Impact of memory technology trends on performance of Web systems

Mauro Andreolini, Michele Colajanni, Riccardo Lancellotti
Department of Information Engineering
University of Modena and Reggio Emilia, Italy
{andreolini.mauro, colajanni.michele, lancellotti.riccardo}@unimo.it

Abstract

The hardware technology continues to improve at a considerable rate. Besides the Moore law increments of the CPU speed, it should be considered that the capacity of the main memory in the last years is increasing at an even more impressive rate. One of the consequences of a continuous increment of memory resource is that we can design and implement memory-embedded Web sites, where both the static resources and the database information is kept in main memory. In this paper, we evaluate the impact of memory trends on the performance of e-commerce sites that continue to be an important reference for Internet-based services in terms of complexity of the hardware/software technology and in terms of performance, availability and scalability requirements. However, most results are valid even for other Web-based services. We demonstrate through experiments on a real system how the system bottlenecks change depending on the amount of memory that is (or will be) available for the Web site data. This analysis allows us to anticipate the interventions on the hardware/software components that could improve the capacity of present and future Web systems for content generation and delivery.

keywords: Web systems, performance evaluation, in-memory databases, bottleneck analysis

1. Introduction

Web technology is the standard interface towards many services exploited through the Internet. This popularity has become the target of new technological trends. The capacity of current hardware components such as processor and main memory is continuously growing. The main interest has always focused on the impressive growth of the CPU power that continues to follow the Moore law [8] although the asymptotic bound seems closer. This interest has, in a certain measure, hidden the fact that in the last years the capacity improvements of the main memory are even more impressive than the CPU increases. The combination of these different hardware resource trends is having or will have important implications on the performance of the Internet-based services. In the traditional client-network-CPU-disk path, the main memory more than the CPU seems to best answer to the continuous bottleneck removals that are occurring in the Net. This tendency is favorable for the user that may experiment lower latencies, but it has the potential to create difficulties to the server component of an interactive Web-based service [10]. If we limit our considerations to Web sites that are of interest for this paper, we can observe that the size of a typical site in terms of stored information tends to grow slower than the typical size of the main memory. Nowadays, some Gbytes of RAM are the entry level of any server machine that supports interactive services with some performance requirements, and even a small site is implemented through a multi-tier logical architecture that is mapped at least on two/three server nodes. Such a platform allows us to design and implement memory-embedded Web sites, where both the static resources and the database information is kept in main memory.

Some consequences of these technological trends are intuitive. For example, we can easily expect that the system capacity will tend to increase and any Web site will be able to guarantee higher throughputs. Other consequences are not fully exploited. For example, it is unclear which will be the new bottlenecks that will bound the maximum capacity of a Web system. In this paper, we aim to evaluate the impact of improvement trends on the performance of e-commerce sites that continue to be an important example of mission critical Internet-based service both in terms of complexity of the hardware/software technology and in terms of performance, availability and scalability requirements. Through endurance and stress tests, and accurate measurement of coarse grain and fine grain performance parameters, we show how system bottlenecks of a prototype e-commerce site change as a function of the amount of available main memory. This trend analysis is important because it allows us to anticipate the interventions on the system that could improve the capacity of present and future Web-based services. The main contributions of this paper are summarized below.

First, we confirm the intuition that memory-embedded Web sites have lower response times and better throughput than sites that keep a significant portion of static and dynamic information on disk, and serve it through the file system and the database server, respectively.

Second, for all considered workload models, the back-end node hosting the database server remains the system bottleneck. However, a fine grain analysis of the system resources of the back-end node allows us to evaluate how the bottlenecks change as a function of available main memory. We distinguish three results. For an almost memory-embedded database, disk accesses are rare, and the bottleneck is represented by the CPU of the database server because of synchronization operations with in-memory data buffers. When a significant portion of the database (e.g., 50%) is kept in memory, the system capacity is typically limited by the number of available socket descriptors, that are a pool of limited resources. When a small part of the database is kept in RAM, the disk operations represent the bottleneck reducing the system capacity.

Multiple papers compare different technologies for dynamic Web sites. For example, in [7, 4] the authors evaluate the performance of J2EE and PHP implementations of the same e-commerce system. However, the performance comparison in these studies is limited to the scalability of each technology and does not focus on the impact of technological trends (such as the growing amount of available memory) on the system bottlenecks. Furthermore, all these studies share the common trait of focusing on a coarse grain performance analysis of the systems at most at the node level.

Other studies illustrate different aspects of Web systems testing. For example, in [11, 5] a fine grained analysis of the performance on HTTP servers is provided. However, these studies focus on Web sites serving mainly static Web resources and do not take into account the complexity and the interactions of a more complex Web site providing highly dynamic and personalized contents.

To the best of our knowledge, our paper is one of the first studies addressing the issue of the impact of technological evolution on Web systems performance. Moreover, the performance evaluation follows a novel approach which integrates both the coarse and the fine grain analysis to provide a deep insight on the system bottlenecks.

The rest of this paper is organized as it follows. Section 2 outlines the main functions of the components of a representative e-commerce site of medium size and average popularity. Section 3 describes the testing plan that will be followed during the experiments. Section 4 describes the experimental testbed and the workload models that we used for the experiments. Section 5 presents the experimental results on the performance impact of memory technological trends. Finally, Section 6 concludes the paper with some final remark.

2. System architectures for Web sites

The trend for building Web sites that provide the users with static and dynamic resources is clear. Independently of the large variety of available software, the typical design is based on a multi-tier logical architecture that tends to separate the three main functions composing a Web site: HTTP interface, application logic and information source. These architecture layers are often referred to as *front-end* tier, *middle* tier and *back-end* tier [2].

In this paper, we focus our attention on the main memory expansion. The data storage system has been influenced by the technological trend of memory size growth. The cost of memory has been constantly decreasing over years and even entry level computers are now equipped with amounts of RAM that were only found in top-level workstations just a few years ago. If we apply this trend to the databases used in Web systems, it is clear the radical shift that this is introducing. On one hand most Web sites are characterized by a limited amount of information seldom exceeding few Gbytes and their growth is far from the exponential trends that characterize technological development. On the other hand, over the years the amount of memory installed on database servers has been steadily increasing. As a consequence many Web sites are characterized by a DBMS that can store the whole database into RAM [9] and this fact is destined to become more and more common in the near future. Besides the obvious performance gain due to the low access time of RAM if compared to disk, a RAM-based database can dramatically alter the performance of the Web system by shifting potential bottlenecks on multiple parts of the Web system.

The service of a Web request is a complex task that triggers the creation of several processes. Requests for static Web objects can be satisfied by the front-end tier and do not usually put any significant load on the system. On the other hand, dynamic requests are usually passed by the front-end to the middle layer which produces the content with the cooperation with the back-end tier. The middle layer operations may be computationally intensive tasks, but in any case the CPU is the most stressed resource. Moreover, the dynamic generation of content needs one or more interactions with the DBMS on the back-end tier. When the DBMS is placed on a different machine, a communication between the application server and the DBMS requires the use of connection descriptors (sockets). These are critical operating system resources, because they are limited in nature. On its turn, the DBMS can place a significant amount of stress on different hardware resources of the server machine depending on the type of required operations. For example, the CPU may be loaded by operations related to a complex query, the disk may be loaded by operations that require many accesses to the mass storage. It is worth noting the different behaviors of the system resources. In particular, sockets are *token-based* resources, which means that only a finite number of sockets is available and can be assigned to processes. When the available tokens are exhausted, additional requests are queued for an unpredictable time (i.e., until a token becomes free). A connection request may fail because

of the time-out deadline is passed or because it cannot be stored in the waiting queue characterized by a finite length. On the other hand, CPU and disk are gracefully degradable resources, because they can be shared among every process requesting access to them. Once the resource is fully utilized, additional requests lead to progressive performance degradation, but typically no waiting request is refused.

3. Performance testing plan

In this paper we aim to evaluate the effects of technological trends on overall system performance, but also to verify whether these trends modify the bottlenecks that determine the maximum system capacity. Depending on the goal of the performance study, we use a *black-box* testing to evaluate the performance of the system and to determine under which workload conditions the system reaches its maximum capacity. Moreover, we use a *white-box* testing to identify the hardware or software resource that represents the system bottleneck for each considered case. In black box testing, the system is viewed as an atomic entity; we consider its behavior as it is seen from the outside. The considered performance indexes are at the *system* level. Popular indexes are the response time, and the throughput that can be measured in terms of served requests (e.g., pages, hits) per second or Mbytes per second. The main goal of these performance indexes is to verify whether the system is providing or not services at an acceptable level of performance. Black-box testing is also useful for detecting trends of system performance as a function of the offered load. It is a common practice to build a load curve for the Web system by considering, for example, the average (or, much better, the 90-percentile) of the response time as a function of increasing request loads. Such a curve allows us to identify a knee region in which a sudden growth of the response time occurs. This region denotes that the system is working at its maximum capacity and at least one of its resources is critically loaded. Black-box testing does not consider the internal components of the system, hence it is impossible to identify the bottleneck that limits the system capacity.

To delve a deeper analysis, a more thorough white-box testing has to be carried out. The Web system is exercised with a client request load around the knee region identified through the black-box testing, and the status of internal resources of the Web system is monitored. To perform this analysis, it is necessary to consider performance indexes at least at the *resource level* that are typically associated to hardware and software (mostly, operating system). Examples include: the utilization of the CPU, disk and network interface, or the amount of free memory at the hardware level; the number of available file and socket descriptors at the level of the operating system.

In our study we take into account cumulative distributions or percentiles instead of average values. Indeed using higher moments is useful in the case of evaluating performance of systems characterized by heavy-tailed distributions such as e-commerce systems [2]. This becomes even more important when we consider that e-commerce systems may be interested to provide services based on some *Service Level Agreement* (SLA).

4. Setup of the experiments

We carried out a set of experiments on a sample Web site providing an e-commerce service deployed on a local area network.

Figure 1 shows the architecture of the prototype system. Our testbed is composed by three nodes. The first node hosts the client emulator, which is used to generate the appropriate volume of user requests to the Web system. The second node hosts the front-end and the middle tiers. In particular we use the Apache Web server for the front-end, while the PHP4 engine engine is used to implement the middle tier providing the application logic. The back-end tier is located on a third node. We choose MySQL as the database server. To reflect a realistic workload scenario, we enabled the support for table locking and two phase commits.

Each node is equipped with a 2.4 GHz hyperthreaded Xeon and is part of cluster of nodes connected through a Fast Ethernet LAN. The basic software of nodes of the Web system includes, besides the standard Linux operating system (kernel version 2.6.8), a set of monitoring tools at node level aiming to collect performance index samples required by the white box testing. In particular we choose the *system activity report* [3] tool to collect resource utilization statistics. This tool samples at regular intervals the utilization of both physical resources such as the CPU, the memory and the disk, and operating system resources such as the number of open sockets and the number of processes. The output of the system monitor is logged for later off-line analysis.

Since we are interested in evaluating the impact on performance of the memory growth trend shifting towards large amounts of RAM in which a whole database can fit, we define the following *memory scenarios*:

All in-memory. This scenario represents the current and future trend of placing the whole database into memory.

Partially in-memory. This scenario represents a situation where the database is too large to fit into main memory or where the amount of RAM on the back-end node has been under-provisioned.

Mostly on-disk. This scenario is an extreme case (not much widespread in today's world) where the memory is so scarce that only a small portion of the database can be cached into main memory.

The database used in our experiments has a size of 450 MB. We implement the three memory scenarios by limiting the amount of available physical RAM at the database

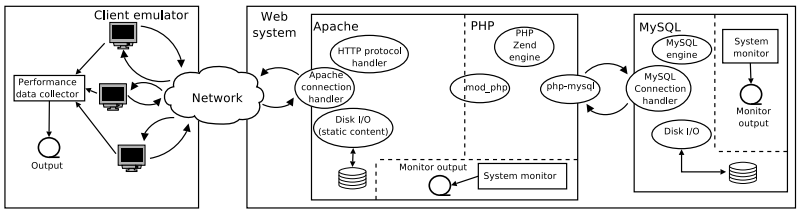


Figure 1. Experimental testbed

Scenario	Available RAM	
	MB	% of DBMS size
All in-memory	1GB	100%
Partially in-memory	256 MB	60%
Mostly on-disk	128 MB	30%

Table 1. Memory scenarios.

Scenario	Static requests	Dynamic requests
Browsing	60%	40%
Buying	5%	95%

Table 2. Composition of the workload scenarios.

node. This is obtained by starting the Linux kernel with the *mem* boot command line option that places an upper bound to the maximum amount of memory handled by the operating system. Table 1 provides the amount of RAM in every scenario as well as the percentage of database that can be stored into main memory.

The choice of the workload model to test an e-commerce system is a problem by itself. Unlike the workload models oriented to browsing where the interaction is basically with the Web server and the mix is mainly oriented to define the number and size of embedded objects together with the user think time, it is impossible to define *THE* model for an e-commerce service because of the dozen of possible alternatives at any level of the multi-tier architecture, often also dependent on the adopted software technology. The research community is being oriented to use the TPC-W benchmarking model that is the only complete specification of an e-commerce site (on-line book store) [1]. In this paper we use a TPC-W like model. We implement two different scenarios, *browsing* and *buying*, which capture client activities towards the e-commerce system. The percentage of requests for dynamic and static Web resources for both scenarios is shown in Table 2. Space reasons lead us to present the experimental results only for the buying scenario.

Web traffic is generated by means of a TPC-W like *client emulator*, which is executed on a separate node. The client emulator creates a fixed number of client processes which instantiate sessions made up of multiple requests to the e-commerce system. For each customer session, the client emulator opens a persistent HTTP connection to the Web server which lasts until the end of the session. Session length has a mean value of 15 minutes. Before initiating the next request, each emulated client waits for a specified think time, which is on average of 7 seconds. The sequence of requests is determined through a state transition matrix that specifies the probability to pass from one Web page to another one.

5. Performance impact of memory size

5.1. Bottleneck identification

We now evaluate how the technological trend towards in-memory databases can alter the performance of a Web system and shift the bottlenecks across different components and resources. Figure 2 reports the results of the black-box testing. This figure shows the 90-percentile of the response time as a function of the client population for the three memory scenarios. A comparison of the three curves shows that the amount of available memory on the back-end node has a deep impact on both the system performance and on the ability to serve a larger population of users.

For each scenario we identify a critical value of the client population that leads to a sudden degradation of the system performance (the so-called knee of the curve). As we proceed from the mostly on-disk (called simply *on-disk* in the following of this section) to the *partially in-memory* to the all in-memory scenario (*in-memory*) the maximum number of users that can be served without performance degradation increases from 30 to 270 to 330 users, respectively. This confirms the intuition that additional main memory on the back-end node implies better performance and higher system capacity.

Having found the critical load that determines a system bottleneck for the three memory scenarios, we now investigate the nature of such bottlenecks. To carry out this more detailed performance analysis we rely on a white-box testing. The first step is to identify the node of the Web system that causes the bottleneck. The standard procedure is to split the contributions of the response time according to the two nodes composing the system: the node hosting the front-end and middle tiers and the node hosting the back-end tier. For every memory scenario we observe a sudden increase

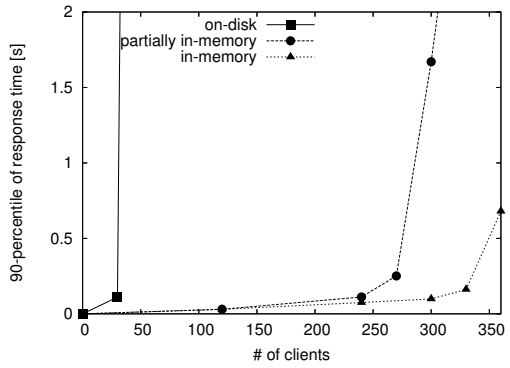


Figure 2. Response time of the Web system for different memory scenarios

of the response time in the back-end tier when the critical client request volume is reached. This increase drives the performance degradation of the system shown in Figure 2 and allows us to conclude that the bottleneck is always related to the database operations. This result is consistent with the observation that performance is affected by the memory scenario, which only interests the node hosting the database.

5.2. Resource-level analysis

Once the knee of the curves has been found, we pass to analyze the causes behind the bottlenecks of the system. We carry out a performance testing for the three main memory scenarios for a population right above the identified knee. For the on-disk scenario we take into account a population of 60 users, for the partially in-memory scenario we focus on a population of 300 users and for the in-memory scenario we consider 360 users. For each scenario we monitor the utilization of system resources. The system monitor provides information on both hardware resources (such as memory, CPU, network and disk utilization) and operating system resources (such as the number of open sockets and the number of processes). From Section 2 we recall that only a subset of these resources is meaningful for the identification of the bottlenecks at the database node. In particular, we will concentrate on the following performance indexes: CPU utilization, disk I/O activity (in terms of disk transactions/second) and the number of open sockets. Our analysis shows that these three indexes allow to identify the bottleneck under every considered memory scenario. Furthermore, the presented results point out that bottlenecks change depending on the amount of available memory.

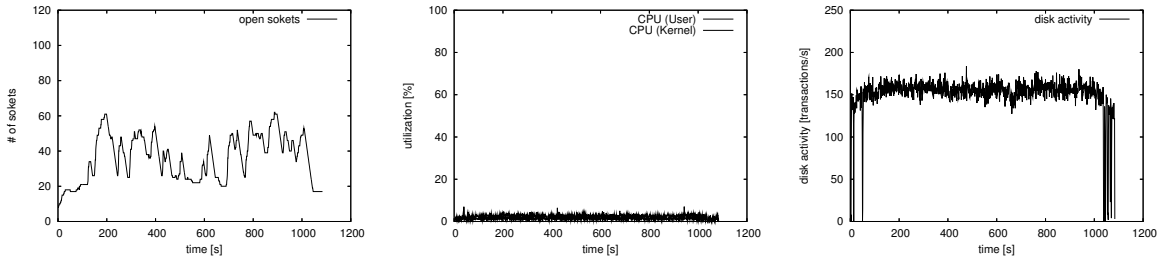
The most significant results of the white-box analysis for the on-disk scenario are shown in Figure 3. In particular, Figure 3(a) shows the number of open sockets throughout the experiment (the time elapsed since the experiment beginning is shown on the x-axis), while Figure 3(b) shows the CPU utilization over the entire experiment. The CPU utilization is split into kernel and user mode. The kernel mode refers to the CPU operations related to the operating system (such as process scheduling, context switches and system call service), while the user mode relates to the running applications. Since we are monitoring the back-end node, the CPU utilization in user mode is due almost exclusively to DBMS operations. Finally, Figure 3(c) shows the disk I/O activity reported as disk transactions per second throughout the experiment.

The first observation deriving from Figure 3(b) is a very low CPU utilization (below 0.1). This allows us to exclude the CPU from the list of possible bottleneck resources. Next, we observe a number of simultaneously open sockets that is significantly lower than in other experiments (see Figure 4(a)), but that is highly variable (typically, ranging from 20 to 60). This leads us to conclude that the number of available sockets does not represent a system bottleneck for this scenario (60 is far below the number of 105 available sockets for the DBMS), however the high variability of the number of open sockets deserve some motivations. Since the DBMS is equipped with a limited amount of memory, most DBMS operations are delayed by the disk accesses. The long DBMS service time increases the number of concurrent client requests, thus leading to a high number of open sockets even for low numbers of user requests.

Finally, we pass to analyze the disk performance. We notice an almost constant throughput in terms of operations. We reproduced the mix of disk read/write/seek typical of database operation using *iozone* [6] to evaluate the maximum disk throughput and found that the value of 160 disk operations per second represents the maximum achievable performance. This proves that, as expected, the disk represents the system bottleneck for the on-disk scenario.

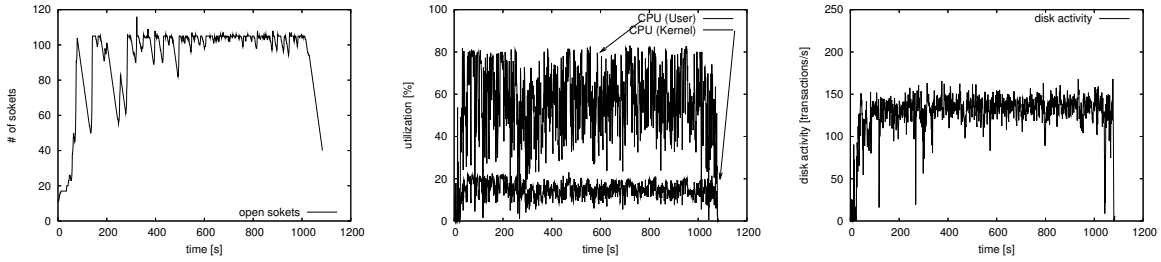
Figures 4(a), (b) and (c) show the socket, CPU and disk utilization for the partially in-memory scenario, respectively. We observe a significant increment of the CPU utilization and of the number of contemporary open sockets, while the disk activities tend to decrease with respect to the on-disk scenario. It is worth to observe that these utilization levels are reached for a system capacity (in terms of served requests) that is more than eight times that related to the on-disk scenario (see Figure 2). This motivates the sudden increment of the CPU utilization and the low decrement of the disk utilization even although more than half of the database is kept in main memory. Nevertheless, neither the CPU nor the disk of the database server are system bottlenecks. Indeed, from Figure 4(a) we can observe that the number of open sockets on the DBMS node is always equal to the maximum capacity (i.e., 105).

Finally, Figure 5 shows the resource utilization for the in-memory scenario. Identifying the system bottleneck for this case is straightforward: the disk is almost not used (Fig-



(a) Open sockets **(b) CPU utilization** **(c) Disk activity**

Figure 3. Resource utilization on back-end node (*on-disk* scenario)



(a) Open sockets **(b) CPU utilization** **(c) Disk activity**

Figure 4. Resource utilization on back-end node (*partially in-memory* scenario)

ure 5(c)); the number of open sockets is far below the maximum limit set to 105 (Figure 5(a)). On the other hand, Figure 5(b) shows a CPU utilization close to 1 with a 0.8/0.2 ratio between the time spent in user and kernel space. The immediate conclusion is that for this scenario the system bottleneck is represented by the CPU capacity.

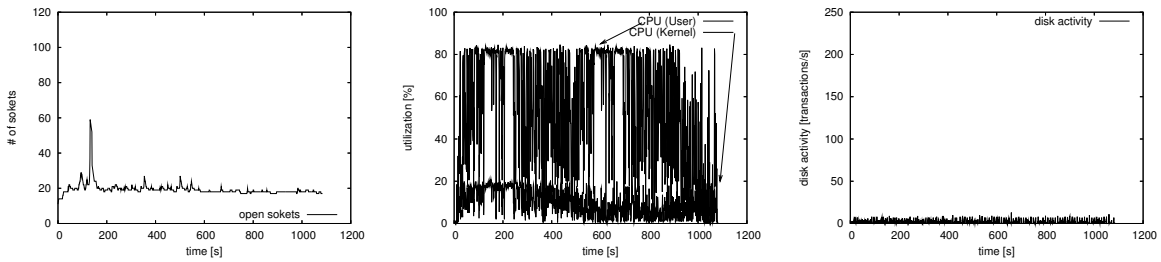
This initially unexpected result suggests that the application level computations are much more intensive than the cost necessary for the system calls. As there is only one major process running on the back-end, we can easily assume that the DBMS process is the real source of the system bottleneck. However, if we limit the analysis at this granularity level, we cannot exactly motivate the high CPU utilization of the database server application. To identify the hot spots in the database server process we should pass to get measurement at the function level. From the system profiler we get that the function that checksums asynchronous I/O buffers uses almost 70% of the CPU time. This function is part of the asynchronous I/O buffer management of MySQL DBMS. Asynchronous I/O is used to improve I/O performance by caching frequently accessed portions of the database, thus bypassing the operating system disk buffer cache. To provide data consistency a checksum is calculated on every buffer. Hence, we can conclude that the asynchronous I/O subsystem is the real bottleneck of the mysqld process.

It is tricky to solve this CPU bottleneck at the database server. The most straightforward option seems to purchase a faster CPU that can provide higher computational power. However, this solution has a limited scalability. The best alternative to improve the database performance is to reduce the checksumming activity by decreasing the number of buffer accesses. For example, this can be easily carried out by augmenting the size of the query cache.

We can conclude that the amount of memory on the back-end node of a multi-tiered Web system is having and will have a fundamental impact on system performance. Although the reduction of the response time and the augment of the system capacity were expected, it is quite interesting to analyze how the memory availability alters in a fundamental way the bottleneck that limits the system performance.

In particular, the possibility of storing half of the Web site information in main memory has a super-linear benefit on the system capacity that in our case improves of more than eight times. Storing even the remaining part of the Web site information in memory augments the system capacity of just an additional 30%. With different system architectures, the effects would be different, even if our experience leads us to conclude that the first half memory storage produces the main increment of the system capacity. Hence, the first conclusion is that it seems very tough to anticipate the effects through some mathematical model.

Moreover, the system capacity is just one face of the medal. Other interesting conclusions can be derived from our experiments. Although the in-memory scenario does not



(a) Open sockets **(b) CPU utilization** **(c) Disk activity**

Figure 5. Resource utilization on back-end node (*in-memory* scenario)

improve much the system capacity with respect to the partially in-memory scenario, the system for this latter scenario is in the worst conditions that is, the disk and the CPU are really highly utilized, and the set of available sockets is exhausted. A system where all resources are critically loaded is something that any manager would like to avoid. In some sense, this situation is even worse than that of the on-disk scenario and of the in-memory scenario where the disk and the CPU are the evident system bottleneck, respectively, but the other resources are not critically loaded.

6. Conclusions

In this paper we evaluated the impact of current technological trends in the main memory on the performance of an e-commerce Web site and on the bottlenecks limiting the service capability of such system.

We confirm the intuition that increasing the amount of memory at the database node improves performance. Furthermore, by means of fine-grained performance analysis, we find that when most of the database can be stored into memory the performance is mainly limited by the CPU power of the database node. On the other hand, when only part of the database can fit into memory, the concurrency in requests is augmented by the longer service time. In this case, software resources such as the number of available socket descriptors for database connections, are the most likely source of performance degradation. Finally, when most of the database is kept on disk, the disk itself becomes the bottleneck resource.

As there is a strict relationship between the bottleneck limiting the performance and the interventions to be undertaken for performance improvement, this study shows that understanding the current technological trend and their implications on Web systems can play and most likely will play in the future a relevant role not only in determining the system performance but also in defining the available intervention options in the case of system consolidation or capacity planning.

References

[1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *Proc. of the IEEE 5th Annual Workshop on Workload Characterization (WWC-5)*, Nov 2002.

[2] M. Andreolini, M. Colajanni, R. Lancellotti, and F. Mazzoni. Fine grain performance evaluation of e-commerce sites. *ACM Performance Evaluation Review*, 32(3), Dec. 2004. Special Issue on E-Commerce.

[3] S. Godard. Syssyat: System performance tools for linux os, 2004. – http://perso.wanadoo.fr/sebastien.godard/.

[4] X. He and Q. Yang. Performance evaluation of distributed web server architectures under e-commerce workloads. In *Proc. of the 1st Int'l Conference on Internet Computing (IC'2000)*, Jun 2000.

[5] Y. Hu, A. Nanda, and Q. Yang. Measurement, analysis and performance improvement of the apache web server. *International Journal of Computers and Their Applications*, 8(4), Dec. 2001.

[6] IOzone filesystem benchmark, 2005. – http://www.iozone.org/.

[7] K. S. Juse, S. Kounev, and A. Buchmann. Petstore-ws: Measuring the performance implications of web services. In *Proc. of the 29th Int'l Conference of the Computer Measurement Group (CMG) on Resource Management and Performance Evaluation of Enterprise Computing Systems - CMG2003*, Dec. 2003.

[8] G. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[9] D. Morse. In memory database Web server. *Dedicated systems magazine*, 2000.

[10] E. M. Nahum, M.-C. Rosu, S. Seshan, and J. Almeida. The effects of wide-area conditions on www server performance. In *Proc. of the 2001 ACM SIGMETRICS int'l conference on Measurement and modeling of computer systems*, pages 257–267, 2001.

[11] H. Xie, L. Bhuyan, and Y.-K. Chang. Benchmarking web server architectures: A simulation study on micro performance. In *Fifth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-02), with HPCA-8*, Feb 2002.