INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Performance Analysis of Computer Systems

Michèle Perucic
School of Computer Science
McGill University, Montréal, Canada

July 2001

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Science

Abstract

With an ever-growing and more productive computer industry, the performance of computer systems has become a major concern. Problems related to computer performance usually occur either because the system is not correctly sized or because its resources are not adequately allocated.

Performance analysis of computer systems is the process of evaluating the current performance of a system by monitoring and studying its behavior under different loads. It involves a deep understanding of the functioning of the basic components of a system. Performance analysis is typically followed by performance tuning, in which required changes are applied to the system in order to achieve optimum performance.

In this thesis, we discuss the basics of performance analysis. The different resources of a system are described and an overview of performance-monitoring tools for these resources is presented. An application of performance analysis is also included: two new major systems at McGill University are analyzed (the library management system ALEPH and the finance system BANNER).

Résumé

L'industrie informatique ne cessant de s'accroître exige une productivité toujours plus grande. Ainsi, la performance des systèmes informatiques est devenue un élément essentiel. Les problèmes reliés à la performance d'un ordinateur surviennent généralement soit parce que le système n'a pas la capacité appropriée, soit parce que les ressources du système n'ont pas été adéquatement accordées.

L'analyse de la performance des systèmes informatiques s'effectue par l'évaluation de la performance courante d'un système, en surveillant et en étudiant son comportement sous différentes charges. Cette analyse exige une bonne connaissance du fonctionnement des composantes de base du système. L'analyse de performance est habituellement suivie d'une optimisation de la performance, opération durant laquelle les changements requis sont apportés au système en vue d'obtenir une performance optimale.

Cette thèse expose les éléments de base de l'analyse de performance. Les différentes composantes d'un système y sont décrites ainsi qu'un aperçu des outils de contrôle nécessaires. Une application de l'analyse de performance est également incluse: deux nouveaux systèmes importants de l'Université McGill sont analysés (le système de gestion bibliothécaire ALEPH et le système de gestion financière BANNER).

Acknowledgements

I would like to express my gratitude to everyone who has made this work possible.

Thanks to my supervisor Professor Gerald Ratzer for his continuous encouragement and support, as well as his precious advice and availability.

Thanks to Vice-Principal Anthony Masi for his financial support, and for the time and guidance he granted me.

Thanks to Gary Bernstein for his initial help in starting this project, and for giving me access to the Network & Communications Services facilities.

Thanks to Professor Roger Rigelhof for his help and advice that also made this work possible.

Many thanks to Shelly Feran for her technical supervision and invaluable help, and without whom this thesis could not have been achieved.

Thanks also to Jacek Slaboszewicz and to Lyne Thibault for their generous help and time.

Thanks to Georges Kanaan for helping me in solving some formatting problems.

I would also like to thank McGill University, where I have spent many useful and pleasant years.

Contents

Chapter 1

Introduction

1.1 Problem Definition

In any computer system, and particularly, in shared multi-user computer systems, performance-related problems commonly occur. These problems are often characterized by a slowdown or a saturation of the system. Performance problems directly affect productivity and are often very complex and hard to solve. However, by regularly monitoring the performance of a system, it is possible to avoid such problems.

Performance analysis is the process of determining the current performance of a computer system by observing and characterizing its behavior. Performance analysis examines how the computer is making use of its resources, and it looks for existing weaknesses in the system.

The main goal of performance analysis is performance tuning. Performance tuning is often mistakenly associated with increasing the speed of a computer system. In fact, tuning is the process of optimizing computer performance by *reconfiguring* the system in order to achieve the best possible performance with the existing resources. When reconfiguring cannot help, additional hardware has to be bought, and an upgrade is necessary.

Performance monitoring and tuning is an essential task that continues during the whole lifetime of a computer system because such a system is continuously evolving (typically the number of end-users increases over time, so the work to be

processed also increases). In fact, performance should be a design goal, i.e. it should be incorporated at the source, in the design phase of a system.

The material covered in this thesis is mainly based on Sun® servers running the Solaris™ operating system, with an emphasis on Oracle database systems, but almost everything presented is applicable to any UNIX-based system. Moreover, when tuning, the overall knowledge is portable between computer platforms, operating systems, and database management systems [DUN98]. The tools described in Chapter 3, however, are UNIX (mainly Solaris)-specific tools.

1.2 Previous Work

Performance of computer systems is a subject of interest for an increasingly large number of people. Many books have been written (and are still being written) about performance, and many vendors have developed different tools that analyze performance on most computer platforms.

In most cases, the books published in the field of performance are written by performance specialists, people who have actually had real experiences in analyzing and tuning all sorts of computer systems in big organizations [CPS98, LOU91, GCO96, JAI91, and ALO99]. Scholars in universities also often have a strong interest in performance [CER98, GEL00 and LIL00].

When working with Sun systems (running the Solaris operating system), the most famous book used is Adrian Cockcroft's *Sun Performance and Tuning* [CPS98]. It is a very rich book containing detailed documentation about the behavior of systems, with a large number of performance recommendations. It also includes a thorough description of the SE Toolkit, a very widely used toolkit developed by Cockcroft for analyzing performance on Solaris systems. A lot of documentation is also available for Sun system administrators from Sun Microsystems at http://docs.sun.com or http://www.sun.com/sun-on-net/performance.html.

Other books focus on capacity planning and performance management on Solaris [CER98] or more globally on UNIX systems performance tuning [LOU91]. These are very practical when analyzing UNIX-based systems, as they review very clearly all the system's aspects to be concerned with when looking at performance.

With increasingly large and complex databases today, database performance has become another very common area of interest. Many books explore the performance of relational database management systems such as Oracle [GCO96 and ALO99].

Some authors are mainly concerned with the theory of performance analysis and of performance measurement more specifically, without worrying about the platform on which the analysis is to be performed [GUN98, GEL00, JAI91, and LIL00]. Their work tends to be quite comprehensive/theoretical, as it is very mathematically oriented (sometimes prerequisites in statistics can help the reader).

As for the existing tools to manage systems performance, they are of wide variety. Besides the basic tools that are often included with the operating system, many commercial products are available. Vendors often refer to performance management using different terms, such as: performance monitoring, performance analysis, capacity planning, troubleshooting, and performance tuning [CPS98].

Performance management, as a subset of the computer industry, has experienced significant growth in the past few years due to an increasing interest in performance issues, and new products are still being developed as systems are getting more and more complex and specialized. The competition among vendors, however tends to lead to similar features and functionality among products.

1.3 Thesis Structure

This thesis is organized in two major parts.

Part I, "Performance analysis: Technical Background", overviews the theory of the vast subject of performance of computer systems.

> Chapter 2, "System Performance Overview", describes the basic resources that affect –and are affected by– performance, with a tuning approach. It also introduces the reader to the basic performance measurement parameters and methods.
>
> Chapter 3, "Performance Monitoring Tools", is an overview of the most common tools used when monitoring and analyzing the performance of a computer system. It describes the tools and techniques for using them and interpreting their output. It also includes a brief overview of some commercial tools.

Part II, "Performance Analysis of two systems: ALEPH and BANNER", is an application of performance analysis as it contains the practical analysis of two systems that went live recently at McGill University.

> Chapter 4, "Performance Analysis of ALEPH", includes a description of McGill's new library management system ALEPH, followed by its performance analysis.
>
> Chapter 5, "Performance Analysis of BANNER", contains an overview and analysis of the new system set up to manage the students, finance, and human resources information systems at McGill.

Part I

Performance Analysis: Technical Background

The following chapters provide the background necessary for performance analysis. Chapter 2 identifies the resources that are to be monitored, and describes the parameters to look for when collecting data, as well as the methods commonly used when analyzing performance. Chapter 3 describes the usage of common tools that are used for taking performance measurements and overviews some commercially available tools.

Chapter 2

System Performance Overview

Performance is a design goal; it is not something to worry about only when a system starts showing symptoms such as a slowdown or when users cannot log in anymore[1]. It is essential to integrate performance monitoring as a regular and continual practice during the whole lifetime of a system.

Indeed, in a system where performance measurements are regularly being taken, it is much easier to localize a performance-related problem when it occurs. Conversely, it is extremely hard to identify a problem when no previous performance measurements have been taken when the system was still healthy.

The following sections try to summarize performance theory in two parts. First, the basic system resources involved in delivering performance are identified and described with an emphasis on tuning. Then, an overview of common performance metrics follows, with a description of basic methods used for performance measurement and analysis.

2.1 System Resources

In any computer system, there are four basic components that interact and affect overall system performance. Those components are often referred to as subsystems or resources, since each one of them is a an integral and vital part of the system:

[1] Note that this is an extreme case. Performance problems are usually identified and solved before this happens.

- The *Central Processing Unit* (CPU) – It fetches instructions from main memory, processes them, and executes them;

- The *Memory* (RAM) – It is the amount of physical/main memory on the system. It stores the operating system, the application programs, and the data in current use so that they can be quickly reached by the processor;

- The *Disk Input/Output* (I/O) Subsystem – It is the hard disk(s) and interface(s) of the system. It stores large amounts of data (including databases);

- The *Network* – It connects computers together, and transports and transmits the information into and out of a computer to another.

System resources are strongly interrelated and the performance of one resource often affects the performance of another resource. Also, tuning one subsystem can improve another subsystem, but it can also penalize yet another subsystem. So optimizing performance is somewhat a matter of making tradeoffs [LOU91 and OST96] (see following subsections).

Resources are meant to work together and in a useful manner. In other words, resources are meant to be used as fully as possible without performance degradation, because a system is meant to be fully utilized (up to 90%). A system where resources are always plentiful is an oversized system (i.e. the hardware bought is more than needed) [GCO96]. To get the best performance out of a computer system, it is important to understand how each of the components behaves and how it interacts with the other resources.

All system performance issues are really resource contention issues [LOU91 and ICCM00]. Every resource has its own set of limitations. When a performance problem exists, it is often very difficult to figure out which resource is causing it because of the complex interaction among the resources.

In the following subsections, we describe the role played by each of the fundamental devices listed above, with an emphasis on potential performance problems affecting each resource and on tuning.

2.1.1 The CPU

The CPU not only executes processes (which are sequences of instructions that form single units of work) but also controls the activities of the other system resources. It moves data to and from memory, reads/writes data from/to disk, and sends/receives data via the network. In current systems, the CPU is generally fast enough to handle the system's workload (i.e. the amount of work that the system has to perform, or the overall demands placed on the system)[2]; it is actually the fastest device in the whole system. So when CPU problems occur, lack of speed is probably not the cause. The problems are usually located somewhere else [ICCM00]. The causes can vary from untuned applications, to users opening multiple sessions of a program, long-running jobs running during peak hours, memory or I/O slowdown, etc. In an Oracle database system, CPU problems can be due to undersized or oversized memory allocations for Oracle memory structures [GCO96].

CPU problems also occur when too many processes are trying to use the CPU at the same time. This is a frequent situation and is referred to as CPU contention. In such a situation, a lot of processes end up waiting[3] in a queue because each one needs a certain number of CPU cycles to execute and the CPU is allocated fairly, so each process only gets a fixed number of CPU cycles (time slice). As the number of processes increase, the CPU gets monopolized. In some cases, a single huge process may also hog the CPU.

---

[2] Usually, in most On-Line Transaction Processing (OLTP) systems, it is reasonable to assume that the workload is the number of processes running on the system.

[3] At any point in time, a live process is either ready, running, or blocked (waiting).

There are a few ways to measure CPU contention. The most intuitive way is to check the load average (see section 3.1). The load average reports the number of active processes at any time. Although useful, this method is insufficient because it does not take into account the interdependence of system resources. Indeed, the load average can indicate a very high load while the CPU is mostly idle. This happens when the system is short on memory or when the I/O is slow (because processes waiting for I/O, for example, are reported as part of CPU time and get added to the load). Another more informative and reliable way to measure CPU contention is to look at detailed processes status reports (including the size, time, and priority of each process) or long-term summaries such as logs [LOU91]. Different ways of getting such information will be described in Chapter 3.

CPU utilization is another measure, which reports on the usage under the system's workload (see section 2.2.1). When the CPU utilization is high, this alone does not mean that there is a problem, but rather that the CPU is working hard and efficiently [ICCM00 and GCO96]. When a CPU problem really exists, the CPU utilization shows a very low idle time corresponding to a very low time waiting for I/O (both less than 5%) [OST96]. In such a case, more CPU power is probably necessary.

It is important to watch how the CPU is spending its time. At any given moment, the CPU is either servicing a user request (user time), or working in system mode to access resources (system time), waiting for I/O (wait time), or idling (idle time). The system time should not become greater than the user time (unless the machine is a Network File System – or NFS[4] – server), because in such a case it would mean that there is a process making many inefficient system calls and consequently wasting the CPU cycles, or that there is a network slowdown [SOB99, CPS98]. In general, experienced administrators believe that the 'user time:system time' ratio should stay around 2:1.

[4] An NFS server stores and provides files to different users over the network, as if the files were on the users' own computers.

Other process management issues should also be checked: scheduling and switching processes [OST96]. The operating system schedules processes and performs context switches (changing parameters for the next program to run). These operations can be time consuming for the CPU, especially when there are too many processes. In an Oracle system, when a lot of time is spent in context switches, this can be very CPU-intensive if the System Global Area (SGA: it is an area in memory that is shared by all the users, and which is used by Oracle to cache database data for faster access) is very large because when small processes are continually created and destroyed, page tables have also to be built up. This becomes particularly expensive when shared memory is locked, because then every page has to be updated and loaded again [GCO96].

Another problem with the CPU is *thrashing* [STA95, GCO96, and ICCM00]. Thrashing is one of the worst situations a system can be in. In such a situation, the CPU cycles become exclusively dedicated to moving processes from memory to disk and back again. Thrashing is due to excessive *paging* and *swapping* to and from memory (see section 2.1.2). To avoid thrashing, it is very important not to run out of memory. In the context of an Oracle system, particular care needs to be given to per-user memory settings [GCO96].

Generally speaking, memory problems affect the CPU more than do disk I/O problems because disk I/O demands CPU only after it releases data, whereas memory can make excessive and continual demands on the CPU (c.f. thrashing). Usually, taking care of the memory and I/O problems will help the CPU, since the CPU services both the operating system and the user programs.

In Client/Server environments, round trip latency when sending a message can cause CPU overload. A lot of overhead is generated by applications sending a message over and over again via the network.

In CPU tuning, it is important to (1) ensure that the computer is doing useful work 24 hours a day (get the system users to run long-running jobs at night or during off-peak hours), (2) set priorities to processes: make big jobs run at lower priority, (3) prevent the system from doing unnecessary work, etc [LOU91].

2.1.2 The Memory

The computer's random access memory (RAM), or simply memory, is the holding place for all the information currently needed and used by the CPU. It usually contains the operating system, the program instructions, the program data, and whatever the CPU is currently working on. Memory is quickly reachable by the CPU because it is physically located close to the processor in the computer. It is not the fastest accessible location though; the cache – similar smaller memory located right next to the processor – is the fastest temporary storage place. However, memory is by far the fastest when compared to disk storage. That is why it is always recommended to have a lot of memory; the more RAM there is in a system, the less frequently the computer has to access instructions and data from the hard disk.

Virtual memory is a mechanism, which allows more memory than there actually is on the computer to be available to programs. The way virtual memory works is by using a portion of the disk space, called *swap* space, and mapping it to memory. This way, 'logical memory' gets bigger than 'real' (physical) memory [STA95, WIS00, and ICCM00]. Note that swap space must be utilized effectively to improve performance: swap partitions should be placed on as many different disks as possible (but only one disk per controller) and on the fastest disks in the system, not on full or fragmented file systems.

In general, on most computer systems, memory is managed as a collection of *pages*, which are units of data of several kilobytes. When the operating system wants to allocate memory for a process, it first looks for any free pages in

physical memory (*scans* memory). If it cannot find any, it selects pages that belong to other active processes and copies them out (they are *paged-out*) to disk so that they get freed for the new process. This operation is called *paging* and is managed by the virtual memory subsystem. When a process needs to access its paged out page again, a *page fault* occurs, and the process has to wait until the page is copied back into memory (*paged-in*). A similar scenario happens with *swapping*, when processes are moved to swap space (on disk). The only difference between paging and swapping is that paging moves individual *pages* of processes to disk, whereas swapping moves *entire* processes to disk. Both paging and swapping need CPU cycles and therefore have an effect on CPU utilization. As the demand for memory increases, the paging and swapping rates also increase, as well as the associated CPU and I/O activity. When this activity gets so intense such that the CPU is practically only shuffling pages around (from memory to disk and back), the system is said to be *thrashing* (c.f. section 2.1.1).

Although excessive paging and swapping are clearly memory contention problems most of the time, they do not always indicate a shortfall in memory. In fact, under a virtual operating system, moderate paging is normal; it means that programs are making use of the extended memory space. Also, moderate swapping is often part of a normal "housekeeping" process: jobs that have been sleeping for a long time are usually swapped out to free memory. Basically, paging and swapping prevent the system from crashing due to a lack of memory [ICCM00 and CPS98].

However, because paging and swapping use the CPU and I/O devices, overall system performance drops significantly when their rates become too high, and performance does not come back to normal before the system's memory requirements are again within its capacity. So excessive paging and swapping really are signs of memory shortage [LOU91, CPS98]. In an Oracle system, the worst situation happens when the system's SGA is swapped out of memory

[GCO96]. Because disk access is so much slower than memory access, performance suffers when the system has to go to the disk in such a way.

It is possible to get information from the operating system about the behavior and performance of memory. Detailed statistics-reporting tools are described in Chapter 3. But how does one determine what excessive paging and swapping are? Usually, the thresholds depend on how the system reacts under load and what kind of performance is considered to be acceptable by the users and the administrator of the system.

In general, the free memory in a system should not go under 5%, otherwise no resources will be left when a new user logs in to the system. However, if free memory is always more than 10% during peak usage times, this means that the memory resource is not being used efficiently or that there was more RAM bought than needed [GCO96].

In Oracle systems, or in multi-user systems in general, each logged-in user is using memory even if he/she is not doing any work. Such inactive users can affect the response times (see definition in section 2.2.1) of the system when memory is scarce [GCO96]. Usually user patterns vary a lot and the amount of memory used by each user depends upon many parameters, such as the program the user is running and the language it is written in, operating system and Oracle parameters, the number of users on the system and how many sessions they are running, etc.

To solve memory problems, many approaches can be adopted [LOU91 and GCO96]:

- Keeping users from doing multiple logins and warning inactive users, will improve performance *and* security.
- By selecting the most appropriate paging algorithm (which is an algorithm that manages and allocates pages) for the system being tuned, the likelihood that

the memory shortage will become severe decreases. For example, configuring the system to start paging earlier might be a good idea.

- Tuning up the application programs so that their patterns of reference to memory are more consistent (i.e. closer) is an effective – though expensive – solution.
- Limiting the number of concurrent tasks will also help.
- In an Oracle system, properly allocating memory resources to Oracle memory structures can have a large impact on performance. When memory resources are properly allocated, cache performance can improve, parsing of SQL statements as well as paging and swapping can be reduced.
- In hopeless situations, terminating the jobs with the largest memory requirements would be a temporary solution.
- Adding memory will almost always improve overall performance.

2.1.3 The Disk I/O Subsystem

The disk is the typical storage device that provides access to large amounts of data. Disk capacity (or the amount of data that can be stored on the disk) is often measured in gigabytes (GB) or even terabytes (TB). In any system, the disk subsystem is vital to overall system performance. Typically, the I/O subsystem devices are orders of magnitude slower than memory [ICCM00]. Whenever possible, needed data should be located in memory rather than on disk, to avoid long waiting times.

On any system, the I/O subsystem is a common source of resource contention problems. When multiple processes are trying to access the same disk resources simultaneously, disk contention occurs. In the I/O subsystem not only are the individual devices (mainly disks) very slow, but also the I/O buses' transfers are limited by their bandwidth and all the programs running on a system must share that finite amount of I/O bandwidth [OST96]. When these limits are reached,

processes usually end up waiting to access the disk, and often CPU activity has to be suspended while I/O activity completes.

An important factor for the performance of a disk subsystem resides in the disk interface standard used (or its connection to the host system). Many disk interface standards exist; the most common ones are EIDE, SCSI, and Fibre Channel. For information about their features, refer to [SES99].

In large database systems, there is a lot of disk activity, so disk performance is crucial. Since physical memory is never large enough to hold all the data needed, disk accesses are inevitable. Because RDBMS (Relational Database Management Systems) such as Oracle store and process extremely large amounts of data, disk I/O issues are a major concern.

There are many aspects that influence the I/O subsystem performance. First, the way in which disk data is accessed – whether it is a read, a write, or an update (read-modify-write) operation – has two possible forms. Data can be accessed on a system *randomly* or *sequentially*. In a *random* access, reads and writes affect many separate small blocks (a few kilobytes of data) randomly; they usually occur in indexed[5] databases or when processes' pages are being paged-in and out. In a *sequential* access, larger and more localized blocks are accessed at once. Sequential reads and writes occur when large amounts of data are being processed or when files are created or copied. Updates occur when a database is making a sequence of transactions; it can be random or sequential. When working on I/O performance, understanding these concepts help in knowing what to expect [ICCM00 and CPS98].

A major improvement of overall disk performance can be achieved by well organizing data on disk, depending on the application(s) running on the system. It

[5] An indexed database is one where indexes are used. Indexes are essentially pointers to database tables that speed up data retrieval from the tables, avoiding full table scans. An indexed database usually reduces accesses to the database on the disk.

is important to configure the disk in such a way to spread the disk workload as evenly as possible across the disks and the controllers/interfaces to avoid potential disk overloading [LOU91, CPS98, GCO96]. In database systems, it is a good practice to separate the disk(s) on which the operating system is installed (and which will be used for paging and swapping – see section 2.1.2) from the other disks containing the database files. It is also good to put the database tables and their indexes on separate disks, because they are accessed simultaneously. The redo logs[6] should also be put on a separate disk with no other activity, because they are written sequentially, and sequential writing is much faster when no concurrent activity is performed on the same disk. Because a lot of database files are written at the same time when a transaction (involving writing to a data file, an index file, and a redo log) is processed, performance is improved when they are put on separate disks [GCO96].

To understand how a disk is organized, it is important to first understand what *filesystems* are and how they are used. A *filesystem* is a data structure that holds and organizes files in a hierarchical way. It is the most common disk layout (UFS – UNIX Filesystem – is the widest used filesystem type). A disk drive is split into several partitions and a disk partition holds one filesystem. Good planning and use of filesystems can help disk performance. It is important, for example, to keep similar files in the same filesystem in order to simplify configuration options, have a larger number of smaller fileystems rather than a few big filesystems (because data is more easily/quickly locatable when the filesystem is small), and put swap areas (c.f. definition in section 2.1.2) into separate partitions (to avoid disk contention) [LOU91]. A common problem with filesystems is *fragmentation*. With time, as filesystems become full, portions of files tend to be scattered on the disk. When a disk becomes very fragmented, performance suffers because files cannot be read (nor written) sequentially anymore. So fragmentation should be controlled and kept to a minimum [GCO96 and LOU91].

---

[6] Every change to the database is written to a redo log. If the database crashes before committing

The alternative to filesystems is to use raw devices (no filesystem) or block devices (data stored in blocks). Depending on the types of operations predominating in the system application(s), the appropriate disk layout should be used in order to maximize performance.

In order to achieve faster data access to large blocks of data on disk, some common disk technologies are available. Disk *striping* involves dividing a contiguous portion of data and spreading it over many separate disks in order to permit to multiple processes to access the sequence of data in parallel. Consequently disk contention is reduced and performance is improved. However, striping will only help when several random reads of the striped data actually occur; it does not affect the performance of small random reads [ICCM00].

Mirroring is another technique used mainly for data security, in case a disk crashes. It consists in writing the data twice, to two separate disks every time a write operation occurs. With mirroring, disk read operations are faster than write operations since the data written has to be written twice. So for a system performing mostly read operations, it is good to have mirroring. However, if the system is mostly performing write operations, mirroring will slow down the writing process.

RAID (Redundant Arrays of Inexpensive/Independent Disks) is a logical way of combining disks, offering advantages in their failure recovery features. It is a group of disks that appear to the system as one disk or as many virtual disks. RAID uses part of the storage capacity to store duplicate data. This technology can be implemented in hardware or in software (with Sun's Volume Manager, for example), and exists in several types and levels and usually implements striping and/or mirroring [CPS98, SES99, and SVM98].

---

all the ongoing changes, the redo logs are used to restore the database to a consistent state.

To detect I/O problems, or to monitor I/O, many tools are available (see Chapter 3). Disk I/O activity can be observed and patterns can be determined. It is important to note the number of reads and writes per second and per disk or partition to find out whether there is an overloaded disk. It is possible to make the correspondence between active disks and active processes, or applications more generally. By doing this mapping, it is possible to figure out what is happening on the system and which disks are more loaded. Disk usage can also be checked and disk space can be managed.

To recapitulate, the following strategies should be included – among others – when working on improving disk I/O performance [OST96, SES99, GCO96, and CPS98]:

- Watching disk capacity, always keeping 10% free disk space, will prevent filesystems from being full. When a disk does become full, the filesystems should be backed up and restored to avoid disk fragmentation. Adding disks also helps.
- Removing accumulated core dump files, checkpoint files created by editors, and other useless data keeps the disk(s) from getting full quickly.
- Configuring the disks in such a way to balance the disk load fairly over all the disks and controllers reduces disk contention and performance problems that are due to poor disk management.
- To reduce the activity on an overloaded disk, some of its most accessed files should be moved to a less active disk in order to achieve a close amount of I/O on each disk in the system
- On an Oracle system, data files and redo log files should be separated to prevent disk contention problems. Striping data files on separate disks can also reduce contention.
- Adding more memory reduces paging and swapping, and consequently disk activity.

- Other aspects that affect disk I/O performance are the speed of the disk controller/interface (SCSI, IDE/EIDE, Fiber Channel, etc.) and the size of the disk cache (the bigger it is, the more data it can hold and thus it avoids getting the data from disk, which is much slower).

2.1.4 The Network

Today, as the trend is to move from mainframes to new downsized (smaller) computers, the vast majority of systems use a *client/server* network architecture because of the significant benefits it presents. In such a configuration, the client (or client*s*, since there are typically *many* clients) makes a service request to the server (usually *one* machine), which fulfills that request. The server coordinates the client's access to its resources and data (databases, for example). It does this by having a daemon –which is a server program– permanently running on it, waiting for and handling requests coming from the client. On a Web server, for example, a daemon called *HTTPD* (Hyper-Text Transfer Protocol Daemon) continuously runs (by default on port 80) and serves Web clients. In an Oracle system, the *listener* daemon called *tnslsnr* runs continuously (by default on port 1521) waiting to service a client connection. Indeed, such server programs are said to *listen* on a port. The end-users always access the server through a 'client' application running on the client machine. The client machine from where the request was initiated incurs most application overhead, and does not affect the other clients. To run applications in a client/server environment, the communication between the client and the server must be made over the network.

For a communication to be established between the two parties, most often a connection must be opened first. This is done most often using TCP (Transmission Control Protocol) because of its reliability. When a machine requests opening a connection (outgoing call) to another machine, it is called an *active open*. Conversely, when a request to open a connection is made to that machine (incoming call) from another one, a *passive open* has to be made. It is

also possible –although unreliable– to send and receive information via the network using UDP (User Datagram Protocol) [WIS00 and CPS98].

The client/server communication over the network is done through *packets*, which are small units of data (typically 64 to 1500 bytes) that are transmitted over networks. Each client/server transaction requires sending a certain number of packets. The network *bandwidth* is a measure of the maximum number of bits per second that a network can transmit, i.e. its capacity.

As applications are becoming more and more complex, networks must ensure fast and reliable service, as well as high bandwidth. Many network technologies exist, including Ethernet, FDDI (Fiber Data Distributed Interface), and ATM (Asynchronous Transfer Mode). For an in-depth overview of network technologies, see [SES99 and CPS98].

Ethernet is the most popular network technology. It is simple and inexpensive to deploy, and is suitable for most moderately large applications [SES99]. However, it is collision-based, as only a single device can transmit data at a time [SES99]. When more than one device transmit simultaneously, the packets collide, generating garbled data, and the packets have to be retransmitted, consequently causing overhead and congestion. A common Ethernet network is usually considered saturated when the network traffic goes over 35% of the total bandwidth [SES99].

A router is a device connected to two or more computers, that decides to which adjacent network point a packet is to be forwarded next (on the way to its destination), based on the current state and availability of the routes; and a switch is a multi-port device for connecting networks and forwarding packets, often comprising a router [WIS00]. Breaking down the network into smaller pieces by using routers and switches generally minimizes collisions (and therefore improves

performance), because the traffic is reduced on each (smaller) portion of the network [LOU91 and SES99].

When a large number of packets arrive simultaneously, the receiving system gets overloaded and starts dropping some (or many) of them because it cannot handle them all. Dropped packets eventually have to be replaced, consequently adding overhead and load on the network. Moreover, dropping packets often indicate the existence of other problems, such as high CPU load that is stopping the network software from responding fast enough [LOU91].

Device failures on the network can often be the cause of data corruption. Because data integrity is essential, it is important to make sure that the data transmitted and received is free from errors. Network protocols often have error-correcting and recovery procedures incorporated. However, sometimes, these procedures can generate overhead that can cause significant traffic and delays [ICCM00].

Many tools are available to monitor the network and are discussed in Chapter 3. It is possible to obtain network traffic patterns and capture packets, determine if a system is dropping packets and at which rate, check the network collision rates, etc. It is also possible to monitor NFS activity. In fact, when NFS is running on a server, NFS requests are very frequent and significantly affect network traffic, and network performance as a consequence.

Network bottlenecks generally occur due to an overloaded network. Since client/server configurations require that data be sent back and forth over the network, it is best to try to reduce the number of packets to be transmitted across the network by tuning the application(s) (program code) on the system. This will prevent network contention problems and improve overall system performance [OST96].

There are also other practices to improve and optimize performance [GCO96, ICCM00, LOU91, and SES99]:

- Although the *number* of packets transmitted influence network performance the most, the *size* of each packet can also affect it. When the network protocol permits it, enlarging the packet size might help.
- Compressing data is another way of reducing the traffic on the network. However, it needs processing power, so should not be performed when there is a CPU bottleneck.
- Minimum transfers over the network can be achieved by always sending *exclusively* what is needed.
- Whenever possible, long-running jobs should stay at the server end.
- Each system on the network should be fast enough to keep up with the network traffic.
- There should be enough network bandwidth for the needs of the applications. If bandwidth is too low, the transfers get very slow.
- The client(s) and server should not be physically too far from each other, since transmission latency/delay also depends on the distance to be traversed by the packets.
- To solve data integrity problems, it is probably best to identify the faulty device and replace it.

2.2 Measuring Performance

Performance analysis involves the *measurement*, the *interpretation*, and the *communication* (or *conveyance*) of a computer's performance [LIL00]. Several fundamental parameter values have to be measured or, in some cases, calculated. Also, many measurement techniques exist, and one has to find the appropriate method to use on the system being analyzed. Measuring Performance is not an easy task and different people tend to disagree on the way it should be done [LIL00]. Performance analysis is often referred to as an *art* [JAI91 and LIL00].

When system performance is poor (typically characterized by a slowdown), it is usually not obvious to find where the problem is coming from. Instead of applying a tuning recipe to solve the problem, it is far more useful to first understand the foundations of performance [ICCM00] and then to apply them. To grasp any aspect of a computer's performance, it is important to first determine and understand *what* has to be measured.

The next subsections describe how to measure performance. In section 2.2.1, we identify and describe the most common –and important!– performance parameters (or performance metrics). Section 2.2.2 introduces several widely used methods to measure performance, and section 2.2.3 focuses on the ways to report the results of the measurements.

2.2.1 Performance Metrics

When measuring performance, we are often interested in timing the *duration* of an action happening on the system, in *counting* the number of times a certain event occurs, and in determining the *size* of some parameter [LIL00]. For example, some basic characteristics of interest would be to measure the time it takes for a process to terminate, to count the number of page faults, or to measure the size of the free memory at some point in time.

Most often, performance is measured in normalized metrics to facilitate comparisons between different machines. All those metrics embed a form or another of the basic characteristics (i.e. duration, count, and size). The most common and fundamental performance metrics are *throughput*, *queue length*, *response time*, and *utilization* [CPS98].

Throughput is a measure of the amount of work performed in a given amount (unit) of time. It is a rate and is measured in 'quantity'/sec, where 'quantity' depends on the device being analyzed ('quantity' is most often bytes or

megabytes). In database systems, for example, the throughput of the system is measured in TPS (Transactions per second); and for networks, the throughput is measured in packets/sec. For CPUs, the throughput can be measured in MIPS (Millions of instructions per second) or MFLOPS (Millions of floating-point operations per second), but these metrics are notoriously unreliable. The throughput of a disk can be reported by tools such as `iostat` (see section 3.1) by adding the rates of read and write operations. As the load on a system increases, the throughput increases too (as they are directly proportional) until it reaches a maximum. After this point, the throughput stops increasing or even starts decreasing because the workload is too great. To calculate the throughput of a system, the number of jobs or transactions completed is divided by the amount of time it took to complete them. So, for the throughput to be easily calculated, it is essential that the workload be clearly defined (see section 2.1.1).

Queue length is the number of processes (or requests) waiting – in a queue – for service. It can be calculated as follows [CPS98]:
Queue length = Throughput x Response time.
Many different queuing models exist and they provide important information for understanding and predicting system performance in some situations. Queuing theory is a very vast topic and is therefore not discussed here; for more information, see [JAI91, GEL00, and LIL00].

Response time is defined in two ways, depending on the authors. It can be (1) a measure of the time between the submission of a command and the reception of *some* response, i.e. the amount of time needed to consume a fixed portion of CPU time (usual definition), or (2) the time between the submission of a command and the completion of the request with a result. It is generally measured in milliseconds (*m*sec). When a system is overloaded, the response time increases, that's why it is sometimes referred to as "a key system-level performance metric" [GUN98]. The response time can be calculated as [CPS98]:
Total Response time = Total Queue length / Throughput,

or as [OST96]: Response time = Service time + wait time.

Service time is defined to be the time it takes to process a request once it has reached the front of the queue. So when the queue is empty (at low utilization levels), service time can be referred equivalently to *response time*, since they are equal. When monitoring I/O, the time to complete a physical I/O should not exceed 100 *m*sec. Service time can be calculated as follows [CPS98]:
Service time = Utilization / Throughput.
Thus, Disk Service time = Disk Utilization / Throughput.
In `iostat`'s output (see section 3.1), the response time (not the service time) of the disk is displayed under `svc_t` [CPS98].

To improve response time and throughput, resource bottlenecks must be avoided. This way, processes will complete faster because the resources they need are available.

The *Utilization* of a resource is a measure of the proportion of time a resource is busy doing work over a given period. It is usually reported as a percentage of *busy time* over the total time of the interval. *Idle time* refers to the period during which a resource is not doing any work. When talking about utilization in general, we usually refer to CPU utilization, where
CPU utilization = user time + system time (see section 2.1.1).
Disk utilization is also a common measure; when disk utilization is high, this often means that the disk is overloaded (it is common for the disk to be the initial bottleneck of the system). It is important to try to balance as much as possible the load in the system so that no resource is utilized more than others. Redistributing the load also improves response time and throughput, since waiting time to use a shared device decreases.

There are also some other common and useful metrics that can be used for performance monitoring; they include:

Think time is the amount of time the user of the system waits while 'thinking', i.e. not doing any work. Think time affects throughput and utilization, as they decrease when it increases.

Bandwidth refers to the speed of data that is transmitted over a medium, and is usually expressed in bits/sec. Bandwidth should not be confused with throughput. Bandwidth is typically the peak or the maximum speed possible (without overheads). The bandwidth of a network is a 'maximum throughput' measure of the number of bits that can be transmitted over the network per second.

Collision rate is the number of collisions that occurred during a period. High collision rates often are a sign of a network bottleneck. This value can be calculated as follows [DUN98]:
Collision rate = number of collisions / sum of input and output packets.

System capacity is a measure of the processing power of the whole system. It includes the CPU(s) speed, the disk I/O transfer rates, the network transmission rate, etc.

Benchmarks (discussed in section 2.2.2) are also used as performance metrics, but are generally very specific to the type of system being analyzed.

2.2.2 Measurement Techniques

In this section, some fundamental measurement concepts are described, showing the advantages and weaknesses of different techniques commonly used to measure performance.

An important thing to be aware of before measuring any system's performance is Heisenberg's Principle: "one cannot measure something without also affecting it in some way" [CPS98]. Indeed, the perturbation caused by a measurement

strategy is a major concern [LIL00]. As the measurement tools themselves need to use the system resources, they indirectly perturb the system and thus affect the measured data.

The basic measurement strategies for a live system are *event-driven* measurements, *tracing*, *sampling*, and *indirect* measurements [LIL00].

In an *event-driven* measurement strategy, performance metrics are calculated based on information recorded every time a preselected event (or events) occurs, such as an I/O operation or a page fault for example. The simplest way it can work is by having a counter count the number of occurrences of the event(s). An advantage of this technique is that it only creates overhead when the event or events of interest occur. However, when recording the information of high-frequency events, this introduces much overhead and perturbation, possibly leading to altered results. Therefore event-driven strategies are usually best suited for low-frequency events.

In a *tracing* strategy, upon the occurrence of a specific event, not only is the information about the occurrence of the event recorded, but also the record keeps track of information about the state of the system at that moment to uniquely identify every event that has occurred. As event-driven strategies only provide high-level information about the system being monitored, tracing programs on the other hand provides the most detailed system behavior information. The drawback of this strategy, however, is that it consumes large amounts of resources (because additional information is recorded) and thus tends to be the method causing the greatest perturbation on the system being analyzed.

Sampling strategies are most commonly used for analyzing system performance (most of the tools described and used in Part II of this thesis are sampling tools). When sampling, metrics of interest are determined by recording system information at fixed time intervals. These intervals are independent of the number

of events occurring on the system. In that sense, a sampling strategy is usually referred to be a statistical one, since it will miss some of the events that occurred, and will not provide the exact same data each time it is used on the same system. However, the statistical summary of a sampling-based experiment will usually show a consistent behavior of the system. The difficulty in such a method lies in choosing the appropriate interval for sampling. While it is important to have a sufficient number of samples to get reliable information, increasing the number of samples (or sampling for a longer period of time) will result in an increase in the perturbation on the system. But since the overhead is independent of the number of times any event occurs, the perturbation can be somewhat controlled by the experimenter as it is up to him/her to decide of the sampling interval. Different sampling tools usually require different sampling intervals (see section 3.1), and for each tool some trade-offs have to be made (often we choose to have insufficient data rather than having unreliable data). Broadly speaking, sampling is one of the methods that perturbs analyzed systems the least.

In an *indirect* measurement strategy, the metric of interest is not accessible and is thus measured indirectly through another metric. The appropriate measurement strategy is usually developed by the experimenter, so its effectiveness and its perturbation rely on him/her.

Another widely used method in evaluating performance of computer systems is *benchmarking*. A benchmark is a test program that assesses the performance of a system on a defined set of tasks. Benchmarks are mostly used to predict the performance of machines that are not yet in production, but they can also be run – with caution– on live systems. Benchmarks often run very specific program to determine the values of particular parameters, such as computing time of a specific mix of integer and floating-point operations on a system. Several types of benchmarks exist and some have become standards which are very widely used, such as SPEC (Standard Performance Evaluation Corporation) benchmarks [SPEC] or TPC (Transaction Processing Performance Council) benchmarks

[TPCW], which come in many sorts and flavors, each destined to be run on a particular type of system. The problem with these benchmarks is that they are never exactly characteristic of the application that runs (or is planned to be run) on a specific system. Benchmarks should be produced by the author of the application, since he/she is the only one who knows the internal complexities of the application. Benchmarking was not used in the performance analysis in Part II, mainly because no appropriate benchmark was found. For further details about benchmark programs and strategies, refer to [LIL00 and PRI89].

Measurements of real systems do not allow much flexibility. It is often desirable to see how the system's behavior changes as certain parameters are varied (for example, deliberately driving the system into a saturated state). However, this is very difficult –if not impossible– to do on live systems because it perturbs the normal behavior of the system (often in unpredictable ways) and can affect the end-users.

Apart from measurement techniques, there are 2 important techniques used when dealing with performance: *modeling* and *simulation* [CER98 and LIL00].

Modeling (also known as analytical modeling) is a fast method mostly used to predict the performance of new hardware on the system. Using measurements, a mathematical description –called a model– of the system is constructed and used to make predictions. The problem with this method is that is has lower validity because of the assumptions, generalizations, and simplifications it uses.

Simulation is a flexible technique commonly used to test software using a generated workload. The results are compared to the expected changes in the system and analyzed. The primary limitation of this method, when doing performance analysis, is that it is practically impossible to model every little detail of the real system being analyzed. For more details about simulation and techniques to avoid simulation mistakes see [JAI91 and LIL00].

2.2.3 Reporting Measurements

After the measurements have been made and the metrics have been collected, the performance analyst has to present the results and make sure they can be clearly communicated. Post-processing of measured data is not an easy task and should therefore be performed carefully [CPS98]. Some authors even refer specifically to data presentation as an art [JAI91], as improper presentation of results can damage a report because the conveyance of results is an essential component in performance analysis [CER98].

The collected data can be simply averaged and tabulated or plotted into a graph. There are three basic forms for reporting statistics [DUN98]: (1) using *snapshots* –showing measured values at a specific point in time, (2) using *histograms* or *line/pie/bar charts* –displaying the distribution of a set of measured values at fixed time intervals, and (3) using *summaries* –to see the total values of different variables.

Besides the *average* (a.k.a. *sample mean*), other important values are also commonly calculated when doing sampling. The most usual (and useful) one is the *variance* or *standard deviation*. Indeed, the average reflects the *mean* value of a variable or parameter measured at different points in time, whereas the variance (or standard deviation, as both basically reflect the same thing) measures the *dispersion* of the values in a distribution or collection of values (i.e. the variability within a distribution). The standard deviation is very important because it shows how far apart corresponding values are in different samples and thus reflects whether the system behaves in a consistent manner over time. Other types of means are thoroughly discussed in [JAI91 and LIL00].

The formulae used to compute the sample mean ($\bar{x}$) and the standard deviation (s) of a sample of n measurements are:

$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$, where the x_i are the individual measured values.

$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$, where $\bar{x}$ is the sample mean.

It is important to have good graphs that show all the aspects of the system's behavior. Enough data about each resource should be collected with the different performance tools (see section 3.1) and there should be graphs for each important performance parameter (such as utilization or throughput, for example) showing its variability and trend over time, and also with respect to the workload. The analysis should determine whether the performance capabilities of all the system's resources are balanced. Comparison graphs should also be included to show how different parameters are influenced by each other's behaviors. The graphs should be self-explanatory but an additional textual explanation is also useful for the analysis. Practical examples of all the different graphs are in sections 4.2 and 5.2. For more specific information on how to make graphs and what bad practices to avoid, see [JAI91].

Chapter 3

Performance Monitoring Tools

Performance monitoring is an essential task to control a system's performance. In order to isolate performance problems (and eventually tune overall system performance), a wide variety of tools is available. When run on a system, the tools provide (either directly or indirectly) performance metrics that report on the system's behavior.

The following sections describe several performance (sampling) tools in two parts. First, the most basic bundled Solaris/UNIX tools (usually called *utilities*) as well as some tools from the SE Toolkit [CPS98] are described, along with some hints for interpreting their output, and with an emphasis on tuning. The second part lists high-level descriptions of several other tools that are available from different vendors.

3.1 Common Monitoring Utilities

The Solaris 2.x operating system software provides many utilities to monitor the system's performance. In this section, we focus on the most common ones and mainly on those used in the performance analysis in Part II of this thesis.

The way these tools work is by having counters in the operating system that are incremented each time an event occurs, keeping track of various system activities while the computer is running. The tools then report the values of these counters.

In the following subsections, we overview the utilities used to monitor each resource in a system, along with a description of what to look for when analyzing performance. In addition to the most essential Solaris/UNIX utilities, some SE scripts[1] are also included. Screenshots are used to illustrate some of the tools; most of these screenshots were obtained by running the tools on the Aleph system.

3.1.1 Monitoring the CPU

When monitoring CPU performance, it is important to analyze all the aspects related to the processor(s). Many utilities exist to manage and isolate CPU performance problems.

- The first crude but useful command to run is **`uptime`**, which displays the current time, how long the system has been up (since last boot), and the load average over the last minute, the last 5 minutes, and the last 15 minutes.

```
% uptime
 12:04pm up 40 day(s), 15:32, 8 users, load average: 3.97, 3.59, 3.31
```

  The load average is the average number of jobs in the run queue. By running `uptime` on a regular basis and observing the load average, one can get a feeling for what is normal and what might be a sign of a problem [LOU91]. It is also useful to look for an increase or decrease by comparing the three values reported. Usually, a load average below 3 indicates a light workload; a load average greater than 5 indicates a heavy workload; and a load average of 10 or more is considered too high [LOU91]. `uptime` is a useful but insufficient tool, as it can sometimes lead to erroneous conclusions (c.f. section 2.1.1).

- **`sar`** (*s*ystem *a*ctivity *r*eporter) is a very useful utility that reports on system activity. It can operate in two modes: in the first mode, the data is extracted

[1] The SE Toolkit was developed by a Sun Microsystems engineer in order to monitor Solaris systems [see CPS98].

from `sadc`, the *s*ystem *a*ctivity *d*ata *c*ollector. In the second mode, discussed here, `sar` collects and reports data upon request (in real time). `sar` takes two arguments; the first argument specifies the number of samples to be collected and the second argument specifies the sampling interval length. In the following example, six samples are taken, at 5-second intervals[2]. With only these arguments (without options), `sar` reports on the percent of time the system is in user mode (`%usr`), in system mode (`%sys`), waiting for I/O (`%wio`), and idle (`%idle`).

```
% sar 5 6

SunOS aleph 5.6 Generic_105181-19 sun4u    03/13/01

15:58:17    %usr    %sys    %wio   %idle
15:58:22      22      13      10      55
15:58:27      20      10      16      54
15:58:32      20      11      27      42
15:58:37      17      10      19      55
15:58:42      10      10      72       8
15:58:47      16      13      67       4

Average       17      11      35      36
```

In the default `sar` output shown here, CPU utilization is reported: it is the sum of `%usr` and `%sys`. CPU utilization should not be constantly low, since this would mean that the system is oversized (c.f. section 2.1). A CPU utilization of 80 or 90% during peak loads means that the system is making good use of the processor(s). However, a continuously high CPU usage can be a sign of CPU contention or a sign of a hardware failure. In such a case, upgrading the CPU (or adding processors) can help improve overall performance [SES99].

On a system running normally, system usage should not get greater than user usage (c.f. section 2.1.1). Moreover, if system time is greater than 30%, the system is considered to be under heavy system CPU Utilization. When this is

[2] It is important not to run `sar` with an interval that is shorter than 5 seconds because in that case, the execution of the tool itself will perturb the system (and consequently alter the results).

associated with a high number of interrupts, a hardware problem or a bottleneck in an I/O device possibly exists [SES99].

The user CPU usage level also should not get too high (60 or 70%). When it does, processes running on the system should be examined and, in the case of CPU contention, an upgrade to a multiprocessor system will help [SES99].

When idle time is constantly less than 20%, this means that the overall system performance is probably suffering [CER98]. Again, conversely, if idle time is constantly too high (greater than 70%), it means that the system is not working efficiently and that it is oversized (unless the CPU is waiting for some external event to happen).

With options, one can get more diverse information from `sar`. For example, `sar -q` shows the occupancy of the process queues (used to determine whether processes are waiting for an available processor). Options in `sar` can also provide information about other resources in the system (memory and I/O; see sections 3.1.2 and 3.1.3).

- **`vmstat`** (virtual memory statistics) mainly reports on virtual memory statistics and is thus introduced in section 3.1.2. However, it also includes CPU-related information, which we describe here.

  `vmstat` reports on process information (i.e. process queues): the number of runnable processes (`r`), the number of blocked processes (`b`), and the number of swapped processes (`w`). It also reports on trap/interrupt rates per second for device interrupts (`in`), system calls (`sy`), and CPU context switches (`cs`). And like `sar`, the percentage of CPU time spent in user time (`us`), system time (`sy`), and in idle time (`id`) are included (see example in section 3.1.2).

  In `vmstat`'s output, one should look for long run queues: the number of runnable processes `r` should not exceed 4 times the number of CPUs,

otherwise processes would be waiting too long for a CPU time slice, and response time would increase [CPS98]. Also, one should look for processes blocked waiting for I/O or paging (`b`), because whenever there are blocked processes, all CPU idle time is treated as wait for I/O time. The faults rates reported by `vmstat` should also remain low, as traps and interrupts generate a lot of overhead.

With the `-s` option (summary mode), `vmstat` includes[3]: the total number of CPU context switches, device interrupts, traps, system calls, the total number of user, system, wait, and idle CPU time, and the percentage of time that the requested data was found in the name lookup cache (`cache hits`). The cache hit rate should stay at 80% or more, otherwise the system probably needs more memory or tuning [SOB99].

- **`mpstat`** (multiple processor statistics) is a very useful utility when analyzing a multiprocessor system. `mpstat` reports on per-processor usage and gives additional information. The following example shows `mpstat`'s output for a two-CPU machine with a sampling interval of 3 seconds and 2 samples. `mpstat` displays 16 columns of data concerning each processor: the CPU number (`CPU`), the number of minor faults (`minf`), the number of major faults (`mjf`), cross CPU calls (`xcal`), interrupts (`intr`), interrupts as threads (`ithr`), context switches (`csw`), involuntary context switches (`icsw`), thread migrations across processors (`migr`), spins acquiring kernel mutex locks (`smtx`), spins acquiring read/write locks (`srw`), system calls (`syscl`), and again: user time (`usr`), system time (`sys`), wait for I/O time (`wt`), and idle time (`idl`). The first sample outputted is generally ignored because it covers the time since the system's last boot:

[3] `vmstat -s` reports absolute system counters since the machine's last reboot.

```
% mpstat 3 2
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0  437   4  869     6    0  511    4   10    6    1   480   10   7  43  40
  1  197   7   61     8    0  674    6   10    6    2   545   11   6  43  40
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0  751  20 1975     9    1  685    7   22   12    0  1790    7  12  65  16
  1  275   5 3143    22    0 1602   16   23   16    0  2196   23   8  60   9
```

`mpstat` is commonly used to determine how the load is balanced across the CPUs. Also one of the key measures to look for in `mpstat`'s output is `smtx`, which reports the number of times the CPU could not obtain a mutex (mutual exclusion[4]) immediately. When `smtx` is greater than 200 for each CPU, system time usually goes up. To solve high levels of mutex contention in Solaris 2.6, mutexes suffering from contention should be identified [CPS98]. The number of context switches should also be watched, as a lot of context switches tend to consume too much CPU (c.f. section 2.1.1).

- **`cpus.se`** is another useful tool for multiprocessor systems. `cpus.se` is part of the SE Toolkit. It simply displays a list of the CPUs that are on the system along with their clock rates, checking that they are all online.

- **`nproc.se`** is another simple but very useful tool from the SE Toolkit. It displays the current number of processes that are running on the system. It is particularly interesting to run this tool with other utilities simultaneously and with the same interval, in order to compare the behavior of a particular resource under a certain load.

- **`ps`** is the typical command used to get information about the active processes on the system. It is generally run with options, because without options it only gives information about processes associated with the controlling terminal and user. With the `-ef` option, `ps` displays a full listing with data concerning every process currently running on the system: the process owner/user ID (`UID`), the process ID (`PID`), the parent process ID (`PPID`), the starting

[4] Mutual exclusion is needed by two (or more) processes when they need to access a resource exclusively (i.e. without sharing it) [STA95].

time/date of the process (STIME), the controlling terminal type (TTY), the cumulative execution time for the process, and the name of the process (CMD):

```
% ps -ef | head -12
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0   May 16 ?       0:01 sched
    root     1     0  0   May 16 ?       2:17 /etc/init -
    root     2     0  0   May 16 ?       0:59 pageout
    root     3     0  1   May 16 ?     810:21 fsflush
    root   739   678  0   May 16 ?       6:45 /opt/SUNWvmsa/../jre -
    root   510     1  0   May 16 ?       0:01 ipmon -s
    root    11     1  0   May 16 ?       0:04 vxconfigd -m boot
    m505 16650 15633  0 13:24:52 ?       0:02 /aleph1/../exe/pc_server_main
    m505 15505     1  0 07:03:01 ?       0:00 csh /aleph1/../proc/sc_server
    root   550     1  0   May 16 ?       0:00 /usr/sbin/inetd -s
    root   575     1  0   May 16 ?       0:02 /opt/sbin/sshd2
    root   566     1  0   May 16 ?       0:04 /usr/sbin/cron
```

Another widely used version of ps is the BSD version (/usr/ucb/ps). With the uaxw option, ps displays a comprehensive performance summary of all process-related information sorted by recent CPU usage. Thus it can be used to obtain a good top-ten listing of the busiest processes currently running on the system. The report of ps uaxw includes: the percentage of CPU time used by the process (%CPU), the percentage of the system's physical memory used by the process (%MEM), the total size of the process in virtual memory (SZ), the amount of physical memory in kB allocated to the process (RSS –the Resident Set Size), the status (S) with three possible values (O meaning on-CPU or running, R meaning runnable and waiting for a CPU to become free, and S meaning sleeping), the time the process started up (START), the total amount of CPU time used so far by the process (TIME):

```
% /usr/ucb/ps uaxw | head
USER      PID %CPU %MEM    SZ   RSS TT   S    START  TIME COMMAND
m505    18535  3.4  1.0 20339 240912 ?    O 06:33:14 119:19 /aleph1/../exe/rts32
m505      358  2.4  1.2 5386 447576 ?     S 12:23:04  3:07 /aleph1/../exe/www_server
m505     1214  2.3  0.1 17656 4144 ?      O   Jul 04 4154:35 /aleph1/../exe/rts32
oracle 18546  1.8  4.1 17029 616440 ?    O 06:33:14 51:48 oraclealeph1
m505     5973  1.7  1.0 4443 238528 ?     S 12:26:29  1:50 /aleph1/../exe/www_server
m505     5975  0.7  0.9 4179 235632 ?     S 12:26:29  1:28 /aleph1/../exe/www_server
m505    22114  0.4  0.7 3303 227256 ?     S 12:36:16  0:01 /aleph1/../exe/pc_server
m505     5389  0.4  1.0 4656 038544 ?     O 12:07:53  2:53 /aleph1/../exe/www_server
root        3  0.4  0.0     0     0 ?     S   Jun 08 2072:19 fsflush
```

So ps gives a global and accurate picture of what is happening on the system. When analyzing ps's output, one should pay particular attention to large

TIME fields corresponding with recent START fields, as this would be a characteristic of a process that is monopolizing the CPU. However, large values of %MEM and RSS are not necessarily an indication of problems; many programs require large amounts of memory to run.

- The **top** utility (not supplied with Solaris) is similar to ps but can also be used as a supplement to ps. It displays a list of the most resource intensive processes (fifteen by default) and updates itself dynamically every five seconds. Although very useful, top consumes a lot of the system's resources and should therefore be used with care (c.f. section 2.2.2).

- **iostat** is the typical I/O statistics reporting utility and is discussed in section 3.1.3. However, we just mention it here because, with some options (such as -xPnce), it also reports on CPU statistics including user time (us), system time (sy), wait for I/O time (wt), and idle time (id).

3.1.2 Monitoring Memory

Memory management is a complex area in systems performance. When running processes need more memory than is available on the machine, memory can become the system bottleneck [CER98]. Memory activity has to be understood in detail in order to solve the problems related to it.

- **vmstat** (virtual memory[5] statistics) is the primary, most used and most useful utility for obtaining memory information. It is also often used to get some disk and CPU activity information. When run without options, the memory statistics (virtual memory counters in kilobytes) reported include: available swap space in kB (swap) and free memory in kB (free), reclaimed pages in pages/sec (re), minor faults[6] in faults/sec (mf), page-ins (pi) and page-outs in

[5] c.f. section 2.1.2 for definition.
[6] A minor fault uses very little CPU time, as it is resolved without having to do a page-in.

kB/sec (`po`), pages freed in kB/sec (`fr`), memory deficit pages in kB (`de`), and scanned pages rate in pages/sec (`sr`). In the following example four samples are taken at 3-second intervals[7]. The first line covers the time since the system was last booted and is generally ignored; each one of the subsequent lines covers the period since the previous line (3 seconds in this case):

```
% vmstat 3 4
 procs     memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr s0 s1 s7 s8   in   sy   cs us sy id
 0 0 0   4720  2608  13 358 907 175 402 0 41 4 4  3  3 1273 1429  101  7  3 90
 0 1 0 4829792 231960 2 189 1029 0 0   0  0  3  6  0  0 1330 24707 4937 15 4 81
 0 0 0 4859872 247840 0 1028 437 2 2   0  0 12 23  0  0 1282 16879 2935 7  3 90
 0 1 0 4848920 240128 1 825 338 10 10 0  0 13 26  0  0 1293 13830 4430 13 3 85
```

The output of `vmstat` is very rich in information about memory. When monitoring paging activity and swapping, several counters should be looked at. The most important values are `po` (the number of page-outs) and `w` (the number of swapped-out processes), as they directly show whether there is paging and swapping activity on the system [LOU91 and CER98]. If `po` is large (significantly greater than zero) during a long period of time, system performance will be affected [LOU91]. Also when the deficit paging parameter (`de`) is non-zero, excessive swapping is performed because of a great memory shortage. Note that the `free` column is not an indicator of a lack of memory, because there may be a great amount of unused memory that has yet to be reclaimed by the page-stealing daemon [CPS98].

Page-in and swap-in activities are normal and should not be used to determine whether there is excessive paging and swapping [LOU91]. Determining whether a certain number of page-outs is excessive depends on the system being analyzed, how it reacts under different loads, and what performance is considered acceptable [LOU91]. However, in general, if page-outs are continuous, then more memory is needed [CER98]. Occasional page-outs on the contrary are part of a normal system behavior [LOU91]. The page-stealing

[7] However, usually, it is more useful to run `vmstat` with a relatively large interval in order to capture meaningful values for paging activity and for most other parameter values [DUN98].

daemon scanning rate (`sr`) is a key memory-shortage indicator when it is continually above 200 pages/second [CPS98]. For swapping also, if a system swaps out often, then there is need for more memory; otherwise, if there is only occasional swapping, then there is nothing to worry about [LOU91 and CER98]. It is also good to make sure that the *available*[8] swap space (`swap`) does not get down too low (32 MB or lower), because it would stop additional processes from starting [CER98].

With the `-s` (summary mode) option, `vmstat` reports on several values since the machine's last reboot[9]: the total number of swap-ins and swap-outs, the total number of page-ins and page-outs, the total number of reclaims, the total number of minor faults, etc. With the `-S` (capital S) option, `vmstat` reports on swapping activity (in kB/sec).

- The **`sar`** utility introduced in section 3.1.1 also reports some memory information, when used with options. `sar -r` shows the available swap in 512-byte blocks (`freeswap`) and the free memory in pages (`freemem`). Also `sar -w` is used to monitor swapping, and `sar -p` and `-g` are used to monitor paging activity.

- **`swap`** is another command used to monitor the swap space. `swap -s` displays summary information about swap space usage and availability, `swap -l` lists the status of each separate swap area, and `swap -a` and `swap -d` are used to add and delete a swap area respectively.

- **`vmmonitor.se`** is a program from the SE Toolkit that is based on `vmstat`. By default, it runs and samples data every 30 seconds, looking for little swap space (minimum default value: 400 kB) and low RAM identified by a high scanning rate of the page daemon (maximum default value: 200 pages/sec). It

[8] When available (not *free*) swap is used up, the system cannot allocate more memory [CPS98].

tries to sound an alarm and displays a time-stamped sample of current counters based on `vmstat`'s output. The environment variables used as thresholds for RAM (`VMMAXSCAN`) and swap (`VMMINSWAP`) can be set to an appropriate value depending on the system being monitored. In the following example, `VMMAXSCAN` is set to 300 pages/second and `VMMINSWAP` is set to 1GB:

```
% se vmmonitor.se
vmmonitor.se thresholds set at 1000000KB min swap space
and 300 pages/s max scan rate

vmmonitor.se problem detected: Thu Apr 12 05:46:01 2001
  procs        memory          page              faults              cpu
 r  b  w     swap    free  si  so  pi  po  sr    in    sy    cs   us sy wt id
 0  1 22 1482594   62378   0   0 1379 396 563  1451   704   807   0  2 27 71
Sustained scan rate over 300 pages/s - need more RAM
```

3.1.3 Monitoring Disk I/O

The disk subsystem is a common source of contention problems and often becomes the bottleneck of a system [LOU91 and CPS98]. As disk performance directly affects overall system performance, it is important to ensure throughput and storage efficiency. Disk activity can be monitored with several utilities, which we discuss here.

- **`iostat`** is the typical disk I/O activity reporter, although it also gives (limited) CPU information. Using the output of `iostat`, one can obtain several parameter values such as the transfer rate of each disk, the throughput, the queue length, etc. The option `-xPnce` shows physical disks in the `cXtXdXsX` format instead of the `sdXX` format (which is most usually displayed) and reports extended per-partition statistics, along with a summary of device errors. The report includes many statistics for each disk: the number of reads and writes per second (`r/s` and `w/s`), the number of kilobytes read and written per second (`kr/s` and `kw/s`), the average number of transactions

[9] Note that `vmstat -s` can report bizarre values when its internal counters overflow. This usually happens when the system is not rebooted for a long time.

waiting in the queue for service (`wait`), the average number of active transactions (`actv`), the average response time in the wait queue in *m*sec (`wsvc_t`), the average response time of active transactions in *m*sec (`asvc_t`), the percentage of time transactions are waiting for service in the wait queue (`%w`), the percentage of time the disk is busy (`%b`), a report on the errors that have occurred, and finally the device names corresponding to each disk. In general, the first sample displayed is ignored, as it covers the total time since the last boot; each subsequent sample covers the time since the prior interval only. In the following example, two reports (samples) of `iostat -xPnce` are run at 3-second intervals:

```
% iostat -xPnce 3 2
        cpu
 us sy wt id
  9  5 43 43
                                  extended device statistics ---- errors ---
  r/s  w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b s/w h/w trn tot device
  0.0  0.0    0.0    0.0  0.0  0.0    0.0   10.0   0   0   0   0   0   0 c5t0d0s0
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0   0   0   0   0 c5t0d0s2
  1.0  2.4    9.4   26.6  0.0  0.1    7.1   27.3   0   2   0   0   0   0 c5t0d0s3
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    6.4   0   0   0   0   0   0 c5t0d0s7

        cpu
 us sy wt id
 18 13 57 12
                                  extended device statistics ---- errors ---
  r/s  w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b s/w h/w trn tot device
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0   0   0   0   0 c5t0d0s0
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0   0   0   0   0 c5t0d0s2
  0.2  0.2    1.8    1.8  0.0  0.0    0.0    7.5   0   0   0   0   0   0 c5t0d0s3
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0   0   0   0   0 c5t0d0s7
```

From these numbers, one can easily calculate several important metrics such as the throughput (= `r/s` + `w/s`), total queue length (= `wait` + `actv`), total response time (= total queue length / throughput), and disk utilization (`%b`).

When watching `iostat`'s output, one should look for disks with high utilization (`r/s`, `w/s`, `kr/s`, and `kw/s`), which correspond with high CPU waiting time (equivalent to `%wio` in `sar`) meaning that there is a disk bottleneck. The average response time should stay under 10 *m*s. Also, the average response times (`asvc_t`) should be low (usually 30 *m*s or less, when the sampling interval is short) for disks that are less than 5% busy (`%b`) [CPS98]. Note however that disks with UFS filesystems might show high

`asvc_t` values that correspond with very low `%b` values (meaning that the disks are idle). This is due to the frequent updates of the filesystem flushing process (fsflush) [CPS98].

- **`df`** is a simple utility that reports on disk space. It is usually used with the `-k` option and displays the amount of available, free, and used space for each filesystem (including remote –NFS– filesystems). It also shows where each filesystem is mounted[10]. `df` is a useful tool for monitoring disk capacity (making sure there is always 10% free space) and disk usage, and can be a good starting point when solving disk contention problems.

- **`sar`** (introduced in section 3.1.1) also reports some I/O information. High `%wio` values mean that there is a lot of time spent waiting (blocked) for I/O operations to complete. This indicates that there is a slow disk, so throughput should be increased or I/O bus capacity should be added or rearranged.

  With the `-d` option, `sar` shows the activity for every block device (such as disk or tape drives). It reports on each device's busy time, utilization, service time, etc.

- **`vmstat`** (described in section 3.1.2) can also be used to get some disk information. Processes blocked waiting for I/O (`b`) in `vmstat`'s output indicate a possible disk bottleneck. Tuning is necessary if this number (`b`) is close to (or greater than) the number of processes in the run queue (`r`). `vmstat` also reports the number of disk operations, for up to four disks (by default), but this information is usually insufficient.

- **`iomonitor.se`** is a program from the SE Toolkit that works like `iostat` but does not display anything until it sees a slow disk. The thresholds for disks busy time and average service time are set by the user, using the environment

[10] Correct filesystem layout is an important factor in avoiding I/O contention problems.

variables `IOMINBUSY` and `IOSLOW`. By default, `IOMINBUSY` is set to 20 (i.e. under 20% busy time is not a problem) and `IOSLOW` is set to 50 (i.e. 50 *m*s or more of average service time is slow). The default sampling interval of `iomonitor.se` is 30 seconds (to avoid false alarms). When it sees a slow disk, `iomonitor.se` tries to sound an alarm and outputs a time-stamped sample of the responsible disk's statistics, using a format similar to `iostat`.

3.1.4 Monitoring the Network

With client/server configurations, networks are gaining more and more importance, as they can become the limiting factor in overall system performance [LOU91]. Thus, network performance monitoring is an essential task to avoid not only network performance problems but also overall system performance problems (c.f. section 2.1.4). Network problems are often difficult to understand, but with the existing monitoring tools, it easier to identify the cause of a problem.

- **`netstat`** is the most common utility for network statistics reporting. The following example shows `netstat` with the `-i` option (which is used to summarize all the interfaces, when there is more than one interface), and specifies a polling interval of 3 seconds; there is no way to tell `netstat` how many reports to generate. The default interface here is `hme0`; in order to specify another interface, the `-I` option is used. Each one of the columns displayed represents network packets over a sampling interval (the interval since the last line). The `input/output packets` are the number of network packets received/sent respectively; the `input/output errs` are the number of errors that have occurred while receiving/sending `packets`, respectively. `colls` is the number of packets that were blocked by network collisions (for a full-duplex Ethernet card and hub, this number will always be 0 on Solaris systems [SOB99]).

```
% netstat -i 3
     input   hme0      output           input  (Total)     output
packets errs  packets errs  colls  packets errs  packets errs  colls
75364653 2     486083593 0     0     98596201 2     509315141 0     0
88       0     99       0     0     131      0     142      0     0
102      0     123      0     0     227      0     248      0     0
52       0     72       0     0     114      0     134      0     0
61       0     54       0     0     120      0     113      0     0
97       0     117      0     0     240      0     260      0     0
87       0     112      0     0     162      0     187      0     0
```

The 6th through 10th columns contain the same information as the 1st through 5th, except they cover all packets on all network interfaces, and not just the default interface.

To display the per-protocol (TCP, IP, UDP, ICMP, etc.) statistics, the `-s` option is used. And to limit the output to just one protocol, the `-P` option is used.

In `netstat`'s output, it is important to watch the input and output errors columns (`input errs` and `output errs`). The number of input errors should not be more than 1% of all incoming packets; otherwise this could indicate packet corruption problems or insufficient network buffers. Also, if the number of output errors is more than 1% of all outgoing packets, this means that there is a hardware problem or that the maximum collision retry count has been exceeded [CER98]. Some authors even put these threshold ratios much lower (at 0.025%) [LOU91 and SES99].

Collisions are a good indicator of the network load, since they increase as the network is more heavily used [LOU91]. The maximum number of collisions that should be tolerated is 10% of the total number of output packets [LOU91 and SES99]. When this ratio (`colls / output packets`) is greater than 10%, this means that the network is overloaded, so the traffic must be reduced (by dividing the network into two or more subnetworks, for example).

- **`netmonitor.se`** is a program from the SE Toolkit that waits for a slow network, and then outputs some statistics using a format similar to `netstat`. Its default sampling interval is 30 seconds long, and it uses the following environment variables to set the thresholds: `NETACTIVE` (minimum packets/sec to worry about; default is 100) and `NETMAXCOLL` (maximum collision rate tolerated; default is 10).

- The **`nfsstat`** utility is used to get statistical information about the NFS (Network File System). When the `-s` option is added, server statistics are reported. When the `-c` option is specified, client statistics are displayed. And with the `-m` option, `nfsstat` reports network statistics for each NFS-mounted filesystem.

- **`nfswatch`** is used to monitor an NFS server. `nfswatch` monitors all incoming network traffic to an NFS file server and divides it into several categories. The number and percentage of packets received in each category is displayed and continuously updated.

- **`ping`** is a simple command that is commonly used to check if a remote system is down. It does this by using the ICMP (Internet Control Message Protocol) to send messages to a remote host, and then waits for an answer to be echoed. If the host responds, it outputs a message saying that the host is alive, along with a measure of the amount of time it took the host to answer back. If the host does not respond within 20 seconds (default timeout value), `ping` reports "no answer from host"; the remote system is most probably down.

- **`traceroute`** is a very useful utility for tracing the path of network packets. It is only supplied with Solaris 7, but can be downloaded from the Internet for earlier versions of Solaris. `traceroute` traces the path/route of a packet from its source to its destination, showing all the hosts through which the packet went along with the round-trip time it took to get to each router. The

following example shows the output of `traceroute` from the local computer `nova.cs.mcgill.ca` to the remote machine `hell.cc.mcgill.ca`. After the warning, the IP address of the target is displayed, followed by the maximum number of hops that will be traced, and the size of the packet that will be used. Each subsequent line is numbered and shows the name and IP address of an intermediate destination host, as well as the time it takes a packet to make a round-trip to that destination and back (there are three numbers because `traceroute` sends three packets to each destination):

```
% traceroute hell.cc.mcgill.ca
traceroute: Warning: ckecksums disabled
traceroute to hell.cc.mcgill.ca (132.206.35.52), 30 hops max, 40 byte packets
 1  132.206.51.251 (132.206.51.251)  0.925 ms  0.947 ms  0.795 ms
 2  132.216.101.254 (132.216.101.254)  1.783 ms  1.216 ms  1.460 ms
 3  burnside-core-msfc.GW.McGill.CA (132.216.216.161) 1.694ms 6.486ms 1.390ms
 4  burnside1-msfc.GW.McGill.CA (132.216.216.42)  2.463 ms  1.686 ms 1.398 ms
 5  hell.CC.McGill.CA (132.206.35.52)  1.816 ms  2.511 ms  1.848 ms
```

The `traceroute` utility is very helpful when trying to identify a network bottleneck or to solve routing problems. When a host is unreachable, `traceroute` shows the path of the packet, where it stops, and what the round-trip delay is.

- **`spray`** is generally used to check whether a system is dropping network packets. It generates network traffic between two hosts and reports on the overall transfer rate and on the number of packets that were received and the number of packets that were dropped. If a lot of packets are being dropped (over 5%), this means that the receiving system is slower in receiving the packets (maybe because it is overloaded), or that the data is being corrupted on the way (less likely) [LOU91]. The `-c` option is used to specify the number of packets to send.

- **`snoop`** is used to capture and analyze network packets, usually in order to track client network activity. Captured packets can be displayed as they are received (i.e. in real-time), but this generates a lot of overhead. It is therefore

better to save the received data for later analysis. However, multiple-machine diagnosis using `snoop` can be very tedious [CER98].

3.1.5 General Monitoring Tools

The previous sections have described utilities that concentrate mainly on a single resource. Several other tools and techniques exist and are used either to monitor the whole system at once[11] (i.e. all the resources together), or are used to automatically launch a command or utility to log data.

- **`perfmeter`** is a utility that is only available in the OpenWindows environment. It displays system performance statistics for a particular host (local or remote) using strip charts (default display format). The display is refreshed every 2 seconds (default sampling interval), and provides summary information for the CPU, memory paging activity, network packets rate, disk I/O, etc. `perfmeter` has many options, such as logging data to a file for example.

- **`xload`** is a GUI-based tool that is part of Solaris (with X-Windows). It displays a periodically updated histogram of the system load average.

- **`xosview`** is an application originally developed for Linux. It is a graphical performance meter presented as a bar graph of the current system state. It displays the status of several system-based parameters, such as CPU usage (usr, nice, sys, free), memory usage (used & shared, buff, cache), swap space usage (used, free), paging activity (in, out, idle), disk activity (read, write, idle), interrupts, etc. Its appearance is fully configurable via command line or X resources.

[11] The `perfmeter`, `xload`, and `xosview` tools described here are all X-Windows applications designed to graphically display system performance measures.

- **cpg.se**, the *c*apacity *p*lanning *g*uide, is a program from the SE Toolkit that gathers information about all the system's resources. It checks paging and swapping, disk saturation, the directory-name lookup cache (DNLC) hit rate, the CPU, and the network interfaces, reporting any errors that have occurred. It has different modes of operation and can work in a continuous or sampling mode.

- **live_test.se** is also program from the SE Toolkit. It tests all aspects resources of the system and displays everything in `live_rules.se`, a set of predefined rules that set thresholds for each resource [see CPS98]. The rules have color codes defining states: White state (idle/inactive), Blue state (imbalance on different instances of the resource), Green state (no problems, active), Amber state (warning condition), Red state (overloaded or problem detected), and Black state (error condition indicating component or system failure). It provides a text-based system monitor and outputs information with 30-seconds intervals (by default).

- **virtual_adrian.se** is a script that works as if Adrian Cockcroft was actually monitoring the system [CPS98]. It reports on anything that does not look well tuned and also uses the SE rules. Parameter thresholds can be set at once using several environment variables in `/etc/rc2.d/S90va_monitor`. The program takes a look at the system every 30 seconds (by default) and only displays something when it finds a problem. Upon detection of a problem, a complaint is made and the data that caused the complaint is displayed. Also, immediate (tuning) changes can then automatically be made to some system variables.

- **cron** is used to start a process that executes commands at specified dates and times. It is particularly useful for regularly scheduled commands (sampling utilities for example). These are entered in `crontab` files using a specific format including the date, hour, and minute.

- **at** is a utility that is used to execute jobs at a later time (but using the same environment setup). It is usually used for commands that are to be executed only once. It is very useful when running sampling tools, as it can serve for example to start or stop them at a certain date and time.

- **System Accounting**: Many processes have very short life spans and thus cannot be seen with tools like `ps` (c.f. section 3.1.1), but these processes may be so frequent that they dominate the system load. The only way to catch them is to ask the system to keep a record of every process that has run, who ran it, what was it, when it started and ended, and how much resource it used. This can be done using the system accounting subsystem. System Accounting is available on Unix systems and can be enabled or disabled. It is a major tool for long-term reporting and logging of system data. The system accounting software is in fact a set of tools that can be used to report on performance information and/or build resource accounting systems. Accounting is turned on at system start-up time. Process accounting is handled by various programs, all of which write records in the master collection file `/var/adm/wtmp`. In addition, when a process terminates, various statistics about the process are written to `/var/adm/pacct`. Using accounting data, one can identify how often programs run, how much CPU time, disk I/O, and memory each program uses, and what the work patterns throughout the week look like. Although collecting accounting data always generates overhead, it is usually insignificant. However, accounting data consumes disk space.

3.2 Other/Commercial Monitoring Tools

Although the basic utilities described in section 3.1 are very commonly used on Sun systems, there are other (mostly commercial) tools that are also used mainly because of their extended functionality. Indeed, these commercial tools/products are generally easy to use (typically all have a GUI) and they facilitate the task of

monitoring performance, as they monitor and analyze all the aspects of a system at once. Also, a lot of them are platform independent (i.e. they can run on most major operating systems).

Many vendors provide several software products with a different service level. Some products perform a more detailed analysis than others, but they typically all help automate the process of performance management. The following is a brief overview of some commercial tools that only highlights the essential features each product; links to the respective vendors' Web sites are provided for further information. The list is organized in alphabetical order, according to the company name:

- **BGS *Best/1*** manages the performance and capacity of all UNIX systems and includes integrated support for several common database systems. Best/1 features visualization and prediction facilities, as well as a real-time monitor that sounds an alarm when a problem occurs. It is possible to configure Best/1 to continuously collect performance data on the system. BGS is at http://www.bgs.com.

- **BMC *PATROL*** is a collection of products with many features. Two most important features are performance management analysis and performance prediction. With performance management analysis, PATROL provides current and historic analysis of systems and applications. The results are graphed and automatically published on the intranet to facilitate the communication within an organization. With performance prediction and capacity management, PATROL provides advanced modeling and analysis of changes that would be implemented in the system's hardware. It also provides potential solutions to performance problems and uses prediction to try to prevent them. BMC is at http://www.bmc.com.

- **Compuware *EcoSYSTEMS*** is a product suite that provides a comprehensive application management. The suite is composed of many useful tools: *Application Expert* helps in profiling and predicting application performance; *Application Vantage* troubleshoots the performance of applications; *EcoTOOLS* manages application availability; *EcoSCOPE* analyzes networked application performance; *EcoPREDICTOR* predicts application and network performance; and *Interval Pro* monitors performance and manages failures of Windows applications. Compuware is at http://www.compuware.com.

- **Entrix Direct *GENSYS Monitor Suite*** monitors applications and devices across multiple platforms. It has a built-in knowledge of typical systems and is also customizable. GENSYS identifies, or even anticipates problems and rectifies them automatically. It can also send alerts using e-mail, fax, GSM, etc., and it is reliable and secure. Entrix Direct is at http://www.entrix-direct.com.

- **FORTEL *SightLine*** software provides real-time performance management for e-Businesses. SightLine provides a view of the service level of the e-Business in real-time and continually analyzes the relationships in the system looking for any threat. This way, it lets the organization identify service level problems before the customers experience poor performance. FORTEL is at http://www.fortel.com.

- **Fujitsu *Softek SANView*** is a product intended to manage Storage Area Networks (SAN). It monitors the performance of all the devices that are connected to the SAN and signals problems before they actually cause damage. Softek SANView can also locate all the devices on the SAN and draw a topology map showing the interoperability and connectivity of the different devices. Softek SANView is fully based on Java and thus is platform independent. Fujitsu Softek is at http://softek.fujitsu.com.

- **HP OpenView *GlancePlus Pak 2000*** is an integrated product for managing a system's availability and performance. GlancePlus Pak 2000 includes three products: *GlancePlus*, which provides real-time diagnostic capabilities; *VantagePoint Performance Agent*, with its historical data collection capabilities; and *Event and Availability Management*, which monitors the system looking for events affecting performance. HP OpenView is at http://www.openview.hp.com.

- **HyPerformix *Strategizer*** and ***SES/WorkBench*** are tools for predictive simulation modeling. Strategizer is a simulation modeling tool for performance analysis that locates potential problems, bottlenecks, and weaknesses before the system is put in production. SES/WorkBench is used to model complex systems with interacting components in a visual simulation environment. It evaluates the effects of each aspect of the modeling. HyPerformix is at http://www.hyperformix.com.

- **ISM (Information systems Manager) *PerfMan*** is a systems management and monitoring tool that can run on many different platforms (including Windows NT/2000, UNIX, OS/390, MVS/ESA). With PerfMan, it is possible to get metrics on hundreds of performance factors affecting the system, including long-term historical data. More information can be found at http://www.perfman.com.

- **Landmark *TMON*** exists is a wide variety of versions for a number of different operating systems *and* database systems. Depending on the system, TMON (meaning The MONitor) provides real-time software and hardware performance information, historical analysis of resource usage trends, quick problem isolation, database application utilization information, etc. Landmark is at http://www.landmark.com.

- **Metron *Athene*** is a software package for performance management on Windows NT, UNIX, and most mainframes. Athene provides performance analysis, capacity planning, database performance reporting, fast automated performance reporting, trend analysis, and analytical modeling. Athene captures, collects, and transfers data to a Performance Database. More information is available at http://www.metron.co.uk.

- **Quest Software *Spotlight*** is a tools for real-time diagnostics. It exists in several versions, each adapted for a particular environment (such as Web servers, Oracle databases/applications, SQL servers, etc.) and operating system. Spotlight works in real-time, displaying graphs of the server processes and flow of data in order to quickly identify congested areas and take appropriate corrective action. More information is at http://www.foglight.com/spotlight-portal.

- **Sun *SyMON*** is a very well known system management product of Sun Microsystems. SyMON (System MONitor) exists in many versions and offers monitoring capabilities for several Sun servers using a configurable CDE/Motif-based GUI. SyMON analyzes system performance in real-time and sends an alert when a problem occurs. It monitors the performance of the CPU, memory, disk, and network. SyMON can also display views of the system's configuration, allowing quick and detailed information access for administrators. It can also isolate potential problems or failed components, and always keeps system log files for future analysis. More information on SyMON is available at http://www.sun.com/products-n-solutions/hardware/docs/Software/System_Management_Products/SyMON/index.html. Note that the single node version of SyMON is available for free with Sun servers.

- **Sysload Software *Sysload*** is a product with several modules, each dedicated to a task in the performance management process. There is one module for monitoring resources usage that can produce reports giving details of the

behavior of the system with an accuracy ranging from one minute up to one year. Another module is dedicated to observing the behavior of the system (in real-time) and optimizing it. Another module uses artificial intelligence technology to quickly analyze and find diagnostics. Sysload permanently collects data and monitors main operating systems, databases, and main applications. Sysload Software is at http://www.sysload.com.

- **TeamQuest** performance software comprises four products: ***Alert***, ***On the Web***, ***View***, and ***Model***. The main feature of Alert is its multi-system monitoring capabilities, as it can monitor hundreds of different systems in a single display and with built-in assistance. On the Web is an automated report-publishing tool that handles the post-processing of the collected data: it publishes the results and delivers them on the Web. View is a comprehensive performance analysis tool. It can detect bottlenecks and report on the system performance of several different systems. Model is a tool that is used for predicting systems behavior using analytical modeling. TeamQuest is at http://www.teamquest.com.

- **Tivoli *Distributed Monitoring*** provides automated monitoring for distributed systems. With centralized information technology, monitoring parameters can be changed for hundreds of (remote) systems directly. It also includes views displays to help for capacity planning and performance trend analysis. The views are also used to predict performance bottlenecks. More Tivoli products can be found at http://www.tivoli.com/products.

- **Web Performance *Trainer 2*** is a tool designed to improve the performance of Web-based applications. Trainer 2 simulates multiple users to find performance bottlenecks, increase performance, or do capacity planning. It is a useful tool for finding out how many users a Web-based application can handle. Web Performance is at http://webperformanceinc.com.

Part II

Performance Analysis of two systems: ALEPH and BANNER

In Part I, we presented an overview of the theory of computer systems performance analysis. In this part, practical applications of performance analysis are carried out on two systems (in Chapter 4 and Chapter 5), using the technical background provided in Part I.

Chapter 4 introduces the library system ALEPH, followed by its performance analysis. Chapter 5 presents the finance system BANNER, along with a performance analysis of the main machines involved. ALEPH and BANNER applications are built similarly: both are a mix of C and Cobol programs, both have a GUI client and a Web user interface, and both use Oracle.

Although Oracle databases are a major component of both systems, the analysis conducted here does not include an Oracle performance analysis per se. However, the analysis covers system aspects that are directly related to the Oracle database and that consequently affect its performance (as an example, I/O performance analysis – and tuning – will result in better Oracle performance since Oracle depends heavily on I/O performance). The analysis is mainly based on – but not limited to – data gathered during the months of March and April 2001.

Chapter 4

Performance Analysis of ALEPH

4.1 ALEPH System Overview

McGill University Libraries have recently installed Ex Libris Inc.'s library management system ALEPH500™ (version 505.12.3). The installation was completed in summer 1999 and the system went live in May 2000. The ALEPH500 system enables users to access the libraries' massive catalogue and resources through a web-based interface. It also provides a PC graphical user interface to the librarians. ALEPH500 (internally called MUSE 2) replaced the ten-year old library management system Notis (MUSE 1) that was running on a CICS mainframe system.

4.1.1 System Hardware Configuration

All of the ALEPH500 components run on a single server called 'Aleph'. The system (server) model is Sun's Enterprise 5500. It has *nine* 400Mhz-CPUs (UltraSPARC-II) sharing 4GB of RAM, *twelve* mirrored disks (so in fact 12*2=24 disks[1]) with almost 100GB of data, and running Solaris 2.6. The disk management software is Sun Enterprise Volume Manager 2.6, and the filesystem type is UFS (Unix File System). The machine has one 100-Mbit full duplex Ethernet interface.

[1] Actually, the system sees more than 24 disks: 12 mirrored disks in the array (A5200), which become 24 because of the mirroring, which then are transformed into 48 disks because of the multipathing (there are 2 paths) + 4 system disks (for holding the operating system) = 52 disks appear in total.

4.1.2 The ALEPH Application

ALEPH500 is a library management system that provides services (1) via Web clients (browsers): for library users, such as professors or students (searching for books, registering, viewing list of items on loan, etc.), and (2) via PC-GUI clients running on MS Windows: for librarians/library staff (for daily library functions – such as cataloging books, doing checkouts, collecting fines, and so on; to run batch jobs, etc.) (see Figure 4-1).

There are 4 database categories, referred to as libraries (or types of records):

- BIBliographic library (mgu01, mgu30),
- AUThority library (mgu10, mgu11, mgu12, mgu13),
- ADMinistrative library (mgu50),
- HOLdings library (mgu60).

(Note: mgu is the name for customized McGill-ALEPH libraries)
Each type of library (BIB, AUT, ADM, and HOL) runs different daemons.

ALEPH's database is based on Oracle 7 RDBMS. Every ALEPH library has a separate root directory and each library directory contains information for administrating the library (configuration files, log files, etc.). Each library's database is implemented within Oracle: each library is implemented as a separate Oracle user, who owns a set of tables containing the data. So when logging in to Oracle as mgu13 (for example), all the tables for mgu13 become ready and accessible. The system actually uses these Oracle users (which have Oracle schemas) internally to do the work.

The ALEPH server requires the m505 (which is now m535) Unix user and Oracle users corresponding to each of the libraries (mgu01, mgu10, mgu11, etc.). ALEPH connects to the Oracle database through a dedicated Oracle user named aleph. Depending on the database user, different rights and permissions are

assigned. Of course, the connection between ALEPH servers/procedures and these Oracle users is transparent to ALEPH end users (using the Web or PC-GUI interfaces).

4.1.3 ALEPH's Intra-Communication

Most of the end users are external users (students, professors, etc.), so they connect to ALEPH via a Web browser. The connection is made through a socket at port number 4535 (or 4536, and so on, up to 4545) where the web server (process `www_server_main`) is running. ALEPH then sends a query to Oracle. The Oracle server replies back to ALEPH and ALEPH to the Web user.

Other users can be librarians or library staff. They connect to ALEPH for different purposes via a PC-GUI[2] client (there are 226 users with librarian accounts). The connection is made at port number 6535 where the PC server (process `pc_server_main`) is running. ALEPH then sends a query to the Oracle server, gets a reply, and returns an answer to the PC-GUI user.

The ALEPH client is really a collection of clients, each dedicated to a particular function: Acquisitions client, Admin client, Cash client, Cataloging client, Circulation client, ILL client, Items client, etc.). Every client has a set of directories (\Bin, \files, \Help, \Tab, etc.) with execution files containing various definitions. All clients have a `client.ini` file, which defines their look and behavior. As examples of clients: The OPAC client of ALEPH enables users to search the database for a record, to view holdings/location info, to place a request, etc. The Users client enables borrowers to review their registration, view list of items on loan, renew loans, etc.

---

[2] Librarians can even send batch jobs through a special web page by filling a form (at aleph.mcgill.ca4535/S/), so they do not need to know UNIX commands.

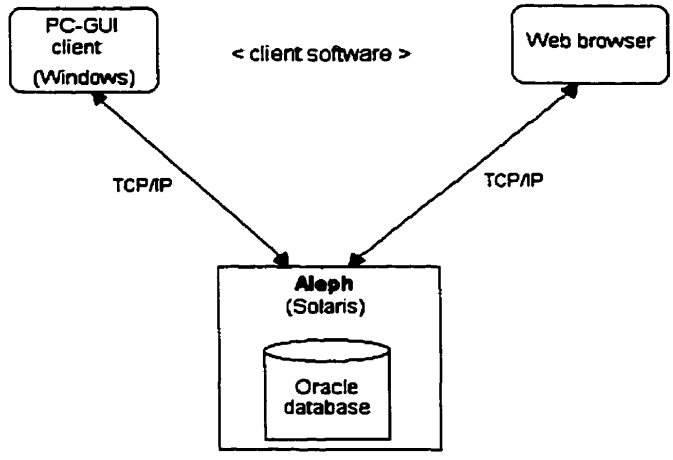

Figure 4-1. ALEPH System Hardware Architecture.

4.1.4 ALEPH Users

The ALEPH license specifies that the maximum number of Web users is 450 (simultaneous access) and the maximum number of PC-GUI users is 226. There is a log file on Aleph (`/home/lyne/stat-session`) keeping track of these numbers by counting the number of Web sessions and the number of PC-GUI sessions, both daily and every half-hour. These statistics are obtained using the ALEPH Utilities s-4-1 and s-4-2.

Other statistics are also being kept. Some of them are obtained directly from ALEPH utilities, whereas the others were extracted using Perl programs written by Lyne Thibault. The statistics, started since May 2000, include:

- The daily and monthly circulation statistics for all sub-libraries (Utility s-1-8);
- Daily user web response times or the number of requests that were serviced within acceptable time boundaries, i.e. within 2 seconds or less;
- The number of hits[3] (GETs) or web pages retrieved (daily, with the total for the month), as well as where the requests came from (McGill domain, McGill

[3] The user response times and the number of hits are not kept for the PC-GUI interface.

DAS, or other). These Apache log files are located in `/var/log/www/` (one file per month).

There are also other Web log files and PC log files in `/aleph1/a53_5/tmp/`, keeping track of the type, the origin, and the time of the transactions performed on the system.

4.1.5 System Use

The two main/primary processes running on the machine are:

- a Web server (ports 4535, and on) having the majority of use, and
- a PC server (port 6535), only used by a few hundred librarians.

There is also a database server "behind the scenes" to which both mentioned servers send queries. The database is a large[4] Oracle database with a web interface and a PC-GUI interface. More information on the processes running on Aleph is included in section 4.2.1.

4.2 Performance Study of Aleph

The analysis presented in this section is based on statistics gathered using most of the tools[5] described in section 3.1. These cover part of the month of February, all of March and all of April 2001.

February, March, and April are interesting months for the analysis of Aleph's performance, as the peak usage loads for the winter semester occur during that period (see Figures 4-2 and 4-3). Figure 4-2 shows the total (monthly) circulation for all McGill's sub-libraries over 13 months. The highest activity during the

[4] McGill University owns over 3 million volumes.
[5] The tools used for Aleph's analysis are listed in Appendix A.

winter semester is observed from February to April, which is the month that marks the end of the winter term. Figure 4-3 shows the ALEPH Web users activity per month, over a year. Again, the (winter) load is highest during February, March, and April.

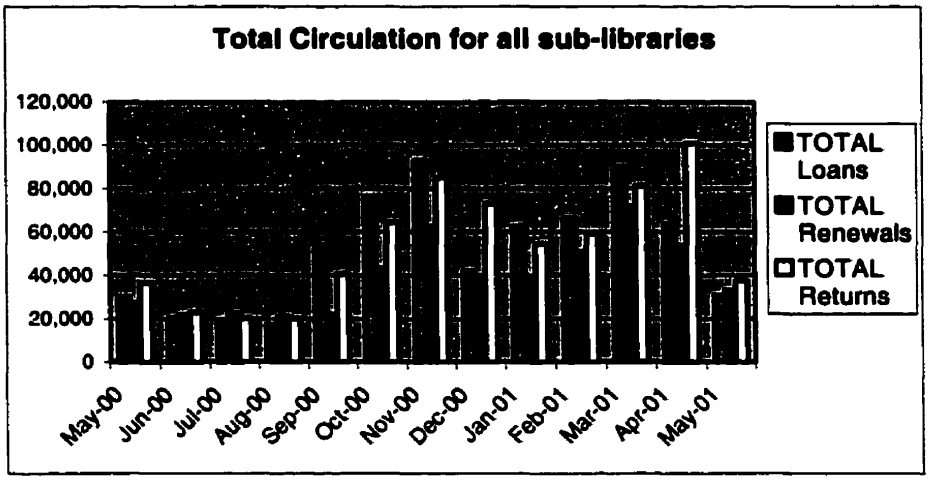

Figure 4-2. Total Circulation for all sub-libraries.

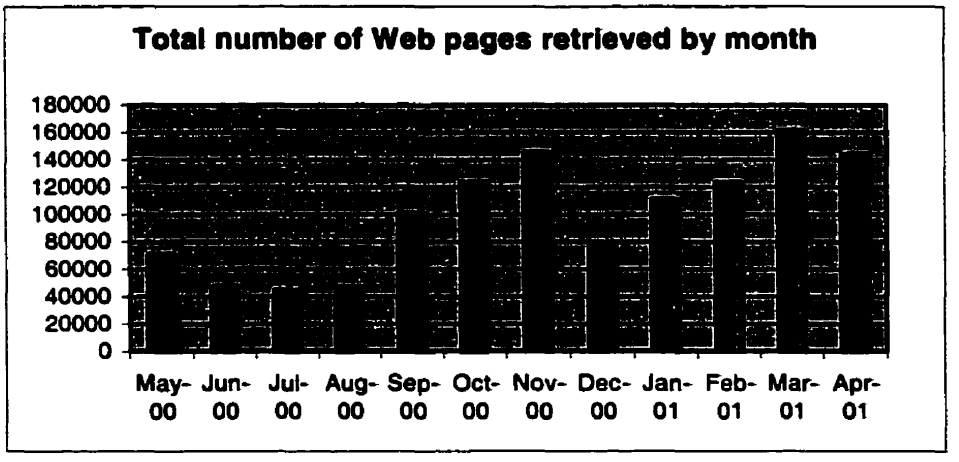

Figure 4-3. Total number of Web pages retrieved by month.

As mentioned in section 4.1.5, ALEPH has a Web server, a PC server and a database server running. In the following subsections, all the relevant aspects are

examined and are divided into four parts: the CPU (4.2.1), memory (4.2.2), disk I/O (4.2.3), and the network (4.2.4).

4.2.1 CPU Usage

Let us first get a general idea about overall system usage by looking at the load average reported in Figure 4-4 by the `uptime` utility (c.f. section 3.1.1). `uptime` was run with a 10-minute sampling interval during two weeks (last week of February and first week of March), and the average load for one day was computed and is shown in Figure 4-4. The standard deviation of the corresponding samples for each day was very low (rarely going over 1), and so the 24-hour load average shown in Figure 4-4 is characteristic of the load on Aleph every day.

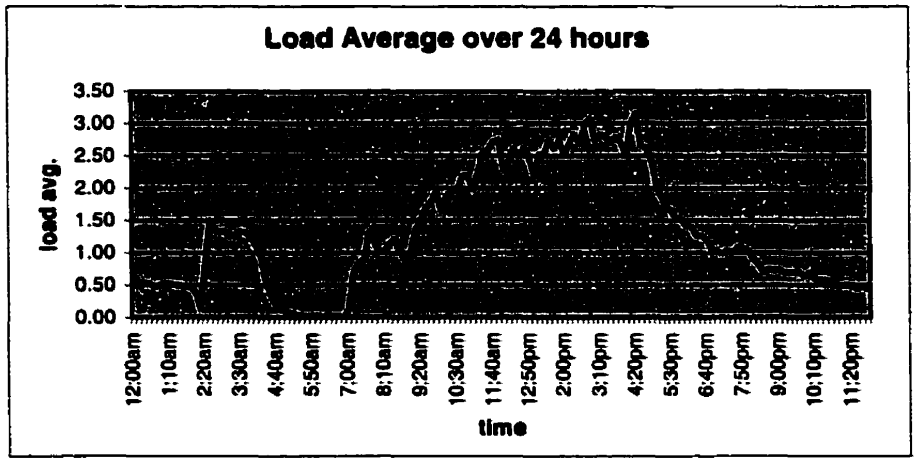

Figure 4-4. Load Average over 24 hours.

Looking at Figure 4-4, we observe an increasing load during the day (working hours) reaching maximum values around 3:00pm and 4:00pm. The increase of load during the night (between 2:15am and 4:30am) is due to the backup that is done on Aleph every night. Overall, we see that the load average – or the average number of jobs in the run queue – is low (always below 3.20), indicating a *light*

workload [LOU91]. Since the load average is continuously low throughout the day, this indicates that the system is probably underused.

The next step is to check the CPU utilization (c.f. definition in section 2.2.1) to determine how the CPU is being used. Using the output of the `sar` utility (c.f. section 3.1.1), Figure 4-5 shows the average CPU utilization over 24 hours. The samples were taken at 5-minute intervals during several days, and the average utilization for one day was computed by averaging the corresponding samples together. The standard deviation of corresponding samples remained low (between 2 and 5) for the majority of the samples, which means that the CPU utilization is consistent day to day.

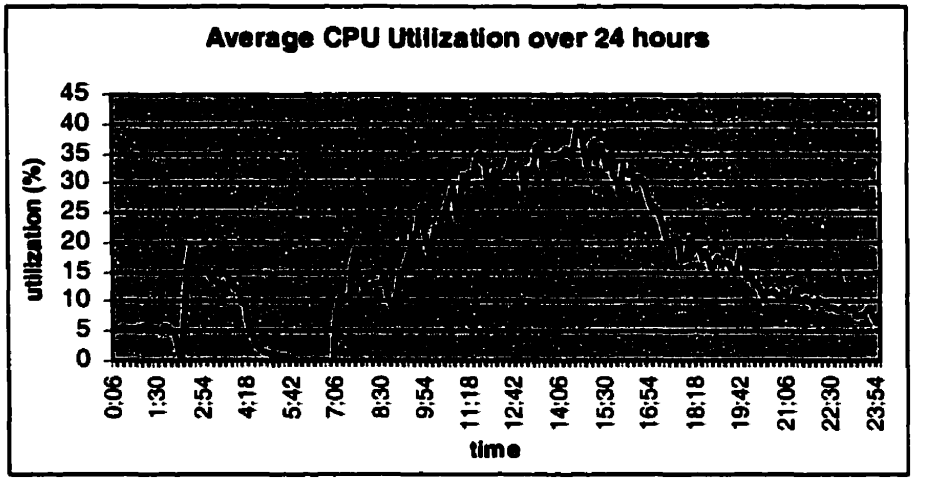

Figure 4-5. Average CPU utilization over 24 hours.

Observing Figure 4-5, we first note the great similarity with the shape of the load average graph in Figure 4-4. The average utilization over one day (24 hours) is 16.80%, which is relatively low. However, let us look at the peak utilization values: these show maximums reaching at most 40%. Since utilization never attains 80 or 90% during high loads, the system probably has more CPU power than needed (c.f. sections 2.1 and 3.1.1).

Although the utilization is a very important measure, it is also important to monitor the individual *user* usage and *system* usage (i.e. the percentage of time the system has been in user mode and in system mode, respectively). Using the output of `sar`, we find an average user time (`%usr`) of 12.46% and an average system time (`%sys`) of 4.34% (both over 24 hours). The user time is relatively low, with a peak time barely reaching 30% (see Figure 4-6). The system time is also low, but since it is less than 30%, then the system is considered to be under (very) light system CPU utilization and the CPU probably is not the next bottleneck of the system (c.f. section 3.1.1). Also, because system time is so low, we do not have to check the number of interrupts or the number of spins acquiring mutex locks (both values reported by `mpstat` –see section 3.1.1).

At this point, it is important to compute the 'user:system time' ratio (`%usr/%sys`). The resulting average over 24 hours is 2.7 (with a standard deviation of 0.5). This value shows no sign of wasted CPU cycles (c.f. section 2.1.1). Using `vmstat -s`, which reports total user and system time since the machine's last reboot, we also find an acceptable ratio value of 2.

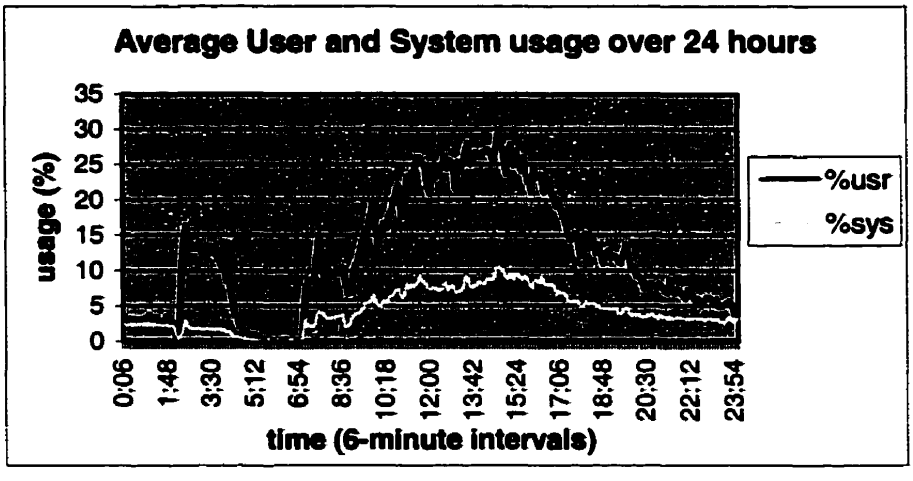

Figure 4-6. Average user and system usage over 24 hours.

Idle time is to be checked next. The average value reported by `sar` is 76.76% over 24 hours, which is quite high. Looking at Figure 4-7, we see that the idle usage (`%idle`) curve never goes down below 43% (which is the idle time value corresponding to peak utilization). This is another sign showing that the system is probably oversized (c.f. section 3.1.1).

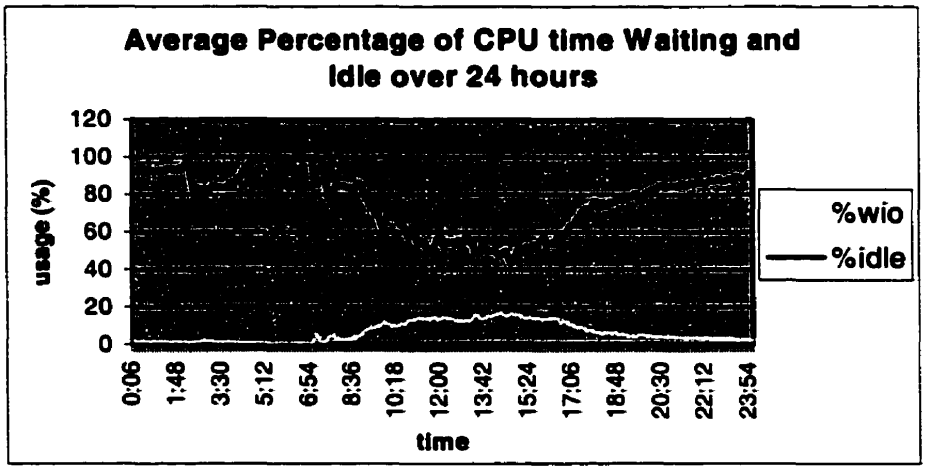

Figure 4-7. Average Percentage of CPU time waiting and idle over 24 hours.

Since Aleph is a multiprocessor system (with nine CPUs), it is interesting to look at the information reported by the `mpstat` utility (c.f. section 3.1.1). Figure 4-8 shows the per-processor utilization.

Overall, the load is balanced across Aleph's nine CPUs, even though CPU 14 seems to be constantly busier. During the backup at night (between 2:15am and 4:30am), however, we see that some CPUs are working harder than the others (CPUs 8 and 9 are 10-20% less utilized than the rest of the CPUs). This is probably due to the composition of the filesystems, so backing up different filesystems causes different loads.

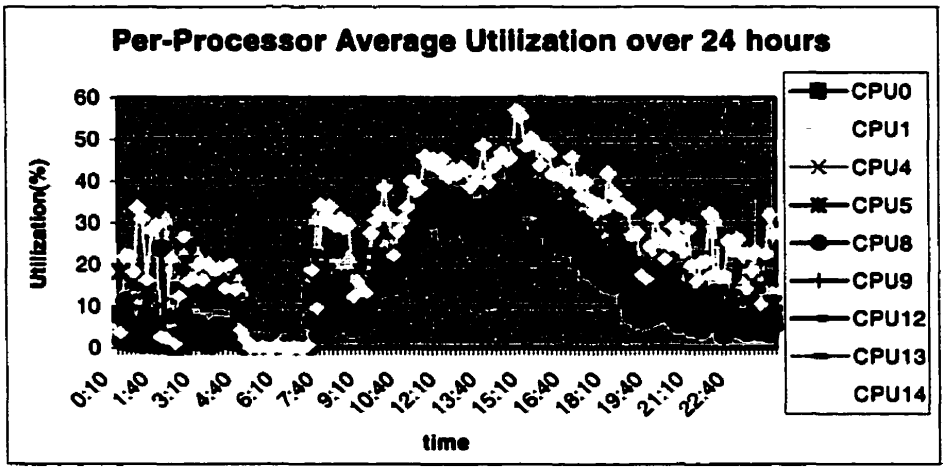

Figure 4-8. Per-processor average utilization over 24 hours.

Let us also check whether all the CPUs were online and working as expected during the period of the measurements. `cpus.se`, which was run on Aleph during the months of March and April, reports that all nine CPUs were working fine (always online at 400Mhz).

The workload is next to be determined. We consider here the number of processes running on the system to be the workload (c.f. section 2.1.1). The `nproc.se` tool (c.f. section 3.1.1) was used to evaluate the workload on Aleph, reporting the number of processes every 10 minutes. Figure 4-9 shows how the number of processes varies over 24 hours. The curve is similar to the load average and utilization curves in Figures 4-4 and 4-5 respectively, but is not exactly the same. The number of processes increases during the day (working hours) reaching a maximum of 395 around 2:00pm, and decreases at night to about 200 processes. Between 2:00am and 7:00am (when ALEPH is shut down daily), the number of processes stays around 60.

Let us take a closer look at those processes and identify them. For that purpose, `ps` and `top` (c.f. section 3.1.1) were run on Aleph. From the information reported by these utilities we find out which are the busiest processes at every moment (in

24 hours). During the day, the busiest processes are the web server (`www_server_main`), the PC server (`pc_server_main`), `rts32` (background daemon that runs when ALEPH is started up – used by ALEPH libraries), and the Oracle server. During the night, these processes (excluding the PC server) continue to run but are fewer (c.f. decrease in the curve around 17:00 in Figure 4-9). At 2:00am (and until 7:00am) the ALEPH application is shut down and the main process observed then is the backup process `dsmc` (IBM's ADSM package).

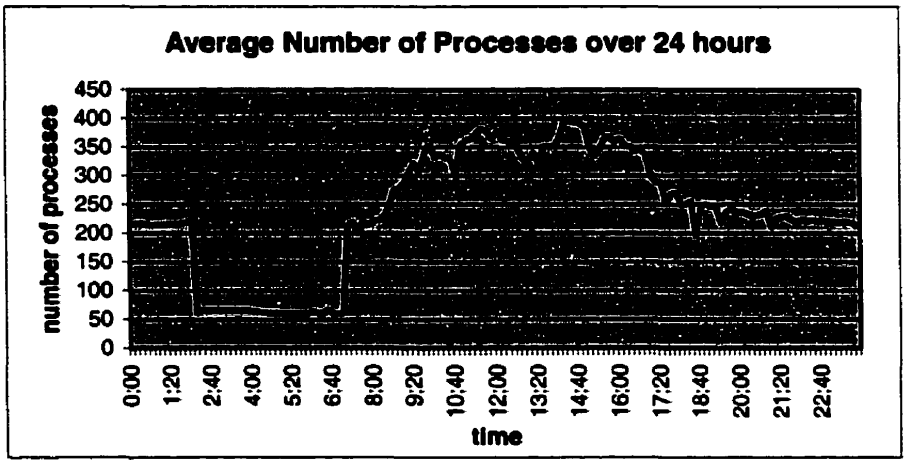

Figure 4-9. Average number of processes running on Aleph over 24 hours.

Another thing to look at is the process queues length (i.e. the number of processes waiting for a CPU time slice). `vmstat` and `sar -q` both report information on process queues (c.f. section 3.1.1). Looking at the output displayed by `vmstat` (run at 5-minute intervals during two weeks), we find that the number of runnable processes (`r`) is constantly 0. This value shows that no processes are waiting for a CPU time slice. It also means that the response time is equal to the service time, since the queue is empty and thus no time is spent waiting (c.f. section 2.2.1). Looking at the measurements of `sar -q` (run at 5-minute intervals during a week), we find a maximum run queue length of 1.6, which is a negligible value, and we deduce that no additional processors are needed [ALO99]. Also, `vmstat` reports on the number of blocked processes (`b`). The maximum value found

during the two weeks is 4, which is also negligible, and means that almost no CPU idle time is treated as wait for I/O time.

Overall, it seems that there is too much CPU power for the current workload on Aleph. Although this guarantees the absence of contention problems, it also means that the system is probably oversized.

4.2.2 Memory Usage

The first thing to check when monitoring memory is whether there is (excessive) paging and swapping activity on the system (c.f. section 2.1.2). We first look at the `po` (number of page-outs) and `w` (number of swapped-out processes) counters reported by `vmstat` (run at 5-minute intervals during two weeks). We find a continuous value of 0 for `w`, which means that the swapping activity on Aleph is very low. However, the values of `po` vary significantly, ranging from 0 to 4561 kB/sec with an average of 180 kB/sec with a very high standard deviation (273.3). Figure 4-10 shows the average number of page-outs happening on the system during a 24-hour period. The values were computed by averaging corresponding samples together during 6 days. Although informative, this graph is not exactly characteristic of the number of page-outs because of the high variability (large standard deviation) among the corresponding values sampled each day.

Overall, the page-outs rate increases during the day with variable peaks that reach very high values (close to 1000 kB/sec at the start and end of the workday). During the night, the page-outs are fewer, and during backup time (2:00am to 7:00am) there are no page-outs at all (value 0). Excluding the very high peaks, the highest number of page-outs stays around 400-500 kB/sec during the heavier workload (c.f. Figure 4-9), which is normal since a system with virtual memory is supposed to be paging when needed (c.f. section 2.1.2). As for the very high peaks, there probably is not a problem of lack of memory because these peaks are not continuous (c.f. section 2.1.2 and 2.1.2).

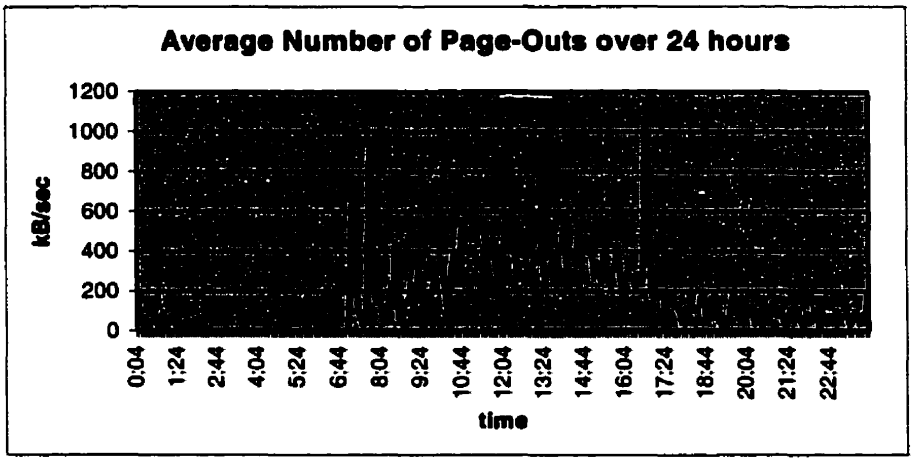

Figure 4-10. Average number of page-outs during 24 hours.

Let us also check the page-stealing daemon scanning rate (`sr`) in `vmstat`'s output to make sure that there is no memory shortage on the system. Figure 4-11 shows the average scanned-pages rate every 5 minutes during 24 hours (computed by averaging corresponding samples together during 6 days). We observe that the rate increases with the workload again (c.f. Figure 4-9) – which is normal/expected – and that the highest values stay around 200 pages/sec. This shows that the system is not suffering from a lack of memory.

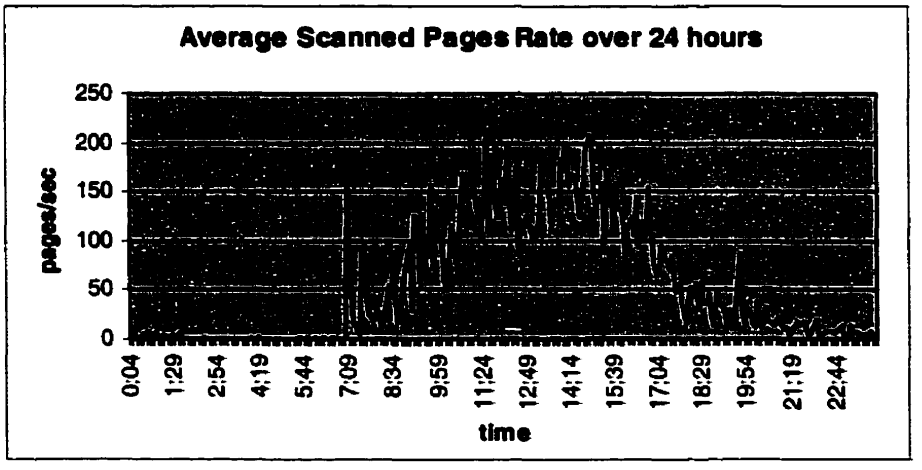

Figure 4-11. Average scanned pages rate over 24 hours.

Let us now look more at the swapping activity. The deficit paging parameter de reported by `vmstat` reflects excessive swapping (and consequently an important memory shortage) when it is non-zero. Looking at the statistics on Aleph, we find this value to be almost always 0, but with a few (exceptional) bursts reaching several thousands of kB. Since these bursts are not continuous, we probably can ignore them. Also, looking at the output of `sar -r`, we find that the available swap space is always large: it reaches 5.5 GB at night and never goes down lower than 2.5 GB under the heavier workload during the day (those values are reported by the `freeswap` counter in 512-byte blocks – c.f. section 3.1.2).

Overall, there does not seem to be any major problem with the memory on Aleph presently. However, memory should be constantly monitored to watch whether the large number of page-outs is increasing, as it could be the cause of a possible memory bottleneck in the future.

4.2.3 Disk Subsystem Usage

Let us now turn our attention to disk I/O activity on Aleph. The machine has 12 mirrored disks (so 24 physical disks) of which 2 are operating system volumes and 10 are for the ALEPH software and the database. Using the output from `iostat`, we can observe the activity of each disk (and also compute several important performance metrics – c.f. section 3.1.3). The operating system disks (the 4 physical disks[6]) appear to have a similar and consistent behavior. Figure 4-12 shows the utilization and throughput (both obtained from `iostat`) of one of these disks. The samples were taken at 5-minute intervals during 24 hours. Both parameters increase during the higher workload (c.f. Figure 4-9), but their values remain low overall (15% maximum utilization and 35 operations/sec maximum throughput).

[6] `c5t0d0s3`, `c5t1d0s4`, `c5t8d0s4`, and `c5t9d0s4`.

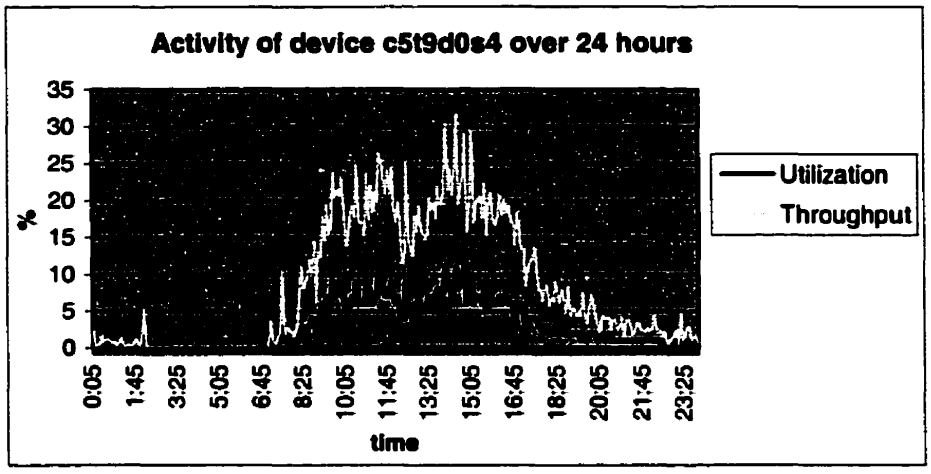

Figure 4-12. Activity of an operating system device over 24 hours.

We also computed the values of the total queue length and total response time (c.f. section 3.1.3). The total queue length calculated is negligible (less than or equal to 0.5) during the whole day, with just one exceptional peak reaching 7.6 around 5:00am for one of the two mirrored disks. The total response time computed is also negligible (under 0.6) for all the samples.

As for the other 10 mirrored (20 physical) disks, the activity of each one them is different from the others. Utilization and throughput vary significantly from one disk to the other. The disks containing the ALEPH software (`c2t2d0s2`, `c3t50d0s2`, `c2t18d0s2`, and `c3t34d0s2` – all in volume `/aleph1`) have very low utilization and throughput values that rarely go over 2% and 2 operations/sec (respectively) during the higher workload in the day. The disks that serve for development purposes (`c3t37d0s2` and `c2t5d0s2` in `/aleph3`) show very unstable utilization and throughput values. A sequence of peaks is observed during the higher workload, with a maximum utilization of 4% and a maximum throughput of 2.6 operations/sec.

The database disks (`c3t32d0s2` and `c2t0d0s2` in `/aleph`, `c3t39d0s2` and `c2t7d0s2` in `/aleph4`, `c3t42d0s2` and `c2t10d0s2` in `/aleph5`, `c3t48d0s2`

and `c2t16d0s2` in `/aleph6`, `c3t54d0s2` and `c2t21d0s2` in `/aleph9`, `c3t56d0s2` and `c2t22d0s2` in `/aleph10`, and `c3t58d0s2` and `c2t24d0s2` in `/aleph11`) have a similar overall activity, although some disks are busier than others. The disks in volume `/aleph5` are by far the most active, with a maximum utilization occasionally exceeding 40% and a maximum throughput reaching 32.8 operations/sec.

The disks in volume `/aleph4` and `/aleph6` show unstable utilization and throughput values with several peaks during the higher workload, all under the 25 threshold (for both parameters). The other disks (in volumes `/aleph`, `/aleph9`, `/aleph10`, and `/aleph11`) have a similar activity pattern with maximum utilization and throughput values ranging from 8 to 11. Figure 4-13 shows the utilization and throughput of one of those disks (device `c2t0d0s4`). We see that the throughput and utilization increase during backup time and during the higher workload in the day (c.f. Figure 4-9), which is normal and expected.

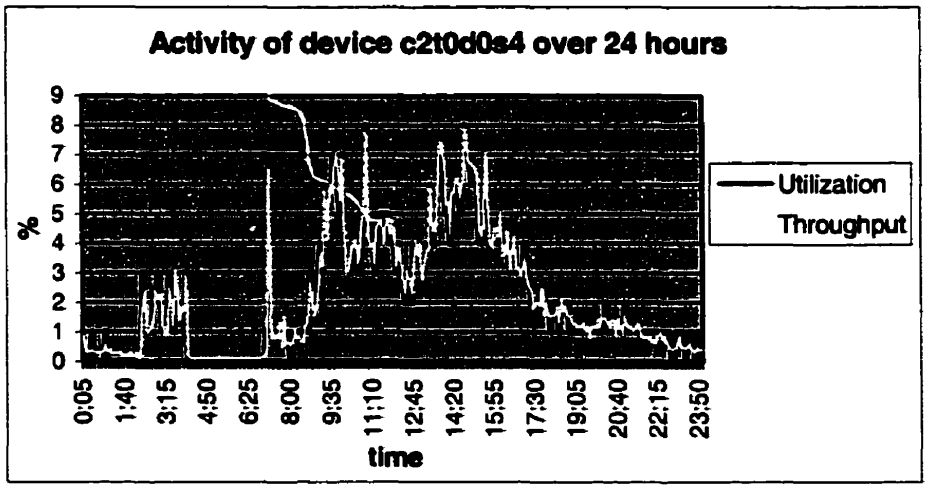

Figure 4-13. Activity of device `c2t0d0s4` over 24 hours.

The queue length and response time values computed for all the non-operating system disks are very low: the maximum queue length and response time for all samples are both less than 1. This means that there are very few processes waiting

for I/O. Indeed, `vmstat` reports a maximum number of blocked processes (`b`) of 4. So there should not be any disk contention or bottleneck.

Let us now look at the disk space usage. `df -k`, which was run once a day for a month (March), shows that the usage of all filesystems is quite stable (i.e. no disk is getting filled up quickly). However some volumes (`/aleph5`, `/aleph6`, and `/aleph9`) are almost full and a cleanup or a redistribution of the data might need to be performed soon.

Overall, all the disks on Aleph seem to be under-utilized. Although low utilization usually guarantees the absence of disk bottlenecks, it means, once again, that the system is oversized.

4.2.4 Network Activity

Finally, let us check the network activity. Using `netstat -i` we obtain information on Aleph's network interface. `netstat -i` was run during two weeks at 5-minute intervals. Figure 4-14 shows the average number of incoming and outgoing network packets during a day. We observe an enormous peak of output packets (almost reaching 70,000) and another smaller peak of input packets between 2:15am and 4:00am. This is the backup[7] time and such an activity therefore normal. During the backup, ten instances of the `dsmc` process are running in parallel (c.f. section 4.2.1). The limiting factor here is probably the network interface of the mainframe.

We also observe an increase in the activity during the day (working hours) that corresponds to the higher workload observed in Figure 4-9.

Over all samples, no input and output errors are reported (value constantly at 0) and the collision rate is stable at 0%. This shows that the network traffic is light.

[7] The backup is done to a remote machine (mainframe) across the network using TCP/IP.

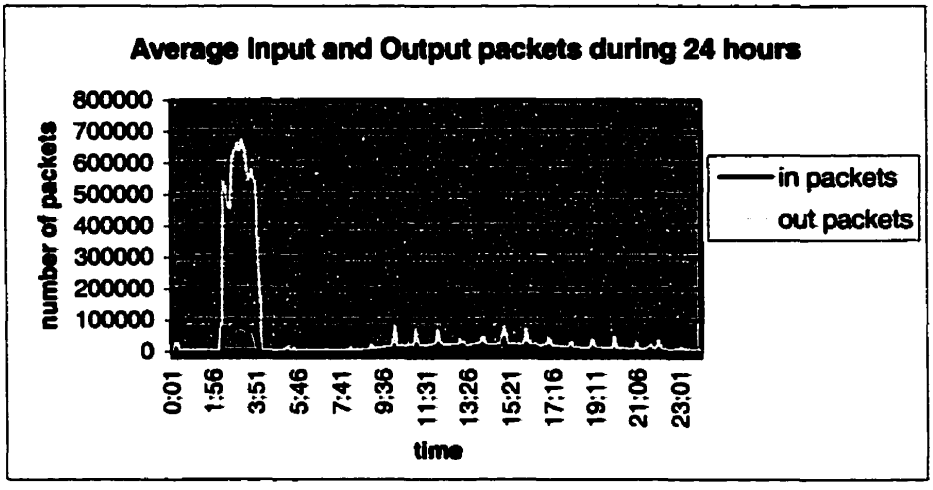

Figure 4-14. Average input and output packets over 24 hours.

Next, we look at the TCP data reported by `netstat -s`, which was run every 20 minutes during two weeks. We can get some statistics about the connections opened. The number of currently established connections (`tcpCurrEstab`) noted varies between 5 (minimum) and 168 (maximum) during one day. We also find maximum numbers of active and passive opens (`tcpActiveOpens` and `tcpPassiveOpens`) of 4,095,296 and 9,737,682 respectively. There is another counter that is worth checking out: `tcpListenDrop`, which counts the number of times a connection was dropped (because of a full listen queue). The value reported is 484 (maximum); because it is greater than zero, the size of the listen queue should be increased [CPS98].

Overall, there does not seem to be any network problem on Aleph. The traffic is very light and thus the network does not constitute a bottleneck for the system.

Finally, the performance of Aleph is good and no major problems or bottlenecks exist on the system. However, the system not fully utilized because of an oversize in the hardware.

Chapter 5

Performance Analysis of BANNER

5.1 BANNER System Overview

The BANNER system, produced by SCT Corporation, is the new Enterprise Resource Planning (ERP[1]) system at McGill that is replacing the mainframe system for the Student, Finance, and Human Resources systems. It is an integrated client/server system that uses common/shared data and provides easy and secure access through PC clients (majority of use) and through the Web, for different categories of users.

The BANNER system is composed of many products that run concurrently and share data. The Finance Information System (FIS) has been live since June 1st, 2000; the Human Resources Information System (HRIS) – excluding the Payroll system (to be installed January 1st, 2002) – is live since January 1st, 2001; and the Student Information System (SIS) is to go live in the near future (the Admissions module of it will go live in November 2001). The General System contains the modules and data that are common to all the BANNER software application systems. Data, procedures, and functions for all the modules are stored in an Oracle database. Client connections are made using either Oracle forms developed with Oracle Developer 2000 (version 6), or through the Web using the Oracle Application Server. Additional Web products are also part of the system.

[1] ERP is a common industry term that designates multi-module application software that typically use a relational database system.

5.1.1 System Hardware Configuration

In the BANNER system hardware architecture many machines are involved (see Figure 5-1). The *core* BANNER system hardware is composed of *two* Solaris database servers, *one* Solaris Oracle Application server, and *one* Novell forms server. Several other systems including additional Web servers and a data warehouse server provide peripheral functionality. The three Solaris servers included in the performance analysis of BANNER (sections 5.2, 5.3, and 5.4) are described below:

1) '**Nimbus**' is the central production machine with the Oracle database. It is a Sun Enterprise 3500 with *four* 400Mhz-CPUs (UltraSPARC-II) sharing 3GB of RAM, with *three* internal SCSI disks and *twenty-two* disks (a mixture of 9- and 18-GB disks) in one A5200 fiber channel array (RAID 1 – mirrored), and running Solaris 2.6. Its disk management software is Sun Volume Manager 2.6, and the filesystem type is UFS. It has one 100-Mbit switched duplex Ethernet interface.

2) '**Neptune**', the development machine, contains the identical software as Nimbus and several copies of the Oracle database (different development teams use different database instances to do their work). It is a Sun Enterprise 450 with *four* 400Mhz-CPUs (UltraSPARC-II) sharing 4GB of RAM, *ten* 18-GB disks (no Volume Manager and no RAID) with UFS, one 100-Mbit switched duplex Ethernet interface, and running Solaris 2.6.

3) '**Poseidon**' runs the Oracle Application Server (Web server). It is a Sun Enterprise 450 with *four* 400Mhz-CPUs (UltraSPARC-II), 2GB of RAM, *one* internal 18-GB SCSI disk with the OS and the Oracle Application Server, and *one* external 18-GB SCSI disk for backups (no Volume Manager and no RAID), with UFS, one 100-Mbit switched duplex Ethernet interface, and runs Solaris 2.6.

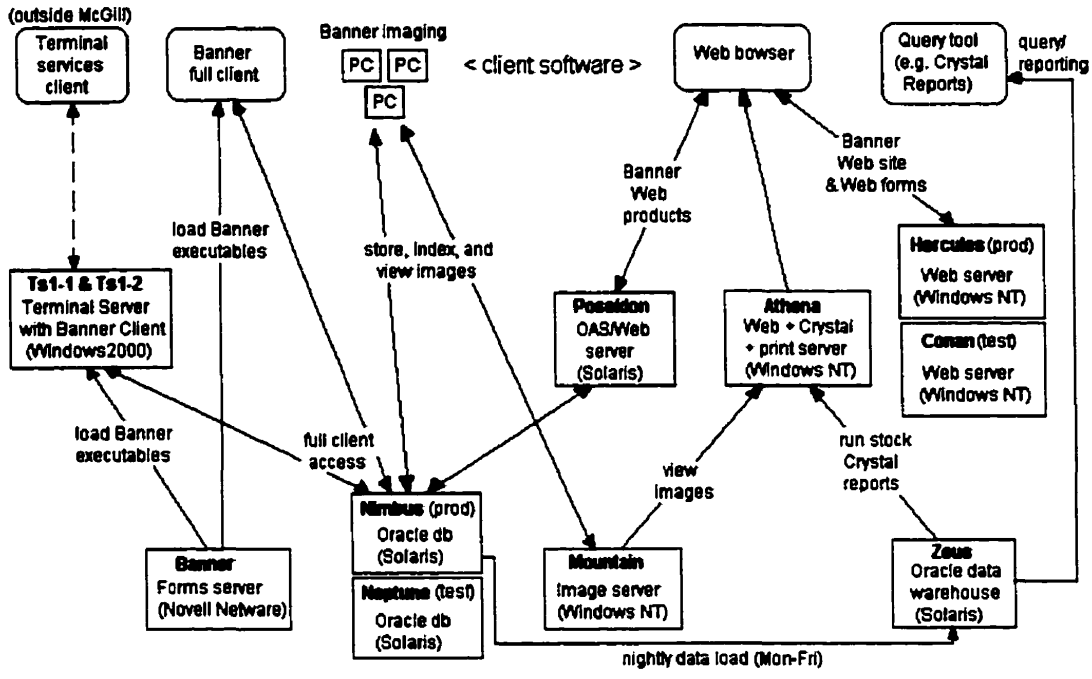

Figure 5-1. BANNER System Hardware Architecture.

There is one more Solaris Server –'Zeus', the Oracle data warehouse server – but it is not included in the BANNER performance analysis, since it contains no BANNER software but only BANNER *data* and Oracle. Its detailed description is therefore omitted from the list above. This Oracle data warehouse is a resource that is used to supplement BANNER's reporting capabilities. Most users access it using the Crystal Reports report tool (either directly from a PC-GUI client or via a recently installed Crystal Web gateway).

The rest of the machines include:

- The BANNER forms server 'Banner' running Novell NetWare, from which the BANNER executables are loaded to the BANNER PC/full clients and to the terminals. The BANNER PC client software consists of the forms runtime executable and about 700 compiled Oracle Developer 2000 forms; copies of

the compiled forms and the Oracle Developer 2000 software are stored on the BANNER forms server;

- The BANNER image server 'Mountain' running Windows NT, which stores images such as invoices scanned by the BANNER imaging PCs;

- The two machines 'Ts1-1' and 'Ts1-2' running Windows 2000 and Microsoft Terminal Server software, which serve users who cannot easily run the PC GUI client locally (Macintosh/UNIX users, users who are located off of McGill's campus in several hospitals);

- The machines used mainly for Crystal Reports (for professors/researchers):
  - the Web, Crystal, and print server 'Athena' running Windows NT, which serves as a Web front-end to the data warehouse;
  - the web server 'Hercules' (production) and 'Conan' (test machine) running Windows NT.

The existing BANNER software installed is version 4. The Oracle version used for it is 7.3.4.5, with a Client-Server and multithreaded architecture. BANNER 5.0 uses Oracle 8.1.7.1. Some of the BANNER 4 databases are currently being moved to Oracle 8.1.7.1.

5.1.2 The BANNER Application

There are two full installs of all the BANNER software (C and Cobol programs, forms source code, database packages, etc.): one on Nimbus and one on Neptune[2]. The compiled Oracle forms and the runtime components for the client are installed on BANNER (the Novell server). The Oracle Application Server (Web server) is installed on Poseidon.

[2] Neptune has one BANNER 4 install and one BANNER 5.1 install.

The BANNER modules are interdependent because they run concurrently and share data; each module has components implemented (or to be implemented):

- The Financial Information System (FIS) is composed of different modules: General Ledger, Purchasing, Accounts Payable (including imaging), Accounts Receivable, Budget Development Grants and Contracts, Stores Inventory, etc.

- The Human Resources Information System (HRIS): General Person, Position Control and Budget Development, Personnel Services Budgeting, Time Entry and Payroll, Employment/Compensation Administration, Benefits/Deductions Administration, Electronic Approvals, etc.

- The Student Information System (SIS): General Person, Recruitment, Admissions, Catalogue, Registration, Student Fees, Academic History, Course and Program Planning, Class Schedule, Faculty Load, etc.

All of the BANNER modules store data in one database. The production copy of this database is on Nimbus and is accessed by all the real BANNER users (professors, departmental administrators, etc.) either through the PC forms client or via the Web server. On the test machine (Neptune) there are *twelve* copies of the database. The databases on Neptune all started out as standard copies of a BANNER database, but now are each used by different development teams to test programs, test loading data, make changes to the BANNER procedures, etc. Because all the modules share tables in the database, it is difficult for the programming teams to all use the same database. The twelve database copies are therefore essential for the testing purposes, even though they consume significantly large amounts of memory on Neptune.

The database is shut down nightly – but the operating system stays up – and is backed up once a week. Every night (Monday to Friday) data is extracted from the BANNER production database on Nimbus and loaded into an Oracle 8.1.6.1

database on Zeus (the data warehouse). Data loads are done daily so that reports to be generated have the most up-to-date information.

5.1.3 BANNER Users

BANNER does not have statistics-reporting facilities like ALEPH does, so it is more difficult to get precise numbers about the users connected at one time and the categories of users of the system.

Generally speaking, the end-users are professors, researchers, departmental administrators, accountants, etc. All of them access the database server (on the production machine Nimbus). The users of Neptune are developers, testers and trainees. The Web server (Poseidon) is presently under-used because most of BANNER users are connected through the fully functional PC client. When the SIS will be live, Poseidon will have its load substantially increased (about 45,000 additional users), since the students will connect through Web browsers. Also, with the Payroll component that will go live in June, there will be an increase of a few thousand users on the system. So the total expected growth rate of BANNER users is very significant. Presently, the only live module is the Finance module (FIS), so most of the current users are performing tasks such as processing purchase orders, balancing accounts, and so on.

The BANNER license is a CPU- or 'power unit'-based license, so there is no contractual limit on the number of users to connect. There are presently 5,255 defined Oracle users who can access the database. Only 1,800 of those are actually used (mostly by the Finance administrators), and just 500 processes can be simultaneously run (i.e. less than 500 users, since some can be have multiple connections).

The listener process on Nimbus (`/opt/app/oracle/product/734/bin/tnslsnr`) has been running since March 2000 and keeps track of all the

connections made to the database. The log file generated (`/opt/app/oracle/product/734/network/log/listener.log`) contains the number of connections – or sessions – made as well as where they came from (Web or BANNER client). This number cannot be used to deduce the number of users of the system, since one user can open many sessions.

5.1.4 System Use

Every machine has one predominant process running:

- On Nimbus, the main process is evidently the database server;
- Similarly, on Neptune, the busiest process is the database server;
- On Poseidon, the main process is the Web server.

In the next sections, the performance analysis of the three BANNER machines Nimbus (section 5.2), Neptune (section 5.3), and Poseidon (section 5.4) is carried out.

5.2 Performance Study of Nimbus

In this section, the central production machine – with the Oracle database and the database server – Nimbus is analyzed. The tools used for this analysis are almost identical to the ones used in section 4.2 (all described in section 3.1), and are listed in Appendix B. These tools were run mainly during the months of March and April, as well as in the beginning of May 2001.

Figure 5-2 shows the total number of connections to the database on Nimbus each month. There is an increase in the number of users of BANNER (as the finance system went live on June 1, 2000, more employees are trained on the system and are now using it regularly, developers are added, etc.) every month. Therefore, compared to the previous months, March and April 2001 are the most interesting months for taking measurements, as more activity can be recorded.

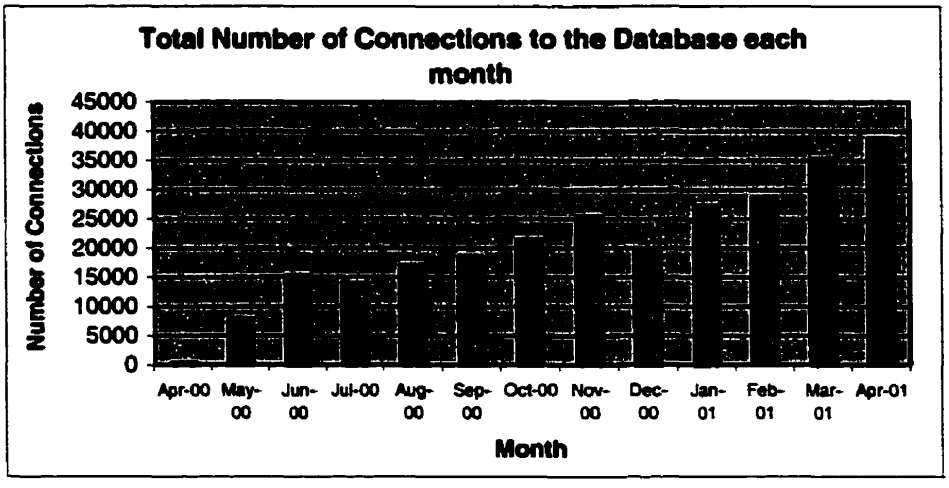

Figure 5-2. Total number of connections to the database each month.

Also, a much higher activity is recorded at the beginning of each month. Figures 5-3 and 5-4 show that during the first week of each month, the number of connections to the database is 1.5 to 2 times bigger than in the following three

weeks. Indeed, the activity generated at the beginning of the month is due to the effective closing of the fiscal period (during the first few days of the month) and to users then checking their new balance (for research grants, for example) for the new month.

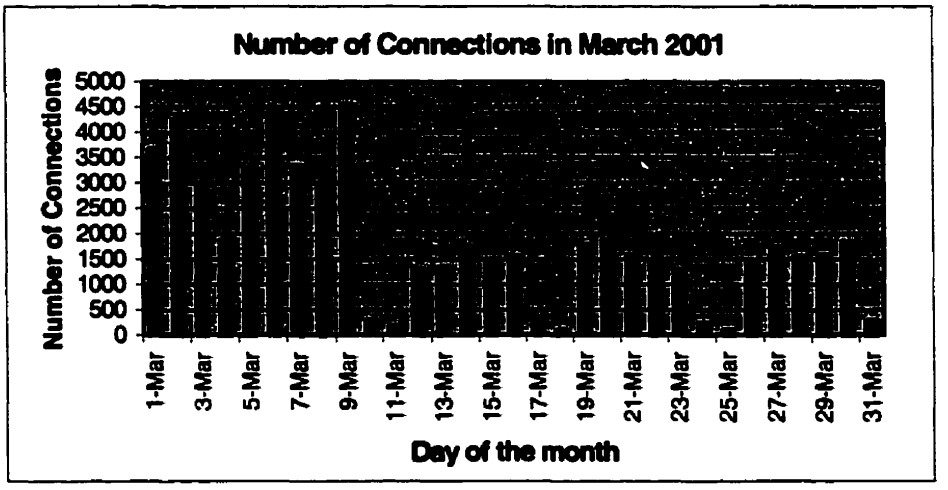

Figure 5-3. Number of connections in March 2001.

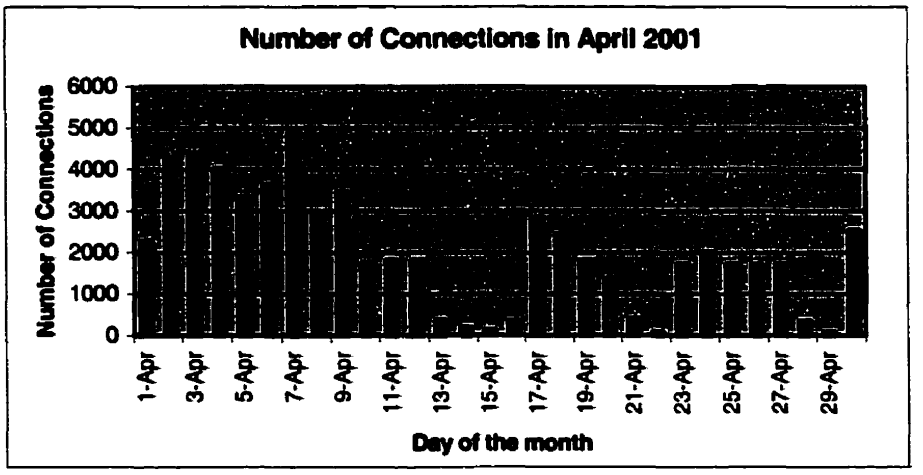

Figure 5-4. Number of connections in April 2001.

The performance analysis of Nimbus is presented next and is divided into the following subsections in order to analyze all the aspects of the system: the CPU

usage (5.2.1), the memory usage (5.2.2), the disk usage (5.2.3), and the network activity (5.2.4).

5.2.1 CPU Usage

Let us start by taking a look at the load average (Figure 5-5) reported by `uptime` (c.f. section 3.1.1). The samples were taken every 5 minutes during ten regular[3] days in the first two weeks of April.

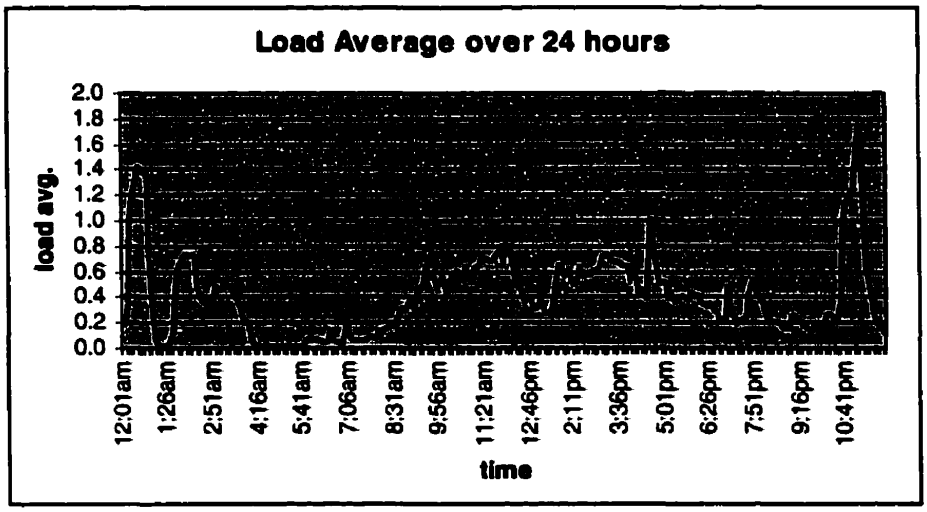

Figure 5-5. Load average on Nimbus over 24 hours.

During the day (working hours), the load starts increasing around 8:00am and keeps on increasing until 12:00pm (to a value of 0.81) where a decrease is observed (because users go on lunch break). The load goes back up around 2:00pm and reaches a maximum at the end of the day (1.02 at 4:36pm). This end-of-workday peak is probably due to the closing of the BANNER clients, which commit to the database and thus generate a higher load.

In the evening, there is still some activity (some finance jobs are carried out), as the BANNER system is available until 10:00pm (production hours are 6:00am to

[3] I.e. excluding the days where the data conversions are performed.

10:00pm). At 10:00pm, the database is put into restricted access and the batch jobs are started in order to extract the data to refresh the data warehouse Zeus (c.f. Figure 5-1). This generates a high load on the system and is observed in Figure 5-5 between 10:30pm and 11:30pm. The following peak observed at around 12:30am is due to the backup of the archive logs (used for recovery). At around 1:30am – and until approximately 4:00am – the database is shut down and is backed-up along with all the files that have been modified (the first higher peak is due to the search of these files).

Figure 5-5 does not include the data from the days where the data conversions[4] are performed. When this data is included, a higher load average is observed during the day reaching a value of 8 around 10:30am.

Overall, the average number of jobs in the run queue (i.e. the load average) seems very low. This probably indicates a light workload [LOU91] and possibly that the system is underused. The next step is to look at the CPU utilization on Nimbus.

Figure 5-6 shows the average CPU utilization over 24 hours, obtained by averaging several weekdays of `sar` statistics. Since the utilization is not exactly the same every day, the graph is not fully characteristic of a 24-hour utilization and the standard deviation is often relatively high (with values ranging from 0 to 25). However, it is still useful to have an idea of the overall behavior of the utilization curve.

We notice, in Figure 5-6, a close resemblance with the load average graph in Figure 5-5. The major peaks correspond and their interpretation is thus the same.

The average utilization over 24 hours is 14.68%, which is relatively low. Also, the maximum utilization is below 50% (45.25% is the maximum value found),

[4] The human resources data is taken out of the mainframe and converted (through conversion programs) to be loaded into the BANNER database. The data conversions are generally started on Fridays at 5:00pm and last until Saturday afternoon (sometime between 12:00pm and 6:00pm).

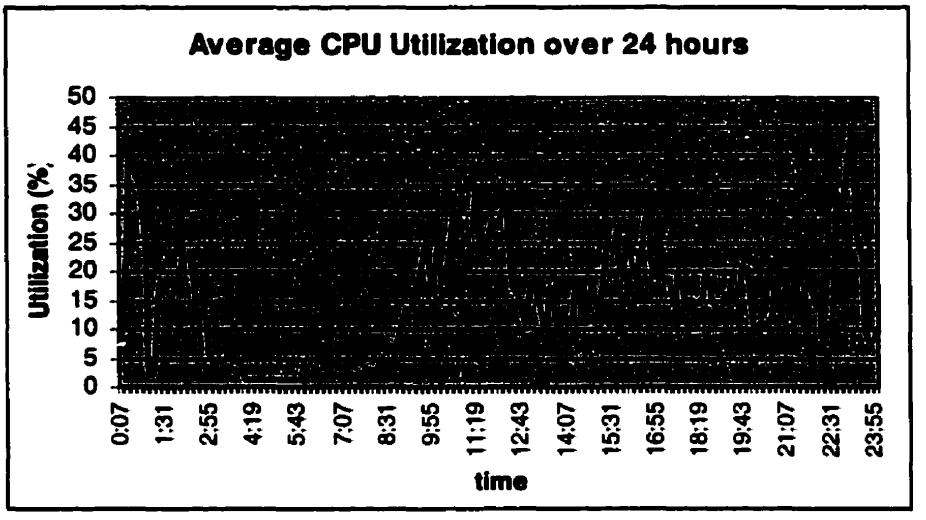

Figure 5-6. Average utilization over 24 hours.

whereas it is supposed to reach 80% or 90% under heavy loads (c.f. sections 2.1 and 3.1.1). However, if we look at the utilization on Saturdays, we see a significant difference. Indeed, Saturday is usually the day where the data conversions (which generate a heavy load on the system) are performed. The average utilization found for Saturday is 31.24% and the maximum utilization is 100% (which is too high and often means that the system is saturated). The average utilization values for each day of the week are shown in Figure 5-7.

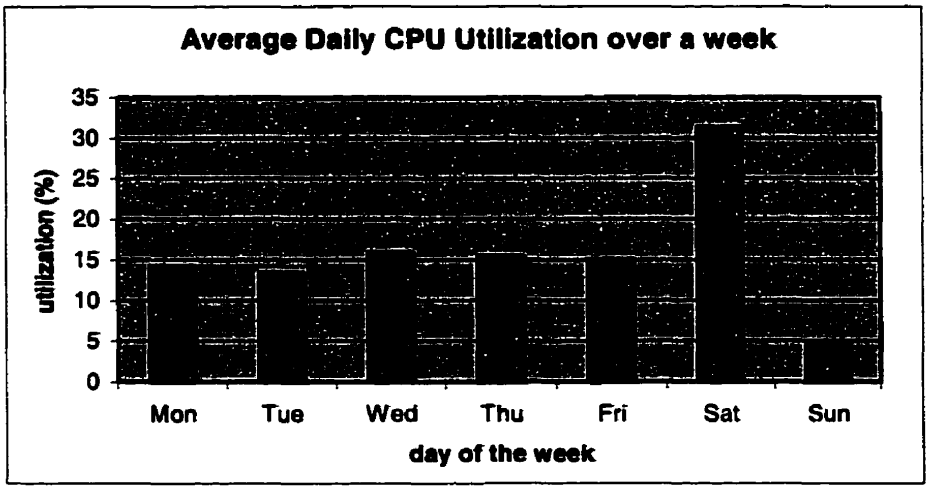

Figure 5-7. Average daily CPU utilization over a week.

Let us now look at the individual values of the user and system usage (i.e. user time and system time) for a usual weekday. Still using the output of `sar`, we find an average user time (`%usr`) of 12% and an average system time (`%sys`) of 2.81%. The user time is relatively low, with a maximum value of 36%. The system time is generally very low, with a maximum of 9.25% (relatively high) during the data extraction time (around 11:00pm).

We can now compute the `%usr/%sys` ratio. We find an average value of 3.71 (with a standard deviation of 1.88), which is good (c.f. section 2.1.1). Also, looking at the total user and system time since the machine's last reboot (reported by `vmstat -s`), we find a ratio value of 4.

As for idle time, we find an average value of 81.5% (reported by `sar`) over 24 hours, and that is a very high value. Idle time only goes down to 33.75% (minimum found) during the data extraction time. This means that during the day, the system CPUs are not being fully utilized.

Let us take a look at the utilization on a per-process basis. Figure 5-8 shows the individual CPUs utilization reported by the `mpstat` utility.

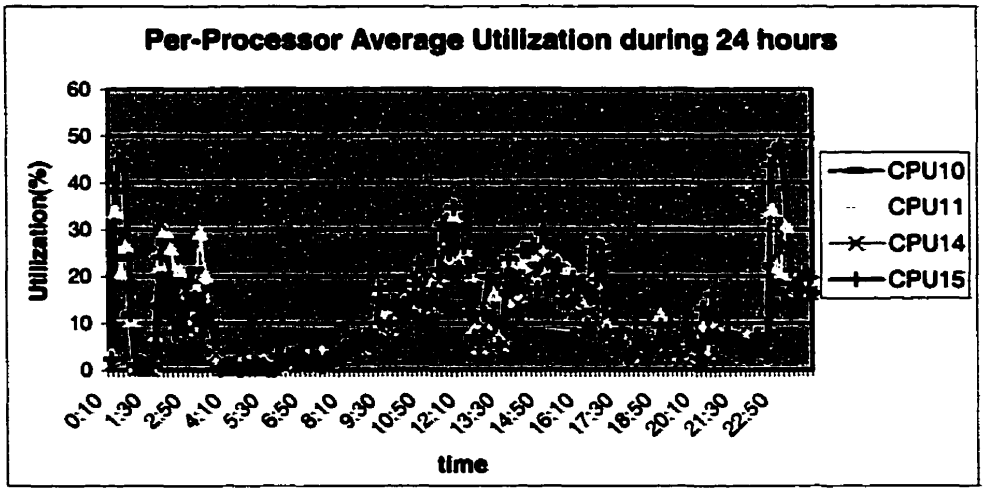

Figure 5-8. Per-processor average utilization over 24 hours.

The utilization of the four CPUs (called CPU 10, 11, 14, and 15) on Nimbus seems balanced overall. The only exception is during the data extraction time, where CPU 15 is busier than the other three.

Looking at the data reported by `cpus.se`, which was run every two hours during the months of March and April, we see that all four CPUs were always online at 400 MHz, as expected.

Let us now evaluate the workload, which we consider to be the number of processes running on the system (c.f. section 2.1.1). The `nproc.se` tool (c.f. section 3.1.1) was run on Nimbus every 10 minutes; Figure 5-9 shows the average number of processes reported during one day (24 hours). The average numbers were computed by calculating the average of corresponding samples of five weekdays in the beginning of April. The standard deviation was relatively low (average of 6.75) and therefore the workload is consistent during the week. On week-ends however, the number of processes goes down significantly (from a maximum of 355 during the week to a maximum of 157 on Saturday).

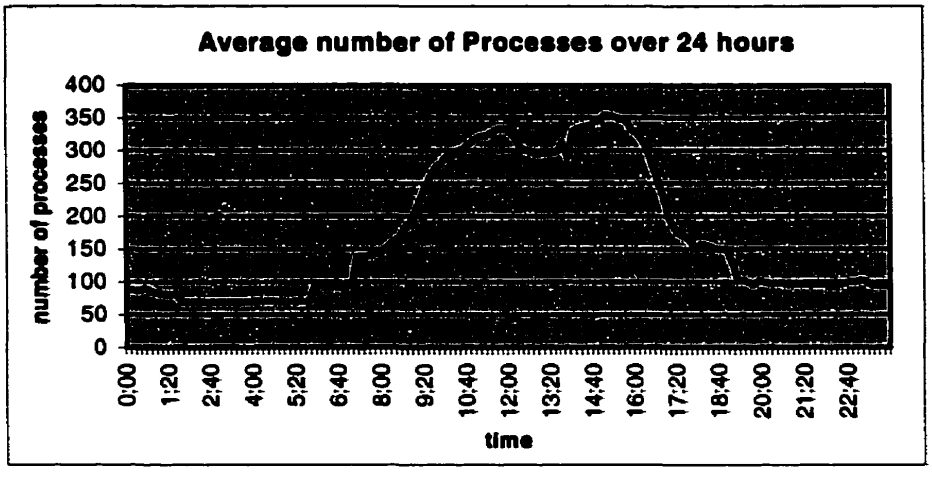

Figure 5-9. Average number of processes during one weekday.

During the day (between 7:00am and 6:00pm), the shape of the curve in Figure 5-9 is close to the daytime load average (Figure 5-5) and utilization (Figure 5-6) curves. The processes start to increase in number at around 7:00am; they keep increasing until lunch time where they decrease, and then increase again early in the afternoon, before starting their decrease till the end of the day. The significant difference with the load average and utilization curves is that no peaks are observed during the night. Indeed, a light workload is running on the machine at night and the reason for this is probably that the batch jobs and the backup processes are few (even though they are considerably large in size – c.f. Figures 5-5 and 5-6).

Using the `ps` and `top` utilities, we can identify those processes running on Nimbus. The busiest process observed during the day is `oracleMCGP`, which represents an Oracle session (from the production database). Also, there are several different processes running on the system 24 hours a day (but more clearly observed outside working hours). The main ones are `sqloper` (the AppWorx[5] Master process – schedules and manages jobs), `sooper` (the AppWorx Agent process – accepts job requests from the Master process and starts them), `sowineng` (the AppWorx daemon that handles connections from the GUI client), `oracleWORX` (an AppWorx Oracle session), and `dsmc` (the backup process – only runs during backup time). Other processes also noted (but less frequently) are `forappl` (the BANNER finance system document approval process) and `fgractg` (the BANNER finance system posting process), which are BANNER C programs run by the scheduler.

Let us now look at the process queues and see the number of processes that are waiting for a CPU time slice, using the output of `vmstat` and `sar -q`. `vmstat`, which was run every 5 minutes during two weeks, reports a very low number of runnable processes (`r`) (almost always 0, with very few exceptional maximal

[5] AppWorx is an automation package used for job scheduling.

values of 5 and 6), which means that almost no processes are waiting for a CPU time slice. We can also assume that the response time is equal to the service time, since the queue is practically empty (c.f. section 2.2.1). Looking at the reports of `sar -q` (run at 5-minute intervals during a week), we find an average run queue length of 0.8 (maximum of 6 on a Saturday, when the data conversions are performed), which is a negligible value, so no additional processors are needed on Nimbus [ALO99].

Let us also look at the number of blocked processes (`b`) reported by `vmstat`. We find processes blocked occasionally, with a maximum of 4 at a time. So there is probably little CPU idle time treated as time waiting for I/O.

Overall, with such a low load average and utilization on Nimbus, it may seem that there is too much CPU power on Nimbus. However, the user population of the system is growing significantly every month (c.f. Figure 5-2), so there should be enough the CPU resource to handle the additional (future) workload.

5.2.2 Memory Usage

Let us first check on the paging and swapping activity on Nimbus. We first look at the `po` (number of page-outs) and `w` (number of swapped-out processes) counters reported by `vmstat` (run at 5-minute intervals during several days). The average number of swapped-out processes is 20.54 (with a maximum of 76 and a minimum of 10). These values are a sign of excessive swapping activity (more on swapping further in this section).

The average number of page-outs computed is 92 kB/sec (with a maximum of 972 kB/sec and a minimum of 1 kB/sec). Figure 5-10 shows the average number of page-outs happening on the system during 24-hour. The values were computed by averaging corresponding samples together during 3 days (in the second half of the month of April). The standard deviation of corresponding samples was variable,

sometimes very high (over 300), so the graph is not exactly characteristic of the 24-hour page-out activity on the system, but it is still useful.

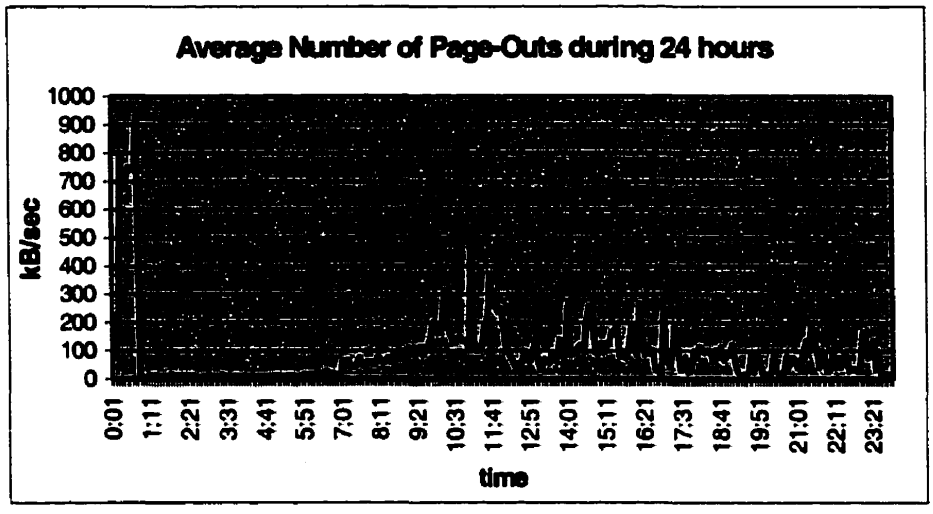

Figure 5-10. Average number of page-outs during 24 hours.

During the day and the night (until midnight), we observe a continuous paging activity with numerous peaks (the higher peaks correspond to the heavier workload during the day – c.f. Figure 5-9). Although the rate of the page-outs around 200 kB/sec is usually acceptable (excluding the very high peak during the backup of archive logs between 12:00am and 1:00am), the continuity of the page-outs is possibly a sign of a lack of memory (c.f. section 3.1.2). The only time where there is very little paging activity is when the database is shut down for backup (from 1:30am until approximately 5:00am).

To get more information about a possible memory shortage, let us have a look at the page-stealing daemon scanning rate (`sr`) in `vmstat`'s output. Figure 5-11 shows the average scanned-pages rate every 5 minutes during 24 hours (computed by averaging corresponding samples together during three days). The activity observed corresponds to the workload variation during the day (c.f. Figure 5-9). The highest `sr` value is very close to 200 pages/sec (excluding the high peak of more than 300 pages/sec during the data extraction time around 11:00pm).

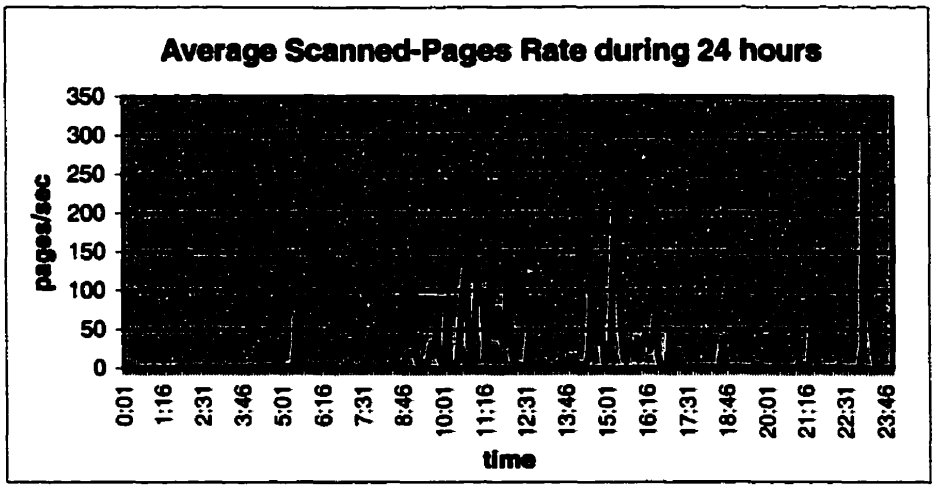

Figure 5-11. Average scanned-pages rate during 24 hours.

But let us now have a closer look at the swapping activity. Let us first check if the deficit paging parameter `de` (reported by `vmstat`) is non-zero. We find that the value of `de` is highly variable (within a day *and* across samples), with very high peaks during the heavier workload during the day (c.f. Figure 5-9) reaching huge values (such as 40,000 kB). This usually means that excessive swapping is happening, and is generally due to an important memory shortage.

Looking at the `freeswap` counter reported by `sar -r`, we find that the minimum available swap space is 1.25 GB (and the maximum is 2 GB), which is sufficient and only means that the swap space configured on the machine is large enough. The excessive swapping observed, however, means that there is a lack of physical memory (that is compensated by having a large swap space), because the swap space is being heavily used.

It seems that the memory on Nimbus is insufficient to serve all the running processes, as several are being swapped out. There is most probably a need for additional memory on this machine.

5.2.3 Disk Subsystem Usage

We now analyze the disk subsystem. Nimbus has 25 physical disks of which one is for the operating system and 24 are for the database and the BANNER software. The operating system disk is divided into several slices, one for each filesystem. The BANNER software is installed on one mirrored volume (2 disks) called `/sctv4`. The database is on 14 volumes, including one for the logs.

Using the output from `iostat`, we can observe the activity of each disk (and also compute several important performance metrics – c.f. section 3.1.3). The operating system disk `c0t0d0s2` has five active partitions. These partitions all have a maximum utilization inferior to 1.75% and a maximum throughput of 1.45 operations/sec (except during the backup at 1:35am, where the throughput goes up to 7.7 operations/sec for some partitions). Figure 5-12 shows the utilization and throughput of one of these partitions. The samples were taken from `iostat` at 5-minute intervals during 24 hours. We see that the utilization increases with the heavier workload during the day (c.f. Figure 5-9), whereas the throughput remains fairly constant throughout the day, except for the backup peak at 1:35am. Overall, utilization and throughput are rather low on this disk.

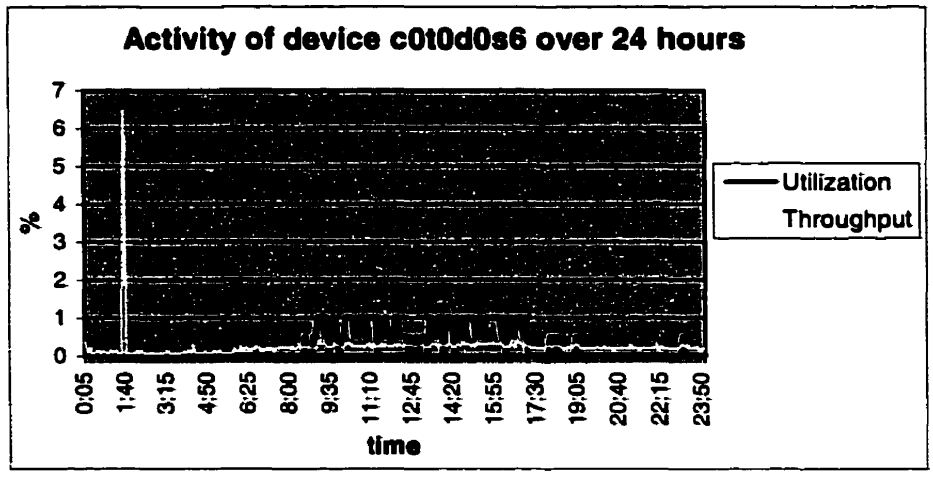

Figure 5-12. Activity of an operating system partition over 24 hours.

Utilization and throughput are also low on the `/sctv4` volume (disks `c2t57d0s2` and `c2t42d0s2`) containing the BANNER software. Both parameters remain under 2 during the whole day, except for a peak throughput of 12 operations/sec at 1:40am, during the backup time.

As for the database disks (`c0t1d0s2` and `c2t33d0s2` in `/dbp1`, `c0t2d0s2` and `c2t35d0s2` in `/dbp2`, `c2t53d0s2` and `c2t32d0s2` in `/dbp3`, `c2t37d0s2` and `c2t36d0s2` in `/dbp4`, `c2t48d0s2` and `c2t38d0s2` in `/dbp5`, `c2t51d0s2` and `c2t39d0s2` in `/dbp6`, `c2t54d0s2` and `c2t41d0s2` in `/dbp7`, `c2t49d0s2` and `c2t58d0s2` in `/dbp8`, `c2t56d0s2` and `c2t50d0s2` in `/dbp9`, `c2t34d0s2` and `c2t52d0s2` in `/dbp10`, `c2t40d0s2` and `c2t55d0s2` in `/dbp11`, `c2t38d0s2` and `c2t41d0s2` in `/dbp12`, and `c2t39d0s2` and `c2t49d0s2` in `/dbp13`), the activity of each pair of mirrored disks is different from the other pairs. Utilization and throughput often vary significantly from one disk to the other.

The disks in volume `/dbp5` are significantly more active than the others, and show many (unstable) utilization and throughput peaks during the day more or less corresponding to the workload peaks (some peaks reach 35% utilization and 43 operations/sec throughput). The second most active volume is `/dbp1`, where the utilization of `c0t1d0s2` reaches 20% and its throughput reaches 38 operations/sec at the beginning of the workday. Actually, if we look at what /dbp5 and /dbp1 contain, we see that /dbp5 has two of the busiest database files for MCGP (the production database) along with the Oracle temp tablespace files, and that /dbp1 contains the active redo logs (c.f. footnote 6 in Chapter 2 for definition).

Volume `/dbp2` shows a very low activity around 1 or 2 (for both utilization and throughput) during the whole day, with only two peaks: one during the backup at 1:45am and one at the beginning of the day between 9:00am and 10:30am. The disks in volumes `/dbp2`, `/dbp3`, `/dbp6`, `/dbp7`, `/dbp8`, and `/dbp13` have a rather similar behavior overall. They have a significantly higher utilization and

throughput varying between the values of 10 and 25 at the start of the workday (between 9:00am and 10:30am), and several other unstable peaks throughout the day. At night, all these disks show peaks due to the backup processes, but some are busier than the others (such as the disks in volumes `/dbp2`, `/dbp3`, and `/dbp7`). The disks in volumes `/dbp4`, `/dbp9`, `/dbp10`, and `/dbp11` are almost inactive during the day and only have one peak usage moment at the backup time.

The utilization and throughput peaks observed throughout the disks during the day (and corresponding with the higher workload) and during the backup time at night are normal and expected.

The total queue length and total response time values (c.f. section 3.1.3) were computed for all the disks on Nimbus. The results obtained show negligible values, both mostly under 0.5 for all the samples. This means that there are very few processes waiting for I/O at one time. In `vmstat`'s output, we find a maximum number of blocked processes (`b`) of 4, which implies the absence of disk contention or bottleneck.

However, the `iomonitor.se` tool (c.f. section 3.1.3), which was run during the last three weeks of April, generated several complaints about slow[6] disks at variable times. The disk that generated the most complaints is `c2t56d0s2` (in `/dbp9`). Also disks `c2t36d0s2` (in `/dbp4`), `c2t41d0s2` (in `/dbp7`), `c2t49d0s2` (in `/dbp8` or `/dbp13`), and `c0t0d0s2` (the operating system disk) generated some complaints (between 24 and 35 complaints in a nineteen-day period). These values should be further investigated for possible configuration problems.

Looking at the output from the `df -k` utility, we can now check the disk space usage. `df -k` was run during the second part of March and all of April. The usage

[6] The `IOSLOW` environment variable was set to 80.0 (see the description of `iomonitor.se` in section 3.1.3).

of the filesystems seems fairly stable but some of volumes have a tendency to fill up relatively quickly (`/dbp3`, `/dbp8`, and `/sctv4`) and need a regular cleanup to avoid disk contention problems.

Overall, there does not seem to be any major problem with the disks on Nimbus. However, most of the disks are under-utilized, as utilization and throughput are very often too low. The disk subsystem might be oversized for the current workload, but as the user population grows, there should be enough disk resources for a heavier workload.

5.2.4 Network Activity

We now look at the network activity on Nimbus. The `netstat -i` utility provides information on the network interface on the system (c.f. section 3.1.4). `netstat -i` was run every 5 minutes during one week. Figure 5-13 shows the average number of incoming and outgoing network packets during 24 hours. The average was computed for each sample for several weekdays. As the network activity varies each day, the graph shown is not a characteristic pattern of a 24-hour period. However, the overall shape of the graph is more or less consistent, especially during the day.

Indeed, when a backup is performed on the machine, the number of output packets increases significantly (up to 80,000), because the backup is done on a remote machine over the network. The number of input packets also increases during the backup, but not too much. During the day, the activity rises and follows the utilization and workload pattern (c.f. Figures 5-8 and 5-9), as expected. The peak observed at the end of the workday (close to 70,000) is probably due to the closing of the BANNER clients over the network, which commit to the database generating a higher activity.

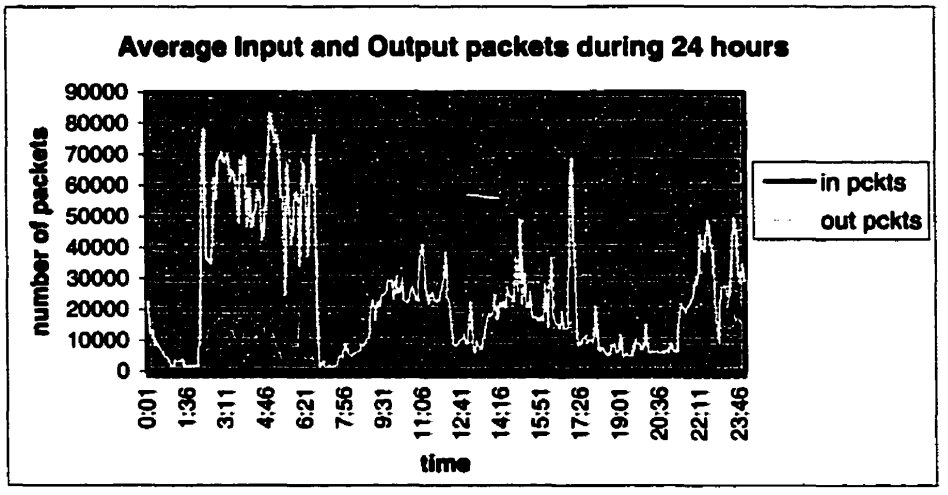

Figure 5-13. Average input and output packets over 24 hours.

In all the `netstat -i` samples collected, no input or output errors were reported and the collision rate on Nimbus remained at 0%. This implies that the traffic on the network is light.

Let us now look at the information reported by `netstat -s`, which was run during two weeks at 20-minute intervals. We are mostly interested in the TCP statistics provided. The number of currently established connections (`tcpCurrEstab`) observed in a 24-hour period varies between 21 and 273. Also the maximum numbers of active and passive opens noted (`tcpActiveOpens` and `tcpPassiveOpens`) are 335,781 and 74,402 respectively. The number of connections dropped because of a full listen queue (`tcpListenDrop`) remained at zero through the whole measurements, which means that the size of the listen queue is appropriate [CPS98].

Overall, the network seems to be performing acceptably. The network traffic is rather light, but as implied in Figure 5-2, the traffic is expected to increase in the near future, so the network throughput will increase as well.

5.3 Performance Study of Neptune

In this section, we look at the performance of the development machine Neptune, which contains the identical software as Nimbus, as well as twelve copies of the Oracle database. In fact, because Neptune is a test machine and several different development teams work on it (using different database instances to do their work[7]), it does not have a consistent workload and behavior, and it is therefore difficult to analyze its performance exactly.

The tools used for this analysis are the same as those used for Nimbus in section 5.2 (all described in section 3.1), and are listed in Appendix C. The period of the measurements is also mainly the months of March and April, as well as the beginning of May 2001.

In the following subsections, we try to evaluate the performance of Neptune by looking at the CPU (5.3.1), the memory (5.3.2), the disks (5.3.3), and the network (5.3.4).

5.3.1 CPU Usage

To have an idea about the overall system usage, we look at the load average reported by `uptime` in Figure 5-14. The samples were taken at 5-minute intervals during the first week of April.

As expected, the load average observed is fluctuating with several different peaks during the 24-hour period. However, the standard deviation calculated over all the corresponding samples is low (maximum value of 3.2), which means that the pattern represented in Figure 5-14 is a typical daily load average on Neptune.

[7] The developers use the database to test programs, test loading data, make changes to the BANNER procedures, etc.

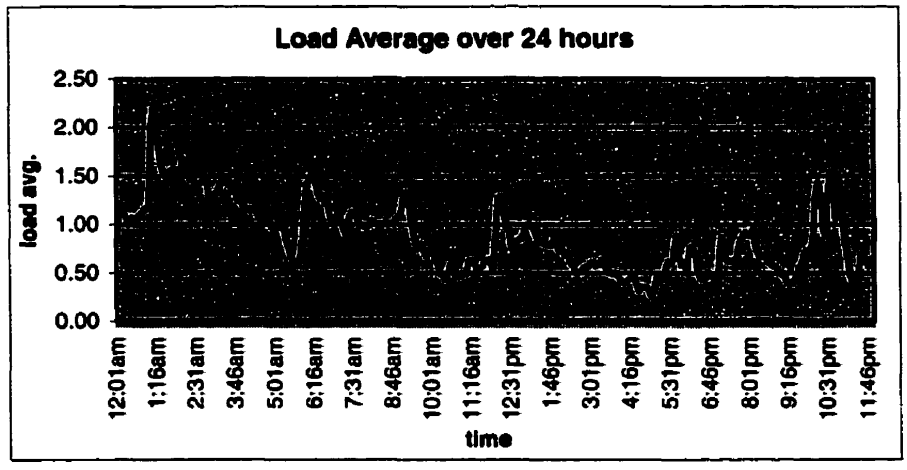

Figure 5-14. Load average on Neptune over 24 hours.

The highest peak observed (around 1:00am with a load of 2.24) is due to the backup, which goes on until 5:00am approximately. Around 6:00am, another peak marks the beginning of the Application Server reload[8]. At 9:15am, the peak noted probably marks the beginning of the workday, as the developers log in. Several different peaks follow during the day and in the evening and are due to different activities performed by the development teams. From 10:00pm to 11:00pm, the successive peaks observed are due to the batch jobs started.

The load average reported for Neptune seems quite low overall. This usually implies that the workload on the system is light [LOU91] and that the system is underused. But we still have to look at the CPU utilization and the workload to draw any conclusion. Also, remember that `uptime` can sometimes be misleading (c.f. section 2.1.1).

The CPU utilization is one of the most reliable measures. However, taking utilization averages on a test machine like Neptune can also lead to erroneous conclusions. This is because the utilization differs significantly from one day to

[8] There are 8 Web servers (or Application Servers) on Poseidon, of which 7 have connections to the database on Neptune and are used by the developers.

the other, as the developers perform different jobs every day and at different times. Figures 5-15, 5-16 and 5-17 characterize such a situation.

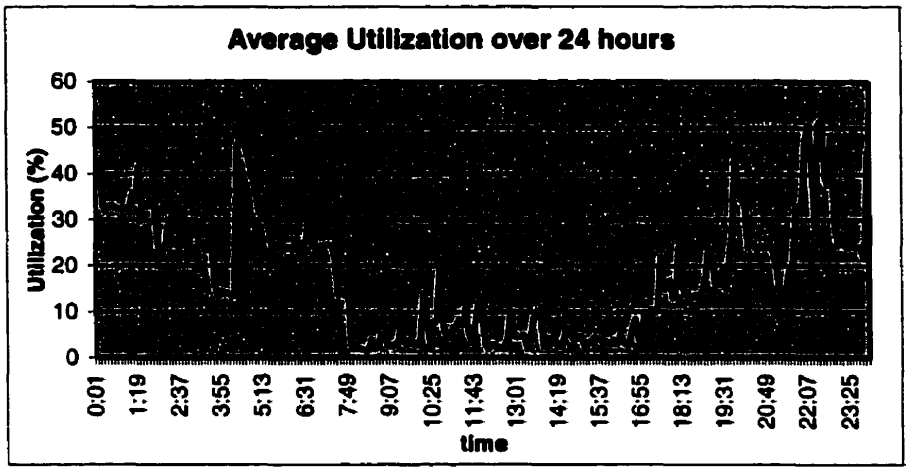

Figure 5-15. Average CPU utilization during 24 hours.

Figure 5-15 shows the 24-hour average CPU utilization of several days. The samples were taken every 5 minutes and corresponding samples for each day were averaged together. Of course, the standard deviation calculated is very high (maximum of 65.76). Looking at the curve in Figure 5-15, we see that the utilization pattern is quite unusual, as it increases during the night and decreases during the day. The backup, Application Server reload, and batch jobs peaks are visible (c.f. Figure 5-14), and the maximum utilization values are close to 50%. This value is low for a maximum, as the utilization is supposed to reach 80 or 90% during high loads on a right-sized system (c.f. sections 2.1 and 3.1.1).

However, Figures 5-16 and 5-17 reveal that the utilization pattern in Figure 5-15 is not representative of the daily utilization of the system, neither in the shape nor in the values.

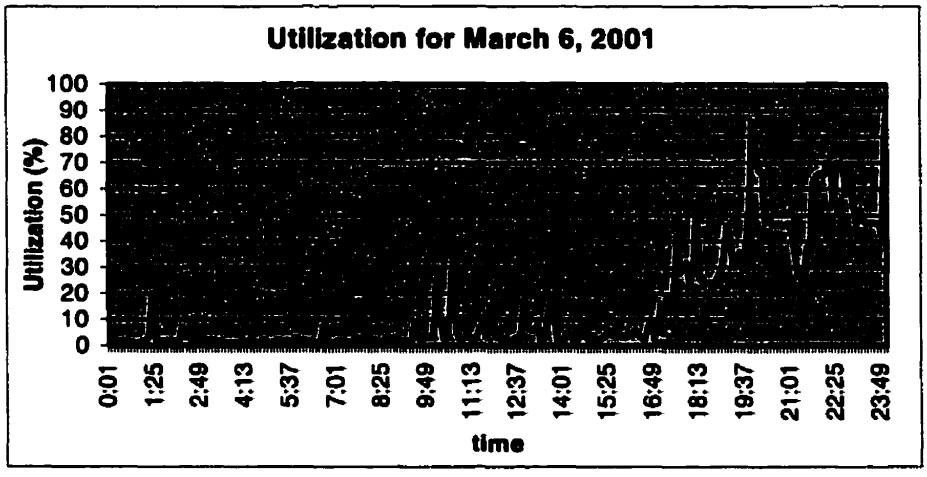

Figure 5-16. Utilization distribution for a weekday (March 6, 2001).

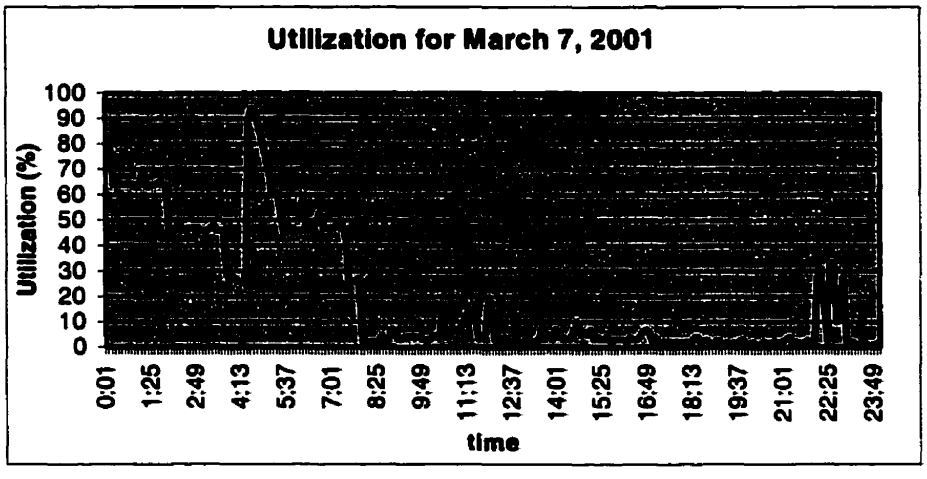

Figure 5-17. Utilization distribution for a weekday (March 7, 2001).

Indeed, looking at Figures 5-16 and 5-17, which are two separate (in this case consecutive) days, we first see that they are totally different; they are actually the complement of each other. Figure 5-16 shows a very low utilization for the first part of the day (generally less than 20%), followed by a significant increase that starts around 5:00pm, reaches almost 90%, and continues all night with different peaks. The explanation for this is probably that a batch job was sent at the end of the workday to run all night.

Figure 5-17 shows a graph that behaves in the opposite manner and which is the continuation of the previous graph (the job started the previous night continues). It starts at high values (80%), then the curve goes down (probably because of the termination of the batch job) to 25%, and then up again for the Application Server reload at around 5:00am reaching a maximum of 93%. At the end of the Application Server reload (before 8:00am), the utilization goes down for the rest of the day and stays around 10% (except for the batch jobs run daily from 10:00pm to 11:00pm that reach 33% utilization).

So by observing Figures 5-16 and 5-17, we understand the utilization pattern in Figure 5-15 (which is a combination of both). We also see that the maximum utilization is not 50%, but rather 90%. This high value is expected from a system under heavy load, however, it should be watched because a utilization that is permanently at 90% is a sign of CPU contention (c.f. section 3.1.1).

Let us now look at the user time and the system time independently (both reported by `sar`). We find a daily average user time (`%usr`) of 15.7% and an average system time (`%sys`) of 2.18%. The maximum `%usr` reaches 90% at heavy utilization and the maximum `%sys` found is 9% (both during the backup time). The `%usr/%sys` ratio computed using `sar` average values is 7. Using `vmstat -s`, we also find a ratio of 7 for the cumulative values (since the machine's last reboot). This ratio value is big, but it guarantees that no CPU cycles are being wasted (c.f. section 2.1.1).

The idle time (`%idle`) reported by `sar` is very high during light utilization (maximum of 95%) and extremely low during heavy utilization (minimum of 2.5%). Over 24 hours, the average idle time is 72.96%, which is quite high and which means that the minimum idle time does not occur very often. Therefore overall performance is probably not suffering (c.f. section 3.1.1).

Because Neptune is a multiprocessor system (with four CPUs), it is interesting to see how each individual CPU is being used and compare its usage to the other CPUs on the system. Figure 5-18 displays this information reported by `mpstat`. The per-processor utilization pattern is similar to the utilization pattern in Figure 5-15. However, some peaks differ, as the average for `mpstat` was taken over different days and the utilization on Neptune is not the exactly the same each day.

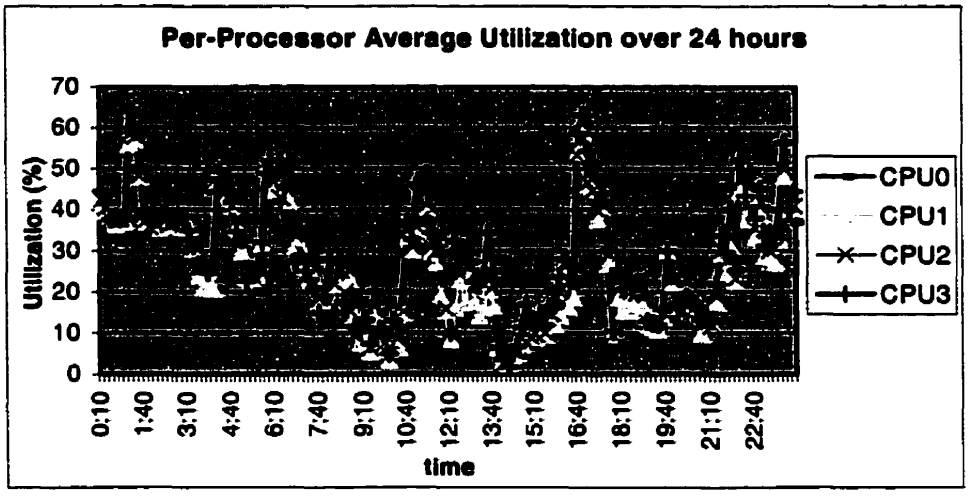

Figure 5-18. Per-processor average utilization over 24 hours.

Overall, the utilization of the four CPUs (CPU 0, 1, 2, and 3) is balanced, so the load is well distributed. Also, the `cpus.se` script, which was run during the whole month of April, reports that all CPUs on Neptune were always online at 400 MHz, as expected.

We now look at the number of processes running on Neptune. It is important to note that Neptune is not a live OLTP system, and therefore the number of processes running cannot be viewed as the workload (c.f. section 2.1.1). Indeed, the number of processes on a test machine is not a good representative of the workload on the system, as many sessions are permanently open, generating numerous – often idle – processes. In such a case, working with the utilization as a reference is much more reliable. Figure 5-19 is only included for illustration.

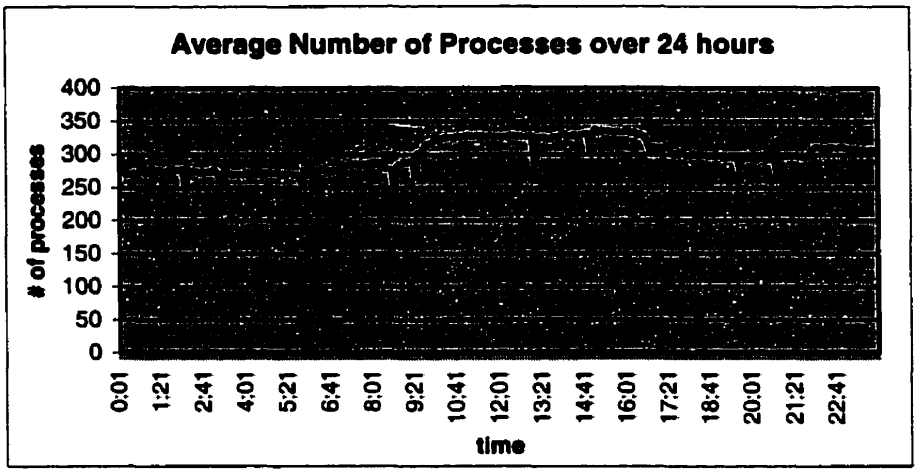

Figure 5-19. Average number of processes running on Neptune over 24 hours.

We observe a fairly stable average number of processes (more or less 300) running on the system the whole day. The reason for this is because many Oracle sessions are open the whole time and some of them are idle. Also the developers tend to stay connected on the system even when they are not working, so several processes are using resources but are actually idle.

Let us now look at the output of `ps` and `top` in order to identify the processes running on Neptune. Overall, we find that the busiest processes on Neptune are the same than those running on Nimbus, the production machine. In addition to `sqloper`, `sooper`, `sowineng` (AppWorx scheduling processes – see section 5.2.1) and the BANNER processes (`forappl` and `fgractg`) several and various Oracle sessions are running on the system all day long.

The process queues are to be checked next. `vmstat` and `sar -q` (which were run every 5 minutes for a month and for a week, respectively) report on the number of processes that are waiting for a CPU time slice. We find a negligible number of runnable processes (`r`), waiting for a CPU time slice. However, `sar -q` shows that the run queue length is often 1, sometimes up to 2, and exceptionally 6. Although non-negligible, these values should not be alarming as long as they are

not frequently large and do not exceed 16 (c.f. section 3.1.1). The number of blocked processes (`b`, reported by `vmstat`) is mostly 0, but often reaches values of 2, 3, and 4 at night. This means that there is some CPU idle time that is treated as time waiting for I/O.

Overall, the CPU power on Neptune seems adequate, as it is being efficiently used during high utilization. However, CPU contention and bottlenecks are possible, since the utilization sometimes tends to be too heavy on the system. The utilization should thus be regularly monitored, and an additional CPU can also help.

5.3.2 Memory Usage

When monitoring memory, the paging and swapping activities are the most important values to be determined. Let us first look at the paging activity on Neptune. We first look at `po` (the number of page-outs, reported by `vmstat`) to find out whether there is excessive paging happening on the system.

The average number of page-outs, computed by averaging the samples (taken every 5 minutes) of four days in March, is 492.49 kB/sec, which is significantly large. Also, even the minimum po value is very large (75.75 kB/sec), and the maximum value is huge (7492.75 kB/sec). The 24-hour distribution of these values is shown in Figure 5-20.

The page-out activity is very high all day long (mostly between 75.75 and 1000 kB/sec). There are two much higher peaks clearly observed. The first peak is around 9:00am and is probably due to the logging in of the developers on the system (note that each Oracle session uses up to 115 MB of memory), and the second (maximal) peak is at around 5:00pm and is due to the shut-down and restart of the `RTRNG` database. The `RTRNG` database is the database used for the

BANNER training courses and is stopped, refreshed (all database files are replaced), then restarted every day at 4:45pm.

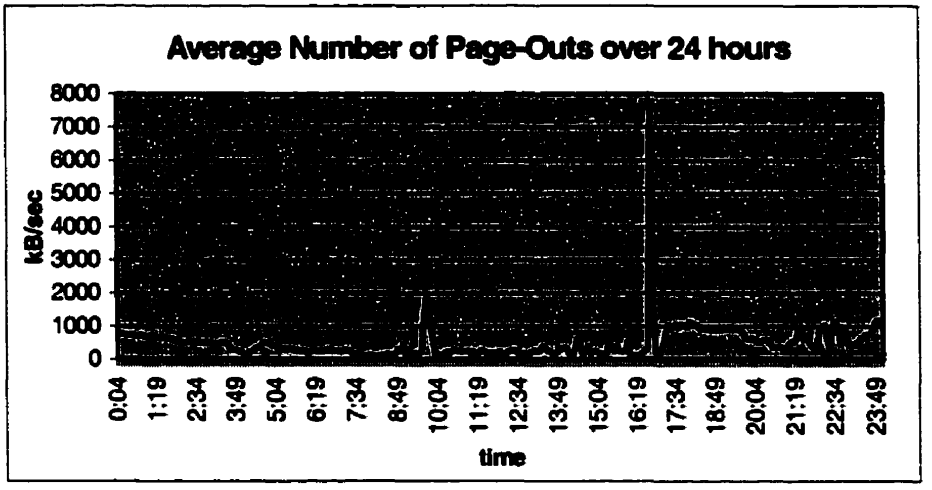

Figure 5-20. Average number of page-outs during 24 hours.

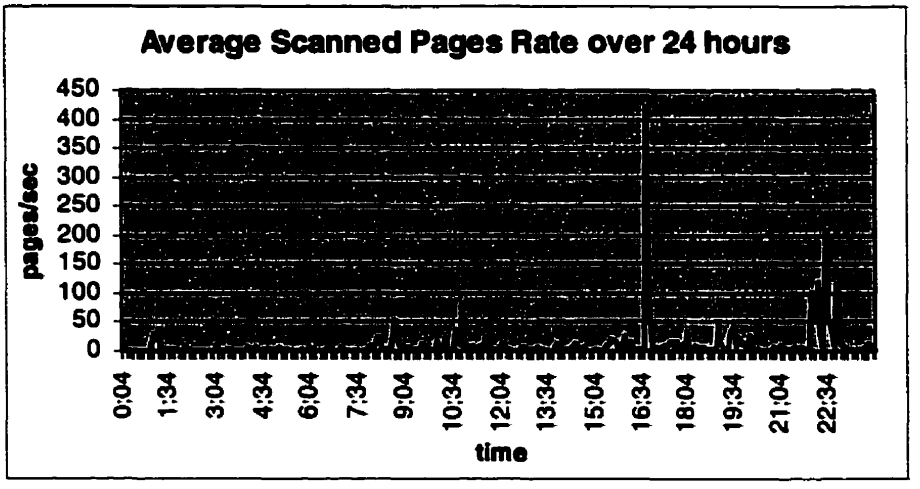

Figure 5-21. Average scanned-pages rate during 24 hours.

Such a continuously high paging activity affects system performance and is possibly due to a serious memory shortage. The page-stealing daemon scanning rate (`sr`, reported by `vmstat`) also shows some activity. We find a rate mostly under 50 pages/sec (in a 24-hour period), with a maximum reaching 421 pages/sec at around 5:00pm (see Figure 5-21). Although these `sr` values are not

extremely high, they do correspond to the `po` values (note the similar patterns in Figures 5-20 and 5-21) and therefore indicate a lack of memory on the machine.

Now let us look at the swapping activity. The values found for `w` (the number of swapped-out processes, reported by `vmstat`) are quite variable: in a sample with 5-minute measurements during a month, some days have constant zero values for `w`, others have a higher values reaching a maximum of 317 swapped-out processes, which is very high. The `w` values were usually found to be constant from one day to the other (more specifically from one day at 5:00pm to the next day at 5:00pm, probably associated with the batch jobs started at that time and that continue until the following day).

The deficit paging parameter (`de`) is to be verified next. Using `vmstat`, we find that the `de` parameter has a very variable activity day to day *and* within a day. However, in general it seems to have high values mostly during the workday. These values can get very high (highest value observed in 3 days of measurements was close to 30,000 kB), meaning that excessive swapping is happening on the system.

Let us check if there is enough swap space configured on the machine. The `freeswap` counter (in `sar -r`'s output) reports a minimum available swap space of 3.8 GB (and a maximum of 5 GB), which is very large. So the excessive swapping happening on the system are most probably due to a lack of physical memory.

The memory on Neptune seems to be insufficient for the current workload, because the system is sometimes desperately paging and swapping processes. The current physical memory is insufficient for all the Oracle instances running. Adding more memory will most probably help improve the performance (on a short time basis).

5.3.3 Disk Subsystem Usage

The disk I/O activity is the next aspect to be studied. The `iostat` utility reports most of the information required. Neptune has 10 physical disks with no volume manager. One disk is for the operating system and the other nine are for the database, the archive logs, and the AppWorx scheduler.

The operating system active disk partitions are six: `c0t0d0s0 (/)`, `c0t0d0s3 (/var)`, `c0t0d0s4 (/home)`, `c0t0d0s5 (/opt)`, `c0t0d0s6 (/usr)`, and `c0t0d0s7 (/archive)`. The activity of all these partitions is similar: the disk utilization is almost always under 1% (excluding a few peaks, such as at the beginning of the backup process around 1:00am, where the utilization reaches a maximum of 11.5%) and the throughput is mostly below 0.5 operations/sec (with a few exceptional peaks, mainly at the start of the backup around 1:00am, reaching a maximum of 13.45 operations/sec). These values are very low and show that the operating system disk is underused.

Figure 5-22 is an example that shows the activity of one of the operating system partitions. The pattern observed shows a utilization that increases during the backup time (one peak), then during the workday (with several peaks from 9:00am to 5:00pm), and at night during the processing of the batch jobs (a few peaks from 10:00pm to 11:00pm). The throughput appears to be almost constant throughout the day with practically just one peak (at the backup time).

Over all the partitions of the operating system disk, the queue length and response time calculated were negligible (with values almost always at zero), meaning that practically no processes are waiting for I/O from that disk at one time. However, the `iomonitor.se` tool (run during the last nineteen days of April) reported many complaints (83 exactly) saying that this disk was slow. Note that the `IOSLOW` environment variable was set to 90.0 (c.f. section 3.1.3). These complaints imply the existence of a possible configuration problem.

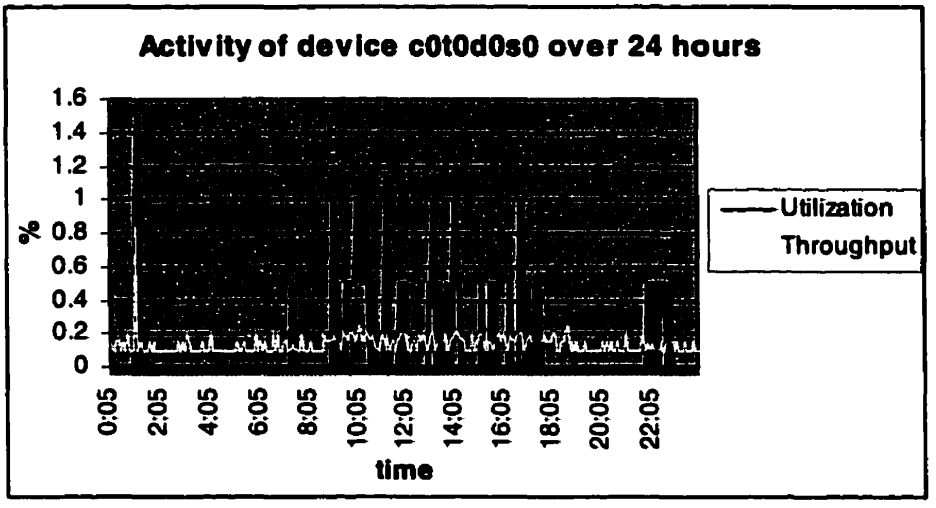

Figure 5-22. Activity of an operating system partition over 24 hours.

The activity observed on the `/opt/appworx` filesystem (device `c2t3d0s6`) is low. The activity pattern shows a utilization and throughput almost constantly lower than the value of 1.55. The only exceptions are the peaks observed at 12:00am (due to the batch jobs), 1:00am (because of the backup) and around 5:00pm (probably due to the closing of the different sessions), reaching a maximum of 7.5% utilization and 12.1 operations/sec throughput.

The `/archive/arch` (device `c2t4d0s6`) filesystem's activity pattern shows successive peaks between the values of 2 and 8 (for both utilization and throughput). Only two major peaks (reaching a maximum of 23.5% utilization and 28.9 ops/sec throughput) are observed: the first happens at around 6:00pm (probably due to the hourly backup of the archive logs and is more active at 6:00pm, possibly because more data is created on the machine between 5:00pm and 6:00pm) and the second peak is around 11:00pm (batch jobs sent). The `iomonitor.se` tool generated several complaints (24 in nineteen days) stating that this disk was slow. However, the complaints were all made between 12:00am and 6:00am, which is mostly backup time and therefore does not imply the existence of a possible configuration problem.

The queue length and response time values computed for `c2t3d0s6` and `c2t4d0s6` were also found to be negligible.

The remaining disks are all database disks. Filesystems `/bdata1`, `/bdata2`, and `/bdata3` (devices `c0t1d0s6`, `c0t2d0s6`, and `c0t3d0s6` respectively) are by far the most active, with a maximum utilization reaching 68.5% and a maximum throughput close to 95 operations/sec. These values show that the disks are being efficiently used. Figure 5-23 shows the activity of one of those disks during 24 hours; the other two disks have a very similar pattern.

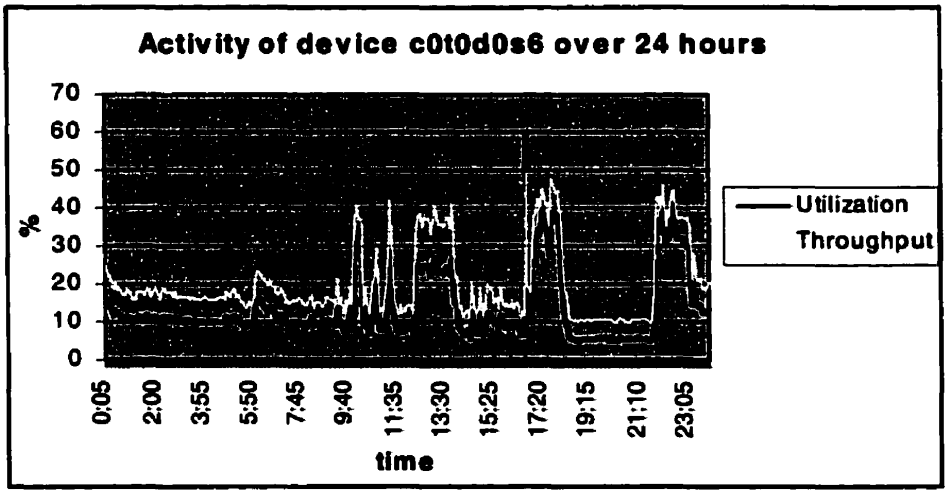

Figure 5-23. Activity of device `c0t2d0s6` (`/bdata2`) over 24 hours.

The utilization and throughput first increase during the workday with several peaks. A relatively higher peak is observed at the end of the workday (probably due to the closing of the sessions and the committing of the databases). The last peak of the day occurs at 10:00pm and goes on until 11:00pm (which is during the batch jobs).

The queue length and response time of these three disks was slightly higher than the other disks. The queue length values computed varied between 0.1 and 0.8 all day. The response time was lower, with values under 0.03.

Disks `c3t3d0s6` (`/bdata4`) and `c5t3d0s6` (`/bdata7`) have a very similar and low activity. The values of utilization and throughput are constant all day and remain under the value of 2 (for both parameters) for the two disks. Only one peak is observed at around 10:30am (maximum utilization of 12.5% and maximum throughput of 31 operations/sec) and is probably due to some job executed by the developers.

The last two active partitions are `c3t4d0s6` (`/bdata5`) and `c4t3d0s6` (`/bdata6`). These have similar utilization and throughput patterns that are almost constant throughout the day. The utilization is permanently between 1.5 and 2%, and the throughput is almost stable between 3 and 3.5 operations/sec.

The response time and queue length values computed for `/bdata4`, `/bdata5`, `/bdata6`, and `/bdata7` were constantly at zero, meaning that no processes at all are waiting for I/O from that disk at one time.

Let us now check the disks space usage using the information reported by `df -k` (in March and April). In general, the usage of the different filesystems seems rather stable, except for three of them. The two filesystems `/archive/arch` and `/opt/appworx` show a very fluctuating usage due to the database archive logs, and the `/home` filesystem (in the operating system disk) has tendency to fill up because they contain the output of the batch jobs. These disks should be regularly verified and cleaned-up to avoid contention problems.

The disks on Neptune do not show any major problem. Some are underutilized and some are efficiently utilized. No change is necessary for the time being, but a redistribution of the load could be useful.

5.3.4 Network Activity

To determine the network activity on the system, we first look at the information reported by `netstat -i`. Figure 5-24 shows the average number of incoming and outgoing network packets on Neptune's interface in a 24-hour period. The values were computed by averaging corresponding (5-minute) samples of 4 days together. Because the network activity on Neptune is very variable day to day (the standard deviation values calculated were *very* large – up to 100,000), the graph in Figure 5-24 is not representative of the network load and is only included for illustration.

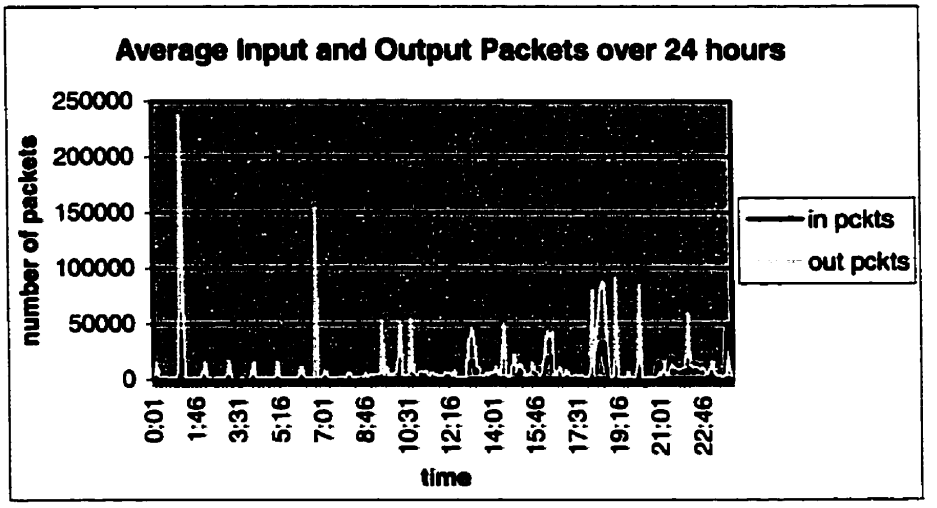

Figure 5-24. Average input and output packets over 24 hours.

Several peaks are observed in Figure 5-24. The first peak at 1:00am is an output packets increase reaching 237,622 (the input packets only increase at that point to 43,220). This is due to the backup, which is done on a remote machine across the network. The next significantly high peak occurs shortly after 6:00am, and is due to the establishment of several Oracle sessions (simultaneously) for the Application Server reload (c.f. Figure 5-14). At that point, both input and output packets reach 155,314. The following peaks are variable and depend on the day (i.e. on what the developers are doing). Throughout the day, incoming and

outgoing packets are very close, showing normal activity. The last peak occurs around 10:00pm and is due to the batch jobs being sent.

In a week of 5-minute measurements of `netstat -i`, no input or output errors were found and the collisions rate was constantly at 0%.

Next, we check the TCP data reported by `netstat -s` (run every 20 minutes during the last two weeks of March). The number of currently established connections (`tcpCurrEstab`) observed is quite variable, with values from 3 to 115, and is not consistent day to day. The number of active and passive opens remain close overall with maximum values of 1,293,448 (`tcpActiveOpens`) and 1,297,981 (`tcpPassiveOpens`), which are relatively large. The number of connections dropped due to a full listen queue (`tcpListenDrop`) started at zero in the first few days of the measurements but quickly increased to reach a maximum of 6 at the end of the measurement period. Therefore, the size of the listen queue might need to be increased.

Overall, the network on Neptune seems to be performing efficiently and the traffic seems to be running smoothly.

5.4 Performance Study of Poseidon

In this section, Poseidon – the machine running Oracle Application Server – is analyzed. This Web server is already live and in use by some BANNER clients. However, its major use will be when the Student Information System goes live in the near future.

The tools used in this analysis are almost the same as those used in the previous sections (all described in section 3.1), and are listed in Appendix D. The measurements were taken mainly during the months of March and April 2001.

In the following subsection the performance of Poseidon is studied in four parts: the CPU usage (5.4.1), the memory usage (5.4.2), the disk usage (5.4.3), and the network activity (5.4.4).

5.4.1 CPU Usage

The first thing to look at is the load average, as it gives an idea about the overall system usage. Figure 5-25 shows the 24-hour load average (reported by `uptime`) computed over 6 weekdays. The sampling interval is 5 minutes.

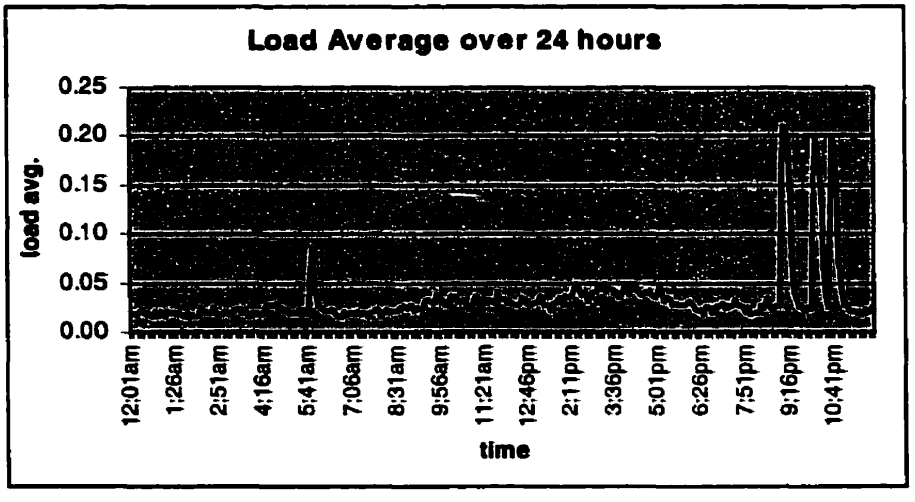

Figure 5-25. Load average on Poseidon over 24 hours.

Throughout the day, the average number of jobs in the run queue (i.e. the load average) is significantly low. It is almost always under 0.05 and the only two peak times happen during the Oracle Application Server reload around 6:00am, and at night (probably during some development work associated with the Web servers running on Poseidon). Even these peaks do not reach 0.25.

Let us now look at the CPU utilization over 24 hours, using `sar`'s output. Figure 5-26 shows the average CPU utilization computed by averaging corresponding (6-minute) samples of several weekdays.

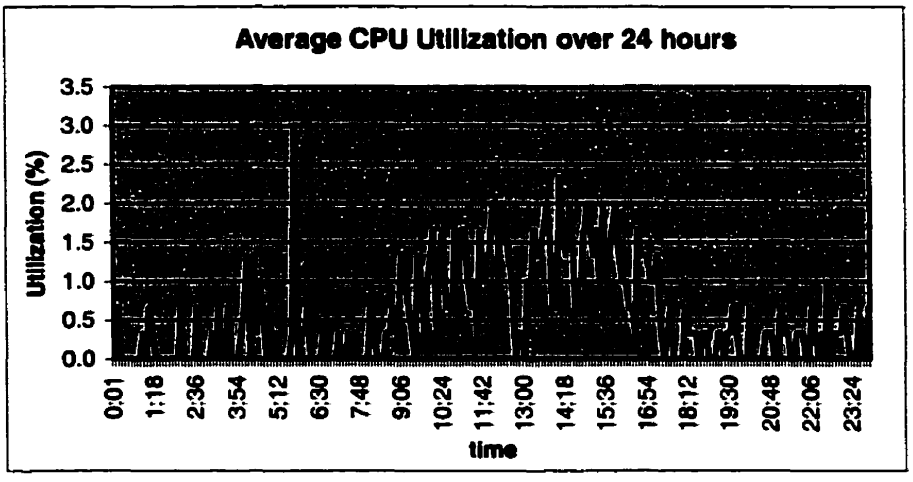

Figure 5-26. Average utilization over 24 hours.

The utilization pattern shows that the highest usage occurs at the Application Server reload time (around 6:00am) and during the workday (between 9:00am and 5:00pm exactly). The utilization starts rising at 9:00am and continues until lunch time. Between 12:30pm to 1:30pm, a significant drop is observed. Shortly after 1:30pm, the utilization starts increasing again (maximum reached around 2:00pm) then starts decreasing later in the afternoon until 5:00pm, when it stabilizes between 0 and 0.5% until the next morning.

Throughout the day, and including all the peaks, the CPU utilization observed is extremely low (maximum of 3%). Figure 5-27 shows a quite interesting comparison of user time and system time. It appears that system time is almost always greater than user time. The only exceptions observed are the two maximal peaks at 6:00am (during the Application Server reload) and at 2:00pm (at the highest utilization of the server).

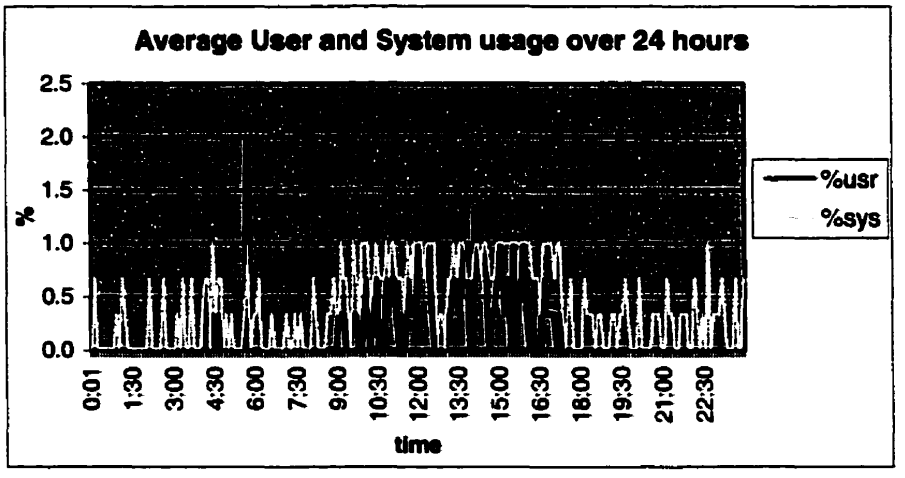

Figure 5-27. Average user and system usage over 24 hours.

The reason for this is that the machine is so underused that the operating system itself appears to be the major CPU-resource consumer. If we look at the numbers, we see that the highest system usage never goes beyond 1%. Therefore, the fact that the system time is almost the double of the user time should not be alarming in this case.

The idle time reported by `sar` (`%idle`) is, of course (by consequence), very large. The minimum value found in four days of measurements is 95%, which means that Poseidon currently has practically no work to do.

Let us now look at the utilization of every process individually. Figure 5-28 shows the per-processor utilization (reported by `mpstat`) over 24 hours.

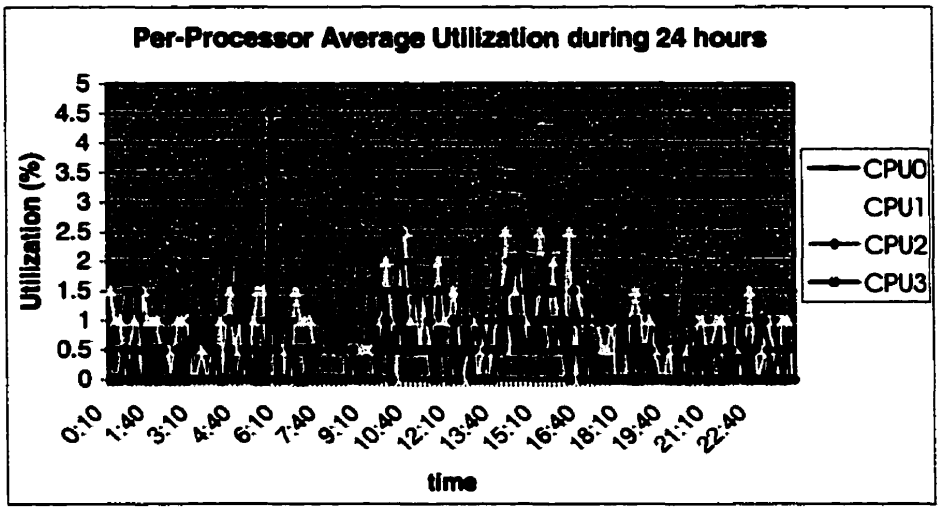

Figure 5-28. Per-processor average utilization over 24 hours.

On the graph, the utilization of the four CPUs (CPU 0, 1, 2, and 3) does not seem balanced. But in fact, *because* the utilization is so low, the scale used here (y-axis) is very precise and thus shows the smallest details of the pattern (which did not happen in the previous analyses in sections 5.2 and 5.3). The overall utilization *is* actually balanced across the CPUs on Poseidon. Also, all four CPUs are working as expected: `cpus.se`, which was run during the second part of March and all of April, reported the four processors always online at 400 MHz.

The next step is to look at the number of processes running on the system. Figure 5-29 shows the average number of processes running on Poseidon over 24 hours. Because the machine's utilization is so low, we cannot view this pattern as the workload pattern. The number of processes is almost constant throughout the day with only a few peaks probably due to the Web development work on the machine. Poseidon actually has eight Web servers running, and only one is used for the production (the others are for development).

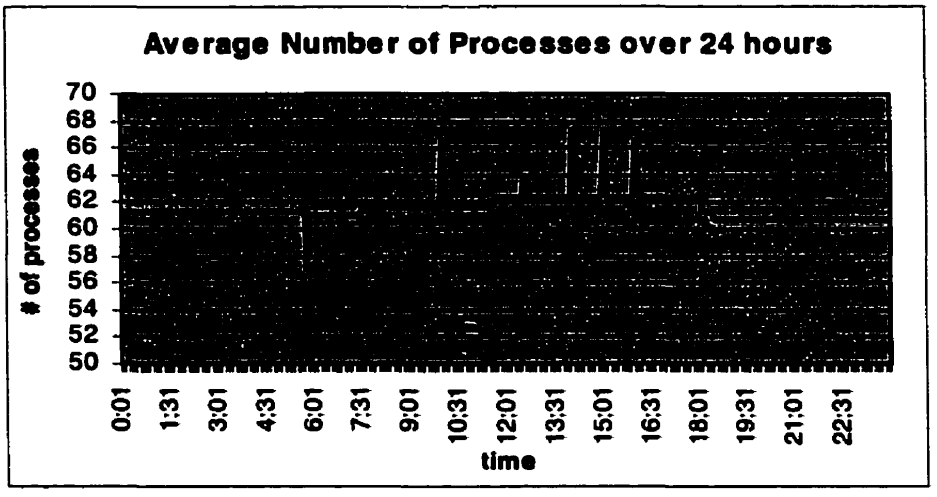

Figure 5-29. Average number of processes on Poseidon over 24 hours.

Using `ps` and `top`, we now look at these processes to identify them. The following processes are running all day on Poseidon: `oassrv`, `oasoorb`, `otsfacsrv`, and `wrksf` (which are Oracle Application Server processes); and `oraweb` (the Oracle Web listener). The backup process `dsmc` is also reported.

Let us now check the process queues (using `vmstat` and `sar -q`) to see whether there are any processes waiting for a CPU time slice. During three weeks of 5-minute measurements, the number of runnable processes `r` was constantly zero. The `sar -q` utility reported a slightly larger queue length (mostly 0 or 1, and exceptionally 3 or 5), but this can be ignored because the values are below 16 (c.f. section 3.1.1). Also, no CPU idle time is treated as time waiting for I/O, as the number of blocked processes `b` (reported by `vmstat`) was continuously 0 in all the measurements.

Overall, the current workload on Poseidon is too light for the existing CPU power because the processors are mostly idle. The expected growth in the user population should make the system use more (efficiently) its resources.

5.4.2 Memory Usage

The first thing to be verified when monitoring memory is the paging and swapping activity to determine whether it is excessive. Let us first look at the paging activity on Poseidon. The average number of page-outs (`po`) reported by `vmstat` (which was run every 5 minutes during three weeks) is 0 kB/sec. Also, the page-stealing daemon scanning rate (`sr`) is 0 pages/sec across all samples. This means that there is absolutely no paging activity happening on the system. It also means that the system is not making use of the virtual memory (c.f. section 2.1.2).

Let us now look at the swapping activity reported by `vmstat`. In the three weeks' measurements, the number of swapped-out processes (`w`) is constantly at 0. The deficit paging parameter (`de`) is also 0 kB across all samples. The configured swap space on the machine varies between 1.4 GB and 1.5 GB. This variation is probably due to the use of the `tmpfs` (temporary filesystem), since `tmpfs` and swap share space.

With the current workload on Poseidon, the physical memory seems to be more than sufficient, as there is practically no paging nor swapping happening on the system. When the system will have a heavier workload, the paging and swapping activity will surely increase, but it is difficult to predict whether there will be a shortage of memory.

5.4.3 Disk Subsystem Usage

We now look at the disk I/O activity. Poseidon only has two disks of which one is currently not being used (`c0t1d0s2`). The operating system disk `c0t0d0s2` has five active partitions: `c0t0d0s0` (`/`), `c0t0d0s3` (`/var`), `c0t0d0s4` (`/home`), `c0t0d0s5` (`/opt`), and `c0t0d0s6` (`/usr`).

Using the output from `iostat`, we observe the activity on each one of these partitions. The first thing we see is that the activity is very low and similar over all partitions. The utilization is always at 0.1% (or very close), except at 4:30am (when the backup starts), where it reaches a maximum value of 5%. The throughput varies more, but is still very low. The average throughput value is between 0.1 and 0.2 operation/sec, except during the backup, where it goes up to a maximum of 9.15 operations/sec for one partition (partition `c0t0d0s5`, or filesystem `/opt`). Figure 5-30 shows the utilization and throughput for one of these partitions/filesystems.

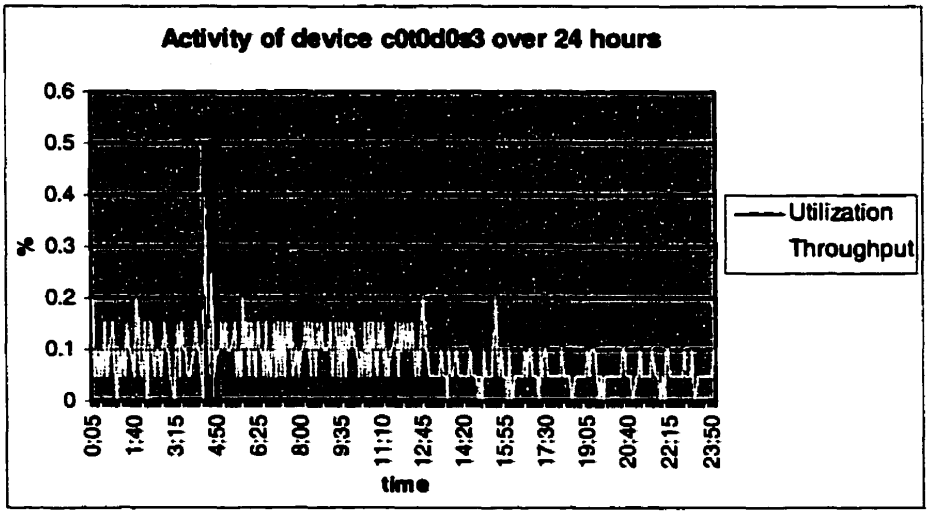

Figure 5-30. Activity of an operating system partition over 24 hours.

Although the utilization and throughput patterns are similar across all partitions, the resemblance is even more notable between two pairs. Filesystems `/` (root) and `/usr` have an almost identical activity, and filesystems `/var` and `/home` show a very close pattern. Filesystem `/opt` has a slightly different behavior, but is a little more active than all the others.

The total queue length and response time values was computed for all partitions. The results obtained were all zero (or very close) even during the (slightly)

heavier utilization. Also, the number of blocked processes (`b`, reported by `vmstat`) was constantly zero.

Looking at the disk space usage (using `df -k`), we see that no filesystem is significantly increasing in size. The usage is very stable, as the activity on the machine is very low.

Overall, the disk subsystem usage on Poseidon is very light. The disk resources should be sufficient for a (future) heavier workload and usage.

5.4.4 Network Activity

We now look at the network activity on Poseidon, using the `netstat -i` utility. Figure 5-31 shows the average number of incoming and outgoing packets on the system's interface over 24 hours. The values were computed by averaging the corresponding samples of several weekdays in the second part of April.

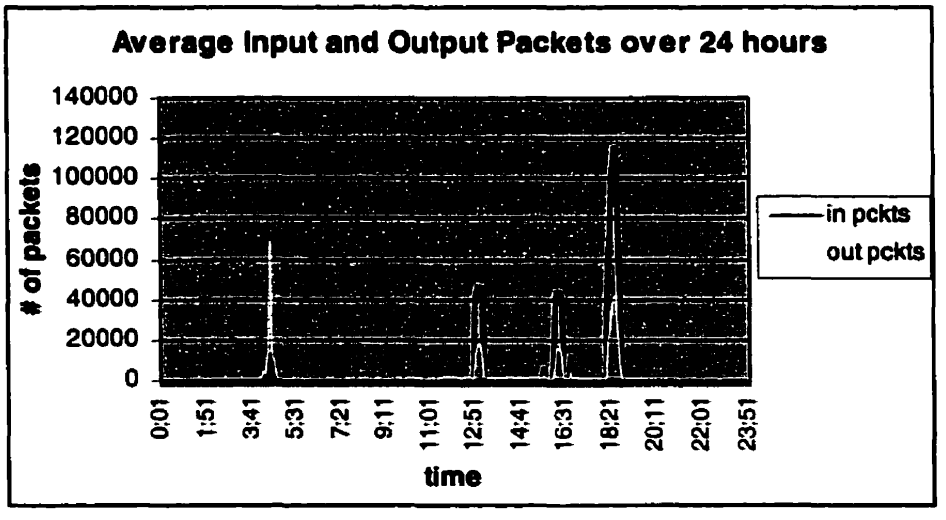

Figure 5-31. Average input and output packets over 24 hours.

Throughout the day, the network activity is very low and only a few peaks are observed. In all daily measurements, two major peaks appear consistently: at

4:30am during the backup, and at 6:30pm, where the peak reaches 115,570 packets (this last peak is not attributable to any regularly scheduled system process). Note that in the first peak, there are much more outgoing packets (since the machine is being backed up to a remote mainframe), and in the second peak the number of incoming packets is much greater.

The other peaks observed in the network activity pattern are lower and are not consistent day to day. They are probably due to the development work related with the Web servers running on Poseidon.

Over all samples, no input or output errors were observed and the collision rate remained at 0%. This is expected, since the traffic on the machine is significantly light.

Let us now look at the TCP information reported by `netstat -s` (run every 20 minutes during the last week of March and the first week of April). The number of currently established connections (`tcpCurrEstab`) is more or less consistent every day with a minimum of 131 and a maximum of 245. The maximum numbers of active and passive opens (`tcpActiveOpens` and `tcpPassiveOpens`) observed are 197,205 and 74,313, resprectively. The number of connections dropped because of a full listen queue (`tcpListenDrop`) remained at zero across the samples, so no changes in size need to be made to the listen queue for now [CPS98].

Again, the network on Poseidon is a very lightly used resource because of the current workload on the machine. As the network activity and traffic increase, further measurements should be taken in order to assess the state of the network. Also, it is important to monitor the other machine's resources as well, when the utilization will increase, in order to determine the performance under the real workload.

Chapter 6

Conclusion

6.1 Summary

The purpose of this research was to present performance analysis of computer systems, highlighting its importance. Indeed, any computer system is subject to performance degradation (or other performance-related problems) with time if its basic resources are not regularly monitored and corrective actions taken if necessary. Performance analysis prevents performance-related problems, which directly affect productivity.

The basic resources to be monitored on a system were described in a performance-tuning approach, and numerous monitoring tools (commercial and non-commercial tools) and techniques for performance measurement were presented.

Two new major systems at McGill University were analyzed in this thesis: the ALEPH library management system and the BANNER finance system. The result of the analysis showed varying activity and usage across the different machines.

The ALEPH system's machine Aleph revealed to be oversized in hardware, as the current utilization and workload on the system are relatively low[1]. However, this usually implies the absence of resource contention problems and bottlenecks. Also, the extra capacity can support special projects, such as the testing and

[1] Note that the measurements and analysis of Aleph covered the busiest period of the year.

deployment of new software releases[2]. Moreover, as the user population of ALEPH increases, the current system should be able to handle a heavier (future) workload.

The BANNER system's machines Nimbus, Neptune, and Poseidon had different behaviors. The production machine Nimbus was found to be slightly oversized in the CPU and disk resources, with a relatively light network traffic. However, the memory subsystem is experiencing some problems when the workload increases. Adding physical memory on Nimbus should help relieve this problem.

The development machine Neptune showed a quite complex and mostly inconsistent behavior and was found to be somewhat undersized[3]. The major problem was with memory, as the current physical memory configured is unable to handle the multiple Oracle instances that are running on the system. Additional memory is clearly needed on Neptune. The other resources' power and size can also be increased, as this would probably help performance too.

The Web server (or Oracle Application Server) Poseidon is currently oversized. In fact, the system was sized for a workload that hasn't yet materialized. Indeed, since the Student Information System (SIS) is not yet live, it is difficult to determine its performance under the 'real' workload. Another set of measurements will be needed when the machine will be running the SIS.

6.2 Future Work

In this thesis, we looked at performance from the computer's view. Studying the performance from the user side (i.e. from the client's perspective) is another interesting task. Moreover, as client machines are generally running MS

[2] In fact, this work is currently being done on Aleph, and the machine is now supporting two releases of the ALEPH software.
[3] In fact, in the capacity planning phase, there was no plan to have twelve databases running on Neptune.

Windows, it would also be interesting to analyze the performance of that operating system (or any other operating system).

Another (more complex) analysis would be to determine the performance of Oracle on the current system. This analysis requires a very deep understanding of the Oracle database system, as well as the platform on which it is running.

One could also concentrate on the network performance between client machines and server interfaces, and compare the performance of different network interfaces or topologies.

When source code is available, application performance analysis is another useful task in which the application running on a system is analyzed (in terms of efficiency of code, etc.) independently of the platform. In such an analysis, mastering the software application in all its aspects is necessary in order to find ways to improve its performance. Another interesting task would be to determine the performance of an application running under different operating systems.

Running different monitoring tools on a system and comparing their operation and results is yet another challenging task. One could use character-based tools, tools having a GUI, commercial tools, etc., resulting in a comprehensive overview of tested monitoring tools.

Appendix A

List of tools run on Aleph

Command/Script	Start Date	Stop Date	Output/File location (on Aleph)	Additional Information
se virtual_adrian.se	Apr. 26, 2000	May 20, 7:03	/var/adm/sa/monitor.log	run by administrator, sampling every 30 seconds
vmstat 300	Feb. 19, 9:31	Feb. 21, 5:36	/home/michele/Stats2001/vmstat300.19FE	run in bg (background), at 5-minute intevals (300)
uptime	Feb. 19, 10:20	Apr. 2, 9:10	/home/michele/Stats2001/uptime.19FE	run with crontab, sampling every 10 minutes
se nproc.se	Feb. 19, 10:30	Apr. 5, 23:50	/home/michele/Stats2001/nproc.19FE	run with crontab, sampling every 10 minutes
sar 30 10	Feb. 26, 9:25	Mar. 12, 9:25	/home/michele/Stats2001/sar.26FE	run with crontab, every 6 minutes (runs for 5 minutes)
vmstat 300	Feb. 27, 10:09	Mar. 14, 16:00	/home/michele/Stats2001/vmstat.27FE	run in bg, at 5-minute intevals
mpstat 300 2	Feb. 28, 0:05	Mar. 19, 8:50	/home/michele/Stats2001/mpstat.28FE	run with crontab, every 10 minutes (avg. over 5 min.)
se cpg.se [head and tail]	Mar. 1, 14:00	Apr. 4, 14:01	/home/michele/Stats2001/cpg.01MR	run with crontab, once a day at 14:00
top	Mar. 5, 0:05	Mar. 12, 9:05	/home/michele/Stats2001/top.05MR	run with crontab, every 15 minutes
/usr/ucb/ps uaxw | head	Mar. 5, 10:00	Mar. 12, 10:00	/home/michele/Stats2001/ucb_ps.05MR	run with crontab, every 15 minutes
se cpus.se	Mar. 5, 12:00	May 1, 14:01	/home/michele/Stats2001/cpus.05MR	run with crontab, every 2 hours
df -k	Mar. 8, 17:00	Apr. 30, 17:00	/home/michele/Stats2001/df-k.08MR	run with crontab, once a day at 17:00
netstat -s	Mar. 12, 0:01	Mar. 26, 9:01	/home/michele/Stats2001/netstat-s.12MR	run with crontab, every 20 minutes
netstat -i 300	Mar. 12, 9:01	Mar. 26, 9:11	/home/michele/Stats2001/netstat-i.12MR	run in bg, at 5-minute intevals
sar 30 10	Mar. 19, 0:01	Apr. 3, 16:42	/home/michele/Stats2001/sar.19MR	run with crontab, every 6 minutes (runs for 5 minutes)
iomonitor*	Mar 19, 0:00	Apr. 10, 15:13	/home/michele/Stats2001/iomonitor.19MR	run in bg, at 15-second intervals
vmmonitor*	Mar. 23, 0:00	Apr. 11, 16:55	/home/michele/Stats2001/vmmonitor.23MR	run in bg, at 15-second intervals
uptime	Mar. 28, 0:01	May 1, 15:06	/home/michele/Stats2001/uptime.28MR	run with crontab, every 5 minutes
iostat -xPnce 300	Apr. 4, 0:00	Apr. 5, 11:25	/home/michele/Stats2001/iostat-xPnce.04AR	run in bg, at 5-minute intervals
sar 300 288	Apr. 4, 0:00	Apr. 4, 23:55	/home/michele/Stats2001/sar.04AR	run in bg, at 5-minute intervals (for 24 hours)
se cpg.se [head and tail]	Apr. 5, 12:00	May 1, 14:01	/home/michele/Stats2001/cpg.05AR	run with crontab, 3x/day at 12:00, 14:00, and 16:00
se nproc.se	Apr. 6, 0:01	May 1, 15:06	/home/michele/Stats2001/nproc.06AR	run with crontab, every 5 minutes
/usr/ucb/ps uaxw | head	Apr. 6, 0:01	Apr. 10, 18:06	/home/michele/Stats2001/ucb_ps.06AR	run with crontab, every 5 minutes

se vmstat.se 300	Apr. 6, 0:01	Apr. 11, 8:51	/home/michele/Stats2001/vmstat.se.06AR	run in bg, at 5-minute intevals
iomonitor*	Apr. 11, 0:00	May 1, 14:38	/home/michele/Stats2001/iomonitor.11AR	run in bg, at 30-second intervals
vmmonitor*	Apr. 12, 0:00	May 1, 15:09	/home/michele/Stats2001/vmmonitor.12AR	run in bg, at 30-second intervals
vmstat -s	Apr. 17, 0:01	Apr. 24, 12:21	/home/michele/Stats2001/vmstat-s.17AR	run with crontab, every 5 minutes
netstat -i 300	Apr. 17, 0:01	Apr. 24, 12:21	/home/michele/Stats2001/netstat-i.17AR	run in bg, at 5-minute intevals
sar -g 60 5	Apr. 17, 0:01	Apr. 24, 12:24	/home/michele/Stats2001/sar-g.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
sar -r 60 5	Apr. 17, 17:43	Apr. 24, 12:24	/home/michele/Stats2001/sar-r.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
sar -q 60 5	Apr. 17, 17:43	Apr. 24, 12:24	/home/michele/Stats2001/sar-q.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
netmonitor*	Apr. 17, 14:25	Apr. 24, 12:00	/home/michele/Stats2001/netmonitor.17AR	run in bg, at 30-second intervals
se live_test.se	Apr. 18, 0:00	Apr.19, 8:57	/home/michele/Stats2001/live_test.18AR	run in bg, at 30-second intervals
iostat -xPnce	Apr. 27, 0:00	Apr. 27, 23:55	/home/michele/Stats2001/iostat-xPnce.27AR	run in bg, at 5-minute intervals

Appendix B

List of tools run on Nimbus

Command/Script	Start Date	Stop Date	Output File Location (on Nimbus)	Additional Information
sar 30 10	Mar. 12, 0:01	Apr. 2, 9:36	/home/michele/Stats2001/sar.12MR	run with crontab, every 6 minutes (runs for 5 minutes)
uptime	Mar. 12, 0:00	Apr. 2, 9:30	/home/michele/Stats2001/uptime.12MR	run with crontab, every 10 minutes
top	Mar. 12, 0:05	Mar. 19, 9:05	/home/michele/Stats2001/top.12MR	run with crontab, every 15 minutes
df -k	Mar. 12, 17:00	Apr. 30, 17:00	/home/michele/Stats2001/df-k.12MR	run with crontab, once a day at 17:00
se nproc.se	Mar. 15, 0:00	Apr. 11, 23:50	/home/michele/Stats2001/nproc.15MR	run with crontab, every 10 minutes
se cpus.se	Mar. 15, 0:01	May 1, 14:01	/home/michele/Stats2001/cpus.15MR	run with crontab, every 2 hours
vmstat -s	Mar. 15, 0:30	Apr. 15, 23:30	/home/michele/Stats2001/vmstat-s.15MR	run with crontab, every hour
netstat -s	Mar. 19, 0:01	Apr. 5, 23:41	/home/michele/Stats2001/netstat-s.19MR	run with crontab, every 20 minutes
mpstat 300 2	Mar. 21, 0:05	Apr. 5, 14:45	/home/michele/Stats2001/mpstat.21MR	run with crontab, every 10 minutes (avg. over 5 minutes)
uptime	Mar. 28, 0:01	May 1, 15:16	/home/michele/Stats2001/uptime.28MR	run with crontab, every 5 minutes
iostat -xPnce 300	Apr. 4, 0:00	Apr. 5, 12:00	/home/michele/Stats2001/iostat-xPnce.04AR	run in bg, at 5-minutes intevals
sar 300 288	Apr. 4, 0:00	Apr. 4, 23:55	/home/michele/Stats2001/sar.04AR	run in bg, at 5 -minutes intervals (for 24 hours)
se cpg.se [head and tail]	Apr. 4, 10:00	May 1, 15:01	/home/michele/Stats2001/cpg.04AR	run with crontab, 4x/day at 10:00, 14:00, 15:00, and 16:00
/usr/ucb/ps uaxw | head	Apr. 5, 0:00	Apr. 10, 23:50	/home/michele/Stats2001/ucb_ps.05AR	run with crontab, every 10 minutes
se nproc.se	Apr. 12, 0:01	May 1, 15:16	/home/michele/Stats2001/nproc.12AR	run with crontab, every 5 minutes
iomonitor*	Apr. 12, 0:00	May 1, 6:00	/home/michele/Stats2001/iomonitor.12AR	run in bg, at 30 -seconds intervals
vmstat 300	Apr. 12, 0:01	Apr. 24, 19:11	/home/michele/Stats2001/vmstat.12AR	run in bg, at 5-minutes intevals
netstat -i 300	Apr. 17, 0:01	Apr.24, 13:24	/home/michele/Stats2001/netstat-i.17AR	run in bg, at 5-minutes intevals
sar -g 60 5	Apr. 17, 0:01	Apr.24, 13:24	/home/michele/Stats2001/sar-g.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
sar -r 60 5	Apr. 17, 0:01	Apr.24, 13:24	/home/michele/Stats2001/sar-r.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
sar -q 60 5	Apr. 17, 0:01	Apr.24, 13:24	/home/michele/Stats2001/sar-q.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
se live_test.se	Apr. 19, 11:30	Apr. 20, 12:50	/home/michele/Stats2001/live_test.19AR	run in bg, at 30 -seconds intervals
vmmonitor*	Apr. 19, 12:00	May 1, 13:26	/home/michele/Stats2001/vmmonitor.19AR	run in bg, at 30 -seconds intervals

netmonitor*	Apr. 19, 12:00	Apr. 25, 12:00	/home/michele/Stats2001/netmonitor.19AR	run in bg, at 30 -seconds intervals
iostat -xPnce 300	Apr. 27, 0:00	Apr. 27, 23:55	/home/michele/Stats2001/iostat-xPnce.27AR	run in bg, at 5-minutes intevals
iostat -xPnce 300	May 10, 0:00	May 10, 23:55	/home/michele/Stats2001/iostat-xPnce.10MA	run in bg, at 5-minutes intevals

Appendix C

List of tools run on Neptune

Command//Script	Start Date	Stop Date	Output file location (on Neptune)	Additional Information
sar 30 10	Mar. 5, 11:55	Mar. 26, 13:36	/home/michele/Stats2001/sar.05MR	run with crontab, every 6 minutes (runs for 5 minutes)
uptime	Mar. 5, 12:00	Apr. 2, 9:20	/home/michele/Stats2001/uptime.05MR	run with crontab, every 10 minutes
df -k	Mar. 7, 18:00	Apr. 30, 18:00	/home/michele/Stats2001/df-k.07MR	run with crontab, once a day at 18:00
top	Mar. 12, 0:05	Mar. 19, 9:05	/home/michele/Stats2001/top.12MR	run with crontab, every 15 minutes
vmstat 300	Mar. 12, 9:04	Apr. 12, 12:05	/home/michele/Stats2001/vmstat.12MR	run in bg, at 5-minutes intevals
vmstat -s	Mar. 15, 0:30	Apr. 15, 23:30	/home/michele/Stats2001/vmstat-s.15MR	run with crontab, every hour
netstat -s	Mar. 19, 0:01	Apr. 5, 23:41	/home/michele/Stats2001/netstat-s.19MR	run with crontab, every 20 minutes
mpstat 300 2	Mar. 21, 0:05	Apr. 5, 14:44	/home/michele/Stats2001/mpstat.21MR	run with crontab, every 10 minutes (avg. over 5 minutes)
uptime	Mar. 28, 0:01	May 1, 15:01	/home/michele/Stats2001/uptime.28MR	run with crontab, every 5 minutes
se nproc.se	Mar. 30, 15:30	Apr. 11, 23:50	/home/michele/Stats2001/nproc.30MR	run with crontab, every 10 minutes
se cpus.se	Mar. 31, 10:00	May 1, 13:00	/home/michele/Stats2001/cpus.31MR	run with crontab, 3x/day
vmstat 5	Apr. 3, 0:00	Apr. 4, 9:46	/home/michele/Stats2001/vmstat-test.03AR	run in bg, at 5-seconds intevals
se vmstat.se 5	Apr. 3, 0:00	Apr. 4, 9:46	/home/michele/Stats2001/vmstat.se-test.03AR	run in bg, at 5-seconds intevals
iostat -xPnce 300	Apr. 4, 0:00	Apr. 5, 12:00	/home/michele/Stats2001/iostat-xPnce.04AR	run in bg, at 5-minutes intevals
sar 300 288	Apr. 4, 0:00	Apr. 4, 23:55	/home/michele/Stats2001/sar.04AR	run in bg, at 5 -minutes intervals (for 24 hours)
se cpg.se [head and tail]	Apr. 4, 10:00	May 1, 15:01	/home/michele/Stats2001/cpg.04AR	run with crontab, 4x/day at 10:00, 14:00, 15:00, and 16:00
/usr/ucb/ps uaxw | head	Apr. 5, 0:00	Apr. 10, 23:50	/home/michele/Stats2001/ucb_ps.05AR	run with crontab, every 10 minutes
se nproc.se	Apr. 12, 0:01	May 1, 15:01	/home/michele/Stats2001/nproc.12AR	run with crontab, every 5 minutes
iomonitor*	Apr. 12, 0:00	May 1, 1:08	/home/michele/Stats2001/iomonitor.12AR	run in bg, at 30 -seconds intervals
vmmonitor*	Apr. 12, 0:00	Apr. 30, 22:48	/home/michele/Stats2001/vmmonitor.12AR	run in bg, at 30 -seconds intervals
se vmstat.se 300	Apr. 17, 0:01	Apr. 24, 11:21	/home/michele/Stats2001/vmstat.se.17AR	run in bg, at 5 -minutes intervals
netstat -i 300	Apr. 17, 0:01	Apr. 24, 19:06	/home/michele/Stats2001/netstat-i.17AR	run in bg, at 5-minutes intevals

sar -g 60 5	Apr. 17, 0:01	Apr. 24, 12:54	/home/michele/Stats2001/sar-g.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
sar -r 60 5	Apr. 17, 0:01	Apr. 24, 12:54	/home/michele/Stats2001/sar-r.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
sar -q 60 5	Apr. 17, 0:01	Apr. 24, 12:54	/home/michele/Stats2001/sar-q.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
se live_test.se	Apr. 19, 10:35	Apr. 20, 10:30	/home/michele/Stats2001/live_test.19AR	run in bg, at 30 -seconds intervals
netmonitor*	Apr. 19, 15:00	Apr. 25, 12:01	/home/michele/Stats2001/netmonitor.19AR	run in bg, at 30 -seconds intervals
iostat -xPnce 300	Apr. 27, 0:00	Apr. 27, 23:55	/home/michele/Stats2001/iostat-xPnce.27AR	run in bg, at 5-minutes intevals
iostat -xPnce 300	May 10, 0:00	May 10, 23:55	/home/michele/Stats2001/iostat-xPnce.10MA	run in bg, at 5-minutes intevals

Appendix D

List of tools run on Poseidon

Command/Script	Start Date	Stop Date	Output file location (on Rosskon)	Additional Information
uptime	Mar. 12, 0:00	Apr. 2, 9:50	/home/michele/Stats2001/uptime.12MR	run with crontab, every 10 minutes
sar 30 10	Mar. 12, 0:01	Apr. 2, 9:48	/home/michele/Stats2001/sar.12MR	run with crontab, every 6 minutes (runs for 5 minutes)
/usr/ucb/ps uaxw | head	Mar. 12, 0:00	Mar. 26, 9:45	/home/michele/Stats2001/ucb_ps.12MR	run with crontab, every 15 minutes
vmstat 300	Mar. 14, 9:51	Apr. 6, 18:16	/home/michele/Stats2001/vmstat.14MR	run in bg from cmd line, at 5-minutes intevals
df -k	Mar. 14, 17:00	Apr. 30, 17:00	/home/michele/Stats2001/df-k.14MR	run with crontab, once a day at 17:00
top	Mar. 15, 0:05	Mar. 26, 9:35	/home/michele/Stats2001/top.15MR	run with crontab, every 15 minutes
se nproc.se	Mar. 15, 0:00	Apr. 11, 23:50	/home/michele/Stats2001/nproc.15MR	run with crontab, every 10 minutes
se cpus.se	Mar. 15, 0:00	May 1, 14:00	/home/michele/Stats2001/cpus.15MR	run with crontab, every 2 hours
vmstat -s	Mar. 15, 0:30	Apr. 15, 23:30	/home/michele/Stats2001/vmstat-s.15MR	run with crontab, every hour
mpstat 300 2	Mar. 21, 0:05	Apr. 9, 17:30	/home/michele/Stats2001/mpstat.21MR	run with crontab, every 10 minutes (avg. over 5 minutes)
iostat -xPnce 60	Mar. 26, 9:59	Apr. 2, 9:42	/home/michele/Stats2001/iostat-xPnce.26MR	run in bg from cmd line, at 1-minute intevals
uptime	Mar. 28, 0:01	May 1, 15:21	/home/michele/Stats2001/uptime.28MR	run with crontab, every 5 minutes
netstat -s	Mar. 28, 11:21	Apr. 9, 17:21	/home/michele/Stats2001/netstat-s.28MR	run with crontab, every 20 minutes
se cpg.se [head and tail]	Apr. 4, 10:00	May 1, 15:01	/home/michele/Stats2001/cpg.04AR	run with crontab, 4x/day at 10:00, 14:00, 15:00, and 16:00
iostat -xPnce 300	Apr. 4, 0:00	Apr. 6, 18:25	/home/michele/Stats2001/iostat-xPnce.04AR	run in bg, at 5-minutes intevals
sar 300 288	Apr. 4, 0:00	Apr. 4, 23:55	/home/michele/Stats2001/sar.04AR	run in bg, at 5-minutes intervals (for 24 hours)
/usr/ucb/ps uaxw | head	Apr. 5, 0:00	Apr. 10, 23:50	/home/michele/Stats2001/ucb_ps.05AR	run with crontab, every 10 minutes
se nproc.se	Apr. 7, 0:01	Apr. 9, 23:56	/home/michele/Stats2001/nproc.07AR	run with crontab, every 5 minutes
se vmstat.se 300	Apr. 7, 0:01	Apr. 10, 9:41	/home/michele/Stats2001/vmstat.se.07AR	run in bg, at 5 -minutes intervals
iostat -xPnce 300	Apr. 7, 0:01	Apr. 10, 14:41	/home/michele/Stats2001/iostat-xPnce.07AR	run in bg, at 5-minutes intevals
se nproc.se	Apr. 12, 0:01	May 1, 15:21	/home/michele/Stats2001/nproc.12AR	run with crontab, every 5 minutes
iomonitor*	Apr. 12, 0:00	May 1, 4:31	/home/michele/Stats2001/iomonitor.12AR	run in bg, at 30 -seconds intervals
vmmonitor*	Apr. 12, 0:00	Apr. 20, 12:11	/home/michele/Stats2001/vmmonitor.12AR	run in bg, at 30 -seconds intervals

netstat -i 300	Apr. 17, 0:01	Apr. 24, 19:16	/home/michele/Stats2001/netstat-i.17AR	run in bg, at 5-minutes intevals
sar -g 60 5	Apr. 17, 0:01	Apr. 24, 13:36	/home/michele/Stats2001/sar-g.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
sar -r 60 5	Apr. 17, 0:01	Apr. 24, 13:36	/home/michele/Stats2001/sar-r.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
sar -q 60 5	Apr. 17, 0:01	Apr. 24, 13:36	/home/michele/Stats2001/sar-q.17AR	run with crontab, every 6 minutes (runs for 5 minutes)
netmonitor*	Apr. 19, 15:00	Apr. 24, 14:10	/home/michele/Stats2001/netmonitor.19AR	run in bg, at 30 -seconds intervals
se live_test.se	Apr. 19, 12:00	Apr. 20, 12:52	/home/michele/Stats2001/live_test.19AR	run in bg, at 30 -seconds intervals
iostat -xPnce 300	Apr. 27, 0:00	Apr. 27, 23:55	/home/michele/Stats2001/iostat-xPnce.27AR	run in bg, at 5-minutes intevals
iostat -xPnce 300	May 10, 0:00	May 10, 23:55	/home/michele/Stats2001/iostat-xPnce.10MA	run in bg, at 5-minutes intevals

Bibliography

[ALO99] *Oracle8 and UNIX Performance Tuning*, by A. Alomari, Prentice Hall, 1999.

[CER98] *Solaris Performance Administration: Performance Measurement, Fine Tuning, and Capacity Planning for releases 2.5.1 and 2.6*, by H. Frank Cervone, McGraw-Hill, 1998.

[CPS98] *Sun Performance and Tuning, Java and the Internet*, 2nd edition, by Adrian Cockcroft and Richard Pettit, Sun Press, 1998.

[DAT00] *An Introduction to Database Systems*, 7th edition, by C. J. Date, Reading Mass.; Harlow, England: Addison-Wesley, 2000.

[DUN98] *Database Performance Tuning Handbook*, by Jeff Dunham, McGraw-Hill, 1998.

[GCO96] *Oracle Performance Tuning*, by Mark Gurry and Peter Corrigan, O'Reilly, 1996.

[GEL00] *System Performance Evaluation: Methodologies and applications*, by Erol Gelenbe, Boca Raton, Fla.; London: CRC Press, 2000.

[GUN98] *The Practical Performance Analyst: Performance-by-design techniques for distributed systems*, Neil J. Gunther, New York; Montreal: McGraw-Hill, 1998.

[ICCM00] Institute for Computer Capacity Management: www.iccmforum.com.

[JAI91] *The Art of Computer Systems Performance Analysis*, by Raj Jain, John Wiley and Sons, 1991.

[LIL00] *Measuring Computer Performance: A practitioner's guide*, by David J. Lilja, Cambridge, UK; New York: Cambridge University Press, 2000.

[LOU91] *System Performance Tuning*, by Mike Loukides, Sebastopol, CA: O'Reilly, 1991.

[OST96] Oracle 7 Server™ Tuning manual, from Oracle Technology Network (http://technet.oracle.com/doc).

[PRI89] *A Benchmark Tutorial*, by W. J. Price, IEEE Micro, Oct. 1989 (28-43).

[SES99] *Delivering Performance on Sun: System Tuning*, Technical White Paper by Greg Schmitz and Allan Esclamado, Sun Microsystems Inc., 1999.

[SOB99] *A Practical Guide to Solaris*, by Mark Sobell, Addison-Wesley, 1999.

[SPEC] The Standard Performance Evaluation Corporation (SPEC) benchmark site at http://www.specbench.org.

[STA95] *Operating Systems*, 2nd edition, by William Stallings, Prentice Hall, 1995.

[SVM98] *Sun StorEdge Volume Manager 2.6 User's Guide*, Sun Press, 1998.

[TPCW] The Transaction Performance Council (TPC) site at http://www.tpc.org.

[WIS00] The IT-Specific encyclopedia at www.whatis.com.