Part 1

The Basics

I went behind the scenes to look at the mechanism.
—Charles Babbage, 1791–1871, the father of computing

The factors that can critically impact the performance and scalability of a software system are abundant. The three factors that have the most impact on the performance and scalability of a software system are the raw capabilities of the underlying hardware platform, the maturity of the underlying software platform (mainly the operating system, various device interface drivers, the supporting virtual machine stack, the run-time environment, etc.), and its own design and implementation. If the software system is an application system built on some middleware systems such as various database servers, application servers, Web servers, and any other types of third-party components, then the performance and scalability of such middleware systems can directly affect the performance and scalability of the application system.

Understanding the performance and scalability of a software system *qualitatively* should begin with a solid understanding of all the performance bits built into the modern computer systems as well as all the performance and scalability implications associated with the various modern software platforms and architectures. Understanding the performance and scalability of a software system *quantitatively* calls for a test framework that can be depended upon to provide reliable information about the true performance and scalability of the software system in question. These ideas motivated me to select the following three chapters for this part:

- Chapter 1—Hardware Platform
- Chapter 2—Software Platform
- Chapter 3—Testing Software Performance and Scalability

Software Performance and Scalability. By Henry H. Liu

The material presented in these three chapters is by no means the cliché you have heard again and again. I have filled in each chapter with real-world case studies so that you can actually feel the performance and scalability pitches associated with each case quantitatively.

1

Hardware Platform

What mathematical problems should a computing machine solve?
—Konrad Zuse, 1934

To build new specifications from given specifications by a prescription.
—His answer in 1936

Computing is the deviation of result specifications to any specifications by a prescription.
—His extended definition in 1946

What performance a software system exhibits often solely depends on the raw speed of the underlying hardware platform, which is largely determined by the central processing unit (CPU) horsepower of a computer. What scalability a software system exhibits depends on the scalability of the architecture of the underlying hardware platform as well. I have had many experiences with customers who reported that slow performance of the software system was simply caused by the use of undersized hardware. It's fair to say that hardware platform is the number one most critical factor in determining the performance and scalability of a software system. We'll see in this chapter the two supporting case studies associated with the Intel® hyperthreading technology and new Intel multicore processor architecture.

As is well known, the astonishing advances of computers can be characterized *quantitatively* by Moore's law. Intel co-founder Gordon E. Moore stated in his 1965 seminal paper that the density of transistors on a computer chip is increasing exponentially, doubling approximately every two years. The trend has continued for more than half a century and is not expected to stop for another decade at least.

The quantitative approach pioneered by Moore has been very effective in quantifying the advances of computers. It has been extended into other areas of computer and software engineering as well, to help refine the methodologies of developing better software and computer architectures [Bernstein and Yuhas, 2005; Laird and

Software Performance and Scalability. By Henry H. Liu

Brennan, 2006; Gabarro, 2006; Hennessy and Patterson, 2007]. This book is an attempt to introduce *quantitativeness* into dealing with the challenges of software performance and scalability facing the software industry today.

To see how modern computers have become so powerful, let's begin with the Turing machine.

1.1 TURING MACHINE

Although Charles Babbage (1791–1871) is known as the father of computing, the most original idea of a computing machine was described by Alan Turing more than seven decades ago in 1936. Turing was a mathematician and is often considered the father of modern computer science.

As shown in Figure 1.1, a Turing machine consists of the following four basic elements:

- A *tape*, which is divided into cells, one next to the other. Each cell contains a symbol from some finite alphabet. This tape is assumed to be infinitely long on both ends. It can be read or written.
- A *head* that can read and write symbols on the tape.
- A *table* of instructions that tell the machine what to do next, based on the current state of the machine and the symbols it is reading on the tape.
- A *state register* that stores the states of the machine.

A Turing machine has two assumptions: one is the unlimited storage space and the other is completing a task regardless of the amount of time it takes. As a theoretical model, it exhibits the great power of abstraction to the highest degree. To some extent, modern computers are as close to Turing machines as modern men are close to cavemen. It's so amazing that today's computers still operate on the same principles

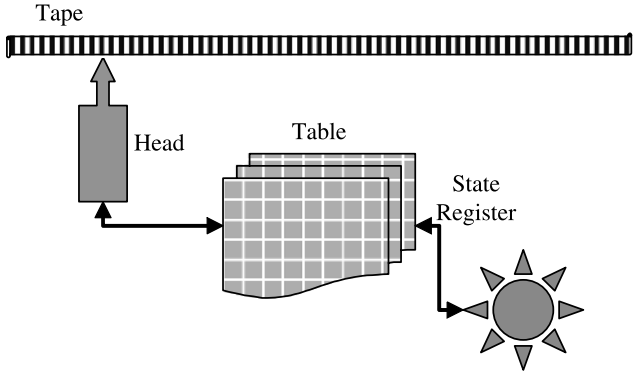


Figure 1.1 *Concept of a Turing machine.*

as Turing proposed seven decades ago. To convince you that this is true, here is a comparison between a Turing machine's basic elements and a modern computer's constituent parts:

- Tape—memory and disks
- Head—I/O controllers (memory bus, disk controllers, and network port)
- Table + state register—CPUs

In the next section, I'll briefly introduce the next milestone in computing history, the von Neumann architecture.

1.2 VON NEUMANN MACHINE

John von Neumann was another mathematician who pioneered in making computers a reality in computing history. He proposed and participated in building a machine named EDVAC (Electronic Discrete Variable Automatic Computer) in 1946. His model is very close to the computers we use today. As shown in Figure 1.2, the von Neumann model consists of four parts: memory, control unit, arithmetic logic unit, and input/output.

Similar to the modern computer architecture, in the von Neumann architecture, memory is where instructions and data are stored, the control unit interprets instructions while coordinating other units, the arithmetic logic unit performs arithmetic and logical operations, and the input/output provides the interface with users.

A most prominent feature of the von Neumann architecture is the concept of stored program. Prior to the von Neumann architecture, all computers were built with fixed programs, much like today's desktop calculators that cannot run Microsoft Office or play video games except for simple calculations. Stored program was a giant jump in making machine hardware be independent of software programs that can run on it. This separation of hardware from software had profound effects on evolving computers.

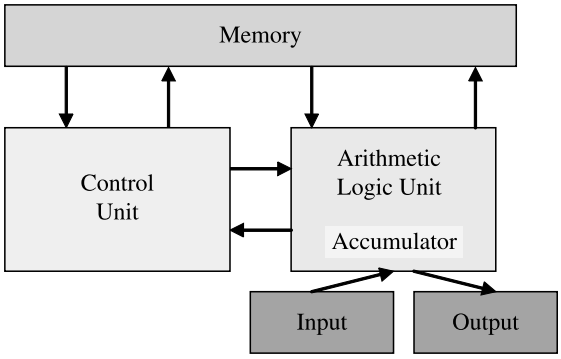


Figure 1.2 *von Neumann architecture.*

The latency associated with data transfer between CPU and memory was noticed as early as the von Neumann architecture. It was known as the *von Neumann bottleneck*, coined by John Backus in his 1977 ACM Turing Award lecture. In order to overcome the von Neumann bottleneck and improve computing efficiency, today's computers add more and more cache between CPU and main memory. Caching at the chip level is one of the many very crucial performance optimization strategies at the chip hardware level and is indispensable for modern computers.

In the next section, I'll give a brief overview about the Zuse machine, which was the earliest generation of commercialized computers. Zuse built his machines independent of the Turing machine and von Neumann machine.

1.3 ZUSE MACHINE

When talking about computing machines, we must mention Konrad Zuse, who was another great pioneer in the history of computing.

In 1934, driven by his dislike of the time-consuming calculations he had to perform as a civil engineer, Konrad Zuse began to formulate his first ideas on computing. He defined the logical architecture of his Z1, Z2, Z3, and Z4 computers. He was completely unaware of any computer-related developments in Germany or in other countries until a very late stage, so he independently conceived and implemented the principles of modern digital computers in isolation.

From the beginning it was clear to Zuse that his computers should be freely programmable, which means that they should be able to read an arbitrary meaningful sequence of instructions from a punch tape. It was also clear to him that the machines should work in the binary number system, because he wanted to construct his computers using binary switching elements. Not only should the numbers be represented in a binary form, but the whole logic of the machine should work using a binary switching mechanism (0–1 principle).

Zuse took performance into account in his designs even from the beginning. He designed a high-performance binary floating point unit in the semilogarithmic representation, which allowed him to calculate very small and very big numbers with sufficient precision. He also implemented a high-performance adder with a one-step carry-ahead and precise arithmetic exceptions handling.

Zuse even funded his own very innovative Zuse KG Company, which produced more than 250 computers with a value of 100 million DM between 1949 and 1969. During his life, Konrad Zuse painted several hundred oil paintings. He held about three dozen exhibitions and sold the paintings. What an interesting life he had!

In the next section, I'll introduce the Intel architecture, which prevails over the other architectures for modern computers. Most likely, you use an Intel architecture based system for your software development work, and you may also deploy your software on Intel architecture based systems for performance and scalability tests. As a matter of fact, I'll mainly use the Intel platform throughout this book for demonstrating software performance optimization and tuning techniques that apply to other platforms as well.

1.4 INTEL MACHINE

Intel architecture based systems are most popular not only for development but also for production. Let's dedicate this section to understanding the Intel architecture based machines.

1.4.1 History of Intel's Chips

Intel started its chip business with a 108 kHz processor in 1971. Since then, its processor family has evolved from year to year through the chain of 4004–8008–8080–8086–80286–80386–80486–Pentium–Pentium Pro–Pentium II–Pentium III/Xeon–Itanium–Pentium 4/Xeon to today's multicore processors. Table 1.1 shows the history of the Intel processor evolution up to 2005 when the multicore microarchitecture was introduced to increase energy efficiency while delivering higher performance.

1.4.2 Hyperthreading

Intel started introducing its hyperthreading (HT) technology with Pentium 4 in 2002. People outside Intel are often confused about what HT exactly is. This is a very relevant subject when you conduct performance and scalability testing, because you need to know if HT is enabled or not on the systems under test. Let's clarify what HT is here.

First, let's see how a two physical processor system works. With a dual-processor system, the two processors are separated from each other physically with two independent sockets. Each of the two processors has its own hardware resources such as arithmetic logical unit (ALU) and cache. The two processors share the main memory only through the system bus, as shown in Figure 1.3.

TABLE 1.1 Evolution of the Intel Processor Family Prior to the Multicore Microarchitecture Introduced in 2005

Year	Processor	CPU Speed	Addressable Memory
1971	4004	108 kHz	640 bytes
1972	8008	200 kHz	16 kilobytes (kB)
1974	8080	2 MHz	64 kB
1978	8086	10 MHz	1 MB
1985	80386	16 MHz	4 GB
1989	80486	50 MHz	4 GB
1993	Pentium	66 MHz	4 GB
1995	Pentium Pro	200 MHz	4 GB
1997/98	Pentium II/Xeon	300/400 MHz	4/64 GB
1999	Pentium III/Xeon	500/555 MHz	4/64 GB
2001	Xeon/Itanium	1.7/0.8 GHz	64 GB/1 TB
2001	Pentium 4/Xeon	2 GHz	4 GB
2003	Pentium 4 HT/Xeon	3/3 GHz	4/64 GB
2004	Itanium 2	1.6 GHz	1 TB
2005	Pentium 4/Xeon MP	3.7/3.6 GHz	4/64 GB

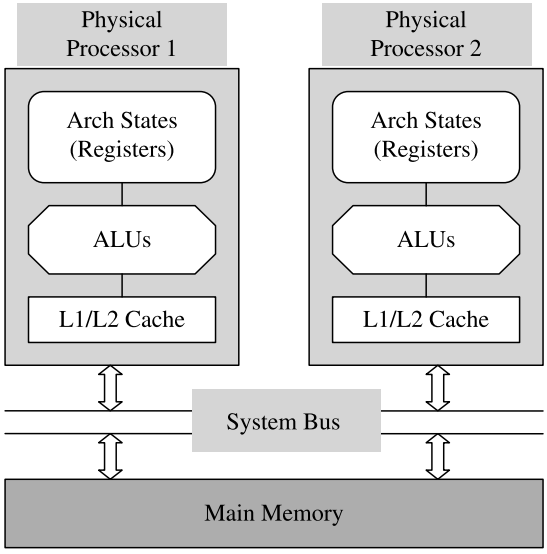


Figure 1.3 *Two physical processors in an Intel system.*

As shown in Figure 1.4, with hyperthreading, only a small set of microarchitecture states is duplicated, while the arithmetic logic units and cache(s) are shared. Compared with a single processor without HT support, the die size of a single processor with HT is increased by less than 5%. As you can imagine, HT may slow down single-threaded applications because of the overhead for synchronizations between

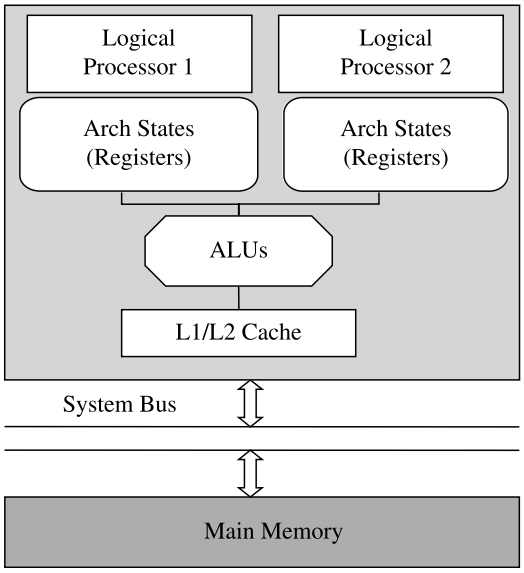


Figure 1.4 *Hyperthreading: two logical processors in an Intel system.*

the two logical processors. However, it is beneficial for multithreaded applications. Of course, a single processor with HT will not be the same as two physical processors without HT from the performance and scalability perspectives for very obvious reasons.

■ Case Study 1.1: Intel Hyperthreading Technology

How effective is hyperthreading? I had a chance to test it with a real-world OLTP (online transaction processing) application. The setup consisted of three servers: a Web server, an application server, and a database server. All servers were configured with two single-core Intel® Xeon™ processors at 3.4-GHz with hyperthreading support. The test client machine was on a similar system as well. The details of the application and the workload used for testing are not important here. The intention here is to illustrate how effective hyperthreading is with this specific setup and application.

Figure 1.5 shows the average response times of the workload with and without hyperthreading for different numbers of virtual users. The workload used for the tests consisted of a series of activities conducted by different types of users. The response time measured was from end to end without including the user's own *think times.* It was averaged over all types of activities.

With this specific test case, the effectiveness of HT depended on the number of users, ranging from 7%, to 23%, and to 33%, for 200, 300, and 400 users, respectively. The maximum improvement of 33% for 400 users is very significant.

As a matter of fact, the effectiveness of HT depends on how busy the systems are without HT when an intended load is applied to the systems under test. If CPUs of a system are relatively idle without HT, then enabling HT would not

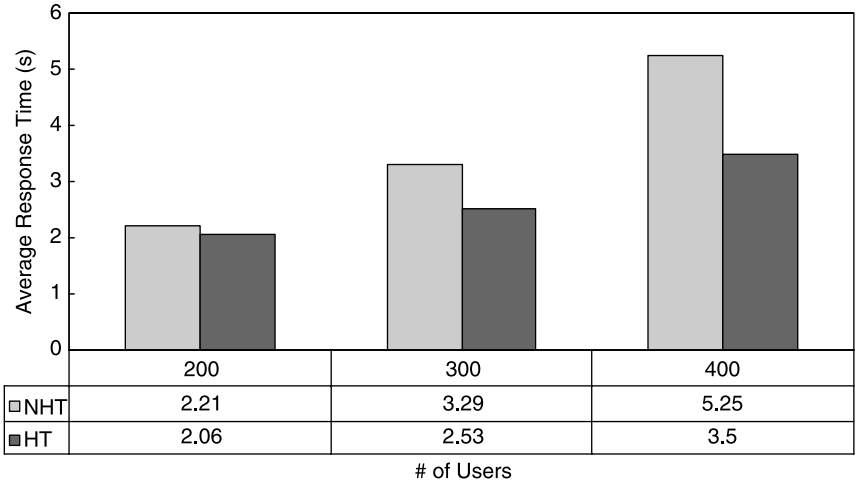
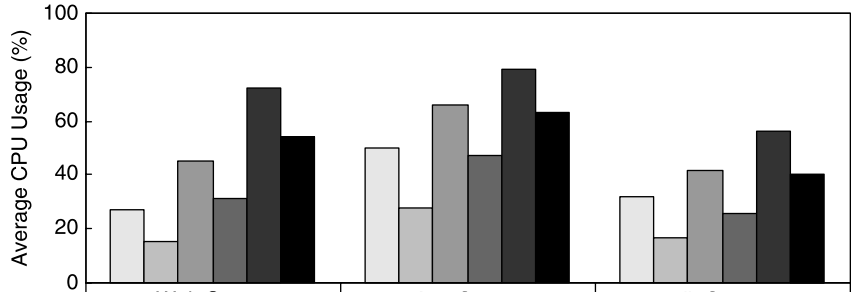


Figure 1.5 *Performance enhancements from hyperthreading (TH) in comparison with non-hyperthreading (NTH) based on a real-world OLTP application.*

help improve the system performance much. However, if the CPUs of a system are relatively busy without HT, enabling HT would provide additional computing power, which helps improve the system performance significantly. So the effectiveness of HT depends on whether a system can be driven to its fullest possible utilization.

In order to help prove the above observation on the circumstances under which HT would be effective, Figure 1.6 shows the CPU usages associated with the Web server, application server, and database server for different numbers of users with hyperthreading turned off and on, respectively. I have to explain that those CPU usage numbers were CPU utilizations averaged over the total number of processors perceived by the Microsoft Windows® 2003 Enterprise Edition operating system. With hyperthreading not turned on, the two single-core processors were perceived as two CPUs. However, when hyperthreading was turned on, the two single-core processors were perceived by the operating system as four processors, so the total CPU utilization would be the average CPU utilization multiplied by four and the maximum total CPU utilization would be 400%.

As is seen, the average CPU utilizations with HT turned on were lower than those with HT off. Take the Web server for 200 users as an example. With HT off, the average system CPU utilization was 27%. However, with HT on, the average system CPU utilization turned to 15%. This doesn't mean that the physical CPUs were about twice busier with HT off than with HT on. If we take into account the fact that those CPU utilization numbers were averaged over the total number



	Web Server	App Server	db Server
200_NHT	27	50	32
200_HT	15	28	17
300_NHT	45	66	42
300_HT	31	47	26
400_NHT	72	79	56
400_HT	54	63	40

Server

Figure 1.6 *Comparisons of server system CPU utilizations between nonhyperthreading (NHT) and hyperthreading (HT).*

of CPUs, it means that with HT off, each of the two CPUs of the Web server was 27% busy, whereas with HT on, each of the four CPUs of the same Web server was 15% busy; so overall the four CPUs in the case of HT-enabled did more work than the two CPUs in the case of HT-disabled; thus the overall system performance has been improved.

In the next section, I'll help you understand what Intel's multicore microarchitecture is about. Of course, multicore is a lot more powerful than hyperthreading, since a dual-core processor is closer to two physical processors than a single-core hyperthreaded processor is.

1.4.3 Intel's Multicore Microarchitecture

In contrast to hyperthreading, the Intel multicore microarchitecture shares nothing above L2 cache, as shown in Figure 1.7 for a dual-core configuration. Therefore both single-threaded and multithreaded applications can benefit from the multiple execution cores. Of course, hyperthreading and multicore do not contradict each other, as one can have each core hyperthreading enabled.

The Intel multicore microarchitecture resulted from the marriage of the other two Intel microarchitectures: NetBurst and Mobile, as shown in Figure 1.8. Note that Intel started to enter the most lucrative market of high-end server systems as early as Pentium Pro. That's how the NetBurst microarchitecture was born with the Xeon family of processors. The Mobile microarchitecture was introduced to respond to

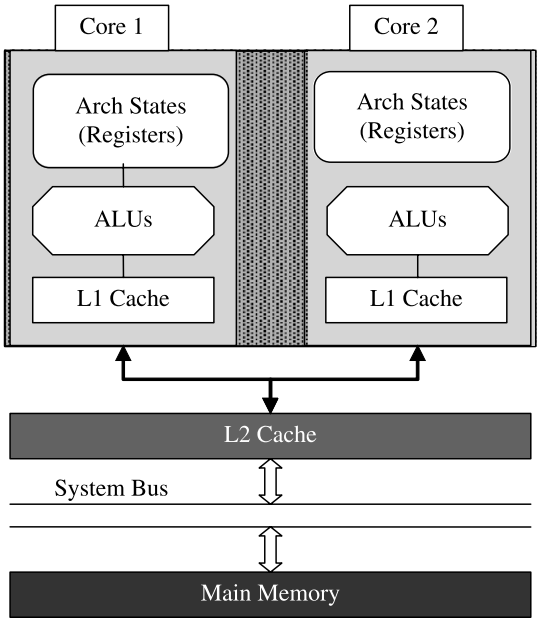


Figure 1.7 *Two execution cores in an Intel processor.*

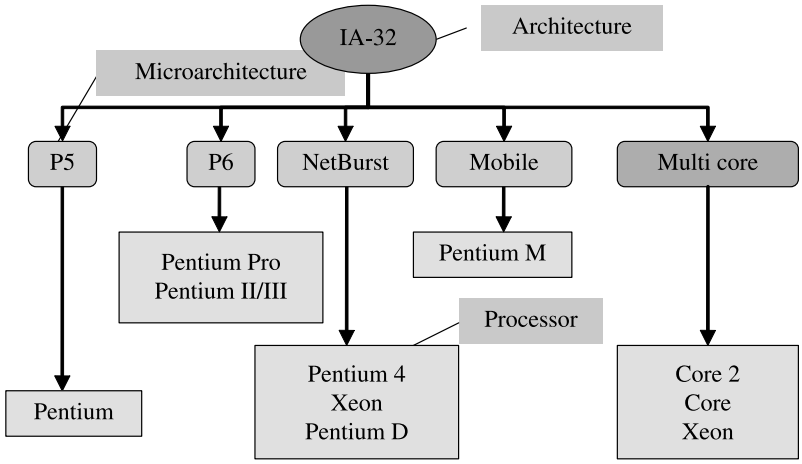


Figure 1.8 *History of the Intel 32 bit microarchitecture.*

the overheated mobile computing demands, for which low-power consumption was one of the most critical requirements. Combining the advantages of high performance from NetBurst and low power consumption from Mobile resulted in the new Intel multicore microarchitecture.

It's very necessary to differentiate among those three terms of *architecture, microarchitecture*, and *processor*:

- Processor architecture refers to the instruction set, registers, and memory data-resident data structure that is public to the programmer. Processor architecture maintains instruction set compatibility so that processors will run the programs written for generations of processors.
- Microarchitecture refers to the implementation of processor architecture in silicon.
- Processors are productized implementation of microarchitecture.

For software performance and scalability tests, one always needs to know the detailed specs of the systems being tested, especially the details of the processors as the brain of a system. It actually takes time to learn all about Intel processors. Here is a more systematic approach to pursuing the details of the Intel processors used in an Intel architecture based system. One should start with the processor number, which uniquely identifies each release of the Intel processors. It's not enough just to know the marketing names of the Intel processors. If you are using Intel architecture based systems for your performance and scalability tests, it's very likely that you are using Intel Xeon processor based systems.

Table 1.2 shows the specs of the latest Intel server processors. The specs include CPU type, CPU clock rate, front-side-bus (FSB) speed, L2/L3 cache, and hyper-threading support. It's interesting to see that Intel architecture is moving toward more and more cores while keeping increasing front-side-bus speed and L2/L3 cache. Hyper-threading support becomes less important as more and more cores can

TABLE 1.2 Intel 32-Bit Server Processors Classified by CPU Model, CPU Clock Rate, FSB (Front Side Bus) Speed, L2 and L3 Cache, and HT (Hyper-Threading) Support

CPU	Clock Rate (GHz)	FSB (NHz)	L2/L3 (MB)	HT
Xeon	3.00–3.80	800	2/–	Yes
Xeon MP	2.83–3.66	667	1/0–8	Yes
Dual-Core	1.66–3.50	667–1600	2/0–16	Some
Quad-Core	1.60–3.20	1066–1333	4–12/–	No
Six-Core	2.13–2.66	1066	9/12	No

be packaged in a single processor. Also the clock rate is not necessarily going higher with more cores. Most of the architectural design decisions were based on the goal of increasing performance by maximizing the parallelism that a multi-core processor can support.

On the desktop side, Intel has recently released a product family of Intel Core™ i7 processors. The Core™ i7 processors adopted a combination of multi-core with hyper-threading to maximize the multi-tasking capability for CPU processing power demanding applications. To maximize the I/O performance, Core™ i7 incorporated many advanced Intel technologies such as Intel® Smart Cache, Intel® QuickPath Interconnect, Intel® HD Boost, and integrated memory controller, etc, into the design. See Figure 1.9 for the image of an Intel Core™ i7 processor.

Now let's say you are using a Dell® PowerEdge® 6800 server. From looking up Dell's website, you would know that this system is using Intel's *3.0 GHz/800 MHz/2 × 2 MB Cache, Dual-Core Intel® Xeon 7041 Processor*. Then from Intel's website about *viewing processor number details* page for Xeon processors, you will find further details about the Dual-Core Xeon 7041 processor: for example, its system type is MP, which means that it can be configured with at least four or more processors. Some processors are labeled UP or DP, which stands for uniprocessor (UP) or dual-processor (DP). Also, it's capable of hyperthreading (HT).

Figure 1.9 *Intel Core™ i7 processor.*

It's very important that you are not confused about the terms of processor, UP/DP/MP, multicore, and hyperthreading when you communicate about exactly what systems you are using. Here is a summary about what these terms imply hierarchically:

- *Processor* implies the separate chip package or socket. A system with one, two, or N processors with N > 2 are called one-way (UP), two-way (DP), or N-way systems (MP).
- A processor could be a dual-core or quad-core processor with two or four cores in that processor. *Cores* are called *execution engines* in Intel's term.
- You can have hyperthreading turned on within each core. Then you would have two computing threads within each core.

Next, I'll provide a case study to demonstrate how important it is to keep up with the latest hardware advances in order to tap the highest possible performance and scalability potentials with a software application. A newer, faster computer system may even cost less than the older, slower one purchased just a couple of years ago.

■ Case Study 1.2: Performance and Scalability Comparison Between Intel's Single-Core and Multicore Processors

Figure 1.10 shows how effective the Intel multicore architecture could be compared with its single-core architecture, demonstrated with a real-world enterprise application that inserts objects into a database. The same tests were conducted with two different setups. In each setup, two identical systems were used, one for the application server, and the other for the database server.

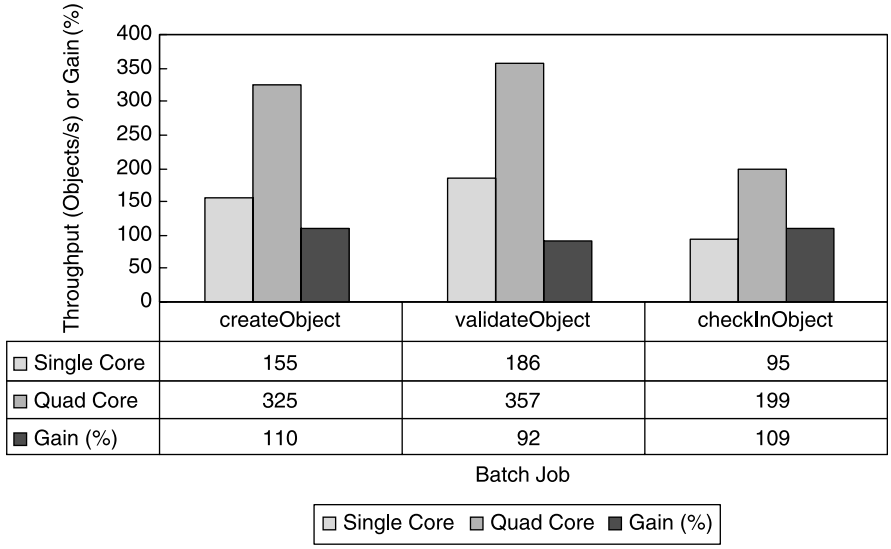


Figure 1.10 *Performance and scalability advantages of the Intel quad core over its single-core architecture.*

With the above test setups, the single-core setup was configured with two identical systems, each of which was equipped with four single-core Xeon processors at 3.67 GHz, whereas the quad-core setup was configured with two identical systems as well, each of which was equipped with two quad-core Xeon processors at 1.86 GHz. The total CPU power was the same between the single-core and quad-core systems. However, the quad-core setup outperformed the single-core setup consistently across all three different types of batch jobs by about a factor of 2, while the cost of each quad-core system was about only half of a single-core system. This shows how important it is to upgrade your hardware in time in order to get the maximum performance and scalability for your application while spending less.

New microarchitecture poses challenges for traditional system monitoring tools in terms of how CPU utilizations should be interpreted when logical or virtual processors are exposed to operating systems as if they were physical processors. This issue will be briefly discussed in the next section.

1.4.4 Challenges for System Monitoring Tools

It is confusing with hyperthreading and multicore with regard to how many physical CPUs a system actually has. For example, when you open up your Windows Task Manager on your system, you might see four CPUs displayed. Then you would wonder whether it's a four-way system, or two-way system dual-core per processor, or actually a single-processor dual-core system with hyperthreading enabled. If you are not sure, ask your system administrator to find out what's actually inside the box regarding the number of CPUs, cores, and hyperthreading.

Keep in mind that with your performance and scalability testing, you need to know exactly what systems you are using, because what systems you use will determine what performance and scalability you will observe for the software you test. Keep also in mind that the traditional operating system utilities fall behind the multicore and hyperthreading technologies. Whether it's a physical processor, a hyperthreaded logical processor, or a core, they all appear as a CPU to the operating system, which imposes challenges for interpreting the log data you collect with the processor performance counter.

Next, I'll introduce Sun machines, which are popular for IT production systems.

1.5 SUN MACHINE

Sun Microsystems® processor lines started with MicroSPARC I at 40–50 MHz introduced in 1992. Table 1.3 shows all Sun processors since 1998. The earlier Sun processors may have been retired in every IT organization. Note that UltraSPARC IV and IV+ are dual-core processors, whereas T1 and T2 are multicore, multithreading processors based on Sun's *CoolThread* technology. T1 and T2 were code-named *Niagara* and *Niagara II* processors. T1 has six pipeline stages, whereas T2 has eight pipeline stages, as shown in Figure 1.11.

TABLE 1.3 Sun UltraSPARC Processors Since 1998

Model	Speed (MHz)	Year	Threads/Core × Cores = Total Number of CPUs
UltraSPARC IIi	333–480	1998	1 × 1 = 1
UltraSPARC III	750–900	2001	1 × 1 = 1
UltraSPARC IIIi	1064–1593	2003	1 × 1 = 1
UltraSPARC IV	1050–1350	2004	1 × 2 = 2
UltraSPARC IV +	1500–1800	2005	1 × 2 = 2
UltraSPARC T1	1000–1400	2005	4 × 8 = 32
UltraSPARC T2	1400	2007	8 × 8 = 64

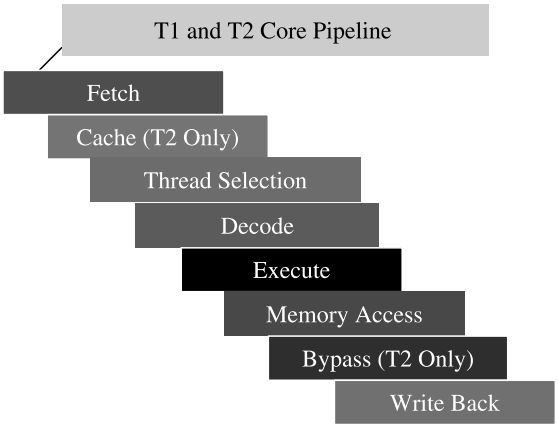


Figure 1.11 *Core pipelines for Sun T1 and T2 multicore, multithreading processors.*

It is helpful to understand how the new generation of Sun processors work. Essentially, one physically packaged processor can contain multiple cores, and one core can contain multiple threads. *Cores* don't share anything above L2 cache, whereas *threads* share everything below the register level. Those threads are termed *computing threads* in Sun's *throughput computing* marketing programs.

One can use the command "*psrinfo –vp*" to check out the processor type and the number of CPUs on a Sun system. However, it's necessary to make sure how many physical processors and logical CPUs or how many *cores* or *threads* are actually installed on the system.

In the next section, I'll show you how you can get to know quickly about your performance and scalability testing systems based on the latest Intel processors.

1.6 SYSTEM UNDER TEST

1.6.1 Processors

Your machine, whether it's a server class machine or a development desktop, is no doubt much more powerful than the machines built more than half a century ago. That's because modern processors have become millions of times faster.

TABLE 1.4 Comparison of Performance Between the IAS Machine and a Typical Modern Machine with Intel Xeon Processors

Spec	IAS	Modern Machine	Improvement (Times)
Memory	5 kB	12 GB	>2 million
Addition time	62 (μs)	1.3 (ns)	50 thousand
Multiplication time	713 (μs)	1.7 (ns)	400 thousand

In order to see the astronomical disparity, Table 1.4 compares the performance of one of the von Neumann machines with one of the typical Intel servers. This von Neumann machine was named the IAS machine, which was the first electronic digital computer built by the Institute for Advanced Study (IAS) at Princeton, New Jersey, USA, in 1952. A 3-GHz, dual-core, Intel Xeon 7041 processor is chosen arbitrarily for comparison. This processor is based on the Intel Core microarchitecture. In order to explain how we arrived at its performance for comparison, we need to explain the concepts of latency and throughput in the context of the Intel Core microarchitecture.

In the context of the Intel Core microarchitecture, latency is the number of processor clocks it takes for an instruction to have its data available for use by another instruction. Throughput is the number of processor clocks it takes for an instruction to execute or perform its calculations. A floating-point addition operation takes a latency of 3 processor clocks and a throughput of 1 processor clock. A single-precision floating-point multiplication operation takes a latency of 4 processor clocks and a throughput of 1 processor clock. Thus we can derive that the addition time and multiplication time of a modern Intel Xeon processor would be about 1.3 nanoseconds and 1.7 nanoseconds, respectively. Given its multicore and multithreading capability, a modern processor could be a million times faster than one manufactured half a century ago.

Even different models of the modern processors manufactured within a few years apart could exhibit drastically different performance and scalability with your software, as we have demonstrated with the preceding case study of the Intel multicore versus single-core comparison.

In the next few sections, let's expand more into the other parts of a computer system that have significant impact on the performance and scalability of a software system in general.

1.6.2 Motherboard

A powerful processor would starve to death without commensurate peripheral components to keep feeding it with instructions and data. In other words, a powerful processor needs a highly efficient environment to support it. That environment is provided by a motherboard, as shown in Figure 1.12.

The server motherboard shown in Figure 1.12 contains two dual-core processors, sixteen memory slots for installing up to 16 GB of RAM, two network ports, internal redundant arrays of inexpensive disks (RAIDs) controllers, peripheral component interconnect (PCI) slots, and a chipset. If you have a system of your own, you can

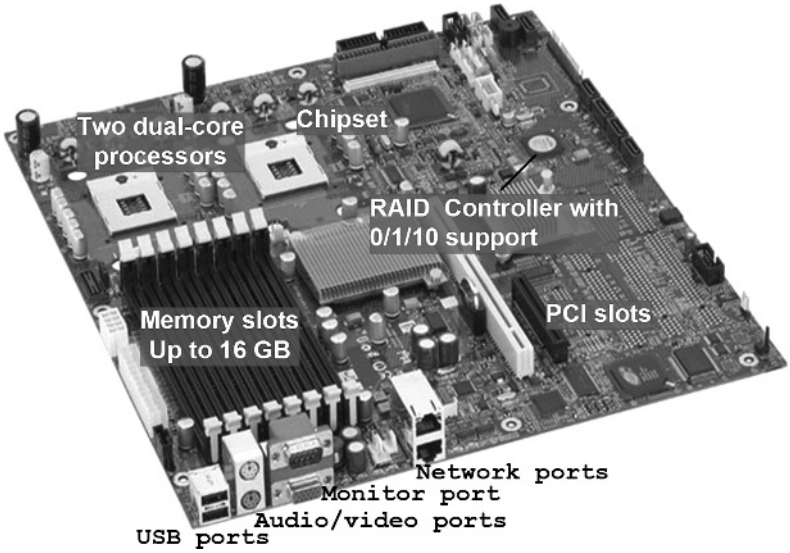


Figure 1.12 *Intel server board SE7520BB2 (courtesy of Intel).*

actually open the box yourself and get familiar with all the components on the motherboard.

Keep in mind that all the components on a motherboard are crucial for achieving super high performance out of today's Intel architecture based systems. When you evaluate your performance and scalability test results, you definitely need to know all the specs of your systems under test. This is also very necessary when you document your test results. I'd like to emphasize again that what performance and scalability you get with your software has a lot to do with what you have *inside* your systems.

You may often hear the other two terms of *chip* and *chipset*. A chip is basically a piece of integrated circuit that may contain millions of transistors. There are different types of chips. For example, processor chips contain an entire processing unit, whereas memory chips contain blank memory. Figure 1.13 shows the Intel Xeon uniprocessor (left) and multiprocessor (right) chips.

In the next section, I'll clarify what chipsets are.

1.6.3 Chipset

A chipset is a group of integrated circuits ("chips") that are designed to work together and are usually marketed as a single product. It is also commonly used to refer to the specialized chips on a motherboard. For example, the Intel E7520 chipset consists of three chips for facilitating data exchange between processors and memory through the front-side bus, and also between processors and secondary storage through the PCI bus.

Figure 1.14 shows that a chipset is partitioned into a memory bridge and an I/O bridge. These two bridges are normally called *north* and *south* bridges. The chipset

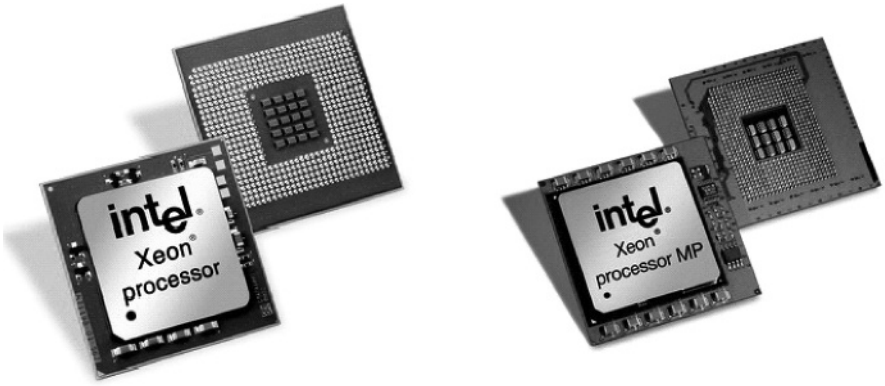


Figure 1.13 *Intel Xeon processor chips (courtesy of Intel).*

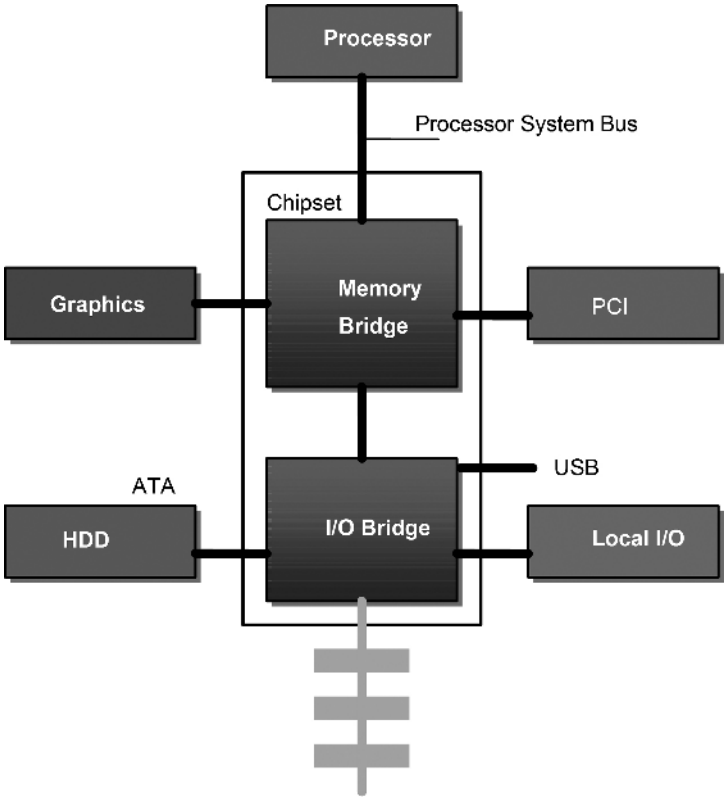


Figure 1.14 *Chipset acting as hubs of communication between a processor and its peripheral components.*

determines the type of processor, memory, and I/O components that a particular system can support. The chipset's efficiency directly affects the overall system performance.

Unfortunately, the components within a chipset are built-in and not very tunable from system performance perspectives. However, you can choose high-end components when you make a purchase to guarantee that you would get the highest possible performance while your budget permits.

Next, let's concentrate on the storage, which is as important as CPUs, since it determines how fast data can be moved among various data storage levels. Some examples will be presented in Chapter 6 to show how important I/O could be for enterprise applications from the system performance perspective.

1.6.4 Storage

Storage hierarchy is another important factor in determining the performance of a system. Figure 1.15 shows the various levels of storage based on the proximity of the storage layer to the processor, in the sequence of registers, caches, main memory, internal disks, and external disks.

In order to understand the impact of storage on the performance of a system, let's take a look at what each level of storage does for the system following the hierarchical sequence as shown in Figure 1.15:

- Registers are internal to a processor. They hold both instructions and data for carrying out arithmetic and logical calculations. They are the fastest of all forms of computer storage, since they are integrated on a CPU's chip, functioning as switches representing various combinations of 0's and 1's, which is how computers work as we all know.
- Cache memory consists of L1, L2, and L3 caches. L1 cache stores both instructions and data for reuse in order to increase the performance or "throughput" of a computer. L2 and L3 caches store the segments of programs that have just been executed, as well as the data already fetched from main memory for reuse in order to reduce the chances of refetching from the main memory. From L1 to L3, the access speed gets slower while the capacity gets larger.
- Main memory stores programs and data needed by the programs to be executed. Typically, modern computer systems are either 32-bit or 64-bit systems. The addressable memories for 32-bit and 64-bit systems are 4 GB and 1 TB, respectively, although the actual memory installed may be smaller or larger. Main memory is volatile in the sense that if the system is turned off, everything stored on it will be lost. Main memory communicates with the processor through the FSB (front-side bus).
- Hard disks are used for nonvolatile, mass storage of persistent data. There are a few different types of hard disks, for example, IDE (integrated drive electronics), SATA (serial advanced technology attachment), and SCSI (small computer system interface). IDE disks are traditionally used for home PCs. SATA disks

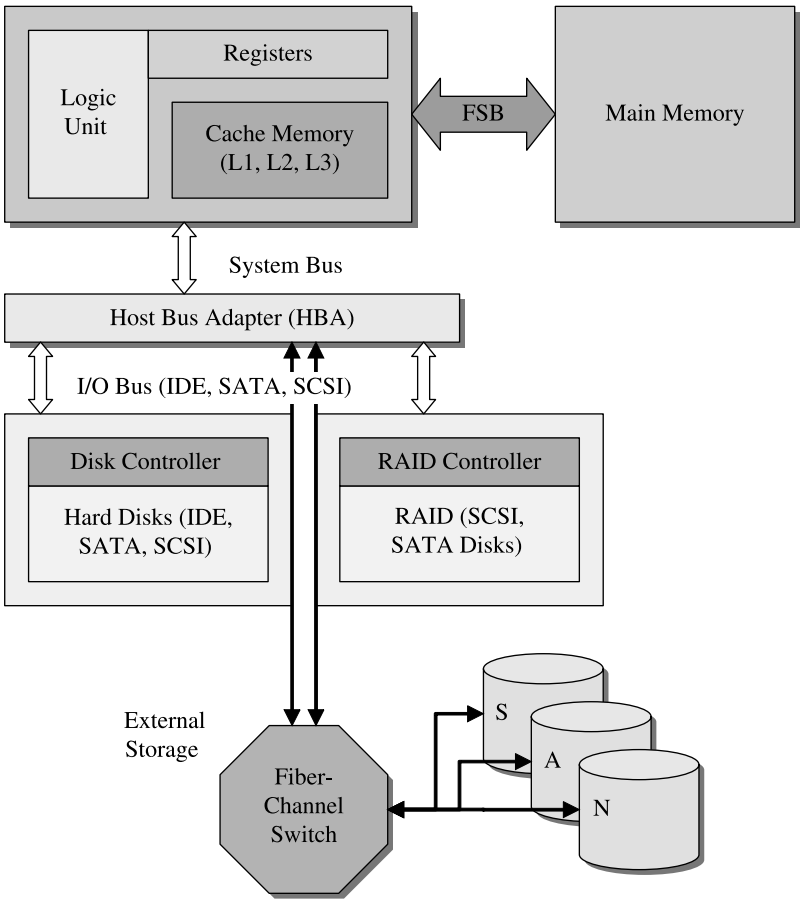


Figure 1.15 *Memory and storage hierarchies in a computer system.*

are used for video/audio production because of their affordability and large capacity (e.g., >700 GB). SCSI drives are used for more demanding workstations and enterprise server systems.

- Note that disks can be used as separate ones or configured as RAID (redundant array of inexpensive disks). I'll discuss more about RAID later, but for the time being, just take it as a logical approach to reorganizing disks for built-in redundancy for failure protection and parallel access to multiple disks for better performance.
- Hard disk drives are accessed over one of a number of bus types, including IDE, SATA, SCSI, and fiber channel. Host adapter or host bus adapter bridges the computer's internal system bus to its I/O bus to enable communications between a processor and its peripheral disk storage devices. Note that most enterprise storage adopts external SAN (storage area network) storage for storing and

processing enterprise data. In this case, a host bus adapter card is needed to connect the host computer with the remote external SAN storage through a fiber-channel switch.

Enterprise applications typically adopt external SAN storage to store business critical data. For performance and data protection purposes, SAN storage devices are configured with mirroring, striping, and error recovery techniques, resulting in various levels of RAID. Let's get to the details of RAID in the next section.

1.6.5 RAID

RAID stands for redundant arrays of inexpensive disks. We mentioned RAID in the previous section, but did not elaborate on it. Most of the IT organizations rarely use local independent disks for their data storage needs. Instead, RAID has been used as a standard approach to storing both static and dynamic enterprise data. From the software performance and scalability perspectives, properly configured RAID may improve data read/write throughput substantially. Therefore, we recommend using RAID rather than local independent disks, especially not one single local disk, for the performance and scalability tests with your enterprise software.

No matter how it is configured, the purposes with a RAID configuration are one, or more, or all of the following:

- Fault tolerance, which guarantees that if some disks of a RAID fail, the system can continue to be operational for sufficiently long so that the failed disks can be fixed without interrupting normal business uses of the system.
- Data integrity, which guarantees that if the data is partially lost or corrupted, the entire original data set can be recovered based on the remaining healthy and valid data.
- Higher data transfer throughput relative to a single or more separate disks, as data can be read from or written to multiple disks in parallel.

With a RAID configuration, multiple physical disks appear as one logical unit to the operating system. There are software RAID configurations and hardware RAID configurations. With a software RAID configuration, an abstraction software layer sits above the disk device drivers, which manage data transfer between physical disks and the operating system. Since this software layer consumes the CPU time of a local system, a software RAID configuration in general is much slower than a hardware RAID configuration. Therefore, if performance is a concern, a hardware RAID configuration is preferred over a software RAID configuration.

With a hardware RAID, a RAID controller is used to perform all calculations for managing the disks and maintaining the properties specified for a RAID configuration. This offloads RAID-related computing tasks from the main CPUs of the host system to RAID controller hardware, which improves the overall system performance. Another advantage of using a hardware RAID is that the built-in cache from a hardware RAID can help improve the overall system performance further.

A RAID could be internal or external to a system. An internal RAID uses the internal local disks and built-in on-board RAID controller of a system to realize a RAID configuration. An external RAID uses a fiber-channel switch to connect a RAID provided by an SAN (storage area network) to the host system.

How to configure and administrate a RAID is beyond the scope of this book. Here we will only provide enough information about the most commonly used RAID configurations so that you know how to choose a RAID configuration for conducting your performance and scalability tests for which I/Os are important.

RAID configurations are also termed *RAID levels*. The following RAID levels are most commonly seen in today's IT environment:

- RAID 0 (stripping). This RAID configuration spreads data across multiple disks. As shown in Figure 1.16a, multiple disks are connected to a RAID controller, which in turn is connected either internally or externally through a fiber channel

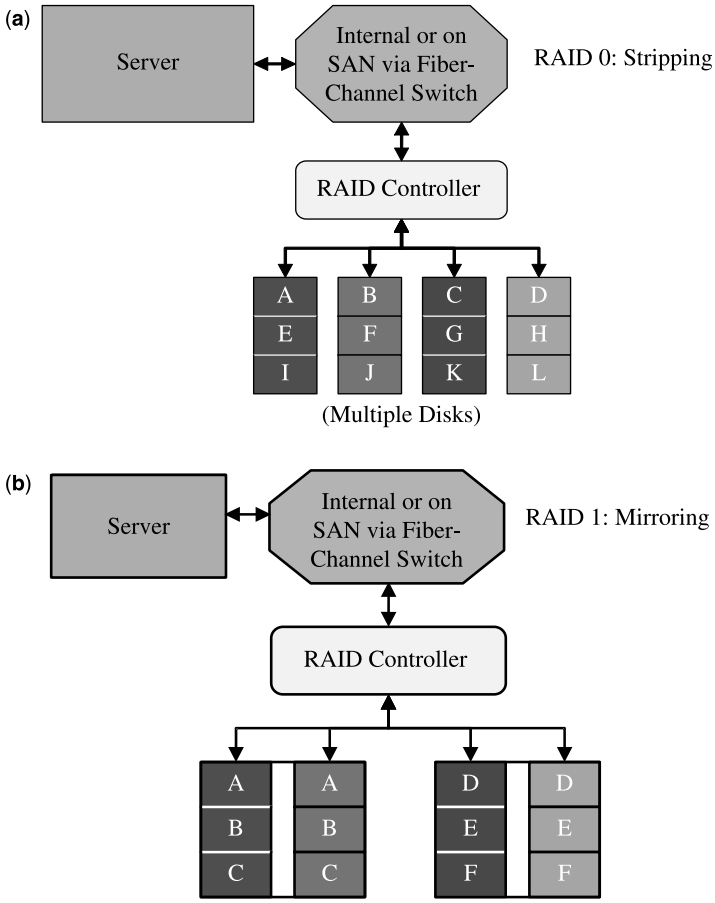


Figure 1.16 *(a) RAID 0 (stripping) configuration; (b) RAID 1 (mirroring) configuration; (c) RAID 10 (mirroring + stripping) configuration; and (d) RAID 5 (stripping +parity) configuration.*

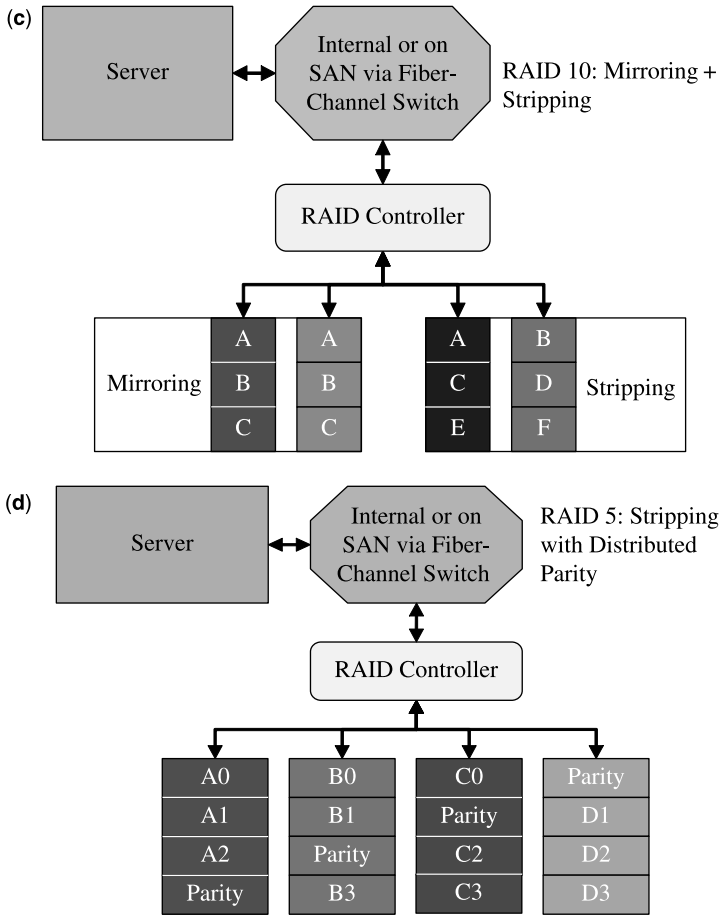


Figure 1.16 *(Continued).*

to the system. With the RAID 0 configuration, there is no redundancy or error recovery as data blocks are just written to multiple disks sequentially in a round-robin fashion.

- RAID 1 (mirroring). This RAID configuration duplicates data on multiple disks. As shown in Figure 1.16b, data is written twice on mirroring disks, which provides data redundancy but not error recovery. Also note that with RAID 1, twice the data storage capacity is required. This RAID configuration might be used in production, but not in performance and scalability testing environment.
- RAID 10. This RAID configuration is a combination of mirroring and stripping with mirroring first and then stripping, as shown in Figure 1.16c.
- RAID 5 (stripping with distributed parity). This RAID configuration spreads both user data and error correction data across multiple disks, as shown in

TABLE 1.5 Comparison Among Various RAID Levels

RAID Level	Advantages	Disadvantages
0	High read and write performance	No fault tolerance
1	Fault tolerance and twice the read throughput of single disks	Same write performance of single disks
10	Fault tolerance and high read and write performance	More disk space used than RAID 5
5	Fault tolerance and high read and write performance with less disk space used than RAID 1	Might be slower than RAID 0 because of parity calculations

Figure 1.16d. With this RAID configuration, error recovery is provided through distributed parity. Further details about various parity distribution algorithms are beyond the scope of this book, and interested readers should consult more specific texts about this subject.

There are other RAID levels such as RAID 2, 3, 4, 6, 7, 0 + 1, and 50, which represent different combinations of stripping, mirroring, and parity algorithms. Since these RAID levels are not used as commonly as the four RAID levels of 0, 1, 10, and 5, they are not explained here. From the performance and scalability testing perspectives, it's sufficient just to know those four most typical RAID configurations.

In a production environment, typically it's either RAID 10 or RAID 5 that is adopted. There is a tendency to recommend RAID 10 over RAID 5 in the published literature for database-intensive enterprise applications. See Table 1.5 for a summary of the advantages and disadvantages of those four typical RAID levels of 0, 1, 10, and 5.

For performance and scalability tests, I typically use RAID 0, which is not only easy to set up but also a good compromise between storage capacity and performance. If you don't have the option of using a RAID configuration other than the local disks, you should use at least three separate local disks for your enterprise data so that you do not always hit one disk hard with your I/O-intensive applications.

In addition to processors, main memory, and secondary storage, networking is another important factor that affects the performance and scalability of a software system under test. In the next section, I'll briefly introduce the concepts that are crucial for understanding the impact of networking on the performance and scalability of a software system.

1.6.6 Networking

Networking is essential for computers to communicate with each other, as computers rarely stand alone. Computers may sit on a LAN (local area network) to communicate with each other, for which the network latency is negligible. If computers communicate with each other across the continents, then the network latency would be a great concern.

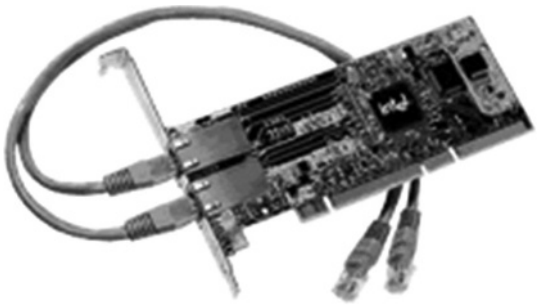


Figure 1.17 *Intel Pro/1000 MT Dual Port Server Adapter.*

In understanding how computer networking works, it's important to understand the following physical and logic entities:

- *Network Adapter*. A network adapter is the physical hardware that enables networking connectivity from a computer to a computer network. Figure 1.17 shows a typical Intel® Pro/1000 MT Dual Port Server Adapter which enables two Gigbit copper server connections in a single PCI slot. This adapter is designed to automatically scale with growing networks by auto-negotiating 10/100/1000 Mbps performance. It can enhance server performance further by teaming the two connections or teaming with other Intel® Pro Server Adapters to achieve multi-Gigbit scalability and redundant failover capability.
- *Network Bandwidth*. Network speed is typically measured by the amount of data that can be transferred in unit time. It is quantified by megabits/second (Mbps) or gigabits/second (Gbps). For example, your home network might be able to run at a maximum theoretical speed of 36 Mbps with your wireless port or at a maximum theoretical speed of 100 Mbps with your regular wired port. When you have two computers at home, you can actually connect your two computers through a crossover cable, which would be much faster than going through the wireless port if you have large amounts of data to transfer from one computer to the other. This is the simplest example of how network bandwidth could determine the networking performance of computers.
- *Network Types*. A network might be as simple as two computers connected through a crossover cable or as complicated as the entire Internet. Between these two extremes, there are many different types of networks whose names end with "AN," such as LAN, (local area network) and WAN (wide area network).
- *Network Latency*. Network latency is a different concept from network bandwidth. It measures how long it takes for a network packet to travel from point A to point B. Typically, network latency is less than 1 ms on a LAN, about 40 ms on a WAN, and 400 ms across continents. You can use the command of *ping <IP> -l size* to check out the latency for the systems that you are concerned with.

- *Network Protocol*. A network protocol governs how data is exchanged between various end points, which are typically the network ports of the computers which send and receive network data packets. The most popular network protocols are the TCP/IP protocols, which stand for transmission control protocol and Internet protocol. A network protocol essentially is the language that all computers and network routers use when they are configured to communicate with each other with that protocol.

From the software performance and scalability testing perspectives, you should have well-isolated networks for the computer servers under test, otherwise the results would be unpredictable. If you have to run your tests with your servers sitting on your corporate business network, you should verify your performance testing results at different times, for example, during busy morning hours and idle night hours, to isolate the effects of the shared network.

If your enterprise applications support global deployment architecture, you may want to use the servers spread across the continents to mimic the effects of network latency from a customer's usage perspectives.

As a software performance engineer, sometimes you may need to troubleshoot some network problems yourself. For example, one of the simplest tasks is to check if a remote server is up using the command "*ping* [*serverName* | *serverIPAddress*]" which works both on Windows and UNIX platforms. You can add the "*-l byteSize*" option to the *ping* command to obtain the network latency between two systems for a given packet size.

Another a little bit more advanced command is "*netstat*" which is helpful for checking the availability of the network ports on a system. Network ports cannot be shared among different applications on a system, and each application requires its own port number. For example, to check if a port number, say port 3269, is available, use the command "*netstat –a | find "3269"*"on Windows or "*netstat –a | grep 3269*" on UNIX.

Usually, these two commands (*ping* and *netstat*) are sufficient for trouble-shooting some very basic network problems. For more complicated network issues, you need to contact your network administrator who has the authority to check the network routers and switches, etc.

1.6.7 Operating System

In addition to hardware, operating systems play a critical role in determining not only the performance and scalability but also the reliability of a software product. It's beyond the scope of this text to discuss in detail how to configure an operating system for the maximum performance, scalability, and reliability for a software product. However, one should be aware of some of the major features of an operating system, for example, the version and the latest service pack, and whether it's a 32-bit or 64-bit system.

It is especially important to know whether the operating system in use is 32-bit or 64-bit, as the performance, scalability, and reliability of a software product may be

eventually limited by the amount of addressable memory space and the amount of total physical RAM. You can check whether an operating system is 32-bit or 64-bit by running some OS-specific script, for example, the *sysdm.cpl* script from the Run dialog box on Windows, or *isainfo −v* on Solaris.

In the next section, we'll discuss what would happen if those Turing assumptions were not satisfied in reality.

1.7 ODDS AGAINST TURING

The Turing model has been very useful as an abstract model for studying the concepts of computing. However, there is a gap between academic research and practical engineering. In this section, I'll show you what it implies when the assumptions made behind the Turing model could not be satisfied in reality.

Of those two assumptions associated with the Turing model, one is related to the unlimited memory or storage, and the other is related to the unlimited time for completing a computing task. Essentially, with these two assumptions, the Turing model doesn't care about how much memory a computer has and how fast the CPUs are. In reality, violation of the first assumption would result in system crashing and violation of the second assumption would result in slow performance, which would result in loss of revenues for customers and possibly penalties on the software vendor as well.

Although today's systems can be built with more and more RAM and faster and faster CPUs, various resource-intensive applications have been pushing the hardware resource limits as well. Issues such as memory leaks and slow CPUs still hinder adequate performance for more and more demanding software systems. Software developers and performance professionals must be committed to making sure that these two typical issues have been addressed properly with their products.

The effect of slow CPUs on the performance and scalability of a software system is obvious, and the impact of memory leaks is obvious as well. If a software product has severe memory leaks, sooner or later, it will crash with out-of-memory errors. In the next section, I'll help you understand what memory leaks are and how to monitor memory leaks. However, be aware that fixing memory leaks might be an unattainable task due to the difficulties of determining where memory leaks occur in the source code.

1.7.1 Memory Leaks

What is memory leak? Memory leak is a term describing the phenomenon of continuous memory consumption growth as a software application keeps running. We cannot expect zero memory growth. In the meanwhile, we cannot tolerate unlimited memory consumption growth without seeing software malfunctioning.

Software applications run as *processes* on computer systems. It's necessary to monitor the processes of a software application to see whether they have memory leaks or not over time.

Let's first take a look at a few memory-related performance counters from the *perfmon* utility on the Windows platform. I typically use the following *process*-related performance counters to monitor memory consumption of the processes I care about:

- *Private bytes*. This counter represents the current size, in bytes, of memory that a process has allocated that cannot be shared with other processes.
- *Working set*. This counter represents the current size, in bytes, of the working set of a process. The working set is the set of memory pages touched recently by the threads in a process. If free memory in a computer is above a threshold, pages are left in the working set of a process even if they are not in use. When free memory falls below a threshold, pages are trimmed from working sets. If they are needed, they will then be soft-faulted back into the working set before leaving main memory.
- *Virtual bytes*. This counter represents the current size, in bytes, of the virtual address space a process is using. Use of virtual address space does not necessarily imply the corresponding use of either disk or main memory pages. Virtual space is finite, and a process can limit its ability to load libraries.

These explanations of each counter are from *perfmon* help. I found that *private bytes* and *working set* are essentially the same, since they are not distinguishable from each other when they are drawn on the same chart, as you will see soon. *Virtual bytes* is more like a chunk of memory allocated for a process at a certain point of time. When *private bytes* or *working set* gets close to *virtual bytes*, more virtual memory will be allocated, which leads to a staircase-like pattern for virtual bytes when more and more memory spaces are needed by a running process.

■ Case Study 1.3: Memory Leaks

Figure 1.18 shows memory leaks from an application server process. The virtual bytes went beyond 2 GB three times. The working set and private bytes curves overlapped and spiked whenever the virtual bytes went beyond 2 GB.

This memory leak test was conducted on a Windows 2003 server. The 32-bit Windows operating system has a 4-GB total addressable address space, which is divided into two 2-GB for the operating system itself and applications, respectively.

One can also monitor memory leaks using the *perfmon* counter of *Available Memory in MBytes*, as shown in Figure 1.19. It's interesting to note that *Available Memory in Mbytes* shown in Figure 1.19 is almost a reverse reflection of the *virtual bytes* shown in Figure 1.18.

Figure 1.20 shows the application server process CPU usage associated with the memory leaks shown in Figure 1.18. It's interesting to note that whenever the virtual bytes reached that 2-GB limit, the application server restarted itself, resulting in all memory releases in the application server process. Similar measures in production environment can be taken in combating memory leaks by manually restarting the applications during the maintenance windows, either weekly or biweekly.

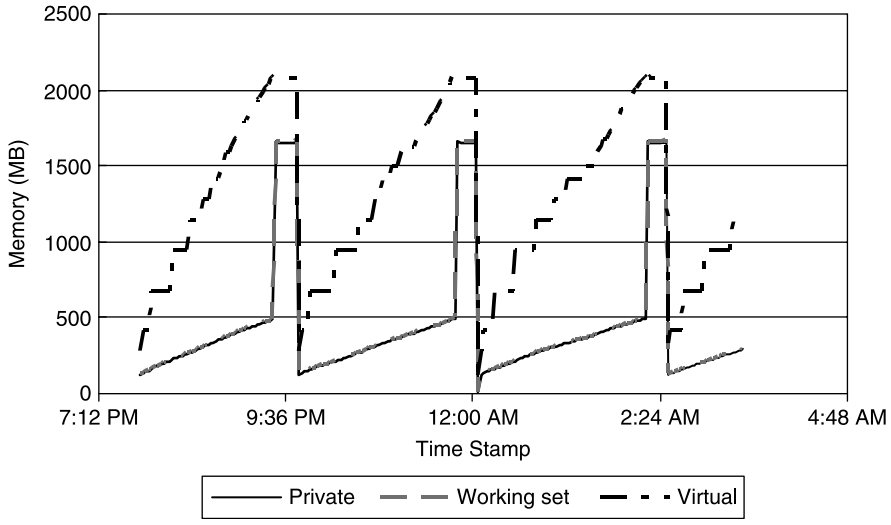


Figure 1.18 *An example of memory growth of an application server process.*

Of course, memory leaks are not remedyless. One can use certain tools such as IBM/Rational Purify™ to detect and fix memory leaks effectively. Figure 1.21 shows the memory consumption of the same application server with the same test after the memory leaks were fixed. It's seen that all memory consumption curves were flat. The virtual memory started with 278.5567 MB and ended with 278.5567 MB.

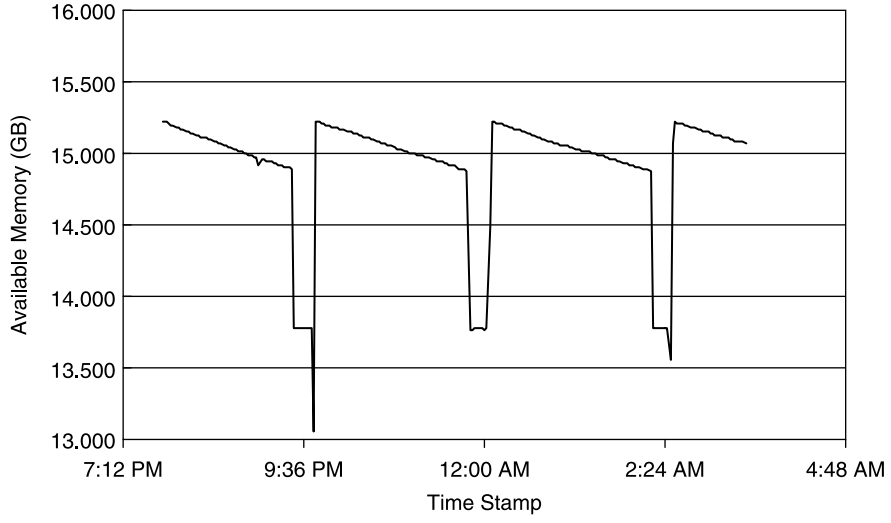


Figure 1.19 *Total available memory on the application server.*

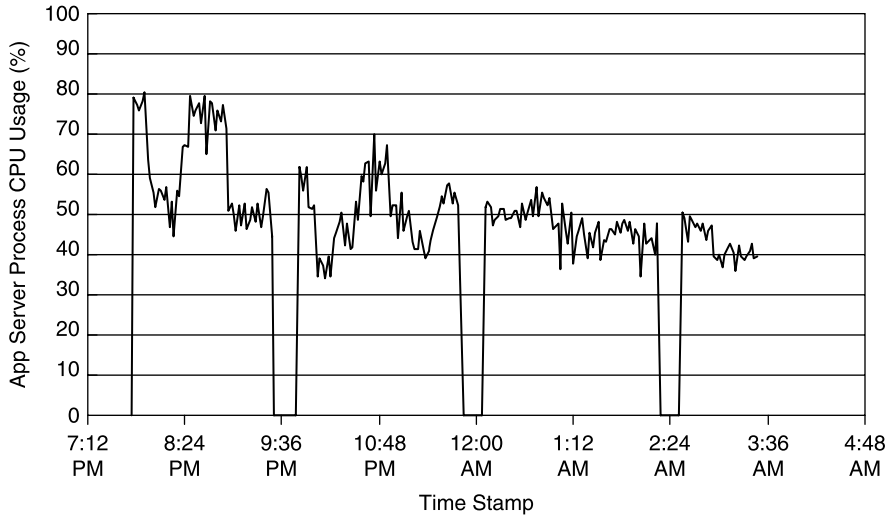


Figure 1.20 *Application server restarted itself whenever its memory leaked beyond the 2-GB limit.*

Although a 32-bit system has a 4-GB addressable virtual address space only, more than 4-GB physical memory can be installed on 32-bit editions of Windows systems with the help of the Intel's Physical Address Extension (PAE) technology. In addition, the 2 GB/2 GB default partition can be adjusted to 3 GB for an application and 1 GB for the kernel with a combination of the 4-GB tuning (4GT) feature and the `IMAGE_FILE_LARGE_ADDRESS_AWARE` value of the `LOADED_IMAGE`

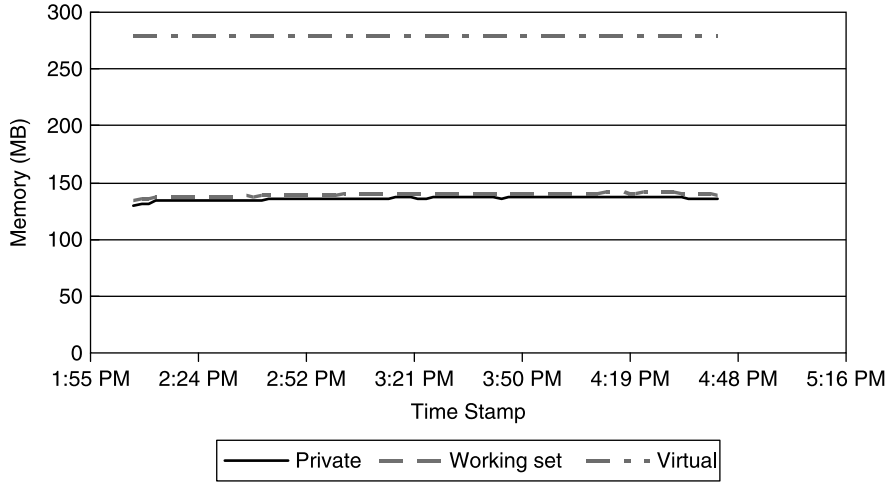


Figure 1.21 *Flat memory growth after memory leaks were fixed.*

structure. On 64-bit editions of Windows, one can increase the 2-GB limit up to 4 GB by setting the parameter `IMAGE_FILE_LARGE_ADDRESS_AWARE` for a 32-bit application.

One might think that moving to a 64-bit version of an operating system might help combat the memory barrier to some extent. Keep in mind that for the same application with the same workload it might take more memory in a 64-bit environment than in a 32-bit environment. This needs to be taken into account when sizing the physical memory requirements for your applications.

Although the maximum addressable space in a 64-bit environment can be as large as 1 TB, the actual addressable space will still be limited by the amount of physical memory installed on the system. Of course, one can adjust the virtual memory (swap space on UNIX or paging file size on Windows) to allow an application to use a largest possible addressable space. Both the swap space on UNIX and paging file size on Windows are configured very conservatively by default. On Windows, the paging file size is typically 2 GB or less by default, while on UNIX, the swap space is configured to be the same as the amount of total physical memory available by default. However, you should check the virtual memory configured on your system if you know that memory is one of the most limiting resources for your application.

On Windows, you can reconfigure paging file size by accessing *Performance Options* from *System Properties* → *Performance* → *Settings* → *Advanced* → *Virtual Memory* [*Change*]. On Solaris, follow the below procedure:

- Run command "*mkfile –v 10000 m /export/data/mySwapFile*" to create your swap file. This command creates a 10GB swap file named *mySwapFile* in the directory */export/data*. Of course, you can be flexible with the amount of space, file name and the directory name to suit your needs.
- Add an entry of "*/export/data/mySwapFile - - swap – no –*" in the file */etc/vfstab*.
- Issue the command "*swap –a /export/data/mySwapFile*".
- Check the swap file added with the command "*swap –l*".

The actual amount of virtual memory (swap or paging file size) to configure is application-dependent. The general rule of thumb is not to exceed 2–3 times the total amount of physical memory installed on the system.

Some of the latest Intel Xeon-based systems support both 32-bit and 64-bit operating systems. Xeon processors are intrinsically 32-bit, but can be extended to 40 bits with Intel's EM64T technology. Therefore, when you have a 64-bit operating system running on a Xeon-based system, the maximum amount of physical memory that can be installed might be limited to 64 GB. Check with the vendor of your system to make sure the exact configurations you have on your system.

In the next section, I'll discuss the service level agreements (SLAs) that have something to do with the other assumption made for the Turing model.

1.7.2 SLAs

Today's business corporations strongly depend on software to manage their daily operations and generate revenue. Apparently, the software they buy should not only do what it is supposed to do, but it should also perform adequately. This means that we do not have unlimited time for the software program to do what it is supposed to do. For some critical business services, there are certain performance requirements that must be met in order to achieve the projected business objectives. When such software performance requirements are written into a service contract, they effectively constitute a service level agreement (SLA) as part of a service contract between service providers and their customers. So the purpose of an SLA is to help make sure that a service provider lives up to its commitments by providing the level of services that both parties agreed upon.

The customers want to get what they paid for, and they cannot tolerate unbounded performance of the software they purchase without having their critical businesses impacted. For example, a stock exchange software program must guarantee the completion of a buy or sell order transaction within a certain amount of time; otherwise the customer may lose money. For Web-based online retailers, their transaction systems must be able to process purchases within a reasonable time frame so that users would not lose patience and go away, causing loss of revenue.

However, just as life insurance doesn't guarantee life, SLAs don't guarantee levels of service. When SLAs are broken in production, the customers would lose revenue and the service providers may get monetary penalties, which would cause hardships to both parties. That's why it's important to have performance designed and built into a product before deploying at a customer's site so that SLAs, if any, will not be broken. This book provides many performance optimization and tuning techniques that can be applied in general to help make sure that the SLAs bound to a software product will be satisfied.

1.8 SIZING HARDWARE

Sizing hardware for a software system is about establishing a quantitative, empirical formula describing the performance that a specific hardware platform is capable of delivering, or for a given performance requirement, what hardware platform would be required to achieve it. Such a sizing guideline would be extremely handy for the customers to decide on what hardware to acquire, or for a software provider to troubleshoot the customer performance escalations caused by undersized hardware.

In order to arrive at an effective sizing guideline, one should try to optimize and tune the software under test for the best possible performance and leave the hardware as the only factor that determines the performance of the software. This section provides an example of the throughput of the software being predominated by the CPU power of both the database server and the application server. The sizing formula has worked very well for many real customer occasions.

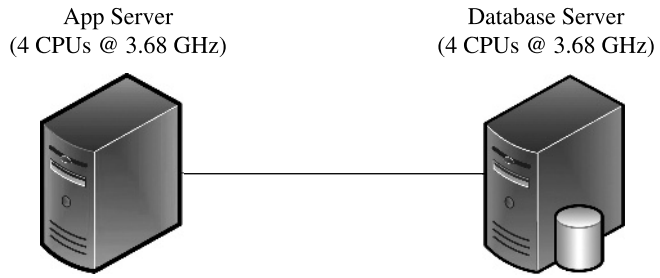


Figure 1.22 *A test configuration for sizing hardware.*

■ Case Study 1.4: Sizing Hardware

Figure 1.22 shows the test configuration, which consists of two servers, a database server, and an application server, for an enterprise software application. The two server systems were identical, with four single-core Intel Xeon processors at 3.68 GHz on each system. Both the application server and the database server were optimally configured with the best performance optimization and tuning practices obtained through extensive internal lab tests. In addition, the two server systems were located on the same subnet, which was another important test condition.

With a specific enterprise application batch job and a rigorous, repeatable test procedure, the following sizing formula, at which level this configuration was capable of processing, was found:

$$2 \text{ Transactions per second per CPU GHz of the application server}$$

with the assumption that the database server is equal to or more powerful than the application server. The CPU power or CPU GHz of the application server was defined as follows:

$$\text{Total CPU GHz of the application server} = 4 \times 3.68 \text{ GHz} = 14.72 \text{ GHz}$$

Based on the above specs, this setup was able to deliver a throughput of 29 transactions/second.

This sizing formula has been consistently verified with several internal and external customer tests. It has been used successfully for resolving a few customer performance escalations, which fell into the following categories:

- Undersized hardware attributed to the low throughput, which was improved to what the sizing formula predicted after upgrading the hardware.
- Inappropriately configured database and application caused low throughput, which was improved to what the sizing formula predicted after configuring the systems to the recommended optimal settings.

This case study is provided as a reference for you to size your application when applicable. Keep in mind that a sizing formula needs to be adjusted with time as the

performance of the application gets improved or degraded due to the changes that have to be introduced in the worst case.

1.9 SUMMARY

Understanding software performance and scalability begins with understanding computers, because it's the hardware that will eventually determine the performance and scalability of a software program after all optimizations and tunings have been exhausted on the software side. In order to help enhance the consciousness of the dependency of software performance and scalability on the underlying hardware platform, we have spent this entire chapter going over all major aspects of computers such as CPUs, memory, disks, and networking. Hopefully, this helps establish a baseline for necessary hardware knowledge required for doing performance and scalability test work, whether it's for benchmarking, sizing, optimization, or tuning.

This chapter is about hardware in general. The next chapter will be about software in general. Chapter 3 will be focusing on all concepts associated with testing software performance and scalability in general. These three chapters constitute the foundations for performing actual software performance and scalability test work. I hope that after studying these three chapters, you will be able to start asking the right questions about software performance and scalability and carry out your tests efficiently to help resolve any performance and scalability challenges associated with your products.

RECOMMENDED READING

For Moore's law, see his publication:

Gordon E. Moore, Cramming more components onto integrated circuits, *Electronics*, Vol. 38, No. 8, April 19, 1965.

To understand computers in general, the following two textbooks are excellent sources:

W. Stallings, *Computer Organization and Architecture—Designing for Performance*, 4th edition, Prentice Hall, 1996.

D. Patterson and J. Hennessy, *Computer Organization & Design—The Hardware/Software Interface*, Morgan Kaufmann, 1994.

See the following texts for quantitative approaches applied to refining software engineering and development:

L. Bernstein and C. M. Yuhas, *Trustworthy Systems Through Quantitative Software Engineering*, John Wiley & Sons, 2005.

S. Gabarro, *Web Application Design and Implementation: Apache 2, PHP5, MySQL, JavaScript and Linux/UNIX*, John Wiley & Sons, 2006.

L. Laird and M. Brennan, *Software Measurement and Estimation: A Practical Approach*, John Wiley & Sons, 2006.

The following text is an excellent source for applying the quantitative approach to understanding computer hardware architecture:

J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, Morgan Kaufmann, 2007.

EXERCISES

1.1. Get to know your computers. If you have access to a system (Windows or UNIX or Linux), find out the following specs on the system:

- System model
- Operating system
  - The exact version
  - 32 bit or 64 bit
- Processors
  - Processor model
  - Number of sockets or processors
  - Number of cores per processor
  - CPU clock rate
  - Hyperthread support if applicable
  - Front-side bus speed if applicable
  - L1/L2/L3 caches
  - Any other differentiating features
- Memory
  - Total amount of physical memory
- Network ports
  - Number of network ports and the speed that is supported
- Storage
  - File system
  - Internal or external
  - RAID level if applicable

1.2. This applies to the 32-bit Windows 2-GB limit. Develop a memory-intensive 32-bit sample application and run it on 32-bit Windows. Observe how the application would crash when the 2-GB limit is reached. Apply the Microsoft/Intel memory tuning techniques on the 32-bit Windows to increase the memory to 3 GB for the application and rerun. Observe the application behavior as the memory usage goes up with the application.

1.3. There is a popular perception that UNIX is faster than Windows. Envision the potential pitfalls and consequences of a recommendation based on such a perception if not verified with actual performance and scalability tests.

1.4. This concerns sizing hardware. The tests show that a setup similar to Figure 1.22, except that each server was equipped with two quad-core CPUs at 1 GHz, was able to deliver a throughput of 19 transactions/second. Apply the sizing formula introduced in Section 1.8 to this test configuration and calculate the expected throughput. Compare the measured throughput with the calculated throughput and what conclusion could you draw from it?

1.5. This concerns sizing hardware. A customer reported a low throughput of 0.8 transactions/second with a setup similar to Figure 1.22, except that both the application server and the database server were installed on a same system with 3 RISC processors at 1.6 GHz each. Based on the regression test, it was known that the application performance had been degraded by about 25%. Using the sizing formula in Section 1.8 with a 25% degradation taken into account, it was calculated that this test configuration should be able to process 3.6 transactions/second. The customer was advised to apply all recommended database and application configuration settings, rerun the test, and look for an expected throughput of 3.6 transactions/second. The customer reported back that processing 15,900 transactions took about 71 minutes after the database server and application server were optimally configured as advised. Compare the newly measured throughput from the customer with the calculated throughput and what conclusion could you draw from it?

1.6. This concerns sizing hardware. A customer reported a low throughput of 2.8 transactions/second with a setup similar to Figure 1.22, except that the application server was equipped with four CPUs with the clock rate either at 2.2 GHz or 3 GHz and the database server had a total CPU power of 16 GHz. The exact CPU frequency of the application server was not provided and the 2.2 GHz or 3 GHz was looked up from the vendor's website based on the server model. In this case, it was estimated using the sizing formula in Section 1.8 that this setup should be able to deliver a throughput of 18–24 transactions/second based on the uncertainty of the application server CPU frequency. After further communications with the customer, it was found that the reported low throughput was due to turning the logging on. The customer was advised to rerun the test with logging off and expect a throughput in the range of 18–24 transactions/second. The customer reported back that a throughput of 22 transactions/second was achieved after logging was turned off and they were happy with that throughput. Calculate the impact of logging on the performance of the system in this case and what conclusion could you draw from it?

1.7. How can you effectively prevent customers from starting with undersized hardware for your software applications? What are the benefits of doing so?