Full-Stack Performance Model Evaluation using Probabilistic Garbage Collection Simulation

Felix Willnecker
Andreas Brunnert
fortiss GmbH
Guerickestr. 25
80805 München, Germany
{willnecker,brunnert}@fortiss.org

Bernhard Koch-Kemper
Helmut Krcmar
Technische Universität München
Boltzmannstr. 3
85748 Garching, Germany
{kochkemp,krcmar}@in.tum.de

ABSTRACT

Performance models can represent the performance relevant aspects of an enterprise application. Corresponding simulation engines use such models for simulating performance metrics (e.g., response times, resource utilization, throughput) and allow for performance evaluations without load testing the actual system. Creating such models manually often outweighs their benefits. Therefore, recent research created performance model generators, which can generate such models out of Application Performance Management software. However, a full-stack evaluation containing all relevant resources of an enterprise application (Central Processing Unit, memory, network and Hard Disk Drive) has not been conducted to the best of our knowledge. This work closes this gap using a pre-release version of the next generation industry benchmark SPECjEnterpriseNEXT of the Standard Performance Evaluation Corporation as example enterprise application, the Palladio Component Model as performance model and the performance model generator of the RETIT Capacity Manager. Furthermore, this work extends the generated model with a probabilistic garbage collection model to simulate memory allocation and releases more accurately.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: measurement techniques, modeling techniques

Keywords

Performance Evaluation, Palladio Component Model, Java, Enterprise Applications, I/O Performance Simulation, Garbage Collection Simulation

1. INTRODUCTION

Evaluating the performance of an enterprise application (EA) requires either load testing this application or creating a performance model to simulate the performance metrics (e.g., response times, resource utilization, throughput) of a system. Simulations are often less cost intensive as they do not require a full scale deployment environment [5]. Performance models and corresponding simulation engines have been introduced to the scientific community [1]. Resource profiles based on the Palladio Component Model (PCM) have demonstrated to accurately represent the Central Processing Unit (CPU) and network demand of EAs [1, 3]. Resource profiles are generated using a performance model generator, as a manual creation of such profiles often outweighs their benefits [5]. These profiles already introduce heap, as the most relevant aspect of memory in EAs, and contain a Hard Disk Drive (HDD) representation [3]. However, these concepts have not been evaluated. This work demonstrates the heap and HDD modeling and simulation capability of resource profiles using the industry benchmark SPECjEnterpriseNEXT[1] as EA and the performance model generator of the RETIT[2] Capacity Manager .

An accurate model needs to take all relevant resources (CPU, network, memory, and HDD) into account. The generation of such a model requires to monitor a running artifact using Application Performance Management (APM) software [8]. This data can be collected during small scale test runs and scaled to the expected workload [3]. In this work we use the RETIT Java EE Monitoring and extend it with HDD demand measurements for Linux based systems. Furthermore, the heap model of resource profiles is extended using a young and old generation garbage collection (GC) model and corresponding simulation to increase the correctness of the heap simulation.

2. GARBAGE COLLECTION MODEL

The RETIT performance model generator already contains a heap representation. This representation is based on a simple memory model where allocated heap is immediately freed when the contained objects are released [2]. The PCM entity *Passive Resource* is used for this purpose [1]. Each model contains one component called *Heap* that contains one *Passive Resource* called *HeapSpace* with the

[1]SPECjEnterpriseNEXT is a trademark of the Standard Performance Evaluation Corp. (SPEC). The SPECjEnterpriseNEXT results or findings in this publication have not been reviewed or accepted by SPEC, therefore no comparison nor performance inference can be made against any published SPEC result.
[2]`http://www.retit.de/`

.

available memory in the Java Virtual Machine (JVM) heap space. Before and after each operation invocation an allocation and free operation is called with the number of bytes allocated or released. The values are derived from monitoring and represent the mean bytes used in this operation invocation. This implementation has certain disadvantages: (i) each JVM manages its own heap space, therefore each JVM instance in the performance model needs a separate *Heap* component (ii) the memory in a JVM is only cleaned when GC occurs. Only after such a GC run a certain amount of heap is freed.

We change the model in order to address the aforementioned disadvantages. Each JVM representation uses its own *Heap* component representing its own heap space. We measure the mean time between different GC runs and the average number of bytes released. We distinguish between two types of GC: young and old generation GC [6]. Even though GC implementations may vary throughout different JVM versions and types, these two GC types occur in most of the GC implementations [6]. For each GC type we add an *Open Workload Scenario* that calls the GC operation in the *Heap* component. The inter-arrival time between these scenarios is the mean time between two GC runs of the same type. The operation call has one parameter: the average number of bytes released per GC run. The operation releases the provided number of bytes in the *HeapSpace*. This probabilistic approach converges the generated performance model to the real memory management in the JVM.

3. EVALUATION

The SPECjEnterpriseNEXT industry benchmark is the successor of the SPECjEnterprise2010 benchmark. Both are Java Enterprise Edition (EE) applications typically used to benchmark the performance of different Java EE application servers. We use a pre-release version[3] of the SPECjEnterpriseNEXT as example EA. This application represents an insurance policy holder that manages car insurances. The application consists of three different components (Insurance Domain, Vehicle Service, Insurance Agent) as depicted in Figure 1. Each component is deployed as one deployment unit in one application server running in a virtual machine (VM). Each VM has 4 CPU cores, runs the Java EE application server Wildfly 8.1.0, the embedded database Derby 10.11.1.1, and has 8 GB of heap for the JVM. The operation system is OpenSuse 13.2 (x86_64) and a 1 GBit/s network connection is used for communication between the different VMs.

The Insurance Customer Driver is based on Faban[4] and executes five different business transactions on the Insurance Domain server, which triggers JAX-RS[5] REST calls on the other two servers [4]. We executed a measurement run without the RETIT Java EE monitoring to minimize instrumentation overhead and use the results to validate our simulation. The response times of the business transactions were measured on the driver. CPU and heap utilization were measured using Java Management Extensions (JMX) and the HDD demand using IOTop[6]. These measurements have been executed for 100, 120, 140 and 160 users in a 17

[3]version from 29.06.2015

[4]`http://faban.org/`

[5]`https://jax-rs-spec.java.net/`

[6]`http://guichaz.free.fr/iotop/`

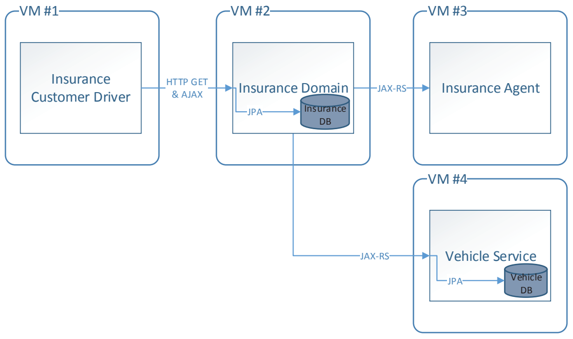

Figure 1: SPECjEnterpriseNEXT deployment

min interval with 5 min ramp up and 2 min ramp down phase. The model generation is based on a run with 100 users and activated RETIT Java EE monitoring on system-entry-point level. The monitoring is extended to derive the resource demand of the HDD. Therefore, the procfs[7] system is used, a pseudo file system containing the read and write bytes per thread.

To calculate the resource capacity of the HDD we use the bonnie++ 1.97 benchmark[8]. For calculating the network latency and bandwidth we use LMBench 3.0 [7]. The number of CPU cores is stored in the resource environment replicas setting for each *Resource Container* and the available heap is stored in the *Passive Resource* of the corresponding heap component for each JVM.

The usage model is based on an *Open Workload Scenario*. The inter-arrival time per business transaction IAT_{BT} is calculated based on the $SteadyStateTime$ of 600s, the total number of calls per business transaction $TotalCalls_{BT}$ and the 100 users of the generation run ($UsersGeneration$) as depicted in Equation 1. The number of users in the simulation $UsersSimulation$ is up-scaled from 100 to 160 users in steps of 20.

$$IAT_{BT} = \frac{SteadyStateTime}{\frac{TotalCalls_{BT}}{UsersGeneration} * UsersSimulation} \quad (1)$$

The inter-arrival time for the garbage collection IAT_{GC} is calculated in a similar way. Instead of the total number of business transactions we use the number of GC events $TotalEvents_{GC}$ intercepted during the generation run as shown in Equation 2. The calculation is conducted for each GC type (young and old generation GC). Again, the number of users $UsersSimulation$ in the simulation is up-scaled from 100 to 160 users in steps of 20.

$$IAT_{GC} = \frac{SteadyStateTime}{\frac{TotalEvents_{GC}}{UsersGeneration} * UsersSimulation} \quad (2)$$

Table 1 shows the results for 100 and 160 users. The complete results are available online[9]. The heap simulation has only been conducted for the Insurance Domain and the Vehicle Service as the heap demand for the Insurance Agent server is almost 0. The relative error of the heap simulation is between 10.93% and 1.03%. HDD demands only

[7]`https://www.kernel.org/doc/Documentation/filesystems/proc.txt`

[8]`http://www.coker.com.au/bonnie++/`

[9]`http://download.fortiss.org/public/pmwt/SSP2015/FullStack_EvaluationResults.(xlsx/pdf)`

Table 1: Measurement and simulation results

Resource	User	Metric	Insurance Domain	Vehicle Service	Insurance Agent
CPU	100	**Measured utilization**	43.80%	51.57%	46.13%
		Simulated utilization	40.48%	48.29%	43.04%
		Relative error	7.58%	6.38%	6.70%
	160	**Measured utilization**	70.19%	77.91%	71.73%
		Simulated utilization	64.75%	77.25%	68.88%
		Relative error	7.75%	0.85%	3.97%
Heap	100	**Measured demand**	1458.58 MB	1435.41 MB	-
		Simulated demand	1299.22 MB	1319.41 MB	-
		Relative error	10.93%	8.08%	-
	120	**Measured demand**	1360.19 MB	1304.46 MB	-
		Simulated demand	1296.29 MB	1317.88 MB	-
		Relative error	4.70%	1.03%	-
HDD	100	**Measured demand**	0.23%	0%	0%
		Simulated demand	0.21%	0%	0%
		Relative error	10.72%	0%	0%
	160	**Measured demand**	0.34%	0%	0%
		Simulated demand	0.35%	0%	0%
		Relative error	0.65%	0%	0%

occur on the Insurance Domain server. Even though, the utilization for the HDD is relative low the simulation delivers accurate results for this resource. The relative error here is between 10.72% and 0.65%. The relative CPU utilization error is below 10%. The utilization in the GC models improves the CPU utilization simulation compared to previous research [9]. The response time error based on the median of simulation and measurements is between 0.16% and 51.65% depending on the business transaction and user scale.

4. CONCLUSIONS

The improvements and extensions to the model generated by the RETIT performance model generator prove to be accurate for all resources. Even though the resource utilization simulation is accurate, the response time error leaves room for improvement. The parameter of a request as well as the number of available database and server threads have impact on the response time simulation. In total the response times of the deployment are quite high, as the embedded database is not very scalable. A setup with an dedicated database server appears to be a better solution but requires additional APM measurement tools or resource demand estimations [9]. The probabilistic GC model delivers promising results, however the average number of bytes could be eliminated as an input parameter for the GC operations. This requires, that the actual number of bytes ready to be freed is stored in a variable of the model. Furthermore, the current evaluation is based on system-entry-point level. The performance model generator of the RETIT Capacity Manager provides component-level model generation which could improve the simulation results presented in this work.

5. ACKNOWLEDGMENTS

This work has been supported by the DevOps Performance Working Group of the Standard Performance Evaluation Corporation (SPEC).

6. REFERENCES

[1] S. Becker, H. Koziolek, and R. Reussner. The Palladio Component Model for Model-Driven Performance Prediction. *Journal of Systems and Software*, 82(1):3–22, 2009. Special Issue: Software Performance - Modeling and Analysis.

[2] F. Brosig, F. Gorsler, N. Huber, and S. Kounev. Evaluating approaches for performance prediction in virtualized environments. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*, pages 404–408, Aug 2013.

[3] A. Brunnert and H. Krcmar. Continuous Performance Evaluation and Capacity Planning Using Resource Profiles for Enterprise Applications. *Journal of Systems and Software*, 2015, `http://dx.doi.org/10.1016/j.jss.2015.08.030`.

[4] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[5] S. Kounev. *Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction.* PhD thesis, Technische Universität Darmstadt, Germany, Aachen, Germany, 2005.

[6] P. Libič, L. Bulej, V. Horky, and P. Tůma. Estimating the impact of code additions on garbage collection overhead. In *Computer Performance Engineering*, volume 9272 of *Lecture Notes in Computer Science*, pages 130–145. Springer International Publishing, 2015.

[7] L. W. McVoy, C. Staelin, et al. lmbench: Portable tools for performance analysis. In *USENIX annual technical conference*, pages 279–294. San Diego, CA, USA, 1996.

[8] F. Willnecker, A. Brunnert, W. Gottesheim, and H. Krcmar. Using Dynatrace Monitoring Data for Generating Performance Models of Java EE Applications. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015)*, pages 103–104, New York, NY, USA, 2015. ACM.

[9] F. Willnecker, M. Dlugi, A. Brunnert, S. Spinner, S. Kounev, W. Gottesheim, and H. Krcmar. Comparing the accuracy of resource demand measurement and estimation techniques. In *Computer Performance Engineering*, Lecture Notes in Computer Science, pages 115–129. Springer International Publishing, 2015.