Resource Management

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 USA
650 960-1300   Fax 650 969-9131

Part No. 805-7268-10
July 1999, Revision A

Send comments about this document to: `blueprints@sun.com`

CHAPTER 1

Introduction

This chapter introduces the business problems that resource management addresses and explains the scope of this book.

Business Problems

If you work in a large data center environment, the following questions might be familiar to you.

- The data center is full of systems. Many of those systems are lightly used, and more applications are always waiting to be brought online. There is no room to add more systems, and the available systems are getting more powerful. How can we add a new application to an existing system, without affecting the level of service provided to its users?
- There are so many small server systems that they are a nightmare to manage. Each server has its own custom setup, and unlike an environment containing desktop machines, it's hard to automate a cloned installation process. How can we combine lots of small servers into a few big ones?
- Very large systems are installed in the data center, and many applications share their resources. How can we measure and control these applications to meet a service level agreement (SLA) that we have with users?
- The Solaris operating environment is being installed as a replacement for mainframes running MVS. How can mainframe techniques be applied to a UNIX system? What is the same and what is new?
- Sun provides a base-level operating environment with many facilities and several unbundled products that extend its capability. How can we tell what combinations of products work together to solve our business problems?

These are the questions answered by this book.

Scope

The scope of this book can be summarized as:

- The best way to use combinations of Sun Microsystems products to manage resources.
- Generic resource management concepts and methods.
- Comparison of UNIX system practices with mainframe class practices.

Sun's range of products provide a lot of built-in flexibility that can be deployed in many ways to meet the requirements of several markets. The documentation for these product covers all the options but normally provides only basic information about how these product can be used with other products. This book, on the other hand, focuses on resource management, looks at products that are relevant, gives detailed and specific information about how to choose the right set of products and features to solve resource management problems.

To solve the problem, products must fit into an overall methodology that addresses the processes of workload consolidation and service level management. The principles are explained with reference to tools and products that can help implement each scenario. To manage the service level you provide, you must be able to measure and control the resources consumed by it.

Resource management is an established discipline in the mainframe operations arena. As Solaris systems are deployed in the data center, mainframe staff must figure out how to apply existing management practices to these unfamiliar systems. One of the common complaints voiced by mainframe staff confronted with UNIX systems is that they don't have the measurement data they need to do capacity planning and performance management properly. These techniques are critical parts of a consolidation and resource management process. Thus, this book provides detailed information on the tools and measurements available for Solaris systems. The Solaris operating environment is one of the best instrumented UNIX implementations, and it is supported by all the vendors of performance tools. However, not all commercial tools report Solaris-specific metrics.

Resource management for UNIX systems is in its infancy compared to common practices under MVS. We are all trying to solve the same set of problems. So over time, the Solaris operating environment will provide comparable resource and service level management features.

CHAPTER 2

Service Level Management

This chapter describes the overall methodology of service level management so that the resource management component can be put into a wider context. It also describes and compares several approaches to resource management.

Service Level Definitions and Interactions

This section starts with a high level view of service level management and defines a terminology that is based on existing practices.

Computer systems are used to provide a service to end users. System and application vendors provide a range of components that can be used to construct a service. System managers are responsible for the quality of this service. A service must be available when it is needed and must have acceptable performance characteristics.

Service level management is the process by which information technology (IT) infrastructure is planned, designed, and implemented to provide the levels of functionality, performance, and availability required to meet business or organizational demands.

Service level management involves interactions between end users, system managers, vendors and computer systems. A common way to capture some of these interactions is with a service level agreement (SLA) between the system managers and the end users. Often, many additional interactions and assumptions are not captured formally.

Service level management interactions are shown in FIGURE 2-1. Each interaction consists of a service definition combined with a workload definition. There are many kinds of service definitions and many views of the workload. The processes involved in Service Level Management include creating service and workload definitions and translating from one definition to another.

The workload definition includes a *schedule* of the work that is run at different times of the day (for example, daytime interactive use, overnight batch, backup, and maintenance periods). For each period, the workload mix is defined in terms of applications, transactions, numbers of users, and work rates.

The service level definition includes availability and performance for *service classes* that map to key applications and transactions. Availability is specified as uptime over a period of time and is often expressed as a percentage (for example 99.95 percent per month). Performance may be specified as response time for interactive transactions or throughput for batch transactions.

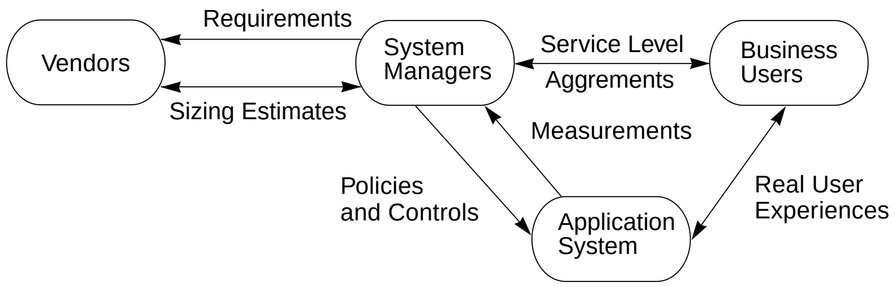

FIGURE 2-1 Service Level Management Interactions

System Level Requirements

System managers first establish a workload definition and the service level requirements. The requirements are communicated to vendors, who respond by proposing a system that meets these requirements.

Sizing Estimates

Vendors measure the system performance using generic benchmarks. They may also work with system managers to define a customer-specific benchmark test. Vendors provide a sizing estimate based on the service level requirements and workload definition. The basis of the sizing estimate can be a published benchmark performance. In some cases, the measured performance on a customer-defined benchmark is used as the basis of a sizing estimate. Vendors provide reliability data for system components. They can also provide availability and performance guarantees for production systems with a defined workload (at a price). Vendors cannot provide unqualified guarantees because typically, many application and environmental dependencies are outside their control.

Service Level Agreements

System managers and end users negotiate an SLA that establishes a user-oriented view of the workload mix and the service levels required. This may take the form: 95th percentile response time of under two seconds for the new-order transaction with up to 600 users online during the peak hour. It is important to specify the workload (in this case the number of users at the peak period), both to provide bounds for what the system is expected to do and to be precise about the measurement interval. Performance measures averaged over shorter intervals will have higher variance and higher peaks.

The agreed-upon service levels could be too demanding or too lax. The system may be quite usable and working well, but still failing an overly demanding service level agreement. It could also be too slow when performing an operation that is not specified in the SLA or whose agreed-to service level is too lax. The involved parties must agree to a continuous process of updating and refining the SLA.

Real User Experiences

The actual service levels experienced by users with a real workload are subjective measures that are very hard to capture. Often problems occur that affect parts of the system not covered explicitly by the service level agreement, or the workload varies from that defined in the service level agreement. One of the biggest challenges in performance management is to obtain measurements that have a good correlation with the real user experience.

Service Level Measurements

The real service levels cannot always be captured directly, but the measurements taken are believed to be representative of the real user experience. These measurements are then compared against the service level agreement to determine whether a problem exists. For example, suppose downtime during the interactive shift is measured and reported. A problem could occur in the network between the users and the application system that causes poor service levels from the end-user point of view but not from the system point of view. It is much easier to measure service levels inside a backend server system than at the user interface on the client system, but it is important to be aware of the limitations of such measurements. A transaction may take place over a wide variety of systems. An order for goods will affect systems inside and outside the company and across application boundaries. This problem must be carefully considered when the service level agreement is made and when the service level measurements are being defined.

Policies and Controls

System managers create policies that direct the resources of the computer system to maintain service levels according to the workload definition specified in those policies. A policy workload definition is closely related to the service level agreement workload definition, but may be modified to satisfy operational constraints. It is translated into terms that map oto system features. Example policies include:

- A maximum of 600 interactive users of the order entry application at any time.
- Order entry application has a 60 percent share of CPU, 30 percent share of network, and 40 percent share of memory.
- If new-order response time is worse than its target, steal resources from other workloads that are overachieving.

The policy is only as effective as the measurements available to it. If the wrong things are being measured, the policy will be ineffective. The policy can control resources directly or indirectly. For example, direct control on CPU time and network bandwidth usage might be used to implement indirect control on disk I/O rates by slowing or stopping a process.

Capacity Planning and Exception Reporting

The measured workload and service levels should be analyzed to extract trends. A capacity planning process can then be used to predict future scenarios and determine action plans to tune or upgrade systems, modify the service level agreement, and proactively avoid service problems.

In cases where the measured workload from the users exceeds the agreed-upon workload definition or where the measured service level falls short of the defined level, an exception report is produced.

Accounting and Chargeback

The accrued usage of resources by each user or workload may be accumulated into an accounting system so that projects can be charged in proportion to the resources they consume.

Resource Management Control Loop

To manage a resource, you must be able to measure and control it. A control loop is set up that measures the performance of an application subsystem, applies policies and goals to decide what to do, then uses controls to change the resources being provided to that subsystem. This loop can be implemented manually on a time scale measured in days or weeks (by reconfiguring and upgrading entire systems), or it can be automated in software and run as often as every few seconds. The control loop is shown in FIGURE 2-2.

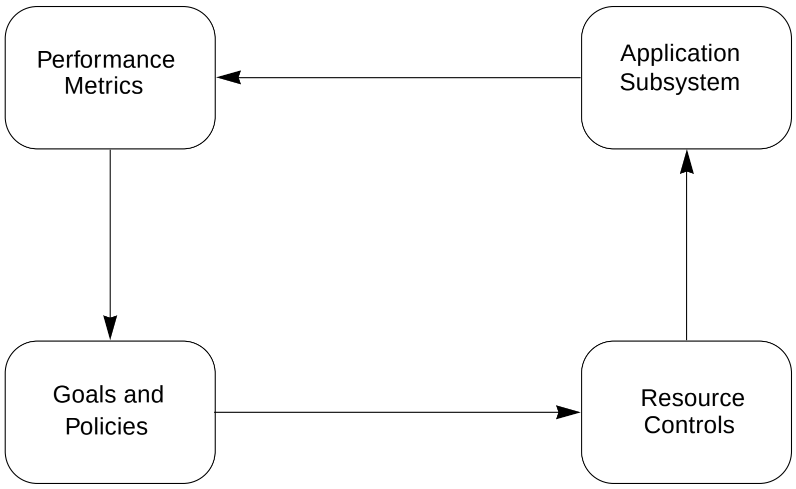

FIGURE 2-2 Resource Management Control Loop

A complete system implements many control loops. A brief digression into basic control theory is provided at this point to help explain the behavior of such systems.

A Simple Approach to Control Theory

We spend so much of our lives operating control loops that it is actually quite intuitive to most people. Designing a control loop is more complex and requires a more explicit understanding of the situation.

You start with an objective, such as to steer a car around a corner while staying in the lane. You apply a control input by turning the steering wheel to the point that will get you around the corner. After a delay. the car responds, You measure the response, compare it with what you wanted, and obtain the error difference. If the difference is zero you don't need to change the control input. If the turn is too tight, you need to reduce the input. If the turn is too wide, you need to increase the input. You have to decide how much extra correction is needed to compensate for being wrong the first time, and also decide whether the car has finished responding fully to the initial input. You may decide to over- or under-correct, and apply the correction gradually or quickly (for example, if you are heading straight for a lamp post!).

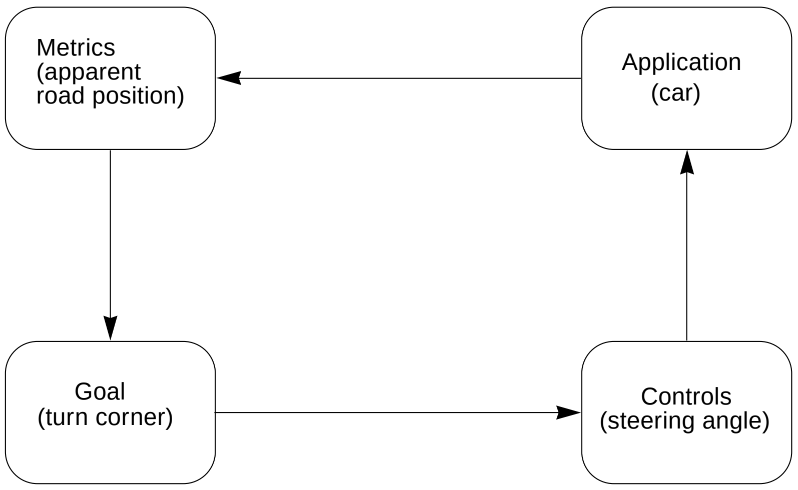

FIGURE 2-3 Example Control Loop

The first time you tried a car driving game on a computer, you probably swung wildly from side to side. This wild swinging is caused by over-correcting too late because you don't have the same motion sensing inputs you have in a real car. When the car is oscillating, your corrections may end up being delayed to the point that you are turning the wheel the wrong way at the wrong time, and you might spin off or crash. Eventually, you learn to react based on what you see on the screen and make smaller corrections more quickly to keep the car on track and stable.

In control terms, you are applying *negative feedback* to the system. You take the error difference between what you wanted and what you got, and apply the inverse of the error to the system to reduce the error in the future. The rate at which you measure and apply corrections is called the *control interval,* and the rate at which the system responds to changes is called the *time constant* for the loop. The amount of the error that you feed back changes the characteristic behavior of the control loop. If you feed back a large proportion of the error with a short control interval, the system is *lightly damped* and will be very responsive to sudden changes but will probably oscillate back and forth. If you feed back a small proportion of the error over a longer control interval, the system is *heavily damped* and will tend to be sluggish and unresponsive with a large time constant.

When you apply these principles to computer system resource management you can see that it is important to average measurements over an appropriate time scale and get the damping factor right. The resource manager needs to respond quickly enough to cope with sudden changes in the workload such as many simultaneous user logins at the start of a shift, while maintaining a steady flow of resources to all the workloads on the system so that response times are consistent and predictable.

Viewpoints of Resource Management

You can measure and control a computer system in many ways. This section examines various approaches to solving resource management problems. The methodology used depends upon the starting point of the developers, for example a network-centric methodology can be extended for use in other areas, so can a storage-centric or server-centric viewpoint.

Diverse Methods

This diversity of approach occurs both because of the products that are available and because of the expectations of the end users who are purchasing solutions. In a large organization, several groups (such as system administrators, network managers, database administrators, and security managers) are responsible for different parts of operations management.

In some organizations, each group is free to use its own methods and obtain its own tools. This can cause demarcation problems because there is so much overlap in the scope of each method. Or a single methodology and tool could be imposed by the dominant group. The problem with such an aproach is that the methodology may be optimal for managing one aspect of the system only and could do a poor job in other areas.

In an ideal world, one all-encompassing mega-tool would implement an integrated methodology and solve all resource management problems. The closer you get to this ideal, the more expensive and complex the tool becomes. And you may not want all of its features. So it is harder to justify purchasing it.

A more pragmatic approach is to integrate simpler, more specialized tools that share information with each other, have a similar look and feel, and can be used as a standalone product as needed.

Today's diverse set of methodologies and tools have very little integration between them and several different user interfaces. The products produced at Sun Microsystems, Inc. are converging on a common user interface style and on common technologies for sharing information to provide better integrated resource management components. This book's primary role is to explain to data center operations managers how to use combinations of the current set of products and technologies. It also indicates the direction of the development and integration work that will produce the next generation of products.

The System-Centric Viewpoint

The system-centric viewpoint focuses on what can be done on a single server using a mixture of hardware and operating system features. The operating system provides basic management capabilities for many components such as attached network and storage devices. But it specializes in managing CPU and memory resources for a single desktop or server system. Dynamic Reconfiguration (DR), processor sets, Solaris Resource Manager™, and Sun Enterprise™ 10000 (also known as Starfire™) Dynamic System Domains (DSDs) are all system-centric resource management technologies. The Sun hardware system-centric resource management tool is the Sun Enterprise SyMON™ 2.0 software. It is based on the Simple Network Management Protocol (SNMP), so it also has some network management capabilities. The Solaris Management Console™ software provides a more generic framework for gathering together operating system administration tools and interfacing to industry standard initiatives such as the web-based management initiative (WebM) and the Common Information Model (CIM). The Starfire system currently uses its own HostView interface running on a separate system service processor (SSP) to manage domains.

The system-centric viewpoint runs into problems when a lot of systems must be managed. Coordinating changes and allocating resources becomes complex quite rapidly. Tools like the Sun Enterprise SyMON 2.0 software can view many systems on one console, but cannot replicate and coordinate changes over multiple systems. One reason why the Sun Enterprise SyMON 2.0 software does not fully support management of the Starfire system is because each domain on the system runs a separate copy of the Solaris operating environment and sees a different subset of the hardware configuration. The next release of this software will be extended to view an SSP and all the domains in a Starfire system as a special kind of cluster so it can be used in place of the HostView interface.

The operating system and devices provide a large number of measurements of utilization, throughput, and component-level response times. There are several good ways to control CPU resources, but at present, there is no way to control the usage of real memory by a workload. The only way to constrain a workload that is using too much memory is to slow down or stop its CPU usage so that it stops referencing its memory, which will then be stolen by other, more active processes.

Manual resource management policies can be implemented using the Sun Enterprise SyMON Health Monitor, which generates alerts when a component of the system becomes overloaded. The system administrator can then tune or reconfigure the system to avoid the problem. Automatic resource management policies are implemented by Solaris Resource Manager, which dynamically adjusts the priorities of processes to ensure that their recent CPU usage tracks the share of the system that has been given as a goal for that user.

The Cluster-Centric Viewpoint

The cluster-centric viewpoint concentrates on coordinating resource management across the cluster. Systems are clustered together to provide higher availability and higher performance than that provided by a single system. A cluster is more complex to install and administer, so tools and methods attempt to automate cluster management to make it more like a single system. From a resource-management viewpoint, the primary issue is load balancing and the extra costs of accessing nonlocal information over the cluster interconnect. Sun has two kinds of clusters. The highly integrated SPARCcluster™ product range is focused on improved availability in commercial environments. Its management tools will eventually become an integrated extension to the SyMON software. For high performance computing, Sun HPC Servers use the platform computing load share facility (LSF) to perform load balancing on much larger and more loosely coupled clusters.

At a cluster level, multiple system level measurements are compared to measure load balance and look for spare resources. The cluster interconnect utilization and proportion of remote data access are important additional measures. The primary resource management control is the choice of where to run new work. It is not currently possible to migrate a running job from one node in a cluster to another, or to checkpoint a job to disk and restart it again later, either on the same node or on a different one.

When deciding on the resources that are available on a node, it is easy to decide if there is some spare CPU power, but very hard to decide if there is enough available real memory. The kernel maintains a free list of memory, but there is also some proportion of memory in use as a file system cache that could be reclaimed to run a new job if there was a way to measure it. This is a current issue for the LSF product.

The Network-Centric Viewpoint

From a network point of view, there are a large number of devices to manage. Many of them are network components with limited monitoring capabilities, such as bridges and routers. The primary resource that is managed is network capacity. At the intersection of servers and networks, there are products that perform protocol based bandwidth management on a per-server basis (such as Solaris™ Bandwidth Manager software) or act as secure firewalls. In the telecommunications industry, network management encompasses all the equipment required to run a global system where the end-points are mostly telephones or mobile cellphones, and the traffic is a mixture of speech and data. In this environment, SNMP is too simple, so the more scalable OSI-based CMIP management protocol is often used. The Solstice™ Enterprise Manager product is a Telco-oriented CMIP and SNMP management system that is used to manage cellular networks. In theory it could be used to manage computer systems and local area networks, but it was not developed to do this. Computer-oriented local and wide area networks are normally managed using SNMP protocols, with the Solstice SunNet Manager™ or HP OpenView products collecting and displaying the data. Both products provide some visibility into what is happening in the computer systems on the network, but they are much more focused on network topology. At this level, resource management is done on a per-network basis, often by controlling the priority of data flows through intelligent routers and switches. The Sun Enterprise SyMON 2.0 software can act as a network management platform as well as a system hardware management platform, which may help to integrate these two viewpoints.

Protocol information along with the information found in packet headers and network addresses form the basis for network measurements. There is no user or process identifier in a packet, so it is hard to directly map network activity to system level activity unless an identifiable server process is dedicated to each protocol. Some protocols measure round trip times for their acknowledgments. This can provide an estimate of network latency between two systems.

Network controls are based on delaying and prioritizing packets based on the protocol and destination data in the packet headers. This can occur at the servers that make up the end points or in the routers that connect them.

Storage-Centric Viewpoint

Storage has recently moved from being a simple attached computer system peripheral to a complex managed entity in its own right. Networked storage using fibre channel puts an interconnection layer in between multiple servers or clusters and multiple storage subsystems. This storage area network (SAN) can contain switches and routers just like local or wide area networks, but the protocol in common use is SCSI over fibre channel rather than IP over Ethernet. A SAN may also span multiple sites, for example, where remote mirroring is being used for disaster recovery. Storage is also now open for access in a heterogeneous multi-vendor environment, where multiple server and storage vendors can all be connected over the SAN. This is an emerging technology, and tools to manage a SAN are still being developed. Sun provides one approach with an industry-wide initiative called Project StoreX. It is based on a distributed pure Java™ technology platform that can run anywhere a JVM is available, scaling from embedded devices through open systems into mainframe and enterprise environments. Project StoreX enables management of any storage resource in a heterogeneous distributed environment, from storage hardware, like devices and switches, to storage software like backup solutions and volume managers. Project StoreX is being promoted as an open multi-vendor standard.

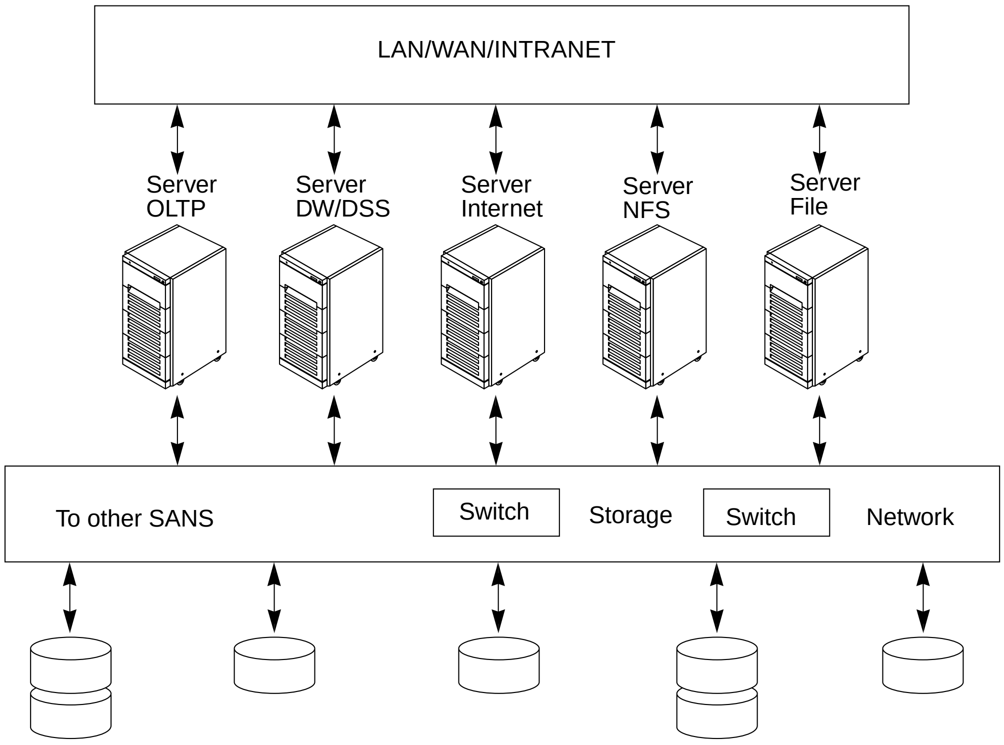

FIGURE 2-4 Storage Area Network

Storage management has two constraints that make it interesting: one is that it must be distributed over many server systems and storage devices to be useful. The other is that these servers and devices come from many vendors and run different operating software. So Project StoreX cannot include a core dependency on the Solaris operating environment or other Sun products. Project StoreX must solve some of the generic problems of clustered and networked resource management. In particular, it manages the state of distributed devices in a persistent manner. Project StoreX must be capable of stand-alone operation, with component management interfaces to other products.

Project StoreX enables resource management of capacity, performance, and availability of data storage. Backup and archival policies can be used to automate migration of data to a tape library. Measurements of capacity and performance characteristics can be combined with availability policies, so the operator will be alerted of any problems. Ultimately storage subsystems will be reconfigured automatically.

At present, Project StoreX does not include a general purpose rule script-based policy engine. It does provide the infrastructure necessary to create polices, allowing the view of storage to be elevated to the Storage Service level.

Integration between the system viewpoint and the storage viewpoint has some of the same problems as the integration of system and network viewpoints. The traffic on the SAN does not contain any indication of which user or process generated the request. Within the Solaris software, it is possible to trace storage accesses on a per-process, per device basis. But the overhead of collecting and analyzing this data is quite high. There may be a need for a Project StoreX Solaris Storage Bandwidth Manager to bridge the two management viewpoints in a way that accounts for, prioritizes, and controls SAN bandwidth on a per process or per user basis.

Database-Centric Viewpoint

A database management system has its own notion of users, manages its own memory and storage allocation, and can appear as a “black box” to the server system on which it runs. Some database vendors have implemented their own resource management capability on a per user or per transaction basis. This may take the form of priorities, shares, or limits on CPU usage per user or per transaction. The database also contains its own logic that implements policies and dynamically controls resources.

The database implementation can sometimes work well with server-based resource management. This is described in more detail in Chapter 4.

Integration is needed between the internal database resource management capabilities and the other resource management viewpoints.

Application-Centric Viewpoint

Large and complex applications such as SAP R/3, Baan, and Oracle Financials contain their own resource management concepts and controls. For example, Oracle Financials implements its own batch queue system to decouple the generation of large reports from the interactive response time of the users. A report is sent to a subsystem called the concurrent manager, which is configured to have a number of parallel streams of work of various types according to the local policy. Concurrent manager activity can be scheduled to occur outside normal working hours, or it can be used to soak up spare cycles during the day.

SAP R/3 measures the response time of important transactions and breaks these down into application server time and backend database server time. As many users connect to an application server and many database servers connect to a single backend database, there is no concept of which user is doing work on the backend system. The application itself must have the instrumentation to keep track of what is going on. The application directly implements the policies and controls.

Integrated Methods

In a large organization, all of the above viewpoints are useful, and some combination of methodologies is probably implemented already. FIGURE 2-5 indicates the relative breadth of coverage of each methodology. A darker box shows that better coverage is available, a white box indicates that little or no support is provided for a combination.

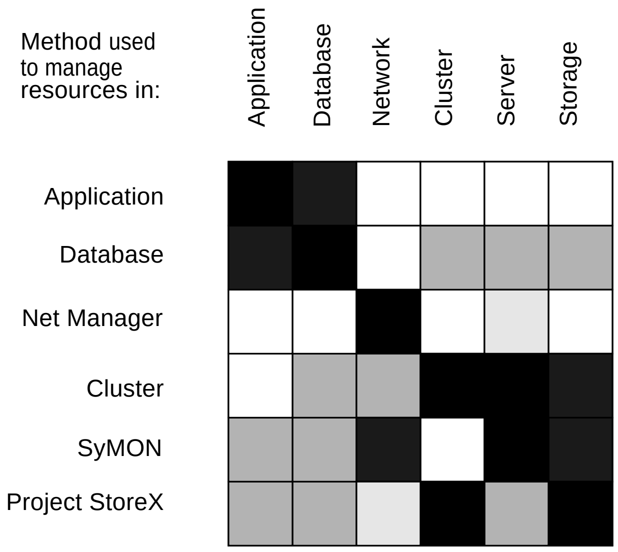

FIGURE 2-5 Integrated Methodology by Viewpoint

The mature methods have remained specialized, but the emerging technologies of Sun Enterprise SyMON 2.0 and Project StoreX address a much broader scope of problems. The product overview section of this book discusses their capabilities in much greater detail.

So far the discussion has been quite abstract. The next chapter introduces several example workloads, showing the appropriate methodologies to manage resources for them.

The Consolidation Process

The consolidation process starts when you identify candidate systems and applications. First measure the resource usage and service levels of those systems so you can see which application workloads will fit together best.

Next, migrate those systems to a common Solaris release and patch revision and do some testing so you can be sure that everything works correctly in the same environment. You also need to make sure that there are no conflicts in the name service configuration and network services files. For example, local password files may need to be merged, and any conflicting port numbers specified in the `/etc/services` file may need to be cleared. If you use a name service such as NIS for all your password and services information, then the systems should already be seeing the same name space and definitions. Using a common name service eases the consolidation process. If you prefer to make all the changes at one time, then you can upgrade as the application is consolidated, but allow for more testing time and a more incremental installation process on the consolidated system

For each group of consolidated applications, you must choose appropriate resource management controls. Once you have consolidated your applications to fewer systems, monitor and re-size the consolidated systems to allow for peak loads. You can either remove excess resources for use elsewhere or identify additional candidate applications to be consolidated onto these systems. Treat this as a rolling upgrade program rather than a one-time big change.

An obvious question that arises is how many systems should the new consolidation contain. Circumstances vary, but the basic principles remain the same. If you treat consolidation as a process, then the number of systems decreases over time and the size of systems increases.

Downtime impacts multiple applications on the consolidated systems. Therefore, when you increase the resources by adding an extra application, you want to do so without rebooting. Consolidated upgrades benefit from systems that can perform dynamic reconfiguration. The midrange Sun Ultra™ Enterprise™ E3000-E6500 servers can perform I/O board reconfiguration with the Solaris 2.6 release, but they require the Solaris 7 release for dynamic reconfiguration of CPU and memory, which causes some application availability issues. The number of system footprints may be too high with midrange servers, and it is hard to reduce the total number of servers effectively. With the high-end Starfire system, Dynamic System Domains (DSDs) solve these problems. DSDs are supported on the Solaris 2.5.1, 2.6, and 7 releases. The total number of DSDs can be reduced as applications are consolidated.

One approach is to use each DSD for a different Solaris revision. You may have a large DSD for the bulk of your Solaris 2.6 applications, a smaller one for applications that have not yet migrated from the Solaris 2.5.1 release, and a development and test

DSD for the Solaris 7 release. Over time, the Solaris 2.5.1 DSD will shrink away and its resources will migrate into the other DSDs. Applications will also migrate into the Solaris 7 DSD. The key benefit here is that this all happens under software control, using a single system footprint in the data center. DSDs are described in detail Chapter 8.

Use the Solaris Resource Manager or the Solaris Bandwidth Manager software or processor sets to control applications within a single copy of the Solaris operating environment.

A consolidated system runs a mixture of workloads. You have to choose relevant processes and aggregate to measure them. The remainder is overhead or unplanned activity. If it is significant, it should be investigated. Break down network workloads as well so that you know which applications are generating the network traffic.

There is a common set of measurements to collect per workload.

- Number of processes and number of users
- End user response times for a selection of operations
- User and System CPU usage
- Real and virtual memory usage and paging rates
- I/O rates to disk and network devices
- Microstate wait timers to see which resources are bottlenecks

To actually perform workload aggregation you have to match patterns in the data.

- match processes on user name
- match processes on command name and arguments
- match processes using processor set binding
- match system accounting data using user and command name
- match network packets on port number and protocol

You usually have to assign disks and file systems to workloads manually. Dealing with shared memory, libraries, and code makes RAM breakdown hard.

When you are accumulating measurements don’t accumulate the `ps` command `CPU%`. It’s a decayed average of recent CPU usage, not an accurate measure of actual CPU usage over an interval. You need to measure the actual process CPU time used in each interval by taking the difference of two measurements. There is more detail on the available measurements and what they mean in Chapter 5.

CHAPTER 3

Policies and Controls

To manage the resources of a system, you must be able to measure and control resource usage. You must also establish policies that determine the controls that are invoked once the available measurements are interpreted. This chapter examines various types of policies and controls and ways to implement them. It also looks at each subsystem in turn to see the resources that can be managed.

Policy Types

There are many types of policies and many ways to implement them. Some of the main classifications are explained below. A recent draft standard defines the terminology of each kind of policy. That terminology has been adopted in this book, and the standard is summarized in this section.

Limits and Error Event Rules

One of the simplest policies is to define limits on a measurement and associate it with an action. This is often implemented as an “if measure passes threshold then action” rule. Products such as the Sun Enterprise SyMON 2.0 software (referred to hereafter as SyMON) predefine many simple limit rules and allow new rules to be set on any measurement. A limit can be defined as a simple rule with a single input measurement. It is also common to have several thresholds with a warning level action and a critical problem level action for the same measure.

An error event is different because it is treated as a discrete on/off event rather than a continuous variable to be compared against a limit.

In either case, an alert is generated and logged. The alert can be transitory and go away when the rule is re-evaluated, or it can be persistent and require a user to acknowledge that it has been seen.

Complex Rules and Hierarchies

More complex rules take several inputs and can maintain historical information such as previous state and running averages. They can also be built out of several simple limit rules. A complex rule is used to establish the state of a component or a subsystem. When rules are combined, they are ranked so that critical problems take precedence over warnings. A hierarchy of rules can be built for a network of systems so that the overall state of the network is indicated by the state of the system that has the worst problem. In turn, that state is based on the state of the subsystem that has the worst problem. A rule state propagation hierarchy is provided as part of the Sun Enterprise SyMON 2.0 product, and many other commercial tools implement this mechanism.

The policy is inherent in the set of rules that are implemented, the thresholds that the rules use, and the actions that occur when a rule becomes active.

Priority

A relative importance level can be given to the work done by a system as part of a policy that prioritizes some activities over others. The Solaris Resource Manager product and others like it assign shares to each user according to a policy decided by the administrator, then accumulate the CPU usage at a per-user level and implement a control based on the number of shares held by each user and the user's place in the hierarchy.

An alternative approach is to specify percentages directly. The Solaris Bandwidth Manager software uses this mechanism to provide a way to specify policies on a per-network packet basis. Each packet is classified by address or protocol and each class is given a priority and a percentage of the total bandwidth that it can use.

Goals

Goal-based policies are prescriptitive rather than reactive. They operate at a higher level. A goal can be translated into a mixture of limits, priorities, and relative importance levels. Goals can include actions for when the goal cannot be met.

A goal can also be thought of as a control loop, where the policy manipulates controls when a measurement deviates from its desired range. As described in Chapter 3, a control loop is a complex thing to manage because its stability characteristics, time constant, and damping factor must be set correctly. The interaction of multiple inter-linked control loops can be problematic.

Goals can be expressed in several ways:

- Response time goals try to monitor the end user response time of a system and control resources so that high priority work maintains its response time goal by taking resources from lower priority work.
- Throughput goals monitor the rate of consumption of a resource for long running jobs and control the relative priority to maintain the desired balance.
- Deadline goals have a way of telling how far a repetitive batch job has gone through its work, and control resources to ensure that the entire job completes by a deadline. For example, a payroll application must complete on time and generate the correct number of pay slips. A goal-based workload manager could monitor the running total.

At present, automated goal-based workload management is a feature found only on mainframes running OS/390 software.

Operational Policies

Some policies are implemented manually as part of operations management. For example, an availability policy can include a goal for uptime and an automatic way to measure and report the uptime over a period. There is no direct control in the system that affects uptime. It is handled by operations staff, who will reconfigure software to work around problems, swap out unreliable hardware, or reconfigure the system into a more resilient configuration if the availability goal is not being met.

In most cases, goal-based policies require manual intervention to complete the control loop. A measure of response time is monitored, and if its goal is not being met, the administrator manually varies the CPU shares, moves work from an overloaded system to another system, or performs a hardware upgrade.

Networked Security Policies

Access to a system varies according to the role of the user. A security policy can prevent access to certain resources or allow designated users to manage subsystems. For example, the SyMON software includes access control lists for operations that change the state of a system, and multiple network domain views to give different administrative roles their own view of the resources being managed. Security and network-based policies can be stored in an LDAP based name service. When users dial into an Internet service provider, they are looked up in a RADIUS authentication database, which can extract a profile from an LDAP server to configure the systems each user is allowed to access and the Solaris Bandwidth Manager configuration is updated to take into account that user’s network address.

Controls

Controls are used to limit or redirect resources.

Limits

A limit prevents a resource from exceeding a preset value. Some limits are system wide (such as the total number of processes allowed on a system) and some operate on a per-user basis (such as a file system quota). Since many limits are implemented by the Solaris software, the action, when a limit is reached, is to return an error code inside the application and possibly send a signal to the process. Well-written applications handle the errors and catch the signals, but applications that do not expect to ever run into a limit might misbehave or abort. During testing, it is a good idea to run with very tight limits and test the behavior of applications as they hit those limits.

Direct and Indirect Controls

A direct control operates on the resource you want to control. For example, the Solaris Resource Manager software controls CPU usage per user by implementing a scheduling class that determines each user's share of the CPU. An indirect control works via dependent resources. For example, to limit the I/O throughput of a process, it is sufficient to be able to measure the I/O throughput and limit the CPU resources for that process. The process could be stopped temporarily to prevent it from issuing read and write system calls at too high a rate. It might be more efficient to add a direct measurement and control capability to the code that implements read and write calls, but that is a more invasive approach that requires changes to the Solaris software itself.

The Solaris Bandwidth Manager product implements a direct control on network packet rates. This can be used to implement an indirect control on the CPU resources taken up by the NFS server code in the kernel. The Solaris Resource Manager software cannot control NFS service directly as NFS is implemented using kernel threads that do not have an associated user-level process.

Standardized Policy Definitions

The Internet draft policy framework standard by Strassner and Ellesson defines its scope thus:

> This document defines a set of terms that the Internet community can use to exchange ideas on how policy creation, administration, management, and distribution could work among policy servers and multiple device types.

The terminology definitions are network oriented but apply equally well to system level policies. Some of the terms defined in this standard are listed here.

- Administrative Domain: A collection of network elements under the same administrative control and grouped together for administrative purposes.
- Network Element (also called a Node): A networking device, such as a router, a switch, or a hub, where resource allocation decisions have to be made and the decisions have to be enforced.
- Policy: The combination of rules and services where rules define the criteria for resource access and usage.
- Policy control: The application of rules to determine whether or not access to a particular resource should be granted.
- Policy Object: Contains policy-related info such as policy elements and is carried in a request or response related to resource allocation decision.
- Policy Element: Subdivision of policy objects; contains single units of information necessary for the evaluation of policy rules. A single policy element carries an user or application identification whereas another policy element may carry user credentials or credit card information. Examples of policy elements include identity of the requesting user or application, user/app credentials, and so on. The policy elements themselves are expected to be independent of which Quality of Service signaling protocol is used.
- Policy Decision Point (PDP): The point where policy decisions are made.
- Policy Enforcement Point (PEP): The point where the policy decisions are actually enforced.
- Policy Ignorant Node (PIN): A network element that does not explicitly support policy control using the mechanisms defined in this standard.
- Resource: Something of value in a network infrastructure to which rules or policy criteria are first applied before access is granted. Examples of resources include the buffers in a router and bandwidth on an interface.
- Service Provider: Controls the network infrastructure and may be responsible for the charging and accounting of services.

General Policy Architecture

The general architecture shown FIGURE 3-1 illustrates one common implementation of a policy that combines the use of a policy repository, a PDP, and a PEP. This diagram is not meant to imply that these entities must be located in physically separate devices, nor is it meant to imply that the only protocols used for communicating policy are those illustrated. Rather, it simply shows one implementation containing the three important entities fundamental to policy: a repository, a PDP, and a PEP.

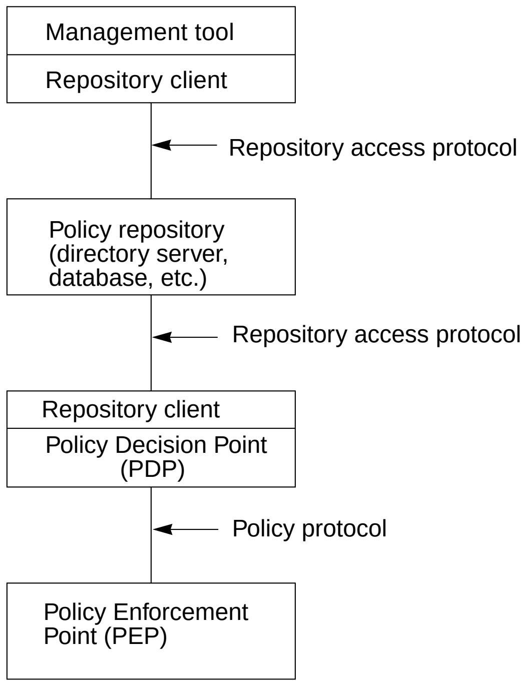
Management tool
Repository client
Repository access protocol
Policy repository (directory server, database, etc.)
Repository access protocol
Repository client
Policy Decision Point (PDP)
Policy protocol
Policy Enforcement Point (PEP)

FIGURE 3-1 Example Policy Architecture

It is assumed that policy decisions will always be made in the PDP and implemented in the PEP. Specifically, the PEP cannot make decisions on its own. This simplifies the definition and modeling of policy while leaving open the possibility for a single device to have both a local PDP (LPDP) as well as a PEP.

In general, the repository access protocol and the policy protocol are different protocols. If the policy repository is a directory, then LDAP is one example of a repository access protocol. However, the policy protocol can be any combination of COPS, SNMP, and Telnet/CLI. Given this rich diversity, a common language is needed to represent policy rules. The rest of the standard document describes the terminology necessary to enable the definition of such a language and discusses how policy is defined, manipulated, and used in the PDP and PEP. For more information, see the complete document at `http://www.ietf.org/internet-drafts/draft-strassner-policy-terms-01.txt`.

Subsystem Policies and Controls

Each component and subsystem implements a set of measurements and controls that allows a policy to manage them.

User-level Controls

User-level controls include limits on the number of logins and limits on the resources used by each user.

Login Limits

The Solaris software implements a blanket login ban for non-root users as described in the `nologin(4)` manual page. It also limits the total number of processes that can be started via the `nproc` kernel tunable. This feature scales with the memory configuration. A further limit is placed on the total number of processes per user using the `maxuprc` kernel tunable. This is a single global limit. The total number of logins per user can also be limited. The current limit can be viewed with `sysdef` or `sar`. The example in FIGURE 3-2 was run on a 64 Mbyte desktop system.

```
% sysdef -i | grep processes
    1002maximum number of processes (v.v_proc)
     997maximum processes per user id (v.v_maxup)
% sar -v 1

SunOS maddan 5.6 Generic sun4m    03/18/99

18:25:12  proc-sz    ov  inod-sz    ov  file-sz    ov   lock-sz
18:25:13  108/1002    0 4634/4634    0  495/495     0    0/0
```

FIGURE 3-2 Determining Solaris Process Limits

When the SRM software is in use, you can view the maximum number of processes that can be limited on a per-user basis using the `limadm` command. You can view the current number and limit using the `liminfo` command, as described in Chapter 7.

Application-level Controls

The main class of application-level controls are those provided by relational databases and transaction processing monitors. These consist mainly of access controls, but the Oracle8*i* database also implements controls on resource consumption and policies for relative importance. This is described in more detail in the Chapter 7.

CPU Power-level Controls

There are many ways to control CPU power. The most basic one is to physically change the CPU configuration itself. A faster CPU shortens the CPU-intensive component of response times. Additional CPUs allow more concurrent work to take place with a similar response time.

Physically changing the CPU configuration requires a power down and reboot on many computer systems. But the Sun Enterprise Server systems allow CPU boards to be added and removed from a running system without powering it down or rebooting it. On the Starfire system, the CPUs can be partitioned into dynamic system domains, and a separate copy of the Solaris operating environment booted in each domain. CPU boards can then be moved from one dynamic system domain to another. This is described in detail in Chapter 8.

With a single copy of the Solaris software, the CPUs can be partitioned into processor sets as described in Chapter 7. Each process is bound to a processor set and constrained to run only on the CPUs that are members of that set. Sets are created and removed dynamically. If one set is overloaded, its processes cannot make use of the CPU power in a different set without manual intervention. Processor sets are most useful when there are a large number of CPUs to partition. This technique is obviously not useful on a uniprocessor system.

The SRM software also works within a single copy of the Solaris operating environment. Unlike sets, it has fine granularity and can be used on a uniprocessor system. When a system gets busy, all CPU power is used automatically. SRM works by biasing CPU usage on a per-user basis, using shares to determine the relative importance of each user.

The `kill` command sends a stop signal to a process and suspends it in the same way that typing Control-Z does in an interactive shell session. Sending a start signal lets the process continue. Stopping and starting processes in this way can control the concurrency of CPU-bound jobs on a system.

The Load Share Facility (LSF) software is described in detail in Chapter 10. LSF software implements a distributed batch queuing system where jobs are submitted and, when resources are available, sent to be run on a system. LSF software implements its own set of policies and controls.

Disk I/O Policies

Access controls via file permissions and access control lists (ACLs) control who can read and write to a file system.

Currently, no direct measures or controls of the rate at which a process is writing to a file system exist. The only information provided for each process is the total read plus write data rate. Block input and output counters are not incremented correctly in current releases of the Solaris operating environment. The block counter problem is filed as bugid 1141605 and is fixed in the next release of Solaris software. This data is not made available by the standard commands; it is part of the "usage" structure that includes microstate accounting as described in Chapter 5.

Because the Solaris software does not have any direct measurements or controls, additional application-specific information is required (such as configuring the location of data files manually) to implement policies.

Disk Space

See Chapter 6 for a description of the disk quota system. Disk quotas are currently implemented separately on each file system type. They are available on the UFS file system and remote mounts of UFS via NFS only. Other file system types either have no quota system or have a separately administered implementation.

Virtual Memory

An application consumes virtual memory when it requests memory from the operating system, and the memory is allocated from a central pool of resources. Virtual memory usage, however, is not directly related to physical memory usage because not all virtual memory has physical memory associated with it. If an application requests 16 Mbytes from the operating system, the operating system will create 16 Mbytes of memory within that application's address space, but will not allocate physical memory to it until that memory is read from or written to.

When you restrict or control the amount of virtual memory that an application can have, you are not controlling the amount of RAM that application can have. Rather you are implementing a policy limit on the maximum amount of virtual address space that process can have. This is an important difference because the limit is enforced when the application first requests memory not while it is using it. When the application hits the limit, it will probably fail because its requests to extend its virtual address space will fail. This may be what you want if the application is likely to impact higher priority work. But careful testing and debugging is required to make applications recover gracefully from memory allocation failures.

Virtual memory can be limited at the system level and at the process level. At a system level, the total amount of virtual memory available is equal to the total amount of swap space available. Each time virtual memory is used, the amount of swap space available drops by the same amount. If one application requests a large amount of virtual memory (for example, `malloc` (1 Gbyte), there is potential for that application to exhaust the system-wide swap space, which will then cause other applications to fail when they request memory.

You can use resource management of virtual memory to prevent a single process from growing too large and consuming all virtual memory resources by limiting the maximum amount of memory that a process or group of processes can use. This can be useful in two cases: to prevent any one user from using all of the available swap space (a denial of service attack) and to prevent a runaway process or leaking process from consuming all of the available swap space.

Base Solaris software can do simple resource management of a process's virtual memory usage. The limits information described in Chapter 6 can be used to limit the maximum amount of virtual memory used by a process. Limits are enforced per process, thus preventing any one process from using an unreasonably large amount of virtual memory. But a user can run many processes, so this does not prevent denial of service attacks.

The SRM software has a mechanism that can limit the maximum amount of virtual memory per user. This implements a similar limits policy as the per-process limit built into the Solaris operating environment, but it can be used to limit a user, regardless of how many processes are running. FIGURE 3-3 shows the three levels of virtual memory that can be controlled.

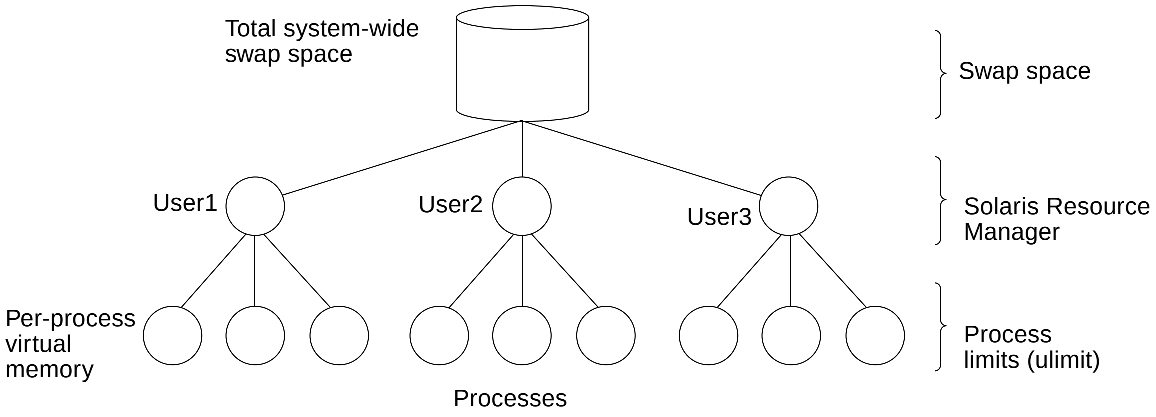

FIGURE 3-3 Three Levels of Virtual Memory Limits

Relationship Between Virtual Memory and Swap Space

It is important to note that the amount of swap space used by a user does not correlate directly to the sum of all that user's processes. A user may have three processes, where each shares a single shared memory segment between them. Each process has the shared memory segment mapped into its address space, but swap space is only accounted for once within these processes. For example, three users each have a 1 Gbyte shared global area mapped into their address space, but each incremental user is not consuming an additional 1 Gbyte of swap from the system-wide pool; they use three Gbytes of virtual memory space, but only one Gbyte of swap space.

It is important to factor this in when using products like SRM software to control a user's virtual memory usage. The virtual memory accounted for by the SRM software is different from the amount of swap space used by the user. Since we are trying to control virtual memory in an attempt to prevent a single user from consuming all of the swap space, we must take care to apply the correct virtual memory limit policy. This sometimes makes it extremely difficult to control swap space usage with SRM software, but the right parameters provide adequate control in most environments.

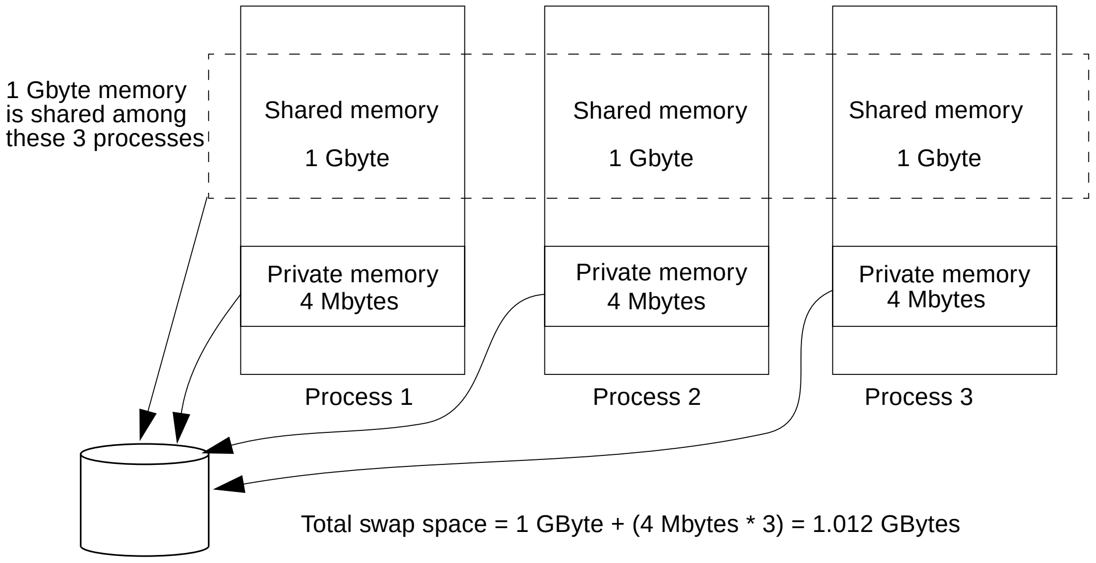

FIGURE 3-4 Swap Usage Only Accounts for Shared Segments Once

In the example illustrated in FIGURE 3-3 and FIGURE 3-4, the SRM software is configured to limit the total virtual memory to 1.012 GBytes, which allows all three processes to execute normally. If one of the three processes has a memory leak, the limit would be hit for that user, affecting only the processes owned by that user. The disadvantage is that if the user starts another process, the same limit is reached. The per-user limits must take into account the number of processes each user is expected to run.

Physical Memory

Resource Management of physical memory means defining policies and controlling the amount of RAM that is allocated to different workloads. In contrast to the virtual memory limit policies, physical memory is controlled by applying importance policies to different types of memory. In the future, it may be possible to apply limit or allocation style policies to physical memory, but that capability is not available in the Solaris operating environment today.

The physical memory management system in the Solaris operating environment can implement different policies for different memory types. By default, the memory management system applies an equal importance policy to different memory subsystems, which sometimes results in unwanted behavior. Before we look at the policies, let’s take a quick look at the different consumers of memory.

The most important consumers of memory in the Solaris operating environment are:

- Kernel memory, used to run the operating system
- Process memory, allocated to processes and applications
- System V shared memory, allocated by the shared memory subsystem by applications such as databases
- Memory used for file system caching

The Default Memory Allocation Policy

Memory in the Solaris operating environment is, by default, allocated on a demand basis with equal importance to each subsystem. When a subsystem requests memory, it is allocated from a central pool of free memory. If sufficient memory is available in the free pool, then an application's request is granted. If free memory is insufficient, then memory is taken from other subsystems to satisfy the request. The equal-importance policy means that the application with the most aggressive memory requests gets the majority of the memory assigned to it. For example, suppose a user starts a `netscape` process that uses 20 Mbytes of memory. That memory is taken from the free pool of memory and allocated to the `netscape` process. When the user starts a `gimp` image editor tool, if there is no free memory in the free memory pool, then memory will be taken from the `netscape` browser and allocated to the `gimp` image editor tool. The Solaris memory allocation policy takes into account recent usage in an attempt to choose the correct application from which to steal memory, but the usage history is very short (less than one minute) in most circumstances.

FIGURE 3-5 shows an example where the memory allocated to the `netscape` process is reduced when the `gimp` process is started.

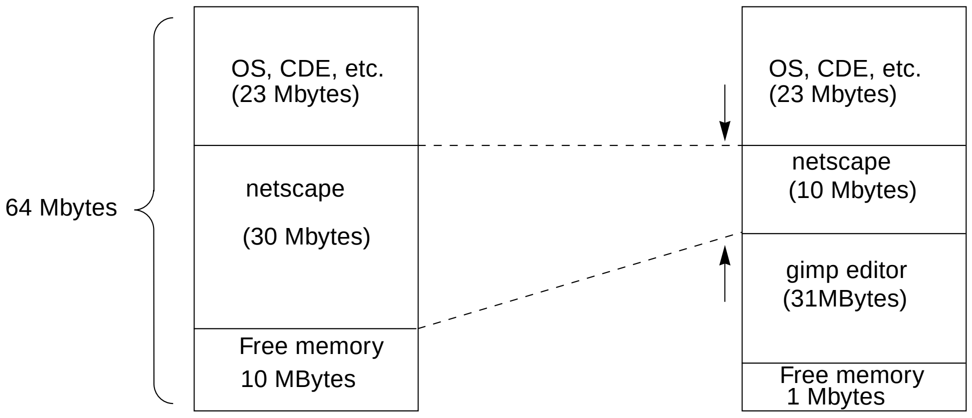

FIGURE 3-5 Memory Allocation with `netscape` and `gimp`

This situation can be avoided by configuring physical memory in the system so that there is always enough memory for each application's requirements. In example shown in FIGURE 3-5, if we configured the system with 128 Mbytes of memory, then both `netscape` and `gimp` could execute at the same time without affecting each other.

However, there is another consumer of memory in the operating system that often causes application memory starvation. That consumer is the file system. The file system uses memory from the free memory pool just like any other application in the system. Because the memory system implements equal importance by default, the file system can squeeze applications in the same way `gimp` did in FIGURE 3-5. Reading a file through the file system causes the file system to use physical memory to cache the file. This memory is consumed in 8-kilobyte chunks as the file is read. The free memory pool is thus depleted and the memory system starts looking for memory that it can use to replenish the free pool. The memory system will take memory from other applications that haven't used portions of their memory recently. For example, if you start a file-based mail tool such as `dtmail`, the memory used to cache the file when `dtmail` reads a 23-Mbyte mail file will be taken from other portions of the system.

Let's revisit the example shown FIGURE 3-5 and look at what happens when we factor in the file system memory usage. FIGURE 3-6 shows the same 64-Mbytes system where we start `dtmail` while `netscape` is running. Again, if we increase the memory to 128 Mbytes, we will provide enough memory for `netscape`, the `dtmail` application, and the `/var/mail` file. What happens if the file we are accessing is many times larger than the memory in the system (for example, several gigabytes)?

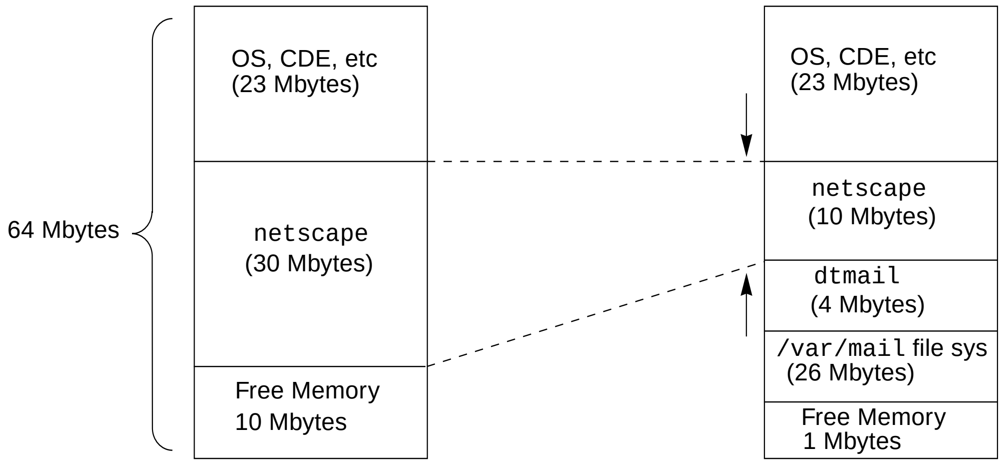

FIGURE 3-6 Memory Allocation with `netscape` and `dtmail`

Priority Paging—Memory Policy by Importance

The Solaris feature priority paging prevents the file system from consuming too much memory. Priority paging implements a memory policy with different importance factors for different memory types. Application memory is allocated at a higher priority than file system memory thus preventing the file system from stealing memory from other applications. Priority paging is implemented in the Solaris 7 operating environment, but it must be enabled with an `/etc/system` parameter as follows:

```
*
* /etc/system file
*
set priority_paging=1
```

To use priority paging with the Solaris 2.6 release, use kernel patch 105181-13; with the Solaris 2.5.1 release, use kernel patch 103640-26 or higher.

With priority paging enabled, the memory system behaves differently. In our example, rather than shrinking Netscape from 30 Mbytes to 10 Mbytes, the system limits the amount of memory that is allocated to the file system. FIGURE 3-7 shows how both `netscape` and `dtmail` can exist on the same system, while the size of the file system cache is held to a reasonable limit.

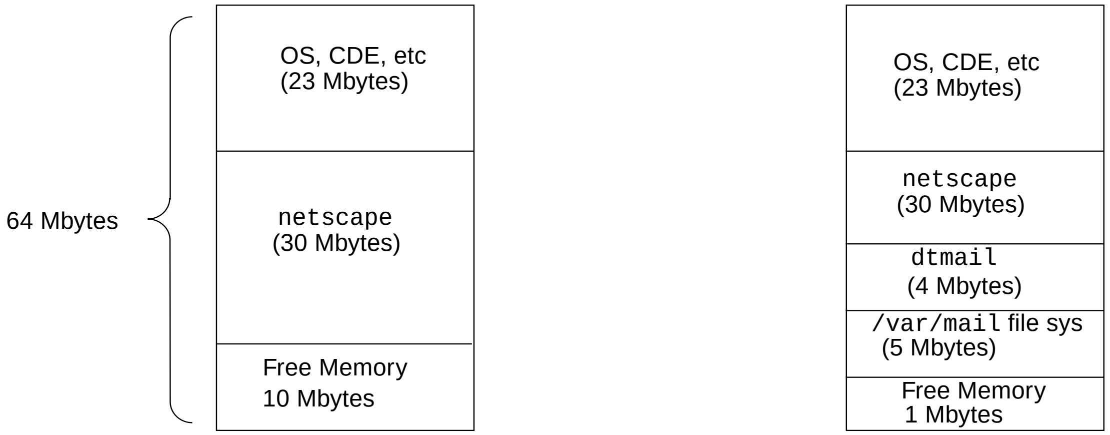

FIGURE 3-7 Memory Allocation with Priority Paging Enabled

The new memory allocation policy can be extremely important for larger systems, where memory paging problems cannot be resolved by adding additional memory. A large database system with a 50 Gbyte+ database on the file system will continuously put memory pressure on the database application with the default memory allocation policy. But priority paging will ensure that the file system only uses free memory for file system caching. As the application grows and shrinks, the size of the file system cache will grow and shrink in keeping with the amount of free memory on the system.

Network Interfaces and Network Services

TCP/IP-based network services are configured using the `/etc/services` and `/etc/inetd.conf` files so that the server responds on a particular port number. Access controls are implemented by removing and enabling particular port specifications from these files. In addition, more sophisticated access control can be implemented using TCP wrappers. These wrappers look at the incoming request packets and can deny access from unauthorized networks. To relay high volumes of traffic in and out of a secured network, a firewall is used. Access policies can be based on the destination address, protocol, or port number.

The Solaris Bandwidth Manager product provides controls on network traffic on each interface of a server. Chapter 9 explains how this product can be used as part of a resource control framework.

Network level resource managers can distribute incoming traffic over multiple systems according to the current load on each system and the network delays between the end user and the available servers. Cisco Local Director and Resonate Central Dispatch, among other products, can be used to direct traffic within a single web site, while Cisco's Distributed Director and Resonate's Global Dispatch products distribute traffic over wide areas, optimizing the wide area network delays. Cisco's products are implemented as hardware routers, while Resonate's products run as a software package on the existing server systems.

CHAPTER 4

Workload Management

Application workloads can be understood in terms of the resources they consume on each of the systems that they are distributed across.

Resource measurements are available at the system level and at the per-process level. Analysis of per-process measurements separates the raw data into application workloads. When several applications are consolidated onto a single system, resource contention can occur. Analysis determines which resource is suffering from contention and which application workloads are involved. This chapter describes several tools that help perform process-based analysis.

This chapter also describes several diverse, real-life situations and the appropriate tools to use with them.

- The first scenario covers resource management issues that occur in Internet service provider (ISP) environments.
- The second scenario portrays at consolidated commercial workloads, which include databases and file services for UNIX systems using NFS and batch workloads.
- The third scenario looks at batch workloads, which are required by most commercial installations.

Workload Analysis Tools

In a distributed environment with discrete applications on separate systems, workloads are analyzed by monitoring the total resource usage of each whole system. The busiest systems can be identified and tuned or upgraded. This approach does not work when multiple workloads are combined on a single system. While all performance monitoring tools can tell you how busy the CPU is in total, few of them can aggregate all the processes that make up a workload and tell you the amount of resource per-user that workload is using.

Customized Process Monitors

The Solaris software provides a great deal of per-process information that is not collected and displayed by the `ps` command or the SyMON software. The data can be viewed and processed by a custom-written process monitor. You could write one from scratch or use the experimental scripts provided as part of the SE (SymbEl Engine) Toolkit. The SE Toolkit is freely available for Solaris systems and is widely used. However, it is not a Sun-supported product. It can be downloaded from the URL `http://www.sun.com/sun-on-net/performance/se3`. The SE Toolkit is based on an interpreter for a dialect of the C language and provides all the per process information in a convenient form that can then be processed further or displayed. The available data and the way the SE Toolkit summarizes it is explained in detail in Chapter 5. For now, we are only interested in how this tool can be used.

The SE Process Class

The basic requirement for the process class is that it should collect both the `psinfo` and `usage` data for every process on the system. The `psinfo` data is what the `ps` command reads and summarizes. The `usage` data is extra information that includes the microstate accounting timers. Sun's developer-oriented Workshop Analyzer uses this data to help tune code during application development, but it is not normally collected by system performance monitors. For consistency, all the data for all processes should be collected at one time and as quickly as possible, then offered for display, one process at a time. This avoids the problem in the `ps` command where the data for the last process is measured after all the other processes have been measured and displayed, so the data is not associated with a consistent timestamp.

The `psinfo` data contains a measure of recent average CPU usage, but what you really want is all data measured over the time interval since the last reading. This is complex because new processes arrive and old ones die. Matching all the data is not as trivial as measuring the performance changes for the CPUs or disks in the system. Potentially, tens of thousands of processes must be tracked.

The code that implements this is quite complex, but all the complexity is hidden in the class code in `/opt/RICHPse/include/process_class.se`. The result is that the most interesting data is available in a very easy-to-use form and a simple script can be written in SE to display selected process information in a text based format similar to the `ps` command. To change the displayed data, it is easy to edit the `printf` format statement in the script directly.

Using the `pea.se` Script

The `pea.se` script is an extended process monitor that acts as a test program for `process_class.se` and displays useful information that is not extracted by standard tools. It is based on the microstate accounting information described in Chapter 5.

```
Usage: se [-DWIDE] pea.se [interval]
```

The script runs continuously and reports on the average data for each active process in the measured interval. This reporting is very different from tools such as `ps`, that print the current data only. There are two display modes: an 80-column format (which is the default and is shown in FIGURE 4-15) and the wide mode, which displays much more information and is shown in FIGURE 4-16. The initial data display includes all processes and shows their average data since the process was created. Any new processes that appear are also treated this way. When a process is measured a second time and is found to have consumed some CPU time, its averages for the measured interval are displayed. Idle processes are ignored. The output is generated every ten seconds by default. The script can report only on processes that it has permission to access. So it must be run as root to see everything in the Solaris 2.5.1 operating environment. However, it sees everything in the Solaris 2.6 operating environment without root permissions.

```
% se pea.se
09:34:06 name  lwp   pid  ppid   uid   usr%   sys% wait% chld%   size   rss   pf
olwm             1   322   299  9506   0.01   0.01  0.03  0.00   2328  1032  0.0
maker5X.exe      1 21508     1  9506   0.55   0.33  0.04  0.00  29696 19000  0.0
perfmeter        1   348     1  9506   0.04   0.02  0.00  0.00   3776  1040  0.0
cmdtool          1   351     1  9506   0.01   0.00  0.03  0.00   3616   960  0.0
cmdtool          1 22815   322  9506   0.08   0.03  2.28  0.00   3616  1552  2.2
xterm            1 22011  9180  9506   0.04   0.03  0.30  0.00   2840  1000  0.0
se.sparc.5.5.1   1 23089 22818  9506   1.92   0.07  0.00  0.00   1744  1608  0.0
fa.htmllite      1 21559     1  9506   0.00   0.00  0.00  0.00   1832    88  0.0
fa.tooltalk      1 21574     1  9506   0.00   0.00  0.00  0.00   2904  1208  0.0
nproc 31  newproc 0  deadproc 0
```

FIGURE 4-15 Output from the `pea.se` Command

Note – The font size in FIGURE 4-16 has been severely reduced.

```
% se -DWIDE pea.se
09:34:51 name  lwp   pid  ppid   uid   usr%   sys% wait% chld%   size   rss   pf  inblk outblk chario   sysc   vctx
ictx   msps
maker5X.exe     1 21508    1  9506   0.86   0.36  0.10  0.00  29696 19088  0.0   0.00   0.00   5811   380  60.03   0.30   0.20
perfmeter       1   348    1  9506   0.03   0.02  0.00  0.00   3776  1040  0.0   0.00   0.00    263    12   1.39   0.20   0.29
cmdtool         1 22815   322  9506   0.04   0.00  0.04  0.00   3624  1928  0.0   0.00   0.00    229     2   0.20   0.30   0.96
se.sparc.5.5.1   1  3792   341  9506   0.12   0.01  0.00  0.00   9832  3376  0.0   0.00   0.00      2     9   0.20   0.10   4.55
se.sparc.5.5.1   1 23097 22818  9506   0.75   0.06  0.00  0.00   1752  1616  0.0   0.00   0.00    119     19   0.10
0.30  20.45
fa.htmllite     1 21559    1  9506   0.00   0.00  0.00  0.00   1832    88  0.0   0.00   0.00      0     0   0.10   0.00   0.06
nproc 31  newproc 0  deadproc 0
```

FIGURE 4-16 Output from the `-DWIDE pea.se` Command

The `pea.se` script is 90 lines of code containing a few simple `printf`s in a loop. The real work is done in `process_class.se` (over 500 lines of code). It can be used by any other script. The default data shown by `pea.se` consists of:

- Current time and process name
- Number of lwps for the process, so you can see which are multithreaded
- Process ID, parent process ID, and user ID
- User and system CPU percentage measured accurately by microstate accounting
- Process time percentage spent waiting in the run queue or for page faults to complete
- CPU percentage accumulated from child processes that have exited
- Virtual address space size and resident set size, in Kbytes
- Page fault per second rate for the process over this interval

When the command is run in wide mode, the following data is added:

- Metadata input and output blocks per second
- Characters transferred by read and write calls
- System call per second rate over this interval
- Voluntary context switches, where the process slept for a reason
- Involuntary context switches where the process was interrupted by higher priority work or exceeded its time slice
- Milliseconds per slice or the calculated average amount of CPU time consumed between each context switch

Several additional metrics are available and can be substituted or added to the display by editing a copy of the script.

Workload Based Summarization

When you have numerous processes, group them together to make them more manageable. If you group them by user name and command, then you can form workloads, which is a powerful way to view the system. The SE Toolkit also includes a workload class, which sits on top of the process class. It pattern matches on user name, command and arguments, and processor set membership. It can work on a first-fit basis, where each process is included only in the first workload that matches. It can also work on a summary basis, where each process is included in every workload that matches. By default, the code allows up to 10 workloads to be specified.

The `pw.se` Test Program for Workload Class

Specifying workloads is the challenge. For simplicity, `pw.se` uses environment variables. The first variable is `PW_COUNT`, the number of workloads. This is followed by `PW_CMD_n`, `PW_ARGS_n`, `PW_USER_n`, and `PW_PRSET_n` where `n` goes from 0 to `PW_COUNT -1`. If no pattern is provided, `pw.se` automatically matches anything. If you run `pw.se` with nothing specified all processes are accumulated into a single catch-all workload. The `size` value is accumulated because it is related to the total swap space usage for the workload although it is inflated due to shared memory. The `rss` value is not, as too much memory is shared for the result to have any useful meaning. The final line also shows the total accumulated over all workloads.

```
16:00:09 nproc 61  newproc 0  deadproc 0
wk  command   args    user procs   usr%   sys% wait% chld%   size   pf
 0                            59   26.3    2.0   5.6   0.0 180900    2
 1                             0    0.0    0.0   0.0   0.0      0    0
 2                             0    0.0    0.0   0.0   0.0      0    0
 3                             0    0.0    0.0   0.0   0.0      0    0
 4                             0    0.0    0.0   0.0   0.0      0    0
 5                             0    0.0    0.0   0.0   0.0      0    0
 6                             0    0.0    0.0   0.0   0.0      0    0
 7                             0    0.0    0.0   0.0   0.0      0    0
 8                             0    0.0    0.0   0.0   0.0      0    0
 9                             0    0.0    0.0   0.0   0.0      0    0
10   Total      *       *     59   26.3    2.0   5.6   0.0 180900    2
```

FIGURE 4-17 Example Output from pw.se

It is easier to use the `pw.sh` script that sets up a workload suitable for monitoring a desktop workstation that is also running a Netscape web server.

```
% more pw.sh
#!/bin/csh

setenv PW_CMD_0 ns-httpd
setenv PW_CMD_1 'se.sparc'
setenv PW_CMD_2 'dtmail'
setenv PW_CMD_3 'dt'
setenv PW_CMD_4 'roam'
setenv PW_CMD_5 'netscape'
setenv PW_CMD_6 'X'
setenv PW_USER_7 $USER
setenv PW_USER_8 'root'
setenv PW_COUNT 10
exec /opt/RICHPse/bin/se -DWIDE pw.se 60
```

FIGURE 4-18 Sample `pw.se` Configuration Script

The script runs with a one minute update rate and uses the wide mode by default. It is useful to note that a workload that has a high `wait%` is either being starved of memory (waiting for page faults) or of CPU power. A high number of page faults for a workload indicates that it is either starting a lot of new processes, doing a lot of file system I/O, or that it is short of memory.

```
16:00:09 nproc 61  newproc 59  deadproc 0
wk  command    args    user count   usr%   sys% wait% chld%   size   pf pwait% ulkwt% chario   sysc   vctx   ictx   msps
 0 ns-httpd                     0    0.0    0.0   0.0   0.0      0    0    0.0    0.0      0      0      0      0   0.00
 1 se.sparc                     1   26.0   13.8   0.8  24.4   2792    6    0.8    0.0  20531      0      9     15  16.55
 2   dtmail                     0    0.0    0.0   0.0   0.0      0    0    0.0    0.0      0      0      0      0   0.00
 3       dt                     3    0.0    0.0   0.0   0.0  11132    0    0.0    0.0     64      0      0      0   0.00
 4     roam                     2    1.3    0.2   0.0   0.0  31840    0    0.0    0.0   8978      0      2      2   3.74
 5 netscape                     2    0.1    0.0   0.0   0.0  34668    0    0.0    0.0   1081      0      0      0   0.00
 6        X                     2    3.2    0.8   0.0   0.0  27188    0    0.0    0.0   4571      0     10      3   3.09
 7                 adrianc     16    0.0    0.0   0.1   0.8  32048    1    0.1    0.0    102      0      9      3   0.03
 8                    root     33    0.0    0.0   0.0   6.4  41224    1    0.0    0.0     47      0      0      3   0.09
 9                              0    0.0    0.0   0.0   0.0      0    0    0.0    0.0      0      0      0      0   0.00
10    Total       *       *    59   30.6   14.9   0.9  31.5 180892    7    0.9    0.0  35375      0     21     23  10.32
```

FIGURE 4-19 Example Output from Configured pw.se

Process Rule

Once you have collected the data, you can write a rule that examines each process or workload and determines, using thresholds, whether that workload is CPU-bound, memory-bound, I/O bound, or suffering from some other problem. This information can then be used as input to determine the resources that must be increased for workloads that are under-performing. A prototype of this rule is implemented in the SE Toolkit, and it can produce the kind of information shown in FIGURE 4-20. The example was taken while saving changes to a large file, hence the process was detected to be I/O bound. In this example the threshold is set to zero. In practice, you would set higher thresholds.

```
% se pry.se 791
...
monpid 791  nproc 94  newproc 0  deadproc 0
15:34:45 name lwmx   pid  ppid   uid   usr%   sys% wait% chld%   size   rss   pf  inblk outblk chario
sysc   vctx   ictx   msps
maker5X.exe      1   791     1  9506   4.77   1.41  0.45  0.00  25600 22352  1.7   0.00   0.00 139538
858  27.13   5.84   1.87
  amber: IObound

amber:  0.2%   [ 0.0%  ] process is delayed by page faults in data file access
```

FIGURE 4-20 Sample Process Monitoring Rule

Now we will look at some sample workloads and see how the tools for workload analysis can be used in practice.

Internet Service Provider Workloads

The Internet provides a challenge for managing computer systems. Instead of a small population of known users who can connect to a server system, millions of users can connect to any server that is on the Internet. External synchronizing events can cause a tidal wave of users to arrive at the server at the same time, as shown in the following examples.

- Sports-related web sites in the USA get a peak load during key games in the “March madness” college basketball season. In general, any television or news media event may advertise a URL to which users can connect, and cause a spike in activity.
- In countries where local phone calls are charged by the minute, the onset of cheap rate calls at 6:00 p.m. each day causes a big spike in the number of dial-in connections at an ISP, creating a major load on the local servers at that time.

- Proxy caching web servers sit between a large number of users and the Internet and funnel all activity through the cache. They are used in corporate intranets and at ISPs. When all the users are active at once, regardless of where they are connecting to, these proxy cache servers get very busy.

On the other hand, many web sites get little or no activity most of the time. It is too expensive to dedicate a single computer system to each web site, so a single system may be configured to respond to hundreds or thousands of internet addresses. This is sometimes known as virtual web hosting. From a user perspective it is not possible to tell that accesses to the different web sites are going to the same physical system. If one of the virtual web sites becomes very busy, the performance of accesses to all the other virtual sites can be affected, and resource management tools are needed to help maintain a fair quality of service for all the sites.

This section looks at three case studies: a proxy web cache workload, virtual web hosting, and a content provider site.

Proxy Web Cache Workload

A caching web server acts as an invisible intermediary between a client browser and the servers that provide content. It cuts down on overall network traffic and provides administrative control over web traffic routing. Performance requirements are quite different from those for a regular web server. After discussing the issues, we'll look at several metrics that must be collected and analyzed to determine the workload mix and the management of resources.

We start by remembering the caching principle of temporal and spacial locality. In the case of a web cache, cacheable objects are always separate and are always read in their entirety with no prefetching. The proxy web cache mostly works by using temporal locality. If cached items are read more than once by different users in a reasonably short time interval, the cache will work well. If every user reads completely different pages, the cache will just get in the way. If one user rereads the same pages, that browser will tend to cache the pages on the client. So the proxy server cache won't be used effectively.

Not all web content is cacheable. Some of the busiest traffic is dynamic by nature, and caching it would prevent the browser from seeing an update.

Cache Effectiveness

Caches commonly have a hit rate of about 20 to 30 percent, with only 60 percent of the data being cacheable. That figure does not take into account the size of each access—just the total number of accesses. Each cache transaction takes some time to complete and adds significant latency to the connection time. In other words, the cache slows down end users significantly, and only a small proportion of the data read is supplied from the cache. All the users go through the cache regardless of their final destination. This funnel effect can increase network load at a local level because accesses to local web servers will normally go via the cache, causing two trips over the local network rather than one.

Direct web service, as shown in FIGURE 4-21, allows the browser to contact web servers directly whenever possible. This contact may involve a firewall proxy that does not provide caching to get out to the public Internet.

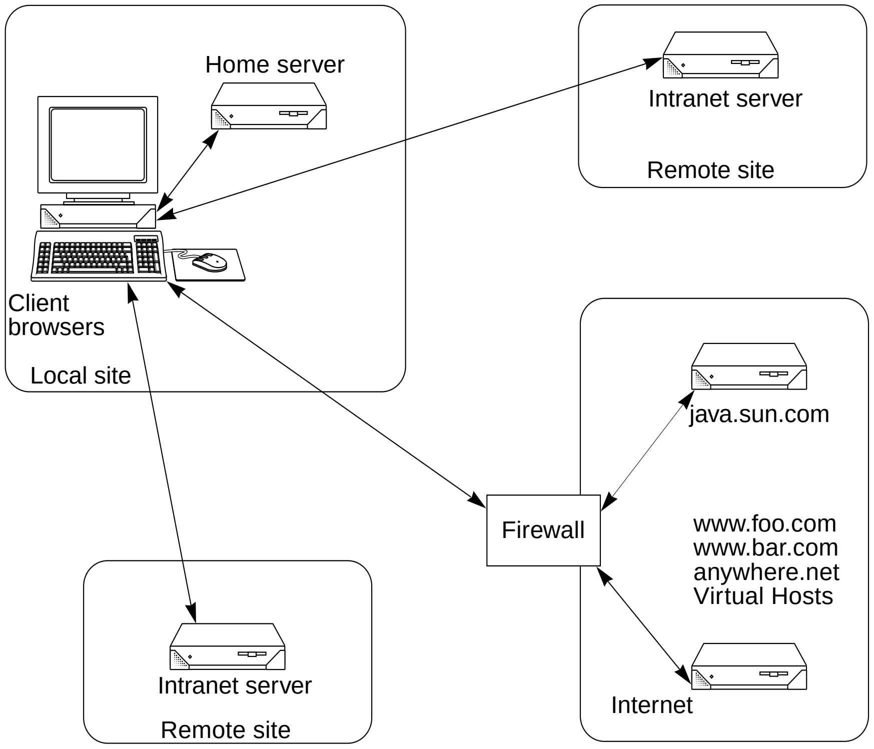

FIGURE 4-21 Direct Web Service Without Proxy Cache

In contrast, when a proxy is introduced, as shown in FIGURE 4-22, all the browsers connect to the proxy by default. An exception can be made for named systems and domains that can be accessed more efficiently by a direct connection.

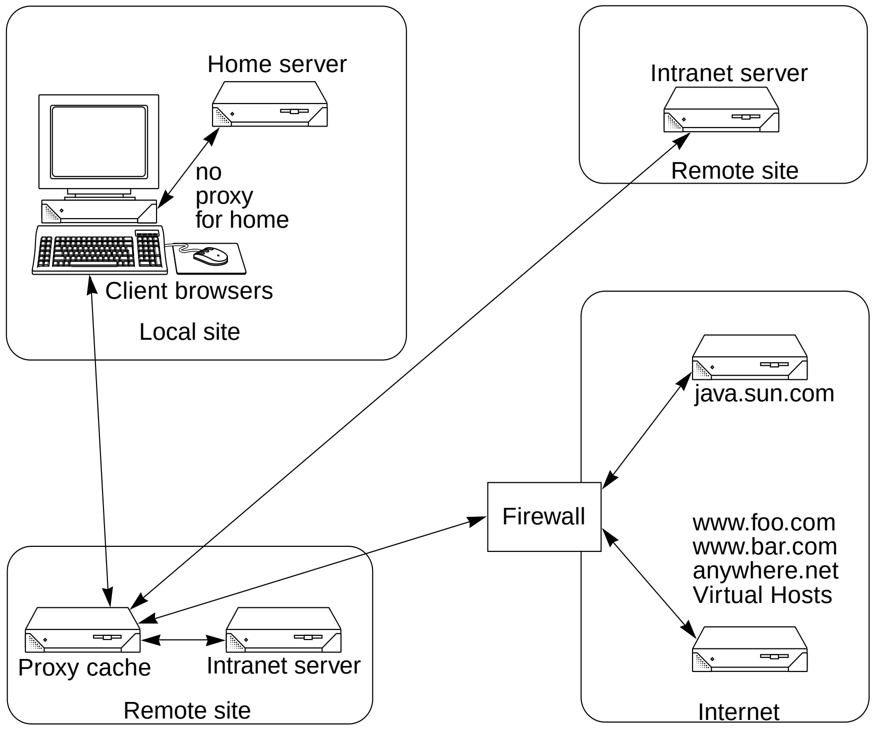

FIGURE 4-22 Indirect Web Service

Caching for Administrative Control

The reduction in wide-area network traffic from a 20 to 30 percent hit rate is worth having, especially since wide-area links are expensive and may be saturated at peak times.

The real reason to set up a proxy cache intranet infrastructure is the administrative control. Security is a big problem. One approach is to set up the firewall gateways to the Internet so that they route web traffic only to and from the proxy caches. The proxy caches can make intelligent routing decisions. For example, Sun has connections to the Internet in Europe and the USA. If a user in Europe accesses an Internet web site in Europe, that access is routed via the local connection. When an access is made to a site in the USA, a choice can be made. The route could go to the USA over Sun’s private network to the USA-based gateway, or it could go directly to the Internet in Europe and find its own way to the USA. The second option reduces the load on expensive private transatlantic links, but the first option can be used as a fallback if the European connection goes down.

Restricted routing also forces every end user who wants to get out to the Internet to do so via a proxy cache. The cache makes routing and filtering decisions and logs every access with the URL and the client IP address (which is usually sufficient to identify the end user). If the corporate policy is “Internet access is provided for business use only during business hours,” then employees who clog up the networks with non-business traffic can be identified. It is probably sufficient to make it known to the employees that their activity is being logged and that they can be traced. Posting an analysis of the most visited sites during working hours for everyone to look at would probably have a sobering effect. Filtering can be added to deny access to popular but unwelcome sites.

The point is that you have a limited amount of expensive wide-area network capacity. There is a strong tendency for end users to consume all the capacity that you provide. Using gentle peer pressure to keep the usage productive seems reasonable and can radically improve the performance of your network for real business use.

Solaris Bandwidth Manager software can be used effectively in this environment to implement policies based on the URL, domain or protocol mix that is desired. For example, if traffic to an Internet stock quote server is crowding out more important traffic in the wide area links of a company intranet, Solaris Bandwidth Manager software can throttle access to that site to a specified percentage of the total. The combination of proxy cache controls and bandwidth management is a powerful and flexible way to optimize use of wide area network and Internet gateway bandwidth.

Clustered Proxy Cache Architectures

The Apache and Netscape™ proxy caches are extensions of a conventional web server. They are used singly, and although they can be used in a hierarchy of caches, there is no optimization between the caches. An alternative is a clustered cache. The Harvest Cache was the original development and is now a commercial product. The Squid cache is a freely available spin-off and is the basis for some commercial products. Squid is used at the biggest cache installations, caching traffic at the country level for very large Internet service providers.

Clustered caches use an intercache protocol (ICP) to talk among themselves and form an explicit hierarchy of siblings and parents. If the load would overwhelm a single machine or if high availability is important, multiple systems are configured as siblings. Each sibling stores data in its cache but also uses the ICP to search the caches of other siblings. The net result: the effective cache size is that of all the siblings combined, the hit rate is improved, and it doesn’t matter which sibling a client visits. The parent-child relationships also form a more efficient hierarchy because the ICP-based connections are much more efficient than individual HTTP transfers. ICP connections are kept open, and transfers have lower latency. When using bandwidth management in a clustered cache architecture, give the inter-cache protocol high priority so that the response time for cache queries does not get affected when the caches are generating a lot of network traffic.

Virtual Web Hosting

A huge number of people and businesses have registered their own personal domain names and want the domain to be active on the internet with a home page. In many cases, there is little activity on these names, but they are assigned a fixed IP address, and somewhere a web server must respond to that address. In the Solaris operating environment, it is possible to configure more than one IP address on a single network interface. The default limit is 256 virtual addresses per interface, but it has tested up to 8000 addresses per interface and could go higher. Use the `ifconfig` command to assign IP addresses to an interface and specify virtual interfaces with a trailing colon and number on the interface name. Use the `ndd` command to query or set the maximum number of addresses per interface. In the example below, two addresses are configured on hme0, and the default of up to 256 addresses is set.

```
% ifconfig -a
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
        inet 127.0.0.1 netmask ff000000
hme0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
        inet 129.136.134.27 netmask ffffff00 broadcast 129.136.134.255
hme0:1: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
        inet 129.136.134.26 netmask ffffff00 broadcast 129.136.134.255
hme1: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
        inet 129.136.23.11 netmask ffffff00 broadcast 129.136.23.255

# /usr/sbin/ndd -get /dev/ip ip_addrs_per_if
256
```

The real limit to the number of virtual addresses is the time it takes to run `ifconfig` command on all of them. When thousands of virtual addresses are configured, the reboot time is affected, which can be a significant problem in a high availability failover scenario. The Solaris 2.6 operating environment was tuned to speed up this process.

Web servers can be configured in several ways to manage many virtual addresses. The most efficient way is to use a feature of the web server so that a single instance of the server listens to many addresses and responds with the appropriate home page. With a multithreaded web server, one process can support many addresses. With web servers like Apache that use a separate process for each connection, more processes are spawned for the addresses that are most active at any point.

Administrative problems may rule out this approach. It may be necessary to run a completely separate web instance of the web server for each address. This way the web server runs using a distinct userid. The administrator of the web site can log in as that user to configure their own site. The web server and any search engine or `cgi-bin` scripts that it starts all run as that user so that accounting, performance monitoring and resource management tools can manage each virtual web site's activity separately. It is common for `cgi-bin` scripts to have errors and be resource intensive. They can get stuck in infinite loops and consume an entire CPU. With thousands of customer web sites setup on a single system, the central administrator cannot control the quality of the custom `cgi-bin` code that is set up on each site.

In the end, this workload is similar in many ways to an old-fashioned central timeshare system. Resource management using the SRM software to control the CPU usage and the number of processes and logins on a per-user basis is extremely effective. The Solaris Bandwidth Manager software can also be used effectively, as there is an explicit administrative mapping from each network address to the userid that is running that web server. Users can be allocated shares of the CPU and network resources according to their need, or can pay more for a larger share and be confident that they will get a better quality of service if the server saturates.

Managing Web Servers With SRM Software

The SRM software manages resources on web servers by controlling the amount of CPU and virtual memory. Three basic topologies are used on systems hosting web servers.

Resource Managing a Consolidated Web Server

A web server can be managed by controlling the resources it can use. In an environment where a web server is being consolidated with other workloads, this basic form of resource management prevents other workloads from affecting the performance of the web server, and vice versa.

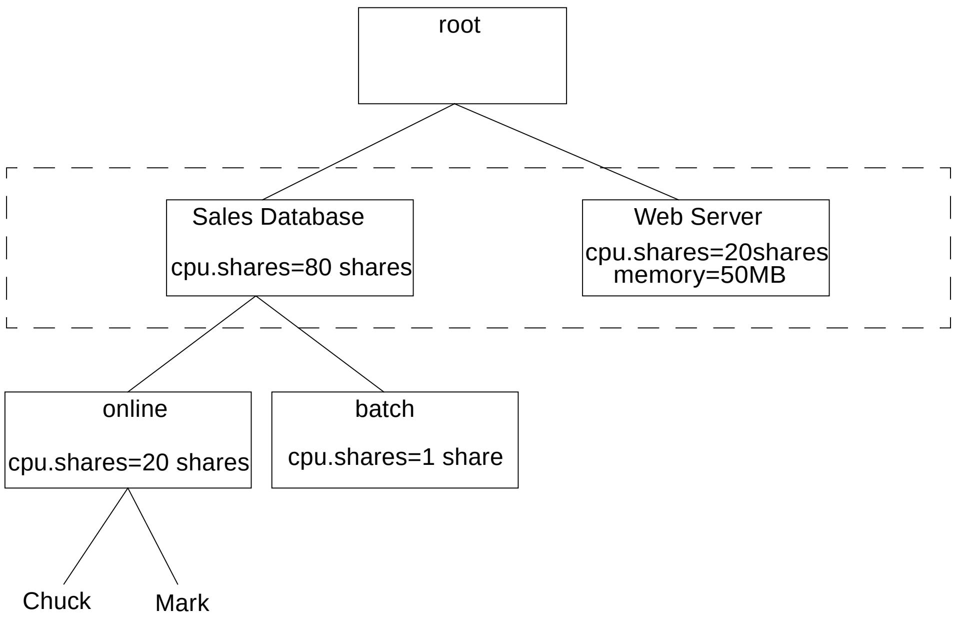

FIGURE 4-23 Resource Management of a Consolidated Web Server

In FIGURE 4-23, the web server is allocated 20 shares, which means that it is guaranteed at least 20 percent of the processor resources should the database place excessive demands on the processor. In addition, if a `cgi-bin` process in the web server runs out of control with a memory leak, the entire system will not run out of swap space; only the web server will be affected.

Managing Resources Within a Single Web Server

It is often necessary to use resource management to control the behavior within a single web server. For example, a single web server may be shared among many users, each with their own CGI-BIN programs. An error in one CGI-BIN program could cause the entire web server to run slowly or, in the case of a memory leak, could even bring down the web server. To prevent this from happening, use the per-process limits.

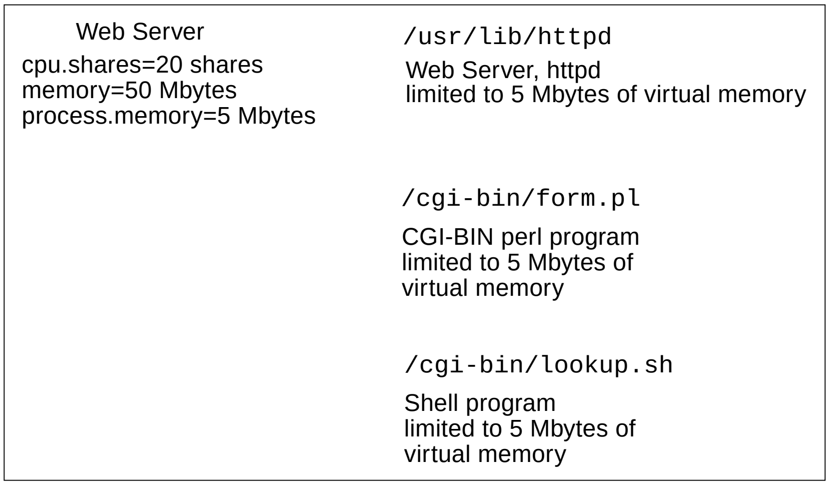

FIGURE 4-24 Resource Management of a Single Web Server

Resource Management of Multiple Virtual Web Servers

Single machines often host multiple virtual web servers. In such cases, there are multiple instances of the `httpd` web server process, and far greater opportunity exists to exploit resource control using the SRM software.

You can run each web server as a different UNIX userid by setting a parameter in the web server configuration file. This effectively attaches each web server to a different lnode in the SRM hierarchy. For example, the Solaris Web Server has the following parameter, as shown in FIGURE 4-25, in the configuration file, `/etc/http/httpd.conf`

```
# Server parameters
server  {
  server_root                   "/var/http/"
  server_user                   "webserver1"
  mime_file                     "/etc/http/mime.types"
  mime_default_type             text/plain
  acl_enable                    "yes"
  acl_file                      "/etc/http/access.acl"
  acl_delegate_depth            3
  cache_enable                  "yes"
  cache_small_file_cache_size   8                     # megabytes
  cache_large_file_cache_size   256                   # megabytes
  cache_max_file_size          1                      # megabytes
  cache_verification_time       10                      # seconds
  comment                   "Sun WebServer Default Configuration"

  # The following are the server wide aliases

  map   /cgi-bin/             /var/http/cgi-bin/             cgi
  map   /sws-icons/            /var/http/demo/sws-icons/
  map   /admin/                /usr/http/admin/

# To enable viewing of server stats via command line,
# uncomment the following line
  map   /sws-stats            dummy                        stats
}
```

FIGURE 4-25 Solaris Web Server Parameter File

By configuring each web server to run as a different UNIX userid, you can set different limits on each web server. This is particularly useful for control and for accounting for resource usage on a machine hosting many web servers. You can make use of most or all of the SRM resource controls and limits:

Shares [`cpu.shares`]	The CPU shares can proportionally allocate resources to the different web servers.
Mem limit [`memory.limit`]	The memory limit can limit the amount of virtual memory that the web server can use. This prevents any one web server from causing another to fail from memory allocation.

Proc mem limit [`memory.plimit`]	The per-process memory limit can limit the amount of virtual memory a single `cgi-bin` process can use. This stops any `cgi-bin` process from bringing down its respective web server.
Process limit [`process.limit`]	The maximum number of processes allowed to attach to a web server. It effectively limits the number of concurrent `cgi-bin` processes.

Commercial Workloads

If you want to consolidate or control the relative importance of applications in commercial workloads, you must manage system resources. For example, if you want to mix decision support and OLTP applications on the same system, you need resource management so that the decision support does not affect the response times for the users of the OLTP system. Even without consolidation, workload resource management is an important tool that can provide resource allocation for users according to importance. (For example, in the case where telesales order users have a higher importance than general accounting users.)

This section examines some aspects of the resource management that apply to these types of problems. It focuses on workload consolidation and resource management of databases and analyzes the following workloads:

- NFS Servers
- Databases
- Batch workloads
- Mail servers

Workload Consolidation

Workload consolidation is the most requested form of commercial workload resource management. It allows multiple applications to co-exist in the same Solaris operating environment with managed resource allocation so that one workload does not adversely affect another. The success of workload consolidation is bound closely to the ability to partition resources between the applications, and it is important to understand the types of resources being managed and how they interact.

The resources used by commercial applications fall into the following categories:

- CPU cycles

- Physical memory
- Virtual memory and swap space
- Disk bandwidth
- Disk space
- Network bandwidth

Each application uses these resources differently, and the ability to consolidate one application with another is governed by how effectively you can manage each resource. Sometimes resources cannot be managed by the Solaris operating environment. These types of applications cannot be mix with others. However, in most cases, combinations of resource management products can solve this problem.

The NFS Server Workload

The unique characteristics of an NFS workload place multiple resource demands on the system. No single product can provide complete resource management of an NFS workload. The resources used by NFS and the different products or techniques that can be used to manage them are shown in TABLE 4-1.

TABLE 4-1 NFS Workload Resources

Resource Type	Management Product/ Method	Comments and Issues
CPU	Processor Sets / nfsd / Solaris Bandwidth Manager	Limit by constraining the system processor set. Control the number of `nfsd` threads `nfsd(1M)`. Indirectly limit the number of NFS operations by using Solaris Bandwidth Manager.
Physical Memory	Priority Paging	Install and enable priority paging so that file systems will not consume all physical memory.
Swap Space	N/R	NFS uses very little swap space.
Disk Bandwidth	Separate Disks/ Solaris Bandwidth Manager	Either put the NFS file systems on a different storage device or indirectly use Solaris Bandwidth Manager to control the NFS request rate.
Disk Space	File System Quotas	Use file system quotas to limit disk space allocation. See “Base Solaris Operating Environment” on page 136.
Network Bandwidth	Solaris Bandwidth Manager	Use Solaris Bandwidth Manager to control the rate of NFS operations.

Controlling NFS CPU Usage

NFS servers are implemented in the Solaris kernel as kernel threads and run in the system class. You can control the amount of resource allocated to the NFS server in three ways:

- Limit the number of NFS threads with `nfsd`
- Limit the amount of CPU allocated to the system class with `psrset`
- Indirectly control the number of NFS ops with Solaris Bandwidth Manager.

You can control the number of NFS threads by changing the parameters to the NFS daemon when the NFS server is started. Edit the start-up line in `/etc/rc3.d/S15nfs.server`:

```
/usr/lib/nfs/nfsd -a 16 (change 16 to number of threads)
```

The actual number of threads required will vary according to the number of requests coming in and the time each thread spends waiting for disk I/O to service the request. There is no hard tie between the maximum number of threads and the number of NFS threads that will be on the CPU concurrently. The best approach is to approximate the number of threads required (somewhere between 16 to 64 per CPU), and then find out if the NFS server is doing its job or using too much CPU time.

NFS is a fairly lightweight operation, so it is unlikely that the NFS server CPU usage is an issue. The CPU time consumed by the NFS server threads accumulates as system time. If the system time is high, and the NFS server statistics show a high rate of NFS server activity, then curtail CPU usage by reducing the number of threads.

A far more effective way to control NFS servers is to use the Solaris Bandwidth Manager product to limit the traffic on the NFS port, 2049, and indirectly cap the amount of CPU used by the NFS server. The disadvantage is that spare CPU capacity can be wasted because managing by bandwidth usage does not reveal how much spare CPU is available.

To understand if you have over constrained NFS's allocation of resources you can use the new NFS `iostat` metrics and look at the `%busy` column.

NFS Metrics

Local disk and NFS usage are functionally interchangeable, so the Solaris 2.6 operating environment was changed to instrument NFS client mount points the same way it does disks. NFS mounts are *always* shown by `iostat` and `sar`. With automounted directories coming and going more often than disks coming online, that change may cause problems for performance tools that don't expect the number of `iostat` or `sar` records to change often.

The full instrumentation includes the wait queue for commands in the client (`biod wait`) that have not yet been sent to the server. The active queue measures commands currently in the server. Utilization (`%busy`) indicates the server mount-point activity level. Note that unlike the case with simple disks, *100% busy does not indicate that the server itself is saturated*; it just indicates that the client always has outstanding requests to that server. An NFS server is much more complex than a disk drive and can handle many more simultaneous requests than a single disk drive can. This is explained in more detail later in this chapter.

The following is an example of the new `-xnP` option, although NFS mounts appear in all formats. Note that the `P` option suppresses disks and shows only disk partitions. The `xn` option breaks down the response time, `svc_t`, into wait and active times and puts the device name at the end of the line. The `vold` entry automounts floppy and CD-ROM devices.

```
% iostat -xnP
                              extended device statistics
  r/s  w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0 vold(pid363)
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0 servdist:/usr/dist
  0.0  0.5    0.0    7.9  0.0  0.0    0.0   20.7   0   1 servhome:/export/home2
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0 servmail:/var/mail
  0.0  1.3    0.0   10.4  0.0  0.2    0.0  128.0   0   2 c0t2d0s0
  0.0  0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0 c0t2d0s2
```

NFS Physical Memory Usage

NFS uses physical memory in two ways: In the kernel each NFS thread consumes some space for a stack and local data, and outside the kernel the data being served by NFS consumes memory as it is cached in the file system. The amount of kernel memory used by NFS is easily manageable because its size doesn't change much and the amount of memory required is small.

The amount of memory used by the file systems for NFS servers is, however, very large and much harder to manage. By default, any non-sequential I/O and non-8KB I/O uses memory at the rate of data passed though the file system. The amount of memory used grows continuously. When there is no more free memory, memory is taken from other applications on the system.

By using priority paging (see "Priority Paging—Memory Policy by Importance" on page 33) you can apply a different resource policy to the memory system to prevent this from happening. With priority paging, the file system can still grow to use free memory, but cannot take memory from other applications on the system. Priority paging should be mandatory for any system that has NFS as one of the consolidation applications.

NFS and Swap Space

NFS uses a very small amount of swap space, and there should be no inter-workload swap space issues from NFS.

NFS Disk Storage Management

You can manage NFS disk storage using UFS disk quotas. See “Disk Quotas” on page 130 for more information.

Controlling NFS with Solaris Bandwidth Manager Software

You can control amount of resources consumed by NFS indirectly by curtailing the amount of network bandwidth on port 2049. The Solaris Bandwidth Manager product provides the means to do this.

First assess the network interfaces that need to be controlled. If clients come in over several network interfaces, all of these interfaces must be brought under control by the Solaris Bandwidth Manager software.

When defining interfaces in the Solaris Bandwidth Manager software, you must specify whether incoming or outgoing traffic needs to be managed. In the case of NFS, network traffic could go in both directions (reads and writes). In the Solaris Bandwidth Manager configuration, this would look as follows:

```
interface hme0_in
   rate      100000000        /* (bits/sec) */
   activate  yes

interface hme_out
   rate       100000000
   activate   yes
```

Next define the service you want to manage. The Solaris Bandwidth Manager software already has two pre-defined classes for NFS:

```
service nfs_udp
    protocol udp
    ports 2049, *
    ports *, 2049

service nfs_tcp
    protocol tcp
    ports 2049, *
    ports *, 2049
```

Put in place a filter that can categorize network traffic in NFS and non-NFS traffic:

```
filter nfs_out
    src
        type    host
        address servername
    dst
        type    subnet
        mask    255.255.255.0
        address 129.146.121.0
    service
        nfs_udp, nfs_tcp
```

The filter in the above example is for managing outgoing NFS traffic to the 129.146.121.0 network. You could decide to leave out the destination and manage NFS traffic to all clients, from wherever they come.

Create another `nfs_in` filter for NFS traffic in the opposite direction. Only the `src` and `dst` parts need to be reversed.

Lastly, create a class that will allocate a specific bandwidth to this filter:

```
class managed_nfs
    interface     hme_out
    bandwidth     10
    max_bandwidth 10
    priority       2
    filter        nfs_out
```

This class sets a guaranteed bandwidth of 10 percent of the available bandwidth (10 Mbytes in case of fast Ethernet). Control the maximum bandwidth by setting an upper bound to the CPU resources that NFS consumes on the host. The key variable is `max_bandwidth`; it specifies an upper bound to the consumed bandwidth that never will be exceeded. You could even set the `bandwidth` variable to 0, but this could lead to NFS starvation if other types of traffic will be managed as well.

The `priority` variable is less important. It will be a factor if other types of traffic are being managed. Generally, higher priorities will have lower average latencies, because the scheduler gives them higher priority if it has the choice (within the bandwidth limitations that were configured).

It is not easy to find a clear correlation between NFS network bandwidth and NFS server CPU utilization. That correlation depends very much on the type of NFS workload for your server. A data-intense NFS environment that moves large files and tends to saturate networks is very different from an attribute-intense environment, which tends to saturate CPUs with lots of small requests for file attributes, such as a software development compile and build server. Experimentation will determine what's good for you. Your administrator could even develop a program that monitors NFS CPU utilization, and if it gets too high, use the Solaris Bandwidth Manager APIs to dynamically limit the bandwidth more, all automatically and in real time.

Database Workloads

Resource management of databases can be somewhat simplified by viewing the database as a "black box" and managing resources around it. This strategy is particularly useful when doing server consolidation because the main objective with consolidation is to partition one workload from an other.

In most commercial database systems several packages interact with the database to provide the overall application environment to the user. The typical data center strategy is to consolidate more workloads onto each system. Resource management of the whole environment requires careful planning and a solid understanding of the interaction between these packages and the database.

A look at a typical example where a single database instance provides database services for two different applications that require access to the same data, uncovers some of the complexities of database resource management. FIGURE 4-26 shows an typical backend database server with a web application used for customer order entry. and an ad-hoc decision support user that generates large queries against a database.

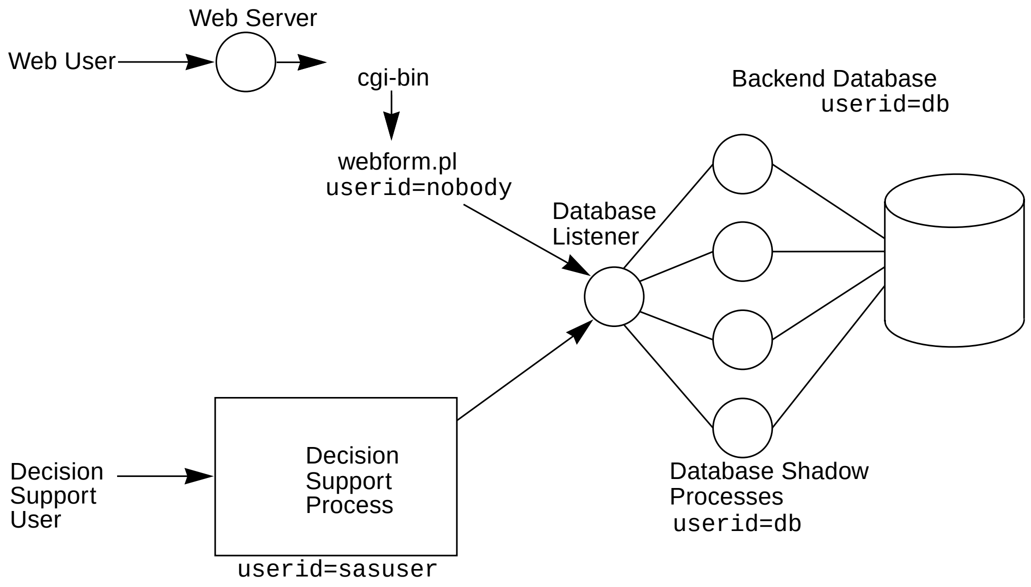

FIGURE 4-26 Practical Example with a Database Server and a descision support process

The life cycle of each transaction crosses several management boundaries. No single control or policy can assign resources to ensure that adequate CPU is provided to meet response times. For example, if the descision support user is retrieving a large number of transactions from the database, the web user may not be able to get adequate response from the database during that time. Ideally, you want to control the allocation of resources as the transaction proceeds though the system. A time line of the transaction though the system is shown in FIGURE 4-27.

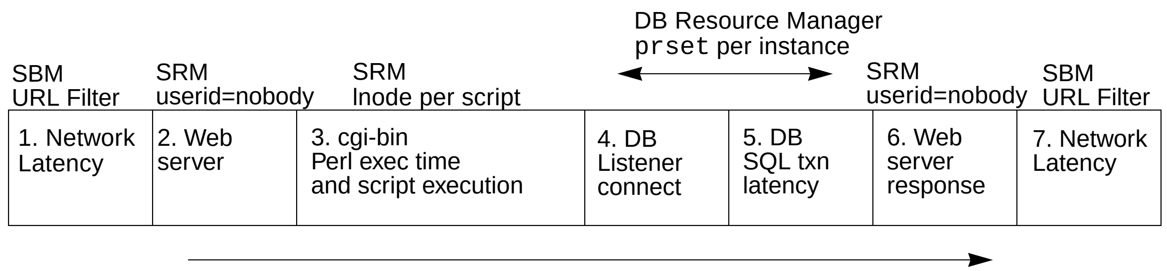

FIGURE 4-27 Transaction Flow and Resource Management Product Assignment

1. Network Latency—The user issues an `http` request from the desktop, which must travel across the intranet/Internet to the web server. This component can be managed by controlling the bandwidth and network priorities into the web server

with the Solaris Bandwidth Manager software. Often the network infrastructure between the client browser and the web server is outside the control of management (for example, the internet). Managing the bandwidth on the network port on the machine that hosts the web server is a useful strategy to ensure that key `http` requests get the required bandwidth, while other lower priority data transfers (such as an `ftp` transfer) are constrained. The Solaris Bandwidth Manager software can control the bandwidth allocation into the web server, and `cgi-bin` can be controlled using bandwidth manager URL prioritization. (See Chapter 4.)

2. The Web Server—The web server usually runs as a single user, and often as the user `nobody`. Ensure that the web server as a whole gets a sufficient share of CPU. Use SRM software to ensure that adequate CPU is allocated by assigning CPU shares to the `nobody` user.

3. The Web Server `cgi-bin` script—The `cgi-bin` script is forked by the web server, and by default also runs as the userid `nobody`. Use the SRM software to limit the amount of CPU the `cgi-bin` script uses by creating a node in the SRM hierarchy for that `cgi-bin` script and using the `srm` user command as a wrapper to the `cgi-bin` script to assign resource control limits to that point in the hierarchy. See “Managing Resources Within a Single Web Server” on page 62 for more information on how to control `cgi-bin` scripts.

4. The Database Listener—The database listener process accepts the incoming connection from database client applications. (in this case the `cgi-bin` Perl script). The database listener forks a database process and connects the client application to the database.

5. The Database Server Session—The Database session can either be a new per-client process or some portion of the existing server processes. You can control the amount of CPU allocated to the database processes by using database vendor-specific resource control tools.

6. Web Server Response—Control passes back to the web server to output the data to the client browser. The same control as in the web server component (2) is implemented here.

7. Network Latency—The response from the web server is transmitted over the network. The Solaris Bandwidth Manager software again controls the bandwidth out of the port on the host to the internet/intranet.

Each stage of the transaction uses different resources. You can apply different techniques to each stage. The more difficult part of using today’s technologies is managing the priority of the transaction once it enters the database. You can use fine-grained resource management to control the amount of CPU allocated to the users in the database, but you must ensure that external control of the database by resource management products is fully supported for the database in question.

Virtual Memory and Databases

The SRM software and resource limits can limit the amount of virtual memory used by processes, users, and workloads. This capability does not manage physical memory, but it effectively restricts the amount of global swap space consumed by each user.

When a user or workload reaches the virtual memory limit, the system returns a memory allocation error to the application, that is calls to `malloc()` fail. The same error code is reported to the application as if the application had run out of swap space.

In practice, many applications do not respond well to memory allocation errors because it is very hard to recover from a lack-of-memory error without making some additional demand for memory. Because of this, it is a high risk to ever let a database server reach its virtual memory limit. If the limit is reached, the database engine may hang or crash, and the database may have to be restarted.

It is best to set a high virtual memory limit so that the limit can never be reached under normal circumstances. However, the virtual memory limit can be used to place a ceiling on the entire database server to stop a user with a memory leak from affecting other databases or workloads on the system.

Database Consolidation

You can consolidate databases using a number of resource management techniques. Choose the technique based on the supportability and functionality of the solution. Not all databases are supported under each of the resource management facilities. Because most consolidated database servers are mission-critical production servers, this should be the primary concern.

The objective for successful consolidation is to provide strong insulation between each database instance. This requires careful resource management of CPU, memory, I/O, and network traffic.

Insulating CPU Usage

Processor usage is a relatively easy resource to isolate. There are several mechanisms by which this can be achieved. There is a trade-off between the strength of insulation and the flexibility of resource allocation. At one end of the scale is Starfire dynamic system domains, which provides the same level of insulation as separate machines but requires manual intervention to shift resources between database instances. There is also a large granularity of resource allocation, which is four processors at a time. At the other end of the spectrum is the SRM software, which provides fine-grained resource management with the ability to dynamically assign resources whenever they are needed. But it also provides much lower insulation between the applications.

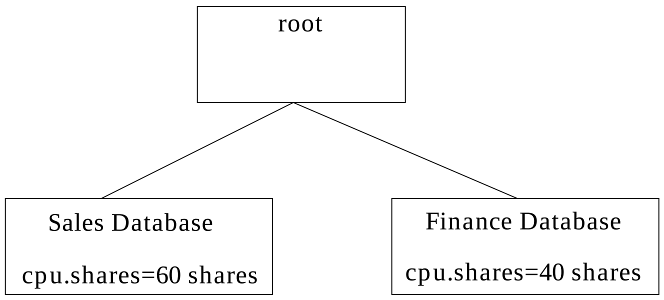

FIGURE 4-28 Consolidation of Two Databases with Solaris Resource Manager

The advantage with the SRM software is that resources are completely dynamic. If one database is not using the resources, they are available to the other database. System resources are not wasted because spare CPU cycles can be used by any other workload requiring them. The more workloads consolidated, the flatter the total resource usage becomes. By doing this, you can often size a system for the average resource requirements instead of the peaks.

Processor sets provide a middle ground. The granularity is one CPU, and the resources are easier to reallocate than with domains, but still relatively static in nature. Processor sets and domains, however, provide the static processor assignment and granularity required by some databases. In some cases, this is the only supported way of dividing resources between databases. Unfortunately, resources are statically assigned. Although some movement of resources is possible, it means that you must size each database instance for most of its peak requirements.

Insulating Memory Usage

Databases have three types of memory usage, which have very different characteristics:

- shared memory area
- private memory area
- file system paging memory

The shared memory area is often the largest component of the database's memory requirements. It is the easiest to insulate between instances because the memory is allocated as shared memory. All mainstream databases allocate this as intimate shared memory (ISM), which wires down the memory so that it can't be paged. Because this memory is wired down, the memory allocated to each instance stays allocated. One instance cannot steal memory from another. However, this poses a challenge for dynamic memory resource allocation, such as dynamic reconfiguration because the wired-down memory must be unallocated before it can be removed from the system, requiring quiescing of the database.

The private memory is the additional memory that a database uses, over and above the large shared memory segment that is typical of most databases. The private memory is the regular process memory used by each process that the database engine comprises, and it is regular pageable memory. For example, the Oracle shadow processes have a small private memory size of about 500k per process. The Oracle parallel query processes may use several megabytes per process. The private memory area requirements for each database vary, but they are generally many times smaller than the global shared memory requirements. The amount of memory used is sometimes configurable. Simple capacity planning ensures that sufficient system-wide memory exists to cater to all of the database private memory requirements. Take care when including Decision Support System (DSS) database instances because they often have huge private memory requirements.

Often overlooked are the memory requirements of the file system. By using raw devices for all tablespaces or by using the direct I/O feature of the UFS file system, the file system bypasses the Solaris page cache. In this case, we do not need to take into consideration the memory use of the file system.

The Solaris page cache causes a huge memory demand, which places undue pressure on the database private memory and breaks down any insulation between established databases. Priority paging puts a hard fence between the file systems and applications, and the file system will only use free memory in the system for its cache. If you plan to run a database on file systems, consider this a mandatory requirement. See Chapter 3 for more information.

In summary, to provide proper memory insulation between databases, ensure the following:

- Use ISM for the shared global areas to wire down memory.
- Use proper capacity planning and parameters to ensure there is sufficient RAM for private memory requirements.
- Use either raw disk devices or ensure that priority paging is installed to prevent file system memory pressure.

Insulating I/O

Ensure that I/O activity from one application does not affect another application in an undesirable way. The best way to insulate I/O within a single database instance is to make sure that application tables are placed on separate I/O devices. This still leaves shared log devices, which can be a source of I/O contention. Ensure that adequate I/O bandwidth, spindles, and, where possible, hardware caching are provided for adequate performance for the log devices.

Databases can be consolidated by simply ensuring that each database is assigned its own I/O devices. This completely insulates one database from another. However, you must also make sure that all data paths from the server to the storage device are carefully isolated. For example, two databases can be placed on their own set of disks within a Sun StorEdge™ A3500 storage system. However, because both sets of disks use the same SCSI channel to the host the databases, I/O is not properly insulated.

If you must use a single storage controller, use capacity planning so that sufficient bandwidth is available to combine both. For OLTP applications this is rarely an issue because the bandwidth requirements are so low. For example, an OLTP system generating 2000 8-kilobyte IOPS only requires 16 Mbytes of I/O bandwidth. A decision support application is completely different. A single decision support workload can generate several hundred megabytes a second of I/O. In addition to this, decision support often generates enough I/O to overwrite the cache continuously (cache wiping), which hurts the OLTP random I/O that makes extensive use of the storage cache. Do not consider a single storage controller for consolidation of decision support workloads with OLTP workloads.

Databases in a Resource Managed Environment

Relationships with databases and resource management products are many and vary with each database type. For example, if you add CPU resources to an instance of Oracle 7, the Oracle engine automatically picks up those resources, whereas some other databases must be restarted to use the newly available resources (it should be noted however that internal table sizes and latch sizing will not be automatically adjusted, and performance may improve after a restart of the database). To understand how different databases work within a resource management environment, look at the different database topologies and explore what happens to each database when you add or remove CPU, memory, or I/O resources.

CHAPTER 5

Workload Measurements

If you want to manage a resource, you must measure its usage.

The generic types of measurements that you can obtain directly or indirectly via related measurements are throughput, utilization, queue length and response time. These measurements are made at several levels, including business operations, application, user, system, network, process, and device level.

This chapter discusses measurement sources and the meaning of the available measurements that are important for resource management.

One problem in documenting the details of Solaris measurements is that far more measurements are available than can be displayed by the standard tools such as `vmstat`, `sar`, and `ps.` Most commercial performance tools read the kernel directly and have access to the full range of measurements, although they do not use all of them. The SyMON product collects a large proportion of the available measurements. The most convenient way to explore this data is to use the SE Toolkit. This freely available but unsupported tool provides full access to all the raw data sources in the Solaris operating environment and generates higher level processed data. It is used to prototype ideas that can then be promoted for incorporation in products, in particular for the Sun Enterprise SyMON 2.0 software.

The SE Toolkit

The SE Toolkit is based on a C language interpreter that is extended to make all the Solaris measurement interfaces available in an easy form. The code that takes metrics and processes them is provided as C source code and runs on the interpreter so that it is easy to trace where data comes from and how it is processed. You can then write your own programs in any language to obtain the same data. The SE Toolkit has been jointly developed by Richard Pettit and Adrian Cockcroft as a “spare time” activity since 1993. Richard worked at Sun but is currently at Foglight

Software, and Adrian is one of the authors of this book. The SE Toolkit can be downloaded from `http://www.sun.com/sun-on-net/performance/se3`. Detailed information and examples of how to write code that reads the kernel and the SE Toolkit itself can be found in *Sun Performance and Tuning, Java and the Internet*, by Adrian Cockcroft and Richard Pettit, Sun Press 1998.

Measurement Levels

Measurements can be classified into several levels. Data from lower levels is aggregated and merged with new data as more valuable higher level measurements are produced.

- Business operations

  Business workloads are broad-based and are not only computer oriented. Use a form that makes sense to managers and non-technical staff to represent the part of the business that is automated by the computer system.

- Applications

  The business operation can be broken down into several applications such as sales and distribution, e-commerce web service, email, file, and print. Examples of application-specific measurements are order entry rate, emails relayed, and web server response time.

- Users

  Each class of user interacts with several applications. The number of users and the work pattern of each class of users should be understood.

- Networks

  Networks connect the users to the applications and link together multiple systems to provide applications that are replicated or distributed. Measure traffic patterns and protocol mixes for each network segment.

- Systems

  System-level measurements show the basic activity and utilization of the memory system and CPUs. Some network measurements, such as TCP/IP throughput, are also available on a per system basis. Per process activity can be aggregated at a per system level then combined with network measurements to measure distributed applications.

- Processes

  Process measurements show the activity of each user and each application. Current process activity can be monitored, and accounting logs provide a record of who ran what when.

- Devices

  Devices, such as disks and network interfaces, are measured independently and aggregated at the system level. There are few ways to link the usage of a device to a process or a user automatically, so detailed information about the configuration of devices and the usage of file systems by applications is needed.

The Application Resource Measurement Standard

The Application Resource Measurement (ARM) standard aims to instrument application response times. The intent is to measure end-to-end response time by tagging each ARM call with a transaction identifier and tracking these transactions as they move from system to system. If a user transaction slows down, the system in the end-to-end chain of measurements that shows the biggest increase in response time can be pinpointed.

This technique sounds very useful, and there is both good and bad news about ARM. The good news is that all vendors support the one standard, and several implementations exist, from Hewlett-Packard and Tivoli (who jointly invented the standard) to the more recently developed BMC Best/1 and Landmark. The bad news is that application code must be instrumented to measure user response time, and sophisticated tools are needed to handle the problems of distributed transaction tracking. Application vendors have shown interest, so more measurements will become available. The latest versions of Baan products include ARM-based instrumentation.

SAP R/3 Measurements

The SAP R/3 enterprise resource planning package has three tiers of systems: user desktops, application servers, and a backend database. The application instruments itself and measures the performance of key user transactions and the response time of the backend database for each application. This information is provided by a facility called CCMS, and it is used by several tools such as BMC Best/1 to provide more detailed application-level management than can be done with just system and device-level information.

Internet Server Measurements

An Internet server complex often provides a range of services and uses applications such as firewalls, web servers, and content generation databases. Each of these applications supports a range of operations that you can log or monitor to discover the rate of use and the mixture of operations. In some cases, the time taken for each operation is also logged, so you can obtain a response time measurement.

One of the most common applications is the proxy web cache service. Because this application often acts as a funnel for all the web traffic going out of a corporate intranet to the Internet itself, a good measure of overall Internet response time can be obtained. An example scenario follows.

Configuring and Monitoring a Proxy Cache

Data is collected on a Netscape 2.5 proxy cache that is serving most of the thousand or so employees in Sun's United Kingdom facility. The server is situated at the largest site, so some users have LAN-based access speeds. However, several other sites in the UK, run over wide area links to the central site. Additionally, a number of users are connected over ISDN and modem dial-up connections.

The access log contains the same data that a normal web server log contains, but several additional values are logged. Netscape supports two extended formats and also allows a custom format to be specified. The interesting data to extract is the mixture of possible cache outcomes that determines whether it is a cache hit or not, as well as the routing information and transfer time. The SE Toolkit `percollator.se` script can parse Netscape and Squid proxy cache log formats and summarize the data.

The route data is analyzed to count all the entries that go to PROXY or SOCKS and report the percentage that are indirect. This is the percentage that gets routed to the Internet, rather than being DIRECT references to other servers in the corporate intranet or incomplete transfers marked with a "-".

The cache finish status was analyzed, and operations are divided into four categories. The NO-CHECK and UP-TO-DATE states are cache hits. The WRITTEN, REFRESHED, and CL-MISMATCH states are misses that cause cache writes. The DO-NOT-CACHE and NON-CACHEABLE states are uncacheable, and anything else is an error or incomplete transfer. The percentages for the first three are recorded.

The total transfer time is recorded. The average transfer time can be calculated, but since the transfer size varies from zero to several megabytes, the average transfer time for each of the size ranges for the mix must also be worked out. The mix in the `percollator.se` code is based on the SPECweb96 size boundaries of up to

1 Kbyte, 1 Kbyte up to 10 Kbytes, 10 Kbytes up to 100 Kbytes, 100 Kbytes up to 1 Mbyte, and over 1 Mbyte. We end up with the percentage of ops and the average transfer time in each size range.

Observed Results in Practice

On a fairly quiet weekday, 280,000 accesses went via this cache, and 56 percent of the accesses went out to the Internet. The cache breakdown was as follows: 34 percent of the accesses hit in the cache, 16 percent missed and caused a cache write, 49 percent of the accesses were not cacheable, and 1 percent ended in some kind of error. A week's worth of data showed that the indirect and cache hit rates vary by 5 to 10 percent of the total from day to day.

The transfer time averaged about 2.3 seconds. The problem with this number is that it included a small number of very large or very slow transfers. The average for transfers up to 1 Kbyte was 1.8 seconds; for 1 to 10 Kbytes, it was 2.5 seconds; for 10 to 100 Kbytes, it was 6 seconds; for 100 Kbytes to 1 Mbyte, it was 40 seconds; and for over 1 Mbyte, it was 188 seconds. Within each of these size bands, connections to the client browser took place over everything from 100-Mbit Ethernet to 28.8-Kbit modems.

Transfer Time Versus Size Distribution

To calculate the transfer time versus size distribution, 10,000 access log entries were taken. After removing all zero content length and zero time transfers, size was plotted against transfer time using log-to-log axes. The result showed that transfer time is not very dependent on size until the size gets into the 10 to 100 Kbyte region, as shown in FIGURE 5-1.

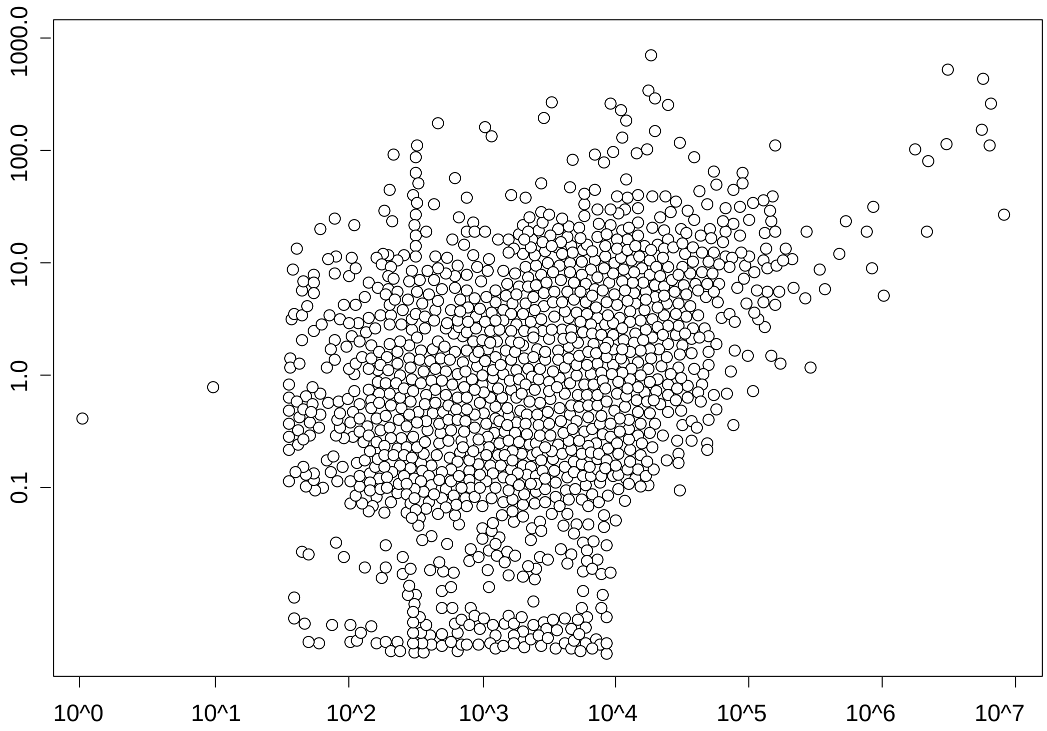

FIGURE 5-1 Log-to-Log Plot of Response Time Versus Size

The plot in FIGURE 5-1 shows bands of transfer times that depend upon the user's location. Many users are locally connected, but others are operating over slower networks. The transfer time includes the time spent waiting for the remote server to respond. Although it does not represent the extra time imposed by the cache, it actually gives a reasonable summary of the response time that users are experiencing with their browsers. This information is useful because you can collect it at a central point rather than attempting to measure the response time at an individual user site.

A probability density histogram of the same data is shown in FIGURE 5-2.

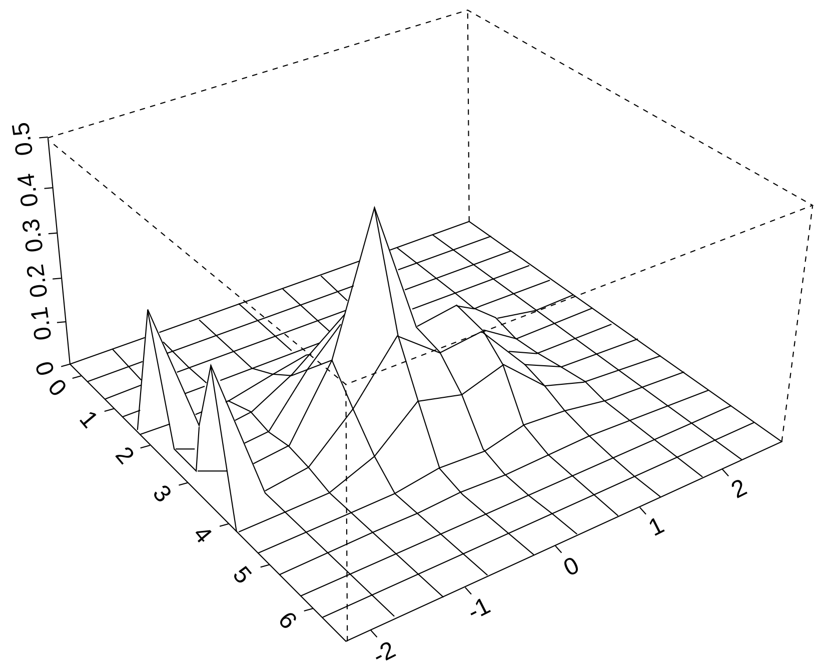

FIGURE 5-2 Perspective Log-Log Plot of Response Time Versus Size Distribution

The plot in FIGURE 5-2 shows that the most probable response is at about 0.5 seconds and 1 Kbyte. Remember that the average response time is 2.3 seconds on this data. It is clear that the average does not tell the whole story.

Internet Servers Summary

Log files are a rich source of performance information. Watching the rate at which they grow and breaking down the mix of logged operations is a powerful technique for getting good application-level performance measurements. You can apply this technique to many other applications, such as `ftp` and mail servers, as well as to any other application that can write a line to a file when it does some work.

Process Information

Many users are familiar with the ps process status command. But the ps command does not provide access to all the information available from the Solaris operating environment. It does provide a common set of data that is generally available on all UNIX systems, so we start by looking at that data, then move to the Solaris-specific data to gain a better insight into what is happening to a process.

The underlying data structures provided by the Solaris operating environment are described in full in the proc(4) manual page. The interface to /proc involves sending ioctl commands or opening special pseudo-files and reading them (a new feature of the Solaris 2.6 release). The data that ps uses is called PIOCPSINFO, and this is what you get back from ioctl. The data is slightly different if you read it from the pseudo-file.

```
proc(4)                  File Formats                  proc(4)

  PIOCPSINFO
     This returns miscellaneous process information such as that
     reported by ps(1). p is a pointer to a prpsinfo structure
     containing at least the following fields:

     typedef struct prpsinfo {
       char        pr_state;    /* numeric process state (see pr_sname) */
      char       pr_sname;   /* printable character representing pr_state
*/
       char        pr_zomb;     /* !=0: process terminated but not waited
for */
       char        pr_nice;     /* nice for cpu usage */
       u_long      pr_flag;     /* process flags */
       int         pr_wstat;    /* if zombie, the wait() status */
       uid_t       pr_uid;      /* real user id */
       uid_t       pr_euid;     /* effective user id */
       gid_t       pr_gid;      /* real group id */
       gid_t       pr_egid;     /* effective group id */
       pid_t       pr_pid;      /* process id */
       pid_t       pr_ppid;     /* process id of parent */
       pid_t       pr_pgrp;     /* pid of process group leader */
       pid_t       pr_sid;      /* session id */
       caddr_t     pr_addr;     /* physical address of process */
       long        pr_size;     /* size of process image in pages */
```

FIGURE 5-3 Process Information Used as Input by the ps Command

```
        long        pr_rssize;   /* resident set size in pages */
        u_long      pr_bysize;   /* size of process image in bytes */
        u_long      pr_byrssize; /* resident set size in bytes */
        caddr_t     pr_wchan;    /* wait addr for sleeping process */
        short      pr_syscall;  /* system call number (if in syscall) */
        id_t       pr_aslwpid;  /* lwp id of the aslwp; zero if no aslwp */
        timestruc_t pr_start;    /* process start time, sec+nsec since epoch
*/
        timestruc_t pr_time;     /* usr+sys cpu time for this process */
        timestruc_t pr_ctime;    /* usr+sys cpu time for reaped children */
        long       pr_pri;      /* priority, high value is high priority */
        char        pr_oldpri;   /* pre-SVR4, low value is high priority */
        char        pr_cpu;      /* pre-SVR4, cpu usage for scheduling */
        u_short     pr_pctcpu;   /* % of recent cpu time, one or all lwps */
        u_short     pr_pctmem;   /* % of system memory used by the process */
        dev_t      pr_ttydev;   /* controlling tty device (PRNODEV if none) */
        char        pr_clname[PRCLSZ]; /* scheduling class name */
        char       pr_fname[PRFNSZ]; /* last component of exec()ed pathname */
        char        pr_psargs[PRARGSZ];/* initial characters of arg list */
        int         pr_argc;     /* initial argument count */
        char        **pr_argv;   /* initial argument vector */
        char        **pr_envp;   /* initial environment vector */
       } prpsinfo_t;
```

FIGURE 5-3 Process Information Used as Input by the `ps` Command *(Continued)*

For a multithreaded process, you can get the data for each lightweight process separately. There's a lot more useful-looking information there, but no sign of the high-resolution microstate accounting that `/usr/proc/bin/ptime` and several SE Toolkit scripts display. They use a separate `ioctl`, `PIOCUSAGE`.

```
proc(4)                    File Formats                    proc(4)

  PIOCUSAGE
     When applied to the process file descriptor, PIOCUSAGE
     returns the process usage information; when applied to an
     lwp file descriptor, it returns usage information for the
     specific lwp.   p points to a prusage structure which is
     filled by the operation. The prusage structure contains at
     least the following fields:

     typedef struct prusage {
          id_t           pr_lwpid;    /* lwp id.  0: process or defunct */
          u_long         pr_count;    /* number of contributing lwps */
```

FIGURE 5-4 Additional Microstate-based Process Information

```
     timestruc_t    pr_tstamp;   /* current time stamp */
     timestruc_t    pr_create;   /* process/lwp creation time stamp */
     timestruc_t    pr_term;     /* process/lwp termination timestamp */
     timestruc_t    pr_rtime;    /* total lwp real (elapsed) time */
     timestruc_t    pr_utime;    /* user level CPU time */
     timestruc_t    pr_stime;    /* system call CPU time */
     timestruc_t    pr_ttime;    /* other system trap CPU time */
     timestruc_t    pr_tftime;   /* text page fault sleep time */
     timestruc_t    pr_dftime;   /* data page fault sleep time */
     timestruc_t    pr_kftime;   /* kernel page fault sleep time */
     timestruc_t    pr_ltime;    /* user lock wait sleep time *
     timestruc_t    pr_slptime;  /* all other sleep time */
     timestruc_t    pr_wtime;    /* wait-cpu (latency) time */
     timestruc_t    pr_stoptime; /* stopped time */
     u_long         pr_minf;     /* minor page faults */
     u_long         pr_majf;     /* major page faults */
     u_long         pr_nswap;    /* swaps */
     u_long         pr_inblk;    /* input blocks */
     u_long         pr_oublk;    /* output blocks */
     u_long         pr_msnd;     /* messages sent */
     u_long         pr_mrcv;     /* messages received */
     u_long         pr_sigs;     /* signals received */
     u_long         pr_vctx;     /* voluntary context switches */
     u_long         pr_ictx;     /* involuntary context switches */
     u_long         pr_sysc;     /* system calls */
     u_long         pr_ioch;     /* chars read and written */
} prusage_t;

PIOCUSAGE can be applied to a zombie   process   (see
PIOCPSINFO).

Applying PIOCUSAGE to a process that does not have micro-
state accounting enabled will enable microstate accounting
and return an estimate of times spent in the various states
up to this point.   Further invocations of PIOCUSAGE will
yield accurate microstate time accounting from this point.
To disable microstate accounting, use PIOCRESET with the
PR_MSACCT flag.
```

FIGURE 5-4 Additional Microstate-based Process Information *(Continued)*

There is a lot of useful data here. The time spent waiting for various events is a key measure. It is summarized by `msacct.se` in the following figure:

```
Elapsed time         3:20:50.049  Current time Fri Jul 26 12:49:28 1996
User CPU time           2:11.723  System call time       1:54.890
System trap time           0.006  Text pfault sleep         0.000
Data pfault sleep          0.023  Kernel pfault sleep       0.000
User lock sleep            0.000  Other sleep time    3:16:43.022
Wait for CPU time          0.382  Stopped time              0.000
```

FIGURE 5-5 Summary of Microstate Process Data Reported by `msacct.se`

Data Access Permissions

To access process data you must have access permissions for entries in `/proc` or run as a setuid root command. In the Solaris 2.5.1 release, using the `ioctl` access method for `/proc`, you can only access processes that you own, unless you log in as root, and the ps command is `setuid root`. In the Solaris 2.6 release, although you cannot access the `/proc/pid` entry for every process, you *can* read `/proc/pid/psinfo` and `/proc/pid/usage` for every process. This means that any user can use the full functionality of `ps` and the process. The code for `process_class.se` conditionally uses the new Solaris 2.6 access method and the slightly changed definition of the `psinfo` data structure.

Microstate Accounting

Microstate accounting is not turned on by default. It slows the system down slightly. It was on by default until the Solaris 2.3 release. From the Solaris 2.4 release on, it is enabled the first time you issue an `ioctl` to read the data. For the Solaris 2.6 access method, the process flags that enable microstate data collection and an inherit on fork option must be set directly via the `/proc/pid/ctl` interface.

The CPU time is normally measured by sampling 100 times a second, the state of all the CPUs from the clock interrupt. Microstate accounting works as follows: A high-resolution timestamp is taken on every state change, every system call, every page fault, and every scheduler change. Microstate accounting doesn’t miss anything, and the results are much more accurate than those from sampled measurements. The normal measures of CPU user and system time made by sampling can be wrong by 20 percent or more because the sample is biased, not random. Process scheduling uses the same clock interrupt used to measure CPU usage. This approach leads to systematic errors in the sampled data. The microstate-measured CPU usage data does not suffer from those errors.

For example, consider a performance monitor that wakes up every ten seconds, reads some data from the kernel, then prints the results and sleeps. On a fast system, the total CPU time consumed per wake-up might be a few milliseconds. On exit from the clock interrupt, the scheduler wakes up processes and kernel threads that have been sleeping until that time. Processes that sleep then consume less than their allotted CPU time quanta always run at the highest timeshare priority. On a lightly loaded system there is no queue for access to the CPU, so immediately after the clock interrupt, it is likely that the performance monitor will be scheduled. If it runs for less than 10 milliseconds, it will have completed and be sleeping again before the next clock interrupt. If you remember that the only way CPU time is allocated is based on what was running when the clock interrupt occurs, you can see that the performance monitor could be sneaking a bite of CPU time whenever the clock interrupt isn't looking. When there is a significant amount of queuing for CPU time, the performance monitor is delayed by a random amount of time, so sometimes the clock interrupt sees it, and the error level decreases.

The error is an artifact of the dual functions of the clock interrupt. If two independent, unsynchronized interrupts are used (one for scheduling and one for performance measurement), then the errors will be averaged away over time. Another approach to the problem is to sample more frequently by running the clock interrupt more often. This does not remove the bias, but it makes it harder to hide small bites of the CPU. The overhead of splitting up the interrupts is not worth implementing. You can increase the CPU clock rate to get more accurate measurements, but the overhead is higher than using direct microstate measurement, which is much more useful and accurate because it measures more interesting state transitions than just CPU activity.

The case where this problem matters most is when a sizing exercise occurs by measuring a lightly loaded system, then scaling the results up for a higher load estimate. Performance models that are calibrated at a light load also suffer from this problem. The best solution is to use a microstate accounting-based tool, or to disable some of the CPUs so that the measurements are made on a fairly busy system.

Some performance tool vendors are currently working to incorporate microstate-based data in their products. An example implementation of a microstate-based monitor was built using the SE Toolkit. The data provided by the SE process monitoring class is shown in FIGURE 5-6.

```
    /* output data for specified process */
    double interval;       /* measured time interval for averages
*/
    double timestamp;       /* last time process was measured */
    double creation;        /* process start time */
    double termination;     /* process termination time stamp */
```

FIGURE 5-6 Combined Process Information from SE Toolkit Process Class

```
    double elapsed;        /* elapsed time for all lwps in process
*/
    double total_user;      /* current totals in seconds */
    double total_system;
    double total_child;     /* child processes that have exited */
    double user_time;       /* user time in this interval */
    double system_time;     /* system call time in this interval */
    double trap_time;       /* system trap time in interval */
    double child_time;      /* child CPU in this interval */
    double text_pf_time;    /* text page fault wait in interval */
    double data_pf_time;    /* data page fault wait in interval */
    double kernel_pf_time;  /* kernel page fault wait in interval
*/
    double user_lock_time;  /* user lock wait in interval */
    double sleep_time;      /* all other sleep time */
    double cpu_wait_time;   /* time on runqueue waiting for CPU */
    double stoptime;        /* time stopped from ^Z */
    ulong syscalls;        /* syscall/interval for this process */
    ulong inblocks;         /* input blocks/interval - metadata
only - not interesting */
    ulong outblocks;        /* output blocks/interval - metadata
only - not interesting */
    ulong vmem_size;        /* size in KB */
    ulong rmem_size;        /* RSS in KB */
    ulong maj_faults;       /* majf/interval */
    ulong min_faults;      /* minf/interval - always zero - bug? */
    ulong total_swaps;      /* swapout count */
    long  priority;         /* current sched priority */
    long  niceness;         /* current nice value */
    char  sched_class[PRCLSZ];/* name of class */
    ulong messages;         /* msgin+msgout/interval */
    ulong signals;          /* signals/interval */
    ulong vcontexts;       /* voluntary context switches/interval
*/
    ulong icontexts;      /* involuntary context switches/interval
*/
    ulong charios;          /* characters in and out/interval */
    ulong lwp_count;        /* number of lwps for the process */
    int   uid;              /* current uid */
    long  ppid;             /* parent pid */
    char  fname[PRFNSZ];    /* last component of exec'd pathname */
    char  args[PRARGSZ];   /* initial part of command name and arg
list */
```

FIGURE 5-6 Combined Process Information from SE Toolkit Process Class *(Continued)*

Most of the data in FIGURE 5-6 is self explanatory. All times are in seconds in double precision with microsecond accuracy. The minor fault counter seems to be broken because it always reports zero. The `inblock` and `outblock` counters are not interesting because they only refer to file system metadata for the old-style buffer cache. The `chario`s counter includes all read and write data for all file descriptors, so you can see the file I/O rate. The `lwp_count` is not the current number of lwps. It is a count of how many `lwps` the process has ever had. If the count is more than one, then the process is multithreaded. It is possible to access each `lwp` in turn and read its `psinfo` and `usage` data. The process data is the sum of these.

Child data is accumulated when a child process exits. The CPU used by the child is added into the data for the parent. This can be used to find processes that are forking many short-lived commands.

Accounting

Who ran what, when, and how much resource was used?

Many processes have very short life spans. You cannot see such processes with `ps`, but they may be so frequent that they dominate the load on your system. The only way to catch them is to have the system keep a record of every process that has run, who ran it, what it was, when it started and ended, and how much resource it used. The answers come from the system accounting subsystem. While you may have some concerns about accounting because of the “big brother is watching you” connotation or the cost of additional overhead, the information is important and valuable. The overhead of collecting accounting data is *always present but is insignificant*. When you turn on accounting, you are just enabling the storage of a few bytes of useful data when a process exits.

Accounting data is most useful when it is measured over a long period of time. This temporal information is useful on a network of workstations as well as on a single, time-shared server. From this information, you can identify how often programs run, how much CPU time, I/O, and memory each program uses, and what work patterns throughout the week look like. To enable accounting to start immediately, enter the three commands shown below.

```
# ln /etc/init.d/acct /etc/rc0.d/K22acct
# ln /etc/init.d/acct /etc/rc2.d/S22acct
# /etc/init.d/acct start
Starting process accounting
```

Refer to the accounting section in the *Solaris System Administration Answerbook* and see the `acctcom` command. Add some `crontab` entries to summarize and checkpoint the accounting logs. Collecting and checkpointing the accounting data itself puts a negligible additional load onto the system, but the summary scripts that run once a day or once a week can have a noticeable effect, so schedule them to run outside business hours.

Your `crontab` file for the `adm` user should contain the following:

```
# crontab -l adm
#ident  “@(#)adm       1.5     92/07/14 SMI”   /* SVr4.0 1.2   */
#min    hour    day     month   weekday
0       *       *       *       *       /usr/lib/acct/ckpacct
30      2       *       *       *       /usr/lib/acct/runacct 2> /var/adm/
acct/nite/fd2log
30      9       *       *       5       /usr/lib/acct/monacct
```

You get a daily accounting summary, but it is best to keep track of the monthly summary stored in `/var/adm/acct/fiscal`. Following is an excerpt from `fiscrpt07`, which is the report for July on this desktop system.

```
Jul 26 09:30 1996                      TOTAL COMMAND SUMMARY FOR FISCAL 07 Page 1

                TOTAL COMMAND SUMMARY
 COMMAND  NUMBER       TOTAL    TOTAL       TOTAL      MEAN     MEAN     HOG          CHARS  BLOCKS
    NAME    CMDS    KCOREMIN  CPU-MIN    REAL-MIN    SIZE-K  CPU-MIN  FACTOR         TRNSFD    READ

  TOTALS   26488 16062007.75  3960.11   494612.41   4055.95     0.15    0.01    17427899648   39944

mae           36  7142887.25  1501.73     2128.50   4756.45    41.71    0.71     2059814144    1653
sundgado      16  3668645.19   964.83     1074.34   3802.36    60.30    0.90      139549181      76
Xsun          29  1342108.55   251.32     9991.62   5340.18     8.67    0.03     2784769024    1295
xlock         32  1027099.38   726.87     4253.34   1413.04    22.71    0.17     4009349888      15
fountain       2   803036.25   165.11      333.65   4863.71    82.55    0.49         378388       1
netscape      22   489512.97    72.39     3647.61   6762.19     3.29    0.02      887353080    2649
maker4X.      10   426182.31    43.77     5004.30   9736.27     4.38    0.01      803267592    3434
wabiprog      53   355574.99    44.32      972.44   8022.87     0.84    0.05      355871360     570
imagetoo      21   257617.08    15.65      688.46  16456.60     0.75    0.02       64291840     387
java         235   203963.64    37.96      346.35   5373.76     0.16    0.11      155950720     240
aviator        2   101012.82    22.93       29.26   4406.20    11.46    0.78        2335744      40
se.sparc      18    46793.09    19.30     6535.43   2424.47     1.07    0.00      631756294      20
xv             3    40930.98     5.58       46.37   7337.93     1.86    0.12      109690880      28
```

The commands reported are sorted by KCOREMIN, which is the product of the amount of CPU time used and the amount of RAM used while the command was active. CPU-MIN is the number of minutes of CPU time. REAL_MIN is the elapsed time for the commands. SIZE-K is an average value for the RSS over the active lifetime of the process. It does not include times when the process was not actually running. (In Solaris 2.4 and earlier releases, a bug causes this measure to be invalid.) HOG FACTOR is the ratio of CPU-MIN to REAL-MIN; a high factor means that this command hogs the CPU whenever it is running. CHARS TRNSFD counts the number of characters read and written. BLOCKS READ counts data read from block devices (basically, local disk file system reads and writes). The underlying data that is collected can be seen in the acct(4) manual page. The data structure is very compact—about 40 bytes, as shown FIGURE 5-7:

```
DESCRIPTION
     Files produced as a result of calling acct(2) have records
     in the form defined by <sys/acct.h>, whose contents are:

    typedef ushort  comp_t;     /* pseudo "floating point" representation */
                                   /* 3-bit base-8 exponent in the high */
                                   /* order bits, and a 13-bit fraction */
                                   /* in the low order bits. */

     struct  acct
     {
             char    ac_flag;      /* Accounting flag */
             char    ac_stat;      /* Exit status */
             uid_t   ac_uid;       /* Accounting user ID */
             gid_t   ac_gid;       /* Accounting group ID */
             dev_t   ac_tty;       /* control tty */
             time_t  ac_btime;     /* Beginning time */
             comp_t  ac_utime;     /* accounting user time in clock */
                                   /* ticks */
             comp_t  ac_stime;     /* accounting system time in clock */
                                   /* ticks */
           comp_t  ac_etime;    /* accounting total elapsed time in clock */
                                   /* ticks */
             comp_t  ac_mem;       /* memory usage in clicks (pages) */
             comp_t  ac_io;        /* chars transferred by read/write */
             comp_t  ac_rw;        /* number of block reads/writes */
             char    ac_comm[8];   /* command name */
     };
```

FIGURE 5-7 Accounting Data Format

Solaris Resource Manager Accounting

The SRM software provides the ability to generate accounting data based on resource usage. This is made available by the accumulation of resource information in the lnode tree. Although the SRM software does not provide resource accounting, it does provide the data and data-extraction tools used to develop a system to generate accounting (or billing) information. See Chapter 7 for more information on SRM Accounting.

Network Accounting Using NetFlow

The Solaris Bandwidth Manager software has built-in support for Cisco NetFlow™ software. This feature allows for detailed network measurements that can be sent to other software packages that can process and analyze this data.

Besides the Solaris Bandwidth Manager software, NetFlow data collection is supported on a variety of Cisco devices, such as the Cisco 7000 and 7500 series routers.

When NetFlow is enabled on the Solaris Bandwidth Manager software or any Cisco equipment supporting NetFlow, network packets are processed and classified into flows. A flow is a unidirectional series of packets that have several parameters in common:

- Source and destination IP address
- Source and destination application port number
- IP protocol type
- IP ToS (Type of Service)

A flow ends when either a FIN or RST packet is received for that flow, or when it times out.

NetFlow-enabled devices send out NetFlow datagrams, which contain records for one or more flows. Combining multiple flow records in one datagram reduces network overhead caused by NetFlow. These datagrams are UDP based, so they can get lost in transit in case of a busy network. NetFlow version 5 and higher add a sequence number to the datagram header, so the receiver of these datagrams can at least know that a datagram was lost.

Each NetFlow datagram record contains detailed flow information, such as the number of bytes in the flow, the number of packets, the duration, source and destination IP address, and so on.

Cisco offers applications that read and process NetFlow datagrams. NetFlow FlowCollector™ is a NetFlow datagram consumer for one or more NetFlow devices. These devices simply point to the host and port number on which the FlowCollector software is running. The FlowCollector aggregates this data, does preprocessing and filtering, and provides several options to save this data to disk (such as flat files). Other applications, such as network analyzing, planning, and billing, can use these files as input.

The NetFlow FlowAnalyzer™ application uses the output from NetFlow FlowCollector. It provides elaborate processing, graphing, and reporting options for network analysis, planning, troubleshooting, and more.

You can run the Solaris Bandwidth Manager software in statistics mode, where it will process packets but not provide the priority scheduling. By using the NetFlow output of the Solaris Bandwidth Manager software, you can obtain detailed network measurements that are not easily measured otherwise. For example, you can find out the bandwidth used by a specific application or user by using NetFlow data analysis. The flow data can also provide latency information of specific applications. Trend data can show the long term impact of the deployment of certain applications.

For more information about Cisco NetFlow, see `http://www.cisco.com/warp/public/732/netflow/index.html`.

Storage Measurements

Individual disks are well instrumented, but storage subsystems are now much more complex than a few individual disks. This section explains the basic Solaris software measures and discusses more complex disk subsystems.

Disk Workloads

There are six basic disk access patterns. Read, write, and update operations can either be sequentially or randomly distributed. Sequential read and write occur when files are copied and created or when large amounts of data are processed. Random read and write can occur in indexed database reads or can be due to page-in or page-out to a file. Update consists of a read-modify-write sequence and can be caused by a database system committing a sequence of transactions in either a sequential or random pattern. When you are working to understand or improve the performance of your disk subsystem, spend some time identifying which of the access-pattern categories are most important to you.

You cannot automatically tell which processes are causing disk activity. The kernel does not collect this information. You may be able to work out where the workload comes from by looking at how an application was installed, but often you must resort to using `truss` on a process or the TNF tracing system. The Solaris software does not offer a way of getting I/O activity on a per-file descriptor basis, so application-specific instrumentation is all we have. Databases such as Oracle can collect and report data on a per-tablespace basis, so if you can map the tablespaces to physical disks, you can tell what is going on programmatically. Such collection and analysis is performed by the BMC Best/1 performance modeling tool so that changes in disk workload can be modeled.

Output Formats and Options for iostat

The iostat command produces output in many forms. When a large number of disks are being reported, the iostat -x variant provides extended statistics and is easier to read because each disk is summarized on a separate line (see FIGURE 5-8). The following values are reported: the number of transfers and kilobytes per second with read and write shown separately, the average number of commands waiting in the queue, the average number of commands actively being processed by the drive, the I/O service time, and the percentages of the time that commands were waiting in the queue, and commands that were active on the drive.

```
% iostat -txc 5
                             extended disk statistics      tty        cpu
disk   r/s  w/s   Kr/s   Kw/s wait actv  svc_t  %w  %b  tin tout us sy wt id
fd0    0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0    0   77 42  9  9 39
sd0    0.0  3.5    0.0   21.2  0.0  0.1   41.6   0  14
sd1    0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd3    0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0
                             extended disk statistics      tty        cpu
disk   r/s  w/s   Kr/s   Kw/s wait actv  svc_t  %w  %b  tin tout us sy wt id
fd0    0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0    0   84 37 17 45  1
sd0    0.0 16.8    0.0  102.4  0.0  0.7   43.1   2  61
sd1    0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd3    0.0  1.0    0.0    8.0  0.0  0.1  114.3   2   4
```

FIGURE 5-8 iostat -x Output

Disk configurations have become extremely large and complex on big server systems. The Starfire system supports a maximum configuration of several thousand disk drives, but dealing with even a few hundred is a problem. When large numbers of disks are configured, the overall failure rate also increases. It is hard to keep an inventory of all the disks, and tools like the SyMON software have to depend upon parsing messages from syslog to see if any faults are reported. The size of each disk is also growing. When more than one type of data is stored on a disk, it's hard to determine which disk partition is active. A series of new features has been introduced in the Solaris 2.6 release to help solve these problems.

- Per-partition data identical to existing per-disk data

  It is now possible to separate root, swap, and home directory activity even if these file systems are on the same disk.

- New "error and identity" data per disk

  You no longer need to scan syslog for errors. Full data is saved from the first SCSI probe to a disk. This data includes vendor, product, revision, serial number, RPM, heads, and size. Soft, hard, and transport error counter categories sum up any problems. The detail option adds the Media Error, Device not ready, No device, Recoverable, Illegal request, and Predictive failure analysis information. Dead or missing disks can still be identified because there is no need to send them another SCSI probe.

- New `iostat` options to present these metrics

  One option (`iostat -M`) shows throughput in Mbytes/s rather than Kbytes/s for high-performance systems. Another option (`-n`) translates disk names into a more useful form, so you don't have to deal with the sd43b format—instead, you get c1t2d5s1. This feature makes it easier to keep track of per-controller load levels in large configurations.

Fast tapes now match the performance impact of disks. We recently ran a tape backup benchmark to see if there were any scalability or throughput limits in the Solaris software, and we were pleased to find that the only real limit is the speed of the tape drives. The final result was a backup rate of an Oracle database at 1 terabyte/hour or about 350 Mbytes/s, which was as fast as the disk subsystem we had configured could perform. To sustain this rate, we used every accessible tape drive, including 24 StorageTEK Redwood tape transports, which run at around 15 Mbytes/s each. We ran this test using the Solaris 2.5.1 operating environment, but there are no measurements of tape drive throughput in the Solaris 2.5.1 release. Tape metrics were added to the Solaris 2.6 release, and you can see the tape drive that is active, the throughput, average transfer size, and service time for each tape drive.

Tapes are instrumented in the same way as disks; they appear in `sar` and `iostat` automatically. Tape read/write operations are instrumented with all the same measures that are used for disks. Rewind and scan/seek are omitted from the service time.

The output format and options of `sar`(1) are fixed by the generic UNIX standard SVID3, but the format and options for `iostat` can be changed. In the Solaris 2.6 release, existing `iostat` options are unchanged. Apart from extra entries that appear for tape drives and NFS mount points, anyone storing `iostat` data from a mixture of Solaris 2.x systems will get a consistent format. New options that extend `iostat` are as follows:

`-E`	full error statistics
`-e`	error summary statistics
`-n`	disk name and NFS mount point translation, extended service time
`-M`	Mbytes/s instead of Kbytes/s
`-P`	partitions only
`-p`	disks and partitions

Here are examples of some of the new `iostat` formats:

```
% iostat -xp
                              extended device statistics
device    r/s  w/s   kr/s   kw/s wait actv  svc_t  %w  %b
sd106     0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd106,a   0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd106,b   0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd106,c   0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0
st47      0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0
```

```
% iostat -xe
                             extended device statistics ---- errors ----
device    r/s  w/s   kr/s   kw/s wait actv  svc_t  %w  %b s/w h/w trn tot
sd106     0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0   0   0   0   0
st47      0.0  0.0    0.0    0.0  0.0  0.0    0.0   0   0   0   0   0   0
```

```
% iostat -E

sd106   Soft Errors: 0 Hard Errors: 0 Transport Errors: 0
Vendor: SEAGATE  Product: ST15230W SUN4.2G Revision: 0626 Serial No:
00193749
RPM: 7200 Heads: 16 Size: 4.29GB <4292075520 bytes>
Media Error: 0 Device Not Ready: 0 No Device: 0 Recoverable: 0
Illegal Request: 0 Predictive Failure Analysis: 0

st47    Soft Errors: 0 Hard Errors: 0 Transport Errors: 0
Vendor: EXABYTE  Product: EXB-8505SMBANSH2 Revision: 0793 Serial No:
```

The Solaris 2.5 Trace Capability

UNIX systems have had a kernel trace capability for many years. It was designed for development and debugging, not for end users. The production kernel is normally built without the trace capability for performance reasons. One of the first production kernels to include tracing was IBM's AIX kernel on the RS/6000 servers. They left it on during early releases to assist in debugging, then decided that tracing was useful enough to pay its way, and the overhead was low. So it is now a permanent feature. Sun also recognized the value of trace information but decided to extend the trace capability to make it more generally useful and to implement it alongside the existing kernel trace system. It was introduced in the Solaris 2.5 operating environment and consists of the following features.

- A self-describing trace output format, called Trace Normal Form (TNF), allows data structures to be embedded in the trace file without the need for an external definition of their types and contents.
- A set of libraries allows user-level programs to generate trace data. In particular, this trace data helps analyze and debug multithreaded applications.

- A well-defined set of kernel probe points covering most important events was implemented.
- A program prex(1) controls probe execution for both user and kernel traces.
- A program tnfxtract(1) reads out the kernel trace buffer, and tnfdump(1) displays TNF data in human-readable ASCII.
- Manual pages exist for all the commands and library calls. The set of implemented kernel probes is documented in tnf_probes(4).

A few things about kernel probes are inconvenient. While user-level probes can write to trace files, the kernel probes write to a ring buffer. This buffer is not a single global ring; it is a buffer per kernel thread. This buffer scheme avoids any need to lock the data structures, so there is no performance loss or contention on multiprocessor systems. You cannot easily tell how big the buffer needs to be. One highly active probe point might loop right round its buffer while others have hardly started. If you are trying to capture every single probe, make the buffer as big as you can. In general, it is best to work with low event rate probes or rely on sampling and put up with missing probes. The tnfxtract routine just takes a snapshot of the buffer, so a second snapshot will include anything left over from the first one. The tnfdump program does quite a lot of work to sort the probe events into time order.

I/O Trace: Commands and Features

The command sequence to initiate an I/O trace is quite simple. You run the commands as root, and you need a directory to hold the output. It's easiest to have two windows open: one to run prex, and the other to go through a cycle of extracting, dumping, and viewing the data as required. The command sequence for prex is to first allocate a buffer (the default is 384 Kbytes; you can make it bigger), enable the io group of probes, make them trace accesses, then turn on the global flag that enables all kernel tracing.

```
# prex -k
Type "help" for help ...
prex> buffer alloc
Buffer of size 393216 bytes allocated
prex> enable io
prex> trace io
prex> ktrace on
```

Now, wait a while or run the program you want to trace. In this case, we ran iostat -x 10 in another window, didn't try to cause any activity, and waited for some slow service time to appear. After approximately one minute, we stopped collecting.

```
prex> ktrace off
```

In the other window we extracted and dumped the data to take a look at it.

```
# mkdir /tmp/tnf
# cd /tmp/tnf
# tnfxtract io.tnf
# tnfdump io.tnf | more
```

Understanding I/O Measurements

To really understand the data presented by `iostat`, `sar`, and other tools, you must look at the raw data being collected by the kernel, remember some history, and do some simple mathematics.

In the old days, disk controllers controlled the disks directly. All the intelligence was in the controller. The disk heads were directly connected to the controller, and the device driver knew exactly which track the disk was reading. As each bit was read from disk, it was buffered in the controller until a whole disk block was ready to be passed to the device driver.

The device driver maintained a queue of waiting requests, which were serviced one at a time by the disk as shown in FIGURE 5-9. From this, the system could report the service time directly as milliseconds-per-seek. The throughput in transfers per second was also reported, as was the percentage of the time that the disk was busy (the utilization). The terms *utilization*, *service time*, *wait time*, *throughput*, and *wait queue length* have well-defined meanings in this scenario because the setup is so simple that a very basic queueing model fits it well. A set of simple equations from queuing theory can be used to derive these values from underlying measurements.

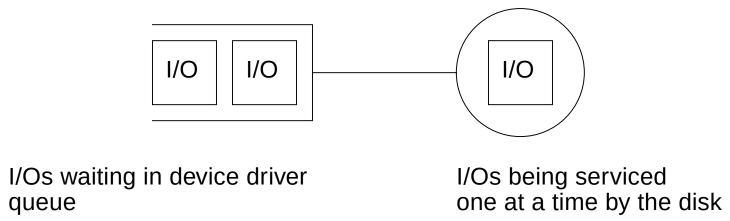
I/O
I/O
I/O
I/Os waiting in device driver queue
I/Os being serviced one at a time by the disk

FIGURE 5-9 Simple Old Disk Model

Nowadays, a standard disk is SCSI based and has an embedded controller. The disk drive contains a small microprocessor and about 1 Mbyte of RAM. It can typically handle up to 64 outstanding requests via SCSI tagged-command queuing. The system uses a SCSI Host Bus adapter (HBA) to talk to the disk. In large systems, there is another level of intelligence and buffering in a hardware RAID controller. The simple model of a disk used by `iostat` and its terminology have become confused. In addition, the same reporting mechanism is used for client side NFS mount points and complex disk volumes setup using Solstice™ DiskSuite™ software.

In the old days, if the device driver sent a request to the disk, the disk would do nothing else until it completed the request. The time it took was the service time, and the average service time was a property of the disk itself. Disks that spin faster and seek faster have lower (better) service times. With today's systems, if the device driver issues a request, that request is queued internally by the RAID controller and the disk drive, and several more requests can be sent before the first one comes back. The service time, as measured by the device driver, varies according to the load level and queue length and is not directly comparable to the old-style service time of a simple disk drive.

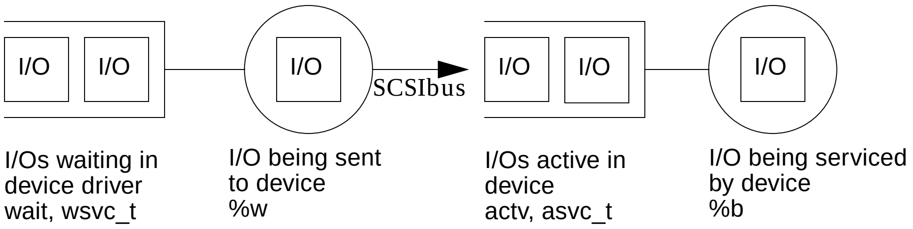
I/O
I/O
I/O
SCSIbus
I/O
I/O
I/O
I/Os waiting in device driver wait, wsvc_t
I/O being sent to device %w
I/Os active in device actv, asvc_t
I/O being serviced by device %b

FIGURE 5-10 Two-Stage Disk Model Used By Solaris 2

The instrumentation provided in the Solaris operating environment takes this change into account by explicitly measuring a two-stage queue: one queue, called the wait queue, in the device driver; and another queue, called the active queue, in the device itself. A read or write command is issued to the device driver and sits in the wait queue until the SCSI bus and disk are both ready. When the command is sent to the disk device, it moves to the active queue until the disk sends its response. The problem with `iostat` is that it tries to report the new measurements in some of the original terminology. The "wait service time" is actually the time spent in the "wait" queue. This is not the right definition of service time in any case, and the word "wait" is used to mean two different things. To sort out what we really have, we must do some mathematics.

Let's start with the actual measurements made by the kernel. For each disk drive (and each disk partition, tape drive, and NFS mount in the Solaris 2.6 release), a small set of counters is updated. An annotated copy of the `kstat`(3K)-based data structure that the SE Toolkit uses is shown in FIGURE 5-11.

```
struct ks_disks {
    long      number$;    /* linear disk number */
    string    name$;      /* name of the device */

    ulonglong nread;      /* number of bytes read */
    ulonglong nwritten;   /* number of bytes written */
    ulong     reads;      /* number of reads */
    ulong     writes;     /* number of writes */
    longlong  wtime;      /* wait queue - time spent waiting */
    longlong  wlentime;   /* wait queue - sum of queue length multiplied
by time at that length */
    longlong  wlastupdate;/* wait queue - time of last update */
   longlong  rtime;     /* active/run queue - time spent active/running */
   longlong  rlentime;   /* active/run queue - sum of queue length * time
at that length */
    longlong  rlastupdate;/* active/run queue - time of last update */
    ulong     wcnt;       /* wait queue - current queue length */
    ulong     rcnt;       /* active/run queue - current queue length */
};
```

FIGURE 5-11 Kernel Disk Information Statistics Data Structure

None of these values are printed out directly by `iostat`, so the basic arithmetic starts here. The first thing to note is that the underlying metrics are cumulative counters or instantaneous values. The values printed by `iostat` are averages over a time interval. We need to take two copies of the above data structure together with a high resolution *timestamp* for each and do some subtraction. We then get the average values between the *start* and *end* times. We'll write it out as plainly as possible, with pseudocode that assumes an array of two values for each measure, indexed by *start* and *end*. T_{hires} is in units of nanoseconds, so we divide to get seconds as *T*.

$$T_{hires} = \text{hires elapsed time} = EndTime - StartTime = timestamp[end] - timestamp[start]$$

$$T = \frac{T_{hires}}{100000000}$$

$$B_{wait} = \text{hires busy time for wait queue} = wtime[end] - wtime[start]$$

$$B_{run} = \text{hires busy time for run queue} = rtime[end] - rtime[start]$$

$$QB_{wait} = \text{wait queue length * time} = wlentime[end] - wlentime[start]$$

$$QB_{run} = \text{run queue length * time} = rlentime[end] - rlentime[start]$$

We assume that all disk commands complete fairly quickly, so the arrival and completion rates are the same in a steady state average, and the throughput of both queues is the same. We'll use completions below because they seem more intuitive in this case.

$$C_{read} = \text{completed reads} = reads[end] - reads[start]$$

$$X_{read} = \text{read throughput} = \text{iostat rps} = \frac{C_{read}}{T}$$

$$C_{write} = \text{completed writes} = writes[end] - writes[start]$$

$$X_{write} = \text{write throughput} = \text{iostat wps} = \frac{C_{write}}{T}$$

$$C = \text{total commands completed} = C_{read} + C_{write}$$

$$X = \text{throughput in commands per second} = \text{iostat tps} = \frac{C}{T} = X_{read} + X_{write}$$

A similar calculation gets us the data rate in kilobytes per second.

$$K_{read} = \text{Kbytes read in the interval} = \frac{nread[end] - nread[start]}{1024}$$

$$X_{kread} = \text{read Kbytes throughput} = \text{iostat Kr/s} = \frac{K_{read}}{T}$$

$$K_{write} = \text{Kbytes written in the interval} = \frac{nwritten[end] - nwritten[start]}{1024}$$

$$X_{kwrite} = \text{write Kbytes throughput} = \text{iostat Kw/s} = \frac{K_{write}}{T}$$

$$X_k = \text{total data rate in Kbytes per second} = \text{iostat Kps} = X_{kread} + X_{kwrite}$$

Next, we obtain the utilization or the busy time as a percentage of the total time:

$$U_{wait} = \text{wait queue utilization} = \text{iostat \%w} = \frac{100 \times B_{wait}}{T_{hires}}$$

$$U_{run} = \text{run queue utilization} = \text{iostat \%b} = \frac{100 \times B_{run}}{T_{hires}}$$

Now, we get to service time, but it is *not* what `iostat` prints out and calls service time. This is the real thing!

$$S_{wait} = \text{average wait queue service time in milliseconds} = \frac{B_{wait}}{C \times 100000}$$

$$S_{run} = \text{average run queue service time in milliseconds} = \frac{B_{run}}{C \times 100000}$$

The meaning of S_{run} is as close as you can get to the old-style disk service time. Remember that the disk can run more than one command at a time and can return those commands in a different order than they were issued.

The data structure contains an instantaneous measure of queue length, but we want the average over the time interval. We get this from that strange "length time" product by dividing it by the busy time:

$$Q_{wait} = \text{average wait queue length} = \text{iostat wait} = \frac{QB_{wait}}{B_{wait}}$$

$$Q_{run} = \text{average run queue length} = \text{iostat actv} = \frac{QB_{run}}{B_{run}}$$

Finally, we get the number that `iostat` calls service time. It is defined as the queue length divided by the throughput, but it is actually the residence or response time and includes all queuing effects:

$$R_{wait} = \text{average wait queue response time} = \text{iostat wsvc\_t} = \frac{Q_{wait}}{X}$$

$$R_{run} = \text{average run queue response time} = \text{iostat asvc\_t} = \frac{Q_{run}}{X}$$

Another way to express response time is in terms of service time and utilization:

$$R_{wait} = \text{average wait queue response time prediction} = \text{iostat wsvc\_t} = \frac{S_{wait}}{1 - B_{wait}}$$

$$R_{run} = \text{average run queue response time prediction} = \text{iostat asvc\_t} = \frac{S_{run}}{1 - B_{run}}$$

This uses a theoretical model of response time that assumes that as you approach 100 percent utilization with a constant service time the response time increases to infinity.

The real definition of service time is the time taken for the first command in line to be processed. Its value is not printed out by `iostat`. Using the SE Toolkit, this deficiency is easily fixed. A corrected version of `iostat` written in SE prints out the data, using the format shown in FIGURE 5-12.

```
% se siostat.se 10
03:42:50  ------throughput------ -----wait queue----- ----active queue----
disk      r/s  w/s   Kr/s   Kw/s qlen res_t svc_t %ut qlen res_t svc_t %ut
c0t2d0s0  0.0  0.2    0.0    1.2 0.00  0.02  0.02   0 0.00 22.87 22.87   0
03:43:00  ------throughput------ -----wait queue----- ----active queue----
disk      r/s  w/s   Kr/s   Kw/s qlen res_t svc_t %ut qlen res_t svc_t %ut
c0t2d0s0  0.0  3.2    0.0   23.1 0.00  0.01  0.01   0 0.72 225.45 16.20   5
```

FIGURE 5-12 SE-based Rewrite of `iostat` to Show Service Time Correctly

The Solaris 2.6 disk instrumentation is complete and accurate. Now that it has been extended to tapes, partitions, and client NFS mount points, there much more can be done with it.

Understanding Resource Utilization Characteristics

One important characteristic of complex I/O subsystems is that the utilization measure can be confusing. When a simple system reaches 100 percent busy, it has also reached its maximum throughput. This is because only one thing is being processed at a time in the I/O device. When the device being monitored is an NFS server, hardware RAID disk subsystem, or a striped volume, it is clearly a much more complex situation. All of these can process many requests in parallel.

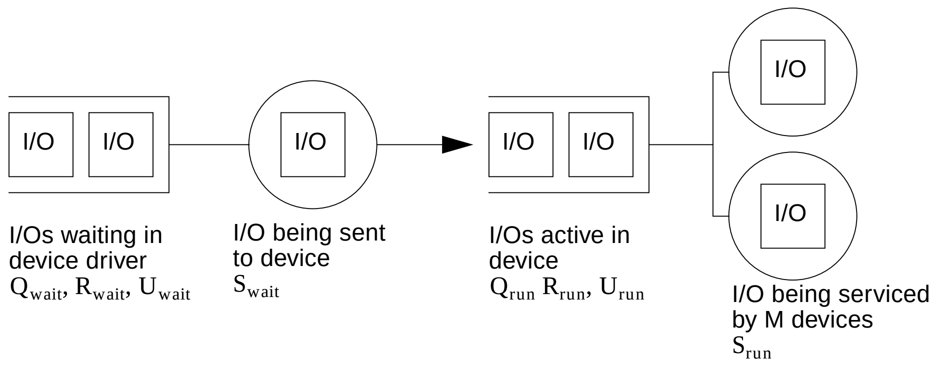
I/O
I/O
I/O
I/O
I/O
I/O
I/O
I/Os waiting in device driver Q_{wait}, R_{wait}, U_{wait}
I/O being sent to device S_{wait}
I/Os active in device Q_{run} R_{run}, U_{run}
I/O being serviced by M devices S_{run}

FIGURE 5-13 Complex I/O Device Queue Model

As long as a single I/O is being serviced at all times, the utilization is reported as 100 percent. This makes sense because it means that the pool of devices is always busy doing something. However, there is enough capacity for additional I/Os to be serviced in parallel. Compared to a simple device, the service time for each I/O is the same, but the queue is being drained more quickly so the average queue length and response time is less, and the peak throughput is more. In effect, the load on each disk is divided by the number of disks, so the underlying utilization is U/M. The approximated model for response time in this case changes so that response time stays lower for longer, but it still heads for infinity at a true 100 percent utilization.

$$R_{wait} = \text{average wait queue response time prediction} = \text{iostat wsvc\_t} = \frac{S_{wait}}{1-(B_{wait}/M)^M}$$

$$R_{run} = \text{average run queue response time prediction} = \text{iostat asvc\_t} = \frac{S_{run}}{1-(B_{run}/M)^M}$$

In practice, some other effects come into play. The drives optimize head movement, so that as the queue gets longer, the average service time decreases, since it is more likely that there is a block nearby to make a shorter seek to. This in itself increases throughput more than traditional simple queueing models would predict.

Network Measurements

The way network measurements can and should be made depends on the objective of the measurement. Do you want to manage the overall health of a network link? Are you interested in the TCP/IP statistics of your server? Do you want to know how much bandwidth is used by a specific application? Each of these questions can be answered if the right tools and techniques are used to do the measuring.

Network Throughput

Using SNMP counters is a good way to get an overall view of network throughput. The Solaris software provides an SNMP daemon which provides the data you need. If your host is managed by the SyMON software, you can browse the SNMP counters through the SyMON GUI, and it is very easy to graph network throughput data by combining some of these counters in a simple formula. For example, to get a graph depicting how much percent of the (half duplex) network segment that is busy, use the following formula:

```
((ifInOctets + ifOutOctets) / time_interval)/ ifSpeed) * 100
```

Most networking devices support SNMP, and the SyMON software can incorporate any third-party MIBs so all links from a switch can be monitored at the same time.

Besides the SyMON software, many other commercial and free applications and utilities manage SNMP devices. One example of an excellent free utility is MRTG and can be downloaded from `http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html`.

Ethernet Statistics

Traditionally, administrators have used collisions as an indicator of network health for Ethernet networks. Collisions are reported using `netstat` with the `-i` flag.

```
doghouse% netstat -i

Name Mtu   Net/Dest  Address    Ipkts   Ierrs Opkts    Oerrs  Collis  Queue
lo0  8232  loopback  localhost  40986       0 40986        0  0       0
hme0 1500  doghouse  doghouse   128128305   0 125004514    0  77754   0
```

The collision rate is `(Collis / Opkts) * 100%`. In our case, that is less than a tenth of a percent. Collisions are absolutely normal and should cause no concern unless collision rates become very high (in the order of 10 to 20 percent or higher). If you have high collision rates, check the total bandwidth utilization on your network link.

Today, more and more switched and full duplex Ethernet networks are being deployed. In these environments, there are no collisions since each port on the switch has its own dedicated bandwidth, and hosts can send network packets whenever they want. If a switch gets overloaded, it just drops packets that it can't process fast enough. Dropped packets have to be retransmitted by the sender.

Netstat has an undocumented option that is invoked with the -k flag. It will dump a lot of statistics that are kept in kernel structures (the `kstats`). You can look at for example the statistics from the `hme` card as follows:

```
doghouse% netstat -k hme0
hme0:
ipackets 133465658 ierrors 0 opackets 130244542 oerrors 0
collisions 83672 defer 115 framing 0 crc 0 sqe 0 code_violations
0 len_errors 0 ifspeed 10 buff 0 oflo 0 uflo 0 missed 0
tx_late_collisions 0 retry_error 0 first_collisions 0 nocarrier 0
inits 6 nocanput 0 allocbfail 0 runt 0 jabber 0 babble 0 tmd_error
0 tx_late_error 0 rx_late_error 0 slv_parity_error 0
tx_parity_error 0 rx_parity_error 0 slv_error_ack 0 tx_error_ack
0 rx_error_ack 0 tx_tag_error 0 rx_tag_error 0 eop_error 0 no_tmds
0 no_tbufs 0 no_rbufs 0 rx_late_collisions 0 rbytes 1523404831
obytes 813184078 multircv 33440 multixmt 3 brdcstrcv 3312500
brdcstxmt 14694 norcvbuf 0 noxmtbuf 0 phy_failures 0
```

Most of these statistics are related to the Ethernet MAC and the network interface card hardware itself. In general, these counters help troubleshoot hardware problems. Every network interface card typically has different counters, which can change with the operating system releases. Sun does not officially support the `netstat` option.

Protocol Statistics

To view TCP, UDP, IP, ICMP, and IGMP statistics, you can use `netstat -s`. The command prints lots of protocol-specific information. Adrian Cockcroft and Rich Pettit describe this in their book *Sun Performance and Tuning, Java and the Internet*. The more interesting counters are incorporated into the SE Toolkit script `nx.se`. The output of `nx.se` looks as follows:

```
% /opt/RICHPse/bin/se nx.se
Current tcp RtoMin is 200, interval 5, start Thu Aug 27 00:23:09 1998
00:23:14 Iseg/s Oseg/s InKB/s OuKB/s Rst/s  Atf/s  Ret%  Icn/s  Ocn/s
tcp         0.0    0.2   0.00   0.01   0.00   0.00   0.0   0.00   0.00
Name    Ipkt/s Opkt/s InKB/s OuKB/s IErr/s OErr/s Coll% NoCP/s Defr/s
hme0        1.2    1.2   0.09   0.20  0.000  0.000   0.0   0.00   0.00
hme1        0.0    0.0   0.00   0.00  0.000  0.000   0.0   0.00   0.00
hme2        0.0    0.0   0.00   0.00  0.000  0.000   0.0   0.00   0.00
```

The script lists TCP as if it were an interface, with input and output segment and data rates, resets per second, outgoing connection attempt fails per second, percentage of bytes retransmitted, and incoming and outgoing connections per second. It then lists all the interfaces. For interfaces that provide this information (at present, only `le` and `hme`) `nx.se` reports kilobytes in and out. NoCP is nocanput. This counts the number of times a received packet was discarded due to slow processing in the TCP/IP stack and lack of buffering on input. `Defr` shows the number of defers that took place. A defer happens when an Ethernet tries to send out a packet, but it finds the medium to be busy, and it has to hold off (defer) the sending until the line is clear.

Glossary

access control list	(ACL) A file that specifies which users can access a particular resource, such as a filesystem.
accounting	Keeping track of resource usage on a machine. SRM provides accounting features.
ACL	See *access control list*.
administration Tool	A GUI tool for configuring Solaris Bandwidth Manager.
administrative domain	A collection of network elements under the same administrative control and grouped together for administrative purposes.
ADOP	See *automatic degree of parallelism*.
alarm	The means by which notification is sent when an exception occurs.
attaching	See *DR Attach.*
Alternate Pathing	(AP) A software mechanism which works in conjunction with Dynamic Reconfiguration (DR) to provide redundant disk and network controllers and their respective physical links. The main purpose of Alternate Pathing is to sustain continuous network and disk I/O when system boards are detached from a machine or DSD (in the case of the Starfire) that is running a live copy of the Solaris operating environment.
AP	See *Alternate Pathing.*
application resource measurement	(ARM) A means of measuring the end-to-end response time of a system.
ARM	See *application resource measurement*.
ASE	Sybase Adaptive Server Enterprise.

automatic degree of parallelism	(ADOP) A feature of the Oracle8*i* Database Resource Manager that attempts to optimize system utilization by automatically adjusting the degree of parallelism for parallel query operations.
backfilling	The execution of a job that is short enough to fit into the time slot during which the processors are reserved, allowing for more efficient use of the available resources. Short jobs are said to backfill processors reserved for large jobs.
BBSRAM	Boot Bus Static Random Access Memory.
blacklist	A file that enables you to specify components, such as system boards, that should not be configured into the system. The blacklist file is read and processed at bringup time.
BMC Best/1	BMC Software's BEST/1 products provide tools to address performance management requirements across OS/390, Parallel Sysplex, SAP R/3, Unix, Windows NT, VM and AS/400 environments.
CBQ	See *Class Based Queuing*.
CCMS	A tool that provides information to SAP R/3 allowing it to measure the performance of key user transactions and the response time of the backend database for applications.
CICS	See *Customer Information Control System*.
CIM	See *Common Information Model*.
Class Based Queuing	(CBQ) The underlying queuing technology used in Solaris Bandwidth Manager.
Classes of Service	A feature supported by Solaris Bandwidth Manager that allows network traffic to be organized so that urgent traffic gets higher priority than less important traffic.
classifier	A component of the Solaris Bandwidth Manager that allocates packets to a class queue. When a packet arrives, the classifier analyzes the packet protocol, ToS value, URL information, source information, destination information and allocates the packet to a class queue where it waits to be processed.
CLI	Command Line Interface, as opposed to Graphical User Interface (GUI).
cluster	A collection of computers interconnected via a high-speed interface that allows the environment to behave as a single unified computing resource.
clustered cache	A method of caching web pages where multiple servers use the intercache protocol (ICP) to talk among themselves and form an explicit hierarchy of siblings and parents. If the load would overwhelm a single server or if high availability is important, multiple servers are configured as siblings. Each sibling stores data in its cache but also uses ICP to search the caches of other siblings.

CMIP	A scalable OSI-based network management protocol that is used in situations where SNMP is not powerful enough.
codepoint	A construct used by the ToS facility of the Solaris Bandwidth Manager. The way that the DSCP bits are set in a packet is called its *codepoint*, and network devices translate DSCP codepoints to device configurations. For example, a DS compliant router that has two priority queues (one for regular priority traffic and one for high priority traffic) could be configured to have several codepoints map to the high priority queue, and several others to the regular priority queue.
Common Information Model	(CIM) A metamodel based on the Unified Modeling Language (UML) that supplies a set of classes with properties and associations. The Common Information Model provides a conceptual framework within which it is possible to organize information about a managed environment.
control interval	In control theory, the rate at which measurements are made and corrections are applied.
CoS	See *Classes of Service*.
Cross-System Coupling Facility	A WLM component that communicates policies, metrics, and control data between Sysplex nodes.
Customer Information Control System	(CICS) An interactive transaction processing system from IBM.
Oracle8*i* Database Resource Manager	A feature of Oracle8*i* that is used to allocate and manage CPU resources among database users and applications. In addition, the Oracle8*i* Database Resource Manager enables the administrator to limit the degree of parallelism of any operation. It is possible to balance one user's resource consumption against other users' by allocating flexibility among tasks of varying importance to achieve overall enterprise goals.
DDI_ATTACH	A function, used by DR (called from the `dr_driver`) that provides the ability to attach a particular instance of a driver without affecting other instances that are servicing separate devices.
DDI/DKI	Device Driver Interface/Device Kernel Interface. These are function call entry points that device drivers should implement in order to fully support DR. DDI/DKI is specified in the "Writing Device Drivers" section of the *Driver Developer Site 1.0 AnswerBook* (`http://docs.sun.com`).
DDI_DETACH	A function, used by DR (called from the `dr_driver`), that provides the ability to detach a particular instance of a driver without affecting other instances that are servicing separate devices.

DDI_RESUME	A function, used by DR (called from the `dr_driver`), that provides the ability to detach a board that contains the kernel cage (OBP, kernel, and non-pageable memory). The kernel cage can only be relocated after all of the drivers throughout the entire DSD (not just on the board being detached) are quiesced to guarantee the data integrity of the kernel cage relocation. `DDI_RESUME` resumes the drivers after the quiesce period.
DDI_SUSPEND	A function, used by DR (called from the `dr_driver`), that provides the ability to detach a board that contains the kernel cage (OBP, kernel, and non-pageable memory). The kernel cage can only be relocated after all of the drivers throughout the entire DSD (not just on the board being detached) are quiesced to guarantee the data integrity of the kernel cage relocation. `DDI_SUSPEND` suspends the drivers to begin the quiesce period.
decay	The period by which historical usage is discounted.
DEN	The Directory Enabled Networks working group. The goal of this group is to offer a standard information model and directory schemas to tie together users and applications with network elements, protocols, and services through specific relationships. By complying to this information model and the DEN schemas, different network equipment and application vendors should be able to build interoperable network elements around a central directory.
detaching	See *DR Detach*.
Diff-Serv	The Differentiated Services working group of the Internet Engineering Task Force (IETF). Diff-Serv addresses network management issues related to end-to-end Quality of Service (QoS) within diverse and complex networks.
DIMM	Dual In-Line memory Module. A memory module with higher capacity and faster performance than SIMMs (Single In-Line Memory Module) It is currently used as the memory source for all Sun Microsystems platforms.
direct control	A means of control that operates on the resource you want to control. For example the Solaris Resource Manager software controls CPU usage per user by implementing a scheduling class that decides who should get what share of the CPU.
DISC	Dynamic internal service classes created by WLM. These classes enable WLM to manage transactions. Each DISC is associated with one or more normal service classes and a given server component. The number of transactions using each route then allow the DISCs to be weighted. Thus, if the external or standard service class goal is not being met, the associated DISCs can be managed (if that is where a bottleneck lies).
distributed queuing system	A batch system product from Florida State University that is available in the public domain. The set of system resources it understands is host (by name), system architecture, operating system type, amount of memory, and CPU usage.
DMTF	Desktop Management Task Force.

DQS	See *distributed queuing system*.
DR	See *Dynamic Reconfiguration*.
DR Attach	The process of bringing a system board under the control of the Solaris operating environment through use of the DR mechanism.
DR Detach	The process of removing a system board from Solaris operating system control through use of the DR mechanism.
DSD	See *Dynamic System Domains*.
DSS	Decision Support System.
DSS/DW	Decision Support System / Data Warehousing.
Dynamic Reconfiguration	(DR) A Sun Microsystems technology supported on the Starfire and other Sun Enterprise servers which allows system boards to be added (attached) or removed (detached) from a single server or domain.
Dynamic System Domains	(DSD) Starfire independent hardware entities formed by the logical association of its system boards. Each domain on the Starfire enjoys complete hardware isolation from other domains, executes its own private version of the Solaris operating system, and is centrally managed by the SSP.
ELIM	See *Extended Load Information Manager.*
Enterprise 10000	See *Sun Enterprise 10000*.
ERP	Enterprise Resource Planning
error event	A discrete on/off event, as opposed to a continuous variable to be compared against a limit.
exception	A condition that represents a problem in processing a job. LSF can watch for several types of exception conditions during a job's life cycle.
exclusive scheduling	A type of scheduling used by LSF that makes it possible to run exclusive jobs on a host. A job only runs exclusively if it is submitted to an exclusive queue. An exclusive job runs by itself on a host. LSF does not send any other jobs to the host until the exclusive job completes.
Extended Load Information Manager	(ELIM) LSF uses the Load Information Manager (LIM) as its resource monitoring tool. To modify or add load indices, an Extended Load Information Manager can be written.
fairshare	A form of scheduling used by LSF to prevent a single user from using up all the available job slots, thus locking out other users. Fairshare scheduling is an alternative to the default first come, first served scheduling. Fairshare scheduling divides the processing power of the LSF cluster among users and

groups to provide fair access to resources for all jobs in a queue. LSF allows fairshare policies to be defined at the queue level so that different queues can have different sharing policies.

FlowAnalyzer (NetFlow) An application that uses the output from NetFlow FlowCollector. It provides very elaborate processing, graphing, and reporting options that can be used for network analysis, planning, trouble shooting, and more

FlowCollector (NetFlow) A NetFlow datagram consumer for one or more NetFlow devices. These devices simply point to the host and port number on which the FlowCollector software is running. The FlowCollector aggregates this data, does preprocessing and filtering, and provides several options to save this data to disk (such as flat files). Other applications such as network analyzing, planning, and billing can use these files as input.

Gigaplane-XB The interconnect on the Starfire that provides main memory access through a point-to-point data router which isolates data traffic between system boards and minimizes any performance degradation when memory interleaving is disabled.

goal Goal-based policies are prescriptitive rather than reactive. A goal can be translated into a mixture of limits, priorities, and relative importance levels. Goals can include actions to be performed when the goal cannot be met.

Health Monitor See *SyMON Health Monitor.*

heavily damped A system is heavily damped if you feed back a small proportion of an error over a longer control interval. A heavily damped system tends to be sluggish and unresponsive when a large time constant is used.

hierarchical fairshare A method of sharing resources, supported by LSF. Hierarchical fairshare enables resources to be allocated to users in a hierarchical manner. Groups of users can collectively be allocated a share, and that share can be further subdivided and given to subgroups, resulting in a share tree.

host based resources Resources that are not shared among hosts, but are tied to individual hosts. An application must run on that host to access such resources. Examples are CPU, memory, and swap space. Using up these resources on one host does not affect the operation of another host.

Hostview A GUI program that runs on the SSP machine (which is a component of an Enterprise 10000 system). Hostview enables you to monitor and control the Enterprise 10000. For example, Hostview can display continuous readouts of power and temperature levels at various locations within the Enterprise 10000 server.

HPC High Performance Computing

HP OpenView	Computer-oriented local and wide area networks are normally managed using SNMP protocols, with Solstice SunNet Manager or HP OpenView products collecting and displaying the data. Both products provide some visibility into what is happening in the computer systems on the network, but they are focused on network topology. Resource management is done on a per-network basis, often by controlling the priority of data flows through intelligent routers and switches.
HTTP	Hypertext Transfer Protocol. HTTP is used by Web servers to host content and respond to HTTP requests from Web browsers.
IBM Workload Manager	A comprehensive tool set for MVS that provides an automated resource management environment, driven by high level business goals, and that, in many cases, is self tuning. Tools are provided to define the business goals or objectives, to control system resources, and to feed metrics concerning these resources back to the resource controller, which attempts to ensure that the goals are met.
IETF	Internet Engineering Task Force.
indirect control	A means of control that works via resources that are dependent upon the resource that is being controlled. For example, to limit the I/O throughput of a process, it is sufficient to be able to measure the I/O throughput and limit the CPU resources for that process.
intercache protocol	(ICP) A protocol used to implement clustered caches. (See *clustered cache*.)
interleaving	See *memory interleaving*.
intimate shared memory	(ISM) A way of allocating memory so that it can't be paged. The shared memory area is often the largest component of a database's memory requirements, and is the easiest to insulate between database instances. Because intimate shared memory is wired down, the memory allocated to each database instance stays allocated and one instance cannot steal memory from another.
Int-Serv	The Integrated Services working group of the Engineering Task Force (IETF).
IP	Internet Protocol. IP is the foundation of the TCP/IP architecture. It operates on the network layer and supports addressing. IP enables data packets to be routed.
ISM	See *intimate shared memory.*
ISP	Internet Service Provider, a company that provides Point-of-Presence access to the Internet.
ISPF	Interactive System Productivity Facility, a generic MVS interface that can be used by the operator/administrator to define, activate, and deactivate policies.

Java Dynamic Management Kit	A JavaBeans based framework for developing and deploying dynamic management based applications. Autonomous agents can be deployed in real-time to perform management tasks for devices on the network.
JTAG	Joint Test Action Group, IEEE Std. 1149.1. JTAG is an alternate communications interface between the SSP machine and the Enterprise 10000 server, and is used when the standard network connection between the SSP and the Enterprise 10000 is unavailable.
Java Virtual Machine	The machine image, implemented in software, upon which Java code runs.
JVM	See *Java Virtual Machine*.
kernel cage	A special data structure (normally contained within a single system board) that controls the dynamic growth of all non-relocatable memory, including the OpenBoot PROM (OBP) and kernel memory. When Dynamic Reconfiguration (DR) is used to detach a system board containing the kernel cage, it is necessary to quiesce the operating system to ensure that no I/O or kernel activity occurs while the kernel cage is being relocated.
kernel memory	Memory that is used to run the operating system.
kernel module	A Solaris Bandwidth Manager module that contains the *classifier* and the *scheduler*.
LAN	See *local area network*.
lightly damped	If you feed back a large proportion of an error with a short control interval, the system is said to be lightly damped. A lightly damped system is very responsive to sudden changes but will probably oscillate back and forth.
LIM	See *Load Information Manager*.
limit	A simple rule with a single input measurement. It is also common to have several thresholds with a warning level action and a critical problem level action for the same measure.
lnode	Limit node, a node in a special resource tree used by Solaris Resource Manager (SRM). SRM is built around lnodes which are a fundamental addition to the Solaris kernel. lnodes correspond to UNIX UIDs, and may represent individual users, groups of users, applications, and special requirements. The lnodes are indexed by UID and are used to record resource allocations policies and accrued resource usage data by processes at the user, group of users, and application level.
Load Information Manager	(LIM) The resource monitoring tool used by LSF. The Load Information Manager process running on each execution host is responsible for collecting load information. The load indices that are collected include: host status, length of run queue, CPU utilization, paging activity, available swap space, available memory, and I/O activity.

Load Share Facility	(LSF) A software facility that provides the capability of executing batch and interactive jobs on a pool of networked computers. The Sun Microsystems High Performance Computing (HPC) package includes the Load Share Facility as a vehicle for launching parallel applications on an HPC cluster. In addition to starting batch jobs, the Load Share Facility also provides load balancing.
local area network	A set of computer systems in relatively close proximity that can communicate by means of networking hardware and software.
LPAR	Logical Partitions, an IBM S/390™ logical entity which runs its own operating system instance and it's allocated resources and managed by PR/SM.
LSF	See *Load Share Facility.*
LWP	Lightweight Process
Management Information Base	A database that contains network management variables and can be accessed via SNMP.
master host	The node where the LSF batch queues reside. When the LSF software initializes, one of the nodes in the cluster is elected to be the master host. This election is based on the order of nodes listed in a configuration file. If the first node listed in the configuration file is inoperative, the next node is chosen, and so forth.
maximum bandwidth	The amount of spare bandwidth allocated to a class by the Solaris Bandwidth Manager. The maximum bandwidth is dependent upon the percentage of bandwidth the class can borrow.
$\mathbf{MDF_{TM}}$	Multiple Domain Facility, an Amdahl Corporation™ technology which provides logical partitioning for its mainframes. By integrating special hardware for each logical partition or domain, Amdahl processor complexes could run multiple operating systems at close to native performance.
memory interleaving	A method of using computer memory that helps increase memory subsystem performance by reducing the probability of hot spots or contention in a few memory banks. This is accomplished by spreading access to multiple memory banks.
Message Passing Interface	(MPI) An industry standard interface used to parallelize applications.
MIB	See *Management Information Base.*
microstate accounting	A method of accounting for resource usage where a high-resolution timestamp is taken on every state change, every system call, every page fault, and every scheduler change. Microstate accounting provides much greater accuracy than sampled measurements.
MPI	See *Message Passing Interface.*
MTS	See *Multi-Threaded Mode.*

Multi-Threaded Mode	A database topology where a single process serves many users.
negative feedback	A method of applying feedback to a system where you take the error difference between what you wanted and what you got, and apply the inverse of the error to the system to reduce the error in the future.
NetFlow	A product from Cisco that is supported by Solaris Bandwidth Manager. NetFlow allows for detailed network measurements that can be sent to other software packages which can process and analyze the data.
Network File System	An application that utilizes TCP/IP to provide distributed file services.
network queuing system	(NQS) A public domain software product that has been enhanced by many hardware vendors. Sterling Software offers a distributed version of NQS called NQS/Exec which is geared toward a supercomputer environment. Limited load balancing is provided as there is no concept of demand queues, since it uses traditional push queues instead. There is also no control over interactive batch jobs.
NFS	See *Network File System*.
NQS	See *Network Queuing System*.
NVRAM	Non-Volatile Random Access Memory
OBP	OpenBoot PROM
ODS	Informix OnLine Dynamic Server
OLTP	Online Transaction Processing
operational policy	A policy that is implemented manually as part of operations management. For example, an availability policy can include a goal for uptime and an automatic way to measure and report the uptime over a period. There is no direct control in the system that affects uptime. It is handled by operations staff.
Oracle8*i* Resource Manager	An Oracle facility that ensures system resources are applied to the most important tasks of the enterprise at the levels required to meet enterprise goals.
PC NetLink	A product from Sun Microsystems that is based on the AT&T Advanced Server for UNIX. PC NetLink adds functionality that was not previously available on Solaris servers with products such Samba and SunLink™ PC™ (a.k.a. Syntax TotalNET Advanced Server). PC NetLink adds file and print services, and enables Solaris servers to act as Microsoft® Windows NT™ Primary Domain Controllers (PDC) or Backup Domain Controllers (BDC). For enterprises with mixed NT and Solaris servers and desktops, PC NetLink 1.0 offers many new options for utilizing hardware resources and minimizing system administration overhead.

Platform Computing Load Share Facility	See *Load Share Facility.*
PDP	See *policy decision point*.
PEP	See *policy enforcement point*.
performance index	The ratio of work completed versus the amount of work which should have been completed in order to meet the goal.
PIN	See *policy ignorant node.*
policy agent	A component of the Solaris Bandwidth Manager that implements the configuration and handles communication with the *kernel module.*
policy control	The application of rules to determine whether or not access to a particular resource should be granted.
policy decision point	In policy administration, the point where policy decisions are made.
policy element	A subdivision of policy objects. A policy element contains single units of information necessary for the evaluation of policy rules. Examples of policy elements include the identity of the requesting user or application, user or application credentials, and so forth. The policy elements themselves are expected to be independent of which Quality of Service signaling protocol is used.
policy enforcement point	In policy administration, the point where policy decisions are enforced.
policy ignorant node	A network element that does not explicitly support policy control using the mechanisms defined in the applicable standard policy.
policy object	An object that contains policy-related information, such as *policy elements*, and is carried in a request or response related to resource allocation decisions.
policy protocol	A protocol for communication between the policy decision point and policy enforcement point. The policy protocol can be any combination of COPS, SNMP, and Telnet/CLI.
POST	Power-ON self tests, a suite of hardware diagnostic tests which ensure full functionality of a system board.
preemptive scheduling	A method of scheduling where a high priority job can bump a lower priority job that is currently running. LSF provides several resource controls to prioritize the order in which batch jobs are run. Batch jobs can be scheduled to run on a first come first served basis, fair sharing between all batch jobs, and preemptive scheduling.
priority	A relative importance level that can be given to the work done by a system as part of a policy that prioritizes some activities over others.

priority decay	See *process priority decay.*
priority paging	A method of implementing a memory policy with different importance factors for different memory types. Application memory is allocated at a higher priority than file system memory, which prevents the file system from stealing memory from other applications. Priority paging is implemented in the Solaris 7 operating environment.
process measurements	Measurements that show the activity of each user and each application.
process memory	Memory allocated to processes and applications.
Process Monitor	An optional module within Sun Enterprise SyMON that can be used to view all the processes on a system. The Process Monitor can also be configured to pattern match and accumulate all the processes that make up a workload.
process priority decay	A process decay method used by SRM, where each processes priority is decayed according to a fixed decay factor at regular intervals (each second).
processor reservation	A method that allows job slots to be reserved for a parallel job until enough are available to start the job. When a job slot is reserved for a job, it is unavailable to other jobs. Processor reservation helps to ensure that large parallel jobs are able to run without under utilizing resources.
processor set	The set of processors available to a system.
Project StoreX	A technology being developed at Sun to address modern storage issues. Storage is now open for access in a heterogeneous multivendor environment, where multiple server and storage vendors can all be connected over the Storage Area Network (SAN). This is an emerging technology, and tools to manage a SAN are still being developed. Project StoreX is based on a distributed pure Java framework that can run on servers from any vendor, interface to other storage management software, and manage any kind of attached storage.
provider DSD	Dynamic Reconfiguration (DR) on the Starfire allows the logical detachment of a system board from a provider DSD (the DSD from which resources are borrowed) and the logical attachment of the same system board to a receptor DSD (the DSD where loaned resources are applied).
provider domain	When relocating resources between DSDs, a “provider domain” is the domain where a system board gets logically detached from to then have it attached to a “receptor domain”.
proxy cache	A method of caching Web pages. A proxy caching Web server sits between a large number of users and the Internet, funneling all activity through the cache. Proxy caches are used in corporate intranets and at ISPs. When all the users are active at once, regardless of where they are connecting to, the proxy cache server will get very busy

PR/SM™	Processor Resource/Systems Manager), an IBM S/390 hardware feature which allows customers to statically allocate processor and I/O resources to LPARs to concurrently run multiple operating system instances on the same machine.
QoS	See *Quality of Service*.
Quality of Service	A measure of the speed and reliability of a service. Solaris Bandwidth Manager provides the means to manage your network resources to provide Quality of Service to network users. QoS is a network-wide issue; if congestion takes place anywhere on the network, it affects the overall quality of service.
RAS	Reliability, Accessibility and Serviceability
receptor DSD	Dynamic Reconfiguration (DR) on the Starfire allows the logical detachment of a system board from a provider DSD (the DSD from which resources are borrowed) and the logical attachment of the same system board to a receptor DSD (the DSD where loaned resources are applied).
receptor domain	When relocating resources between DSDs, a "receptor domain" is the domain which receives a system board after having it logically detached from a "provider domain."
repository access protocol	The protocol used to communicate between a policy repository and the repository client. LDAP is one example of a repository access protocol.
Resource Management Facility	A component of WLM that tracks metrics including progress against goals.
RMF	See *Resource Management Facility*.
RSVP	A protocol (part of the Int-Serv framework), that provides applications the ability to have multiple levels of Quality of Service (QoS) when delivering data across the network. RSVP provides a way for an application to communicate its desired level of service to the network components. It requires each hop from end-to-end be RSVP-enabled, including the application itself (through an API). Bandwidth is reserved at each hop along the way before transmitting begins, guaranteeing that enough resources will be available for the duration of the connection.
SAN	See *Storage Area Network*.
scheduler	A component of the Solaris Resource Manager (SRM) that schedules users and applications.
scheduler term	The period of time during which the Solaris Resource Manager (SRM) ensures that a particular user or application receives its fair share of resources.

security policy	A type of policy that aims at preventing access to certain resources or allowing designated users to manage subsystems. For example, Sun Enterprise SyMON 2.0 software includes access control lists for operations that change the state of a system, and multiple network domain views to give different administrative roles their own view of the resources being managed.
SE Toolkit	A toolkit that can be used to develop customized process monitors. The Solaris software can provide a great deal of per-process information that is not collected and displayed by the ps command or Sun Enterprise SyMON 2.0 software. The data can be viewed and processed by a custom written process monitor. You could write one from scratch or use the experimental scripts provided as part of the SE Toolkit. The SE Toolkit is a freely available but unsupported product for Solaris systems. It can be downloaded from the `http://www.sun.com/sun-on-net/performance/se3`.
server consolidation	A current trend by data centers to reduce cost of server ownership by reducing physical footprint and reducing number and management cost of multivendor platforms. The basis of server consolidation is to combine applications and data contained in several smaller servers into a single larger server.
service class	A class that defines a set of goals, together with periods, duration, and importance. A number of individual processes and CICS/IMS transactions can be assigned membership of a service class. They will then become subject to the specified goals and constraints, including those imposed by any resource group subscribed to by the class. In essence, this is analogous to the SRM lnode, which effectively defines a resource management policy that can be subscribed to.
Service Level Agreement	A written agreement between system managers and end users that captures the expectations and interactions between end users, system managers, vendors, and computer systems. Often, many additional interactions and assumptions are not captured formally.
Service Level Management	The process by which information technology (IT) infrastructure is planned, designed, and implemented to provide the levels of functionality, performance, and availability required to meet business or organizational demands.
service provider	In a network policy, the service provider controls the network infrastructure and may be responsible for the charging and accounting of services.
service time	The time it takes for an I/O device to service a request. This can be complex to measure. For example, with today's disk storage systems, the device driver issues a request, that request is queued internally by the RAID controller and the disk drive, and several more requests can be sent before the first one comes back. The service time, as measured by the device driver, varies according to the load level and queue length and is not directly comparable to the old-style service time of a simple disk drive.

SEVM	Sun Enterprise Volume Manager, technically equivalent to Veritas Volume Manager.
ShareII	A resource management product from product from Softway. The Solaris Resource Manager (SRM) is based on ShareII.
shared resources	A resource that is not tied to a specific host, but is associated with the entire cluster, or a specific subset of hosts within the cluster. Examples of shared resources include: floating licenses for software packages, disk space on a file server which is mounted by several machines, and the physical network connecting the hosts.
SHR Scheduler	A component of the Solaris Resource Manager (SRM) that controls the CPU resources. Users are dynamically allocated CPU time in proportion to the number of shares they possess (analogous to shares in a company), and in inverse proportion to their recent usage. The important feature of the SHR scheduler is that while it manages the scheduling of individual threads, it also portions CPU resources between users.
Simple Network Management Protocol	(SNMP) An open network protocol used by network management systems that are based on TCP/IP.
SLA	See *Service Level Agreement*.
SNIA	Storage Network Industry Association.
SNMP	See *Simple Network Management Protocol.*
Solaris Bandwidth Manager	A product from Sun that provides the means to manage your network resources to provide Quality of Service (QoS) to network users. It allows network traffic to be allocated to separate Classes of Service (CoS), so that urgent traffic gets higher priority than less important traffic. Different classes of service can be guaranteed a portion of the network bandwidth, leading to more predictable network loads and overall system behavior. Service Level Agreements can be defined and translated into Solaris Bandwidth Manager controls and policies. Tools and APIs provide an interface for monitoring, billing, and accounting options.
Solaris Management Console	An application that provides a generic framework for gathering together operating system administration tools and interfacing to industry standard initiatives such as the Web-based management initiative (WebM) and the Common Information Model (CIM).
Solaris Resource Manager	(SRM) A software tool for enabling resource availability for users, groups, and applications. The Solaris Resource Manager provides the ability to allocate and control major system resources such as CPU, virtual memory, and number of processes. The Solaris Resource Manager software is the key enabler for server consolidation and increased system resource utilization.

Solstice SunNet Manager Computer-oriented local and wide area networks are normally managed using SNMP protocols, with Solstice SunNet Manager or HP OpenView products collecting and displaying the data. Both products provide some visibility into what is happening in the computer systems on the network, but they are focused on network topology. Resource management is done on a per-network basis, often by controlling the priority of data flows through intelligent routers and switches.

SPARCcluster A highly integrated product line that is focused on improved availability in commercial environments. Its management tools will eventually become an integrated extension to the Sun Enterprise SyMON2.0 software. For High Performance Computing, Sun HPC Servers use the Platform Computing Load Share Facility (LSF) to perform load balancing on much larger and more loosely coupled clusters.

SRM See *Solaris Resource Manager.*

SRM(IBM) The System Resource Manager of WLM. The term SRM(IBM) is used in this book to differentiate it from Solaris Resource Manager. SRM(IBM) provides the algorithms for managing resources and caters for dynamic switching between compatibility and goal modes.

SSP System Service Processor. Starfire's system administrator & system monitoring interface. The SSP configures the Starfire hardware, through a private ethernet link, to create domains. The SSP collects hardware logs, provides boot functions, and produces consoles for each domain.

Starfire See *Sun Enterprise 10000.*

static resources Host information that does not change over time, such as the maximum RAM available to processes running on the host.

Storage Area Network (SAN) A complex managed storage system, where networked storage using fiber channel makes up an interconnection layer between multiple servers or clusters and multiple storage subsystems. A storage area network can contain switches and routers just like local or wide area networks, but the protocol in common use is SCSI over fiber channel rather than IP over ethernet. A storage area network may also span multiple sites, for example where remote mirroring is being used for disaster recovery.

StoreX A technology developed at Sun that enables management of any storage resource in a heterogeneous distributed environment, from storage hardware like devices and switches, to storage software like backup solutions and volume managers. For more information about StoreX, see `http://www.sun.com/storage/storex/`.

submission host In a typical LSF workload configuration, the submission host is the node where the user or operator submits the task to be performed.

Sun Enterprise 10000	A highly scalable 64-processor (UltraSparc II) SMP server with up to 64 Gbytes of memory and over 20 Tbytes of disk space.
Sun Enterprise SyMON 2.0	A product developed by Sun to act as a user interface to hardware features. It is a powerful and extensible system and network monitoring platform that is used to manage other products. Sun Enterprise SyMON 2.0 is a Java-based monitor with multiple user consoles that can monitor multiple systems using the secure extensions to SNMPv2 to communicate over the network.
SunNet Manager	See *Solstice SunNet Manager.*
SyMON	See *Sun Enterprise SyMON 2.0*.
SyMON Health Monitor	A SyMON module that can be used in a resource management scenario to determine if a system has enough resources to run comfortably. For example, if the CPU state is reported as “red”, then either less work or more CPU power may be needed on that system. Similarly, if the memory rule reports “red” then the system may need more memory
system level measurements	A type of measurement. System level measurements show the basic activity and utilization of the memory system and CPUs. Some network measurements such as TCP/IP throughput are also available on a per system basis. Per process activity can be aggregated at a per system level then combined with network measurements to measure distributed applications.
Teamquest	A workload analysis product. See www.teamquest.com.
time constant	In control theory, the rate at which a system responds to changes.
TNF	See *trace normal form*.
ToS	See *Type of Service*.
trace normal form	(TNF) A format used to implement tracing (which makes it possible to trace the execution steps of user and kernel processes). Trace normal form, which is supported by the Solaris operating environment, provides a self-describing trace output format. Trace normal form allows data structures to be embedded in the trace file without the need for an external definition of their types and contents.
Type of Service	(ToS) A header field contained in IP packets. Its purpose is to convey information about how the packet should be routed. The Solaris Bandwidth Manager can use this information when classifying a packet. It can also change the information, to influence how the packet is routed.
UDB	DB2 Universal Database.

usage decay	A form of decay used by SRM. The user scheduler is the most important and visible portion of SRM and it implements usage decays which control long term CPU allocation responsiveness.
virtual memory	A type of memory that is allocated from a central resource pool and is consumed by an application when it requests memory from the operating system. Virtual memory is not directly related to physical memory usage because virtual memory is not always associated with physical memory. For example, if an application requests 16 Mbytes from the operating system, the operating system will create 16 Mbytes of memory within that application's address space but will not allocate physical memory to it until that memory is read from or written to.
virtual Web hosting	A web server configuration where a single server is configured to respond to hundreds or thousands of Internet addresses. Virtual web hosting is often used in situations where web sites receive little or no activity most of the time. In these situations, it is usually too expensive to dedicate a single computer system to each web site.
WAN	See *wide area network.*
WebM	Web-based management initiative
wide area network	A network that provides connectivity across a large geographical area.
WLM	See *IBM Workload Manager.*
Workload Manager	See *IBM Workload Manager.*
XCF	See *Cross-System Coupling Facility.*