RC24418 (W0711-096) November 14, 2007
Computer Science

IBM Research Report

End-to-End Performance of Commercial Applications in the Face of Changing Hardware

Joefon Jann, R. Sarma Burugula, Niteesh Dubey, Pratap Pattnaik
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

End-to-End Performance of Commercial Applications in the face of changing Hardware

Joefon Jann, R. Sarma Burugula, Niteesh Dubey, Pratap Pattnaik
IBM T. J. Watson Research Center, 1101 Kitchawan Rd.,Yorktown Heights, NY 10598,USA
{joefon,burugula,niteesh,pratap}@us.ibm.com

ABSTRACT
This paper investigates the changes in AIX behavior, or the lack of them, and the resulting performance impact from a generational change in servers in a typical large scale eCommerce application environment without extensive tuning of the OS and the application stack for the changing hardware. We have investigated the performance and impediments to performance at the microprocessor level and at the OS level. This paper dissects the performance data as observed from the OS and from hardware performance counters, and suggests areas for further improvements.

Categories and Subject Descriptors
D.4.0 [Operating Systems]: -Process Management – Multiprocessing, scheduling, Synchronization. -Storage Management – Virtual Memory, Secondary storage. -Performance – Measurements.

General Terms
Measurement, Performance, Experimentation.

Keywords
AIX, POWER5+, POWER6, WebSphere.

1. INTRODUCTION
In the early days of the Information Technology (IT) industry, software and hardware were developed and released in close coordination, such that the software was well tuned to perform well on the hardware platform. However, in the past few years, several factors have led to the existence of multiple generations of hardware running the same software version and vice versa, resulting in suboptimal (from the performance point of view) combinations of software and hardware. One of the reasons is the deviation between release dates of the software and those of the hardware, as depicted in Figure 1. Another important reason is that IT has become a critical and embedded part of business processes in today's organizations. As a result, businesses are reluctant to make changes to their IT environment unless these changes result in a significant improvement in the performance of the business process. In this new environment, IT reliability and non retrogression of performance is much more important than IT performance. The businesses' unwillingness to make changes to their IT infrastructures has led to the separation of the IT stack into multiple layers – hardware, operating system, databases, middleware, applications.

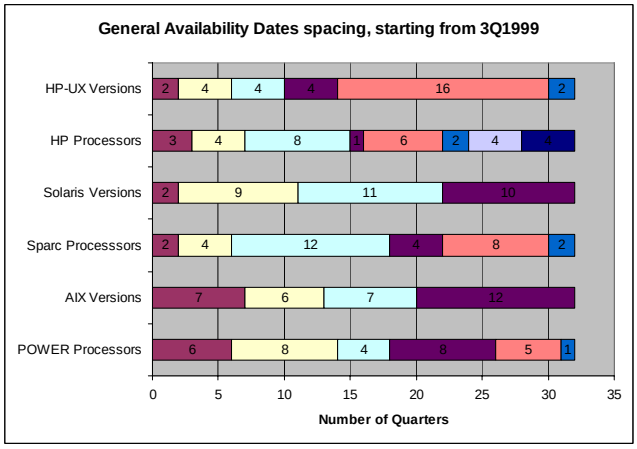

Figure 1: General Availability and lifetime the various releases of different Software and Hardware components.

Another trend in the IT industry is that the applications are increasingly being developed using reflective and dynamic languages and hence are becoming increasingly containerized. This implies that application developers are increasingly detached from the core technological changes taking place in the lower layers such as operating systems and microprocessors. The applications' interactions with the lower layers are left entirely to the middleware which provides container abstractions to the applications. Application developers nowadays focus primarily on object life-cycle management, object instantiation, inheritance and passivation, etc., rather than on the memory layout, the scheduling, the I/O issues, etc. This separation of the application developers from the underlying hardware and systems software technologies has led to significant growth in the application developers' productivity. However, it also implies that the burden of optimization shifts from the application developers to the systems architects of the lower layers.

In light of the above trends, this paper investigates the performance benefits that are obtained from a generational change in servers in a typical large scale electronic commerce application without tuning the application stack. We have investigated the factors of performance and impediments to performance at the microprocessor level and at the OS level. The focus of this paper is not to fine-tune a given application for the best performance on a platform, but to get a measure of how much performance benefits are obtained from a generational change in the hardware, and to identify areas for further improvements. In this paper we used the following setup to do our performance evaluations:

1. A system built with IBM POWER5+ processors [1] (1.9 GHz, 4 processors, each with 2 Simultaneous Multi Threading (SMT) threads) and a system with IBM POWER6 processors [2] (4.7 GHz, 4 processors, each with 2 SMT threads).
2. IBM AIX 5.3 operating system [4]
3. Advanced and production level middleware such as WebSphere v6.1 and DB2 v9, which reflect realistic scenarios, and which themselves contain a number of optimizations of their own.
4. A realistic eCommerce workload, namely Trade6, which simulates a commercial web-based business. [5]

The following section describes this experimental setup in more details.

2. EXPERIMENTAL SETUP

We have primarily used 3 different system configurations for our experiments, as given below.

Table 1: Configurations of the machines used in the Experiments

Name of Experiment	*Processor Type*	*# of Clients*	*Amount of Memory*	*# of Disks*
Exp1P5+_1D	POWER5+ 1.9GHz	75	20GB	1
Exp2P6_1D	POWER6 4.7GHz	75	20GB	1
Exp3P6_4D	POWER6 4.7GHz	75	20GB	4

Each of the experiments uses a 4-processor system with SMT turned on, so that AIX views the system as having 8 logical processors. We have used 64KB page sizes for WebSphere and DB2 in all the experiments. The memory size has been kept the same in all 3 experiments. In Exp3P6_4D, we have quadrupled number of disks used, after we observed a significant amount of CPU idle time due to I/O wait in Exp2P6_1D. More details about this performance gap are given in a later section (3.3.4).

The next few subsections describe in more details the components of the experimental setup.

2.1 WebSphere and Trade6

Web application servers (WAS) provide a framework for executing eCommerce transactions. The interaction between the stateless HTTP browser and the potentially stateful back-end application which executes the business logic needed for the specific commercial tasks is handled by the application server. The execution of the business logic for some of the commercial transactions involves interaction of the application server with major subsystems, such as databases, transaction monitors, messaging subsystems, etc., and some of them may even be executed under separate OS instances. In the current eCommerce deployments, most of this framework is built using the Enterprise Java Bean (EJB) and Java servelet technologies, and the following is simplified explanation of the typical Web application flow, using the J2EE framework, through a Web application sever.

- A user's request (for information such as a stock quote, or for updating an account) is presented to the eCommerce site through a web browser. The computer executing the browser (which we will refer to as client) sends this request as an HTTP request to the Web server. The Web server, using its configuration data, identifies that the request is destined for a Java servlet and passes the request onto the servlet runtime engine code. The appropriate servlet is invoked by the engine. In our experiment, this engine is a part of WebSphere.
- The servlet knows which method it needs to invoke to obtain the information to satisfy the client request. The servlet calls on an Enterprise JavaBean (EJB) to conduct the query. An EJB is a collection of Java objects, encapsulating a particular business process and embodies the appropriate business logic. The operating environment (such as the model used for security, recovery, transactional integrity, and the logistics to access databases to create persistent information) is handled by WebSphere or similar middleware.
- The business logic is written into an EJB, and if needed, a connection is made to the back-end database.
- Upon completion of the business logic, via EJB, the Java servlet engine regains control and manages the generation of the response page which is served back to the client that made the original request. To do that, an appropriate Java Server Page (JSP) is selected to help generate the dynamic content. The computed Web page containing the results of the query is then sent back to the client via the Web server.

IBM WebSphere Application Server (WAS) is a popular, Java technology based, Web application server which provides a rich, e-business application deployment environment with a complete set of application services, including capabilities for transaction management, security, clustering, performance, availability, connectivity and scalability. In keeping with the J2EE component architecture, which WebSphere implements, the servlets and JSPs run in a Web container, and EJBs run in an EJB container.

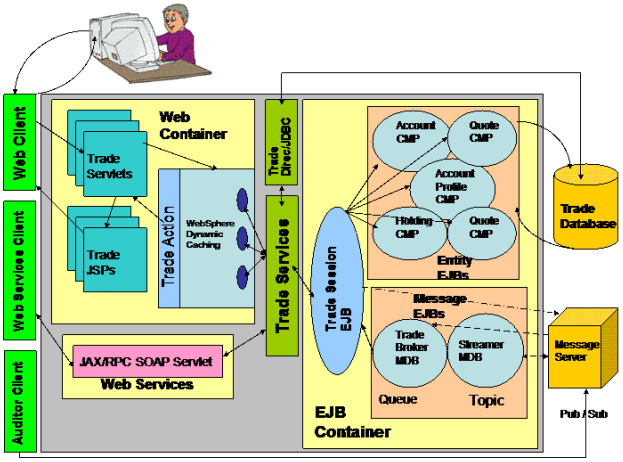

Figure 2: Trade6 Components

The Trade6 Benchmark (Figure 2) is a WebSphere end-to-end performance benchmarking application which allows users to perform typical online brokerage-type transactions, such as selling shares from existing accounts, purchasing shares of stock, browsing current stock price for a ticker symbol, etc. The Trade6 application (Figure 2) is a collection of Java classes, Java Servlets, Java Server Pages and Enterprise Java Beans that service the requests made by registered users. This application is run and managed by the WebSphere Application Server.

WebSphere Application Server uses a back end database to store Trade6 transactions. In our experiments, we used the DB2 V9 as the backend database. DB2 V9 is a Relational Database Management System with several advanced features such as adaptive, self-tuning memory allocation, which helps reduce or eliminate the task of configuring your DB2 server by continuously updating configuration parameters, resizing buffer pools and dynamically determining the total amount of memory to be used by the database.

2.2 POWER6 vs POWER5+ processors

POWER5+ [1] and POWER6 [2] processors are both dual core 64-bit highly efficient superscalar processors supporting simultaneous multi-threading (SMT) with 2 threads. While they share a lot of common architectural features, there are notable differences between the 2 processor types and some of the major differences are described below. Figure 3 provides a schematic diagram of the 2 processor types.

1. ***Higher processor frequency in POWER6***

   The POWER6 processor runs at about 2.5 times the clock frequency of the POWER5+ processor (Table 1). It accomplishes this by a combination of VLSI technologies and micro-architectural changes. POWER5+ was built using 130nm silicon on insulator (SOI) IBM CMOS technology [1], while POWER6 is built using 65nm SOI IBM CMOS technology [2]. Also the POWER6 processor is a 13-FO4 design requiring a 24-stage pipeline, while the POWER5+ is a 23-FO4 design yielding a 16-stage pipeline.

2. ***In-order vs Out-of-order Execution***

   The POWER5+ processor uses out-of-order execution, while POWER6 employs in-order execution. Since POWER6 uses only in-order execution, it does not need register renaming. Even though it only has in-order execution, POWER6 uses the Load Look Ahead (LLA) technique to bring data into L1 and ERAT during the stall cycles. Briefly, LLA works by prefetching the data for the subsequent instructions when there is cache miss due to a load. The computational results of these instructions are discarded after the initial load miss returns, and re-execution of all the instructions starts from the initial load miss with the expectation that the LLA primed the cache and ERAT. See reference [2] for a detailed explanation of LLA. To enhance the effectiveness of LLA, POWER6 also employs a Load Miss Queue (LMQ) of 8 entries to support outstanding L1 cache misses.

3. ***Dedicated L2 cache per core vs Shared L2***

   Each core in the POWER6 chip has its own dedicated 4MB L2 cache, whereas in the POWER5+ chip, both the cores share 1.9MB L2 cache.

4. ***Store Queue (SQ)***

   Since both processor types have write-through L1 cache, they employ a Store Queue to keep the contents of store instructions. When the processor issues a store, the SQ keeps the contents of the store until the data is written into L2 cache. If the processor issues any load instruction while the data is still in SQ, the data is served by the SQ instead of the L1 cache. Although this feature is present in both processor types, our results in section 3.2 show that SQ is more beneficial to POWER6 because of its higher frequency.

5. ***Address Translation Tables***

   POWER6 uses Effective-to-Real-Address-Translation (ERAT) tables only, instead of both the Translation Look-aside Buffers (TLB) and ERATs used in POWER5+. Each POWER5+ core has a 2048-entry TLB table which is not present in POWER6, whereas each POWER6 core has a 128-entry Instruction ERAT (I-ERAT) table and a 128-entry Data ERAT (D-ERAT) table. The I-ERAT supports 4KB and 64KB page sizes whereas the D-ERAT supports 4KB, 64KB, and 16MB page sizes.

The schematic diagrams of POWER5+ and POWER6 processors are given below.

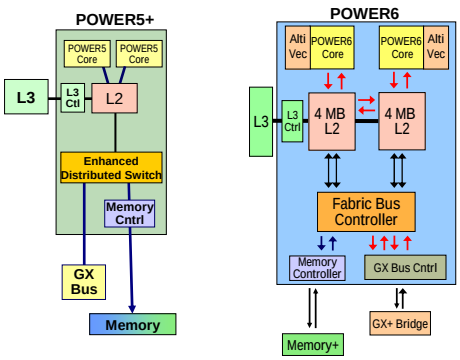

Figure 3: Schematic diagrams of POWER5+ and POWER6 microprocessors

The table below summarizes the feature differences between the 2 processor types used in our experiments.

Table 2: Comparison of POWER5+ and POWER6 processor Features

	POWER5+	POWER6
Frequency	1.9 Ghz	4.7 Ghz
TLB capacity, associativity	2048 entries, 4-way	N.A.
L1 ICache capacity, associativity, line size	64 KB, 2-way, 128B	64 KB, 4-way, 128B
L1 DCache capacity, associativity, line size	32 KB, 4-way, 128B	64 KB, 8-way, 128B
L2 Cache capacity, associativity, line size	1.9 MB (shared), 10-way, 128B	4 MB, 8-way, 128B
L3 Cache capacity, associativity, line size	36 MB, 12-way, 256B	32 MB, 16-way, 128B
Address Translation	Effective-to-Real : 128-entry I-ERAT, 128-entry D-ERAT Virtual-to-Real: 2048-entry TLB	Effective-to-Real : 128-entry I-ERAT, 128-entry D-ERAT

2.3 AIX 5.3 Operating System [4]

AIX 5.3 is a highly-scalable, robust, high-performance, and industry-leading UNIX server operating system with a kernel optimized for the POWER hardware platforms. AIX shares a lot of commonalities with other UNIX operating systems, particularly with respect to commands and application interfaces. However, there are significant differences in the implementation of various kernel subcomponents, most prominently in the memory management subsystem. AIX 5.3 implements many advanced features such as affinity scheduling, thread-sensitive scheduling, multiple concurrent page-able page-sizes, hardware-thread priority management, and hardware-page copier, to efficiently exploit the POWER architecture, In addition, AIX supports many industry-leading Advanced Virtualization features in close collaboration with the POWER hardware and hypervisor. In this paper, our focus is to observe if there are major changes in OS behavior when the underlying processor type changes significantly (from POWER5+ to POWER6), without changing the application software and the OS levels.

3. RESULTS

This section discusses the results of our experiments on POWER5+ and POWER6 systems using Trade6 as the workload.

3.1 Results as observed by the client application (Trade6 driver)

Figure 4 shows the distribution of the application throughput observed by the Trade6 client for all 3 experiments. The graphs provide a high level view of the Trade6 throughputs as a function of elapsed time. In the averages presented in the rest of this paper, we have eliminated the measurements for the initial warm up period, and have computed the average values for a typical "steady-state" period, namely, using the values measured between elapsed time of 300 and 540 seconds in all 3 experiments.

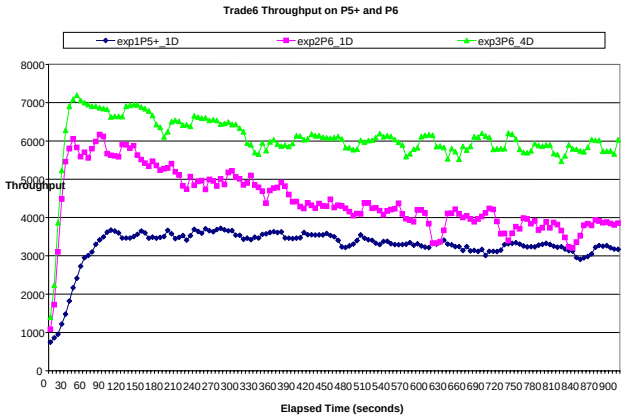

Figure 4: Throughput observed by the Trade6 application as a function of Elapsed time

Table 3 summarizes the graphs in Figure 4 by computing the average throughput after Trade6 has stabilized.

Table 3: Average throughput observed by Trade6

	Exp1P5+_1D	*Exp2P6_1D*	*Exp3P6_4D*
Throughput (= #Webpages served per second)	3466	4458	6001

As we can see from the above results, the throughput increases by about 30% just by moving from the POWER5+ processor type to

the POWER6 processor type. Because, the POWER6 clock frequency is almost 2.5 times that of the POWER5+ processor, , but the throughput had not increased 2.5 fold, we set out to investigate what the reasons could be.

In the following subsections, we analyze the measurements captured at 2 different levels of the IT stack (namely, at the processor micro architectural level, and at the OS level) and show where the performance bottlenecks occurred.

3.2 Results at the Processor Micro-architectural level

Table 4: Processor Micro-architectural Level Metrics (Average values)

	Exp1P5+_1D	*Exp2P6_1D*	*Exp3P6_4D*
Trade6 Throughput (#wPg / sec.)	3466	4458	6001
#Instructions Completed / wPg	1,104,959	1,048,033	1,083,740
CPI (Cycles Per Instruction)	3.38	4.49	4.49
#L1 Data Cache Refs (Loads+Stores) / wPg	630,279	421,667	413,769
#L1 Data Load references / wPg	386,101	249,598	253,901
#L1 D-Cache Load misses / wPg	41,876	15,386	15,589
Ratio of (L1 D-Cache Load misses / L1 D-Cache Load Refs.)	.11	.06	.06

From the above table, we can see that the numbers of instructions executed per web-page by the POWER5+ and POWER6 processors are about the same (only 5% difference between Exp1 and Exp2, and 2% difference between Exp1 and Exp3). However, we can also observe several big differences between the 2 platforms, namely, differences in CPI, and in cache activities.

From Table 4, we see that the CPI for POWER6 is about 33% higher than that for POWER5+. Figure 5 below gives the breakdown of the CPI measurements on the POWER5+ AND POWER6 processors (from Exp3P6_4D experiment).

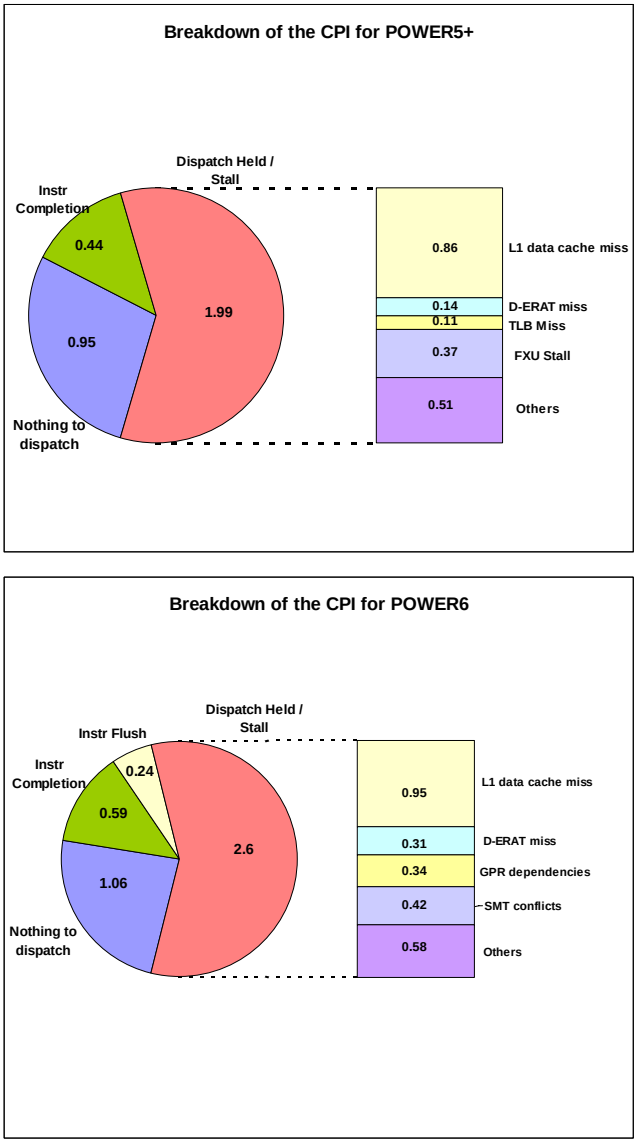

Figure 5: Breakdown of the CPI in POWER5+ and POWER6 (Exp3P6_4D) processors

Figure 5 shows that the dominating factors contributing to the CPI in both processor types are the Data Cache misses, and the Nothing-to-dispatch situations (which are primarily due to I-Cache misses). For POWER6, of the 47% of CPI due to Dispatch Stalls, about 21% are due to L1 Data cache misses. Note that despite the large number of stalls due to data cache misses, the POWER6 processor is still able to get significant efficiency by using techniques such as LLA. Another source of the stalls in POWER6 is GPR dependencies, which contributes to about 8% of the CPI. These are caused by the lack of register renaming in POWER6, as described in section 2.2. The overall distribution of the cause of high CPI for POWER6 is very similar to that of POWER5+, even though the 2 processor types have significantly different micro-architecture. From this we observe that the micro-

architecture of a well designed out-of-order and that of an aggressive in-order processor are equally suitable for today's eCommerce applications, and their significance to performance will depend on the higher levels of the end-to-end stack.

We can also notice from Table 4 that the L1 Data cache load references and misses per web-page are significantly less on POWER6 than those on POWER5+. We believe that the reduction in L1 Data cache load references is due to a higher utilization of the Store Queue (SQ) in POWER6. The combination of (1) a 16-entry SQ per POWER6 core, (2) the nature of eCommerce applications in which loads are closely followed by stores, and (3) the higher clock frequency of the POWER6 core, causes a higher utilization of the SQ than on the POWER5+. The L1 Data Cache references also show that a large portion of the instructions in today's eCommerce applications put significant stress on the memory subsystem in the processor.

This decrease in the L1 D-Cache Load miss ratio from POWER5+ to POWER6 can be attributable to the doubled size and associativity of the L1 Data cache in POWER6.

Based on these results, we believe that recompilation of today's eCommerce applications will not significantly improve their performance on the POWER6 platform. Recompilation of the application in our experiments may reduce the stalls due to GPR conflicts; however, since the portion of Dispatch Stalls due to GPR dependencies is only .34 (out of total value of 4.49 for the CPI), recompilation will cause at most 7% reduction in the CPI value.

The high CPI on both platforms is the reflection of the tendency in the new generation of eCommerce applications to focus in the higher level abstractions and issues such as object life-cycle management, instantiation and passivation rather than on improving the performance of the application for a given micro architectural design. We believe that this trend will continue as the world is increasingly moving towards reflective, dynamic and object-oriented programming environments such as Ruby, PHP, Python, Perl, Java, etc.

3.3 Results at the Operating System Level

While the results at the processor level are significantly different in many parameters, the results at the operating system level between the three experiments are not much different except in one area – I/O subsystem. As we have shown below, the results in process and memory subsystems are quite similar between both the platforms.

3.3.1 System Calls

Table 5: System Calls

	Exp1P5+_1D	Exp2P6_1D	Exp3P6_4D
#syscalls / sec	71,591	95,776	131,157
#syscalls / wPg	20.65	21.48	21.85

As we can see from the above table, there are no significant differences in the number of system calls invoked per web-page among the 3 different experiments. This can be expected because we ran the same application in all 3 experiments.

3.3.2 Process Subsystem

We have captured several different metrics in the process subsystem to see if the OS behaved differently between the 2 hardware platforms.

Table 6: Process or Thread related metrics

	Exp1P5+_1D	Exp2P6_1D	Exp3P6_4D
User	74.48	47.65	62.58
System	13.50	12.05	15.80
Idle	7.83	29.60	17.0
I/O wait	4.13	10.65	4.65
#runnable threads	28.18	13.33	19.63
#threads waiting for disk I/O	1.48	1.78	1.10

Table 6 shows that the processor utilization has gone down quite a bit (from 88% to 60%) when we ran the same application on POWER5+ versus POWER6. One can also observe that the number of runnable threads has decreased from Exp1P5+_1D to Exp2P6_1D. Moreover, the I/O wait time has increased dramatically. Based on these observations from Exp2P6_1D, we concluded that the I/O subsystem's performance had not kept up with the CPU performance when we moved from POWER5+ to POWER6. To validate our inference, we constructed experiment 3 (Exp3P6_4D) by increasing number of disks used for DB2 from 1 to 4. As the results in the above table show, this experiment validated our inference that the performance gap between the processor and the I/O subsystem has widened going from POWER5+ and POWER6. In a later section, we discuss this gap in more details, and also provide a quantitative metric for measuring this gap.

The 5th row in Table 6 shows the average number of runnable threads that are either currently running or waiting to be scheduled on a CPU, and the next row shows the average number of threads waiting for an I/O to be completed. We can see that in Exp2P6_1D more threads are waiting for I/O than in Exp1P5+_1D, which further points to the relative inefficiency of the I/O subsystem in Exp2P6_1D.

Comparing experiments 1 and 3, we see that the processors are staying idle for a larger percentage of time in Exp3P6_4D. This is probably due to the fact that the applications WebSphere, DB2 cannot generate enough load to consume the available resources in experiment 3. When we tuned the WebSphere and DB2 to use more threads and increased the number of clients, the Idle time and I/O wait time have been both reduced by than half, resulting in increase in the throughput.

We have also measured the number of context switches per second in the OS. These numbers, given in Table 7 below, show that the number of context-switches per web-page do not significantly differ across all 3 experiments. The number of context switches is roughly proportional to the throughput obtained.

Table 7: Number of Context Switches

	Exp1P5+_1D	Exp2P6_1D	Exp3P6_4D
#Context Switches per second	40,924	52,611	70,572
#Context switches per web-page	11.80	11.80	11.76

3.3.3 Memory Subsystem

Based on the results in Table 8, we can see that the memory subsystem does not behave very differently on a POWER6 system compared to that on a POWER5+ system.

Table 8: Memory subsystem Metrics

	Exp1P5+_1D	Exp2P6_1D	Exp3P6_4D
#Zero Page-faults / sec	168.08	151.73	153.53
#Zero Page-faults / wPg	0.048	0.034	0.026
#Non-zero Page-faults / sec	140.18	163.48	179.98
#Non-zero Page-faults / wPg	0.040	0.037	0.030
#Page-ins / sec	18.28	38.95	53.53
#Page-ins / wPg	0.005	0.009	0.009
#Page-outs / sec	266.85	300.88	444.23
#Page-outs / wPg	0.077	0.067	0.074

From the above table, we can see that the total number of page-faults per second is not substantially different across all 3 experiments. However, we can also see that the number of page faults generated on a per-web-page basis has decreased from POWER5+ platform to POWER6 platform. One of the reasons for this behavior could be that there is a lot of memory-page sharing between the data accessed by Trade6 for serving multiple requests. As more requests are being served, the memory page-sharing also increases.

The number of page-ins and page-outs in the above table mainly represent the I/O activity caused by the database. The DB2 database in our experiments uses a JFS2 file system to store its data. From the above table, we see that the number of page-outs is significantly higher than the number of page-ins. This is due to the fact that DB2 has to store the results of every Trade6 transaction into persistent storage, i.e. a database page will be written to the disk multiple times during its lifetime.

3.3.4 I/O Subsystem

As described in the 2 subsections above, we can see that most of the performance issues arose from the I/O subsystem of the server.

As described before, we have identified the I/O subsystem as the bottleneck when we noticed that the CPUs were waiting much more for I/Os to be completed in Exp2P6_1D. Once we addressed this problem by striping the database across 4 disks instead of 1, the I/O bottleneck had decreased substantially.

We have also measured the number of device interrupts and the number of IO starts and completions, to characterize the gap between CPU and IO performance. The following table lists these results, and we include our analysis below the table.

Table 9: I/O subsystem Metrics

	Exp1P5+_1D	Exp2P6_1D	Exp3P6_4D
# Device interrupts / sec	5711	5586	7407
# Device Interrupts / wpg	1.65	1.25	1.23
# start IOs / sec	286	340	498
# IOdones / sec	262	289	445
# IOdones/ startIOs	0.92	0.85	0.89

From the above table, we can observe the following:

a) The number of device interrupts per web-page decreases as the processor speed increases.

b) The number of IO-dones per second are less than the number of startIOs per second. This is because the JFS2 filesystem coalesces multiple I/O requests into a single one and invokes the memory subsystem's I/O-completion-notification interface only once. We observe that this feature can be used to construct a metric for measuring the performance gap between the processor and the I/O subsystem. **The metric is the ratio of IOdones/IOstarts.** When the processor subsystem is significantly faster than the I/O subsystem, the ratio of IOdones/IOstarts will be much lower than 1. When both the subsystems are in sync, then this ratio is closer to 1.

4. CONCLUSIONS

In this paper, we have compared 2 generations of the IBM POWER processors running the same application stack and operating system. We have primarily analyzed the OS behavior on these 2 generations of hardware to see if there are significant changes in the OS behavior. Based on our observations, the following conclusions can be made:

1. Modern eCommerce applications are increasingly built out of easy-to-program, generalized but non-optimized software components, resulting in substantive stress on the memory and storage subsystems of the computer. A large portion of the instructions executed for Trade6 application are load/stores, as shown in section 3.2.
2. The increasing integration of IT into the business processes has pressured the various layers of the IT stack to be released/updated at different times. Hence, there is a need for multiple versions of each layer of the IT stack to coexist and to perform reasonably well with different releases of the other layers.
3. Merely cranking up the frequency of the processor to a much higher level does not necessarily provide proportional performance/throughput increases in the end-user applications. Nevertheless, we have observed that one can gain significant improvements in the end-user application throughput with minor tunings at the system hardware configuration level or at the OS level. For example, after observing bottlenecks in the I/O subsystem, spreading the database across 4 disks instead of using 1 disk has provided significant throughput improvements.
4. The higher CPI measured in the POWER6 experiments persisted despite our tuning attempts at the OS level and at the I/O hardware level, and despite the increased on-core cache sizes over POWER5+. And, as our analysis in section 3.2 shows, recompiling for POWER6 would not have made much difference to the CPI.
5. From Table 6, we see that the AIX kernel consumed a larger percent of the CPU time in experiment 3 than in experiment 2 (both on POWER6). The cause of the increase is probably the I/O configuration changes (spreading the database across 4 disks instead of 1) we made to reduce the CPU idle and I/O wait time. Because I/O activities at the OS level are mostly kernel mode activities, more I/O activities in the system resulted in more CPU cycles being executed in kernel mode. However, the 4 disk configuration in experiment 3 yielded a significant improvement in throughput, leading us to conclude that careful balance of resources by the OS, particularly in the I/O space optimization, has the potential to yield significant performance for future eCommerce applications.
6. It is very important to keep the performance of the memory subsystem and the I/O subsystem in sync with the processor performance in order to optimize eCommerce throughput and to maintain similar OS behavior. We believe that the ratio **IOdones/IOstarts** can be used as an indicator of how well balanced the CPU subsystem and I/O subsystems are on the AIX/SystemP platform.
7. Within the limited scope of our experiments, we have shown how to systematically analyze the data to identify changes in the OS behavior, and have attempted to explain those changes in light of the differences in the 2 microprocessor implementations.
8. In summary, the overall OS behavior did not seem to change significantly across these 2 generations of hardware processor type.

5. REFERENCES

[1] Sinharoy. B., Kalla. R.N., Tendler. J.M., Eickemeyer. R.J., and Joyner. J.B. 2005: POWER5 system Microarchitecture. IBM Journal of Research and Development, Vol 49, No 4/5, 2005

[2] Le H.Q., Starke W.J., Fields J.S., O'Connel F.P., Nguyen D.Q., Ronchetti B.J., Sauer W.M., Schwarz E.M., Vaden M.T. 2007: IBM POWER6 Microarchitecture. IBM Journal of Research and Development, Vol.51, No. 6, Nov. 2007

[3] Mackerras. P., Mathews. T.S., Swanberg. R.C 2005: Operating System exploitation of the POWER5 system. IBM Journal of Research and Development Vol 49, No. 4/5, July/September 2005.

[4] AIX 5.3 Information Center : http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.doc/doc/base/aixinformation.htm

[5] IBM WebSphere Performance and Trade6 benchmark http://www.ibm.com/software/webservers/appserv/was/performance.html