Performance Mimicking Benchmarks for Multi-tier Applications

Subhasri Duttagupta
Performance Engineering Research Center
Tata Consultancy Services
Mumbai, India
subhasri.duttagupta@tcs.com

Mukund Kumar
Performance Engineering Research Center
Tata Consultancy Services
Mumbai, India
mukund.k@tcs.com

Varsha Apte
Dept of Computer Science and Engg
Indian Institute of Technology
Mumbai, India
varsha@iitb.ac.in

ABSTRACT

Predicting performance of multi-tier enterprise applications for a *target* platform is of significant importance to IT industries especially when target environment is unavailable for deployment. Performance modeling techniques depend on accurate estimation of resource demands for a specific application. This paper proposes a methodology for deriving Performance Mimicking benchmarks (PMBs) that can predict resource demand of *application server* of multi-tier on-line transaction processing applications on a target environment. PMBs do not require the actual application to be deployed on the target itself. These benchmarks invoke similar method calls as the application at different layers in the technology stack that contribute significantly to CPU utilization. Further, they mimic all send and receive interactions with external servers (e.g., database server) and web clients. Ability of PMBs for service demand prediction is validated with a number of sample multi-tier applications including SPECjEnterprise2010 on disparate hardware configurations. These service demands when used in a modified version of Mean Value Analysis algorithm, can predict throughput and response time with accuracy close to 90%.

CCS Concepts

•General and reference → Performance; •Software and its engineering → *Software performance;*

Keywords

Performance, Prediction, Benchmarks, Cross-platform

1. INTRODUCTION

Performance characterization of a multi-tier enterprise application which is to be deployed on a particular production platform is a critical step prior to its release. However, due to various practical constraints, the production platform is often not available for application deployment. Instead, applications are deployed and performance-tested on a separate test environment which is often different from the production environment. Thus, there is a need for predicting application performance on the production (*target*) platform given its performance on the test (*source*) platform. Prediction of metrics such as request throughput and server utilization can be done using existing performance modeling tools [3] provided the resource demands of the application (e.g. CPU execution time of a request) are known. In this paper, we describe a methodology for generating benchmarks which can be exploited for obtaining estimate of CPU service demand of multi-tier applications, specifically, that of the application tier.

There are a large number of factors that can affect the CPU service time of an enterprise application—workload characteristics, the specific technology stack used by the application, the amount of data exchanged with other servers and the platform architecture characteristics. Given the above, a constructive approach of building a parameterized model of the execution time of an application program is prohibitively complex. Instead, the paper presents an approach for developing a set of simpler "benchmarks" which are executed on the target platform. The information acquired from the benchmark execution along with application profiling information collected on the source platform are used to estimate application performance on the target. Unlike applications, these benchmarks may not require installation and population of the application database or depend on multiple distributed services running on different servers. Hence, their deployments are expected to be simpler.

Benchmarks built using the proposed approach mimic the performance of the actual application by having the right mix of the type of CPU operations as used by the application. These Performance Mimicking Benchmarks (PMBs) make similar lower-level method calls corresponding to different layers in the technology stack. In addition, they mimic the send and receive calls of the application as shown in Figure 1. While profiling a number of mult-tier applications, it is observed that much of the application server CPU resource demand come from interactions among the tiers and calls to various layers as compared to the methods doing the actual business processing. Hence, the benchmarks are designed to capture the method calls made across different tiers. As a result, we expect that the ratio of service demands of PMB and that of the application remains almost unchanged on the source as well as the target platform. Thus, PMB helps

ICPE'16 Companion, March 12-18, 2016, Delft, Netherlands

DOI: http://dx.doi.org/10.1145/2859889.2859898

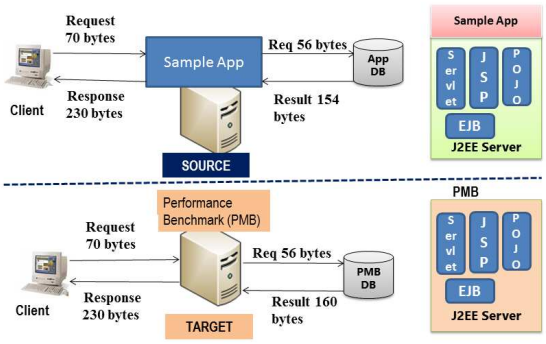

Figure 1: Illustration of Performance Mimicking Benchmark.

in estimating the demand of the application on the target platform.

Our paper is different from existing work in the sense that we aim for the benchmark to attain a representative estimate of service demand of the application rather than the same resource demand as the application. Since exact similarity is not required, there is potential for the benchmark to be re-used to represent multiple applications. We believe these PMBs can also help in reducing the time to market by providing a baseline performance estimate of the application on a given target platform. This paper makes the following specific contributions:

- We propose a method for creating benchmarks that require only application level profiling rather than kernel-level tracing or execution environment instrumentation.
- The benchmarks are designed to provide a representative value of service demand of the application on a deployment platform and can help in obtaining early performance estimate in the software life-cycle.
- We validate our approach by carrying out performance profiling of a number of applications across three architectural platforms. The maximum error in prediction of service demand is found to be under 13%.

2. RELATED WORK

Basic two most commonly used methods for performance prediction are simulation and analytical modeling [3],[8]. These methods have been tried out earlier for enterprise applications but usage of these techniques for doing cross-platform performance prediction is limited due to the effort involved and complexity in identifying the model parameters. As an initial attempt, we have tried using standard SPEC CPU ratings for the target and source machines along with the service demand on a source platform to predict performance of an application of interest. But this technique fails to provide good accuracy across different architectures as shown in our earlier work [4].

The idea of running a simple benchmark that clones the application in terms of a set of performance metrics is proposed by Joshi et al [6] in the domain of embedded systems and in the virtualized environment by Zheng et al [12]. Performance prediction for scientific applications based on similarity with a previously profiled set of benchmarks is proposed in [5], [10]. However, enterprise applications differ dramatically from scientific ones in their use of disk and network IO and CPU instruction mix. There are several approaches using Palladio Component Model for performance prediction of component-based applications. Krogman et al [9] extract behavior model of the application using reverse engineering and rely on bytecode benchmarking for predicting performance on target platforms. Authors in [2] have used the model successfully for multi-tier J2EE applications.

Tak et al propose an approach where a lightweight program ("PseudoApp") [11] is written that mimics essential application program characteristics, but is much easier to deploy on the target platform. The PseudoApp is built by capturing system calls of the original application and reproducing them on the target , and a basic integer looping technique is used to capture the CPU usage between system calls. For complex transactions involving a large number of string processing routines this approach may fail to provide good accuracy.

3. GENERATING PERFORMANCE MIMICKING BENCHMARK

In this section, we describe the methodology for deriving a performance mimicking benchmark (PMB) for a real-world enterprise application. Aim of PMB is to emulate application in terms of resource utilization at the application server. Since it is difficult to mimic application functionality without access to the source code, we mimic method calls at the technology layers (like application container or databases system) and to the operating system. The PMB is implemented as a web application and deployed using the same container as the actual application. Further, PMB uses the same software stack of the application server and the database server (e.g., JBoss and Postgres). We hypothesize that a PMB built for a sample application achieves the same ratio of service demands between source and target platforms. If the resource demands of the sample application and PMB are APP_{source} and PMB_{source} on the source platform, then by obtaining resource demand of PMB on the target environment as PMB_{target}, the resource demand of the application can be estimated as follows:

$$APP_{target} = \frac{APP_{source} \times PMB_{target}}{PMB_{source}}$$

In the following sections, the main steps involved in generating PMB and actions that the PMB performs are discussed.

3.1 Main Steps

In order to generate PMB, a sample application is first profiled through a standard profiler (e.g., YourKit, JVisualVM etc) to obtain the lower level method calls made by the application. Further, individual transactions of the application are analyzed through a packet analyzer. The information gathered using the profiler as well as the packet analyzer is incorporated in an input file and PMBs are generated using the input file. The overall process of building PMB is outlined in Figure 2. The other important aspect of PMB is a PMB skeleton. Based on the technology stack being used, the types of lower level calls would be different and hence, the CPU consumption due to method invocations could be different. As a preliminary step in the process of automatically building PMB, currently it is assumed that a

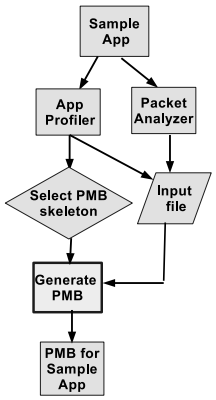

Figure 2: Process of Building PMBs

repository of basic skeletons of PMBs catering to different technologies are available. For example, under Java frameworks, there are PMB skeletons for Spring MVC, Apache Axis, Hibernate and Struts etc. Based on the profiler information, an application mapper module is used to select the appropriate skeleton. Thus, we expect different PMBs to be built for different technology stack. Using a PMB skeleton and application specific information incorporated in the input file, the actual PMB is built. In the following sections we further elaborate on different aspects of a PMB.

3.2 Mimicking Send/Receive Calls

PMB attempts to mimic all network calls made by the application. A web application can interact wit many external services - among these services, we consider interactions with database server for retrieving and storing relevant information and with web clients accessing the application. The mimicking benchmark sends and receives the same number of bytes to and from web clients. The size of requests and replies is determined during application profiling using a network packet analyzer tool that runs on the same server where application server is running. This information is gathered by accessing different transactions of the application through a web client and by tracking all the packets between the application server and the web client.

Next aspect of PMB is to mimic the send and receive calls between the actual application and its database server. The database response time and result set size varies with the type of query. The application is profiled with various combinations of the parameters and the average result set size is gathered. The PMB is written using such a query request that receives an average result set size. To avoid access to application database, a dummy database is created on the same machine as the application's database server in the source environment. The PMB makes requests to and receives responses from this dummy database server.

The dummy DB consists of a number of tables with multiple fields of different sizes. Based on the size of the records retrieved in the actual application, PMB makes SQL queries dealing with that many fields as to match the actual record size. Further, PMB requires to know the number of distinct beans used in the application. This is used for matching the behavior of PMB with the actual application. However, CPU service demand does not change due to higher or lower query response time. Hence, currently this factor is ignored.

3.3 Making Similar Method Calls

It is assumed that the PMB uses the same technology stacks as the application. Further, it uses similar higher level components as the application and deployed in the same container as the actual application. For example, if the application is a J2EE application and its components are Enterprise JavaBeans (EJB), Plain Old Java Objects (POJO) and web components (i.e., Servlets, JavaServerPages (JSP), then PMBs are written using similar components. A standard profiler is used to obtain a list of important methods to be mimicked. A method is marked as important to mimic based on two factors: (1) Invocation count - this is the total number of times (v_i) that a specific method is invoked by all its callers during the execution of a transaction or a higher level method call. (2) "Own" execution time - this is the amount of time (s_i) a method takes excluding all the subcalls it makes. Then, the total CPU demand of the method i is $v_i \times s_i$.

Assuming that at the application server the methods are mostly CPU intensive, the total CPU demand is proportional to the sum of demands for all such methods. Our methodology of building PMB makes use of this specific observation.

$$PMB_{source} = \sum_i s_i \times v_i$$

3.4 Mimicking Data Flow Sequences

This aspect of benchmark helps in mimicking the behavior of individual transactions of the application. For every transaction we capture the information on how the request and response data are flowing through different tiers. For an OLTP application, three tiers are identified as: client tier, application/web tier and database tier. The requests from client passes through these tiers in a certain sequence before the response is received by the client. Even though a simple flow could be web tier followed by database tier and database tier to web tier and back to client tier, in some requests database tier may not be used (e.g., forwarding requests from one jsp page to another) or in some requests, it may pass through web tier and database tier a number of times before the actual response is sent back to the client (e.g, first retrieving a state list followed by branch list within a state). Hence, two such sample data flows are (1) client to web, web to db, db to web and web to client and (2) client to web, web to db, db to web, web to db, db to web, then web to client etc. This data flow information is also gathered using a standard application profiler by tracing through method calls of a transaction and is available as part of the input file.

3.5 Generating PMBs

The previous sections describe how three different types of information are gathered regarding the application. All these information regarding the application are specified in an input file and, PMBs are generated using the input file. A sample input file is specified here. For each transaction, we require a separate input file. A transaction can be defined as set of logical operations that starts from user requests to receiving the entire response from the server.

The first part of the input file specifies the transaction

name and the number of http web requests the transaction makes. The next part specifies the data flow within the transaction and for each pair of tiers such as between the client and web tier the amount of bytes transferred.

```
name_of_transactions = view catalog;
no_of_requests = 1;
desc_of_dataflow = cl_web_db_web_cl;
no_of_bytes_cl_web = 70;
no_of_bytes_web_db = 56;
no_of_bytes_db_web = 154;
no_of_bytes_web_cl = 230;
type_of_sqls = Select;

 com.ibatis.dao.impl.invoke     1
 com.ibatis.ResultMap.getResults  2
 com.mysql.jdbc.InputStream.read  8
 org.apache.jsp.catalog_jsp   1
 org.apache.struts.doStartTag   2
 org.apache.struts.WriteTag   2
```

Example 1: Sample Input File

Such network byte information is gathered in Section 3.2. Based on the data flow, there may be many such statements regarding the number of bytes transferred. The next statement says the type of SQL statement whether it is insert, delete, update or select query. The last section provides a detailed method list and their invocations. PMB incorporates equivalent method calls corresponding to them and currently this step is done manually.

PMBs built using the above mentioned steps ensure that during execution of a PMB, CPU performs a mix of important operations that is similar to the sample application's mix on a source platform. Further, PMBs can be used to derive the service demand of the application on the target platform as well.

4. EXPERIMENTAL RESULTS

In this section, we evaluate the effectiveness of performance mimicking benchmarks in predicting service demands for different target platforms. We have done the evaluation on four web applications written in Java: DellDVD— an online DVD store, JPet— an online Petstore which is a standard J2EE benchmark, NGCel— an in-house reporting application on next-generation cellular phone usage and SPECjEnterprise2010 – an industry standard SPEC benchmark. These applications allow PMB to be tested for different scenarios such as rendering display with or without jsp pages, using strut framework, using J2EE container etc. The technology stack of the PMBs currently consists of either Apache Tomcat container or glassfish J2EE application server with MySQL as the database server. For this paper, we restrict our discussion to only CPU resource demand of application tier and we have not considered local disk read-/writes at the web/application tier. Further, finding the effect of configuration changes in database tier is not considered in this paper.

We evaluate these applications on three different server configurations as given in Table 1 above.

4.1 NGCel Results

In this section, we show the results for NGCel application. We observe that application service demands vary with concurrency in all the applications. Hence, NGCel transactions

Table 1: Server Categories for Sample Applications

Server Type	Features
Intel	8 Core CPU 2.48 GHz Xeon with 1MB L2 cache, 8 GB Physical RAM
AMD	Quad Core AMD Opteron CPU 2.19 GHz with 2MB L2 cache, 4 GB RAM
Solaris	4 Core UltraSparc CPU 1.5 GHz with 32MB cache, 16 GB RAM

are run with different number of users and CPU utilization and throughput values are obtained. Then utilization law is used to estimate the service demands. For this application, the source platform is taken as Intel server. Based on the ratio of service demands of NGCel to PMB on Intel platform, and service demand of PMB on a target platform, demand is predicted on AMD and Solaris. We also compute error% for three different transactions of NGCel in Table 2. We observe that error percentage is less than 10% in all situations. For DellDVD application also, the maximum error in service demand prediction by PMB is found to be 13%. We illustrate the error computation using service demand of Best Month transaction in Table 2. The ratio of service demands of NGCel and its PMB on Intel server for 400 users is obtained as = 1.39/1.18 = 1.177. This is used to predict demand on AMD server as =1.17 x demand of PMB = 1.17 x 1.91=2.25 ms. But the actual service demand on AMD server is 2.2 ms. Hence,

$$Error\% = \frac{(|2.2 - 2.25|) \times 100}{2.2} = 2.26$$

4.2 Comparison with AppLite Benchmarks

The goal of this section is to compare the accuracy of prediction by PMB with that of equivalent technique available in the literature. While PseudoApp [11] application claims to provide similar resource consumption as the application, it is done on Xen platform using kernel tracing tool. Our source and target platforms are primarily Linux and we intend to restrict ourselves to only application level profiling. Hence, we develop an *AppLite* benchmark that mimics the send and receive calls of the actual application by sending to and receiving same amount of bytes from a simple socket server. Further CPU utilization of the actual application is matched by AppLite by doing some integer arithmetic operations. The parameters of arithmetic operation are adjusted according to the transaction being profiled. This matching of service demand is done only on one platform e.g., Intel as a Source while the same AppLite is run on other platforms (AMD and Solaris as Targets) to predict service demand of the sample application.

Figure 4.2(a) compares the error% for PMB and AppLite in case of JPet application for 400 users. The error% in prediction of AppLite on AMD platform varies between 11% and 38%. However, by using PMB, the average error% in prediction is around 6%. Further, on Solaris platform, the average error% of AppLite is 40% while with PMB it is around 5%. The prediction error percentage variation across AMD and Solaris can be due to difference in architecture and various system parameters. However, the PMB is found to be more accurate because it is designed to capture major method calls that result in significant CPU consumption.

Table 2: Actual, predicted values of service demand (in ms) and error% in prediction for various transactions of NGCel at different concurrencies (shown in bracket) with Intel machine as source, and AMD and Solaris as target platforms

NGCel Transaction	**APP Intel**	PMB Intel	PMB AMD	PMB Solaris	APP AMD	Predicted AMD	Error% AMD	APP Solaris	Predicted Solaris	**Error%** Solaris
Most used (400)	1.98	1.89	3.14	4.29	3.6	3.3	8.5	4.92	4.51	9.54
Most used (500)	2.00	1.83	3.23	4.44	3.68	3.52	4.4	5.02	4.85	3.78
Best Month (400)	1.39	1.18	1.91	2.01	2.2	2.25	2.26	2.17	2.37	9.19
Best Month (500)	1.44	1.23	2.03	2.05	2.3	2.39	4.2	2.26	2.42	7.03
Home Page (400)	1.81	1.58	2.25	3.76	2.55	2.58	1.15	4.41	4.31	2.13
Home Page (500)	1.78	1.56	2.26	4.1	2.6	2.58	0.72	4.91	4.68	4.7

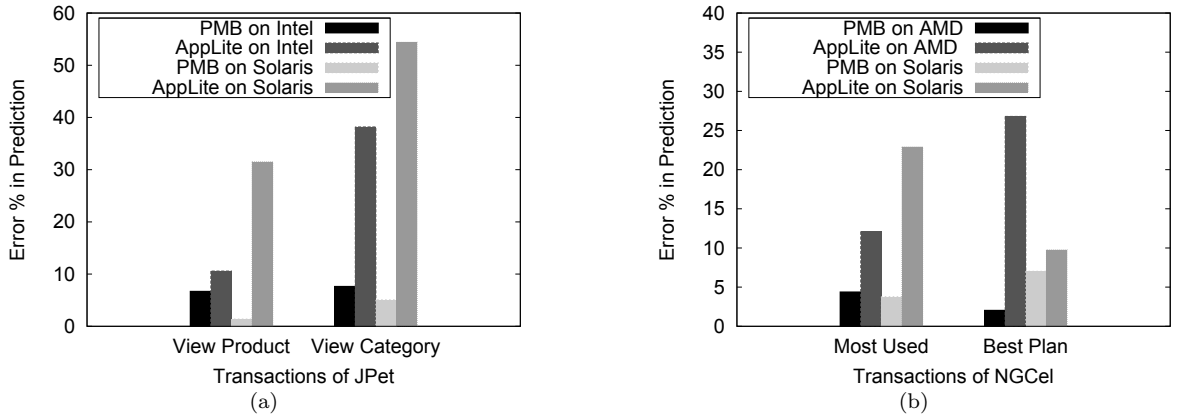

Figure 3: Comparison of PMB with AppLite

Further, it attempts to mimic processing of result sets. It is also seen that AppLite performs poorly for transactions where network I/O has a large contribution to CPU consumption. The reason behind this is that the ratio of service demands of AppLite between two platforms increases with the higher network load.

In Figure 4.2(b), the error percentages in prediction of service demands for PMB and AppLite are compared for NGCel application with 500 users. Though in this case the maximum error in prediction is around 27% for AppLite, PMB has the highest error of 7%. Thus, we achieve much better accuracy using PMB.

4.3 SPECjEnterprise2010 Results

In order to validate our approach with real-like applications, we develop PMB for SPECjEnterprise2010 which is an industry standard benchmark developed by the Standard Performance Evaluation Corp (SPEC). The SPECjEnterprise2010 application provides a business scenario for an automobile manufacturer [1]. Though the application supports the Supplier domain for the SCM, the Manufacturing domain and the Orders domain for the CRM, PMB is built only for the Orders domain since this is the only domain that is accessible through web interface. PMB runs on glassfish application server with MySQL as database server and is designed to support Sell and Purchase transactions. In Figure 4, we show the error percentage in predicting service demand of SPECjEnterprise2010 using that of PMB on AMD and Solaris platform. In this case, Intel platform is the source platform. We observe that independent of concurrency, the error% is less than 12%. We also obtain similar prediction

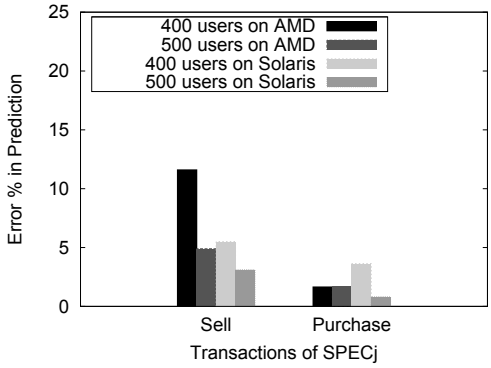

Figure 4: Error % of PMB for SPECjEnterprise2010

on the Solaris platform and verify that the error% is less than 5%.

4.4 Predicting Performance Metrics

Our proposed methodology is complimentary to performance modeling technique and the predicted service demands can be utilized as input to a modeling technique to finally predict performance metrics such as throughput, response time and CPU utilization. Since PMBs are used to predict service demands for discrete user levels, we choose a modified version of mean value algorithm [7] that uses exact MVA but allows load dependent service demands. The algorithm accepts an array of service demands at distinct load values. For this experiment, the SPECjEnterprise2010

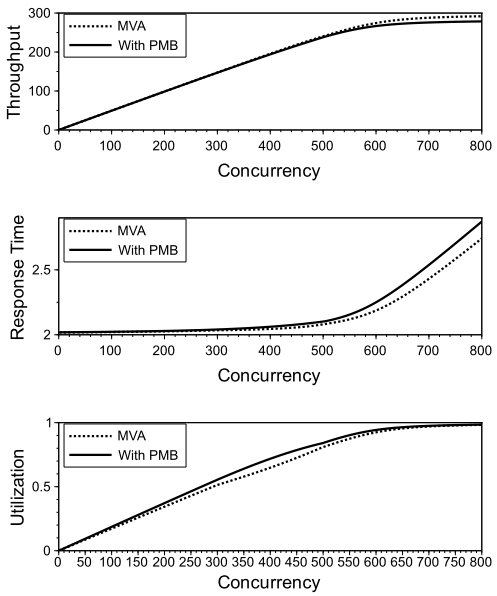

Figure 5: Performance prediction using MVA for SPECj

benchmark is executed on a AMD platform for 300, 400 and 500 users. We observe that for all user levels up-to 700 users, the average error in throughput and response time is found to be less than 5%. Figure 5 compares throughput, response time (in secs) and CPU utilization (scale of $0-1$) at the application server as predicted by this MVA algorithm for two scenarios (1) when actual service demands of SPECjEnterprise2010 benchmark is used and (2) when PMB is used to predict service demands. We can verify that PMB is able to provide same level of accuracy in actual performance prediction as the original application.

4.5 Result Analysis

Some important observations are as follows: (1) for a given transaction, service demand of the PMB is usually less than that of the actual application. This is because the PMB does not represent the business logic of the application, only its significant lower level method calls. (2) The magnitude of error varies from one transaction to another transaction. However, the PMB is able to achieve less than 10% error on an average. (3) The AppLite benchmark performs worse than PMB in almost all situations. For the Solaris platform, the AppLite prediction error is as high as 54% for some transactions. This is because the service demand of an application performing different types of CPU operations varies differently from source to target platforms than an application performing only integer operations.

5. CONCLUSIONS

In this paper, we presented a methodology for generating simple performance mimicking benchmarks which are used for cross-platform performance prediction of multi-tier applications. Our approach requires only application level profiling. The PMBs invoke similar method calls as the application at the technology layer and also mimic the network calls. Thus, the PMBs incur similar CPU consumption as the application on a given platform. The ability of PMBs to predict service demand accurately has been demonstrated for simple Java web application, strut-based framework and J2EE application. We expect the benchmarks to be more generic and reusable, and we only would require an appropriate input file to be generated based on the profiling information corresponding to a specific application. We are in the process of building a repository of parameterized benchmarks for various technology stacks in the J2EE domain and include other technology stacks such as .NET and PHP etc. Finally, to address migration of all the tiers in a multi-tier application, the proposed approach requires to be complemented with a strategy for predicting performance of database tier from source to target platform.

6. REFERENCES

[1] *Standard Performance Evaluation Corporation.* https://www.spec.org/jEnterprise2010/.

[2] A. Brunnert, C. Vogele, and H. Krcmar. Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications. In *European Workshop, EPEW*, 2013.

[3] A. Deshpande and V. Apte. Perfcenter: a performance modeling tool for application hosting centers. In *Proceedings of the 7th Int. Workshop on Software and Performance, WOSP*, pages 79–90, 2009.

[4] S. Duttagupta, R. Virk, and A. Khanapurkar. Performance Extrapolation Across Servers. In *Computer Measurement Groups Conference*, 2013.

[5] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. De Bosschere. Performance prediction based on inherent program similarity. In *Proc. of Conference on Parallel architectures and compilation techniques*, pages 114–122. ACM, 2006.

[6] A. Joshi, L. Eeckhout, R. Bell, and L. John. Performance cloning: A technique for disseminating proprietary applications as benchmarks. In *Proceedings of IEEE conf on Workload Characterization*, 2006.

[7] A. Kattepur and M. Nambiar. Performance modeling of multi-tiered applications with varying service demands. In *APDCM, IEEE IPDPS Workshops*, 2015.

[8] S. Kounev. Performance Modeling and Evaluation of Distributed Component-based Systems Using Queuing Petri Nets. *IEEE Trans. on Software Engineering*, pages 486–502, 2006.

[9] K. Krogmann, M. Kuperberg, and R. Reussner. Using genetic search for reverse engineering of parametric behavior models for performance prediction. *Software Engineering, IEEE Transactions on*, 36(6):865–877, 2010.

[10] S. Sharkawi, D. Desota, R. Panda, R. Indukuru, S. Stevens, V. Taylor, and X. Wu. Performance projection of HPC applications using SPEC CFP2006 benchmarks. In *Proc. of IPDPS*, 2009.

[11] B. C. Tak, C. Tang, H. Huang, and L. Wang. Pseudoapp:performance prediction for application migration to cloud. In *IFIP Symposium on Integrated Network Management*, pages 303–310, 2013.

[12] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner. Justrunit: Experiment-based management of virtualized data centers. In *Proc. of USENIX Annual technical conference*, 2009.