Impact of Java Application Server Evolution on Computer System Performance

Peng-fei Chuang[†], Celal Ozturk[§], Khun Ban[†], Huijun Yan[†], Kingsum Chow[†], Resit Sendag[§]

[†] Intel Corporation; {peng-fei.chuang, khun.ban, huijun.yan, kingsum.chow}@intel.com
[§] Dept. of Electrical, Computer and Biomedical Eng., Univ. of Rhode Island; {cozturk, sendag}@ele.uri.edu

***Abstract* — Advancement in hardware and Java Application Server techniques has prompted the designs and implementations of enterprise Java applications with greater capability. The increase in their versatility and the likely needs for backward compatibility consequently introduce higher code complexity and can potentially increase the code size. In this paper, we compare the high-level code changes in SPECjAppServer2004 and in SPECjEnterprise2010, and its effects on overall system behavior. We characterize the workload differences in three different layers – the high-level software characteristics, the system behavior, and low-level characteristics. Our data show that SPECjEnterprise2010 has larger memory footprint and there is a shift in the execution time from OS to Java code and JVM (garbage collection and JIT compilation) during steady state of the program execution. We observe 74% increase in the JIT compiled code and 59% increase in the number of classes used at steady state. This behavior change also impacts the performance of memory hierarchy and other microarchitecture level components such as the branch predictor. We observe 61% and 10% increase in the ITLB misses per instruction (MPI) for large pages and small pages, respectively, and 12% increase in the branch mispredictions per instruction. Finally, last level cache shows 12% increase in MPI.**

I. INTRODUCTION

The advancement in Java Enterprise Edition (Java EE) application servers and state-of-the-art processors has prompted the designs and implementations of enterprise Java applications with greater capability. Modern enterprise Java applications are designed to support more versatile application models, complex workflows, inter-application communication and inter-product-family integration. They also emphasize on the ease of use through predefined libraries and modules, as well as programming language annotation. These help software developers add complex new features. In some case, the use of extendable APIs can add the appealing capability of customization, which enables the customer to tailor the products to suit their business needs. New software releases can also introduce the need for backward compatibility. This in turn increases the size of the code. Modular approach/design philosophy of software development improves the ease of use. However, it may generate larger number of access classes and requires more runtime optimization. With the introduction of 32-nm processors, we further increase the level of the integration of multi-core processors as well as add more features to support software functionality (e.g. virtualization, encryption, etc).

In response to the advancement in both the hardware and software, SPECjEnterprise2010 (SjE10) [1] is introduced to benchmark the performance of a modern Java EE application server and the underlying hardware. SjE10 replaces SPECjAppServer2004 (SjAS04) [2] and targets for Java EE 5.0 standard. Due to the fast improvement in the Java EE standards, Standard Performance Evaluation Corporation (SPEC) retires the SjAS04 and use SjE10 as the new standard to evaluate the performance of Java EE application servers. Understanding the nature of such commercial workloads is critical to develop the next generation of servers and identify promising directions for systems and software research.

This paper analyzes the differences between SjAS04 and SjE10 running on a high-end server. The high-level software characteristics, the system behavior, and low-level characteristics are discussed for SjE10 in comparison with SjAS04. An effective system tuning methodology [3] is used to tune the performance for both workloads.

The evaluation of hardware performance monitor data shows that at the system level, SjE10 demands more system resources, such as heap memory, than SjAS04 when the application server achieves the same CPU utilization for both workloads. The two workloads show very similar execution profiles with the SjE10 spending a larger proportion of the execution on Java code and JVM and smaller proportion of the execution time on OS code. At the code level, in comparison with the SjAS04, smaller proportion of the SjE10 code is the Java EE container and application and a larger proportion of it is Java Library. During run-time, JVM Just-In-Time compiled (JIT'ed) code is much larger compared to SjAS04. Also, larger number of classes are loaded and used during steady-state. At the microarchitectural level, SjE10 has a much higher path length. It also experiences higher percentage of last-level cache (LLC) misses and instruction translation lookaside buffer (ITLB) misses. However, SjAS04 has higher data TLB (DTLB) miss rate. Finally, SjE10 has higher branch misprediction rate.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of SjAS04 and SjE10 and explains the differences. In Section 3, we discuss the experimental setup. Section 4 describes the comparison for high-level software characteristics. Section 5 shows the system behavior. In Section 6, we present the low-level characteristics. Section 7 surveys the related work. Finally, Section 8 concludes.

II. SPECJAPPSERVER2004 AND SPECJENTERPRISE2010

SjAS04 and SjE10 are multi-tier benchmarks applications developed by SPEC for measuring the performance of Java EE technology-based application servers. They are designed to exercise the Java EE Application Server, the Java Virtual Machine (JVM), as well as the server systems under test (SUT). SjE10 is the newest member of the series of SPEC's Java Application Server benchmarks and is a replacement of SjAS04 benchmark.

Both applications implement a web-based user interface that allows the customers of the applications, in our case automobile dealers, to keep track of their accounts, keep track of dealership inventory, sell automobiles, manage a shopping cart and fulfill purchase orders of automobiles.

SjAS04 exercises all major Java EE 1.3 technologies implemented by compliant application such as the web container, the EJB2.0 container, Java Message Service (JMS) and message driven beans (MDB), transaction management, and database connectivity. It also heavily exercises all parts of the underlying infrastructure that make up the application environment, including hardware, JVM software, database software, JDBC drivers, and the system network. Basically, SjAS04 is a Java EE benchmark meant to measure the scalability and performance of Java EE servers and containers.

SjE10 is an enhanced version of the SjAS04 that includes a modified workload and conforms to Java EE 5.0 standard. SjE10 is designed to test the performance of a representative Java EE application and each of the components that make up the application environment, e.g., hardware, application server, JVM, database.

The main difference between the two benchmarks is that SjAS04 is a Java EE 1.3 application and uses a web layer and EJBs for the clients' interactions with the server. In the SjE10 benchmark, the load drivers access the application through a web layer (for the dealer domain) and through EJBs and Web Services (for the manufacturing domain) to stress more of the capabilities of the Java EE application servers. In addition, SjE10 adds more extensive use of the JMS and MDB infrastructure.

Even though these Java Application Server benchmarks are designed with the same philosophy to measure the performance of Java EE technology-based application servers, their performances are not comparable to each other due to different optimization opportunities and constraints imposed for each of them.

III. EXPERIMENTAL SETUP AND METHODOLOGY

This section describes how we collected the data and characterized the software evolution between SjAS04 and SjE10 benchmarks. Our main focus in this paper is to characterize the software evolution between these two commercial workloads and to identify the impact of this evolution on the system that they are running. The benchmark scores are not provided in this paper because the performances of these workloads are not comparable and should not be compared [1].

To identify the impact of the software evolution on the system, we used the same hardware configuration and the same software stack for both workloads. We also used the same database servers, drivers and emulators which are powerful enough for both workloads.

Table 1 shows the configuration setup for the application server. Our SUT is an Intel® Xeon® X7560 running the Red Hat Enterprise Linux 5.5 OS, a leading JVM and a leading Java EE application server. The database server is deployed on a second Intel® Xeon® X7560 system, and is tuned such that it is not the performance bottleneck. Furthermore, an effective performance tuning methodology [3] was employed to achieve similar CPU utilizations on the application server in the experiments. The OS and the JVM have been configured to use large pages (2MB compared to the default size of 4KB). Large pages improve performance of long-running Java applications with a variety of heap sizes and especially applications with large heaps. We used aggregately 15GB of heap size per socket and we applied the same memory optimization parameters for both workloads. Our application server CPU saturation values are very similar between the workloads. To achieve fairness, we also normalized some of the data we present in this paper by the corresponding CPU saturation value. In SjAS04 and SjE10, many factors such as database performance, thread queues on the driver, and the response times, can affect the application server CPU utilizations. Because the two benchmarks in this paper are Java EE application servers, our analyses focus on the application server performance and we exclude analyses on the database and driver sides.

Table 1 Hardware and Software configurations for the application server (SUT)

Hardware	4P (4 cores each) Intel® Xeon® X7560 @2.26GHz. 64GB RAM. Hyper-threading: On
Software	Red Hat Enterprise Linux 5.5 a leading JVM a leading Java EE application server

Data from various components of the system and its software stack have been collected using a suite of standard

performance tools. For example, Intel® VTune™ Performance Analyzer [4] provides a high-level view of system performance with information about CPU utilization and memory. We use a hardware performance counters monitoring tool that makes it possible to collect accurate, sampled hardware data from counters that simultaneously track various processor events. We also collected runtime information emitted by the JVM, which provides memory/heap utilization statistics. Finally, the JVM had instrumentation (without noticeable overhead) for collecting code coverage statistics.

IV. High-level Software Analysis

In this section, we compare the SjAS04 and SjE10 workloads in terms of their JIT'ed code size, the number of classes used and code coverage. We use transactions to normalize the data because a transaction is a unit of work for both workloads and it provides us a better picture in characterizing the software evolution between the benchmarks. However, it is important to note that one transaction of SjAS04 is not equivalent to a transaction of SjE10 since they are different workloads.

A. Number of classes loaded and size of the JIT'ed codes

A JVM JIT compiler is a code generator that converts Java bytecodes (executed by the interpreter) into native machine code. A JVM can use runtime profile to guide the JIT compilation and re-optimize portion of the code to generate more efficient codes. JIT compilation is usually invoked only for the parts of an application which are performance bottlenecks.

JIT'ed code sizes of SjAS04 and SjE10 are 87MB and 152MB, respectively. SjE10 has 74% larger JIT'ed code size compared to SjAS04, which is an indication that the workload has become more complex. Figure 1 compares the number of classes used by SjAS04 and SjE10. As shown in the Figure, Sj04 uses 10,431 classes and SjE10 uses 14,301 classes throughout their execution. This 34% increase in the number of classes used also explains the major code size increase in SjE10.

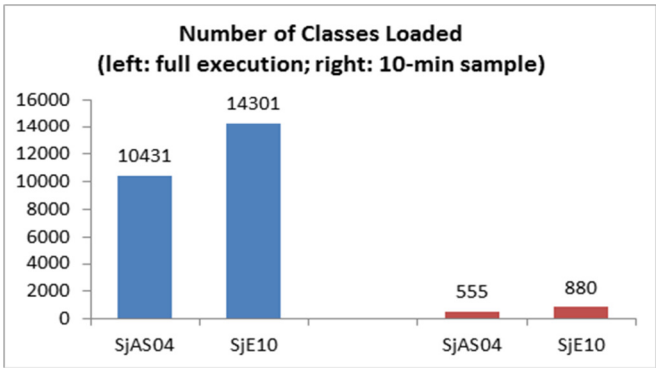

Figure 1 Comparison of the numbers of classes the benchmarks use throughout the execution and at steady-state

Since a number of classes are used in the initialization of these workloads, in Figure 1 we also compare the number of the classes loaded for SjAS04 and SjE10 during steady state. The number of classes in steady state is measured during a 10-minute sampling period. Since each transaction takes a few seconds to complete for both benchmarks, the samples cover all major code regions. The increase in the number of classes loaded is more pronounced during steady state (59% during steady state versus 34% overall). We can also see that both workloads use a very small portion of their codes during steady state: 5.3% for SjAS04 and 6.1% for SjE10.

B. Common and Unique Classes

To determine whether or not the code size change is primarily due to the loaded classes that are unique to the execution of workloads, we analyzed the common and unique classes loaded by SjAS04 and SjE10 during the full execution of the two workloads and compare them in Table 2. Common classes are the ones that are common to the execution of both workloads. Unique classes to SjE10 are only loaded during execution of SjE10, but are not loaded during execution of SjAS04 and vice versa.

Results show that common class sizes also increase significantly (from 65MB to 91MB) as well as the increase in the number and sizes of unique classes. We believe the reason for the increase in the size of the common classes that are loaded during execution of both SjAS04 and SjE10 is that the new workload exercises the new Java EE 5.0 standard and therefore new methods may be added to the common classes to facilitate this support[1]. The Java EE application server that we use supports both Java EE 1.3 and 5.0 standards.

Table 2 Common and Unique Classes Loaded

	SjAS04		SjE10	
	count	size	Count	size
Common	7453	65MB	7453	91MB
Unique	2978	22MB	6848	61MB

C. Code Coverage

Finally, based on which code module each class loaded at steady state belongs to, we analyzed and categorized the code in SjAS04 and SjE10 in Figure 2[2]. The data shows, at the code level, in comparison with the SjAS04, smaller proportion of the SjE10 code is the Java EE application server (46% versus 29%) and application (7% versus 2%), but a larger proportion of it is Java libraries (29% versus 34%).

[1] We did not have access to the Java EE application server source code. Therefore, we did not analyze this further.

[2] The class profiling was done by using JVM Tool Interface. The tool does not introduce noticeable performance interference.

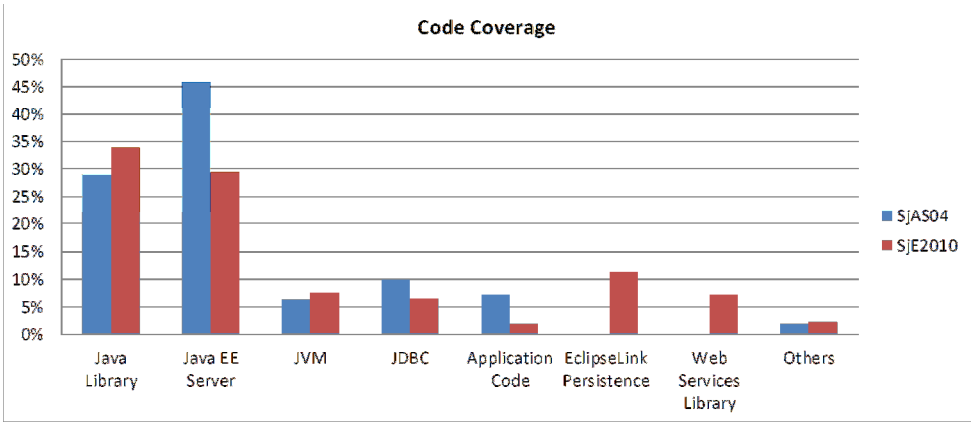

Figure 2 Code coverage: Categorization of the classes loaded into their functions.

Since only 7% and 2% of the classes loaded were from the SjAS04 and SjE10 benchmarks, respectively, the performance of the application codes themselves are relatively less important to the overall system performance. What is significant is how they utilize application server features.

V. System Behavior

This section focuses on the application system behavior comparison, including CPU utilization, heap allocation rate, and execution profiles between SjAS04 and SjE10. As in Section IV, all the measured statistics shown in this section are per transaction based.

A. Execution time breakdown

The software stack running on the application server includes a variety of components – the OS, the Java Application Server, the JVM, and other small components. All the software components affect the workload execution and the overall system behavior. To show the contributions of major software components to the retired instruction counts, we used Intel® VTune™ Performance Analyzer to collect the execution profiles. The data is collected off the garbage collection (GC) pause time to avoid the GC's impact on the execution profile. Due to the use of pause the world GC, the execution profiles off and on the GC time may behave differently, but the analysis of the execution profile on the GC time can be performed in a similar way.

Figure 3 presents the execution time breakdowns for SjAS04 and SjE10. We can see that both benchmarks have similar execution time breakdown among the major software components. The breakdown helps understand what software components really take the execution resource and can be the possible optimization targets. Java retires 74% of the overall instructions for SjAS04 and 77.1% of the total instructions for SjE10. The OS is the second largest component and it weighs 20.3% and 15.8% of the retired instructions for SjAS04 and SjE10, respectively. The number of retired instructions by the JVM increases from 5.7% for SjAS04 to 7.1% for SjE10. Overall, we observe a shift in the execution time from OS to Java code and JVM (GC and JIT compilation) during steady state of the program execution.

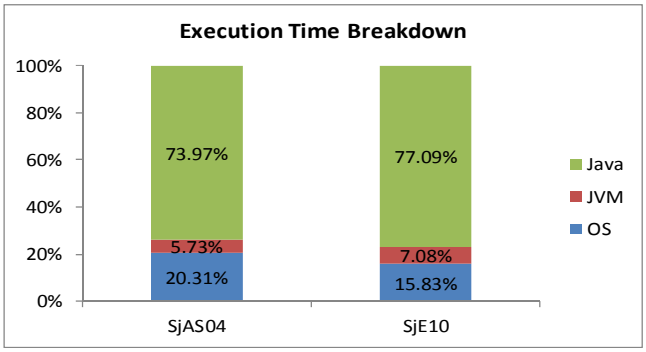

Figure 3 Breakdowns of execution time into Java (application code, JVM (GC, JIT, etc.) and OS

B. CPU utilization

Figure 4 shows the breakdown of application server CPU utilization for the two benchmarks. The kernel and user CPU time breakdowns are very similar between the two benchmarks.

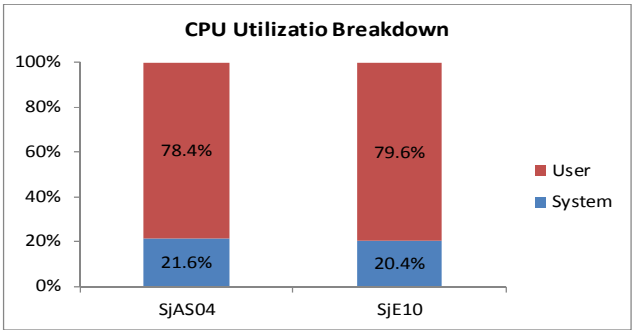

Figure 4 CPU utilization breakdown among user time and kernel time on both workloads

The kernel/user execution breakdown is 21.6%/78.4% for SjAS04 and 20.4%/79.6% for SjE10. The two workloads have different response time requirements and transaction

queues, which can affect the kernel/user execution time breakdown.

C. Allocation Rate

The memory usage is mainly due to the heap usage by Java. SjE10 is about 1.7 times more demanding with 5.5 GB/sec compared to the 3.3 GB/sec for SjAS04. This is an indication that SjE10 has a larger memory footprint.

VI. Low-level Characteristics

In addition to software optimizations, microarchitecture is an important factor of the application server performance. Understanding the microarchitecture performance characteristics can help understand the strengths and the weaknesses of the microarchitecture and design better future processors. In this section, we examine and compare the processor-independent microarchitecture performance characteristics of SjAS04 and SjE10.

A. Path Length

The path length is the number of instructions retired per transaction by the application server. Our data show that SjE10 has a 47% larger path length than SjAS04.

B. Instruction type distribution

During a 10-minute sampling period in the steady state, SjAS04 and SjE10 have similar distribution in instruction types, as shown in Figure 5. The largest change observed is the increase in the proportion of number of branches in SjE10. Branch instructions correspond to 16.3% for SjAS04 and 19.6% of the instructions for SjE10.

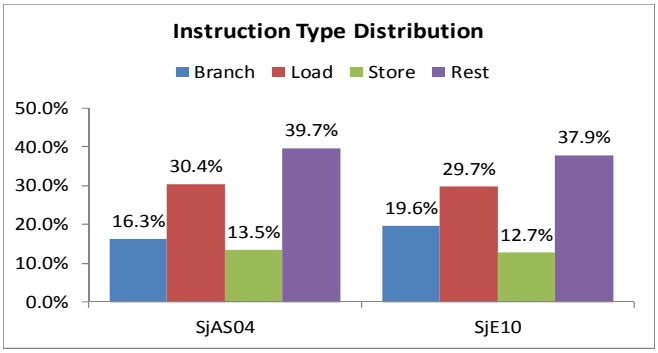

Figure 5 Instruction type breakdown for both benchmarks

The increase in the proportion of branches is likely due to the workload design changes. The distributions of PC-relative branches, returns, and indirect branches are fairly similar between the two benchmarks (data not shown). Finally, we also observe a reduction in the proportion of memory operations, from 43.9% for SjAS04 to 42.4% for SjE10.

C. Memory Hierarchy

The changes in TLB behaviors from SjAS04 to SjE10 are shown in Figure 6. The ITLB miss rate is about 61% higher for large pages and 10% higher for small pages. SjE10 has smaller DTLB miss rate – 10% smaller for small pages and 1% for large pages. Utilizing large pages for JIT'ed code and other components of execution stack leads to additional performance gain. However, address translation performance of SjE10 suggests that there is room for improving ITLB hit rates by implementing object locality optimizations. Increasing the sizes of ITLBs so that they could better maintain large working sets could further improve overall performance.

Figure 7 shows the ratios of miss rates of different caches between SjAS04 and SjE10. In both benchmarks, there is much higher instruction cache miss rate than the data cache miss rate (data not shown). SjE10 has 15% lower IL1 MPI and 14% higher DL1 MPI than SjAS04. Also, L2 MPI is 8% lower and LLC MPI is 12% higher for SjE10. The higher LLC MPI indicates higher pressure on the memory hierarchy, overall.

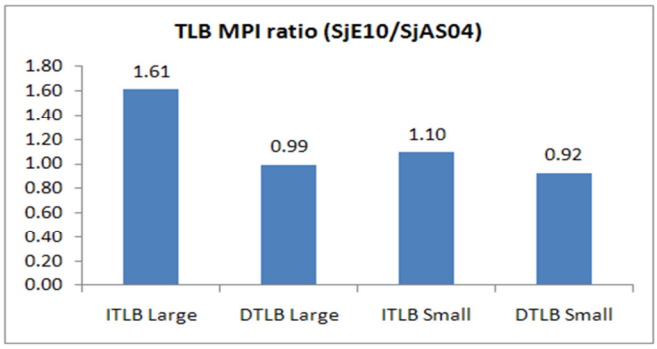

Figure 6 Ratio (SjE10/SjAS04) of instruction and data TLB MPI (large and small pages)

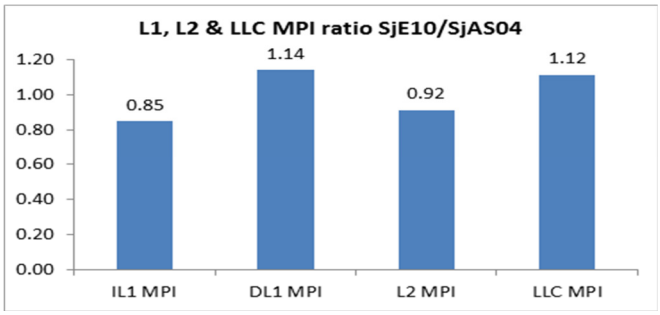

Figure 7 Ratios (SjE10/SjAS04) of IL1, DL1, L2 and LLC MPIs

VII. Related Work

SjE10 and SPECjAppServer-series (SjAS) benchmarks are commercial benchmarks for evaluating the performance of Java EE application server and the underlying hardware platform. SjE10 [1] is a replacement of the SjAS04 benchmark. The previous SjAS versions include SjAS01 (derived from ECperf) [5], SjAS02 [6], and SjAS04 [2].

Overall, these workloads are important server benchmarks that assist in the design of servers.

SPECjAppServer2004 benchmark has been widely studied [7] [8] [9]. In [7], Su et al. studied and compared the performance of SjAS04 and its predecessor, SjAS02. Their findings show that the new benchmark, SjAS04 has higher demand in system resources, such as network utilization and heap usage. This trend also continues in the transition from SjAS04 to SjE10. Shuf and Steiner [9] presented a detailed investigation in the performance of SjAS04 on Power4 architecture. Their findings show that on the studied platform, the system performance correlate more strongly to branch mispredictions, translation misses, and I-cache misses than the performance of other micro-architectural components. In [8], Shiv et al. analyzed the performance of SjAS04 on Intel Core 2 Duo platform. They concluded that cache/memory overhead of the CMP architectures may be a limiting factor of performance scaling. Based on their findings, they proposed a hardware mechanism that could potentially reduce cache misses. Stoodley [10] studied the differences between SjAS04 and SPECjbb2000 and SPECjvm98 applications. Their study of SjAS04 finds that the workload has a flat method profile. We also observed similar behavior for both SjAS04 and SjE10.

Earlier versions of SjAS benchmarks were also studied extensively [11] [12] [13] [14] [15]. Chow et al. [14] studied the variance of ECperf-like workloads and the relationship between Java application workload implementations and CPU designs. Karlsson et al. [13] performed a detailed architectural level analysis of SjAS01. Karlsson et al. [11] [12] also compared performance of SjAS01 with SPECjbb2000 and investigated the effects of scaling on a memory subsystem. Chow et al. [3] [15] developed a system performance tuning methodology for SjAS01 and SjAS02.

Java based server workloads other than SjAS benchmarks were also investigated by both academia and industry. Cain et al. [16] implemented Java-based TPC-W and studied its architectural behavior in both real systems and simulators. Another Java based two-tier benchmark, VolanoMark, was studied by Luo et al. [17]. The memory behavior and the architecture influence of SPECjvm98 were investigated in [18] and [19]. Finally, Li et al. [20] use a complete system simulation to characterize SPECjvm98 benchmarks.

VIII. CONCLUSION

This paper presents a detailed comparison of a new workload – SjE10 and its predecessor – SjAS04. Both workloads are complex multi-tier Java EE application server benchmarks. The two workloads are set up in the same hardware and software configurations. The comparison is performed when both workloads achieve similar application server CPU utilizations. The analysis is based on three layers: the high-level software characteristics, the system behavior, and the low-level microarchitecture performance characteristics.

Our data shows that at the code level, in comparison with the SjAS04, a larger proportion of the SjE10 code is the Java library, and Java EE application server and application code are relatively smaller. Also, JIT'ed code size is much larger compared to SjAS04 and a larger number of classes are loaded and used during steady-state. At the system level, SjE10 demands more system resources, such as heap memory. The two workloads show similar execution profiles with the SjE10 spending larger proportion of the execution on Java code and JVM, but less execution time in OS code. At the microarchitecural level SjE10 has a higher path length. It also experiences higher percentage of LLC and ITLB misses, and branch mispredictions.

REFERENCES

[1] [Online]. http://www.spec.org/jEnterprise2010/index.html

[2] [Online]. http://www.spec.org/jAppServer2004/index.html

[3] Kingsum Chow, Ricardo Morin, and Kumar Shiv, "Enterprise Java Performance: Best Practices," *Intel Technical Journal*, 2003.

[4] J. Reinders, *VTune Performance Analyzer Essential*.: Intel Press, 2005.

[5] [Online]. http://www.spec.org/jAppServer2001/index.html

[6] [Online]. http://www.spec.org/jAppServer2002/index.html

[7] L. Su, K. Chow, K. Shiv, and A. Jha, "A Comparison of SPECjAppServer2002 and SPECjAppServer2004," in *Proc. of the CAECW-8*, 2005.

[8] K. Shiv et al., "Addressing Cache/Memory Overheads in Enterprise Java CMP Servers," in *Proc. of the 10th IISWC*, 2007, pp. 66-75.

[9] Y. Shuf and I. M. Steiner, "Characterizing a Complex J2EE Workload: A Comprehensive Analysis and Opportunities for Optimizations," in *Preceedings of the 2007 ISPASS*, 2007, pp. 44-53.

[10] M. Stoodley, "Challenges to improving the performance of middleware applications," in *Presented at 3rd Workshop on Managed Runtime Environments (MRE-3)*, 2005.

[11] K. Karlsson, K. Moore, E. Hagersten, and D. Wood, "Memory characterization of the ECperf benchmark," in *2nd Annual Workshop on Memory Performance Issues (WMPI)*, 2002.

[12] M. Karlsson, E. Hagersten, K. E. Moore, and D. A. Wood, "Exploring processor design options for java-based Middleware," in *Proceedings of 34th ICPP*, 2005.

[13] M. Karlsson, et al., "Memory system behavior of java-based middleware," in *Proceedings of the 9th HPCA*, 2003.

[14] K. Chow, et al., "Characterization of Java Application Server Workloads," in *Preceedings of IEEE WWC-4*, 2002, pp. 175-181.

[15] K. Chow and G. Deisher, "SPECjAppServer2002 Performance Tuning," *WebLogic Developer's Journal*, September 2003.

[16] H. Cain et al., "An Architectural Evaluation of Java TPC-W," in *Proceedings of the HPCA-7*, 2001.

[17] Y. Luo et al., "Workload Characterization of Multithreaded Java Servers," in *Proceedings of the 2001 ISPASS*, 2001.

[18] J. Kim and Y. Hsu, "Memory System Behavior of Java Programs: Methodology and Analysis," in *Proc. of SIGMETRICS'00*, 2000.

[19] R. Radhakrishnan et al., "Architectural Issues in Java Runtime Systems," in *Proceedings of the 6th HPCA*, 2000.

[20] T. Li et al., "Using complete system simulation to characterize SPECjvm98 benchmarks," in *Proceedings of ICS'02*, 2000.