade to address the problem of analyzing the performance of software systems. Much of the work consists of methodologies and theoretical proposals of solutions. Some of the solutions have been implemented as tools. We review some of the theoretical work, as well as tools based on various methodologies, in this section.

The Method of Layers [21] was among the early approaches, which proposed a model and an analytical solution for capturing the "layered" behaviour of software systems - i.e. where one "layer" of servers uses the services of another layer of servers, where the lowest layer is that of the hardware devices. All calls between layers were assumed to be *synchronous*.

The Stochastic Rendezvous Network [26] is a powerful model, which captures many interesting behaviours of a distributed system. Apart from synchronous calls to servers at different layers, the model allows execution of a service in phases. This allows capturing of post-processing that is done by a server, even after a reply is sent back to the caller. Some extensions to the "power" of the model have also been made (e.g. adding the capability of modeling forks and joins [19]) in the processing of the request.

Another significant body of work has been done by Balsamo et al [3, 4, 7] who have used multiclass queueing networks with finite capacity queues to model software architectures, including those that allow synchronous calls between servers.

Some amount of work has also been done in developing more accurate (as opposed to generic) models of a system, so that more detailed behaviour can be specified and analyzed [12], or in developing models specific to certain software technologies [20, 27].

Some of the performance modeling methodologies have been implemented as software tools. The solution method of the SRVN model has been implemented in a tool called the LQNSolver [13]. The tool $SPE \bullet ED$ [24] is a commercially available product that offers a variety of features that support what is termed as "Software Performance Engineering (SPE)". SPE facilitates the integration of the process of performance analysis into the software development cycle . The tool accepts a "software execution model" and a "system execution model" specification and provides the necessary performance measures of the system.

Recently, much of the effort has been directed towards mapping standard specification formalisms to the input that these tools work with (e.g. translating Use Case Maps to

LQNs [19]). Much recent work has focused on the UML-SPT profile [18] as it is being accepted as a standard for specification of performance parameters of an distributed, object oriented system [16, 28]. Additionally, some papers have proposed performance specification languages, that can serve as an intermediate formats, between UML and the language accepted by a specific modeling tool [14, 23].

A detailed survey of the work done in this area is out of the scope of this paper, and the reader is instead referred to the excellent review by Balsamo et al [5]. While a variety of tools seem to exist for the purpose of analyzing software systems, we claim that none of these are suitable for analysis from the "data center" point of view (with the possible exception of OPNET's IT Guru Systems Planner, which, however, is a commercial tool). We claim that the existing tools are ideal as tools to be used during development, when the details of deployment may not be available. However, at the stage when an application is ready to be deployed in a hosting center, the architecture of the data center (machine specifications and network architecture) starts playing an important part. A data center architect would require a tool that allows him/her to run various what-if scenarios of deployment and configuration, and see their effect on the performance of the application, and on the utilization of data center resources. The formalism we propose captures the natural "language" that data center architects speak - in terms of number of machines of a certain type, the "specs" of a type of machine, the "deployment" of a software application on a certain machine, and the deployment of the machines on various different data centers that might be separated by long distance network links. *PerfCenter* accepts specification at this level, and generates and solves the underlying queueing model which captures the complexity of the contention for resources.

In the next section, we present the elements of our system model, which will clarify how our formalism is suited towards the analysis of an application that is about to be deployed in a data center.

3. PERFCENTER

The PerfCenter system model attempts to formalize a "data center" roughly hierarchically, as a typical architect would. Thus we define host machines, the application servers, networks, and provide finer level details of the resources that constitute the host machines and the network. We also specify deployment and configuration details for the servers.[2] The deployment is specified in terms of servers on machines, and machines on LANs. The specification is provided using a simple text interface.

To understand the system model of PerfCenter in detail, we use the example of a Web based email application. The various software components required for this application are the Web server, the IMAP server, the authentication server, and the SMTP server. The application is hosted on one or more hosts. The hosts may be connected by a LAN, or may even be separated by WANs. This application can be used in various ways: we can login to the system, read messages, send messages, delete messages. We term these as "use case scenarios" or simply "scenarios".

Figure 1: Message sequence chart for the login scenario

Figure 2: Activity Diagram depicting the processing when a user logs in

---

[2]In this paper, we use the term "server" to imply the server process, e.g. the Web Server or the IMAP server. The "hardware" server is termed as *machine* or a *host*.
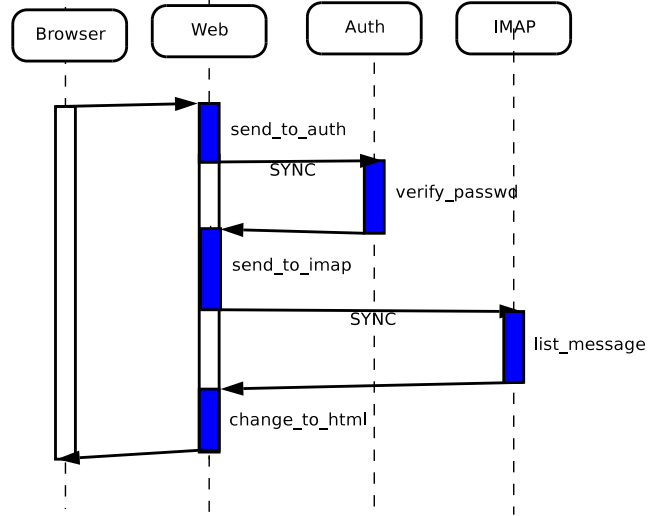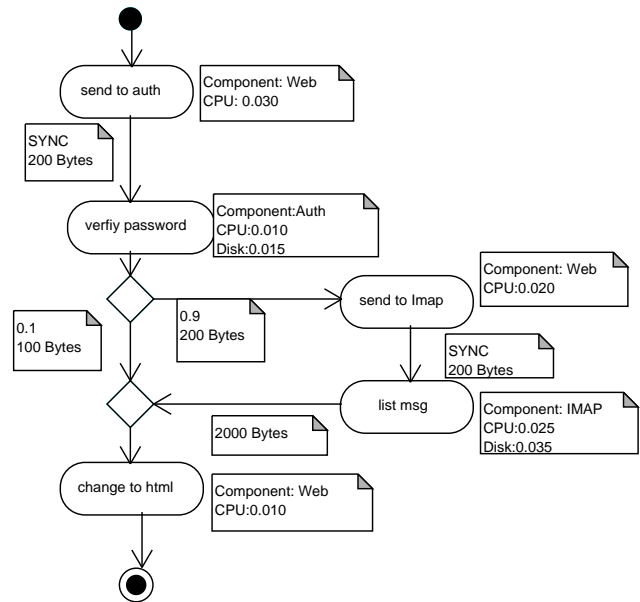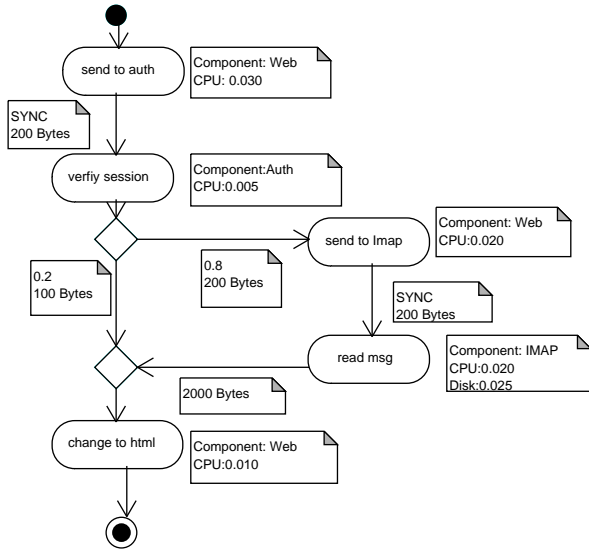
Figure 3: Activity Diagram depicting the processing when a user reads a message
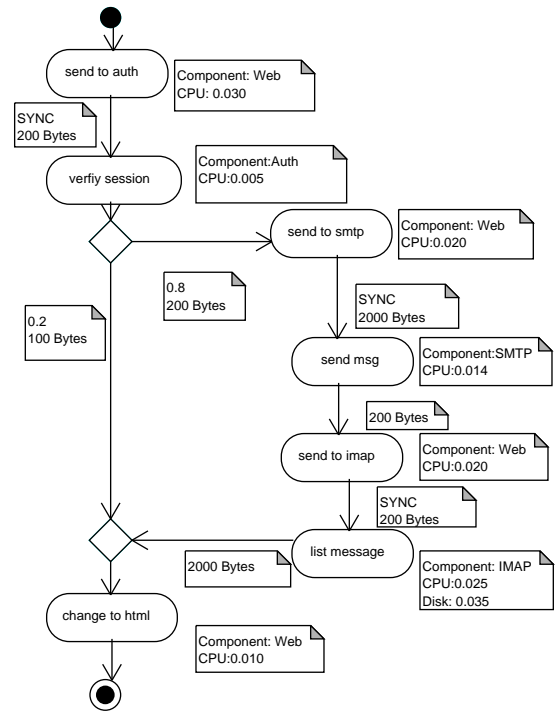

Figure 5: Activity Diagram depicting the processing when a user sending a message
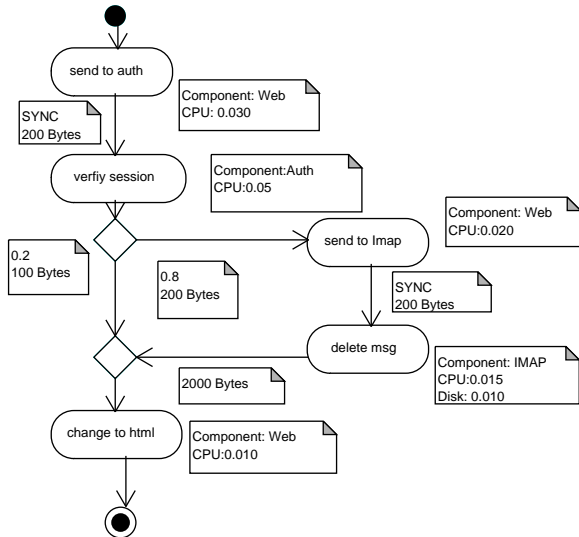

Figure 4: Activity Diagram depicting the processing when a user deletes a message

If such an application is to be deployed in a data center, we would want to arrive at a satisfactory deployment and configuration that would maximize the performance seen by the users, while minimizing the data center resources used. Specifically, in such an application, the *scenario response times* would be a primary measure of interest for the users. From the point of view of the application owner, the *capacity* of the system in terms of throughput or number of users would be of most interest. From the point of view of the data center architect, *optimal utilization* of resources is the top priority. All these *performance measures* depend on various factors: how each scenario uses the software components at various tiers, the resources consumed at each step of execution of the scenario, how the hardware resources are shared, and how the network resources are shared. PerfCenter allows for easy specification of all these factors.

The sequence of actions performed by the application components to execute a scenario can be expressed using *message sequence charts* [2] or *activity diagrams* [1]. Figure 1 shows how a login scenario would be performed on the system in terms of the processing to be done on the various servers. The figure depicts the message sequence chart corresponding to this scenario. To login, the user supplies his user name and password to the Web server. When this request arrives at the Web server, the threads of the server use the devices such as the CPU, disk of its host machine to process this request (indicated by a shaded rectangle labelled "send_to_auth"). We term this local processing as a *task* in our model. After processing the *send_to_auth* task, the Web server "calls" the authentication server to verify the user credentials. The authentication server also clearly requires the use of local devices to process this query ("verify_password"). On receiving the reply, the Web server again does some local

processing ("send_to_imap"). Assuming the user is authenticated, the Web server now calls the IMAP server, which returns a list of the messages that the user has received ("list_message"), which the Web server formats and displays to the user ("change_to_html"). The label "SYNC" on the arrows indicate that the calls made were *synchronous*. That is, the thread making the call is blocked until it receives a reply. On the other hand, in our model, an *asynchronous* call implies that the calling server thread simply forwards the request to the invoked server, without waiting for a reply. It then becomes free to pick up the next request for processing. When a reply arrives, say from the authentication server, it will queue again at the Web server for processing.

The same scenario can also be depicted using activity diagrams. Since it is easier to depict branches in activity diagrams, in the rest of the paper, we depict scenarios using activity diagrams. Figure 2 shows the activity diagram corresponding to the login scenario. In this case, we also show the execution path followed if the user credentials cannot be verified. Similarly, Figures 3, 4 and 5, and show the activity diagrams corresponding to the read, delete and send scenario respectively.

Note that, for a moment, if we ignore contention for hardware, the information required to estimate, say the scenario response times, would be: the probability that an arriving scenario is of certain type (e.g. login, or read), the execution time of the tasks (on all the devices that a task may use), the branching probabilities in the activity graph; the *server* characteristics such as number of threads, and buffer size (if any); and lastly the *load* on the system. The load may be either *open* (specified using scenario arrival rates) or *closed* (specified using number of users, and think time).

Now, let us consider the impact of contention for hardware resources, or equivalently, the problem of "sizing" the hosts, and deploying the software components on the hosts. Suppose we have four machines available for this application. Every machine can have different hardware. For example, a particular machine may have two or four CPUs, another machine may have CPUs of a different speed. How we deploy the Web, IMAP, SMTP and authentication server on these machines will impact the end-user performance, as well as the utilization of the machines.

Suppose that there are two data centers, on two LANs separated by a WAN, and suppose further that the authentication server must be housed in "Data Center 2". In this case, we have to answer the additional question of whether the WAN link between the two LANs has enough capacity for the new application, and how this affects the scenario response times. PerfCenter abstract the WAN as a point to point link and for that link, parameters such as the MTU (Maximum Transmission Unit), transmission rate and propagation time need to be defined. The MTU is specified so that the packet queue at the link can be modeled. The network delay would then depend on the size of the message sent from one server to another when a remote call is made. Figures 2- 5 show the average message size in bytes when a call is made from one server to another for processing a request. We assume that the network delay within a LAN is negligible.

Figure 6 shows a sample deployment of the machines on the network, and servers on machines that may be an "initial" deployment plan. The figure also shows the WAN link parameters, and the server configuration settings. It shows
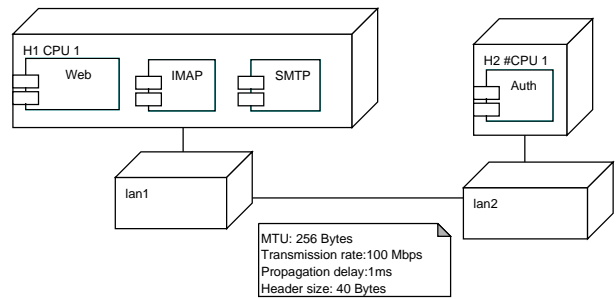

Figure 6: Sample Deployment of the email system

the authentication server deployed on host H2, Web, IMAP and SMTP on host H1 respectively. The architects would now require a performance modeling tool to improve on this deployment. Apart from the host and server deployment, optimal values for server configuration parameters such as the number of threads and/or the buffer size would have to be arrived at. In the following, we explain how this can be done using PerfCenter.

3.1 PerfCenter Specification Model

The previous section described the various components of system that can be specified to PerfCenter for analysis. This section explains how the specification translates into an input file for PerfCenter. We illustrate how the deployment in Figure 6 corresponds to the input file shown in Figure 7. The components of the input file are explained in the following list.

- **Variables**: The input file allows for use of variables to specify any numeric value. These variables have to be defined before actual use. This definition is done in the variable block.
  ```
  variable
  <var-1> <value>
  ..
  <var-n> <value>
  end
  ```
  Lines 1 to 10 from input file define the variables used.

- **Devices**: The devices present in the various machines are listed here. All the devices listed in this block may be used later in the input file as components of various machines.
  ```
  device
  <device-name-1>
  ..
  <device-name-n>
  end
  ```
  Lines 12 to 15 in the input file list the devices present for in the available machines.

- **Host**: For every type of machine in the system, we specify a name and a list of hardware devices present in the machine. Every device is further specified with a count, indicating the number of such devices present in the machine and the details of buffer size and scheduling policy used at the device. The scheduling policy can be LIFO, FIFO or PS. More complex user-defined scheduling policies can also be specified, as is explained in detail in the PerfCenter User Manual [11]. Device

definition could also mention relative speed of the device. The relative speed specifies how fast or slow the particular device is with respect to a normalized device used to calculate the service times of the task. Thus, a task needs only half the service time for processing on a device with speedup of two.

```
host <hostname> <[#this-type-machines]>
<device-type-1> count <#devices>
<device-type-1> buffer <buffersize>
<device-type-1> schedP <lifo|fifo|ps>
<device-type-1> speedup <relativespeed>
..
<device-type-n> count <#devices>
<device-type-n> buffer <buffersize>
<device-type-n> schedP <lifo|fifo|ps>
<device-type-n> speedup <relativespeed>
end
```

In the input file, lines 17 to 59 define hosts, including all the devices, and the device details within them.

- **Servers**: For each server, the number of threads, the buffer size and the scheduling policy followed at the server is specified. The various tasks that a server executes are listed in the definition.

```
server <servername>
thread count <#threads>
thread buffer <buffersize>
thread schedP <lifo|fifo>
task <task-name-1>
..
task <task-name-n>
end
```

Lines 61 to 93 describe the servers, along with their requisite details.

- **Task** : A task is a continuous piece of computation by a thread, on one or more devices of its host machine for a specified amount of time. In PerfCenter, we assume that a task uses the devices that it needs, serially (i.e. one after the other). Thus a task is the piece of computation that a thread would do up to the point that it makes a remote call to another thread, or up to the point that it exits, whichever is earlier. The computation done by the thread after the remote call "returns", is treated as a *another* task.

Every task listed in the server definition, is specified by a list of its execution times on the various hardware devices such as CPU, disk etc.

```
task <task-name>
<device-type-1> servt <servicetime>
..
<device-type-n> servt <servicetime>
end
```

Lines 95 to 137 of input depict the task definitions. The approximate execution times for a task on various devices can be obtained by using measurement tools such as AutoPerf[22] or estimating it from monitoring data [10], or by analyzing the code itself [17].

- **Network**: This block describes the network configuration by specifying the LANs present in the system. Every LAN in the system is declared as shown in lines 139 to 142.

```
lan
<lan-name-1>
..
<lan-name-n>
end
```

For every pair of LANs, the WAN connecting them is abstracted as a link whose parameters have to be specified. The parameters should include the Maximum Transmission Unit (MTU), the header size, the propagation delay and transmission rate.

```
link <linkname> <lanname1> <lanname2>
trans <transmissionspeed>
mtu <MTUsize>
prop <propagationdelay>
headersize <sizeinbytes> bytes
end
```

In the input file, lines 144 to 149 show how these parameters are defined for the link between LAN1 and LAN2.

- **Deployment**: There are two kinds of deployment; the deployment of machines over the network, and the deployment of servers over machines. To specify both these kinds of deployment, the keyword deploy is used.

```
deploy <server-name> <machine-name>
deploy <machine-name> <lan-name>
```

Lines 156 to 159 specify server deployment on machines, while lines 151 to 154 describe how the machines should be deployed on LANs.

- **Scenarios**: Scenarios represent the manner in which the system components interact with each other to provide service to customers. Figure 2 shows the login scenario in the form of an activity diagram. With every scenario, a probability is specified indicating the fraction of requests invoking that particular scenario. Within a scenario, we allow for probabilistic branching. This branching is done using keyword branch. Every branch is contained within a block starting with branch prob <branch-probability> and ending with end. We also allow for nested branches.

```
scenario <scename> prob <inv-prob>
<taskname1> <taskname2> <message-size> [SYNC]
..
branch prob <branch-probability>
<taskname3> <taskname4> <message-size> [SYNC]
..
end
..
end
```

Every activity diagram is translated to depict the scenario in a form that can be input to PerfCenter as shown in lines 161 to 210.

- **Load**: Load on the system can be specified in two ways. In case the system has open arrivals, an arrival rate value can be used to indicate the load.

```
loadparams
arate <arrival-rate>
end
```

Otherwise, if the system supports closed arrivals, the number of users can be specified using the keyword noofusers and the average time between receiving a response and sending a request by a user is specified

using the keyword `thinktime`. The keyword `exp` indicates that the `thinktime` is exponentially distributed.

```
loadparams
noofusers <#users>
thinktime exp(<average-think-time>)
end
```

Lines 212 to 215 describe the load on the system in input file.

3.2 PerfCenter Measures

PerfCenter provides performance measures in the form of response times, throughputs, queue lengths etc. Every device in the system, is identified as
`<machinename>:<devicename>`, and every server instance as `<machinename>:<servername>`. For every resource, we have the following functions:

- utilization - `util(<resname>)`
- throughput - `tput(<resname>)`
- average queue length - `qlen(<resname>)`
- average service time - `avgservt(<resname>)`
- average waiting time - `waitt(<resname>)`
- average response time - `respt(<resname>)`
- arrival rate - `arate(<resname>)`
- blocking probability i.e. the probability that an incoming request will be dropped at the queue due to insufficient buffersize - `blockprob(<resname>)`
- all parameters listed above - `qparams(<resname>)`

The functions `tput()`, `blockprob()` and `respt()` can also be used without passing any parameters, in which case they return the average scenario performance measures. If `scenarioname` is passed as input, the scenario performance can be obtained. The use of these functions has been illustrated on lines 227 to 236 in Figure 7.

3.3 Analysis Utilities

To carry out performance analysis conveniently, PerfCenter allows for constructs such as loops (for, while etc), keywords like set, and variables. Using loops, the tool can be used for efficient running of multiple design scenarios.

These analysis utilities can prove very useful to determine system capacity. For example, the input file illustrates use of while loop in lines 233 to 236. The loop allows prediction of system performance for number of users from 1 to 100.

3.4 Analysis Method

PerfCenter uses both analytical and simulation methods for performance prediction. The analytical technique has been explained in detail in [25]. Its current implementation only supports open arrivals into the system. The simulation technique supports all the features presented in this paper.

To use the simulation technique in PerfCenter, we need to specify some simulation parameters. PerfCenter uses the *independent replications* [8] method to estimate the performance measures. Averages from a single run are calculated after discarding the values corresponding to the initial "transient" part of the simulation. The following lines show how the required model parameters are specified.

```
modelparams
method simulation
type <open|closed>
replicationno <#replications>
noofrequests <#requests>
confint true
```

```
sampleno <#samples>
startupsampleno <#samples>
<measure-1> <confidence interval level>
..
<measure-n> <confidence interval level>
end
```

The number of replications and the total number of requests simulated per replication are specified by `replicationno` and `noofrequests` respectively. The `sampleno` corresponds to the actual number of samples of measures that will be taken in each run. This number is used for measures such as utilization or throughput, which are continuous-time measures. The `startupsampleno` corresponds to the number of samples that should be discarded, to make sure that the initialization bias is removed. A list of the output measures for which confidence intervals are to be calculated along with the confidence level is then specified.

We now illustrate the use of PerfCenter with a case study of a Webmail application.

4. DEPLOYMENT AND CONFIGURATION DESIGN USING PERFCENTER

For the Webmail application of Section 3, the data center architect's decision involves choosing appropriate host configurations, appropriate deployment of application servers on the hosts, sizing the network link, and finally, configuring the four servers.

We present two scenarios in the following. The first is that of sizing for a small user group and the second is that of scaling to an extremely large user group. For both these scenarios, we show PerfCenter's usefulness in arriving at a satisfactory deployment and configuration design.

4.1 Small User Group Scenario

Consider the example of a company that needs to set up an internal Webmail system, for its 500 employees. We assume that at a given time not more than 100 employees access Webmail. Each user sends a request to the application after an average of 3 seconds after getting a response. An average scenario response time requirement of less than one second has been specified for this application.

An initial system deployment (DPLY1) to support this system is shown in Figure 6. PerfCenter can be run on the input file (Figure 7) corresponding to this deployment, to estimate the performance measures for this system. Note that this is a "closed arrivals" model. Table 1 shows predicted utilization of host CPUs and disks for this deployment and Figure 10 shows the response time as a function of number of users. As can be seen, the response time exceeds 1 second at just 30 users, which is unacceptable. Also, it can be noted that host H1 is over-utilized (98%). This suggests that the initial deployment is not suitable, and other deployments need to be tried. Initially, we will assume that software resources such as number of threads and buffer size are not a bottleneck (set to a very high number in the input file). Similarly, we assume the WAN link capacity to be 100Mbps, which is high enough to not be a bottleneck.

4.2 Identifying server deployment

Since a machine was getting over-utilized in DPLY1, in the next deployment we add two new machines to LAN1, hosts

```
1 :  variable
2 :  nusr 1
3 :  diskspeedupfactor3 1
4 :  cpuspeedupfactor3 1
5 :  nwp  100
6 :  webcnt 4000
7 :  authcnt 4000
8 :  imapcnt 4000
9 :  smptcnt 4000
10 : end
11 :
12 : device
13 : cpu
14 : disk
15 : end
16 :
17 : host H1
18 : cpu count 1
19 : cpu buffer 99999
20 : cpu schedP fcfs
21 : cpu speedup 1
22 : disk count 1
23 : disk buffer 99999
24 : disk schedP fcfs
25 : disk speedup 1
26 : end
27 :
28 : host H2
29 : cpu count 1
30 : cpu buffer 99999
31 : cpu schedP fcfs
32 : cpu speedup 1
33 : disk count 1
34 : disk buffer 99999
35 : disk schedP fcfs
36 : disk speedup 1
37 : end
38 :
39 : host H3
40 : cpu count 1
41 : cpu buffer 99999
42 : cpu schedP fcfs
43 : cpu speedup cpuspeedupfactor3
44 : disk count 1
45 : disk buffer 99999
46 : disk schedP fcfs
47 : disk speedup diskspeedupfactor3
48 : end
49 :
50 : host H4
51 : cpu count 1
52 : cpu buffer 99999
53 : cpu schedP fcfs
54 : cpu speedup 1
55 : disk count 1
56 : disk buffer 99999
57 : disk schedP fcfs
58 : disk speedup 1
59 : end
60 :
61 : server web
62 : thread count webcnt
63 : thread buffer 99999
64 : thread schedP fcfs
65 : task send_to_auth
66 : task send_to_imap
67 : task change_to_html
68 : task send_to_smtp
69 : end
70 :
71 : server auth
72 : thread count authcnt
73 : thread buffer 99999
74 : thread schedP fcfs
75 : task verify_passwd
76 : task verify_session
77 : end
78 :

79 :  server imap
80 :  thread count imapcnt
81 :  thread buffer 99999
82 :  thread schedP fcfs
83 :  task list_message
84 :  task read_message
85 :  task delete_message
86 :  end
87 :
88 :  server smtp
89 :  thread count smtpcnt
90 :  thread buffer 99999
91 :  thread schedP fcfs
92 :  task send_message
93 :  end
94 :
95 :  task send_to_auth
96 :  cpu servt 0.030
97 :  end
98 :
99 :  task send_message
100 : cpu servt 0.014
101 : end
102 :
103 : task send_to_imap
104 : cpu servt 0.020
105 : end
106 :
107 : task send_to_smtp
108 : cpu servt 0.020
109 : end
110 :
111 : task change_to_html
112 : cpu servt 0.010
113 : end
114 :
115 : task verify_passwd
116 : cpu servt 0.010
117 : disk servt 0.020
118 : end
119 :
120 : task verify_session
121 : cpu servt 0.005
122 : end
123 :
124 : task list_message
125 : cpu servt 0.025
126 : disk servt 0.050
127 : end
128 :
129 : task read_message
130 : cpu servt 0.020
131 : disk servt 0.035
132 : end
133 :
134 : task delete_message
135 : cpu servt 0.015
136 : disk servt 0.020
137 : end
138 :
139 : lan
140 : lan1
141 : lan2
142 : end
143 :
144 : link lk1 lan1 lan2
145 : trans nwp Mbps
146 : mtu 256 bytes
147 : prop 1 ms
148 : headersize 40 bytes
149 : end
150 :
151 : deploy H1 lan1
152 : deploy H2 lan2
153 : deploy H3 lan1
154 : deploy H4 lan1
155 :
156 : deploy web H1
157 : deploy imap H1

158 : deploy auth H2
159 : deploy smtp H1
160 :
161 : scenario Login prob 0.35
162 :     send_to_auth  verify_passwd 200 SYNC
163 :     branch prob 0.1
164 :        verify_passwd  change_to_html 100
165 :     end
166 :     branch prob 0.9
167 :        verify_passwd  send_to_imap 200
168 :        send_to_imap list_message 200 SYNC
169 :        list_message  change_to_html 2000
170 :     end
171 : end
172 :
173 :
174 : scenario Send prob 0.2
175 :     send_to_auth  verify_session 200 SYNC
176 :     branch prob 0.2
177 :        verify_session  change_to_html 100
178 :     end
179 :     branch prob 0.8
180 :        verify_session send_to_smtp 200
181 :        send_to_smtp send_message 2000 S
182 :        send_message  send_to_imap 200
183 :        send_to_imap list_message 300 SYN
184 :        list_message change_to_html 2000
185 :     end
186 : end
187 :
188 : scenario Read prob 0.25
189 :     send_to_auth  verify_session 200 SYNC
190 :     branch prob 0.2
191 :        verify_session  change_to_html 100
192 :     end
193 :     branch prob 0.8
194 :        verify_session  send_to_imap 200
195 :        send_to_imap read_message 200 SYNC
196 :        read_message  change_to_html 2000
197 :     end
198 : end
199 :
200 : scenario Delete prob 0.2
201 :     send_to_auth  verify_session 200 SYNC
202 :     branch prob 0.2
203 :        verify_session  change_to_html 100
204 :     end
205 :     branch prob 0.8
206 :        verify_session  send_to_imap 200
207 :        send_to_imap delete_message 200 SYNC
208 :        delete_message  change_to_html 200
209 :     end
210 : end
211 :
212 : loadparams
213 :    noofusers    nusr
214 :    thinktime    exp(3)
215 : end
216 :
217 : modelparams
218 :    method simulation
219 :    type closed
220 :    noofrequests 10000
221 :    confint false
222 :    replicationno 1
223 : end
224 :
225 : nusr=5
226 :
227 : print "Resource Utilization"
228 :
229 : print "Tput "+tput()
230 : nusr=1
231 :
232 : print "Users,ResponseTime"
233 : while(nusr <= 100 )
234 :     print nusr+", "+ respt()
235 :     nusr = nusr+1
236 : end
```

Figure 7: Input file for PerfCenter

H3 and H4. Assuming that the Web server will require the most CPU capacity, we host the Web server on two machines H1 and H4. IMAP and SMTP are moved from host H1 to host H3. Figure 8 shows the new deployment. The input file is appended with the following lines to reflect the new deployment.

```
undeploy imap H1
undeploy smtp H1
deploy imap H3
deploy smtp H3
deploy web H4
```
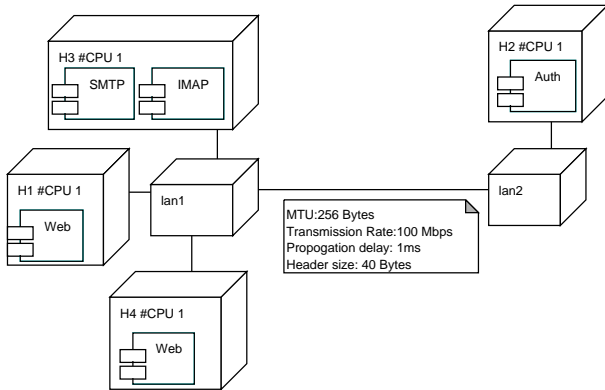

Figure 8: DPLY2: Relieving host H1 bottleneck

Table 1 shows the performance predicted for this deployment (DPLY2). We observe that the disk of H3 (on which IMAP is hosted) is over-utilized (80.6%). We therefore, replace H3 by a faster machine that would double the CPU speed and the disk access rate. Further, for the purpose of consolidation, we host the Web server on one machine, H1, after adding one CPU to it. Figure 9 shows the new deployment (DPLY3). We analyze this deployment by appending the following lines to the input file:

```
undeploy web H4
set H1:cpu:count 2
diskspeedupfactor3 = 2
cpuspeedupfactor3 = 2
```
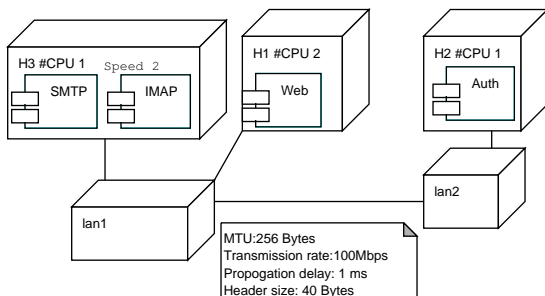

Figure 9: DPLY3: Disk upgraded, Web server consolidated

PerfCenter predicts Web server to be more utilized with 77.6%. If we wish to leave some headroom, one more processor could be added to H1 resulting in deployment con-

figuration DPLY4 (Figure 13). The summary of utilizations for all four configurations are shown in table 1.

	H1 CPU %	H2 CPU %	H3 CPU %	H4 CPU %	IMAP Host Disk%	H2 Disk %
DPLY1	98.1	8.2	NA	NA	41.2	8.8
DPLY2	67.5	15.9	48.6	75.0	80.6	17.0
DPLY3	77.6	17.7	23.9	NA	44.2	18.7
DPLY4	53.8	18.4	27.7	NA	47.1	19.5

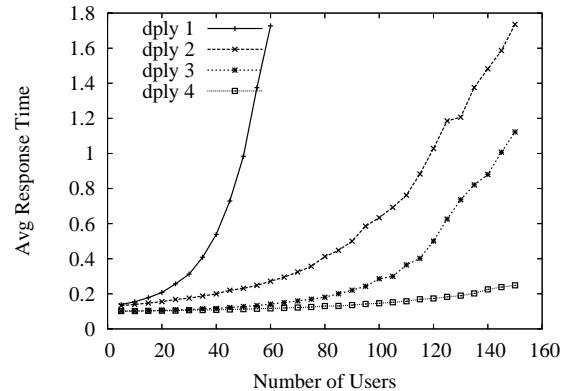**Table 1: Host Device Utilizations**

Figure 10: Average Scenario Response Time

Figure 10 shows a plot of the estimated average scenario response times for all four deployments. As we are designing for 100 users, response time is acceptable for the deployment alternatives DPLY2-DPLY4. However, DPLY4 best balances the desire for adequate utilization of resources (while still leaving some headroom), with the user requirement of scenario response times, and hence is chosen as the final deployment for this application.

4.3 Identifying network link capacity

Now that the deployment is finalized, we move on to to estimating the network link capacity required for Webmail application (which was assumed so far to be 100 Mbps). Table 2 shows link utilizations predicted using PerfCenter for transmission rates of 256Kbps and 1Mbps. In both cases, the propagation delay is assumed to remain at 1 ms. The table shows that a 256 kbps link should be sufficient for this application.

	256Kbps	1Mbps
lan1 to lan2	20.1%	5.1%
lan2 to lan1	18.7%	4.8%

Table 2: Network Utilization

4.4 Selecting Server Thread Count

The number of threads to be set for a server is important for performance delivered by the server. If this number is set too low, hardware resources remain underutilized. If this number is high it increases memory utilization and the influx of connections will bring the server to a standstill.

Our goal is to determine the minimum number of server threads necessary to utilize the hardware resource, which still delivers acceptable response time, for a given load. In this case we do this sizing for 100 users (with think time of 3 seconds).

We focus on thread sizing for the Web server. This is because it is the front-end server - the thread sizes for the "downstream" servers can be determined easily once the Web server threads are set. The values of average scenario response time, Web server utilization, Web server host CPU utilization for number of threads varying from 1 to 15 are plotted in Figure 11 and Figure 12 . In the figure, as the thread count increases, the utilization of the Web server threads decrease and that of the CPU increases.
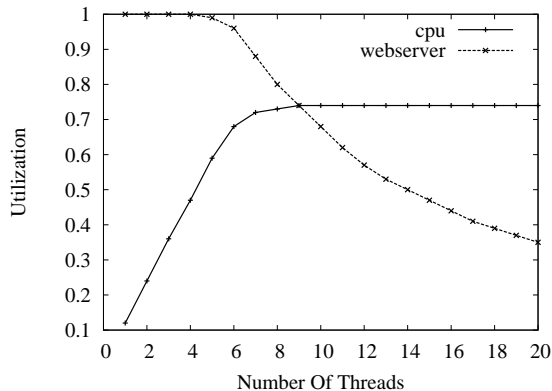

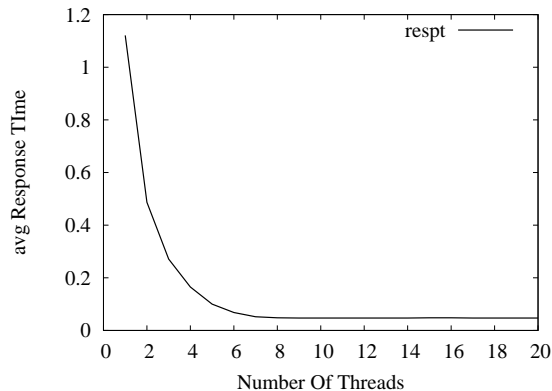Figure 11: Utilization vs Web server thread count

Figure 12: ResponseTime vs Webserver thread count

From Figure 12 we note that after thread count reaches seven, response time becomes constant. Increasing thread count further does not have any effect on the average response time. Also, from Figure 11 when thread count is seven the CPU utilization flattens out. Hence we can conclude that for this configuration, seven could be the ideal value of thread count.

The final deployment and configuration for the Web mail application is shown in Figure 13. PerfCenter can be used to determine the maximum capacity of this configuration. As shown in Figure 14, the maximum throughput achieved by the system is around 30 requests/sec. Figure 15 shows the

response time performance for this system. Note that the 95% confidence intervals for these two measures have been shown in the corresponding plots.
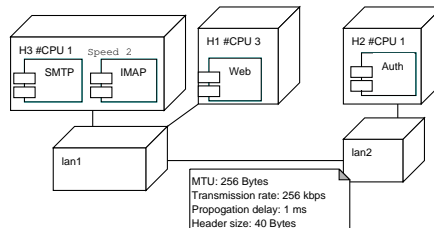

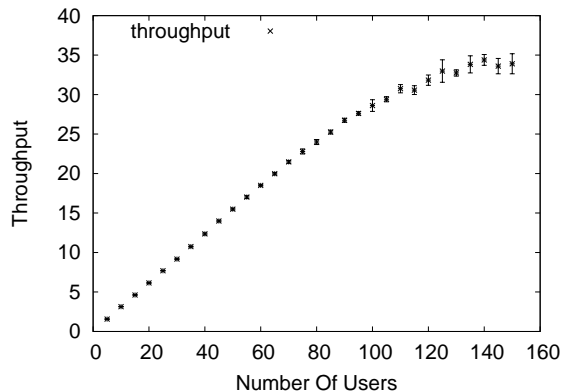Figure 13: DPLY4: Recommended configuration for small user group

Figure 14: Maximum Throughput with 95% confidence interval

4.5 Web Mail Scaling

Suppose that we now want to upgrade the Webmail system to support requests arriving at rate of 2000 requests/sec (specified as open arrivals). The underlying systems should be upgraded to handle this heavy load.

To perform hardware upgrade, the host with the highest utilization is identified and upgraded. If this new configuration does not support the required load of 2000 requests/sec, then we again upgrade the host having highest utilization. These steps are repeated till a configuration is found that is predicted to support an arrival rate of 2000 requests/sec. For hardware upgrade we again assume that there are no software bottlenecks.

Step 1: In the earlier configuration as shown in Figure 13, maximum throughput achieved by the system was 30 requests/sec. The Web Server host H1 was the bottleneck resource.

The scaled system should support almost 70 times the load of the earlier system. Thus significantly higher CPU and disk capacity is required for this system. We assume that two new disk assemblies are available to us which speed up the disk access by a factor of 80 in host H3 and by a factor of 20 in host H2. Similarly, we upgrade host H1 to a 32 processor machine. We assume that we have another machine, H4 of this type. We also upgrade H2 and H3 to have 12 processors each. The CPU speeds are also no
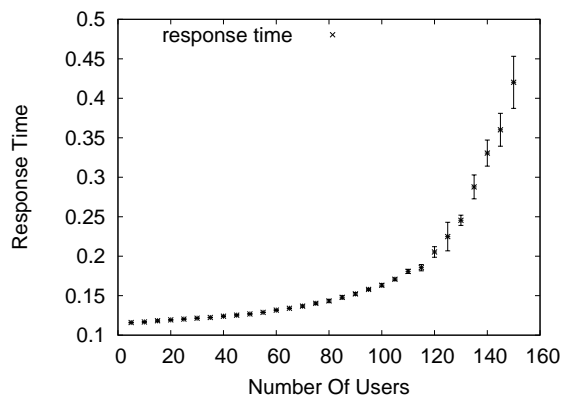
Figure 15: Response time behavior with increasing load, with 95% confidence interval

Step		H1	H2	H3	H4	H5
Step1	CPU count	32	12	12	32	
	Utilization %	88.0	100	75.8	87.7	
	CPUSpeedup	2	1	4	2	
	Disk Speedup		20	80		
	Utilization%		51.6	46.5		
Step2	CPU count	32	32	18	32	32
	CPUSpeedup	2	1	4	2	2
	Utilization %	64.5	55.9	57.0	63.1	58.7
	Disk Speedup		20	80		
	Utilization%		58	52.6		

Table 3: Resource Utilizations for Scaled-up Deployment

longer the same, the speed up factors are specified in the input by modifying the "speedup" attribute of the CPUs. The speedup factor can be based on raw CPU speed, or the speedup achieved by some benchmark applications.

```
diskspeedupfactor2 =20
diskspeedupfactor3=80
deploy web H4
set H1:cpu:count 32
set H4:cpu:count 32
set H3:cpu:count 12
set H2:cpu:count 12
cpuspeedupfactor1=2
cpuspeedupfactor3=4
cpuspeedupfactor4=2
```

PerfCenter predicts that this configuration fails at load of 2000 requests/sec, with H2 becoming over-utilized at 100%. Throughput achieved for this configuration is 1755 requests/sec. Resource utilizations predicted by PerfCenter for this configuration are as shown in Table 3.

Step 2: Since host H2 CPUs are the bottleneck, we upgrade H2. We also upgrade H3 and add a 32-processor machine H5, and deploy the Web server on it.

```
host H5
cpu count 32
cpu buffer 99999
cpu schedP fcfs
```

Web-H1	Web-H4	Web-H5	IMAP	SMTP	Auth
45	45	45	135	135	135

Table 4: Server Thread Sizing

```
cpu speedup 2
end
deploy web H5
set H2:cpu:count 32
set H3:cpu:count 18
```

For this deployment, PerfCenter estimates the utilizations as shown in Table 3. In this configuration, utilization of all hosts is below 70%. This configuration supports the specified arrival rate of 2000 requests/sec while keeping processor and disk utilizations uniform across all hosts.

Following the approach discussed in Section 4.4 we can estimate the new thread count value for all the servers as shown in Table 4.

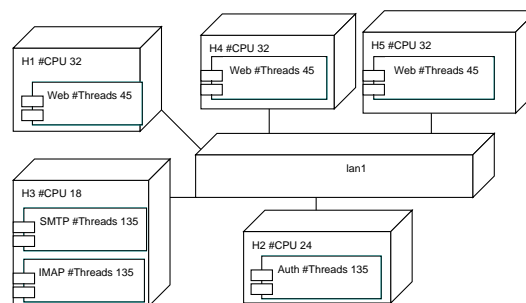Final recommended deployment and configuration arrived at by using PerfCenter is shown in Figure 16.

Figure 16: Recommended configuration for scaled up system

5. CONCLUSIONS AND FUTURE WORK

We presented a tool PerfCenter, that is convenient for representing the hardware and software architecture of an application which is to be deployed in a data center. We explained the PerfCenter's abstraction and its specification language in detail, and presented a deployment case-study using a Web-mail example. The example further drove in the usefulness of PerfCenter in evaluating various deployment scenarios. PerfCenter also provides a non-trivial network abstraction, which again, is relevant if the application is to be deployed over a WAN.

Some validation of the results provided by PerfCenter, by comparison with measured values, has been reported in [25]. However, further thorough validation is required to increase the confidence in the tool.

There are a multitude of features that can be added to PerfCenter. Among the ones that are in progress already are the incorporation of soft resources other than threads. E.g., PerfCenter will soon allow specification of a resource such as a piece of "synchronized" code (in other words a "critical section"). PerfCenter currently also does not support the expression of passive resources such as memory. We will allow such constructs in the near future.

6. REFERENCES

[1] S. W. Ambler. UML 2 Activity Diagrams. http://www.agilemodeling.com/artifacts/activityDiagram.htm, August 2006.

[2] S. W. Ambler. UML 2 Sequence Diagrams. http://www.agilemodeling.com/artifacts/sequenceDiagram.htm, August 2006.

[3] F. Andolfi, F. Aquilani, S. Balsamo, and P. Inverardi. Deriving performance models of software architectures from message sequence charts. In *WOSP '00: Proceedings of the 2nd International Workshop on Software and Performance.*, pages 47–57, 2000.

[4] S. Balsamo, F. Aquilani, and P. Inverardi. An approach to performance evaluation of software architectures. In *Proceedings of the first International Workshop on Software and Performance*, pages 178–190, 1998.

[5] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

[6] S. Balsamo and M. Marzolla. Performance evaluation of UML software architectures with multiclass queueing network models. In *Proceedings of the 5th International Workshop on Software and Performance*, pages 37–42, 2005.

[7] S. Balsamo, V. D. N. Personè, and P. Inverardi. A review on queueing network models with finite capacity queues for software architectures performance prediction. *Performance Evaluation*, 51(2-4):269–288, 2003.

[8] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Pearson Prentice-Hall, 2005.

[9] V. Cortellessa, M. Gentile, and M. Pizzuti. Xprit: An XML-based tool to translate UML diagrams into execution graphs and queueing networks. In *QEST '04: Proceedings of the 1st International Conference on Quantitative Evaluation of Systems*, pages 342–343. IEEE Computer Society, 2004.

[10] P. Cremonesi and G. Casale. How to Select Significant Workloads in Performance Models. In *Int. CMG Conference*, pages 183–192, 2007.

[11] A. Deshpande and V. Apte. PerfCenter User Manual. www.cse.iitb.ac.in/perfnet/perfcenter/manual.pdf.

[12] D. A. M. e; and H. Gomaa. A method for design and performance modeling of client/server systems. *IEEE Transactions on Software Engineering*, 26(11):1066–1085, 2000.

[13] R. G. Franks. *Performance analysis of distributed server systems*. PhD thesis, Carleton University, 2000. Adviser-C. Murray Woodside.

[14] V. Grassi, R. Mirandola, and A. Sabetta. From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In *Proceedings of the 5th International Workshop on Software and Performance*, pages 25–36, 2005.

[15] G. P. Gu and D. Petriu. XSLT transformation from UML models to LQN performance models. In *WOSP '02:Proceedings of the 3rd International Workshop on Software and Performance*, 2002.

[16] G. P. Gu and D. C. Petriu. From UML to LQN by XML algebra-based model transformations. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 99–110, 2005.

[17] H. H. Liu. Service Demand Models for Enterprise Software Applications. In *Int. CMG Conference*, pages 249–260, 2005.

[18] I. Object Management Group. UML Profile for Schedulability, Performance, and Time Specification. http://www.omg.org/cgi-bin/doc?formal/2005-01-02, January 2005.

[19] D. C. Petriu and C. M. Woodside. Software performance models from system scenarios in use case maps. In *TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, pages 141–158. Springer-Verlag, 2002.

[20] P. Reeser and R. Hariharan. Analytic model of web servers in distributed environments. In *WOSP '00: Proceedings of the 2nd International Workshop on Software and Performance*, pages 158–167, 2000.

[21] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Transactions on Software Engineering*, 21(8):689–700, 1995.

[22] S. S. Shirodkar and V. Apte. AutoPerf: an automated load generator and performance measurement tool for multi-tier software systems. In *Proceedings of the 16th International Conference on the World Wide Web*, pages 1291–1292, 2007.

[23] C. U. Smith, C. M. Lladó, V. Cortellessa, A. D. Marco, and L. G. Williams. From UML models to software performance results: an SPE process based on XML interchange formats. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 87–98, 2005.

[24] C. U. Smith and L. G. Williams. Performance engineering evaluation of corba-based distributed systems with spe*ed. In *TOOLS '98: Proceedings of the 10th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, pages 321–335. Springer-Verlag, 1998.

[25] R. P. Verlekar, V. Apte, P. Goyal, and B. Agarwal. PerfCenter: A methodology and tool for performance analysis of application hosting centers. In *The 15th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, October 2007.

[26] C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transactions Computers*, 44(1):20–34, 1995.

[27] J. Xu, A. Oufimtsev, M. Woodside, and L. Murphy. Performance modeling and prediction of enterprise javabeans with layered queuing network templates. In *Proceedings of the Conference on Specification and verification of component-based systems*, 2005.

[28] J. Xu, M. Woodside, and D. Petriu. Performance analysis of a Software Design using the UML Profile for Schedulability, Performance and Time. In *Proc. 13th Int Conf. on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS 2003)*, pages 291 – 310, 2003.