PerfCenterLite: Extrapolating Load Test Results for Performance Prediction of Multi-Tier Applications

Varsha Apte* Nadeesh T. V.
Department of Computer Science and Engineering
Indian Institute of Technology - Bombay,Mumbai 400 076, India
varsha,nadeeshtv@cse.iitb.ac.in

ABSTRACT
Performance modeling is an important step in the lifecycle of a typical Web-based multi-tier application. However, while most practitioners are comfortable carrying out load tests on a Web application on a testbed, they find sophisticated performance modeling tools difficult to use because many inputs required by them are difficult to obtain. Chief among these is the service times of various types of requests at various resources in the multi-tier system (e.g. CPU execution time required at the Web server by a “Login” request). In this paper, we present PerfCenterLite, a tool focused on ease of use for practitioners of performance analysis. The tool (a) provides a spreadsheet template for describing the application architecture and (b) accepts standard performance metrics obtained from load testing of the application. PerfCenterLite then uses mathematical estimation techniques and transforms this input into a full-fledged performance model as required by a sophisticated performance modeling tool. Validation experiments show that performance metrics predicted using PerfCenterLite match well with measured values.

Categories and Subject Descriptors
H.4 [Performance of systems]: Modeling Techniques

General Terms
Simulation Tool

Keywords
Performance modeling, load test results

1. INTRODUCTION
Performance analysis remains an important step in the release cycle of an Internet application. A multi-tier Internet application is called as such, because it typically comprises of “tiers” such as Web Tier, Application Tier and Database Tier. A load-testing effort which consists of deploying the application in a testbed environment and generating synthetic load on it using load-generation tools is the most common method of characterizing the performance of an application, and identifying system bottlenecks. Load testing provides application performance metrics such as response time and throughput, and server metrics such as CPU utilization, at a particular load level.

Load testing results are, however, specific to the testbed and workload for which they have been produced. But it is often the case that the environment on which an application is to be deployed (the target) is different from the testbed environment. For example, the Web server host machine in the target environment could be an 8-CPU machine while the testbed host had only two CPUs.

Thus performance modeling is an important step in the process of performance analysis of the application. Performance models are typically queuing models of the various resources that make up the application software and hardware [6]. However, queuing models need to be parameterized - i.e. the values of the various parameters required by queuing models need to be determined before the model can be evaluated. A specific input required for building a queuing model of a resource - say the CPU of a machine, is the service time of each of the various classes of requests that come to the CPU. In case of a multi-tier application, a request may visit various servers in the various “tiers” multiple times before it is completed. In such a case the service time at every visit at every tier is required as an input to the underlying queuing model.

There are many modeling tools that will accept as input, detailed application software and hardware descriptions with quantified parameters (such as service times), then evaluate the model and provide estimates of performance metrics for these parameters. However, practicing performance analysts find such tools overly detailed and complex to use. E.g., they find it tedious to provide the message flow for each request class using the typical formalisms of existing modeling tools. They further find it daunting (or may not have the required technical skill) to measure per-request service times at resources at each tier [10]. We believe that the widespread adoption of performance modeling techniques has been hindered by these problems.

In this paper, we present a performance modeling tool called PerfCenterLite which addresses these problems by abiding by a “less-is-more” design principle. It works with
minimal load-testing results, minimal hardware and software description and either mathematically estimates or makes judicious assumptions about missing details, and proceeds to create and evaluate a model of the application on the target environment.

PerfCenterLite's input is accepted in the form of a spreadsheet template that must be filled by the performance analyst. The tool then carries out several transformations to create input specification that is accepted by a richer tool called PerfCenter [3] that we use as the "modeling engine" of PerfCenterLite. Some of these transformations are straightforward syntactic mappings, some require judicious practical assumptions, and some require the use of mathematical estimation.

The novelty and contributions manifested in PerfCenterLite are as follows:

- While methodologies for service demand estimation have been studied to great depth [4, 7], to our knowledge, PerfCenterLite is the only tool for application performance modeling that applies one such method. This allows it to work exclusively with metrics obtained from load-testing such as request throughput and server resource utilization to quantify parameters of the underlying performance model.
- PerfCenterLite's input constructs have been carefully chosen to be simple and appealing to a practitioner - we use a spreadsheet template to describe the model, and allow the user to describe the "testbed" and the "target" hardware environment, so that the question of performance "extrapolation" from the testbed to the target is directly answered.
- We validated predictions made by our tool by comparing with measured results of the application performance, and found the match to be well within acceptable error limits.

The rest of the paper is organized is follows: in Section 2 we introduce PerfCenterLite input format and our algorithms for transforming this input and generating a model specification file for PerfCenter [3]. Section 3 discusses the validation experiments and Section 4 describes related work. We conclude the paper in Section 5.

2. PERFCENTERLITE

PerfCenterLite was designed with three main requirements in mind: (a) The user should not need to learn a new "specification language" or even a new GUI; (b) the user should not have to specify detailed message sequence charts (MSC) for each request and (c) the user should be able to provide standard load testing results as input; specifically, the user should not have to estimate individual request service demands.

Thus, PerfCenterLite accepts the model description in the form of a comma-separated-values (csv) file (Figure 1). The user can edit the file in any spreadsheet tool and simply enter the model description and parameter values in the relevant cells. Figure 1 shows the file which specifies the performance model of an application called WebCalendar, which is a standard two-tier Web application for calendar management.

This file is processed by PerfCenterLite (which is implemented in python) and an input file for PerfCenter [3], which is an advanced performance modeling tool, is generated. PerfCenter is then invoked on this input file and the predicted performance metrics are displayed back in "csv" form by PerfCenterLite.

Before using PerfCenterLite, one must first carry out load tests of the application for various scenario (i.e. request) probability mixes, and record the client-side metrics such as throughput and response time, and additionally record the corresponding server resource utilizations. The user must additionally know basic details regarding the software and hardware architecture of the application.

We first describe PerfCenterLite model specification, and then describe how this is mapped to a PerfCenter model description file.

2.1 PerfCenterLite Input Specification

PerfCenterLite input is divided into sections and subsections (prefixed by "#" and "##" respectively) as shown in Figure 1, which then consist of a list of descriptive labels (which are not to be edited) whose values are entered in neighboring cells. Many sections are self-explanatory. We elaborate here on a select few:

- Software Architecture. In this section, we first simply list the tiers (tier details), and then in the Scenario_details section, the message sequence for each request can be described simply in terms of these tiers. Note that a "scenario" in PerfCenterLite (and PerfCenter) is short for "use case scenario" and refers to a request type.

In case of WebCalendar, the structure of the message flow for all three requests is the same (Figure 2) and thus a "default flow" is described as: Web→DB→Web for all the requests.

- Performance Testing Data from Measurement Testbed. We expect data from multiple load tests, carried out with different scenario mixes to be entered here. Thus, the different scenario mixes are first declared and then the scenario probability values for each mix are specified. Then the performance metrics measured for each mix are specified.

The blocks scenario_metrics, disk_usage, network_usage and CPU_utilization are specified separately for each mix.

- Target Environment Hardware Architecture is similar to the testbed hardware description. Here, the user may want to directly specify a speed-up expected of the target host relative to the testbed host (e.g. based on some benchmarks), in which case the CPU frequency specification is ignored.

PerfCenterLite transforms this specification to the input accepted by PerfCenter, as described next.

2.2 PerfCenterLite transformations

Many of the transformations done by PerfCenterLite (e.g. hardware architecture) are straightforward syntactic transformations. However, some are not, and we describe only those here.

2.2.1 Software servers and Scenarios

A PerfCenterLite "tier" maps to a PerfCenter "server" [3] almost trivially. We only note here that although PerfCenter requires as input the number of threads in each server, PerfCenterLite does not. We instead assume that the server...
threads will never be a bottleneck. We implement this assumption by setting the number of threads in each server in PerfCenter to be equal to the maximum number of simultaneous users in the system, as specified in the workload parameters section of PerfCenterLite.

In PerfCenter terminology, each visit to a server by a request is called a “task”, and these tasks need to be specified per server in its input file. Scenarios in PerfCenter are also described in terms of these tasks. Furthermore, PerfCenter expects specification of service times on each device for each of these tasks.

PerfCenterLite creates this task list for each server by parsing the message sequence flow given in terms of the tier names (scenario_details) as follows: every visit to a tier results in the generation of a different server “task”. This task is given a unique name of the following format:

\[ \text{task}_n \_ < \text{scenario name} > \_ < \text{servername} > \]

The scenario message flow in PerfCenter syntax is then created using these auto-generated task names. The task service times are estimated using the load testing data, and testbed and target hardware specifications. We describe this estimation in the next section.

2.2.2 Service Demand Estimation

PerfCenterLite takes scenario throughputs and per-process CPU utilization values of each host to estimate the CPU service demand, by applying the Utilization Law \([6]\), on multiple load test results. A system of equations with the service demands as “unknowns” is generated and solved using known methods \([5]\). This gives us the service demands of the requests on the testbed hardware. The disk and network

---

**Figure 1:** PerfCenterLite Input File of WebCalendar Created Using Load Test Results

**Figure 2:** Message sequence charts of WebCalendar requests (scenarios)
usage (bytes written and read) per request is also estimated similarly.

Note that if a request makes multiple visits to a server, the service demands estimated in this manner correspond to the total of the per-visit service times. However we are required to estimate per-visit service times on the testbed, and then extrapolate them to the target machine. Our estimation proceeds as follows.

First, we simply divide the estimated service demand of a request from a particular scenario by the number of visits to a server to get per-visit service time.

Then, we use the following rules for extrapolating the testbed task service times to the task service times on the target hardware:

- If "CPU speed-up" specified is 1, the CPU service time is scaled linearly with the target CPU frequency setting specified, and is assumed to not change with number of cores. Thus if $\tau_{\text{testbed}}$ is the estimated CPU service time of a task on the testbed, $f_{\text{testbed}}$ is the CPU frequency of the host on which this task runs in the testbed, and $f_{\text{target}}$ is the CPU frequency of the target host on which this task will run, then the scaled task service time is simply: $\tau_{\text{target}} = \tau_{\text{testbed}} \cdot \frac{f_{\text{target}}}{f_{\text{testbed}}}$.
- If a cpu-speedup, $r$, not equal to 1 is specified, the frequencies are ignored, and this speed up is used directly to estimate the service time as: $\tau_{\text{target}} = \tau_{\text{testbed}} \cdot r$.
- Disk service time is scaled linearly according to the disk speeds provided and is assumed to not change with the number of disks.

### 3. VALIDATION

We validate the accuracy of performance metrics predicted by PerfCenterLite by comparing the predicted values with measured values on the target platform. We used two applications for validation: WebCalendar, and “DellDVD” which is an online e-commerce store benchmark.

Table 1 describes all the validation experiments. Experiments 1-3 are on WebCalendar, and Experiment 4 is on DellDVD.

<table>
<thead>
<tr>
<th>Experiment Description</th>
<th>Testbed Hardware</th>
<th>Target Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Target machine has more cores, at higher frequency (same processor architecture)</td>
<td>Web server: 2-core machine with 1.4GHz Intel Xeon E5 processors (“IntelXeon”). DB Server: 2-core machine with 2.26 GHz Intel I3 processors (“IntelI3”)</td>
<td>Web server: 8-core Intel <a href="mailto:Xeon@1.7GHz">Xeon@1.7GHz</a>. DB Server: same as testbed</td>
</tr>
<tr>
<td>2 Testbed has only one machine, Target has two</td>
<td>Web server &amp; DB server on the same machine: “IntelXeon”</td>
<td>Web server: 8-core machine with 1.4 GHz Intel Xeon. DB Server: “IntelI3”</td>
</tr>
<tr>
<td>3 Target has different processor architecture</td>
<td>Web server: IntelXeon, DB Server: IntelI3</td>
<td>Web server: AMD 8-core processors, 1.4 Ghz. DB Server: same as testbed</td>
</tr>
<tr>
<td>4 Application has software bottleneck</td>
<td>Web server: One Intel Xeon <a href="mailto:E5@1.2GHz">E5@1.2GHz</a>. DB server: IntelI3</td>
<td>Web server: Intel Xeon E5, 2 cores @1.6GHz. DB server: IntelI3</td>
</tr>
</tbody>
</table>

Table 1: Testbed and Target Hardware Descriptions of all Experiments

In Figure 6 (the DellDVD validation) we can see that for mix1. Similarly, results from the load tests corresponding to mix2 and mix3 were also specified (not shown in the figure). We specify a different scenario mix in the target load parameters (as shown in Figure 1). These mixes are the same for Experiments 1-3.

In Experiment 3, CPU micro-benchmarks suggested that although the CPU frequencies were the same, a speed up factor of two was observed in the execution time on the AMD machine as compared to the Intel machine. This speedup was used in the PerfCenterLite model specification.

In Experiment 4, the DellDVD user session consisted of ten different request types (scenarios). We carried out only three different load tests with different mixes.

Figures 3-6 show the comparison of measured vs predicted values of average request response time, average request throughput and Web server CPU utilization, for Experiments 1-4.

We can see that for Experiments 1-3 there is a good match between measured and predicted values for all the metrics, up to a fairly high CPU utilization level. The values diverge only after 80% CPU utilization.

In Figure 6 (the DellDVD validation) we can see that for up to a load level of 50, the measured and predicted values match well, even though we used only three load tests for estimating ten transactions’ service demands. However, after that, there is a drastic divergence. The measured Web CPU utilization flattens out at 0.4, while the model predicts a steady increase. Similarly, measured throughput flattened out, while the model predicted an increase. We eliminated other possibilities of hardware bottlenecks and came to the conclusion that this exposes a “software bottleneck” in the DellDVD implementation. Since PerfCenterLite is designed to assume no software bottlenecks, it is not able to predict the metrics accurately when load reaches close to the software bottleneck’s capacity. This is also the likely reason for the divergence of metrics in Experiments 1-3 after utilization level of 80%.

### 4. RELATED WORK

There are various tools available for performance modeling of multi-tier applications. Many are very rich in features and in terms of the behaviours of multi-tier applications that they can capture. All are based on queuing networks or similar stochastic models which can model contention for resources.

LQNSolver [11] is a generic solver for Layered Queuing Networks (LQNs) and can be used to model multi-tier appli-
Figure 3: Measured vs Modeled Performance Metrics Validation for increased number of cores and higher frequency (Experiment 1)

Figure 4: Measured vs Modeled Performance Metrics Validation for scale-out deployment (Experiment 2)

Figure 5: Measured vs Modeled Performance Metrics Validation for different processor architecture (Experiment 3)

Figure 6: Measured vs Modeled Performance Metrics Validation for Software Bottle Neck (Experiment 4)
cations as LQNs. LQNSolver is rich in its “modeling power” - one can model various scheduling mechanisms, forking and joining of parallel executions and a post-remote call execution phase.

PerfCenter [3] on which PerfCenterLite is based, is more directly geared towards server application modeling, and it is easier to specify application architecture in its input language. PerfCenterLite is a wrapper tool to convert a lightweight specification file of the application to a detailed model specification file. PerfCenterLite is a compelling offering.

All of the above tools require message sequence charts as input and they also require the all-important service demand of every task on every resource in the hardware infrastructure. The specific problem of obtaining resource demands for parameterizing such models of multi-tier applications has also been addressed. E.g., a measurement tool called AutoPerf [8] has been specifically built to obtain per-request service demands at all tiers.

Various methods have been proposed to solve the problem of deriving service demands from resource utilizations and request throughputs [2, 4, 7, 12]. They all essentially formulate the problem as that of fitting a multi-variate linear function relating the request throughputs to the resource utilizations. However, to our knowledge no tool exists that implements these approaches.

While the tools and methods described above are no doubt rich and sophisticated, our claim is that this richness itself becomes a drawback in the use of sound and rigorous model-based performance engineering in practice. PerfCenterLite therefore adopts a minimalist approach towards model creation. We have done away with detailed specification of use-case scenarios and resources demands used by tasks in such scenarios, and reduced it to specifying default tier interactions. We take as input aggregates of all resource utilizations, and throughputs - which are standard measurements resulting from a load test. Finally, we offer a spreadsheet template, which should be trivial for anybody to fill in. These ideas distinguish our approach sharply from existing approaches and tools.

5. SUMMARY AND CONCLUSIONS

In this paper, we presented PerfCenterLite, which is essentially a wrapper tool to convert a lightweight specification of the workload, hardware and software architecture of an application to a detailed model specification file of the tool PerfCenter. The tool directly addressed the concern of a typical performance analysis team namely: “We have the load testing results of this application - how do we extrapolate these to estimate application performance in a different environment?” Furthermore, input specification was designed carefully to be in the “familiarity zone” of practicing performance engineers.

We carried out validation experiments for various prediction scenarios of the WebCalendar application and found that predictions match measured values very well.

Future improvements in PerfCenterLite will attempt to increase its modeling power, e.g. to capture power-managed devices, virtualized deployments - while still keeping the data collection and specification requirement very simple. The tool can also aim to give better feedback to the user e.g., whether there is software bottleneck in the system.

Acknowledgements

The authors would like to thank Manoj Nambari and Subhasri Duttagupta of Tata Consultancy Services, Mumbai for their valuable critiques of PerfCenter.

6. REFERENCES


