Dinesh Chandra Verma

Principles of Computer Systems and Network Management

Springer

Principles of Computer Systems and Network Management

Dinesh Chandra Verma

Principles of Computer Systems and Network Management

Springer

Dinesh Chandra Verma
IBM T.J. Watson Research Center
Yorktown Heights
NY 10598
USA
dverma@us.ibm.com

ISBN 978-0-387-89008-1 e-ISBN 978-0-387-89009-8
DOI 10.1007/978-0-387-89009-8
Springer Dordrecht Heidelberg London New York

Library of Congress Control Number: 2009928696

To Paridhi
who stood by me through the entire process
of writing this book.

Preface

As computer systems and networks have evolved and grown more complex, the role of the IT department in most companies has transformed primarily to ensuring that they continue to operate without disruption. IT spending, as reported by a variety of studies, shows the trend that most of the expenses associated with IT are related to the task of operating and managing installed computer systems and applications. Furthermore, the growth in that expense category is outstripping the expense associated with developing new applications. As a consequence, there is a pressing need in the companies and organizations to find qualified people who can manage the installed base of computer systems and networks. This marks a significant shift from the previous trend in companies where the bulk of the IT department expenses were targeted on development of new computer applications.

The shift from developing new applications to managing existing systems is a natural consequence of the maturity of IT industry. Computers are now ubiquitous in every walk of life, and the number of installed successful applications grows steadily over the time. Each installed successful application in a company lasts for a long duration. Consequently, the number of installed applications is much larger than the number of projects focused on developing new applications. While there always will be new applications and systems being developed within companies, the predominance of managing and operating existing applications is likely to continue. A natural consequence of this is that the IT marketplace will continue to shift toward a technical population skilled at operating and managing existing computer applications and systems, as opposed to a technical population skilled at developing new computer applications and systems.

The education and training provided in our computer science courses, however, has not kept pace with the shift in the demand for computer scientists. While most computer science programs have a rich set of courses teaching the students the skills required to develop new systems, there is a paucity of courses that train them to manage and operate installed computer systems, both hardware and software. As a result, there is a mismatch between the demands of the IT marketplace and the supply of the technical talent from our universities. Fortunately, several universities have noticed this shift in the demand and have

started to offer courses in systems and network management. This book is intended to provide the supporting textbook to facilitate the teaching of such courses.

This book tries to define the fundamental principles that every student of systems management ought to be aware of. The algorithms, architectures, and design techniques used for different aspects of systems management are presented in an abstract manner. The technical knowledge required for different aspects of systems management is quite large, spanning mathematical domains such as queuing theory, time-series analysis, graph theory as well as programming domains such as configuration management. The book focuses on showing how the concepts from those domains are applied, rather than on the details of the specific domain – which can be found in many excellent related textbooks.

The abstract principles based approach requires decoupling systems management from the survey of the different management tools that exist currently – in both the open-source community and the commercial product offerings. This is not a book that provides a survey of existing management tools, nor tells the reader how to use such tools. This is a book that provides a survey of the techniques that are used to build those tools. However, this books enables the reader to compare the relative strengths and weaknesses of the different techniques used in these tools.

Apart from the students and teachers who are involved in learning or teaching a course on computer management, the book can also be used as a reference for understanding the basics of systems management. It would be useful for IT practitioners in companies developing systems or network management products. Such practitioners need to embody many of these principles in their products and offerings. Finally, the book should be a useful companion to practitioners involved in software development. Such developers are under an increasing pressure to deliver software that is more usable and manageable.

Structurally, the book is divided into 10 chapters. The first chapter provides an introduction and history of the field of systems management, along with an overview of three specific type of computing environments that are used as examples in subsequent chapters. It also provides an overview of the four stages in the life cycle of a computer system: planning, implementation, operations, and upgrade.

The second chapter discusses the principles involved in planning and implementation stage, i.e., how to design computer systems that can satisfy a variety of requirements ranging from performance and reliability requirements to power management requirements.

The third chapter provides an overview of the tasks required for operations management of computer systems. Operational computer systems management can be defined as consisting of two basic functions, discovery and monitoring, coupled with five analysis functions – fault management, configuration management, accounting management, performance management and security management. Chapters 4–9 deal with each of these functions respectively.

Chapter 4 discusses the subject of discovery. Discovery is the process of finding out the inventory of different components that exist in a computing environment, and the different ways in which the inventory of discovered assets can be maintained.

Chapter 5 discusses the different ways to monitor management data in computer systems, and to store them for analysis. Approaches to deal with scalability of data as well as techniques to deal with errors and data cleansing operations are discussed.

Chapters 6 and 7 discuss algorithms and approaches for fault management and configuration management, respectively.

Chapter 8 discusses the task of performance management and accounting management, including a discussion of capacity planning for computer systems.

Chapter 9 discusses the subject of security management, including a discussion of operational aspects such as security policy specification.

Chapter 10 discusses topics that are related to the principles of systems management, including subjects such as IT Service Management, ITIL, and helpdesk systems.

Overall, this book provides a holistic approach covering all aspects of systems management, and will prepare a student of computer science to take on a career dealing with systems operation and management in the new IT industry.

Contents

Chapter 1
Introduction

1.1 Introduction

Computer systems and networks are indispensable components of the modern industry. Regardless of the size of the business, be it a small shop operated by a single family or a large international conglomerate, the profitability of any business operation has a critical dependency upon proper functioning of the computer systems supporting that business. Even on the consumer side, households having multiple computers are the norm in most industrialized nations. For many of our daily needs, from browsing the news to printing out an assignment, we depend upon computers and network connectivity. Computers enable us to enjoy the fruits of modern civilization in all walks of life. However, the benefits provided by computers are attainable only when they are operating flawlessly.

Systems management is the branch of computer science that deals with the techniques required to ensure that computer systems operate flawlessly. The definition of a flaw in the computer system is any kind of problem which prevents users from attaining the benefits that the computer system can offer. A flawless operation is one in which the user is able to obtain such benefits.

A flawless operation does not mean that the computer system needs to operate at its optimal configuration, but simply that the system is operating in a manner that the user is satisfied. A computer link that is configured to operate at only a fraction of the capacity may be operating flawlessly if the total bandwidth demand on the link is an even smaller fraction. Flawless operation can be attained by properly designing and operating any computer system.

Flawless operations cannot be expected to happen automatically in a computer system without human intervention. Human beings need to be involved in a variety of roles in order to ensure that the computer system is operating well. As a matter of fact, the involvement of a human being needs to happen even before the computer systems may have been obtained or procured. In order for most computer systems to operate as desired, a human being must plan the system so that it can meet the anticipated needs when it is operational.

The exact functions that are performed by the human operator to ensure flawless operation of the system depend upon the nature of the computing

D.C. Verma, *Principles of Computer Systems and Network Management*,
DOI 10.1007/978-0-387-89009-8_1, © Springer Science+Business Media, LLC 2009

environment the operator is managing. Nevertheless, there are some common responsibilities that need to be conducted by the operators. There are a set of common principles, techniques, and approaches that can be used to deal with most of the challenges that may arise in ensuring a flawless operation of the network. The goal of this book is to provide a compilation of those common principles, techniques, and approaches.

An understanding of these common principles would enable people in charge of ensuring flawless operation of computer systems and networks to operate these systems more efficiently. On many occasions, the task of flawless operation can be aided by means of software systems – or more precisely systems management software. An understanding of the common principles would aid in development of better management software.

In this book, we will use the term computer system to refer to a collection of electronic devices that are collectively used to provide a set of useful services to human users. The computer system in this sense is a broad term which includes a collection of personal computers, workstations, mainframes, network routers, firewalls, etc. In technical publications, the computer system defined in this context is also referred to by other terms such as IT system, IT infrastructure, and distributed systems.

In order to ensure a flawless operation, we need to look at the entire life cycle of a computer system – beginning from when the plans for the system are conceived to when the system is taken off from active service. The life cycle of the computer system is described in the next section, with a discussion of the different management activities that need to be performed during each of the different stages of the life cycle. The next sections of this chapter describe some common computer systems in different environments and discuss the system management functions that need to be done in those environments throughout the life cycle. Finally, this chapter provides a brief overview of how systems management has evolved over the years.

1.2 Computer System Life Cycle

Any computer system can be viewed as progressing through a four-stage life cycle as shown in Fig. 1.1. The four stages are planning, implementation, operation, and upgrade/termination. Each of these stages is an important aspect in the life cycle of the computer systems and has its own unique requirements for the flawless operation of the computer systems.

The life cycle of the computer system in any business begins as soon as the owners of the business decide that they need a computer system to take care of some aspects of their business. Prior to the beginning of the life cycle, some key business decisions may be made such as a cap on the total amount of money to be spent on the implementation and rollout of the system, the upper limit on monthly operating expenses, the function to be performed by the computer system, the selection of the entity doing the implementation or managing

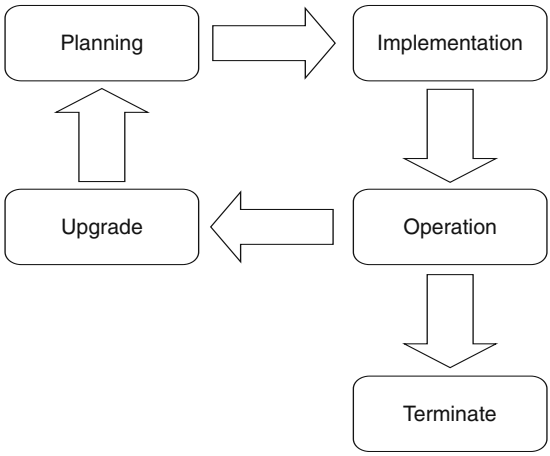


Fig. 1.1 Life cycle of computer systems

operation of the system – either an in-house entity or an external vendor, and expected revenues or cost savings or another measure of business utility derived from the computer system. When purchasing a personal computer for the home, similar types of decisions need to be taken.

The business aspects of computer systems' life cycle are outside the scope of this book, which focus primarily on the technology required to ensure the computer system's flawless operation once the business decision to acquire the computer system has been made. Nevertheless, many decisions made during the business planning stage, e.g., the budget limits, have a significant impact on the smooth operation of the system.

Once the business decision to acquire a computer system is made, the life cycle begins with the *planning phase*. The planning phase requires development of detailed plans and the design for the computer system. During the planning phase, decisions regarding how the computer system would look like are made. These include decisions about the number and types of physical machines needed to implement the systems, the network connectivity that needs to be obtained, the applications that need to be procured and installed on these systems, the configuration to put in place for the applications and machines, and the type of systems management infrastructure that ought to be put in place to maximize the probability of a flawless operation.

After the plans and designs are completed and approved, the *implementation phase* of the life cycle begins. In the implementation phase, the planning decisions are put into practice. During the implementation phase, the different types of physical machines are obtained, the applications installed, customized applications developed, and testing undertaken to validate that the computer system will perform properly once it is operational and that the implemented system conforms to the specifications that are put forth in the planning phase.

Planning and implementation for systems management purposes are very different than planning and implementation for development of new software or applications. Systems management is viewing these activities from the perspective of Information Technology department of a corporation. Such departments are usually responsible for acquiring, installing, and managing computer systems. This role is quite distinct from that of the development organizations within the same company that are responsible for developing software applications. Planning and implementation, from a systems management perspective, are determining the set of existing software and hardware components to acquire and assemble to create a computer system. In this sense, these planning and implementation are radically different from software engineering and development practices needed for developing new application software.

Once implemented, the system enters the *operation phase* of the life cycle. In the operation phase, the system is live and performing its functions. The bulk of system management functions is performed during this phase. A management system is required during the operation phase to monitor the performance and health of the computer system, to diagnose the cause of any failures that happen within the system, to ensure the security of the system, and to provide any information needed for accounting and bookkeeping purposes.

Each system has a finite operational lifetime. After the expiration of this lifetime, the system would enter either the *upgrade* or *terminate* phase. In the upgrade or terminate phase, an operational system needs to be modified or changed. An upgrade may be required for many reasons, e.g., the performance of the operational system may be below the desired level, new technology may allow for a cheaper mode of operation, or merger with another company may require changes to the operational computer systems. An example of such a change occurring due to mergers or acquisitions would be the switching of the domain names of the computers. The upgrade phase requires activities similar to the planning and implementation phase, except with the additional twist that the planning and implementations have to take cognizance of the fact that they need to be applied to an existing system, as opposed to being developed for a new installation. In the trade, it is common to refer to new installations as green-field while upgrades are referred to as brown-field.

When a system needs to be terminated, special considerations need to be taken into account that appropriate information and data have been transferred from the existing system, and that a smooth transition be made from the old system to the new system that will replace it.

The operational stage in the life cycle is the most dominant one from the systems management perspective. Some practitioners in the field take the position that the operational stage is the only part where systems management is truly needed. Nevertheless, a student of the subject should study the principles and concepts that are useful in other stages of the life cycle, since decisions made at those stage have a significant impact on how the system works during the operational phase.

Having looked at the life cycle of the computer systems, let us now look at some typical computer systems that need to be managed, how the life cycle applies to those systems, and what type of management issues arise in each type of computer system. The systems presented below are simplified representations of real computer systems, and their primary purpose is to illustrate the life cycle and management needs during different stages of the life cycle. In actual businesses, the systems deployed in real life tend to be much more complex and irregular than the simplified examples we will be discussing. Computer systems we discuss include a shared hosting data center, a network, and an enterprise computing infrastructure.

1.3 Shared Hosting Data Center (SHDC)

Many companies perform the task of hosting applications and data systems for other companies. Web-hosting companies are ubiquitous on the Internet. Many companies in industries other than information technology frequently choose to hire hosting companies to run, manage, and operate their data centers and applications. As the first example of a computer system, let us examine the infrastructure of a shared hosting data center provider. The collection of the different hardware and software components that are used to provide the hosting data center is an example of a computer system. The techniques to manage such a computer system are the subject of this book.

A typical computing environment one may encounter in a shared hosting data center is shown in Fig. 1.2. The shared hosting data center is operated by a data center operator. The data center operator supports multiple customers who are hosting their servers and applications at the center. The data center connects to the public Internet through an access router. Typically the access link to the Internet would be replicated to provide resiliency. The data center

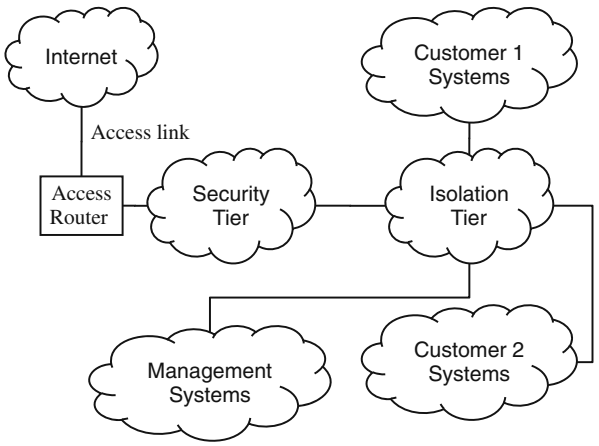


Fig 1.2 A shared hosting data center

would have service agreements with one or more telecommunication companies to provide access to the data center with assurances about the bandwidth and resiliency of the provided access links.

Immediately following the access router, a shared hosting data center would have a security tier consisting of devices such as firewalls, intrusion detection systems, and honeypots. The devices in this tier are designed to protect against any security threats that one may experience from the Internet. These security threats may be malicious person trying to gain unauthorized access to the computers of the data center, someone trying to launch a denial of service attack, someone trying to inject a virus into the computers of the data center, or any number of other possible threats.

After the security tier, a shared hosting data center would have a tier of devices designed to provide isolation among the different customers of the shared data. Such isolation may be provided using a variety of techniques, including firewalls, virtual private networks (VPN), and packet filters. The isolation tier prevents one customer from accessing the machines belonging to another customer.

The next tier consists of segments dedicated to individual customers and for internal operations of the data center. One of these computer segments would be the common management system that the shared hosting data center operator uses to manage the entire infrastructure. Internal to each customer segments there may be several tiers of computing infrastructure embedded. For example, if a customer is hosting a web site it is common to have a three-tier structure consisting of caching proxies, web application servers, and database systems. Each of these tiers may be preceded by load balancers that distribute incoming workload among different machines.

Let us now look at the different stages of the life cycle of the computer system in an SHDC. During the planning phase one needs to decide the number and capacity of Internet connections required for the data center. The SHDC operator may want to select two or three different telecommunication companies to get greater resiliency. It needs to decide whether or not to use a wide area load-balancing solution to spread traffic around the three companies providing the Internet connection. It may also opt to use one of the telecom companies as the primary connection provider and others as backup connection providers to deal with failures. It also needs to decide the type and number of security devices to install in order to implement the desired security and isolation tier functions. It will need to select the access routers and number of servers in each customer segment to meet the combined traffic load of the customers.

During the implementation phase the primary task is of acquiring and installing the machines that make up each tier of the data center. Proper attention has to be paid to physical separation between the customer segments and for efficient power management of the systems.

During the operation stage, administrators need to make sure that the Internet connection is up and the servers assigned to each customer are live and operational. These administrators would use applications running in the

management systems tier to monitor and administer the data center. Helpdesk support may be required to address any problems a customer may have. Fault detection systems may be needed to continuously monitor the infrastructure for any down or malfunctioning machines. In the event of a fault, steps need to be taken to replace or repair the malfunctioning or failed system with minimal disruption of service. Depending on the terms of the contract signed by the customer the operator may need to provide periodic performance reports demonstrating that they have met the desired performance and availability requirements of the hosted application.

If the data center is being terminated, the customer data and applications may need to be transferred to a new data center. Care must be taken to close out the accounts and clean out any customer data from the servers which are being decommissioned.

Thus, each stage of the life cycle has its own set of requirements and a common set of principles that can be applied to ensure the flawless operation of the computer systems of an SHDC operator.

1.4 Large Enterprise

As another example, let us consider the computing systems that one can encounter in a typical enterprise. An enterprise is a company, usually a commercial business, which has premises at several geographically dispersed sites. The IT environment of a large enterprise would be similar to the IT environment of a large university or a government organization, and they can also be considered as enterprises. A typical enterprise's IT environment consists of three major components: computer servers, client machines (including personal computers, laptops, and personal digital assistants), and an enterprise intranet. The typical structure of the enterprise system would be as shown in Fig. 1.3.

The enterprise consists of several sites that are connected together by an intranet. The intranet is typically a private network owned and operated by the enterprise IT department or a subcontracted company. One or more of these sites may be connected to the public Internet. Some of these sites serve as data centers, i.e., they contain several servers which run applications that are required to be operational and available to any user within the enterprise. Some of the servers at a data center may be accessible to users within the enterprise as well as users outside the enterprise. Other servers may be accessible only to users within the enterprise. Firewalls positioned between the Internet, the enterprise intranet, and the data centers ensure the separation and access control restrictions among the external and internal users.

In addition to the data centers, the enterprise has many sites where the primary users are employees with personal computers or laptops. These users access the servers and the Internet using their personal computers. Furthermore, some of the employees may be using personal digital assistants and

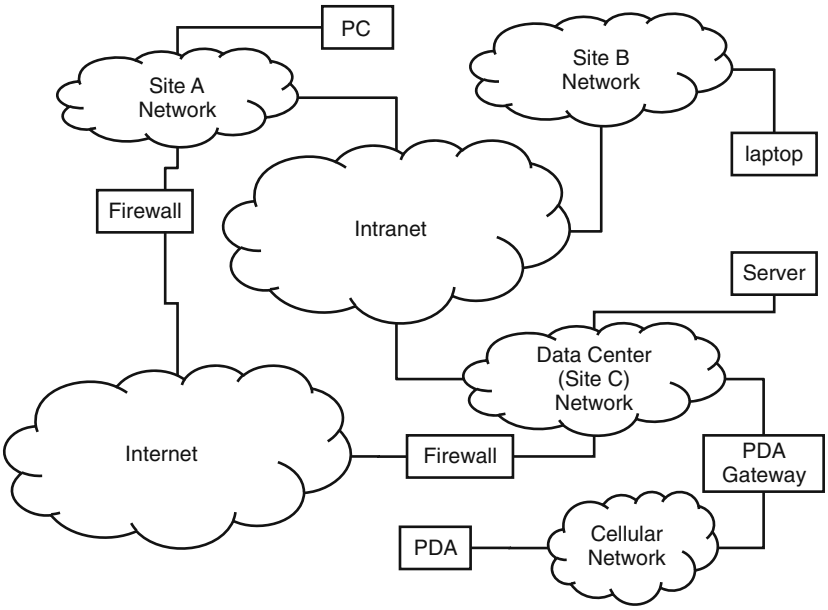


Fig. 1.3 Structure of an enterprise IT environment

accessing some of the servers in the enterprise through a cellular network or a wireless network. The enterprise needs to run the servers which support access to those applications through the wireless networks.

Thus, the system administrator in the enterprise environment would need to worry about the applications running on personal computers, laptop machines, and servers. Furthermore, the enterprise needs to support the connectivity of the various types of devices by supporting the local intranet, access to the public Internet and in some cases access to a wireless or cellular network for some applications such as e-mail access through a personal digital assistant.

Systems management in an enterprise is a complex affair with many different types of computer systems that have their own specific requirements. As a result, many enterprises have different organizations in their information technology department in charge of managing different portions of their infrastructure. Typically, the personal computers and laptops are going to be managed by a PC support division, the servers managed by a server support division, and the networks by a network support division. Some of the more complex applications may have their own support divisions, e.g., one may find separate support people for IP telephony if it is an implemented application in the enterprise. The task of the systems management in each of the support division is to ensure that the problems arising in their portion of the IT infrastructure are resolved quickly, and that the applications in the enterprise are running properly with the desired level of performance.

Many enterprises have a large computer infrastructure, and this infrastructure grows continuously due to addition of new branches, acquisition of new companies, and similar additions to existing infrastructure. Let us consider the life cycle of an enterprise system when a new branch of the enterprise is opened. During the planning phase of the life cycle, the planners need to determine if the branch site will host servers of any type, or will it be primarily client machines with network access to the intranet and Internet. They need to decide whether the site ought to be connected to the public Internet or not, and the amount of bandwidth needed on the links to the intranet and the Internet. They also need to determine the structure of the networking that will be required internally within the site. During the implementation phase, the requisite machines and applications need to be installed at the site and tested for correct operation.

During the operation phase, the system managers need to address problems that are reported by the employees of the enterprise. The problems may be diagnosed automatically by monitoring systems deployed by the system managers, or may be reported by employees reporting a problem or their inability to access a given server or application, forgetting their passwords or providing access to new employees who have joined the organization. Other functions performed by the system managers during the operation phase may include upgrading software versions of the installed applications, dealing with security incidents such as the attack of a new computer virus, reporting performance and problem resolution reports, as well as assisting other employees with any issues in using specific applications.

The proper operation of the applications in the enterprise requires coordination among all the different support departments. If a user is not able to access an application, the problem may lie with the configuration of his/her laptop computer, an issue in the intranet, or a problem with the server side of the application. The different support teams need to be aware of the outages, scheduled downtimes, and status of other parts of the enterprise if they need to be able to provide a smooth operation of the computer enterprise.

1.5 Network Service Provider

In this section, we describe the structure of computer systems that can be found in a company which supports only one type of service to its customers, providing them with connectivity to other networks, either providing them with network connectivity to other sites on their private intranet or providing access to the public Internet. Most network service providers provide a menu of connectivity services much richer than the simple connectivity model we are describing, but the example will suffice to illustrate the issues encountered in the field of systems management.

The structure of the network service provider network can be seen in Fig. 1.4. The provider would have a core network consisting of several routers. These

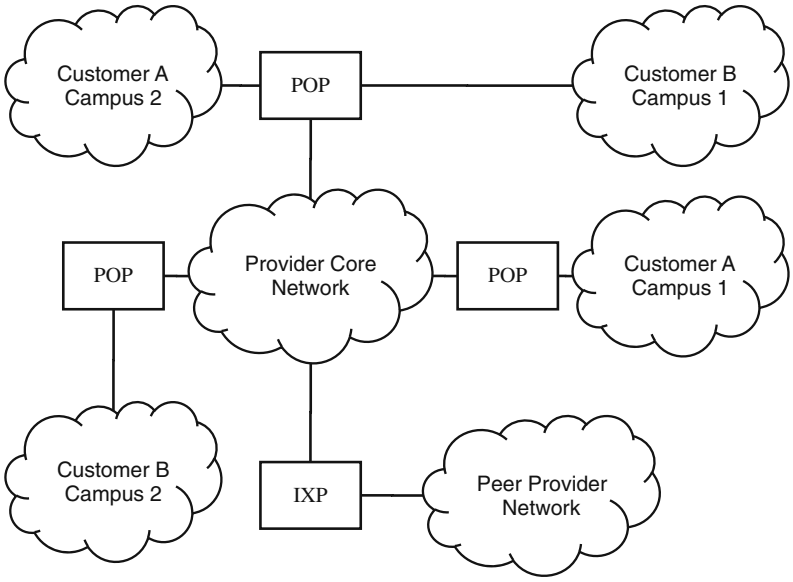


Fig 1.4 Structure of a service provider's network

routers will be operational on a set of links which may be deployed using technologies such as SONET or high-speed ATM links.

The network service provider would have multiple points of presence (POPs) at various cities. POPs are sites used to access the provider network by its customers. Each POP consists of several routers which provide access to one or more customers and connect the POP to the provider core network. Customers may connect to POPs using leased lines. For some customers, the ISP may place an access router on the customer's premises, and connect it to the POP using a metropolitan area network or a leased line.

In addition to the POPs, the network provider needs to partner with other network providers in order to connect to the public Internet. The public Internet is nothing but the collection of the networks of all the network providers in the world. These peering points are known variously as Internet exchange points or (IXPs), NAP (network access point), MAE (metropolitan area exchange), or FIX (federal Internet exchange). An IXP can connect a regional network provider to a national provider or act as conduit among networks belonging to several large providers. Different network providers would have peering agreements among themselves as to which traffic they would accept from other ISPs at an exchange point. Very large service providers also have private peering arrangements with each other.

The points of presence and exchange points of a provider are interconnected by its core network. The ISP's core network of the ISP consists of several routers connected by means of high-bandwidth circuits that may be owned by

the provider or leased from other bandwidth vendors. The core network would have a network segment which will be used exclusively for managing the operation of the network.

Let us now consider the decisions that need to be made during the different stages of the network life cycle. The decisions taken by the provider can significantly impact the operational nature of the network. The provider needs to select the cities to have a point of presence in and the locations to establish exchange points with other peer networks. It also needs to determine which of the other peer networks it ought to connect to and choose the right bandwidth for the various links, both the ones connecting it to the customers and peer network providers, as well as the ones that make its core network.

During the implementation phase, the network service provider will need to obtain the physical premises and the equipment it needs to provide its services at various locations, and test their proper operation. The provider also needs to validate that the implementation has been done in accordance with the provided specifications, and that the system is performing to the specifications that are desired.

During the operation phase, the provider needs to monitor its network for any performance or availability problems, track its usage for planning any capacity updates, and resolve any problems raised by the customers regarding the operation of the network. Customers may face problems related to connectivity as well as those related to the performance or usability of the network. Another key function that needs to be performed during operation is the collection of appropriate information to bill customers for their network usage.

When the network needs to be upgraded, the provider needs to take into account the issues related to network planning and implementation, as well as determine how best to upgrade the network without disrupting the operations and services provided to the current users.

All the above functions need to be performed for one of the many services that a provider may offer. Commercial networking service providers offer much more than basic Internet Protocol connectivity to their customers. The networking system of a commercial provider usually includes many different technologies, e.g., they may use SONET over optical fibers to obtain point-to-point connectivity in part of the network, use ATM switches on top of SONET links to create a physical network, then use IP to create another network on top of ATM. In another portion of the network, the provider may provide a DSL service to its residential customers and provide IP on top of a DSL network. While it is beyond the scope of the book to explain the alphabet soup of networking protocols like SONET, ATM, IP, and DSL, each of these layers of the networking stack has its own idiosyncrasies which require specialized technical expertise. As a result, many commercial networking service providers would operate separate network management system for each such technology. Additionally, for each application that runs on the network stack, e.g., an Internet Telephony service may need its own management infrastructure. In order to work properly, these management infrastructures need to be linked

together. Resolution of problems in an application often requires frequent interactions between the different management systems of the provider.

As can be seen from a brief examination of three typical computer environments, computer systems management is a complex and challenging task with many different aspects. Although there are many systems management problems that are unique to the different environments, there are many common requirements and needs that cut across a variety of computer systems.

Studying those common requirements and the approaches that can be used to address the problems in different contexts is the scope of the book. In subsequent chapters of the book, we will look at the systems management challenges at each stage of the life cycle of a computer system, and explore the principles that can be used to address the challenges during that stage of the life cycle.

1.6 History of Systems Management

As we look into the different principles underlying systems and network management, it is useful to get a historical perspective of how the field of systems management has evolved over the years. Systems management has morphed itself in various ways as new technologies and architectures have emerged in computer systems over the years.

In the 1960s, the dominant form of computer systems in most companies were large servers and mainframes. Each server supported many users by means of different terminals. These servers permitted management via dedicated operator consoles [1] which was a special terminal usually set in a privileged area. Each manufacturer provided its own version of a management console. The management console typically consisted of a text-oriented display of current status of the system parameters and allowed changes in the configuration files of the mainframe. There was little interoperability among the management systems provided by different manufacturers. Each server had a different management console which would not be connected together.

In the 1980s, developments in networking technology and open standards such as the Internet Protocol enabled machines from different manufacturers to communicate with each other. This led to the development of large networks of computers and open standards were needed to manage the networks. Simultaneously, the development of personal computers and the advent of client–server model for computing led to a rethinking of the ways in which systems management ought to be done. The need for standards drove to specifications such as SNMP (Simple Network Management Protocol) and MIB (management information base) format from the IETF [2], as well as the Common Management Information Protocol [3] from ITU. SNMP subsequently eclipsed CMIP as the preferred management scheme for computer networks.

Both SNMP and CMIP mimicked the dominant client–server computing paradigm of the 1980s and had an agent–manager model. In the client – server paradigm, a server application is used to provide services to many client applications. In the context of management, a *manager* provided management function to many different *agents*. An agent in each managed device provided access to the information required for management, and centralized manager software would display the information to an operator. The agent–manager architecture is the predominant management architecture in most deployed systems today.

Another key innovation of the SNMP management model was the standardization of management information through standard formats (the MIBs). Prior to the development of the standards, management information used to come in many different formats and it was difficult to compare the information from two different devices. Representing the management information in a common standard format reduced the complexity of management significantly.

Systems management technology has always adopted the dominant computing paradigms at any given time. In the early 1990s, CORBA (Common Object Request Broker Architecture) [4] gained popularity as an approach to implement distributed systems. Several system management products were developed using an agent–manager model where CORBA was used as the communications protocol instead of SNMP. While SNMP was used primarily for network devices and MIB specifications made it clumsy to represent structured information, CORBA objects could represent arbitrarily complex management information. SNMP also had a relatively weak security mechanism. A dominant telecommunications network in the 1990s, TINA (Telecommunications Information Networking Architecture) [5] based its management specifications on CORBA. CORBA-based management systems are common for managing servers and systems within an enterprise.

As the Internet based on a new protocol HTTP (HyperText Transfer Protocol) grew in prominence in the second half of the 1990s, it exposed some of the deficiencies with CORBA as management architecture. Enterprises tended to have several tiers of firewalls to guard against security threats from outside and within their networks, and CORBA protocols needed to get holes in the firewalls to communicate across each other. CORBA provided a mechanism for the manager and agents to communicate with each other, but did not provide a standard for management information, i.e., CORBA as a management protocol did not have an analogue of SNMP MIBs, and each management product vendor defined its own specifications for the same.

The Desktop Management Task Force (DMTF) [6] was formed in the 1990s to develop a common information model for general systems management – an analogue of MIBs for network management. The leading standard from that body, CIM (common information model) used an object-oriented approach to represent systems management information in a standard way. The other standard from DMTF, Web-Based Enterprise Management (WBEM), provides a transport protocol like SNMP except based on top of web-based

management. The acceptance of CIM as well as WBEM has been relatively spotty in the field, even though most leading computer companies have been participating in the standard definition process.

In addition to the developments in protocols and management information representations, the field of systems management has been quick to adopt practices and benefits from emerging technologies in the field of computing. As relational databases [7] grew in prominence, the nature of the manager in most implementations took the form of a database where management information was stored, and management functions were essentially operating and manipulating that database. Subsequently, the popularity of the Internet browser has led to many management consoles being redefined to be browser-based, where the user interface is a browser, and all management operations are done by means of scripts run at a web server. As the concept of usability and human factors have taken hold of the industry, new initiatives such as policy-based management and autonomic computing have been attempted with the goal of improving the usability of management.

Some attempts to adopt new technology for systems management have not proven very successful, e.g., initiatives to use mobile codes and active networking for management purposes, economic theory-based management paradigms, peer-to-peer management paradigms, and delegation based management approaches have not had any significant traction in the marketplace till the time of the writing of this book. A survey of such techniques can be found in [8]. Research is continuing on exploiting other emerging technology, e.g., web services, system virtualization, and business process modeling, for the purpose of systems management.

1.7 Summary

This chapter provides an introduction to the life cycle of a computer process and outlines the management needs of the computer system in each of the life cycle stages. It introduces three idealized environments for the computer systems, the shared hosting data center, the enterprise system, and the networking service provider, and discusses the system management needs in each environment during various stages of the computer system life cycle. The primary purpose of this chapter is to introduce the basic concepts and to set the stage for the other chapters.

1.8 Review Questions

1. What is computer systems management? What is the primary goal of computer systems management?
2. What are the key stages in the life cycle of a computer system? What are the principle problems addressed at each stage of the life cycle?

3. What type of computer systems do you have at your home? What are the steps you need to take to ensure that the computers are operating without any problems?
4. What are the implications of decisions made at any one stage of the life cycle on the other stages of the computer life cycle? List some of the impacts a decision made during the planning stage can have during the operation stage of the system.
5. *Project:* Do a survey of the computing systems installed at your university. To what extent does the system resemble the simplified model of enterprise network? What are the differences from the simplified model presented in this book?

References

1. A. Westerinen and W. Bumpus, The Continuing Evolution of Distributed Systems Management, IEICE Transactions on Information and Systems, E86-D, (11): 2256–2261, November 2003.
2. J. D. Case et al., Simple Network Management Protocol, Internet Engineering Task Force Request for Comments RFC 1157, May 1990.
3. ISO/IEC JTC1/SC21/WG4 N571, Information Processing Systems – Open Systems Interconnection, Systems Management: Overview, July 1988.
4. D. Slama, J. Garbis, and P. Russell, Enterprise CORBA, Prentice Hall, March 1999.
5. Proceedings of Telecommunication Information Networking Architecture Conference, Oahu, HI, April 1999.
6. Desktop Management Task Force, URL http://www.dmtf.org.
7. C. Allen, C. Creary, and S. Chatwin, Introduction to Relational Databases, McGraw Hill, November 2003.
8. R. Boutaba and J. Xiao, Network Management: State of the Art, In Proceedings of the IFIP 17th World Computer Congress – Tc6 Stream on Communication Systems: the State of the Art, Deventer, Netherlands, August 2002.

Chapter 2
Planning and Implementation

Computer systems planning and implementation are essential ingredients for smooth and efficient operation of any computer system. Planning is a required prerequisite when installing a new computer system or upgrading an existing computer system to address new requirements. Planning marks the first stage of the life cycle of computer systems. In this chapter, we look at the principles and techniques that lie underneath computer systems planning. We also discuss issues related to the implementation phase.

The input for computer systems planning is a set of requirements and the output is a plan for the computer system that will be implemented. The plan would typically include the types of physical machines and software to be acquired, the topology in which they would be interconnected, and the attributes such as capacity or processing power required of the hardware and software.

The planning process can range from a simple back-of-the-envelope calculation to a complex multi-layered multi-stage exercise involving a large team of planners and designers. Nevertheless, the goal of the planning process is to come up with the best possible design of the computer system which would satisfy the requirements at hand.

In order to understand the basic principles of planning, we need to first examine the type of requirements that are encountered in the planning process. Each requirement imposes constraints on some attributes of the computer system. Section 2.1 describes some of these requirements. It is easier to evaluate what the performance, availability, or security characteristics of a planned system are than to determine the best system that meets a set of requirements. Consequently, Section 2.2 discusses techniques for analyzing computer plans to understand the extent to which a given plan meets any specific requirements, followed by Section 2.3 which describes the general techniques to translate a set of requirements into a plan for the computer system. Finally, Section 2.4 discusses the issues related to the implementation of the design, corresponding to the implementation phase of the life cycle.

D.C. Verma, *Principles of Computer Systems and Network Management*,
DOI 10.1007/978-0-387-89009-8_2, © Springer Science+Business Media, LLC 2009

2.1 Requirements

Requirements for planning on any computer systems can be divided into two categories, functional and non-functional.

Functional requirements define what the system is expected to do, e.g., provide the function of a web site, an account payable utility, or a communications network. Functional requirements are satisfied by selecting the right set of components to build the system and by following good software engineering practices for new software development.

The focus of systems management is on *non-functional requirements*, which describe not what the system does, but how well it performs its functions. Non-functional requirements include constraints on cost, performance, capacity, availability, resiliency, and power consumption.

Each requirement provides some constraints on the attributes of the computer system. We look at each of these requirements in the different business environments that one may encounter.

2.1.1 Performance Requirements

The performance requirements on the computer system specify objectives such as maximum delay in the response of the computer system and the throughput of the system. The exact specification depends on the computer system that needs to be planned. The following examples illustrate some of the typical performance constraints that may be required in the simplified business environments we discussed in the first chapter.

Example 1: A new customer needs to be supported at a shared hosting site. The customer is an online book retailer and needs to host a web server which should be able to support up to 1000 customers per hour with the average response time of any web request submitted by a nearby client to exceed no more than 300 ms.

Example 2: An enterprise is opening a branch office in White Plains, New York. The branch will have 50 employees in the market intelligence department who will need access to the Internet as well as to the competitive market intelligence databases located in Hoboken, New Jersey. The average request to the market intelligence database should take less than 500 ms to response, and the employees should be able to get a satisfactory response time from the Internet web sites.

Example 3: A network service provider needs to provide access to a new customer between its locations in New York and San Francisco. The customer needs to have a maximum latency of 100 ms on packets between the access routers in the two cities, and a loss rate not to exceed 10 per million packets transferred. A throughput of 600 Mbp is required between both cities.

One of the challenges in real-world planning exercises is that the performance constraints may frequently be specified in relatively vague terms, and many key assumptions may not be stated. In the first example above, the performance constraint does not specify what the nature of the web request by an average customer is. The bandwidth and server load required by a web page containing simple text are likely to be much lower than a web page which is very heavy on graphics, and the appropriate mix of the content of the web pages for the customer is not specified. In the second example, the concept of satisfactory performance is ill-defined. In the third example, the average size of the packet could have a significant role in the planning process, but is not specified.

While further discussions with the person specifying the performance constraints may be able to provide some of the missing information, in many cases the planner needs to rely on data obtained from other operational systems or draw upon past experience to make reasonable estimates to provide the information that is missing.

2.1.2 Resiliency and Availability Requirements

Resiliency pertains to the amount of time the system is available for its users. Many applications are required to be available 24×7, or continuously without any perceptible failure to the users of the system. Although the system runs on unreliable components, using redundant components and options for backup sites can mask any failures that may occur within the computing environment.

Some examples of resiliency that may be required by a customer are listed below:

Example 1: A new customer needs to be supported at a shared hosting site. The customer is an online book retailer and needs to host a web server which should be available round the clock and the time for scheduled maintenance downtime should not exceed 1 h per month.

Example 2: An enterprise is opening a branch office in White Plains, New York. The Internet access to the branch must be available with a reliability of 99.99% and access to the database housing market intelligence servers must have an availability of 98.99%.

Example 3: A network service provider needs to provide access to a new customer between its locations in New York and San Francisco. If connectivity between the two cities is disrupted for more than 30 min continuously during any month, the customer gets a 25% refund on the monthly bill. If more than three such events happen in a month, the customer will get free service for the month.

In common practice, resiliency requirements tend to have the same type of vagueness and lack of specificity that we discussed in the case of performance requirements. As in the case before, the specificity is added in by making assumptions regarding the operating point of the system. In

Example 1, the definition of available is left in vague terms. In Example 2, the definition of when a system is available is interlinked with the system having a satisfactory level of performance. The satisfactory level of performance is not specified. In the third example, the definition of a disrupted connectivity is vague and imprecise. In all of these cases, the system planner needs to provide for the missing details by making reasonable estimate for the attributes that are required to make them specific.

In some cases, availability and resiliency constraints may require continuous operations even in the case of a disaster. These disaster recovery requirements can take the following format:

Example 4: A new customer needs to be supported at a shared hosting site. In the case of a disastrous event such as an earthquake, flood, fire, or terrorist attack on the data center infrastructure, the customer servers and applications need to be operational at a backup site with a total service disruption time not to exceed 2 h.

Meeting requirements for continuous operation in the presence of local disasters can be planned for, and requires designing the system to be available from more than one site, located some distance apart. It requires support during the operational state of the computer system, as well as proper design and selection of facilities in order to ensure protection against possible disasters.

2.1.3 Power and Thermal Requirements

Modern computing systems are notorious power hogs. Although each personal computer, laptop, or cell phone may consume only a small amount of power, in a place with many such devices, the total power consumption can rapidly add up. The problem is especially acute in large data centers, where there are hundreds or thousands of computers consuming an inordinate amount of electricity in total. While that electricity drives the bulk of important and necessary computing, a significant portion of that computing power gets converted to heat. A large data center requires significant investment in air conditioning to maintain temperatures at the desired operating level. The heat generated by the equipment also imposes constraints on how closely two pieces of computing equipment can be placed together. If you place two computers too close to each other, the increase in temperature can cause them to fail in unpredictable ways.

Let us get an appreciation of the amount of heat that is generated by the computing equipment by looking at some numbers. A typical laptop computer consumes about 90 W of power, which is generating an amount of heat similar to that of an incandescent light bulb operating at that temperature. The fans in the laptop computer work efficiently to dissipate the amount of heat away, so one does not feel the heat to the same extent as holding an incandescent light bulb in one's lap. A large-scale server would typically consume power in the

range of 5–10 kW, and a fully loaded rack of blade servers can consume up to 20 kW of power. With hundreds of such racks in a data center, the power bills of a data center are usually significant. At the extreme end of the spectrum, a super-computer running at 3 P flops will consume about 8.7 MW or the typical power consumption of 8700 homes.[1] A large part of this power consumption gets converted to heat.

With such a large amount of heat output, proper attention needs to be given to the thermal layout of the data center servers, and the plan for computers need to ensure that all computers are located and operated in environments where they are able to work in temperatures that are acceptable to them.

When designing the data center, one needs to take into account the heat output of the entire system, the availability and capacity of the air conditioning, and the noise output of the data center for operators working in the data center.

2.1.4 Security Requirements

Although not explicitly stated in many requirements, proper security considerations must be given to any computer system which will be connected to any type of network. Virtually all computer systems at the present age are networked and vulnerable to a variety of security exposures. The planning of any computer system needs to take into account the security threats that the system may be subject to and build safeguards to reduce exposure to those threats.

Almost every enterprise has some types of security policy document which would define a broad level of security guidelines that need to be followed. At the minimum, such security policy defines requirements for identification, authentication,, and access control. Identification is the process of establishing a unique identity to all users of the computer system. Authentication is the process of validating the identity of a user who is trying to access the network. Authorization is the process of defining which user can access the computer system and ensuring that only those users who are authorized to access a specific system are allowed to do so. Even if a security policy does not officially exist, it is good practice to determine the security requirements that will prevent most of the anticipated threats or misuse of the system.

Some examples of security requirements that may be imposed due to existing policies are listed below.

Example 1: All servers connected to the network must be protected by a local packet filtering firewall.

Example 2: All personal computers must have a local personal firewall or IPsec filters installed.

---

[1] IBM unveils BlueGene/P supercomputer to break Petaflop barrier, Wolfgang Gruener, June 26, 2007. http://www.tgdaily.com/content/view/32645/135/.

Example 3: Publicly accessible computers (e.g., in a university environment) can only be operated on a network that is isolated from networks used by other computers in the enterprise.

When designing the topology of the computer system, all security requirements, explicit or implicit, need to be taken into account. They may require establishing additional components on a personal computer or laptop, introducing additional devices in the topology of a distributed system, or putting constraints on the configuration of the software and hardware elements in the computer system.

2.1.5 Manageability Requirements

Although seldom specified explicitly, the astute planner of a computer system incorporates requirements for managing a computer system effectively. The introduction of manageability requires incorporation of functions and features which will allow an operator to manage the computer system effectively during the operational stage of the life cycle. Making the system manageable requires providing the following set of functions:

1. The ability to monitor the status (up or down) of the different components of the computer system.
2. The ability to monitor the performance of the overall system.
3. The ability to validate that the system is performing correctly.
4. The ability to manage the configuration of the various components of the computer system through a single interface.

The introduction of manageability features requires that special provisions be made for a management system that is able to provide the requisite functionality for the computer system.

2.1.6 Backward Compatibility

A computer system being planned can be either green-field or brown-field. In green-field computer systems, the entire system is being planned from scratch and there are no requirements for backward compatibility with any existing system. Brown-field computer systems, on the other hand, are intended as extensions, modifications, or upgrades to existing computer system. A web-hosting site being planned for a new company will be a green-field system. A new application added to a data center with an existing set of applications will be an instance of a brown-field computer system. The number of brown-field computer system updates generally exceeds the number of green-field computer systems' installations one is likely to encounter.

Even ostensibly green-field systems may have requirements to be backward compatible with existing systems. As an example, consider a system that is being designed for a new branch office of an existing enterprise. While the new branch office is being developed from scratch, and some elements of its infrastructure can be designed afresh, the branch office would need to support access to the existing infrastructure in the enterprise. The set of client applications it will support will be very similar to the set already running in the enterprise, and its systems would need to support policies and principles that have been set for the existing branches.

Taking backward compatibility with existing systems into account is of paramount importance in all types of systems design. After taking the constraints imposed by the existing infrastructure, the space of possible designs for the system may be reduced significantly, leading to quicker decisions regarding the plans for the new computer system.

2.1.7 Other Requirements

In addition to the requirements described above, other requirements may need to be taken into consideration when planning for a computer system. One common requirement is that of cost. The cost of a system may need to be kept below a threshold for initial installation or the recurring periodic charges for maintenance may need to be kept below a certain amount. In some organizations, there may be a requirement to keep the number of physical machines to the minimum possible to reduce the chore of tracking and maintaining the different physical assets.

Existing partnerships or alliances may dictate that a specific type of computer software or hardware system be used for the development of the new computer system. Governmental regulations and guidelines may dictate constraints on how the information maintained in the computer system is managed. In some businesses, there are specific requirements on the type of equipment that ought to be located in critical infrastructure, e.g., the telecommunication companies frequently require that equipment in their networks be NEBS (Network Equipment Building System) certified. NEBS is a set of guidelines developed in the 1970s on various metrics like resistance of fire, motion, etc., that must be satisfied by equipment used in telephonic switching systems.

The various requirements that need to be taken into account when planning a computer system sometimes conflict with each other, e.g., designing a high degree of resiliency and good performance may increase the cost of the project beyond the acceptable range. Therefore, planning often requires making trade-off decisions between the different requirements, and selecting a solution which appears to be a satisfactory match for most of the requirements at hand.

In order to determine how to design a system to satisfy given levels of performance, resiliency, availability, and other requirement criteria, we first

need to be able to compute the corresponding metrics when presented with a system plan. In the next section, we look at the various means for evaluating a computer system to determine what levels of performance, availability, resiliency, and other requirements it can satisfy.

2.2 Evaluating Computer Systems

The different types of requirements we discussed in the previous section included performance, resiliency, power, security, manageability, backward compatibility, and cost. In order to design systems that satisfy the requirements, we first need the ability to determine the same attributes for any computer system at hand. The process of studying computer systems to understand its performance, resiliency, security, cost, and other characteristics is known as the evaluation of computer systems.

There are three types of techniques used in the evaluation of computer systems, namely measurement, mathematical analysis, and simulation. In the first technique, measurement, one measures the attributes of the computer system empirically.

Measurement techniques require access to a computer system and are usually done during the operational stage of the life cycle. Measurement techniques can also be used when a system has been partially or completely implemented.

The second technique, *mathematical analysis*, requires building a model of the computer system and analyzing the attributes of the model. The model building and analysis can be done during any stage of the life cycle.

Simulation techniques can also be used at any stage of the life cycle and are a good choice when the model of the computer system is too difficult to analyze mathematically. In simulation techniques, software is written that mimics the behavior of the computer system. The simulation software is run to mimic system behavior for some amount of time and the values of the different system attributes observed from the behavior of the simulation software during the running period.

During the planning stage of the life cycle, simulation and mathematical analysis are the only feasible options. When the planning is being done as part of an upgrade to an existing system, measurement experiments done on the operational system can provide realistic inputs to feed into the models used for either mathematical analysis or simulation experiments.

A mathematical analysis or simulation of a computer system requires three activities to be performed [5]:

- Formulation of the model
- Solving the formulated model using analysis or implementing the model using simulation
- Calibration and validation of the model

In model formulation, a simplified model of the computer system, one that can be implemented in simulation software or analyzed mathematically is developed. The attributes of the model can then be analyzed or simulated. Both analysis and simulation need to make assumptions about the input to the model, and these assumptions need to be validated. The validation is easier to do during the operational life stage of the system when the results of the analysis or simulation can be compared against that of the real system. When a system does not actually exist to compare against, validation and calibration are done by other means such as looking for internal consistency in the results, or checking the similarity of results done via multiple independent modeling or simulation exercises.

When faced with the task of developing a computer system plan to meet the requirements, we need to start with a model. A useful model for planning is to consider the computer system as a collection of components. The components are interconnected together to create the model of the entire computer system. If the attributes (performance, throughput, failure characteristics) of individual components are known, then one can compose those attributes across the topology to obtain the attributes of the overall computer system.

As an example, consider a new customer that needs to be supported at a shared hosting site. The customer is an online book retailer and needs to host a web server at the shared hosted data center facility. The planner had developed a plan which specifies that the customer infrastructure will consist of three web servers and a load balancer arranged in the topology shown in Fig. 2.1.

In this plan, the customer infrastructure is connected to the shared infrastructure through the load balancer, which is used to distribute incoming requests to one of the three web servers. The network switches provide the

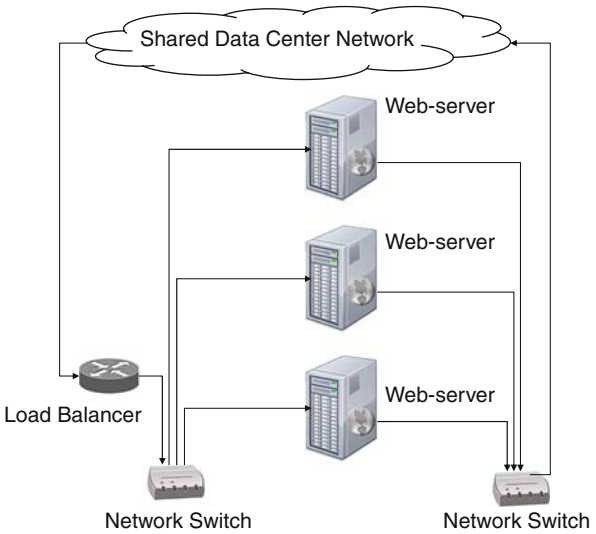


Fig. 2.1 Model of a web site

required network connectivity among the different components. The plan specifies the manufacturer and model numbers of each box shown in the figure.

If we know the performance, availability, and resiliency attributes of each of the individual components of the plan, we can compute the same attributes of the overall system. Both analysis and simulation techniques can be used for this purpose. The computed attributes can be compared against the requirements to see whether they have been satisfied.

The next few sections discuss the various attributes that can be evaluated using similar models.

2.2.1 Evaluating Computer Systems Performance

The evaluation of computer systems performance is an area that has been well studied and covered in many different books. The material in this section is a summary of some key concepts from a combination of several venerable texts dealing with the subject [1–5].

A computer system is a conglomerate of several different components. To understand the overall performance of computer systems, we look at the various ways in which the performance attributes of the individual components of the system can be aggregated together.

Performance of a computer system or any component can be evaluated by looking at the type of work requests the computer system needs to handle, and how it reacts to those set of work requests. The performance of any single component (or computer system) can be characterized by means of the following attributes:

- *Workload*: The amount of work per unit of time encountered by the component. Workload may be characterized as requests per second or in terms of inbound bandwidth such as packets per second.
- *Response*: The amount of time it takes for the component to respond to a request.
- *Loss Rate*: The fraction of requests that do not receive a response or receive an erroneous response.
- *Throughput*: The amount of work per unit of time that receives a normal response through the component. Sometimes it is also referred to as goodput.
- *Capacity*: The maximum possible throughput of the component.
- *Utilization*: The fraction of time the component is busy.

The exact definition of the workload is dependent on the nature of the component. The workload of a web server could be measured in terms of web requests per second made on the server; the workload of a database server would be measured in terms of the database transaction requests made per second on the database server; and the workload on a network switch can be measured in terms of packets per second it is required to transmit. Similarly, the response of a web

server can be measured as the difference between the time when the request is received and when the computation required by that request is completed, while the response of a network switch is measured as the latency between the time when a packet is received by the network and when the packet is transmitted out.

Different requests on a web server may require different amounts of processing at the server as well as require different amounts of bandwidths. In a realistic assessment of the performance of the system, the variations in the workload need to be accounted for. Nevertheless, approximations regarding the average performance of the system can be obtained by making the assumption that all the requests made in the workload are identical with average attributes of the entire workload.

2.2.1.1 General Principles for Performance Evaluation

There are some properties about performance which are generally true for a wide variety of computer systems and provide a handy way to do a quick analysis of any computer system performance. These principles are described in brief below.

Utilization Law

The utilization law states that *the average utilization of any component in the computer system is the throughput of the system multiplied by the amount of service time each request requires at the server*.

The utilization law is a simple statement of relationship between two performance metrics that will be valid for any computer system when measured over a reasonably large time period. The throughput of the system when measured over this time period will be the number of completed tasks divided by the time period. The time during this period that the component is busy can be obtained by multiplying the number of completed tasks by the average service time per task. The utilization is the ratio of the busy period of the component over the time period and can be readily seen to satisfy the utilization law.

The utilization law can be used for determining what size or capacity of a component to acquire for any computer system. Suppose we want to keep a server at a web site to have a utilization of no more than 50%, and we know that it takes an average of 50 ms to process each request at the server. In that case, the system throughput will be 10 requests/s.

As another example, let us consider a router which is processing packets at a rate of 100 packets per second. Each packet processing takes 2 ms. The utilization of the router is going to be 20%.

Little's Law

Little's law states that *the average number of pending requests in any computer system equals the average rate of arrival of requests multiplied by the average time spent by the request in the system*.

In other words, if we take the average latency of a computer system and multiply it with the average throughput of the system, we will get the average number of requests in the system. Little's law does require the system to be stable (i.e., the actual throughput needs to be less than the capacity of the computer system) and requires averages over a reasonably large period. Little's law is valid for any type of queuing system where people are serviced first come first served, not just for computer systems.

There are some obvious uses of the law in planning for computer systems. The following examples illustrate some of the possible uses of Little's law in making planning decisions regarding computer systems.

Example 1: Gaudy Gadgets Limited is rolling out its web site and is expecting to draw a hit rate of 100 requests/s. Gaudy Gadgets wants to maintain an average of 250 ms in the system for each request. The web-hosting software is designed so that each thread in the software handles a web request, and a server is capable of running up to 10 threads with satisfactory performance. How many servers should Gaudy Gadgets used for its web site?

Answer: The average number of active requests we would expect in the system would be 250 ms $\times$ 100 request/s or 25 requests. Since each server can handle 10 requests at the maximum, we require a minimum of three servers in the system. This implies that on the average there will be 25 threads active in the system with 5 idle threads waiting for new requests to arrive.

Example 2: Simple Corporation has decided to implement Voice over IP telephony on their intranet. The requirement is to have no more than an average of 4 requests pending at the IP-PBX system, and the delay within the PBX to be an average of 50 ms. Simple Corporation expects an average call volume of 500 calls/s. How many such PBXes will they need?

Answer: Using Little's Law, 4 equals the request rate times 50 ms, giving the rate for each PBX to be 80 calls/s. For a call volume of 500 calls/s, they would need to have nine systems.

Little's law provides an estimate of the average properties of a system. Good planning principle dictates that one should not plan a computer system to perform satisfactorily not just for the average workload, but be capable of supporting the peak workload with acceptable performance. The next principle discusses a way of estimating the peak load on any component.

Forced Flow Law

The forced flow law states that *the throughput through different components of a system is proportional to the number of times that component needs to handle each request*.

In other words, let us consider a system which consists of several components, and the throughput is C requests per second. If the throughput at the kth

component is C_k requests per second, then each resource on the average visits the kth component C_k/C times.

The forced flow law allows us to compute the throughput at a specific component when the throughput of the overall system is known. As an example, consider an authentication proxy in front of a web server. Each request into the system visits the authentication proxy twice, first for the forward request where the proxy authenticates the request and second time when the response needs to be sent out to the requesting client. If the web system has a throughput of 50 requests/s, then the authenticating proxy needs to have a throughput of 100 requests/s and the web server has a throughput of 50 requests/s.

As another example, let us consider a database system in which each request needs to access the data in the storage area network three times on the average and the database server twice. If the overall throughput of the system is 300 requests/s, then the storage area network gets 900 requests/s and the database server 600 requests/s.

The forced flow law can also be applied to systems where there is a probabilistic distribution of how requests are forwarded to the different systems. Consider a caching web proxy which is located in front of a web server. The overall system supports a request rate of 60 requests/s, but each request has an 80% probability of getting served from the cache and a 20% probability of going to the back-end web server. Then the request load on the web-caching proxy is 60 requests/s (it has to handle all of the requests), while the request load on the web server is 12 requests/s. If the system were designed to have a load balancer that distributed load equally among three web-caching proxies and a single web server, then each caching proxy would have a workload of 20 requests/s, while the web server would have the workload of 12 requests/s.

The forced flow law allows a planner to estimate the workload on each component from the overall work that is coming into the system.

3-Sigma Rule: The 3-sigma rule states that, *Most of the values in any distribution with a mean of m and variance of σ lie within the range of* $m-3\sigma$ *and* $m+3\sigma$.

The 3-sigma rule is derived from properties of normal distribution in which 99.7% of the points are expected to be in the range of $m-3\sigma$ and $m+3\sigma$, and from the observation that the probability distribution of multiple independent variables approaches a standard distribution as the number of variables being added increases.

Even if real distributions do not tend to be normal, at least not to the extent that will satisfy any decent statistician, practicing computer planners can use the 3-sigma rule as a rule of thumb in making design decisions, as illustrated by some of the examples below.

Example 1: ACME Widgets Incorporated intends to launch its web site so that it is capable of handling customers at the average rate of 100 requests/s. ACME is planning to use Apache web-hosting software on Linux using commercial off-the-shelf PCs costing 2000 dollars each, and has instrumented

that each machine can handle a request rate of 50 requests/s. ACME can also host the site on a larger server capable of handling 300 requests/s which costs 8000 dollars. A load balancer that can handle 500 requests/s costs 2000 dollars. ACME anticipates that variance in the customer request rates is 30 requests/s. Which of the two options provides a cheaper solution for ACME?

The 3-sigma rule indicates that the request rate is highly likely to be between 10 requests/s and 190 requests/s. ACME needs to plan the site for 190 request/s. It can use the server for $8000 which handles that request, or it can use four PCs and a load balancer at the total cost of $10,000. Therefore ACME will be better off with the more expensive server for hosting their web site.

Example 2: Corn Oil Corporation wants to open a new branch location in San Diego, California, with 50 employees. Their existing branch location statistics show that each employee uses an average of 40 kbp bandwidth with a variance of 10 kbp over time. Access links are available at the speed of 1.5 Mbp (T1 line), 3 Mbps (DSL), or 45 Mbp (T3). What type of access link should Corn Oil Corporation plan for?

The 3-sigma rule indicates that the data center should be planned for 50*(40 + 3*10) or 3.5 Mbp access link. Thus, the T3 line is the appropriate choice for Corn Coil for the new data center.

Safety Margins

Any type of analysis or simulation performed to determine performance is based on several simplifying assumptions and educated estimates about the anticipated workload and processing capacity of the different components in the network. Such estimations are hard to make accurately and can often be in error.

Difficulties in estimation are not a phenomenon unique to computer science. The same problem is encountered in civil, electrical, and mechanical engineering disciplines. In all of those fields, engineers build in safety margins on top of the design decisions that they make. Thus, when faced with the task of designing a bridge, it is common principle to err on the side of caution and put in a safety margin so that the bridge is able to handle more load and stress than it is rated at. A similar safety margin ought to be designed into the design of computer systems.

Building in a safety margin means that the system be designed to handle a higher amount of workload than it is expected to achieve. A useful rule of thumb is to design the system to be able to handle twice or even thrice the amount of load that is expected on the average.

There is always a trade-off between the cost of a system and safety margin requirements. If one designs a system that is twice or thrice as capable as required, that system will cost more. However, this additional cost needs to be compared against the costs that will be incurred if the additional workload

on the system cannot be sustained. If a system fails to perform, it can result in a loss of revenue, adverse publicity, and in some cases costly litigation. In many cases, those costs are prohibitively expensive. Thus, the extra hardware or software expense incurred in designing a system with an appropriate safety margin provides a more cost-effective solution.

Even with safety margins, one would need to take care of situation like flash crowds. A flash-crowd situation is when the load on a system increases suddenly by a large amount. In the Internet era, flash crowds are experienced frequently by relatively obscure systems that suddenly get publicized by a major media outlet due to an event. Under Section 2.3.2 on meeting resiliency requirements, we discuss some approaches to handle the challenges of handling flash crowds.

2.2.1.2 Queuing Theory

Performance evaluation of most computer systems can be done using queuing theory models. Queuing theory is the branch of mathematics that studies the behavior of queues. The work in each component of the computer system can be viewed as a queuing system when requests queue up to be processed by the component, and the amount of pending requests and the rate at which they are serviced determine the performance of the system.

In queuing theory, a notation of A/B/C is commonly used to denote the model of the queue, where A describes how work arrives at the queue, B denotes how the queue services the work, and C denotes the number of servers at a queue, e.g., a multi-threaded application with 10 concurrent threads can be viewed as a queuing system with 10 servers. The common terms used for A and B describing how work or service is distributed include M (which stands for the Poisson process), G (which stands for a general process), and D (which stands for a deterministic process). Thus, an M/M/1 queue is a system where work arrives according to a Poisson process, the service times are distributed according to a Poisson process and there is a single server for the queue.

A Poisson arrival process is a process in which requests arrive at the queue such that the probability that no work has arrived within time t is given by $e^{-\lambda t}$ where e is the base of the natural logarithm and λ is a constant, and each request arrives independently of the previous request. Although no real-life process in computer systems is truly Poisson, it serves as a good approximation for performance planning purposes in a wide variety of circumstances. Similarly, a Poisson service process will be one in which the probability that a job in progress is not completed within time t is given by $e^{-\mu t}$ where μ is a constant. Using the notations in this paragraph, μ will be the service rate and λ will be the arrival rate of requests at the queue.

A deterministic process is one in which the requests arrive at fixed intervals, e.g., a request every one second, or where each service time is a constant amount. Deterministic processes may be better approximation of reality in many systems. A packet forwarding router needs to process the headers of the packets which requires examination of a fixed number of bytes, with additional

processing required on a few rare occasions. It can be modeled accurately by a deterministic process. Similarly, the processing of calls at a telephone switch can be approximated as a deterministic service time. However, the mathematical models and equations that characterize a deterministic system are much more complex than that of the Poisson process.

Given the characteristics of the arrival process and the service process, queuing models provide information about the delay, service time, and average number of requests in the system. For the M/M/1 system where the arrivals are Poisson with arrival rate λ and the service rate of μ, the utilization of the system is $\rho = \lambda/\mu$. Assuming that ρ is less than 1, the average delay in the system is $1/\mu(1-\rho)$ and the number of requests in the system is $\rho/(1-\rho)$. For an M/D/1 system where the arrival rate is λ and each request takes a fixed amount of time μ to service, with $\lambda<\mu$, the average time spent in the system is $1/2(\mu-\lambda) + 1/2\mu$ and the total number of requests in the system is $1 + \rho^2/2(1-\rho)$ where ρ is λ/μ.

2.2.1.3 Queuing Networks

If the performance characteristics are known for each component of a computer system, then the overall capacity and throughput of the system can be determined by analyzing the topology of the computer system as described in the plan.

The field of queuing networks applies queuing theory to a system consisting of many queues connected together. Queuing networks can be described as either open on closed. In open queuing networks, work enters and leaves the system after completion. In closed queuing networks all requests leaving any queue are sent out to another queue in the system. The open queuing network is more frequently used to model computer systems and networks.

In general, it is hard to get closed-form equations with results for a queuing network. Even if we make simplifying assumptions about the rate at which work arrives into the queuing network, these assumptions do not remain valid once the work leaves the first queue and joins another queue in the network. The process describing requests leaving a queue may be very different from the process describing how work in entering a queue for most reasonably realistic models of service times at a queue.

A notable exception to this distortion is the M/M/1 queue. If the arrival process at a queue is Poisson and the service time is Poisson, then departure process is also Poisson. If we further add two additional criteria, then the entire network becomes much more analyzable. The first assumption is that work leaving one queue is randomly distributed to other queues or leaves the system. The second assumption is that none of the queues in the system is overloaded; then we get a simplified model called the Jackson's network, which has some nice properties, and its performance can be analyzed. None of these assumptions will be strictly true in any real system, but the models still are of great value in the planning process.

An efficient scheme to analyze queuing networks is mean value analysis. This approach can provide the mean value of queue lengths, response times, and throughput of the different nodes in the systems. Mean value analysis uses the utilization law, the forced flow law, Little's law along with queuing theory results for a single queue to come up with estimates for the mean values. A variety of other types of queuing models and queuing networks have been developed and analyzed in the field, and a comprehensive treatment can be found in many texts [6–8].

Despite the various simplifying assumptions built into the system, queuing theoretical analysis can provide useful insight into the average properties of the system. The mean values calculated, along with a safety margin built to account for the error in assumptions, can provide a useful indication of the system performance.

Canonical Delay and Throughput Curves

Another useful principle in estimating the performance characteristics of any component or computer system is the canonical delay and throughput curves that are typical of most computer systems. When measurements are taken of any computer systems, and the attributes such as delay and throughput monitored of the system, the curves typically would have the shape as shown in Fig. 2.2.

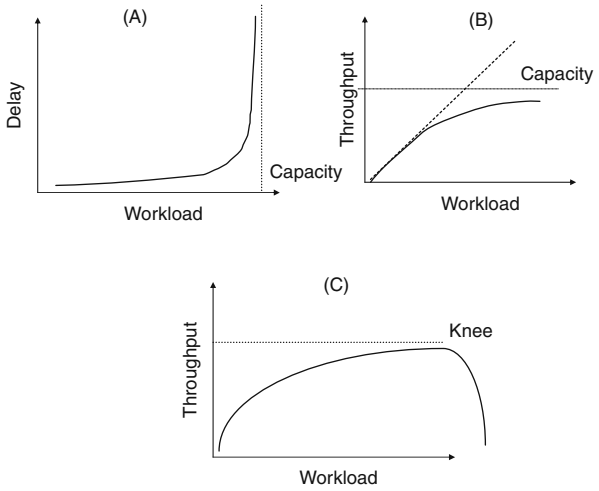


Fig. 2.2 Canonical delay and throughput curves

Figure 2.2(a) shows the typical curve of the delay encountered in the system as a function of the workload offered. When the system workload approaches the maximum capacity of the system, the delay in the system becomes extremely large.

The curve in Fig. 2.2(a) is frequently referred to as the hockey-stick function. It looks like a hockey stick if you look at the right one-third of the curve. The

key characteristics of the hockey-stick function are that it is linear when the workload is significantly lower than the capacity and grows very rapidly when the workload is very close to the capacity. In between, there is a region where the delay is sensitive to the workload that is offered.

The throughput curve in Fig. 2.2(b) also shows a similar characteristic. It shows how much throughput (the amount of work that is completed successfully by the system) changes as a function of the work offered as long as the workload is below the capacity of the system. When the system is only lightly loaded, almost all of the workload offered is completed successfully. The dashed inclined line shows the curve which would be obtained if all of the workload offered was completed successfully. As the workload increases, the effective throughput deviates from the dashed line, and more of the workload is not completed properly. The remainder of the workload will be requests that encounter some error condition or are dropped from the system. This type of throughput curve is characteristic of systems that drop additional work from the queue if it exceeds a threshold, e.g., a network router which will discard packets once its memory buffers are full.

Figure 2.2(c) shows the system behavior when the workload offers exceed the capacity of the system. In these cases, the system throughput decreases from its capacity, and drops much more sharply with a characteristic knee point which marks the significant degradation. Under these conditions, the system is said to be thrashing. A thrashing system is bad for any type of performance and needs to be avoided. The thrashing system is characteristic of a system which is overloaded but is not dropping additional work. A computer server which has too many processes (none of which are terminated) can show thrashing behavior. Similarly, a reliable transport protocol on the network which does not give up attempting to transfer data despite losses in the network will also show such behavior.

The three curves shown in Fig. 2.2 are typical for most computer systems. When planning for a new system, it is advisable to design the computer systems so that they stay close to the linear throughput–workload relationship. The challenge, however, is that the exact point where the performance begins to deviate from the linear or where the knee for performance thrashing occurs depends on system characteristics and is hard to pinpoint exactly. Nevertheless, as a rule of thumb, if the workload on any component be kept at or below 40% of the total capacity, the performance can be expected to be almost linear (i.e., deviate less than 10–15% from the linear estimation).

In commercial workloads, it is not unusual to find computers running at 10% of their total capacity even at the peak demand. Except in some special cases, e.g., in the case of mainframes where pricing plans make it advantageous to run the system at maximum possible utilization, it is better to plan computer system so that every component is running near the linear throughput–workload relationship. For all but the high-end systems, the cost of having an additional component in the computer system is negligible compared to the other costs of running the system (e.g., the salaries of the management staff or

the danger of running into performance problems and associated penalties). Therefore, it is better to target a design point in which all systems will run at low utilization.

2.2.1.4 An Example

As an example of the application of the principles of queuing theory, let us consider the simple topology illustrated in Fig. 2.3, which represents the different components and interconnections one may expect to find in a web site. The topology can be viewed as the representation of the system shown in Fig. 2.1. Web requests entering the system are first handled by a load balancer LB which has the capacity of handling up to 100 requests/s. The requests are then forwarded to the network switch N1 which can handle up to 100 Mbp. Work is then distributed among the three web servers (W1, W2, and W3), each capable of handling up to 40 requests/s. The web servers send their response back to the external clients through another network N2 which has the ability to handle 100 Mbp.

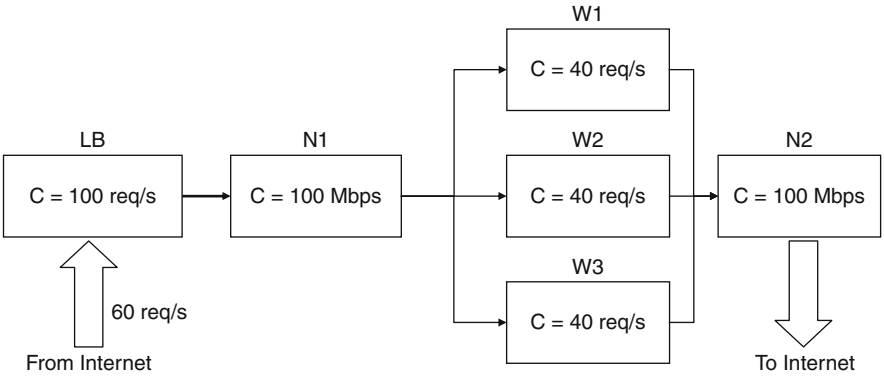


Fig. 2.3 Example of performance analysis

Let us examine what the average delay in the system will be when the inbound request rate is 60 requests/s. The load balancer distributes them randomly between the three web servers, and each gets a workload of 20 requests/s. Based on the M/M/1 model (we make the simplifying assumption that it applies), we get that the average delay in the system for each of the web servers would be 75 ms and the average delay in the load balancer will be 25 ms. Using Little's law, we can estimate that the average number of requests at the load balancer will be 3/2 and the average number of requests being processed or waiting at each of the web servers will be 1.

In order to get the end-to-end delay of the system, we will need to estimate the number of bytes each request is generating. If we assume that each of the

requests requires 1 kB packets on the network and each response is 100 kB, then the total workload on network N1 is 0.48 Mbp and the workload on network N2 is 48 Mbp. Thus, an estimate of average delay in both networks would be negligible compared to the delay in the servers.

The overall end-to-end delay in the system would therefore be 100 ms. If the web site were to offer a guaranteed response time to a set of customers, it would likely build in a safety margin of 3 and offer to meet a guaranteed response time of 300 ms.

Note that this example assumes that the load balancer LB is of a nature that requests can be received from the Internet on one network N1 and responses sent on another network N2. With some implementations of the load balancer technology, the responses will need to flow back through the network N1 and the load balancer LB for proper operation. Estimating the delay in this case is left as an exercise to the user.

2.2.1.5 Simulations

Although queuing theory provides a good model for estimating performance, there are only a few limited types of systems which can satisfy the mathematical constraints required to come up with a closed-form mathematical solutions. Even with many simplifying assumptions, the mathematical models become very complex for any realistic system. As a result, one needs to resort to simulations to understand the performance of most system designs and plans.

A simulation is a simplified model of the actual system into which one can drive a workload and observe metrics such as delays, response time, and other performance attributes of the system. The different components in a computer system can be modeled as independent processes that react to the requests coming to them. As each request is processed, it is forwarded to the other components which process them the way they would do in the real system. The software can be developed in one of two ways, either by modeling each component as one of many parallel running computer threads or by maintaining a time-ordered list of events that happen in the system and updating every component as the event list is processed one at a time. During the running of the software, instrumentation in the simulation package allows the collection of data such as delays, loss rates, and queue lengths at different components. After a reasonable length of the simulation results, a reasonable estimate of the overall performance of the system can be obtained.

In order for the simulation results to be meaningful, the workload driving the simulation needs to be carefully selected to reflect the real workload anticipated in the system. In some cases, workload may be driven off the traces or log information collected from an existing system – in which it mimics the behavior of the system from which the log was collected. Trace-driven simulations tend to give more realistic results – as long as the environment from which the trace was collected is viewed upon as being similar to the system being designed. The other way to drive the workload is by generating it as a random process. To get

reasonable estimates of the workload, a process that is realistic estimate of the workload needs to be selected and simulated.

The other key challenge in any simulation environment is to avoid the bugs that unfortunately creep into any implementation and instrumentation. Simulations can be used as a means to validate and double-check theoretic results. The output of the simulation itself should satisfy the constraints such as Little's law, forced flow law, and the utilization law. Verifying these details provides a validation function for the simulation results.

Several simulation packages providing helpful libraries for simulating the performance of computer systems and networks are available. These include products from commercial vendors as well as freely available Internet downloads. The use of any of these packages can speed up the simulation process significantly.

When interpreting simulation results and performance numbers obtained from the simulation results, one must be aware of the assumptions and limitations in the simulation model. It is generally a good idea to add a safety margin to the output of any simulation in order to get a safe performance metric that can be offered to a customer.

2.2.2 Evaluating Resiliency and Availability

The resiliency of a system is its ability to function correctly in the presence of faults. Within a computer system, faults can arise due to a variety of reasons such as

(i) Machine hardware fails due to end of its life
(ii) Software installed in the system has a bug
(iii) A disastrous event disrupts operations at a site
(iv) A machine fails abruptly because of an external event (e.g., an operator accidentally knocks a server off its perch)

The resiliency of a computer system is a measure of how well it is able to perform in the presence of such interruptions and disruptions. Given a computer system plan, there are three approaches that can be used to assess the reliability of the computer systems, namely (a) reliability analysis of the overall system using the reliability data of each component, (b) analyzing the topology of the system to identify critical components, and (c) analyzing the common types of faults and assessing their impact on systems operations.

Resiliency and reliability are subjects that have been studied more thoroughly in engineering disciplines such as avionics, space exploration, and material sciences than in context of computer systems. Theoretical measures defining resilience have been developed largely in those fields, and some of those principles are also applicable to the field of computer systems. An excellent coverage of applying these concepts to computer systems and an overview of the theory behind reliability can be found in books such as [9,10,11].

As per reliability theory, the reliability of a system is formally defined as the probability that the system continues to operate for some time period without faults. If (t) is the probability that at least one fault occurs by time t (i.e., $F(t)$ is the cumulative probability function of faults over time), then $1-F(t)$ is the reliability function of the system. This measure of reliability can be simplified to other metrics, e.g., the mean time to failure will be the expected time by which a fault would happen. Alternatively, if one is looking at a fixed time period of evaluation, e.g., looking at system operation for a month, reliability can be translated to a probability of system operating without any problems for a month.

If the reliability metric for a component is known, then they can be composed together to obtain the reliability of the overall computer system. Reliability analysis provides techniques to do this method of analysis.

2.2.2.1 *Reliability Analysis*

In order to perform reliability analysis of a system, we take the system interconnection topology and convert it into a reliability block diagram of the system. The components are the same in both representations, but the interconnection in the reliability block diagram is restricted so that two components that are connected directly are either connected in series or connected in parallel. Two components are connected in series if the failure of one of the component causes the other component to be rendered inoperative for the computer system to be used. Two components are connected in parallel if the system can perform its function if either of the two components is functional.

Figure 2.4 shows the reliability block diagram that will result for the topology described in Fig. 2.2. The same components of Fig. 2.1 are shown with the explicit enumeration of the links connecting the different boxes of Fig. 2.1. L_0 is the link between the load balancer and the network switch N1 while links L_{1i} connect the network switch N1 with the ith web server, and links L_{2i} connect the ith web server with network switch N2.

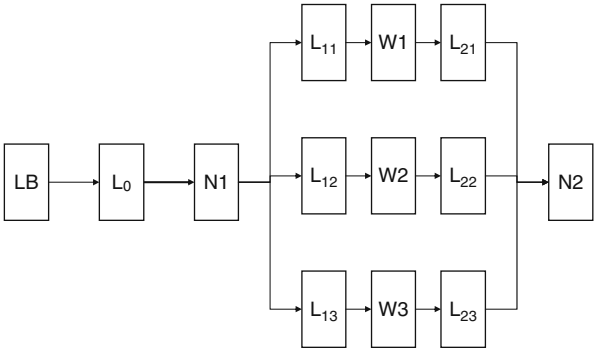


Fig. 2.4 Reliability block diagram of the web site

The reliability metric of any number of components connected in series can be calculated based on the reliability metric of the individual components. If we consider the reliability metric as the probability that no failures occur within a month, then the reliability of the three components L_{11}, W1, and L_{21} will be the product of $R(L_{11})$, R(W1), and $R(L_{21})$, where R(c) represents the reliability of an individual component. The reliability of two components C_1 and C_2 connected in parallel will be $1-(1-R(C_1))*(1-R(C_2))$. Thus, the reliability of the parallel set of web servers R(WS) will be

$$1 - \{(1 - R(L_{11})R(W1)R(L_{21}))(1 - R(L_{11})R(W1)R(L_{21}))(1 - R(L_{11})R(W1)R(L_{21}))\}.$$

The reliability of the overall system will be R(LB).R(N1).R(WS).R(N2).

For other reliability metrics, the computation of the metric can also be performed based on the structure of the reliability block diagram and the definition of the reliability metric.

The only remaining item to complete the analysis of reliability models is determining the reliability of individual components themselves. The reliability of an individual web server or system is not a number that is usually readily available. For some type of equipment, there may be experimental data available that provide a good reliability metric, e.g., within the telecommunications industry, equipments undergo rigorous laboratory testing and they may have data to provide reasonable estimates of the failure of the components. The military has standards and guidelines in place to assess the reliability of electronic components. If a component has been used in an existing operational system, and a history of failures and repair times for that component are available, then that historical data can be used to estimate the reliability metric of the component.

If no data are available from any source about the reliability of components, then the only recourse for the planner is to make educated guesses about the reliability of specific components. With appropriate and careful estimates, meaningful comparisons of relative reliability of different types of plans can be made. In many such comparisons, the actual unknown reliability metrics may cancel out or reasonable estimates about the reliability of one component versus another can be used for.

As an example, let us consider the reliability block diagram of two systems, one shown in Fig. 2.5(a) and the other shown in Fig. 2.5(b). If r_1, r_2, and r_3 are the reliability of the three components, then the reliability of the system in Fig. 2.5(a) is given by

$$r_1 * (1 - (1 - r_2)^3) * r_3$$

and that of the system in Fig. 2.5(b) is given by

$$(1 - (1 - r_1)^2) * (1 - (1 - r_2)^2) * r_3.$$

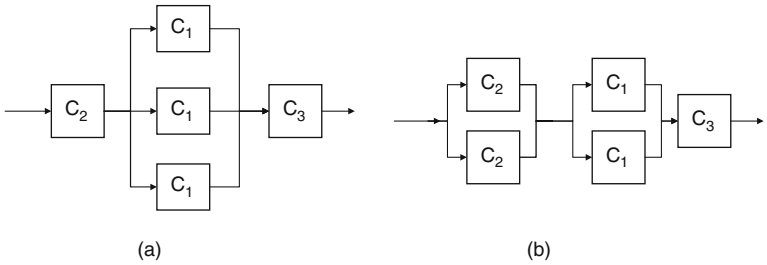


Fig. 2.5 Comparison of relative reliability

Since the third component is common, we can see that (a) is more reliable than (b) if $r_1 * (1-(1-r_2)^3) > (1-(1-r_1)^2) * (1-(1-r_2)^2)$, and less reliable otherwise. Since with typical reliability numbers both r_1 and r_2 are fairly close to 1, we can approximate $r_1 * (1-(1-r_2)^3)$ as $r_1 + 3r_2 - 3$ and $(1-(1-r_1)^2) * (1-(1-r_2)^2)$ as $2r_1 + 2r_2 - 3$. Comparing these two numbers, we can see that design (a) will be more reliable than (b) if $r_1 < r_2$. The exact reliability of the three components in this comparison is immaterial, and only the relative reliability of two components is required to make a meaningful comparison of the different plans.

Reliability block diagrams can be used for analyzing the reliability of any system for which the system operation can be characterized by means of one or more inputs coming into the system and one or more outputs leaving the system. Fortunately, that matches well with most of the computer systems one is likely to encounter in practice.

2.2.2.2 Critical Component Identification

An important function in analyzing the reliability of any system is the identification of critical component. A critical component is a component whose failure renders the system incapable of performing its operation. A common technique to determine critical components is by finding the minimum cut in the topology of the system. The minimum cut may be determined either by running a minimum cut algorithm on the reliability block diagram of the system.

The algorithm to find minimum cuts, also known as the max-flow min-cut algorithm, is a well-known algorithm in graph theory [12], and a good description can be found in any work dealing with graph theory such as [13]. Logically, the algorithm starts with the input and output into the graph representing the system, and then finds the minimum set of nodes, which when removed will result in the disruption of the work flowing between the input and the output. These nodes define the minimum cut of the system. A system may have more than one minimum cut.

The failure of any node in a minimum cut of the system causes the failure of the entire system. If the size of a minimum cut is 1, then that node is a critical component in the system. If the size of any minimum cut is K, then at least K failures must occur in the system before the system fails to operate as required.

When a system has multiple inputs and multiple outputs, then the definition of the minimum cut is specific to the pair of input and output that is selected. In these cases, the system can continue to operate and provide service to some flows, even when the service is disrupted to other flows. In such environments, one needs to associate each node with a metric that is a measure of the disruption caused by the loss of that node. One such metric could be the percentage of flows in the system that will be disrupted if the node were to fail. The most critical node will be the node that has the highest value of the disruption metric.

As an example of this approach, let us consider the structure of a hypothetical network represented by the topology shown in Fig. 2.6. The routers in the middle of the network (represented by square boxes) are used to provide connectivity to four customer networks shown as the clouds. There are thus 12 possible communication flows among the 4 customer networks. The bold numbers outside the routers represent the number of flows out of these 12 which will not be possible if the router were to fail. As can be seen from the figure, the routers providing access connectivity to the customers are the most critical ones, since their failure disrupts the flows from the connected customer network to other networks, thereby disrupting a quarter of the traffic flows. On the other hand, the other routers are interconnected by links, enabling the traffic flows to be rerouted via other paths when they fail. Thus, their failure does not disrupt any flows, since the flow can be rerouted via other routers.

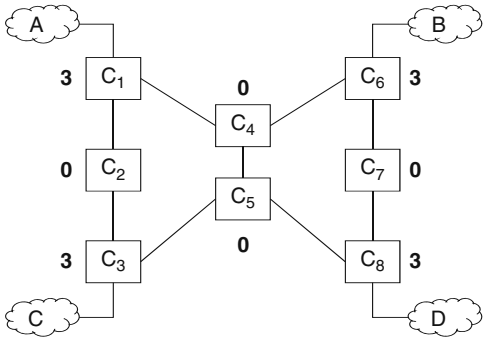


Fig. 2.6 Criticality of routers in a network

Figure 2.6 also illustrates a reason why formal resiliency analysis methods are better to understand critical components of a network, rather than visual analysis or rules of thumb. From a visual inspection, it would appear that the routers in the central column would be the most critical, but inspection of the feasible paths shows it is otherwise.

The assignment of criticality by the number of disrupted flows is a special case of assigning importance to different components for reliability purposes using the number of cuts [14]. There are other measures of criticality or importance that can be assigned to different components in a system [15]. One measure used frequently is the Birnbaum importance – defined as the rate of increase in overall system reliability as a ratio of the increase in the reliability of a component. If analytic estimates of system reliability exist, then the component with the highest Birnbaum importance can be identified readily. Other measures of criticality, based on probabilities of different kinds of faults, and the contribution of a component to the occurrence of that fault can also be defined [16].

2.2.2.3 Failure Effects Modeling

You may have noticed one issue with the reliability modeling discussed thus far. While reliability of the system can be derived once the topology and the estimates of individual components have been obtained, the definition of the reliability block diagram for any reasonably large system would be an onerous exercise, and analyzing its properties subject to various simplifying assumptions. In the case of software systems, doing a reliability block diagram and analyzing it are not a commonly established practice. Furthermore, as we discussed in the analysis section, getting good measures of the reliability of a component may be problematic in many cases. The principle of failure effects modeling provides a way to analyze the reliability of a system without necessarily going into quantitative estimates of reliability.

Using the failure effects modeling analysis (FEMA), the reliability of a system is analyzed by looking at the likely failures that can occur in the system. The analysis calls upon the reliability evaluator to list the following information for each type of failure that can occur in the system:

- The component that may fail
- The manner in which that component may fail
- The effect of the failure of the component
- The possible cause of the failure
- The controls and corrective measures that exist to mitigate that failure
- The corrective actions that can be taken furthermore for that failures

Optionally, some of the analysis may include a quantitative measure of the severity of the failure or its impact on overall critical measure. Different industries, e.g., automotive, aviation, military, and the government, have developed their own specifications for the format and nature for failure mode and effects analysis.

This type of analysis can be conducted as an audit exercise on the design of a planned system or on the information available for an operational system. Unlike the modeling of the system for quantitative analysis, this approach requires an individual or team with expertise in the operation of the specific

computer system to systematically go through the enumeration of possible faults in the system, and ensuring that the impacts of any types of failure of the component on those systems are considered.

The failure mode and effects analysis is usually considered a bottom-up approach, where all types of faults that can happen on a component are considered, and the impact of those components of the overall system analyzed. An alternative approach for performing a similar audit will be to consider the type of error that can happen in the overall system, and then decompose that into a set of explanations (failure of components that can likely cause those events). That type of investigation may be done using a technique called the fault tree analysis. In fault tree analysis, each system error is represented graphically as the Boolean combination (using AND and OR combinations) of underlying cause, and the resulting tree analyzed to understand which combinations can cause the overall system failure.

Although FEMA bypasses the need to do extensive quantitative modeling and analysis of the system, it still requires a reasonable attention to detail to ensure that all potential failure modes of components have been taken into account.

Doing a FEMA audit of a system can often reveal cases for system downtime that were not considered in the original design. One common case for system failure could be that the system needs to operate under environments that it was not originally designed for. The other common reason for system failure would be when an unpredictable disastrous event affects the system. The disastrous event could be a major event such as an earthquake or fire in the office building housing the servers, or it can be as mundane as an employee tripping over a wire and unplugging a critical server.

2.2.3 Power and Thermal Analysis

One aspect of the power consumption analysis of a computer system design is relatively straightforward to do. Each component of the computer system plan consumes power, and each hardware device that is required to implement the plan will specify the power it consumes. Adding up the power consumption figures of all the components will quickly provide an estimate of the overall power used by the design of the computer system.

Unfortunately, the life of a computer systems planner is never so easy. Two anecdotes that are part of the computer systems management community folklore are listed below to explain the issues in analyzing the power requirements of the company. Many variations of these anecdotes can also be found floating around in various places of the cyberspace.

The first anecdote deals with the availability issue of a new data center that was installed. The applications and machines installed in the system worked perfectly fine, except that the machine running a critical application crashed

and rebooted itself with unfailing regularity every Thursday evening. The software and hardware team tried hard to find the bug in the computer system which caused the system to crash periodically, but to no avail. Finally, one programmer who had stayed up in the center late one Thursday night to discover the problem found the cause. Every Thursday, the janitor came to clean the premises, and since all the power outlets in the center were busy, unplugged the most easily accessible plug to operate the vacuum cleaner, thereby crashing the server which was connected to the outlet. Being the most considerate gentleman, the janitor always remembered to plug the disconnected server back in, thereby rebooting the server.

The second anecdote deals with that of a small computer company that was acquired by a larger corporation. The data center or the acquired company was running reliably without any problem for the last 3 years. Soon after the acquisitions, the decision was made to consolidate all the machines into a single data center at the new company. Shortly after the consolidation, the software running on the servers started to show erratic errors, and the servers started to crash unexpectedly without any apparent problems. No janitorial assistance was found in this case. Upon subsequent analysis, it transpired that the machine of the acquisition were placed too close together in the data center, causing the temperature of the servers to rise beyond their operating range, and causing unexpected crashes of the system.

Proper power management of any system requires planning a system such that the total power consumption of the machines at any location does not overwhelm the electric supply of the location, and that the computing equipment is located physically in spots which causes them to operate at a reasonable range of temperature.

In smaller enterprises or home-based business, the management of power and thermal requirements may not be a significant issue since the number of machines is fairly small, and they are kept reasonably far away from each other so that the combined heating due to the locations of servers is not a major issue. However, in data centers, thermal management requirements ought to be given due considerations. We focus on evaluating the thermal characteristics and planning for large-scale data centers.

There is one key difference between evaluating the thermal characteristics of a computer system plan as compared to other types of plan design. As we will see in Section 2.3.1, there are different levels of plans that can be developed to meet the requirements imposed on a computer system. The evaluation of thermal characteristics of a computer system plan is dependent on its physical layout, while the other characteristics such as performance or availability can be determined on the basis of its logical layout. Nevertheless, understanding the thermal characteristics of a computer plan is essential to good thermal management of the data center.

For analyzing thermal and power requirements of the computer system, we will assume that the data center cooling characteristics are fixed. Significant improvements in the data center power and thermal management can be

obtained by making changes to the manner in which it is cooled [17, 18], but a system manager is more likely to find that he/she has to place IT equipment on data center racks where the cooling facilities are fixed and cannot be modified.

One method to understand the thermal characteristics of a physical layout of a computer system is to develop a thermal map of the computer system after the deployment of the computer system. The thermal map gives a plot of the temperature of different points of a data center which results from an interaction of the heat generation from the different types of equipment, and the airflow created by the cooling system. Software allowing such type of thermal mapping is available from many suppliers and major server vendors. The basic technique used for modeling the thermal loss is computational fluid dynamics [17–19]. This modeling can be viewed as a simulation of the thermal characteristics of the system. Many vendors also offer the services to physically record the temperature at various points in a data center and to create a physical map of the temperature at various locations.

Even if a full thermal map is not available, the physical characteristics of the data center layout provide for easy ways to check whether a given plan satisfies the constraints of the thermal characteristics of the data center. In any computer equipment, there is an air intake side which takes in cold air and blows it over the computer processor to cool it down, with the hot air exiting from the other side of the equipment. Thus, we can identify a hot side of the equipment and a cold side of the equipment depending on the airflow characteristics associated with it.

In a data center, cold air is usually blown from under the floors of the data center and the hot air returned from the ceiling to the air-conditioning units. In data centers, there are racks of computer equipment that are usually aligned together into large rows. One of the recommended practices for large-scale data center is that the equipment be placed so as to create alternating " cold aisles" and " hot aisles" between the rows of racks. In other words, equipment is placed so that either the hot sides of equipments in adjacent racks face each other or else the cold sides of the equipments face each other. The setup of the data center aisles in these cases looks like the configuration shown in Fig. 2.7.

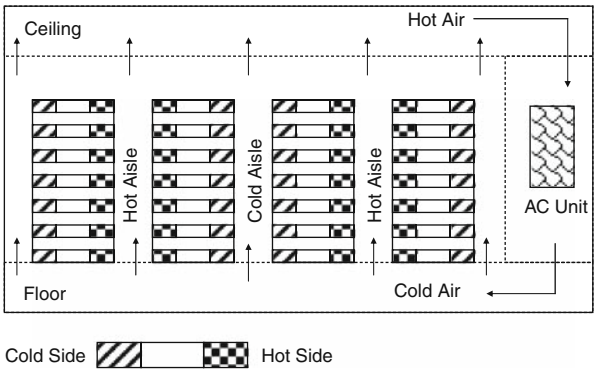


Fig. 2.7 Hot and cold aisles in a typical data center

We provide a simple back-of-the-envelope technique to compute the temperature of the different servers placed in the racks in such an environment. These calculations are modeled after the simplified calculation techniques discussed in [20,21]. The temperature in the hot aisle as well as the cold aisle increases from the bottom of the data center to the top of the data center. If we look at a single rack, one can model the thermal properties of the single rack as shown in Fig. 2.8. The temperature at the top of the cold aisle, before it mixes with the top of the hot aisle can be estimated to be the same as the ambient room temperature. The air-conditioning system pumps cold air through the floors at the cold aisle, and does not do so at the hot aisle. Therefore, the temperature of the floor of the first rack in the hot aisle can be approximated as the ambient room temperature, and the temperature of the floor of the first rack in the cold aisle is the temperature at which cold air is pumped by the air-conditioning system, usually 55°F.

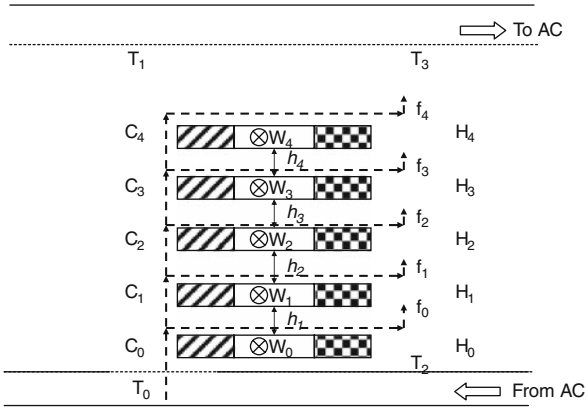


Fig. 2.8 Airflow in the server racks

Suppose the air-conditioning unit is pumping in cold air from the floor of the cold aisle at the rate of F CFM (cubic feet per minute), and the equipment located in ith position has a fan that causes f_i CFM of air to be pushed over the equipment from the cold aisle to the hot aisle. The wattage of the ith equipment is W_i and the distance between the ith and ith equipment is given by h_i. The temperature in the cold aisle after the ith equipment is C_i and the temperature in the hot aisle after the ith equipment is H_i.

The boundary conditions are the temperature at the bottom and the top of the cold and hot aisles, which are marked as T_0–T_3 in the figure.

Assuming that the combined airflow through all of the equipment Σf_i is less than F, we can obtain the temperature on the cold aisle of the side by the approximation

$$C_0 = T_0,$$
$$C_l = T_0 + (T_1 - T_0)(f_0 + f_1 + \cdots + f_i)/F.$$

On the hot aisle, the temperature at each layer of the equipment is increased directly in proportion to the wattage of the equipment, and inversely in proportion to the airflow through the equipment and the separation between the layers of the equipment.

Therefore,

$$H_i = T_2 + (T_3 - T_2)\,(W_0/h_0f_0 + \cdots + W_i/h_if_i)/(W_0/h_0f_0 + \cdots + W_n/h_nf_n).$$

Assuming that the temperature at the top of the cold aisle and at the bottom of the hot aisle is the ambient room temperature, the above analysis provides an approximation to the actual temperature of the equipment at the data centers. The temperature at the top of the hot aisle can be estimated by noting that the temperature recorded at the input of the air conditioner will be the average of the temperatures at the top of the hot and the cold aisles.

Proper functioning of the equipment requires that all equipments in the rack be operating in a temperature range which it is designed for. By comparing the temperatures on the corresponding side in the hot aisle with the allowed operating temperature control range, one can validate whether the data center is meeting the desired thermal characteristics of all of the devices. If the constraint is not satisfied, then the separation among the different pieces of the equipment ought to be increased.

Since the temperature in the hot aisle is the least at the bottom, it would also be appropriate to place equipment that has the lowest range of operational temperature at the lower end of the aisle.

Thermal and power analysis is one of the many functions that one needs to consider as part of the facilities planning for data centers. A more comprehensive list of functions and activities that need to be taken into account when planning computing facilities can be found in [22].

2.2.4 Computer System Security Analysis

It is common in the design process of computer systems that no explicit security requirements may be stated explicitly. However, it is implicitly understood that any computer system would be implemented so as to protect against the standard security threats and vulnerabilities that may be expected against the normal operation of the computer system. In this section, we discuss some of the general analysis of security concerns that can be done in a general way during the planning process.

The challenge in computer security analysis then is that of determining that a computer system has been planned so as to guard against the potential threats against the system. Such an evaluation may be made using one of two approaches – either comparing how well the computer system compares against a specific set of established criteria for evaluating security in computer products

or checklists or performing a vulnerability analysis of the system. There have also been steps taken toward quantitative evaluation of security assessment, but quantitative methods have not been widely adopted within the systems management community yet.

2.2.4.1 Comparison Against Checklists

When an enterprise has a set of standing security policies, the security policies can be converted into a checklist of items that can be validated against the proposed design of a security system. In many systems, more than one set of checklists may be used, with each checklist being used to compare against some aspects of the security design. Several standard security checklists are offered and maintained by national and international organizations for the security of enterprise IT security. A set of checklists are available from the US National Institute of Standards and Technology at the URL http://checklists.nist.gov/. The checklists specify the set of guidelines and principles that ought to be enforced in any installation of the computer system. Obtaining the relevant set of checklists from this or other resources and validating that they are satisfied provide an improved level of assurance that the security requirements.

In systems designed by the governmental and defense agencies of many countries, including the United States, Canada, and countries in the European Union, any product that is used in a solution is required to satisfy standards known as the common criteria. The common criteria was developed by unifying a European standard for evaluating security products (ITSEC) and a US Department of Defense standard (TCSEC – also known as the Orange Book). The specification allows a security product, e.g., a firewall or an operating system, to be certified as operating at specific levels of security criteria. Common criteria provides different levels of certification depending on how rigorously security processes were followed in developing a product. However, common criteria only provides an assurance regarding the process that was used for developing the product – without necessarily making any statement regarding the security functionality of the product. Nevertheless, if one is in an organization that requires adherence to any security certification, part of the security evaluation would require validating that the products included in the plan satisfy those constraints. For security systems that include cryptography modules, another standard that is used for certification is the FIPS-140 certification whose levels provide a measurement of the level of security offered by a certified product.

2.2.4.2 Vulnerability Analysis

Another approach to evaluate the security of a system is to perform the evaluation of the proposed plan against a set of possible vulnerabilities. To a large extent, such a vulnerability analysis can be viewed as an analogue of the

failure effects modeling analysis used for resiliency, except that the failure being modeled in this case is security vulnerabilities.

When vulnerability analysis is done in this manner for a planned computer system, a team of experts sits down and prepares a spreadsheet that contains the following typical items for the computer system:

- The potential source of the threat (e.g., an external hacker, an insider)
- The potential motive of the source (e.g., financial, curiosity, or revenge)
- The potential actions the attacker may take
- The impact of the potential actions
- The safeguards planned in the system against the potential source of attack.

The exercise serves to enumerate all the potential attacks and exposes the set of potential threats against which no appropriate plans exist.

The two generic approaches of comparing against predefined checklists and performing an analysis against requirements can also be used for evaluating whether a proposed design meets the manageability. They can also be used to check backward compatibility requirements.

2.3 Planning to Satisfy Requirements

The techniques described in the previous sections can be used to determine the different attributes of a computer system plan that shows the set of components and their interconnection. In this section, we will explore the ways to do the reverse, i.e., given a set of requirements, determine a computer system plan that is able to satisfy the given set of requirements.

The problem of finding a computer system plan that is the best one to meet a given set of requirements is harder than the reverse problem. In some cases, it can even be shown that determining the best plan is NP complete. To make matters worse, one set of requirements may be conflicting with another set (e.g., cost, performance, and security requirements often conflict with each other).

The planning process consists of making four choices: (a) determining the number and location of physical machines, (b) determining the software of each machine, (c) determining the capacity and configuration of the machines at each location, and (d) determining the interconnection topology among the different machines.

In many cases, some of the steps may be obvious from the requirement specifications while other steps (or part of a step) may require significant investigation. When a wide area network is designed, the locations of access routers are known, but one needs to determine the locations of internal network routers (or switches) and the capacity of links between them. A discussion of algorithms for wide area network design and planning, given a set of requirements is available in, [23,24]. When a system needs to be developed for installation is a data center, the location of all of the machines will be in the various

racks of the data center and is known. However, the number of machines and their connectivity that ought to be used need to be determined.

The planned network needs to support all of the requirements at the same time. The complexity of the planning process varies largely. In some cases, it may be as trivial as choosing among two or more solutions offered by competing vendors of machines, while in other cases it may require an iterative decision process. The iterative process is defined for a comprehensive overview here, but it should be understood that in many contexts some of the steps may not be needed.

2.3.1 Systems Planning Process

The systems planning process can be viewed as developing a system design or plan by following a four-step process as shown in Fig. 2.9. In the first step of the process, an architecture for the system is developed. The architecture is mapped to a logical design in the next step. The logical design is then translated to a concrete design. Finally, the concrete design is translated into the layout of the computer system.

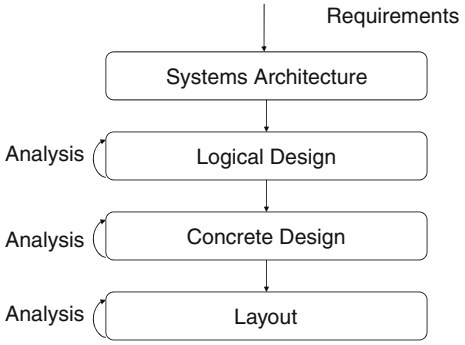


Fig. 2.9 Systems planning process

A more detailed description of the activities and contents of these steps is provided below:

Architecture: In the architecture step, decisions are made regarding the different layers or zones that will be present within the system. An architecture describes the higher level decomposition of the system into well-defined areas, where each area provides a specific function required for the functioning of the computer system, and specific protocol choices are made for systems communicating with each other. As an example, the architecture for a shared hosting data center may call for a security zone, a caching zone, and a web-serving zone, and mandate that the HTTP protocol is used universally for communication among the zones.

Logical plan: In the logical plan of the system, the types of physical machines that are to be located in each architectural zone are specified. The logical plan may specify that the security zone of a web-hosted site will consist of a set of packet filtering firewalls, the caching zone will consist of a load balancer followed by three web caches, and the web serving zone will consist of a load balancer followed by two web servers, and that each zone will be connected by a local area network. The performance, reliability, and throughput characteristics of each of the machines are determined.

Concrete plan: In the concrete plan of the system, the exact hardware and software of each machine are specified. The vendor and machine type for each of the devices are determined.

Physical layout: In the physical layout of the system, the location of the hardware assets is determined. This includes specifying which of the different rack locations a server would be located in, and the closest in which the different communications equipment will be placed in.

During each of the stages, the plan or design that is produced needs to be analyzed to validate that it satisfies all the requirements desired from the system. The set of requirements that are evaluated may not be the same at each layer of the design, e.g., the layout may only look at thermal requirements, the logical design may look at performance and availability, and the architecture ensures that manageability and security requirements have been satisfied.

And it is also worth remembering that the architecture definition from a system's planning perspective is very different than the architecture of a software or hardware system. When a system planner is creating the architecture for the system, the goal is to reuse existing pieces of hardware and software that are commercially available or developed previously.

2.3.1.1 Architecture Definition

When developing the architecture of a computer system, it is common to reuse existing architectures that are known to be used as standard best practice. It is a common practice for an enterprise to develop their own customized architecture for specific applications or systems. In general, the architectures tend to be application-specific. The architecture of an IP telephony system would be very different than the architecture of a campus network. Nevertheless, there are some common principles that can be used to advantage when making architectural decisions and definitions. Some of the most common architectural principles that one encounters in practice are specified below.

Tiering

An architecture consisting of multiple tiers is common in many data centers. In a tiered architecture, the architecture consists of several tiers, and components in one tier only communicate with the components in the tier adjacent to

themselves. Each tier consists of a set of computers which are dedicated to performing one specific type of function.

The three-tier architecture is commonly used for many web sites. In the three-tier architecture, the first tier consists of components that are responsible for rendering content for viewing and presentation to the browser of the user, the second tier consists of components responsible for manipulating and generating the information to be provided to the user, and the third tier consists of components that store any persistent data required for generating the information required of the user. Typically, the first tier would be implemented using web servers (e.g., Apache), the second tier would be implemented using application servers (e.g., tomcat, IBM WebSphere Application Server), and the third tier implemented using database servers. The protocol used between the client and the first tier is HTTP, the protocol between the web server and application server is also typically HTTP, and SQL queries using a remote procedure call paradigm can be used between the second and third tiers. The three-tier architecture is shown in Fig. 2.10(a). A similar architecture can be found in many other enterprise applications as well.

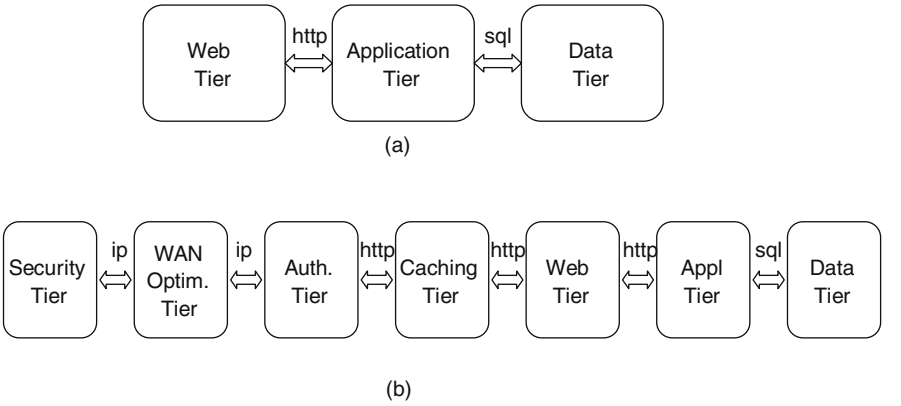


Fig. 2.10 Tiered architectures

For some applications, three tiers may be too many, and they may choose to have only two tiers, the first one consisting of web servers and the second one consisting of database servers. On the other hand, for some environments, additional tiers may be added to provide a secure and scalable environment. As an example, a data center may choose to define an architecture consisting of seven tiers shown in Fig. 2.10(b). The architecture has the applications in data center separated by a protection tier from the public Internet. This tier may be followed by a WAN optimization tier which is responsible for selecting from multiple network providers that provide access to the site. The next tier may contain authentication servers, followed by a tier of caching proxies. The three tiers below the data center may be the web server, application servers, and databases.

Perimeter Defense

Perimeter defense is a commonly used architecture principle in many environments. Perimeter defense design defines the entire environment into zones, with any two zones of an architecture that can potentially be accessed by two different organizations that do not fully trust each other be separated by an intervening security zone.

An example of an architecture employing the perimeter defense principle is the concept of demilitarized zones in data centers. The architecture defines four zones: an internal zone consisting of the applications used internally by an enterprise, an internal security zone preventing access to the internal zone by any system except the authorized ones in the demilitarized zone, a demilitarized zone consisting of applications that can act as proxies for external users and are only allowed to access internal applications using a limited set of protocols, and an external firewall which prevents access to anything except the applications in the demilitarized zone. The demilitarized zone architecture is shown in Fig. 2.11(a).

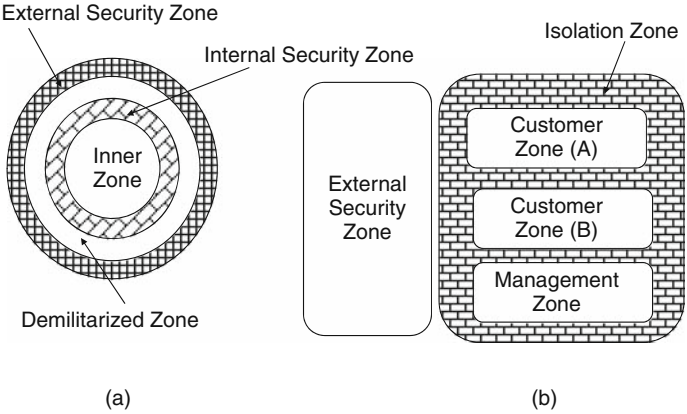


Fig. 2.11 Tiered architectures

Another example of perimeter defense is found in the shared hosting data center introduced in Chapter 1. The architecture of such a system consists of different zones per customer, separated by a customer isolation zone, one possible zone for a shared management functions, and a security zone isolating the data center from external Internet. The architecture of the perimeter defense in such systems is shown in Fig. 2.11(b).

Separation of Management and Operations

Another common architecture principle is the separation of regular operations with control and management operations. In such architectures, a separate zone is created for management and control operations. The management system

may manifest itself as a separate operations center where administrators monitor and control the operation of the other zones which are focused on operational aspect. Such an operations center is commonplace is almost all enterprises or service providers with a complex IT infrastructure.

Another place where this architectural principle manifests itself is in campus and branch networks where the role of machines that provide normal operations is separated out from the machines that provide support roles. Machines that provide support roles by running applications such as the domain name server, DHCP servers, or SLP servers are located in a separate control zone while the DHCP client or SLP client is incorporated in the personal computers and printers that are used for normal operations.

Hierarchical Architecture

In many networks and distributed systems architecture, a hierarchical structure and organization is common. In the hierarchical architecture, different zones of the computer system are arranged in a hierarchy. A core zone is at the top of the hierarchy, with other zones hanging off the zone as branches, and yet other zones may hang off as sub-branches from the branch zones.

A common example of the hierarchical architecture is the network architecture in many enterprise intranets as well as telecommunications network providers. The network architecture in an enterprise will typically consist of a campus network architecture which usually will have the function of connecting all the machines in the campus to access networks of the enterprise. The access networks would be metropolitan area networks connecting multiple branches together. A core network would connect the different access networks together.

The hierarchical architecture is shown in Fig. 2.12. The core zone is zone I and zones II and III forming the next two levels of hierarchy.

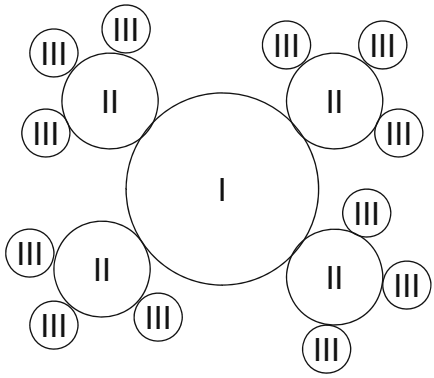


Fig. 2.12 Hierarchical architecture

The hierarchical architecture often manifests itself as hub-and-spoke design. One of the points in each of the zones acts as the hub to which the other components in the zone connect together. Then, the hubs are connected

together into a cluster of their own, and another hub (or super-hub) selected among them to aggregate further up the hierarchy.

In addition to communication networks, the hierarchical principle is found in architectures for high-volume message queuing systems, text messaging, and storage area networks.

Standardization of Components

Another key principle in architecture of many systems is the use of standard components in each zone. Thus, each zone is assumed to consist of only a single type of device or at the least a very small number of devices.

Standardization is a common technique used to improve manageability of a system. As an example, if the architecture of an enterprise IT infrastructure consists of client machines, access networks, servers, and storage devices, one would often find the enterprise developing standards mandating a single type of configuration (including hardware and software) for the clients, and its analogue in other zones of the architecture. By reducing the variability in the types of devices one may encounter, the management overhead and expertise required of support staff for troubleshooting, configuring and maintenance of the devices is reduced significantly.

Business needs and practical configurations (an enterprise may acquire another company which uses a different type of machines) prevent absolute standardization in any moderately sized enterprise, but the principle is still useful to simplify the management aspects of the system.

2.3.1.2 Logical Plan Development

Having the architecture of the computer system being planned at hand simplifies the task of designing a system to meet the desired requirements significantly. Since the architecture definition consists of defining different zones, with each zone consisting of a similar set of components, one can design each zone to meet the set of requirements required of that zone.

The first step in the development of the logical plan is to translate the set of requirements for the overall system into a set of requirements for each zone of the architecture. The next step consists of determining the number and interconnection of the components so that they can meet the required performance, resiliency, reliability, and other requirements.

In order to illustrate the development of a logical plan, let us use the example of a web site which needs to meet the following requirements:

- *R1:* The customer is an online book retailer and needs to host a web server which should be able to support up to 1000 customers per hour with the average response time of any web request submitted by a nearby client to exceed no more than 300 ms.

- *R2:* The site should be available round the clock and the time for scheduled maintenance down time should not exceed 1 h per month.
- *R3:* In the case of a disastrous event such as an earthquake, flood, fire, or terrorist attack on the data center infrastructure, the customer servers and applications need to be operational at a backup site with a total service disruption time not to exceed 2 h.
- *R4:* The site should be protected against hackers and intrusions from the Internet.

The architecture selected for the site calls is a tiered one with four zones: a security zone, a caching zone, a web server zone, and a database zone. The logical plan needs to stay within the parameters of the architecture, but satisfies the requirements stated above.

In order to develop the logical design, the first step is to decompose the requirements of the system into the corresponding requirements on each of the zones. From the architecture, we can see that requirement R4 will be satisfied due to the presence of the security zone, but requirements R1–R3 needs to be translated to the requirements of each zone. Requirements R2 and R3 probably translate down to each of the zones as is (with only perhaps a reduced requirement on the downtime allowed per month), and request R1 needs to be translated into performance requirements of each zone.

The performance requirement of the system has a throughput requirement and a latency requirement. We can use the forced flow law discussed in Section 2.2.1 to translate the throughput requirement to that of the different zones. All requests need to flow through the security zone, but a fraction p of the requests are going to be served by the caching zone, and a $(1-p)$ of those requests going to the web site. Each such request will be serviced twice by the web site zone and the web-caching zone, once on the request, the other time on the response, and once by the database zone. Therefore, a request of 1000 customers per hour translates to a request of 1000 $(2-p)$ requests for the cache, $2000(1-p)$ for the web site zone, and 1000 $(1-p)$ for the database zone. Assuming p is 80%, this translates to a throughput requirement of 1200 requests/h for the caching zone, 400 requests/h for the web server zone, and 200 requests/h for the database zone. We are making the simplifying assumption that the processing time at the caching site and web server on the request and response path are the same.

For the same numbers, we can see that the response time criteria can be broken into the relationship $(1-p)c + p(2c+2w+d) < 300$ where c, w, and d are the response times of the caching, web-serving, and data zones, respectively. While different response time values can be selected to satisfy this constraint, let us say that we break the response time requirement by dividing it equally so that the target latencies are $c = 80$ ms, $w = 50$ ms, and $d = 100$ ms.

The design choice now comes down to selecting a system of web caches which can provide a delay of 80 ms or less with the rate of 1200 requests/h, a set of web servers which can provide a delay of 50 ms or less at 400 request/h, and a set of database servers which can provide a delay of 100 ms or less at 200 requests/h.

We now need to look at the available components that we have at each tier, and what their performance characteristics are. We can then select the right number of components at each zone to meet the performance requirements. If we have a caching server that provides a delay of 80 ms or less at the rate of 500 requests/h, web servers that provide a delay of 50 ms at 250 requests/h and a database that provides a delay of 100 ms at the rate of 300 requests/h, then we can see that the logical design ought to consist of a load balancer with three caching servers, a load balancer with two web servers, and a single database server. Additionally, we would need to include the network switch connecting the different tiers in the logical design, and the devices included in the security zone (the right number of firewalls that can sustain the rate of 1000 requests/h).

In order to maintain the reliability and availability, the principle used is logical design is that of redundancy. Each of the components (load balancers, servers, database) can be replicated and connectivity arranged so that one of the components is the primary, and the other is a backup in case the primary fails. For meeting the requirements of disaster recovery, a backup site which can become operational if the original site goes down needs to be selected and maintained so as to be ready to take over in case of disaster.

For other types of architectures, similar reasoning can lead to a reasonably well-determined set of logical plans for the computer system. Once the logical design has been made, the next decision point is the selection of the hardware and software packages, and determining which of the different racks in the data center the different pieces of equipment ought to be plugged into.

It is possible to develop software programs which can take a set of requirement specifications and output a set of computer system plans. However, the same steps can be invoked manually by system planners, using computer software as assisting tools in some of the steps.

2.3.1.3 Approaches for Satisfying Requirements

At any stage of the planning process, one needs to ensure that the requirements imposed on the system are satisfied. In the most generic method, one can come up with a new plan for each stage of the planning process, and then evaluate the plan characteristics against the desired requirements. In the previous example, we used some simple analysis reasoning to determine that the requirements are satisfied. However, that determination is based on several estimates (e.g., probability of cache hit and expected delays based on some catalogue information). All of these estimates have a margin of error. In order to get a better assurance that the requirements will be met in the logical design, some general techniques can be used. We discuss some of those techniques below.

Reuse from a Catalogue

In many organizations, there is an existing set of logical designs that comply with the architecture adopted by the product and whose performance,

resiliency, and other characteristics are known and validated from prior experimental studies or simulation results. Thus, one may know the performance of four potential configurations in the four-zone-tiered architecture, such as

- Configuration A: 2 firewalls, 4 caches, 3 web servers, 1 database
- Configuration B: 2 firewalls, 6 caches, 3 web servers, 1 database
- Configuration C: 3 firewalls, 6 caches, 4 web servers, 2 database
- Configuration D: 3 firewalls, 10 caches, 4 web servers, 2 database

Furthermore, experimental validation has shown that performance numbers of the configurations are as follows:

- Configuration A: 600 requests/h throughput; 300 ms average delay
- Configuration B: 1000 requests/h throughput; 200 ms average delay
- Configuration C: 1500 requests/h throughput; 200 ms average delay
- Configuration D: 2000 requests/h throughput; 100 ms average delay

Given any requirement specification less than 2000 requests/s and an average delay requirement over 100 ms, we can select one of the above configurations which is good enough to satisfy the desired delay and throughput constraints.

Extrapolate from a Catalogue

The selection from a catalogue may not always work if the desired requirements do not lie within the range which the configurations in the catalogue satisfy. Furthermore, a lower cost configuration may be obtained if one adopts a configuration that is slightly different from the one specified in the catalogue.

Looking at the configurations in the example above, one can notice that the only difference between configurations A and B and configurations C and D is in the number of web caches. One may therefore draw a reasonable conclusion that interpolating the performance numbers as a function of web caches between the range of those two configurations may provide another potential configuration with its own set of performance attributes, e.g. average configurations A and B would mean that a throughput of 800 requests/h at an average delay of 250 ms second might be a reasonable guess for a configuration with 5 caches. This is still an estimate, but might be okay as a rule of thumb.

In general, one can try to extrapolate among the parameters of configurations with known performance numbers and come up with a better configuration to meet a specific performance target.

2.3.1.4 Dynamic Composition of Configurations

A more sophisticated combination among different known configurations can be done by means of dynamically composition different configurations with known performance. As an example, if we had performance, reliability, and availability characteristics of a number of cache-only configuration, server-only configurations, and database-only configurations, one can try different

combinations of the ones with known performance, estimate the architectural zone that is the performance bottleneck, and try out a variety of combinations and estimate the performance provided by that combination.

A dynamic composition approach that takes the different sub-components with known performance characteristics, expert-defined rules to combine the sub-components properly, and then using a branch-and-bound algorithm to search for combinations that can satisfy a given set of requirements is proposed in [25]. The approach provides more flexibility in using the catalogue of known configurations and can create new configurations that are not in the catalogue.

It is worth reminding ourselves that when one is extrapolating from the catalogue or dynamically composing known configurations, simplifying assumptions are made to estimate the performance of the new configuration, which may not always hold when the system is actually deployed. Thus, a reasonable safety margin ought to be used in combination with the results of such estimated or generated configurations.

2.4 Implementation

The implementation phase of a computer system life cycle has received little attention in the computer science research community. Part of the reason for a lack of interest is that the implementation phase deals largely with the non-technical issues of obtaining the devices needed for putting a computer system in place, wiring the different components in place, configuring the different components, and testing the overall system for correctness. Nevertheless, there are important considerations during the implementation phase which can have a significant impact on how the system behaves during operation.

The objective in the implementation phase is to correctly implement the planned system within the budget limits. Generally, this requires a good project management on the implementation process, with careful attention to work breakdown structures and keeping track of the completion of the different operations. Often, a program management tool would be needed to keep track of the progress of the project. An extensive discussion regarding the aspects that need attention during project management of computer systems is provided in [26, 27].

After the system is implemented, it needs to be subjected to a variety of checks to validate that it has been implemented correctly. This validation also needs to be done as a part of the network management during the operational phase of the life cycle.

2.5 Summary

This chapter discusses the systems management needs during the planning and implementations stages of the life cycle of a computer system. In the planning stage, the design of the computer system needs to be developed so that it can

satisfy a variety of requirements. The requirements may include constraints on performance, availability, security, and power consumptions. Given a plan, the analysis techniques that can check whether the requirements are satisfied are introduced. Analysis techniques include mathematical models, simulations, as well as comparing against checklists and best practices documents. The process of iterative decomposition which is used to develop plans to satisfy specific requirements is briefly described in this chapter. The issues involved in the implementation phase are also discussed briefly.

2.6 Review Questions

1. What is the difference between functional and non-functional requirements placed upon a computer system?
2. What are the different types of non-functional requirements that need to be considered in the design of a computer system?
3. Discuss the difference in the nature of the requirements for computer systems which are used to (a) host a free search site on the Internet, (b) process credit card transactions by a bank, (c) maintain driver's license information by a government agency, and (d) control the operations of the Mars explorer.
4. Determine the average delays in a network switch which has an average arrival rate of 10,000 packets/s and there are 50 packets in the switch on the average.
5. A check clearing system requires three passes through a logging system, two passes through a validation system, and one pass through a matching system. If we want to design the system to handle 100 requests/s, what should the capacity of each of the individual component systems be?
6. Discuss the pros and cons of using the safety margin rule for determining the maximum throughput that can be sustained by a web site. Why does a flash crowd disrupt the principle of safety margins?
7. The rule of 3-sigma is only statistically valid for normal distributions. Many realistic distributions are not necessarily normal. Discuss whether it is appropriate to apply the 3-sigma rule under these conditions.
8. Project: Examine the computer network that is installed in your university. Perform a resiliency analysis of the network and a performance analysis using the queuing network model.
9. Discuss the relative thermal efficiency of using a large data center hosting 10,000 machines with 10 smaller data centers each hosting 1000 machines. How would the performance and resiliency characteristics of the two models differ?
10. Discuss the different stages of the planning process. For what type of computer systems is it appropriate to go through the entire planning process. Under which circumstances could you plan a system using a simpler process?

References

1. D. Menasce et al., Performance by Design: Computer Capacity Planning by Example, Prentice Hall, 2004.
2. D. Berteskas and R. Gallager, Data Networks (2nd Edition), Prentice Hall, 1992.
3. R. Jain, The Art of Computer Systems Performance Analysis, Wiley 1991.
4. E. Lazowska, J. Zohorjan, S. Graham and K. Sevcik, Comptuer Systems Analysis using Network Models, Prentice Hall, 1984. Available at http://www.cs.washington.edu/homes/lazowska/qsp/Contents.pdf.
5. D. Ferrari, Computer Systems Performance Analysis, Prentice Hall, 1978.
6. D. Gross and C. Harris, Fundamentals of Queueing Theory, Wiley, 1998.
7. L. Kleinrock, Queueing Systems. Volume 1: Theory, Wiley, 1975.
8. S. Bose, An Introduction to Queueing Systems (Network and Systems Management), Springer, 2001.
9. M. Slooman, Reliability of Computer Systems & Networks, Wiley Interscience, 2001.
10. M. Xie, K. Poh and Y. Dai, Computing System Reliability: Models and Analysis, Springer, 2004.
11. H. Kumamoto and E. Henley, Probabilistic Risk Assessment and Management for Engineers and Scientists, Wiley-IEEE Press, 2000.
12. L.R. Ford, and D.R. Fulkerson, Flows in Networks. Princeton, NJ: Princeton University Press, Princeton, NJ, 1962.
13. R. Tarjan, Data Structures and Network Algorithms, Society of Industrial Mathematics, 1987.
14. D. Butler, An Importance Ranking for System Components Based upon Cuts, Operations Research, 25(5): 874–879, September – October, 1977).
15. T. Pham, Handbook of Reliability Engineering, Springer-Verlag, 2003.
16. F. Hwang, A hierarchy of importance indices, IEEE Transactions on Reliability, 54(1): 169–172, March 2005.
17. M. Beitelmal and C. Patel, Thermo-Fluids Provisioning of a High Performance High density data center, Hewlett Packard Technical Report HP-2004-146, September 2004.
18. C. Patel, R. Sharma, C. Bash, and A. Beitelmal, Thermal Considerations in Cooling Large Scale High Compute Density Data Centers, ITherm 2002 – Eighth Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems, San Diego, CA, May 2002.
19. T. Maniwa, Soaring Power Consumption Drives Thermal Analysis Tools, Electronic Design, September 2002, http://electronicdesign.com/Articles/Index.cfm?ArticleID=2677
20. R. Simmons, Using a simple air recirculation model to explore computer rack cooling, Electronics Cooling Magazine, February 2007. http://www.electronics-cooling.com/articles/2007/feb/cc/
21. R. Simmons, Estimating the effect of intercoolers for computer rack cooling, Electronics Cooling Magazine, May 2007. http://www.electronics-cooling.com/articles/2007/may/cc/
22. R. Cahn, Wide Area Network Design: Concepts & Tools for Optimization, Morgan Kaufmann, 1998.
23. R. Snevely, Enterprise Data Center Design and Methodology, Sun BluePrints, Prentice Hall, 2002.
24. T. Kenyon, High Performance Data Network Design, IDC Technologies, 2001.
25. T. Eilam et al., Managing the Configuration Complexity of Distributed Applications in Internet Data Centers, IEEE Communications Magazine, March 2006, pp. 166–177.
26. J. Marchewka, Information Technology Project Management, Wiley, 2006.
27. K. Schwalbe, Information Technology Project Management, Course Technology, 2007.

Chapter 3
Operations Management

The operational life cycle is the longest life stage in the life cycle of a computer system and also the most important from the perspective of systems management. The job of systems management is to keep the system running smoothly without problems during this life stage.

Operations management is generally done by a small team of systems administrators, often subdivided into expertise areas. For any but the smallest networks or IT infrastructures, one will find an identifiable operation center which is responsible for ensuring smooth operations of the entire IT infrastructure (see Fig. 3.1). In order to perform their functions, operations center staff need access to some information from the infrastructure being managed – this information is called management data. Operations center staff may also issue commands to the information center to change its configuration or retrieve portions of management data.

Management applications are software components used by the systems administrator to assist them in their functions. The algorithms implemented in the management applications or techniques used by the operations center staff are the focus of this and the next few chapters of the text.

In the first section of this chapter, we look at the organization of a typical operation center, followed by a discussion of the types of management data. Section 3.3 describes the typical protocols used for acquiring the management data from devices to the operations center. The next section describes the structure of the management data. Section 3.6 discusses the structure of the management application that runs at the operations center. Section 3.7 is a discussion of the functions that the management applications need to perform at the operations center.

3.1 Operations Center

The operations center can be viewed as the main point for control and command of a distributed system environment. The operations center is usually a dedicated physical location from where systems and network administrators can oversee the operations of the overall system. The staff at the operations

D.C. Verma, *Principles of Computer Systems and Network Management*,
DOI 10.1007/978-0-387-89009-8_3,

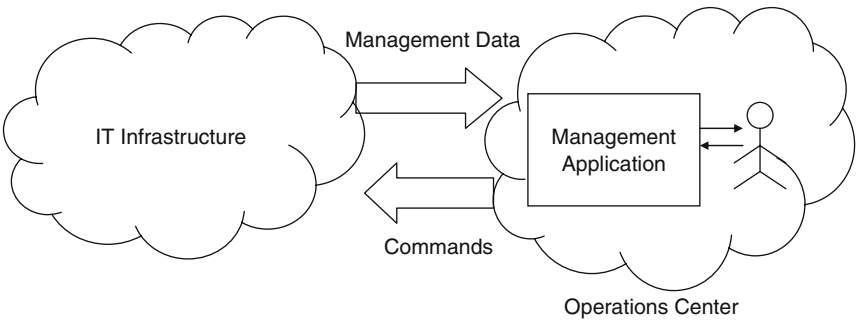


Fig. 3.1 Operations center

center are the administrators of the overall IT infrastructure of the company. The information collected by the operations center staff from the IT infrastructure to assist them in their functions is management data. The complexity and setup of the operations center vary depending on the size and scope of the operations.

An operation center would usually consist of a large number of screens, each screen displaying management data related to one or more components of the IT infrastructure. Information on the screens shows the overall status of the system operation at a glance, and any problems associated with network operations are displayed visually. Each identified problem or issue that can affect operations is assigned to one or more of the staff at the operations center, who will have their own computing infrastructure and management software enabling them to identify and correct the problem.

There are many companies that are exclusively focused on providing the network connectivity to their customers. Their operations center focuses on network management and is called network operations center (NoC). A flavor of the complexity of a network operations center can be seen from the history of AT&T network management operations [1] as well as other network operations center of several universities who have put their information publicly on the web.

In an enterprise, the IT infrastructure consists of client devices, the network, servers, and storage devices. The operations center is responsible for smooth functioning of every element of the infrastructure. For a large enterprise, each of the individual components itself may require an operations center. Thus, the operations center may itself consist of a server operation center, a network operations center, a desktop operations center, etc. Some of the most critical applications in the enterprise may have their own operations center as well. And it is possible to have a further subdivision of responsibilities. In a large telecommunications company, there are likely to be different operations centers for managing the optical network infrastructure, the MPLS infrastructure, and the IP layer connectivity, each operation center looking at a different layer of the networking stack. Such a structure is shown in Fig. 3.2.

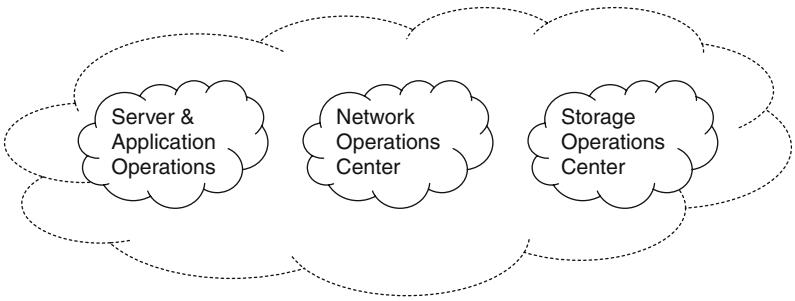


Fig. 3.2 Distributed operations center

It is not required that the different operations centers of an enterprise be in the same physical location. It is not unusual for operations centers looking at different aspects of the IT infrastructure to be located in different geographical locations. Each operations center looks at their part of the IT infrastructure. The specialization of operations center allows the growth of expertise to ensure smooth operations for one focused part of the IT infrastructure. However, when a problem or issue arises which requires cooperation across more than one operations center, adequate provisions need to be made to make information from one operations center visible to information from another operations center.

One key element of any operations center is the customer help desk. The help desk is usually its own separate operations center and is the initial point of contact for the users of the IT infrastructure who may have encountered any type of difficulty when using the computer systems. Some of the problems can be resolved by the staff manning the phones at the help desk (e.g., resetting forgotten passwords), while other problems (e.g., a system being down) require attention from the specialized operations center where experts can look at the problem and diagnose them. The effectiveness of management in any company can be judged by the number of calls coming into the help desk. A better managed network will have fewer problems being reported into the data center. The only exception will be the case when no calls are coming into the help desk, which probably indicates that the help desk itself is suffering from some kind of outage.

At the present time, most large companies have a global presence and their IT infrastructure is spread throughout the world. Many companies have taken advantage of the global workforce to create multiple operations center for the same function of their IT infrastructure. Thus, a company may have three server operations center, one located in the United States, the other located in Europe, and a third one located in India. As the Sun moves around the globe, the responsibility for managing the operations of the server shifts among the staff of the different operations center to provide a continuous set of operations.

The operations center in a large company can have a fairly complex infrastructure. In the next section, we look at the functions that the staff at the operations center need to provide.

3.2 Management Data

Acquiring the data required for the management of the computer system is a pre-requisite for doing any type of management function. A computer system can contain many different types of devices and a diverse set of software packages. Obtaining the management information from such a diverse set of items, most manufactured and developed by independent companies, is a challenge without some coordination and agreed-upon set of standards. Fortunately, a set of good standards exist which simplify the task of acquiring the management data in a consistent manner.

All management standards that are in prevalent use today use an agent–manager structure illustrated in Fig. 3.3. A managed entity is a device or application that is being managed. In this structure, each managed entity in the computer system would have a management agent which either comes pre-installed from the manufacturer of the managed entity or can be installed on the managed entity during the implementation phase. The agent communicates with a manager, which is a software instance running at the operations center. The agents collect any management data from the local managed entity and transmit it to the manager using a common protocol that both the agent and the manager can understand. The manager collects the management data from all the agents and keeps it in a repository to perform different management functions and operations.

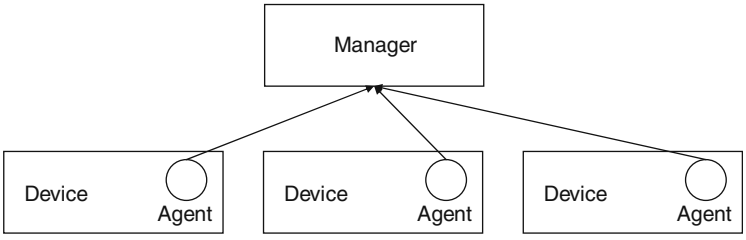


Fig. 3.3 Manager agent structure

When a computer system is large, it is not possible for a single manager to communicate with all the agents on all the devices within the network. For scaling management operations to large systems, a hierarchy of managers is used. The hierarchy consists of managers at different levels, with the level 0 managers talking to agents on the different devices to obtain the management

data. The level 0 managers can consolidate the management data and pass it over to level 1 managers, and level 1 managers can pass it over to level 2 managers, and so on.

There are a few different options for how the hierarchy can be implemented. The level 0 managers can be implemented as simple pass-through proxies, in the sense that their function is simply to aggregate the information collected from the devices and pass it along as a consolidated digest of all the management information. It is also possible to have the managers at each level be responsible for a subset of management functions and only pass some metadata informing the higher level manager some metadata regarding the management data that it has collected. The management information is distributed among all the managers in the hierarchy and is provided to other managers when they need to use it for any reason.

The protocols used for manager–agent communications can also be used for inter-manager communication, but they would tend to be rather inefficient. As a result, proprietary protocols are often used between instances of managers that are provided by the same company. With the recent increase in acceptance of web services, they provide an alternative means for communication among different instances of managers.

Let us illustrate the challenges associated with acquiring management data in the general manager agent model by means of an example.

Let us consider the case of an organization which runs multiple web sites, some of which are based on Apache and some of which are based on Microsoft IIS. The organization runs the Apache-based web sites on three different types of servers (x86-based Linux servers, IBM AIX servers, and Sun Solaris machines) while the IIS is run on the Windows-based platform only. The management system monitors the logs generated by the two applications and is required to produce monthly aggregate performance reports regarding the web sites. For the purposes of illustration, let us assume that the servers, applications, and network devices needed to run the web sites do not experience any downtime, crashes, or errors during the period and the management data are simply the performance statistics that need to be compiled into a monthly performance report.

The management system is a software package running at the network operations center. The agent can be an instance of the software running on the servers hosting the web site application. The agent can keep track of the logs generated on each server for a limited amount of period (e.g., 6 h) and then uploads the logs generated to the management system, which processes and stores the data for a month to compile the monthly report.

The agent and the management system need to have some protocol to get the management data from the agent to the management device. The protocol could be that the management system does a remote log in into the server and does an `ftp` of the log files over, or it could be the agent compressing the logs for the last 6 h into a compact version and mailing it over, or it could be the

agent doing a preprocessing of the information and sending the processed results for the last 6 h over to the management system.

Another issue deals with the format of the management data that each of the software packages produces. Microsoft IIS by default will generate a log file with each line containing comma-separated entries of the format:

```
Client IP Address, User Name, Date, Time, Service and
Instance, Server Name, Server IP, Time Taken, Client
Bytes Sent, Server Bytes Sent, Service Status Code,
Windows Status Code, Request Type, Target of Opera-
tion, Parameters
```

Apache by default will generate a log file with each line containing space-separated entries of the format:

```
Client IP Address, Client Identity per RFC 1413, User
Identity, Time and Data, Request Line from Host, Status
Code, Size of Entries
```

While both web server softwares can be configured to generate different types of log formats, the actual log generated by both software instances differs in many respects, e.g., one uses a comma-separated text, the other uses a space-separated text; data and time are separate in one format, while contained as a single field in the other. There are some additional information fields in IIS log, and some fields like the RFC 1413 client identity are not very reliable or useful in the final management report.

In order to manage the web sites, the organization needs four flavors of the agent (one for each combination of Apache/Linux, Apache/Sun, Apache/AIX, and IIS/Windows) and the ability to process two different formats of the logs. And this is just for the simple toy example we have selected here. In a real organization, there are about a hundred applications that need to be managed, and instead of just 2 possible software packages that implement an application, there are 5–10 different software packages that can be used for each application, and the applications run on servers, machines, and other systems that are obtained from many different vendors.

The organization would have an easier time of managing the information if all server platforms came pre-installed with an agent that monitored the performance of the servers and sent it to the management system using a standard protocol or convention. The other item that could simplify the management would be if the management data from the different applications and servers were produced in a common format.

One could argue that the management problem can be simplified if the organization could simply enforce all of its web sites to be based on a single software package of a single platform. Unfortunately, it is very hard to maintain and enforce a homogenous environment in any large organization. The organization does not want to become dependent on a single supplier. Sometimes, one software package has some unique features that

make the development of the specific web site more effective. An attractive financial package offered by suppliers of a system may make it cheaper to install a new instance of the web sites. In general, uniformity in IT infrastructure is a Utopia all organizations seek, but rarely are able to achieve.

If there were no widely accepted standards, then the management system would have to develop its own conventions for the protocol as well as the management data format.

Some protocols have become common standards for communications between the agents and the manager. These include Simple Network Management Protocol (SNMP), the Common Object Repository Broker Architecture (CORBA), and Web-Based Enterprise Management (WBEM). Some conventions for representing management information include the management information base (MIB) format commonly used with SNMP, and the common information model (CIM) for server and desktop management.

3.3 Manager Agent Protocols

In the next few sections, we survey some of the commonly used protocols for manager–agent and inter-manager communications. These include SNMP, CORBA, WBEM, and web services.

3.3.1 Remote Consoles

Each of the components in a computer system offers a way for a user sitting on a console in front of the component to enter management commands. The management commands include the ability to change the component's configuration parameter, as well as the ability to read any counters, statistics, and status indicators of the component.

In order to manage any device, any technique that allows the same commands to be executed remotely and in an automated manner from the manager will suffice as a protocol to access that device. In most network devices, the command line instructions (CLI) can be accessed by initiating a telnet session to the device. Many operating systems provide an ability for a user to log in remotely to the system. On variants of the Unix operating system, two such programs are the remote shell (rsh) which provides the ability to get a console window on a Unix machine remotely and rlogin which allows the invocation of commands on the Unix machine from another system – remotely log in onto the machine, executing the command, and relaying the results back to the requesting system.

Such mechanisms essentially enable the console of the device to be accessible to a remote machine. Each device has its own set of configuration commands,

and a manager needs to support the characteristics of each device. Nevertheless, such a remote console may be the only method of accessing and managing many devices.

3.3.2 Simple Network Management Protocol (SNMP)

SNMP is a management protocol commonly used for monitoring network devices and networking support on end client platforms. It is a standard protocol defined by the Internet Engineering Task Force (IETF). It is an application level protocol which has implementations running over both UDP and TCP, the two most commonly used transport protocols in the Internet.

The operations in SNMP protocol are closely tied to the representation of management data in the MIB format (discussed in greater detail in Section 3.3.2). Management data in MIBs are represented as a hierarchy of management information entry, and each entry has a unique identifier.

The SNMP protocol has been defined in three versions:

SNMPv1: The first version of SNMP was standardized in 1990 and supported four basic operations: Get, GetNext, Set, and Trap. The Get operation is used to read the value of a MIB entry by specifying its identifier, and the Getnext command could get the next entry on from a table of entries. The Set command was used to assign values to a specific MIB entry, and Trap command was used by agents to send a notification to an agent, e.g., a manager can use the Set command to establish a threshold in a MIB entry, and the agent can send a trap to the manager when a counter exceeded that threshold.

SNMP v1 used a simple request response model and each Get operation could effectively read a single entry, leading to a rather chatty protocol for reading the entire management information. SNMP v1 did not make any provisions for security, leading to most administrators using it only for monitoring of MIB variables and not for setting any configuration parameters.

SNMP v2: The second version of SNMP standardized in 1996 added two new operations to SNMP v1. The first operation was a GetBulk which could read more data efficiently from multiple rows in a MIB table. The second operation added was Inform intended for manager to manager communication. Inform can be viewed as an analogue of the trap command for inter-manager communication.

Although SNMP v2 was more efficient than SNMP, it suffered from the same security restrictions and thus its role was confined to simply reading of monitoring information from the MIBs.

SNMP v3: SNMP v3 added security to the earlier versions of SNMP. It provided for authentication of SNMP agents and managers, encryption of the communication among the agents and managers, as well as supporting access

control rules defining which entity can modify MIB entries. Despite the increased security of SNMPv3, SNMP remains largely a monitoring protocol for information.

A more detailed description of SNMP, including the format of its messages and their encoding, can be found in books such as [2–4].

As the same time as SNMP, another protocol CMIP was developed by the OSI for the purpose of network management. CMIP did not gain much acceptance and is of historical interest only. It is being mentioned just for the sake of completeness.

3.3.3 Common Object Repository Broker Architecture (CORBA)

While SNMP was the dominant protocol for monitoring management data for networking where management data were defined in structured MIBs, it was relatively cumbersome to use for retrieving management information for applications and systems. When building the management protocols between their managers and agents, many systems management vendors turned to CORBA to be able to exchange more complicated monitoring information more efficiently.

CORBA is a protocol developed for general-purpose distributed computing and not specifically targeted at management. It provides the ability to define software objects (the type of software entities that one defines in object-oriented programming [xxx]) whose interfaces can be invoked by other software objects running on a different machine. When a piece of software tries to invoke an object on a remote machine, an intervening object repository broker (ORB) acts as an intermediary and finds the object which is the target of the invocation, passing parameters between the invoker and returning the results. Objects that are being invoked define their interfaces using an interface definition language (IDL) defined by the CORBA standards.

Using CORBA as the management protocol, the agents on devices being managed can define a set of interfaces they want to expose to the manager for it to retrieve any monitoring information or to set any configuration parameters. The manager can then invoke the right interfaces to perform that operation on the desired agent.

Unlike SNMP, the agents can describe arbitrary objects with many different attributes and structures and access them efficiently in a single call. The CORBA architecture also allowed for authentication. Furthermore, any vendor of management software could define their own objects with rich functionality as required for their agents, and thus implement sophisticated management functions easily. In the 1990s, CORBA-based management was the prevailing mechanism for most enterprise servers and applications management.

Two industry trends have impacted CORBA-based management adoption in an adverse way. The growth of the Internet and associated security threats

resulted in many enterprises putting firewalls at strategic locations in their enterprise. CORBA was not designed to cross firewalls, and one could not just open a few select ports on the firewall to enable CORBA communication. This created a significant problem for CORBA-based management applications. Second, CORBA as a paradigm for developing distributed middleware was supplanted by newer techniques based on service-oriented architecture (SOA).

3.3.4 Web-Based Enterprise Management (WBEM)

The growth of the Internet has caused virtual universal support of the HTTP protocol (the base protocol for web browsing) in almost all types of machines, and all firewalls are configured or can be configured to allow access through the HTTP protocol. WBEM is a set of specifications by DMTF [5] which define a protocol that exploits HTTP to provide management commands between the manager and the agent.

WBEM assumes that the management information is described by means of a CIM specification (see Section 3.3.3 for details). The management information and the operation (reading or changing the management information) are encoded in an XML [6] representation called CIM-XML and then transported over the HTTP protocol. Each agent implements an HTTP server to receive such requests, an engine to decipher and respond to the CIM-XML requests, and internal software that can collect management information from the local device and represent it using the CIM model specification. Each manager needs to implement an HTTP client to access the agent and libraries to encode requests onto a CIM-XML message and to decipher the response.

WBEM enables operations on management operations that include basic reading and writing of management information, plus creating new instances of resources identified in the CIM model. The schema representing the management information structure can also be manipulated directly using the operations provided by WBEM.

WBEM is supported in many operating systems including Linux and Windows.

3.3.5 Web Services

Web services can be viewed as a way to provide remote invocation of functions using the HTTP protocol in conjunction with other application formats and specifications – or a mechanism to provide CORBA-type functionality over the http protocol. A web service provides a way for a client running on machine to invoke a server that supports a set of interfaces that are exposed and defined

using a web service definition language (WSDL). In its most common format, web services use SOAP (a simple request response protocol) running over HTTP with XML-encoded messages to send requests and receive responses.

A web service can be used to provide an effective communication protocol to any application that can publish a WSDL. It can be used for communication between a manager and an agent, as well as for communication between two managers. Web services offer the flexibility of defining an extensible interface, while riding on top of standard protocols like XML and HTTP that are almost universally supported.

An extension to web services known as Web Services Distributed Management (WSDM) provides base capabilities for managers to talk to agents using a web service framework. Although no formal standards have been defined for using web service as a protocol among managers, its flexibility makes it suitable for such usage.

A web service interface also provides an effective mechanism for communication among peer managers – managers who are not in a hierarchical relationship with each other. As mentioned earlier, the operations center in a large enterprise may be distributed along different domains. A web service interface provides an effective means for communication and exchange of management data among different instances of a manager.

Drawbacks of the web services approach for management-oriented communications would be its complexity and performance, along with a lack of progress in defining web services security standards. Despite its weaknesses, web services may become an important player in the management domain given the industry momentum behind the technology.

3.3.6 NetConf

NetConf is a standard protocol defined by IETF in 2006 for the explicit purpose of remote configuration of network devices. It is a set of XML-based configuration commands that can be invoked remotely using an underlying protocol such as SSH.

Netconf is essentially a method to invoke remote procedure calls on configuration information stored at the managed device. The netconf protocol provides for reading, modifying, and deleting the configuration information and for reporting the results of the operation to the manager.

The protocol specification does not put any constraints on the configuration information structure, so the primary advantage of the protocol is in providing the manager with a common way to invoke operations on the configuration files at the device. Each manager needs to understand the differences and nuances between the configuration information on the different types of devices.

3.3.7 Comparison of the Different Management Protocols

If we look at the variety of the protocols that are used between the agents and the manager, two observations can be readily made:

1. The management systems are opportunistic and make use of any transport protocol that is readily available for that domain.
2. Many of the protocols used in network management are general-purpose distributed computing protocols and not specially designed for management functions.

These two are reflective of the fact that the actual transport protocol between the agent and managers plays just a supporting role in the task of systems and network management. While one can find enthusiastic supporters of each of the protocols listed above, each management system will support any protocol that enables them to get the management data from the managed device's agents. The adoption of a common standard protocol simplifies the task of the manager and is valuable in that respect, but any assertion that a specific protocol is superior to others is likely to be counterproductive.

The comparison table (Table 3.1) should be read with that perspective in mind and with the realization that most management systems would support several of the protocols listed below to enable them to manage the maximum number of devices.

Table 3.1 Comparison of management protocols

	Remote console	SNMP	CORBA	WBEM	Web services	NetConf
Primary domain	All devices	Networking	Servers and applications	Servers and applications	Servers	Networking
Monitoring support	Yes	Yes	Yes	Yes	Yes	No
Configuration support	Yes	No	Yes	No	Yes	Yes
Security	Password	Yes in v3, none in v1 and v2	Yes	Yes, http security	Yet to be defined	Yes
Data restrictions	None	MIB representation	None	CIM representation	None	None
Firewall traversal	Easy	Feasible but insecure	Poor	Good	Good	Good

3.4 Management Information Structure

The management protocols provide an ability to move the management data from the agents to the manager. It is important that the manager be able to interpret, parse, and analyze the information contained in the management data. The challenges in managing systems arise from a large extent due to the fact that there are many types of devices, and they all generate management data in different formats and in different manner.

3.4.1 Management Information Base

The SNMP protocol described in Section 3.2.3 assumes that management data that are being retrieved by it are represented as a management information base or MIB. Each MIB is a collection of managed objects. Each managed object has an object identifier, a set of attributes, and/or definitions that define the structure of the management data available at the managed entities. They provide an abstraction of the format in which management data are visible externally and need not necessarily correspond to the manner in which managed data are stored at the managed entity. One of the primary functions of the SNMP agent in any managed entity is to map the MIB-defined abstraction of the management data to the manner in which the data are stored locally.

Each management object has an object identifier which is a unique hierarchically structured name. The object identifier in SNMP has a structure of integers that are separated by periods, e.g., an object identifier might look like 1.3.6.1.2.1.6.5. Each of the integers in the period identifies a hierarchy under which the name is assigned. The first digit in the object identifier states the administrator of the node, 1 is used to indicate ISO (http://iso.org), 2 is used to represent ITU-T (formerly known as CCITT) (http://www.itu.int/ITU-T/index.html), and 3 for a managed object jointly administered by ISO and ITU-T. The second integer identifies a group under the organization identified by the first digit. ISO uses 3 to identify other (non-ISO) organizations. The third integer identifies a group under the one identified by the first two digits and so on; 1.3.6 identifies the US Department of Defense; and 1.3.6.1 identifies the Internet Engineering Task Force. All MIB OIDs will normally begin with 1.3.6.1.

The fifth integer (after 1.3.6.1) is 1 if the group is for the OSI directory, 2 if the group relates to network management (which all standard MIBs do), 3 if it is an experimental identifier, and 4 for private. All standard MIBs would normally begin with 1.3.6.1.2.

Under the private subgroup, group 1 is used for enterprise. Any company may define its own proprietary MIBs under the OID tree 1.3.6.1.4.1.xxx where xxx is the identifier associated with that company. A list of such identifiers is maintained by Internet Assigned Numbers Authority (IANA) and is available at URL http://www.iana.org/assignments/enterprise-numbers.

This naming structure allows for naming managed objects by following a hierarchical structure and allows for extensions of new managed objects by specific technology areas in Internet standards, as well as new management objects by different organizations. The MIBs defined by organizations using the 1.3.6.1.4.1 prefix in name are known as enterprise MIBs.

The definition of any management object is done by means of rules called Structure of Management Information (SMI). SMI provides the guidelines using which new MIBs can be defined. As mentioned previously, a MIB definition is a collection of management objects. According to SMI, each management object is defined using the typical format shown below:

```
objectName OBJECT-TYPE
        SYNTAX       syntax
        MAX-ACCESS   max-ccess
        STATUS       status
        DESCRIPTION description
        ::={group #}
```

In the above format, the entries that are underlined are items that need to be specified for each new management object definition, and the text in all capitals are keywords marking specific aspects of the definition. The {group, #} after the assignment operator (::=) is the OID of the managed object, where group refers to the name of an existing management object which is the group to which this managed object belongs, and the # is the specific extension assigned to this managed object. The OID of the new object is the OID of the group field followed by a period and the # field.

The objectName is the name of the new managed object. The keyword OBJECT-TYPE denotes that the description is that of a managed object. The SNTAX keyword is used to identify the type of managed object. A managed object could be an integer, a string, a counter, the number of time ticks, and a few other types that are defined in the standards [RFC 2578]. The syntax field enumerates which syntax is to be used for this management object. The ACCESS keyword is used to identify the type of access users have to this object. Legal values for access field include read_only, read_write, not_accessible, read_create, and accessible_for_notify. The STATUS keyword is used to specify if the object is current, obsolete, or deprecated and allows backward compatibility of OIDs as MIB definitions evolve over time. The DESCRIPTION value is a text description of the managed object.

A management object may also contain other optional fields specifying the units used to measure it, a reference to a document with more details, a default value, and a few other fields. More details can be found in the IETF RFC 2578.

A management object may also define conceptual tables in its MIB definitions. Tables are conceptual in that they are simply sequences of management objects, but can also be indexed using a row number or a table number.

The keyword OBJECT-TYPE is an instance of an ASN.1 macro. ASN.1 is a ITU-T standard for data representation and the SMI conventions are based on ASN.1 specifications. In addition to the OBJECT-TYPE macro, a MIB may also contain macros of MODULE-IDENTITY and NOTIFICATION-TYPE. The module identity provides information about the OID of the group and other information like the owning organization and contact information. Another macro lists the type of notifications that can be sent by the agent supporting the MIB. A MIB can also contain declaration of short-names for groups and OIDs, information about any other previous MIB definition it depends on using (done using an IMPORTS clause), and is bracketed within a BEGIN and END clause. Thus, a MIB definition would look like the following:

```
MY-EXAMPLE-MIB DEFINITIONS ::=BEGIN
    IMPORTS
        Definition1 FROM prior-mib-definition
        Definition2 FROM prior-mib-definition-2
    MyExample MODUE_IDENTITY ::= {Definition1, 5}
        oid_name OBJECT IDENTIFIER ::== {group, #}
         ...
        object OBJECT TYPE :== {::== {group, #}
     END
```

The MIB model is very simple but has proven to be a very effective tool for providing management data to many different networking technologies. A few MIB definitions are standardized by the IETF, but several more MIBs are available as enterprise MIBs from the manufacturers of different types of networking gear.

Further details on MIBs and their usage can be found in the IETF RFCs [7–9] and in books [1–5, 10].

3.4.2 Common Information Model

The common information model is a specification from the Distributed Management Task Force. An information model is a representation of the structure of the information contained in the management data. Unlike the MIB model, which uses a simple collection of management objects to make up the management data, the CIM model leverages techniques from object-oriented design to have a richer representation of management data.

In essence, CIM defines a schema describing the structure of management data. A schema is a description of the different classes that can be present in the management data and the relationship between these classes. The schemas are represented visually using UML (unified modeling language) [11] and for software processing using a textual representation called MOF (managed object format).

CIM is structured into these distinct layers:

- Core model – a set of base classes that are applicable to all areas of management.
- Common model – a set of base classes that are common to particular management areas, such as systems, applications, networks, and devices. The set of classes here define the management information for these areas and are a set of base classes that can be extended for specific technologies.
- Extension schemas – technology-specific extensions of the common model that are specific to environments, such as operating systems (for example, an extension for UNIX or mainframe environment).

All CIM classes are named with a prefix of CIM, but the prefix is usually omitted in conventional usage. We will follow the same convention in this book.

The root class in the information model is the class ManagedElement. All CIM classes are derived from this class. A managed element can have various relationships to other managed elements. Some of the relations include one managed element being a component of another managed element and one managed element having a dependency on another managed element. ManagedElement is an abstract class with its only attributes being its name and a description.

An immediate child of the ManagedElement is the ManagedSystemElement. This object represents any managed element that can have an operational status in the managed device. ManagedSystemElements may be either physical entities (e.g., interfaces, adapters, servers, machines) or logical (e.g., a TCP connection). Correspondingly there are two children of ManagedSystemElement, a PhysicalElement and a LogicalElement.

Another child class of ManagedElement is used to describe the set of statistical data associated with the managed element. This is further subclassed to provide some common types of statistical information, and technology-specific extensions can provide further subclasses.

Figure 3.4 shows a view of a few classes in the CIM model, including the system class and the network class, which are from the common model. The relationship between different classes is shown as an example, and the set of schema definition where each class is defined is provided. The details of the attributes of each class are not defined, but can be found in the details specifications available at the DMTF web site.

Thus, by following an object-oriented paradigm, CIM provides a mechanism to express all management data in a consistent format. A more detailed description of the CIM object models, including the description of the various classes and relationships among them, can be downloaded from the DMTF web site, http://www.dmtf.org.

3.4.3 Issues with Standard Representation

While the emergence of standards like CIM is a step in the right direction for simplifying the complexity of management data, several management problems remain even with the definition of the various standards. Some of these problems are listed below:

Incomplete Coverage: The standards exist only for a subset of the management data. There are no widely adopted standards for management data in applications. Even in the networking area where the MIB standards are well established, MIBs are available only for a subset of all the management information that is needed. Each equipment manufacturer can provide

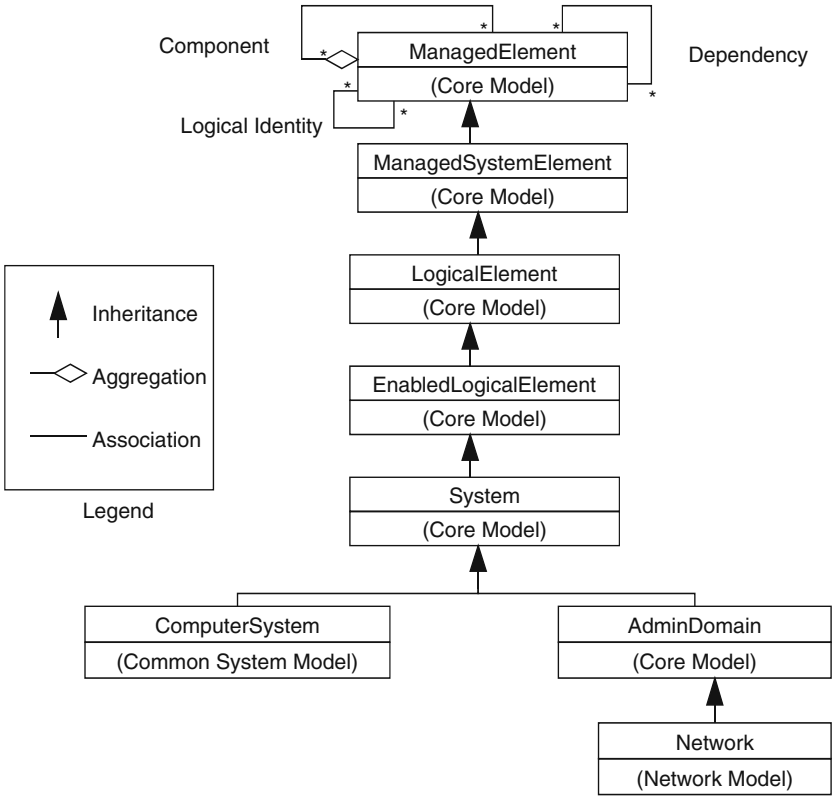


Fig. 3.4 A subset of CIM class hierarchy

extensions to the MIBs for features that are supported in their devices. These enterprise MIBs are rarely compatible with each other. Similarly, enterprise extensions to CIM model are rarely compatible with each other.

Slow Pace of Standards: The computer industry is an industry with a very high degree of innovation, and new and better hardware and software technologies are introduced every year. The standard bodies are slow to define new management (MIBs or CIM models) for such technologies.

Inefficiency and Complexity of Standards: The standards are often inefficient compared to alternative modes for managing a technology. The CIM model has a fair bit of complexity built into it. The SNMP model is relatively inefficient in getting a large amount of management data. As a result, proprietary protocols may often provide a better way to get management data from the device to the management system.

MIB and Schema Explosion: Although a MIB or a CIM schema provides a standard way to represent information, there are too many MIBs and too many schemas. The management of the MIBs and schema classes themselves poses a challenge to a management system.

Despite these limitations of standards, virtually everyone is better off with having the standards in place rather than not having any standards at all. With a standard model and protocol, a management system needs to worry only about a single way to access management information from a large number of devices. The manufacturers of devices are able to present their management information in a consistent format and make their devices more manageable. Furthermore, the users of the computer systems who use the device find it easier to understand and manage those devices.

3.5 Device Agent Structure

On each managed device in the computer system, one needs an agent which would be able to collect operations and perform configuration operations on behalf of the management system. The device agent may be supporting a standard or it may be a special agent which is coded for the purpose of a custom-built management application. In this section, we look at the software structure of the device agent.

Any device agent has two functions to fulfill: (a) it needs to communicate with the management system on the other end and (b) it needs to have an interface to the local device so that it can retrieve required information and invoke commands on behalf of the management system. Accordingly, most device agents tend to have a structure depicted in Fig. 3.5.

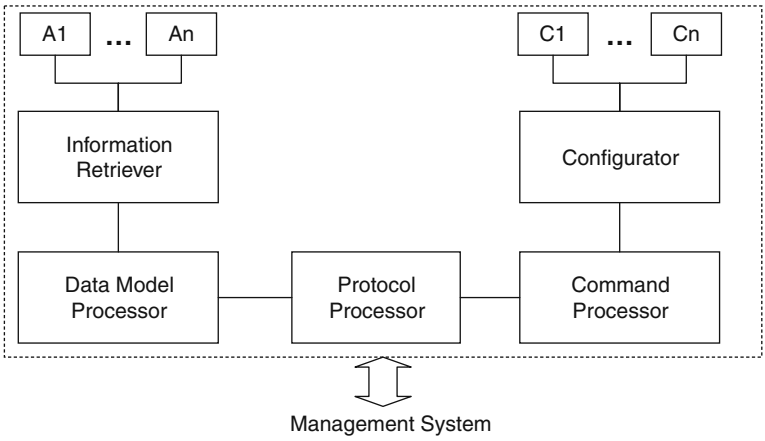


Fig. 3.5 Structure of a device agent

As shown in the figure, each device agent would have an interface with the management system on which it would communicate with the protocol required between the management system and the agent. The protocol processor interface is a software module that parses the message coming from the management

system and converts any outgoing messages to conform to the protocol on the wire. A protocol message may require either information to be read from the device or modify some element on the device. The protocol processor can pass the information either to the data model processor or the command processor. The data model processor is responsible for converting information present in the system to the format required by the protocol on the wire. The command processor is responsible for interpreting any modification or configuration changes requested by the management system.

The data model processor obtains local information from the information retriever, a software module which collets information from different adapters. Each adapter is responsible for getting information from a specific component on the system, e.g., on a computer system, one adapter may obtain information about TCP connection statistics while another component obtains information regarding the file system statistics. On the command and processing side, one would see a similar adapter structure. The configurator is a library module which determines which component needs to be changed or modified and then invokes the right command using an adapter to communicate with the appropriate system component. The adapters are also known as subagents in several implementations.

In this hypothetical design, we have shown the configuration/modification components as different from the information collection components. In practice, one may combine these so that a single adapter performs both the configuration and the modification for a specific system component. Nevertheless, there is value in treating these components as separate, since many management agents are designed only for reading information, while others are designed explicitly to manage the configuration of the different components.

3.6 Management Application Structure

Having looked at the structure of the management agents on the device side, let us now look at the structure of the management system. The management system can consist of more than one computer systems, depending on the size and scale of the system being managed. In some cases, a single management application (e.g., a fault monitoring system) may require to be run on several machines.

The structure of a typical management system is shown in Fig. 3.6. The information collected from the various agents is stored into a database shown as raw management data. Before storing into the raw management data, the system would need to convert the data into a canonical format that is often represented by the schema of the database storing the raw management data. The schema for the raw data is selected by the installation of the system, since different types of agents produce different types of management data.

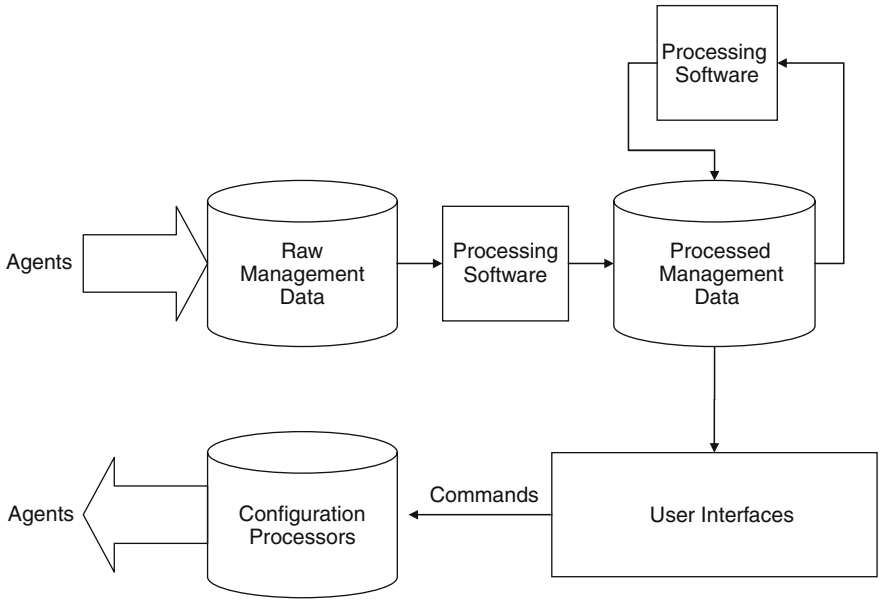


Fig. 3.6 Structure of the management system

The raw data need to go through pre-filtering and other types of processing in order to weed out redundant data. The raw management data are then stored into the database shown as the processed management data. In some management systems, the raw data may not be explicitly stored in a database, but instead processed as it arrives and then stored in the processed management database.

The information in the processed management database may be passed through several other processing modules. All the processing modules look at the information and manipulate it in a variety of ways. Some may correlate the information with source of information outside the management system, others may remove duplicates or consolidate some data (e.g., instead of keeping 10 different statistics collected about the traffic through a router, they may combine it into a time-averaged value). The number of processing modules and the nature of their task are dependent on the management functions being performed.

After passing through a series of processing functions, the data are displayed to the administrators at the operations center. The displayed data may provide status information on the health of the computer system, identify any area that needs attention, and point out any failure in the system. Most system operators would expect to see normal operations most of the time and would investigate any item in the visual display that shows a deviation from the expected mode of operation.

The operators may also choose to configure and modify the state of the computer system from the operations center. In that case, they issue the command using an user interface (which may or may not be the same user interface used for monitoring). The configuration or reconfiguration commands are used by the configuration processors, which are responsible for making the changes through the various agents in the devices. Configuration processing would include functions such as reordering the commands so as to do them in the optimal manner possible, convert simplified commands into more complex configuration scripts, or validate that the command is being issued to an active component in the computer system. In general, the command processors would need access to information from the processed management data in order to perform their tasks.

The type of functions that are implemented by the different processing elements is described briefly in the next section.

3.7 Operations Center Function

Management is traditionally classified into five functions abbreviated as FCAPS, which is an abbreviation for fault, configuration, accounting, performance, and security. The concept of these five functions of network management originates from various network management specifications provided by the International Telecommunications Union (ITU). While these specifications were targeted more at the telephone network rather than general systems management, the FCAPS paradigm provides a reasonable conceptual model to study the different aspects of general systems management.

The focus of fault management is on detecting and identifying the failures that occur in the network and taking corrective actions to work around those failures. The corrective action may be taken by an automated system or with human intervention. Configuration management deals with ensuring that different elements (hardware as well as software) that are required for the operation of the system are configured in an appropriate manner. Accounting management deals with maintaining charges and costs that are associated with the use of the system. Performance management involves ensuring that the system is operating with an acceptable level of response time. Security management deals with mechanisms to thwart hackers and ensuring that system and information access is restricted to duly authorized and authenticated users.

At each operations center, the FCAPS functions are applied to the part of the IT infrastructure that the operations center staff is responsible for. Each operation center is responsible for specific portions of the infrastructure, e.g., servers, networking, or a specific layer of networking, and the staff at the center need to have good operational knowledge of the appropriate domain and technology. Nevertheless, the basic techniques and algorithms used for each of the FCAPS

functions have a significant amount of commonality, and these common concepts form the basis of many of the subsequent chapters.

To provide any aspect of FCAPS management, the activity chain shown in Fig. 3.7 needs to be undertaken. The management activity chain consists of discovery, monitoring, and analysis, which may be followed by reporting or reconfiguring.

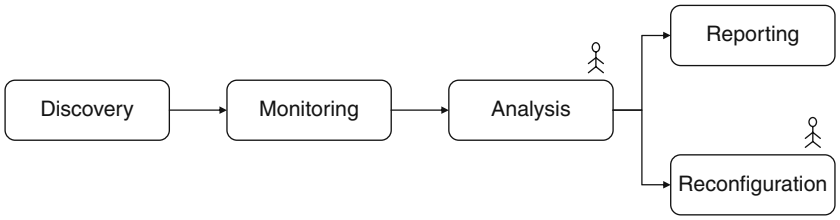


Fig. 3.7 Activity sequence for systems management

The first management activity required of the operations center staff is that they know what the infrastructure is and what its various constituents are. The operations staff cannot manage elements of the infrastructure whose existence they are not aware of. The first function at the operations center is the function of discovery of the elements of the infrastructure. The discovered elements in the infrastructure form the inventory of assets in the IT infrastructure. An asset is any element of the IT infrastructure. The assets that are found as a result of the discovery may be physical, e.g., networking gear, server boxes, and other equipment, or they may be virtual entities such as installed software packages and their configuration.

Monitoring consists of retrieving the right management data from the IT infrastructure. For each function in FCAP, a different part of management data may need to be monitored. Performance monitoring may require reading in counters of assorted types from different elements of the infrastructure, while fault monitoring may require processing reading error logs within the system.

Analysis consists of understanding the monitored data, processing it to reduce its volume, and determining the right reaction to any issue that arises from the analysis. The analysis step would be very specific to the FCAPS function being managed. For fault management, the analysis step would consist of identifying the root cause of the fault. For configuration management, the analysis step may require determining the differences between the observed and expected configurations of elements. For accounting, analysis will consist of determining which of the different users of the system are responsible for using different system resources. For performance management, analysis would mean combining monitored information from various sources to determine a consolidated set of performance metrics. For security management, analysis would be detection of any security threats or problems in the system.

The output from the analysis would either be put forward as a report to an external entity, e.g., accounting analysis may result in a report to a billing system to be processed further, or performance analysis may result in a periodic system performance result; or the output may result in a reconfiguration of the system, e.g., fault analysis may result in some elements of the system being reconfigured or performance analysis may result in changing some parameters of the system elements to improve the responsiveness of the system.

Under the current state of the art is network management; there is a significant amount of manual intervention that is required for the analysis and reconfiguration steps. Manual intervention is required in analysis to identify the root cause of failures or abrupt performance changes. Reconfiguration is also done manually. The other functions, discovery, monitoring, and reportings can be automated to a large extent with only a little bit of manual intervention such as setting preferences and options for the automated processes. The activities with significant human involvement are marked with stick figures in Fig. 3.7.

The tasks performed at any operations center can be categorized using three axes – an axis representing one of the FCAPS functions and the other axis representing the steps in the activity sequence. The third axis is the domain of management – which can be used for either the technology domain or set of assets over which the operations center staff have management authority. The network operations center would perform various tasks corresponding to the 5×5 matrix shown in Fig. 3.8 that is relevant to the networking technology area, and the server operations center would perform similar tasks to the 5×5 matrix except when applied to the domain of servers within their management control. Other types of management systems would have the analogous set of tasks applied to some other domain.

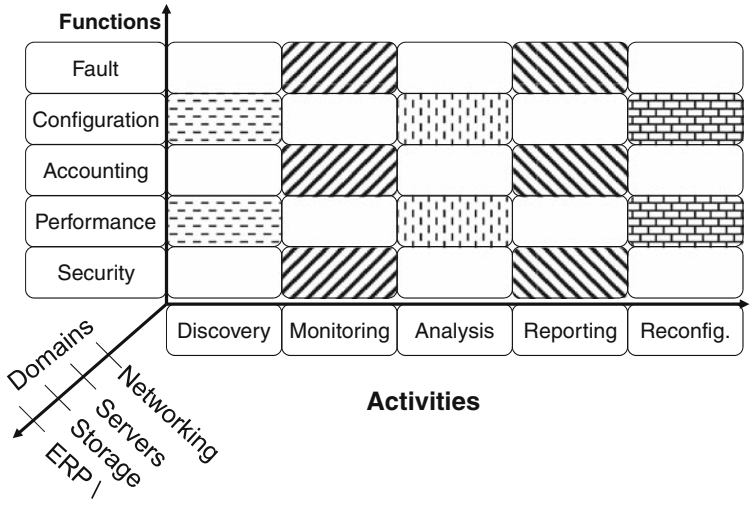


Fig. 3.8 The matrix of management tasks

The following few chapters of the book will discuss the different tasks in this matrix. We first discuss the two columns of discovery and monitoring, since there are several common elements in these two tasks across all the five areas of management. Subsequently, we discuss the functions across the columns in each of the areas since the analysis and reporting tasks are highly specialized to the specific function being performed.

3.8 Summary

This chapter introduces the concept of the operations center which is the site focused on systems and network management during the operation life cycle stage of the computer system. It discusses the protocols that are used and commonly available for obtaining management information from the different devices to the operations center. The structure of management information and the standards related to representation of management information are introduced and compared.

The chapter introduces the FCAPS model for management, which are the five primary management functions performed during the operations life cycle. The subsequent chapters in this book consider each of these functions in detail.

3.9 Review Questions

1. What is an operations center for systems management?
2. Discuss the pros versus cons for having a distributed operations center versus a centralized operations center for systems management?
3. Large companies often have operations centers that move with the Sun, e.g., different operations centers will manage the global computing infrastructure at different times. What are the benefits and the drawbacks of having such a system?
4. Distribution of operations center function can be done according to a geographic hierarchy or according to the different functions. Discuss the relative merits of each approach.
5. Discuss the relative strengths and weaknesses of SNMP, CORBA, and WEBM as a protocol for manager–agent communication.
6. Why is a change in versions of a system management protocol slow to gain acceptance in the marketplace?
7. Discuss the pros and cons of using a proprietary protocol for manager–agent communication versus using an existing standard protocol.
8. What conceptual extensions are provided by the CIM model that are not present in the SNMP MIB data.

9. SNMP has been very successful in providing network monitoring data. What are the challenges in extending SNMP and defining MIBs so that the SNMP becomes the existing standard for collecting management information for servers and software applications?
10. *Project:* Look at an open-source implementation of a management agent and compare it to the reference architecture described in this chapter. What are the similarities and differences between the implementation and the conceptual architecture. What may be the reasons for those differences?

References

1. AT&T, History of Network Management, available at URL http://www.corp.att.com/history/nethistory/management.html.
2. W. Stallings, SNMP, SNMPv2, SNMPv3 and RMON 1 and 2, Addison-Wesley, 1999.
3. M. Rose, The Simple Book, Prentice Hall, 1994
4. M. Rose and K. McCloghrie, How to Manage your Network using SNMP, Prentice Hall, 1995.
5. L. Parziale et al., TCP/IP Tutorial and Technical Overview, IBM Redbook 243376, December 2006, Chapter 17, Network Management, http://www.redbooks.ibm.com/pubs/pdfs/redbooks/gg243376.pdf.
6. T. Bray et al., XML 1.1 (Second Edition), W3C Recommendation, August 2006. http://www.w3.org/TR/2006/REC-xml11-20060816/
7. K. McCloghrie et al., Textual Conventions for SMIv2, Internet Engineering Task Force Request for Comments 2579, April 1999. http://www.ietf.org/rfc/rfc2579.txt
8. K. McCloghrie et al., Conformance Statements for SMIV2, Internet Engineering Task Force Request for Comments 2580, April 1999. http://www.ietf.org/rfc/rfc2580.txt
9. K. McCloghrie et al., Management Information Base for Network Management of TCP/IP-based Internets: MIB-II, Internet Engineering Task Force Request for Comments 1158, March 1991. http://www.ietf.org/rfc/rfc1158.txt.
10. K. McCloghrie et al., Structure of Management Information Version 2 (SMIv2), Internet Engineering Task Force Request for Comments 2578, April 1999. http://www.ietf.org/rfc/rfc2578.txt
11. G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Lanuage Reference Manual, Addison-Wesley, July 2004.

Chapter 4
Discovery

Before any operational management of a computer system can be done, applications at the operations center must become aware of the different components of the computer system and the configuration of the different components. Without good discovery mechanisms, the operations center staff will not be able to manage the computing environment effectively. Therefore, discovery of the components in an environment is a crucial step in systems and networks management during operation stage of the life cycle.

Discovery is the process of identifying all of the manageable assets there are in the computer system being managed. Manageable assets may be of two types – physical assets which can be associated with a physical box or a library of software packages installed on a box and virtual assets which are other entities defined atop physical assets. Examples of physical assets include the set of network routers and servers, or copies of a word-processing software installed in different machines in an enterprise. Examples of virtual assets include the virtual network partitions in a data center, or the virtual web servers running in a hosted environment. Discovery may be done by a variety of mechanisms, some leveraging the capabilities of the assets being discovered in the system and others operating without any dependency on the asset being discovered.

The discovery process is used to obtain two types of information: (i) determining all the different hardware and software assets deployed in a distributed computer system, i.e., taking an inventory of the system and (ii) determining how the different hardware and software assets are interconnected, i.e. discovering the topology of the assets in the inventory. An inventory may be needed of physical assets as well as virtual assets. Examples of virtual assets include virtual networks or partitions created within a data center, connections established for the customers of a telecommunications company, or virtual web servers created on a physical web server.

In this chapter, we look at the various approaches that can be used to obtain the inventory and topology of different types of manageable assets. We begin with a discussion of the generic discovery process and the generic techniques to obtain the information and subsequently how these techniques are used within the realm of networking, servers, and personal

D.C. Verma, *Principles of Computer Systems and Network Management*,
DOI 10.1007/978-0-387-89009-8_4, © Springer Science+Business Media, LLC 2009

computers. Finally, we discuss how the information obtained in the discovery process can be stored efficiently at the operations center.

4.1 Discovery Approaches

The goal of the discovery process is to populate databases containing the information about managed assets in the operations center. We can view the discovery process being implemented by discovery software that collects information from the various elements in the computer system and populates a database containing inventory and topology information as shown in Fig. 4.1.

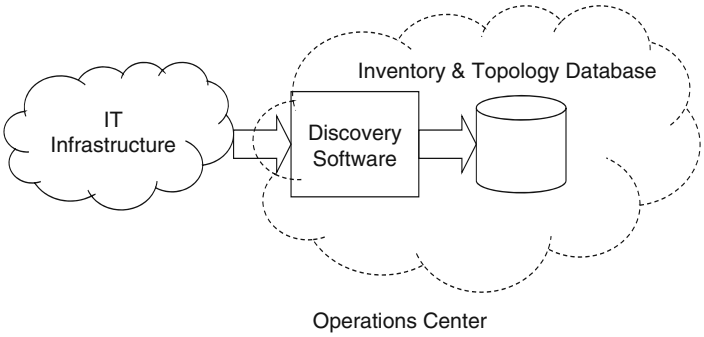


Fig. 4.1 Discovery process

In this section, we look at some of the techniques that are commonly used to discover the machines deployed in an organization. These techniques are general approaches, and specific instances of each technique can be found in specific domains such as networking or desktop management.

4.1.1 Manual Inventory

While not the most technically sophisticated method for discovering machines in an infrastructure, manual inventory is a common means for identifying all the assets an organization may have. In the manual inventory process, a human being or a team of human beings goes through the entire infrastructure of an enterprise in a methodical manner enumerating the different types of machines and their attributes. As an example, one may go through all the rooms in a building and make a list of all the machines that are located in each room, recording their identity, serial number, and other attributes.

Manual inventory is a laborious and time-consuming task. It is also a rather repetitive task, so mistakes are easy to make. However, there is a big benefit to

manual inventory. It can reveal systems that are turned off and disconnected from the network. It can be applied to peripherals such as monitors which are kept as backup and replacements, where no automated means will ever receive it.

Due to the amount of effort involved in manual inventory, it is best to use this process as a way to validate the correct operation of any automated discovery tool occasionally, or in the cases where no automated technique for discovery exists.

4.1.2 Dictionary/Directory Queries

In many types of computer systems, information about operation system is stored in some type of directory or dictionary. For proper network operations, all servers accessible to clients need to be stored within the network domain name system (DNS) directly. The set of applications installed in a windows machine typically create information records about themselves in the Windows Registry. In other types of environments, there may be information stored about the devices or applications running on devices in a directory. Sometimes the directory is maintained manually, e.g., an organization may record the serial numbers of laptop computers issued to employees and that may be recorded in a directory maintained by manual entry.

When a directory listing information about the set of assets is available, it presents an initial point for discovering a set of machines that may be available in the organization. Directory entries can provide information about all resources that are supposed to be in the system, regardless of the current state of the machine (whether it is up or down, connected or disconnected). However, directory records may be subject to data entry errors if they are maintained manually. Sometimes, the directory entry may contain information which may be out of date, e.g., about a machine that has been decommissioned but the directory entry was never updated.

In most cases, access to directory information provides an initial list of machines or assets to be included in the inventory. Other discovery processes are used to discover items that may be present in the system, but not recorded in the directory, or when the attributes of a system differ from the ones recorded in the directory.

4.1.3 Self-Advertisement

Many machines and systems come with software on them that advertise their presence to a server on the network. When booted up, the machines will broadcast their information on a multicast or predefined address, where a listening application can identify the different machines that have come up on the system

and are sending advertisements of this nature. The advertisement protocol may be proprietary to the vendor of the machine, or it may be following some established industry standards.

When self-advertisement is done by machines and software packages coming up in an environment, and such advertisement is done periodically, it is easy to discover the systems doing such advertisements. Since the advertising message also includes information as to how the specific device may be contacted, the listening management system can query the newly discovered device to obtain a larger set of parameters than the ones that are carried in the advertisement method.

4.1.4 Passive Observation

During the operation of any computer system, several routine functions are performed automatically by the different elements of a computer system. Clients make requests to servers, network routers exchange information to determine the optimal paths, clients machines coming up send a messages to a server to obtain their dynamically assigned network addresses, etc. By watching the flow of information happening in the environment, one can discover the presence of many devices and software systems in the computer infrastructure.

As an example, one may observe the network packets that are flowing on an enterprise network on an IP-based intranet. The network packets contain the IP network addresses of the senders and the receivers. By collecting the set of network addresses that are actively communicating at a router, one can determine the set of machines that are communicating via that router. Duplicates can be eliminated and the unique identity of the different machines identified. By examining higher level protocol headers, the other active applications on the network at each of the machines can also be identified.

Another type of passive monitoring in the network can be done by monitoring the routing exchanges in the network. Some of the routing protocols exchange the complete topology of all nodes in an Autonomous System (usually a single administrator network). Tapping into the information of the routing protocol, e.g., by listening in to the bytes sent out by a local neighbor, the identity of most elements in the network can be determined.

4.1.5 Agent-Based Discovery

A common technique for discovery is through specialized agents installed on the different machines. An agent is a software package installed on a machine

that runs on specific event triggers or periodically. This technique is commonly used for servers and client stations for discovering applications that are installed on them. In this generic approach, a discovery agent in installed on all the machines that are owned by an organization. The discovery agent can then collect information about all the installed applications on a machine, decipher the machine's identity, and then update this information onto a database maintained in a management server.

In order to discover the machine, the agent needs to be installed on each machine and send information/update about the machine to the some network directory. In many cases, the details of the server to which the information needs to be sent needs to be configured. The agents are usually installed and configured on the machine when it is first put into use in the organization.

Agents can use their own proprietary protocol to communicate and register themselves with the management system, and thus provide a more efficient method for communicating their presence. However, installing and maintaining agents on all the machines in an organization is a complex task which can become daunting in larger enterprises which have many different types of systems.

4.1.6 Active Probing

Yet another generic mechanism used for discovery is the method of probing the entire IT infrastructure to identify all of the machines that are present. The probing is usually done by means of a software running at the operations center. In the probing method, the probing software starts from a set of known machines (e.g., from a set of machines read out from a directory system) and then finds information about the neighbors of the machine, as well as applications installed on the machine. To determine the applications on the machine, the probe software may send test messages to specific servers to see if any response is received. The nature and type of response that is received will indicate which application is running on the box.

Within the network, probing often takes the format for reading management MIBs or local device configuration and using them to discover other devices such as neighbors that can be probed further. In some cases, the probes may read information about other machines that may have exchanged messages with the machine being probed.

Probing is usually time consuming, especially when a large number of nodes are involved. Nevertheless, it is effective in determining the real interconnection topology of the IT systems, as well as identifying which applications are running on the various machines.

4.2 Discovery of Specific Types of IT Infrastructure

In this section, we see how generic techniques discussed in the previous section are applied and used in the domain of discovering machines in specific type of machines. The machine types we consider are discovery of the network devices, discovery of the servers in a data center, discovery of client machines, and the discovery of applications installed in client or server machines.

4.2.1 Discovering Servers

There are several techniques that can be used to discover servers working in an enterprise or data center as described below.

4.2.1.1 DNS Zone Transfer

The Domain Name Service (DNS) provides a convenient means for obtaining information about any network accessible server in an IT infrastructure. DNS [1] is a mechanism used to translate human-readable machine names to their network addresses on the Internet. Such a translation is necessary for any machine in the network to connect to the server, and thus it is obligatory for all server machines to register their names and IP addresses to a DNS server in their domain (a domain is a set of machines under control of a single management system). The DNS system consists of all the DNS servers that communicate with each other to resolve the names for any machine on the Internet, or equivalently on an IP-based intranet.

In order to improve performance and reliability, DNS was designed to accommodate more than one server to handle names for any single domain. These servers need to maintain information consistently among all of them. Toward this goal, a functionality called zone transfer was defined by the DNS specifications [1] and is widely implemented. Zone transfer function allows a secondary name server to query the primary name server of a domain to get a quick dump of all the machines registered within that domain. A management tool can also make the same query and request the names of all the machines that belong to a specific domain, e.g., such as watson.ibm.com. The DNS server would respond with a relatively long message that would contain the names and IP addresses of all the machines that are registered as having a machine name ending with the suffix of watson.ibm.com. The information returned by the DNS zone transfer for each machine consists of the domain name of each device registered with the domain, and the IP addresses of each of the interfaces that the device has, and provides a listing of all the devices that are registered with the DNS system.

The information obtained by zone transfer only contains the machines that are registered with the DNS system and typically will not consist of machines

that are assigned addresses dynamically using a protocol like DHCP. Furthermore, the DNS database consists of a static registration of machines and the IP addresses assigned to them, and it may have stale information listing machines that are registered with the DNS system but are no longer being active. The DNS listings do not provide any information about the interconnection of different machines, although the structure of IP addresses can be used to infer some ideas about the relationships among servers. In some installations, zone transfers may be limited to a small number of authorized machines, and the management system must belong to that set.

4.2.1.2 Traffic Analysis

An alternative way to discover server and client workstations that are active in an IT infrastructure is to observe the flows of packets in the network and identify the active packets from the header information contained in those packets. The technique works better for servers whose network addresses are fixed, as opposed to clients whose network addresses may frequently change.

In this method, packets flowing through an appropriately located point in the network can be collected and inspected by a software package. Assuming that the network is based on the Internet Protocol (IP) technology, each packet carries the IP address of the two end points of the communication. By collecting the address of both the source and the destination addresses of the packets communicating on the network, the network address of the servers in the system can be determined.

Some of the addresses collected in this manner would be of machines that are not in the administrative domain of the management system. If the management system is aware of the range of IP addresses that are present on its network, it can filter out the addresses that are not within the range of permissible addresses. In IP networks, specification of these network addresses is done using the concept of subnet masks.

In IP networks, addresses are 32 bit numbers that are assigned in continuous blocks. In each block, a set of bits are kept constant, while the other bits change for different machines in the network. The subnet mask is another 32 bit binary number, defined so that it contains a 1 for those addresses which will remain constant in the subnet. A subnet is specified by a subnet mask, and bitwise comparison of any network with subnet mask readily identifies if an address belongs to a subnet or not.

Traffic analysis can also determine which applications are running on each discovered server. In the IP protocol, it is assumed that the transport protocol (TCP and UDP being two common examples) running on an IP network supports the concept of ports. A port is like a mailbox at each server on which applications can send and receive data packets like people pick and drop letters from their individual mailboxes in an apartment complex. The ports on each computer in the TCP/IP protocol suite are numbered between 0 and 65535. Most common applications running on servers run on a default port

number. The standard set of applications would typically use a default port number assigned by Internet Assigned Number Authority (IANA) [2]. Applications developed in house would typically run on their own port assigned by the system administrators. Although there is no guarantee that an application is not running on a port other than the one assigned to it, most applications on servers would tend to follow the convention.

By examining the ports on which packets are being exchanged between machines, one could identify which applications are running on different servers in the network. If a port number is not readily identified, one would need to look into the specifics of the application header beyond the transport protocol (TCP or UDP) to determine which application is running. Since most applications tend to use headers that follow specific conventions, it is possible to identify a large variety of applications by examination of the packets in the network.

Traffic analysis finds the set of servers and applications that are active in the network. It cannot detect the set of servers that are disconnected from the network, or not communicating actively on the network. The discovery process only identifies the network address of the servers, can be extended to identify applications running on the servers, and would needs to be augmented by other mechanisms if additional information is required about a server such as the serial number of its machine, or the amount of memory installed in the machine.

4.2.1.3 Agent-Based Discovery

The discovery of servers and their interconnection can be simplified significantly if all the servers in an IT infrastructure have the same type of software agent installed on them. The software agent installation can be part of a standard software installation on the servers in an organization. Agents can be active or passive, an active agent being configured to connect to a management server, while a passive agent being configured to receive commands from a management server and respond to them.

An active agent can use the self-advertisement to report itself to the management system. The only drawback in an active agent system would be that the management system may be flooded by too many self-advertisements simultaneously, and a common scheme used to stagger the requests around is to add a random delay before an agent self-advertises after the start-up period.

A passive agent can augment the information obtained from a directory zone transfer or a network-based traffic analysis discovery during discovery. The agent, when contacted by the management system, can collect detailed information about the local machine and transfer them over to the management system.

In most cases, the agent should be able to read the local management information expressed as SNMP MIBs or as DMTF CIM data model and provide them to the management system to populate its inventory table.

4.2.1.4 Agent-Less Discovery

Installing and configuring agents on all the servers can be a chore, and thus a model of discovering a server machine and its detailed information without requiring an agent can be very attractive. Agent-less discovery systems are designed to augment information about a set of discovered machines. Their main implementation is in the form of remote scripts that are executed on the servers by the management system in order to complete details that are not readily available from the discovery mechanism.

In order to invoke the scripts, the management system needs to have the right credentials to run the scripts on the server after logging in. Such credentials can be obtained easily if all the servers in an enterprise have the same administrator use rid and password. Since such a system may be quite insecure, another alternative would be to obtain the right credentials for logging in and running a remote script to be obtained from a directory server.

4.2.2 Discovering Client Machines

The counterpart of discovering servers is the task of discovering client machines. As opposed to the server machines, which would tend to have a fixed network address, the client machines typically will not have an assigned network address. The address of the machine is assigned dynamically using a protocol such as DHCP. A few client machines may have fixed network addresses but they tend to be the exception rather than the norm.

In such an environment, discovering machines using a DNS zone transfer or network traffic analysis is not a viable option. The client machines are not likely to be registered with the domain name server, and the lack of a one-to-one association between the network address and the client makes it hard to track client machines using traffic analysis. While network traffic analysis can identify the current IP address of a client machine, it would not be easy to distinguish between two machines that have gotten the same IP address after some time interval has elapsed and when a machine has been idle for some time and has started retransmitting after some time interval.

Nevertheless, the management system can collect the list of all registered hosts at the DHCP server and thereby obtain the list of all active clients. In order to uniquely identify the clients, the management system needs to have an identity other than the network address associated with each client. In many cases, the address of the network adapter (the MAC address of the machine) can serve as the unique identifier. In other cases, where the DHCP address is provided only to a set of authorized clients

who need to present their credentials, the credentials can be used as the unique identifier.

The approach of having an agent installed at each of the machines works well for client machines as well. The agents would need to advertise their presence to a management system with the current network address and any identifying information about the machine. The management system can record the attributes of the machine in its inventory database.

4.2.3 Discovering Applications on Servers and Clients

The applications installed at a server or client need to be discovered and recorded in a management database for several reasons. One reason is to take an inventory of all installed applications in an organization, e.g., to validate that no illegal copies of software are being used. Another reason is to validate that the set of applications involved are consistent with corporate policies.

When discovering applications, one can try to discover all applications that are installed at a machine or all the applications that are running at the machine. The latter will tend to be a more dynamic set than the former, but both types of discovery have their usage. Information about all the applications installed on the machine can be collected at a central server to keep track of application licenses that may be needed, while the number of application instances actually running can be used to analyze performance problems and/or configuration problems.

In general, the discovery of applications is done by agents running on the machines. Alternatively, the discovery of applications may be done by means of an agent-less system that logs in remotely to the machine and runs a discovery script. In both cases, the software needs to determine which applications are installed or are active at the machine.

In order to identify the set of applications installed on a machine, the following general techniques are used by the application discovery software:

FingerPrinting: Most installed applications leave a fingerprint of their installation by creating files with known names and frequently in a well-known location. In other cases, an installed application will result in a specific tree structured in the file system. Discovery tools using the fingerprinting method run through all the files in a system starting from the root and apply inference rules to determine which software is installed on the machine. The inference rules are specific to the operating system which is installed on the machine as well as the specifics of the software that is being checked for installation.

Some application developers may have been nice enough to provide identification about the application being developed in the executables or binaries of the application, e.g., the Microsoft development tools enable programmers to insert information into their programs using the VersionInfo command. In such cases, the information in the headers of

the executables can be used to determine the application. However, not all programmers may have put the information in there.

Fingerprinting can detect software that is installed properly and is contained in a list of software with known fingerprints. Using it for custom-built software requires adding rules to identify the new software. As the number of software packages potentially installed at any machine increases, fingerprinting becomes less efficient. Furthermore, as the size of disk space and consequently the number of files available at any machine increases, fingerprinting can be quite expensive in terms of the time taken to run a full scan of the entire system.

Registry Scanning: In many operating systems, especially the Windows operating system, a system registry contains information about all installed applications. Going through the registry and scanning the list of all applications that are installed provided a good initial list of the applications installed at the system. The windows registry contains pointers that tell each application where to look for its files or configuration data. Scanning the registry will provide information on all the software that is installed on the machine.

In some operating systems such as Linux, a centralized repository of installed applications does not exist. For such operating systems, there is no option but to scan the file system for fingerprints of installed applications.

In order to identify the set of applications that are running on the system, the following methods can be used:

Process Examination: In all operating systems, there are commands available that will enumerate the set of active processes on each machine. Each of the active processes corresponds to an application that is currently running on the system. The agent can catalogue the set of active processes, determine the binary that is being executed by the process using mappings available from the operating system of the same, and then identify the application associated with that binary. In some operating systems, commands that identify the set of active applications may be available.

Port Scanning: Applications that are running on a server and waiting for connections from remote applications can be identified by means of a port- scan. A port-scanning program will report the process of a binary running on each port on which the process is waiting to receive connections from the external host. The port scan identifies processes that are communicating with other machines, and thus is useful in assessing security exposures.

Traffic Analysis: As mentioned previously in Section 4.2.1, traffic analysis can identify the applications running on servers by examining the structure of internal headers in a communication. The traffic analysis approach will work as long as the information flow is not encrypted so that the content of the packet information (or at least the headers) can be examined and analyzed for the application that may have generated those headers.

4.2.4 Discovering Layer-3 Network Devices

Networks are defined traditionally into seven layers, with the physical layer being layer 1 and the application layer being the seventh one. Layer 3 refers to the networking protocol layer according to his classification system. In most modern networks, layer-3 is the Internet Protocol. The discovery of the layer-3 network connectivity is simplified by the fact that there are known standards that can assist in discovering IP network topology. The SNMP standard provides a way for discovering devices that are visible in an IP network.

Some of the methods described earlier, e.g., analyzing network traffic or DNS zone transfers, work well for end points of communication. They are not very effective at identifying the intermediary IP routers, primarily because the network router is the target of end point of very few connections. Routers are not likely to have an entry in the DNS system, and the IP protocol headers only contain the end points of a connection, without any information about the intermediary routers. The exception is the use of some options that enable the recording of intermediate routers on the path, but that option is not very likely to be turned on in most networks.

The approach that works effectively at level 3 networks is the reading of SNMP MIBs that contain information about the neighbors of a node. For any router or server that supports the SNMP basic standards, a MIB is available that lists the IP address of the interfaces available at the machine, as well as a MIB that summarizes the routing table used for forwarding IP packets at the machine. The routing table consists of entries which contain three elements, a destination address, the network address of the next hop router, and the interface via which the next hop router can be reached.

Starting from any router or server in the network, a SNMP manager can retrieve the list of all the interfaces on a machine and then walk through the routing table to determine all the neighboring nodes of the router and the interface through which that neighbor is reachable. The SNMP manager can then select one of the neighbors and retrieve the list of all the neighbors that this node has and then iterate through the entire graph in this manner. The manager may choose to traverse the graph in any manner possible, e.g., using a breadth first strategy or a depth-first strategy, or any other strategy to decide which node ought to be discovered next.

SNMP traversal in this manner does have some limitations. One limitation is that it takes a significant amount of time to poll and discover a large network. If some of the machines are not operational for any reason, the discovery mechanism will not provide complete results. Furthermore, the management tool needs to have the right credentials to obtain the desired information from the SNMP agents in the environments where the SNMP agents implement some form of access control.

When a management system does not have the right credentials to perform SNMP queries on the routers, a more expensive way to discover the network is

through the use of protocols such as `ping` and `traceroute`. Using the ping method, the discovery software generates the addresses of the different potential machines in each IP subnet that it discovers. Then it can use the ping command to check which of the machines are available and which ones are not.

Using the traceroute command, one can trace the route of the network packets flowing between any two nodes. By invoking the command on one set of paths provides information about the routers in the intermediate path, and thus identifies their network identity. In many cases, such techniques have been used to map the topology of Internet or other large networks where the administrative control over all of the nodes in the network cannot be assumed.

Using traceroute to detect the topology has several limitations. Some of the intervening nodes may have turned off traceroute for security reasons. The time reported by the traceroute tool is very erratic since traceroute uses the ICMP protocol, which is generally processed on a different and slower path in most router implementations than regular forwarding. Furthermore, the way the tool is designed, traceroute sends many packets into the network to try to discover paths and can be a burden on the network.

4.2.5 Discovering Layer-2 Network Devices

The layer-2 network refers to a single physical or logical subnet which runs below layer 3 (IP network). An Ethernet segment, a ATM network and, a MPLS network are examples of layer-2 networks that support an IP-based layer-3 network on top.

Due to their design, link layer discovery is transparent to network management applications. Thus, the discovery of the link layer environment using a standard network management product is hard.

Discovery of devices on a layer-2 network is enabled by means of link level discovery protocols. A standard link level discovery protocol (LLDP) is available on many vendors and works on a design principle similar to that in proprietary link level discovery protocols such as Cisco Discovery Protocol or the Nortel Discovery Protocol. Most of the link layer discovery protocols work by means of advertisement. Each node in the subnet periodically broadcasts information about itself to a multicast address in the layer-2 network. The other nodes in the network receive the packet and update a table of adjacent neighbors the nodes maintain. The information can then be forwarded up to a network management agent which can report it to a network management system.

The IEEE 802.1AB standard defines a LLDP protocol for use in mixed-vendor networks which allows discovery of the devices in the physical network level. The protocol allows all nodes in the same physical subnetwork to discover each other (since each is broadcasting its information to other nodes) and represents it as a standard SNMP MIB. Subsequently, a

network management station that is able to query the node remotely can find the set of devices that are on the physical layer-2 network on the device.

In order to discover all the devices in a computer system infrastructure, a combination of techniques for discovering the various devices needs to be used. The information using each of the techniques is complementary and can be used to correct errors that may be introduced due to the user of any one single technique.

4.3 Storing Discovered Information

The goal of the discovery process is to populate databases containing the information about managed assets in the operations center. In this section, we look at the type of information that needs to be contained in the database.

At the present time, the dominant paradigm in database is the use of relational databases. Relational databases represent information as consisting of several tables. Information in the tables is obtained by performing different types of queries among the different tables. The main method for accessing information from relational databases is using the Simple Query Language or SQL [3].

The inventory database needs to contain information about the different types of assets. In general, one can assume that there will be a separate table for each different type of asset in the system. Within each table, each asset will have a record where several attributes of the asset will be listed. For each asset, one or more attributes can be combined to provide a unique identity for that asset – these attributes are usually used as the key for that table. Thus, an entry for a personal computer may contain the serial number and the manufacturer of the computer (the unique identity can be constituted from the manufacturer name and the serial number), the model, the size of its hard disk and memory, the owner of the personal computer, and the building location and room number where the personal computer is located. Other types of assets will have different attributes associated with them.

The topology database shows the relationship among the different assets in the database. The relationships can be among a pair of assets (e.g., a link connecting two devices), may be hierarchical in nature (e.g., applications running on a server), or it may be many-to-many (e.g., devices connected on the same network segment). In each case, a suitable representation for the relationship needs to be maintained within the database.

While storing an inventory of the assets in a relational database is relatively straight forward, storing relations in such a system requires some additional work. Relational database requires tables to be defined statically, with a fixed set of columns. Maintaining arbitrary interconnection information in such systems, where a node may have multiple neighbors, requires that the attributes in the tables be constructed with some thought.

The topological relationship among different entities in the system may be a hierarchy, a general graph with point-to-point links only, or a general relationship. For each of the types of relationship, a choice is needed to represent the relationship appropriately in the relational database.

4.3.1 Representing Hierarchical Relationships

A hierarchical relationship is one in which a distinct parent–child relationship exists among different assets in the management system. As an example, an organization has different physical locations. At each physical location, there are several servers, networks, and storage devices. Each device has a set of software packages installed on the system and a set of hardware interfaces attached to the server.

Let us assume that the discovery process has discovered the different elements that are as shown in Fig. 4.2. The hierarchical relationship among the different types of discovered assets, the locations of the servers, and the applications installed at each server are shown in the figure.

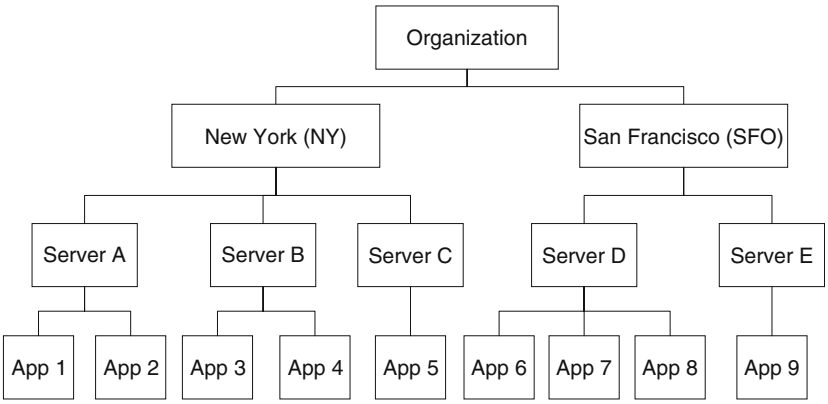


Fig. 4.2 Example of hierarchy

In a relational database system, a common way to represent hierarchical relationship is to list the parent node as an attribute in the asset. As an example, an application instance may have an attribute indicating the server on which it is executing. Since each child has exactly one parent, one needs to introduce only one additional column in each table to account for this relationship. In this case, the table of assets representing the different assets shown in Fig. 4.2 will appear in a manner as shown in Fig. 4.3.

Common operations performed on the entries of an asset in the database include (a) finding its parent node, (b) finding its immediate children, (c) finding all the nodes in its subtree, (d) removing the node, and (e) adding a new node as

Location	Parent
NY	Org
SFO.	Org

Server	Parent
A	NY
B	NY
C	NY
D	SFO
E	SFO

Application	Parent
1	A
2	A
3	B
4	B
5	C
6	D
7	D
8	D
9	E

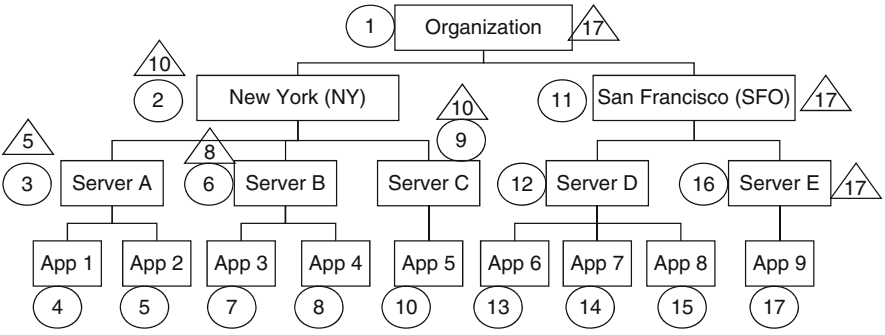
Fig. 4.3 Representation of hierarchy in relational databases

a child of an existing node. Listing the parent as an attribute makes the determination of the parent node trivial. The immediate children of the node can be found by searching through the list of all nodes to find those with a specific parent node. When a new node is added, it simply has to list the correct parent as one of its attributes.

However, the operation of deleting and determining all children of a node requires some work with this representation. When all children of a node need to be determined, one needs to find the immediate children and recursively apply the process on the intermediate children. In any moderately sized database table, such recursive searches, done through database join commands, can be prohibitively expensive. Similarly, when an asset is removed from the database, one needs to search for all of its existing children and either remove the children or update their parent node to the new node that will be its parent (in case they are not being deleted along with the parent node).

A more efficient approach to represent hierarchies in a relational database is to associate two numbers with each node in the hierarchy in addition to its parent. Thus, two attributes are added to each row corresponding to the asset in their tables in the database. The first number represents the sequence of the node when the tree hierarchy is traversed in a depth-first search. Let us call this number the index of the node. The second number is the largest index of any child of the node. Let us call this number the rchild of the node. The children of any node can be determined by searching for nodes whose index is between the two numbers associated with the node, thus requiring only one search among the database tables. If each of the tables are arranged so that they are easily searchable on the index attribute, then searching for children can be much more efficient with such a representation. On the flip side, when adding or deleting nodes in the structure, one needs to recalculate the index of all affected parent nodes.

Figure 4.4 demonstrates how the tables in the relational database would look if we were to use the depth-first index to store the tree structure shown in Fig. 4.2. Each node is associated with two numbers, one in a circle and one in a triangle. The number in the circle is the position in which a depth-first traversal



Org	Parent	Index	Right
Organization	Null	1	17

Location	Parent	Index	Right
NY	Org	2	10
SFO.	Org	11	17

Server	Parent	Index	Right
A	NY	3	5
B	NY	6	18
C	NY	9	10
D	SFO	12	15
E	SFO	16	17

Application	Parent	Index	Right
1	A	4	4
2	A	5	5
3	B	7	7
4	B	8	8
5	C	10	10
6	D	13	13
7	D	14	14
8	D	15	15
9	E	17	17

Fig. 4.4 Depth-first search index method

of the tree will visit the specified node, i.e., the index of the node. The number in the triangle is the index of the rightmost child in the tree, or equivalently the highest index among any of the children in the node.

Let us examine how the different operations will be performed using this data structure. The parent node can be obtained by following the parent pointer. The immediate children of a node can be found by searching for all nodes with indices greater than its index but less than its rchild value, where the parent pointer is the node itself. All the children of a node can be done by searching for all nodes with indices greater than its index but less than its rchild value. When a node is added or removed in the structure, one needs to traverse the entire tree and recompute the index and rchild value of the nodes in the tree. As an optimization, one cannot make any changes in the tree for a deleted leaf node. It leaves some gaps in the DFS indices but does not affect correctness of any operation.

If one is expecting infrequent changes in the structure of the tree, but frequent queries regarding the children of the tree, then the depth-first search index method can be a much more efficient way of storing a hierarchy. In systems management, one would not frequently expect changes in the physical assets deployed in an organization, so this approach works well for keeping track of physical assets.

A third way to represent hierarchies is to store the information about the parent node as well as the path from the root of the hierarchy to the node itself. If the database system provides the ability to search for entries starting with a specific prefix, then a single search of the table of nodes with entries starting with a specific prefix returns them all. However, when the parent of any node changes, one needs to update the entries corresponding to all the nodes in the sub-tree under that node.

A good discussion of different techniques to represent hierarchical information in databases and SQL code to perform common operations can be found in [4], which uses the term adjacency list representation for storing a reference to parent as an attribute and nested set representation for a variation of the method using depth-first indices.

4.3.2 Representing General Graphs

In many cases, the topology information consists of similar types of nodes that are interconnected by point-to-point links. A common situation arises in maintaining an inventory of elements of a network. When one is managing one layer of the network stack, e.g., managing the IP network on an enterprise, or managing the SONET links in a telecommunications company, the topology information consists of a large graph with several nodes interconnected among each other using point-to-point links.

Since a node may have several links, and the numbers may vary from node to node, it is not trivial to add a column per edge to the table containing the description of nodes in a relational database. The following approaches can be used to represent graphs in the relational database system.

If there is an upper limit on the number of edges that can belong to any node, then additional attributes allowing the definition of the maximum possible number of edges can be defined. If there are M maximum possible number of edges in the system, and we require K attributes to describe each edge, then one table describing the nodes will have M*K additional attributes. The table will have the attributes filled in for existing edge and left as blanks or some predetermined values for edges that are not present.

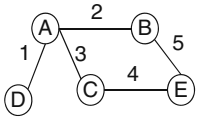
If we do not want to waste the space in database tables due to the number of empty attributes that appear using the previous method, an alternative approach will be to define a new table for each edge. In the edge table, there will be an entry for each edge, and the edge will have two columns identifying the nodes on either side of the edge. The edge can have additional attributes represented by additional columns. In order to find all edges belonging to a node, one would search through the edge table looking for the entries where the value of one of the node columns matches the node we are looking for.

Another option is to have an independent table of edges, and a table of nodes, but add an additional attribute to the node table. The attribute of the

node table is a concatenation of all the edges that the node contains and points to the appropriate entry in the table of edges. In that manner, the edges corresponding to a node can be efficiently looked up. However, additional processing needs to be done in order to extract the list of edges that are stored in a concatenated format in the database.

The manipulation of such lists containing the complex information is usually done by means of additional software modules that are used to access the relational database by the management system software.

Figure 4.5 shows a sample network and the representation of that network in the relational database system using the three approaches described above. Approach (i) shows the technique of listing the adjacent nodes up to a maximum limit. Approach (ii) shows the listing of a separate edge table, and Approach (iii) shows the listing of an additional aggregate column in the nodes table. In each of the tables, additional attributes may be stored with the node or the edge, which are not shown in the example figure.



(i)

Node	Nbr 1	Nbr 2	Nbr 3
A	D	B	C
B	A	E	-
C	A	E	-
D	A	-	-
E	C	B	-

(ii)

Node	Edge	N1	N2
A	1	A	D
B	2	A	B
C	3	A	C
D	4	C	E
E	5	B	E

(iii)

Node	Links	Edge	N1	N2
A	1,2,3	1	A	D
B	2,5	2	A	B
C	3,4	3	A	C
D	1	4	C	E
E	4,5	5	B	E

Fig. 4.5 Storing a graph in relational database

4.3.3 Representing Generic Relationships

In a graph model considered in Section 4.3.2, an edge connected exactly two nodes. In general systems topology, it is possible to have edges that connect more than one node together, e.g., an Ethernet segment may interconnect several machines. Another relationship that is usually found in the systems topology is the concept of aggregation, e.g., one type of node contains many instances of other types of nodes.

In some cases, the relationship may be one to many, e.g., if we make the assumption that each machine will only be connected to one Ethernet segment. Although this relationship is not true in data centers, it is likely to hold true in the case of client desktop environments. In the case of one-to-many relationships, an additional attribute can be added in the table of machines identifying

the Ethernet segment on which it is connected. When one needs to find out all the machines on the same Ethernet segment, one can search through the table of machines looking for those which mention the specific Ethernet segment.

In the more general case, the relationship can be many to many. A machine may belong to more than one Ethernet segment. In such cases, one way to represent the relationship in the database is to introduce a new table of relationships. Each relationship in this case can be viewed as a virtual node in the graph, where other nodes include the Ethernet segments and the machines. We can now draw an edge from the relationship node to each of the Ethernet segments which are included in the relationship and similarly an edge from the relationship node to each of the machines involved in the relationship. Now, one can use the general approach to represent graphs in database tables as described in Section 4.1.2.

Aggregation relationships may be handled in a similar manner. For each aggregation or set of nodes that is defined, one can introduce a virtual node in a graph. The virtual node is connected to the nodes that are included in the aggregation. Then, the approaches used in Section 4.1.2 can be used to represent the aggregation relationships.

4.3.4 Other Types of Databases

The representation of general systems management and arbitrary topologies in a relational database can sometimes be difficult. One option is to use a different type of database. In addition to relational databases, other types of database representations that can be found commercially include databases following the network model, databases following the hierarchical model, and databases following an object-oriented model. The network model of the database provides a more natural model for representing the topology and interconnection among the different objects. Management information that is represented using an object-oriented paradigm such as CIM may be more convenient to represent in an object-oriented database. However, relational database is the type of dominant commercial deployments currently, and some techniques such as network databases are only supported in older products. Thus, an implementation choice of the other types of database models may be fraught with some practical constraints.

Another option would be for the management system to implement its own method for storing the management inventory and interconnection topology. In these cases, the management system is using its own private database implementation. The implementation of a private database requires more effort, but may be useful in some cases where performance, scalability, or responsiveness requirement of the management system cannot be readily satisfied by implementing onto a relational database system.

4.4 Summary

This chapter discusses the topic of discovery, which is a crucial initial step for any type of management during the operation life cycle stage of a computer system. Discovery is the process of creating an inventory of all managed assets in a computer system and for discovery of the dependencies between those assets. The different approaches for discovery are introduced, followed by a discussion on specific techniques that can be used for network devices, servers, and application software. The different techniques that can be used to represent the discovery information in relational databases are introduced.

4.5 Review Questions

1. What is discovery? Why is it important for systems management?
2. The discovery model discussed in this section assumes that the discovered information is stored in the database. What are the characteristics that the computer system must satisfy for this assumption to be valid. Can you think of any computer systems for which those characteristics may not be satisfied?
3. What are the two primary types of information that are discovered by the discovery process?
4. Why are relational databases used for storing discovery information even when the representation of relationships among object is not readily supported in such databases?
5. What are the alternative approaches to store hierarchical relations in a relational database? What are their relative strengths and weaknesses?
6. What are the alternative approaches to store a graph in a relational database? Discuss the pros and cons of each approach.
7. For what types of computer systems is manual inventory the only way of discovering the computer system components?
8. How would you design a system for discovering the components of (a) a computer network (b) a shared hosted data center, and (c) an enterprise IT infrastructure?
9. What type of components can be discovered with passive observation? What type of components are likely to be missed in a discovery system which only uses passive observation for discovering components?
10. What are the pros and cons of an agent-based discovery mechanism as compared to agent-less discovery mechanism?
11. Describe the different approaches that can be used to discover all applications that are installed on client workstations in an enterprise.
12. *Project*: Develop the architecture and prototype of a system that will discover the different components of the computer system operational at your college.

References

1. C. Liu and P. Albitz, DNS and Bind, O'Reilly, 2006.
2. IANA List of Assigned Port Numbers, http://www.iana.org/assignments/port-numbers
3. R. Sheldon, SQL: A Beginner's Guide, McGraw-Hill Osborne, 2003.
4. J. Celko, Joe Celko's Tree and Hierarchies in SQL for Smarties, Morgan Kaufmann, May 2004.

Chapter 5
Monitoring

After all the elements of a computer system have been discovered, they need to be monitored. Monitoring of a computer system is defined as the process of obtaining the status and configuration information of the various elements of a computer system and consolidating that information. Information consolidation may involve the tasks of preparing reports to be viewed by an administrator, cleaning of the raw-monitored information, and consolidation of the monitored information into more compact representations.

The management of a computer system requires the monitoring of a variety of data. In the first section of this chapter, we look at some of the different types of status information that need to be monitored for systems management. The next section discusses a generic model for monitoring structure in a network operations system. The next sections expand upon the various aspects of the generic model.

5.1 Monitored Information

Monitoring is a function that is a prerequisite to almost all of the management functions that one can envision. Before any functional aspects of the FCAPS functions can be managed, the relevant information from all elements of the computer system needs to be monitored and provided to the human administrator or software analysis engines that will assist the administrator to manage that function. The nature of monitored information that needs to be collected for managing each function tends to differ significantly.

Despite the difference in the needs of each functional management, the type of monitored information can be divided into a few logical groups. The list below enumerates the typical logical groups of information that is monitored for each element of a compute system:

Status Information: Each element of a computer system at any time may be turned on and functioning properly, not be on, or be turned on but not functioning properly. The status of elements can have a significant impact

D.C. Verma, *Principles of Computer Systems and Network Management*,
DOI 10.1007/978-0-387-89009-8_5, © Springer Science+Business Media, LLC 2009

on the performance of the overall computer system and is required for many functions in all five areas of FCAPS.

Configuration Information: The configuration of an element consists of all attributes of that element which can be modified by an administrator. The configuration of an element needs to be monitored to validate that it is proper, not a cause of a performance degradation, or responsible for some faults in the system.

Usage and Performance Statistics: The usage information of an element consists of all attributes regarding its throughput and the number of active users it supports. The usage information would include information such as the number of packets or users or sessions that are active in the system. Other metrics, such as a high water mark of buffer occupancy or the number of memory buffers used are also examples of usage information. The performance information collected from an element includes the delay and latency of requests that are successfully completed at the element.

Error Information: The error information includes information about faults and incorrect operation at the element. Error information may include error codes as well as information about the number of errors that have been encountered in some preceding time interval.

Topology Information: Although the topology information is obtained during the discovery process, changes to the topology may need to be obtained as part of the monitoring process. Obtaining the topology information is essential for systems where the locations of some devices may be changing frequently, e.g., in the case of cellular telephone or handheld devices, as well as for mobile ad hoc wireless networks.

In various types of IT environments, the nature of the above information may be different, e.g., the throughput information in the case of a web server will be the number of HTTP requests received, and in the case of a network router the number of packets transmitted by a router, and in the case of an e-mail server, the number of email messages processed. Nevertheless, there is an inherent similarity in the metrics that are collected, and the characteristics of the information that is collected similar in nature.

5.2 Generic Model for Monitoring

Although there are many different monitoring systems and products in the marketplace, the structure of most monitoring systems used in management can be described by means of a generic architecture. The generic architecture is shown in Fig. 5.1 and consists of five layers of monitoring functions shown in the figure.

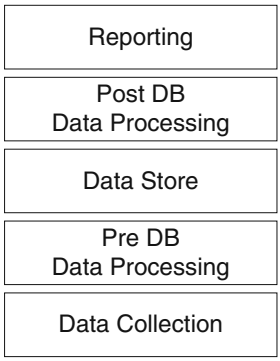


Fig. 5.1 Generic structure of monitoring applications

The first component of the monitoring model is the data collection component. The task of the data collection component is to retrieve the relevant modules of information from the various devices and software components that make up the computer system. The data collection module produces the raw information collected from the different devices.

The raw data from the devices are subjected to various types of processing before it is stored into a database (DB). This processing is done in real time as the data are received from the system or network. The pre-DB data processing is involved in the task of reducing the data volume so that it is manageable. The data processing is also responsible for identifying any data that may be missing, or data that may be redundant. Additionally, a format conversion from the type of data collected from the devices and the format in which the data are stored into the database is also done at this stage.

The database stores monitoring information for persistence and off-line analysis and reporting. Most large systems produce large amounts of data, which need to be stored in a database so that it can be analyzed subsequently for historical trends and to perform analysis which is difficult to perform in real time.

The post-DB processing performs the requisite analysis on this data. The bulk of intelligence in network and systems management is focused on the various types of data processing algorithms that can be applied at this stage. The reporting deals with presenting the information in an effective manner to the administrator.

In the next several sections, we look at the technical challenges and the algorithms required in the data collection, pre-database processing, and the database components of the system. The post-database processing and the reporting schemes are discussed in the subsequent chapters since they tend to be specific to the type of management function being performed.

5.3 Data Collection

Data collection is the first step of the monitoring process. In the data collection step, different types of information are collected from the various devices and retrieved for storage into a central database. The two biggest challenges in the data collection infrastructure are scalability and heterogeneity.

The data collection system needs to obtain information from a large number of devices and components that make up a computer system. In a large enterprise environment, there may be thousands of such components, while in a large telecommunications network, there may be millions of such components. Obtaining the data from such a large number of devices poses significant challenges. Monitoring traffic should not interfere significantly with the operational traffic in the network and should not impact the operation of applications running in the environment. This makes the task of monitoring large systems especially hard.

In addition to having a large number of components, there is a significant diversity in the types of components that may be present in a digital system. Each of the components may produce management information in a different format and manner. Managing the large number of diverse types of devices can add significantly to the complexity of the data collection infrastructure.

In order to address these challenges, data collection systems use a variety of techniques to build a scalable monitoring infrastructure that can handle a large number of element devices. Data collection techniques can be classified along two dimensions – one on the approach used for data collection and the other on the system architecture used for data collection. The approach may either be passive, in which the data collection infrastructure observes the happenings in the system, or it may be active in which the data collection infrastructure sends its own requests into the system in order to retrieve the desired monitoring information. Similarly, a variety of different architectures can be used in various monitoring systems to have a scalable system for a diverse set of devices and software packages.

5.3.1 Passive Monitoring

In passive monitoring, the management system collects the information that is available from the regular operation of the computer system without the introduction of any additional workload that causes the computer system to perform additional work. In passive monitoring, the only extra traffic/work introduced by the monitoring environment is the work required to transmit the generated information to the management console. When an agent is installed on the machine that will transmit the information periodically to the management server, we consider that as a part of passive monitoring as well.

Passive monitoring can be used in any computer system which produces information that can be observed for management purposes. Examples of such information include log-files generated by computer applications, MIB variables available from different types of networking devices and CIM values available through a standard interfaces, and information about the network topology that may be available from routing databases maintained by the networking protocols such as OSPF or BGP during operation. Passive monitoring can provide a significant amount of useful information to the management system.

Passive monitoring can be performed for many different types of computer systems. In the sections below, we discuss some typical methods for passive monitoring of different types of environments.

5.3.1.1 Applications

An application can be monitored passively in several ways. Some applications produce a log file which provides information regarding their internal operations, any errors they encounter, and the type of requests that they are processing. As an example, if we look at the Apache web server, it would produce a variety of log files, such as

- *Server Error Log*: This file will record any errors which the Apache web server encounters. It would list the date and time of the error, a description of the error, debugging outputs from CGI scripts running on the web server.
- *Server Access Log*: The access log records all requests that are received at the server. It usually shows the data and time of a request, the URL that was requested, and the amount of bytes transferred, and can be customized at an installation to provide more or less information on each request.
- *PID File*: The application would typically record its process identity where it is running into a PID file.
- *Rewrite Log*: When Apache is used in a model where some incoming URL are modified by the server to represent another URL at the same site, the information about modifications to the URL is typically recorded in the rewrite log.

Depending on the configuration, the system may produce more than these log files. The different types of log files and their location can be determined by analyzing the configuration file of the application.

In order to retrieve the information from the log files and relay it to the management system, an agent running at the same server where the application is running can read the configuration files and determine the location of the log files. The agent can read the output of the log files periodically, condense the information, and then send them over to the management system. In order to retrieve the last few files, the agent may use programs such as `tail` available on Unix systems which allow the listing of the last lines of a file.

Alternatively, the log files may be read or monitored by a remote application running on a different machine which occasionally logs onto the machine running the web server and collects the output from the log files. The log files can then be sent over to the remote application for processing. Using this approach, the cycles on the machine hosting the application are not consumed to compress of process the contents of the log file. Thus, this approach may be a better one when the application running on the target machine is CPU intensive.

Many applications that generate logs allow for rotation of logs in order to ensure that a single log file does not grow too large. Using log rotation, the application is configured so that it generates a log file for a limited period of time or till the log file grows to a certain size. If the information being monitored is not required to be attained in real time, the log file can be written over once it has been rotated out of use. In other cases, applications allow the flexibility of sending output to a specific process instead of to a file directly. The sending of the output to a process allows for easier rotation of the files. In those cases, a configuration of the application to send the process to the monitoring agent can be used to ease the task of retrieving and monitoring the data.

An alternative way to perform passive monitoring on application is via the use of a proxy. A proxy is a software program which supports the same interface as an application and is interposed in the network between the clients of the application and the application. Proxies can be instrumented to collect a variety of information, e.g., time spent by a request at the server, the nature and content of the request and responses, and other specifics which may not be available or easy to generate as part of the log files generated by the application. Proxies are available for almost all types of applications that tend to be deployed as servers. They are used for a variety of purposes, e.g., as caches to improve response time, to add authentication and access control, or to perform some transformations on the requests before they are handled by the application.

Inserting a proxy between the clients and the application adds an additional latency to the processing of the requests, but does not generate any additional traffic on the network. Furthermore, collecting information at a proxy does not cause any additional workload at the machine running the application itself. Proxies can therefore be used to collect a variety of monitoring information without any significant disturbance to the system. Proxies can send the information to the management system as a client of the management system or can simply store them as local log files which are then read by the management system.

5.3.1.2 Servers, Personal Computers, and Laptops

Servers, personal computers, and laptops generate a variety of management information, which can be stored in various locations on the machines. Each of the various operating systems that are used commonly on the machine provides a variety of mechanisms to retrieve the information. A common form of

such a mechanism is the availability of several commands in the operating system that allow the monitoring of key statistics related to the server.

An agent running on the machine is able to capture several statistics about the operation of the machine and report them to the management system. The agent can invoke the commands to obtain the information, compress them if appropriate, and then transmit them to the management system.

If the system supports the standard model for management information, such as the CIM model, then the agent can invoke the management information using the standard interfaces.

There is one potential problem that can arise when agents are monitoring the information from local servers/machines and reporting them to a central management station. If too many agents try to send the information at the same time to the management station, then many requests can arrive at the management station at the same time, degrading the performance of the management station. In a worst possible case, the agents may become synchronized resulting in an overload at the server even if the server is capable of handling the requests if they arrived staggered in time. Synchronization of agents is not unusual if several machines are powered up at the same time, and their agents are configured to use the same fixed time interval for sending their monitored information to the management station.

One way to avoid the overload at the management station is through the use of randomization. Suppose the management agents want to report the information periodically at an interval of T. The management agents do not configure themselves to send the reports at time period of T, but at the end of every transmission, schedule the next transmission at a time which is selected uniformly between the range of 0 and $2T$. The average interval at which the information will be transmitted will still be T, but different agents will have staggered their sending time so that the overall load on the management station will be spread out.

More sophisticated methods for managing the load on the management station can also be developed. As an example, the agent may monitor the time it takes for its monitored information to be uploaded to the management station. If the time taken to upload the management station takes a longer time than a specific threshold, then the agent would wait longer to transmit the next set of information (e.g., wait for twice the usual amount or choose a random interval between 0 and $3T$ instead of 0 and $2T$). When the amount of time taken to upload the information is close to what would be expected on an unloaded management station, the agent can reduce the time period for the next upload, reducing the time interval between information uploads. By using adaptive schemes of this nature, the agents can avoid synchronization and upload information to management information efficiently.

5.3.1.3 Networks

One method for passive observation of networks is by using the notion of SNMP traps. A trap is a command in SNMP which is sent asynchronously by an SNMP agent to the management system. A network manager can configure a device to generate SNMP traps on specific events. These traps provide a mechanism to obtain information when specific conditions happen.

An alternative method for passive observation of the network is by a management system participating in selected network protocols. By participating in selected protocols, e.g., routing protocols, information about the current state of the network can be obtained.

The most common method used for passive monitoring in network consists of observing packets that are passing through one or more points in the network and gleaning management information from the observation of those flows. Since most network protocols follow a well-known pattern in their communication, the observation of packets can provide significant information about the applications and systems connected on the network.

In order to inspect network flows, one needs to obtain a copy of the packets that are flowing within the network. Such a copy of the packet can be obtained by methods such as the following:

- Some network devices provide the capability of port mirroring. A network device (e.g., a switch or a router) provides interconnection among several other machines and consists of a number of ports which can communicate packets among each other. Some of these devices can be configured so that a specific port receives copies of packets sent or received on another port. When a network administrator has access to such a switch, a monitoring device can be connected to the port that replicates packets on an active port.
- Network taps are devices that can be interposed on a network link to obtain a copy of packets that are sent on the link. They can be viewed as network devices that are dedicated to the task of port mirroring. The network tap can be viewed as a device consisting of three ports, two of which are used to connect two devices that would have been connected originally on the line and the third port used by a monitoring device. Depending on the specifics of the manufacturer, two ports may be provided for monitoring each side of the line, the line packets replicated to more than one set of monitoring devices, or multiple sets of lines and monitoring ports be bundled together to provide a concise method to tap into multiple lines.
- In older versions of Ethernet, packets were broadcast to all computers on the local area network. Usually the receiving station discards any packets not intended for it. In such networks, some receiving stations can be put into a promiscuous mode where they receive a copy of all packets transmitted on the network, regardless of who the receiver was, and pass it up to the software for processing. Modern wireless networks also are broadcast media, and a receiver on a wireless network can collect all packets transmitted on the

network, regardless of its intended recipient. Devices that process and analyze packets in this manner are usually referred to as packet sniffers.
- Some network devices (usually routers) can be configured to store a copy of packets received on specific set of packets flowing between a set of machines or a subset of machines and report them to monitoring system. At the Internet Protocol level, some devices support the IPFIX (IP Flow Information eXport) protocol that allows the collection of such information. IPFIX is a standardized version of previous collection protocols such as Cisco NetFlow or Juniper Jflow. It allows monitoring system to obtain a set of packets flowing between specific sets of machines.

Once the packets from the network are obtained, they can be processed in various manners to retrieve the relevant network information from them. Network packets can be analyzed as part of security management (to check for viruses in traffic or security attacks that may be happening), performance management (to keep track of network bandwidth or system performance), and as part of accounting management (to keep track of who is using the system and network resources).

5.3.2 Active Monitoring

In contrast to passive monitoring, active monitoring requires making explicit measurement requests to determine the information available on the network. Different types of techniques are used to monitor various types of information that is available at the different sources. Active monitoring imposes additional load on the network, but can provide directed information that may be hard or difficult to obtain from passive monitoring alone.

5.3.2.1 Applications

Active monitoring of applications usually is done by means of a monitoring application that generates synthetic load on the application. The synthetic load is dependent on the application being monitored and is usually run so as to not cause an overdue load on the server, or to disrupt its normal functioning.

As an example, one can monitor the response time and uptime of a web site by having a monitoring application that periodically retrieves a page from the web site and records the outcome and latency in obtaining the page. By repeating the process periodically at a slow rate, e.g., once every 5 min, one can keep track of the performance and availability of the web site in a regular manner without generating significant additional load on the site. For other applications, similar type of synthetic transaction generators can be used to obtain the performance of the application for a specific type of transaction.

All computer applications, at an abstract level, can be viewed as manipulators of information, and transactions essentially requests to create, read,

update, and delete (*crud*) the information maintained by that application. The state of an application is the set of information the application maintains and manages at its primary function. Application transactions can be classified into ones that cause a change in the state of the application and ones that do not change the state. Among the *crud* operations of a transaction, the read operation and some update operations do not typically modify the state of the application. The operations of create, delete, and most updates modify the state of the application.

Monitoring an application by synthetic transactions can be performed safely as long as the synthetic transaction is one that does not modify the state of the application. Such types of synthetic workload can be generated so as to maintain a reasonable workload while being able to capture the management information at the right level of detail.

For many applications, it is desirable to monitor the behavior of the system for transactions that modify the state of the applications. There are two common ways to do active monitoring of such transactions.

The first way is to create some dummy pieces of information whose only purpose is to provide targets for synthetic monitoring workload. As an example, let us consider an application that is managing bank deposit accounts and conducting transfers of funds across accounts. One can set up two dummy accounts with some fixed amount of funds and conduct transactions of transferring money between these dummy accounts. While the transactions cause changes in the state of the application, such state change is only among the dummy accounts, and does not affect the accounts of any real users of the bank. The same strategy can be used for monitoring of various web-based applications, where specific URLs are created that are accessed only by the monitoring application to obtain different quantities related to the operational system.

The second way is to test the performance using two transactions, one canceling out the effect of the previous transaction. For example, in the previous case, one can have the first transaction transfer a small amount of funds from the first account to the second, and then second transaction transfer the same amount back from the second account to the first. The second synthetic transaction has nullified the effect of the first synthetic transaction. The two transactions taken together do not modify the state of the application.

The concept can be generalized further. If a combination of N transactions, where N can be any positive number, results in a net no change to the state of the application, then the monitoring system can invoke those N transactions in sequence to obtain information about performance, availability, and proper operation of the system, without modifying the state of the application at all. The monitoring system should complete the entire set of transactions so as to leave the system without any change in state after the set is completed. When planning for such monitoring, the performance generator should also account for the case where a transaction may fail, and the system should have a recovery transaction in case any step of the transaction has an error.

5.3.2.2 Servers, Personal Computers, and Laptops

The monitoring of servers, personal computers, laptops, and other devices that are end points of communications can be done using an active approach. Active monitoring can employ either an agent on the end device or may be done without any agent.

An agent is a software on the device that can collect monitoring information. Such an agent can wait for monitoring requests from the management system and respond to the requests with specific monitoring information that it may have collected locally. The operation of the agent would be similar to that in the case of passive monitoring described earlier in the chapter, with the only difference being that the agent waits for the requests from the management server and respond to those requests, rather than sending out notifications on its own. The management server polls each of the agents on the servers in an order and at intervals that it selects, adjusting its polling frequency and order so as to cause no more than an acceptable amount of load on the network.

Active monitoring using the management station to poll the servers is better than having each agent send information asynchronously to the management system at periodic intervals if one considers the amount of information received at the management system. If the agents are sending information unsolicited to the management system, the lack of coordination among the different agents means that the management system may become overloaded if too many agents happen to be sending their information at the same time. Although the randomization and adaptation techniques discussed earlier reduce the probability of an information overload at the management station, an active polling approach by the management station avoids such a situation from arising at all.

An agent-based approach, however, requires that the agent be installed on the end device before it is deployed in operation. If the agent is a standard feature of the machines when they come from their manufacturer, this requires no effort at all. However, if the agent is a customized one, installing the agent on each machine deployed in an installation can pose some logistics challenges. One could institute a policy where a central IT location installs the agent on each newly acquired machine, running a risk that the central location could become a bottleneck in obtaining new devices, or one can mandate a policy that each new device obtained by anyone install the agent regardless of who procures it, running a risk that the policy may not be uniformly enforced. While either of these or other approaches can be made to work to ensure that the right management agent is present, it does impose an additional complexity to the task of management.

An agent-less approach is more advantageous in that it does not require any additional software installation on the machines. Agent-less management systems would typically have the management station executing a remote script on the device that collects the desired management information. The management system would need to first discover the set of devices that are in the environment, using techniques discussed in Chapter 4, and then logon to those systems

to collect the information. The key issue with agent-less system would be the job of managing credentials that allow the management system to logon to the devices but do not allow any unauthorized user to access the server to obtain the management information.

One solution to the task of credential management is to use a common administrator logon for all of the servers/machines in the environment. That works well as long as the credentials for the administrator logon are not compromised. A compromised administrator password can cause a security nightmare.

An alternative approach is to store the passwords and devices into a directory that is accessible over the network. The management station will then acquire device-specific passwords by querying the directory system, and then use the specified password to access the device and collect the required management information.

5.3.2.3 Networks

Active monitoring of the network infrastructure can be done by polling the SNMP agents that are available on almost all network devices. The agents provide access to all of the information that is available in the MIB format at each of the devices.

In addition to SNMP, two other common methods for obtaining performance and availability of the network include the use to tools like `ping` and `traceroute`. These are tools available on most computers and were intended as convenient utilities to check if another machine on the system was alive. However, they are commonly used as management tools during operations.

The ping command allows a computer to check if another computer is available on the network. The traceroute command tries to find all the nodes that are in the path between the computer running the program and a remote computer. The commands also provide an estimate of the delay in the network between the computer and the remote devices.

By running these commands between selected points in the network, one can obtain an estimate of the delays and loss rates among different points in the network. Further analysis of the overlaps among the paths between the different locations that are pinging each other can reveal more information.

Ping and traceroute have been used in a variety of studies related to Internet topology and Internet performance. The biggest advantage of these programs is that they do not require any administrative privileges to invoke. The biggest drawback of these programs is the inefficiency of the programs. Traceroute, in particular, relies on sending multiple messages into the network and observing when they fail. Therefore, its wide spread usage can have an adverse affect on network performance.

5.4 Pre-DB Data Processing

There are three main objectives of the processing performance at the management system before the information is stored: (i) reducing the volume of information by reducing redundant information, (ii) cleaning the data by removing erroneous or incomplete data, and (iii) converting the information to a format that information will be stored in the database.

5.4.1 Data Reduction

Most large-scale enterprise systems are designed to run uninterrupted for months, if not years, and thus the amount of management information that is collected can become large very rapidly. Therefore, reducing any redundant or duplicate information, or compressing the information in any manner can be very effective in reducing the total cost of managing and maintaining the information.

Before the acquired data is stored into the database, the only techniques that can be used to reduce the volume of data need to be ones that can process the data in real time. As different pieces of the management information are obtained, the reduction operation needs to be done on them in real time on the fly before storing in the database. This type of data reduction is very different in nature than the data reduction required in order to compress and store management information for long-term archival purposes.

Note that the real-time reduction in data can be done in multiple ways so as to incorporate the knowledge obtained from previous instances of the data that is seen. The data reduction process may maintain its own database (usually an in-memory database smaller than the database which will store the management information) to store and analyze the incoming management information. The data reduction process may also look at the past information stored in the database to determine if the new piece of management information is a duplicate of existing information, or for other types of analysis.

The following methods can be used to reduce the volume of the data before it is stored in the database.

Aggregation: Although management information can be obtained periodically from the different devices that are managed, there are many types of information that can be averaged or combined together to create a single instance that needs to be stored in the database. As an example, performance metrics such as buffer occupancy levels at a router or utilization measures at a server can be averaged over larger intervals and the average (or the average and the variance) stored in the database instead of individual measurements themselves. In some cases, it may make sense to average across multiple devices that are logically part of a larger unit. As an example, if a Web site is structured as consisting of five web servers,

the number of requests/second processed at each of the site can be averaged (or summed for the total web site requests/second) and stored at a single entity rather than being stored as five different entries in the database.

Thresholding: For some type of management information, the information needs to be stored in the database only if it exceeds a threshold or given condition. As an example, an administrator may determine that they only want to record instances when the CPU utilization at a server exceeds 60%. In this case, all measurements that are reported to be less than the threshold are not stored in the database. In a system where error events such as lost packets or requests not completed successfully are reported to the management station, the management station may choose to store them in the database only when the rate of alerts exceeds some threshold. Defining thresholds allows the system to ignore the occasional error that may be an acceptable part of system operation focusing more on the errors that occur due to a significant change in the state of system operation.

Duplicate Elimination: In order to reduce the volume of information stored in the database, duplicate and redundant pieces of management information can be eliminated. In some cases, e.g., the failure of an element in the computer system, the failed element or one of its neighbors sends fault alert messages at periodic intervals. In many cases, all but one of the alerts can be suppressed in order to reduce the volume of data. In some instances, one may want to maintain a cumulative count of the number of alerts that have been received as duplicates. In general, duplicates can be detected by defining rules which describe the fields which need to be compared to determine whether two different pieces of monitored information are the same. The rules may also define what actions to take when duplicate messages of information are retrieved.

5.4.2 Data Cleansing

In addition to reducing the volume of data that is collected, an important part of the pre-processing step is to make sure that the data that are collected is valid. Invalid or incorrect data can throw off the analysis of management information that will be performed once it is stored in a database. Data cleansing is the process of validating management information being retrieved from the computer system.

As in the case of data reduction, the main focus of data cleansing in the pre-DB processing is to perform the cleansing in real time before the information is stored in the database. The data-cleansing process may store some of the information in a local in-memory database to perform an analysis of the cleansing operation.

There may be many reasons for errors in the management information that is being collected. The collection process may have failed for a specific device resulting in an impartial or erroneous piece of management information being collected. A device may have been implemented incorrectly and providing wrong values for a specific type of variable being read. An error in the network may have caused an incorrect value of the management information.

The steps in data cleansing are shown in Fig. 5.2. They include (i) tokenizing the management information into fields, (ii) checking for validity of data in each field, (iii) checking of validity of data across fields, and (iv) repairing the field data.

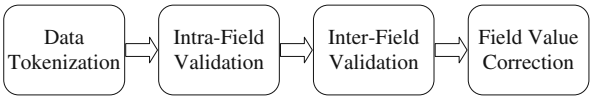


Fig. 5.2 Steps of data cleansing

In the data tokenization step, the collected management information is divided into records of several fields, e.g., a performance measurement may be broken down into a record consisting of four fields – a time stamp, a device identity, a metric identifier, and the value of the metric that was collected. Data tokens are created using the domain knowledge about the specific type of management information that is collected and is usually a simple transformation of information into a set of records.

Once the data are tokenized, the values of each of the fields can be checked to see if they are valid. The following types of checks can be used to determine if the value of a field is correct.

Bounds Check: Usually for each field, there is a range of reasonable values that can be expected. For fields that take on continuous numeric values, the reasonable values may have a range with an established maximum and minimum bound. If the data lies outside these bounds, it is most likely erroneous. For fields that can take on a enumerated value (e.g., an index may only take the values of 1, 2, and 3), a value that is not in the enumeration is an indication of data error.

Statistical Checks: For most numeric data, the measurement of values over time would be expected to show a level of continuity, i.e., its values would be expected to lie around the vicinity of values measured in the past. As part of the aggregation in data reduction discussed above, an online running average of the mean and variance of the field value can be determined. One can then apply Chebyshev's inequality to find out a value which is highly unlikely to occur. If the nature of distribution of the values in the field is known, then better estimates of outlying values can also be made.

Chebyshev's inequality states that for any random variable with a mean of μ and a variance of σ, the probability that the value differs from the mean by

more than k times the variances is inversely proportional to the square of k. In other words, for any random variable X.

$$\Pr(|X - \mu| \geq k\sigma) \leq 1/k^2.$$

The probability that a measured value deviates from its current mean by more than three times its variances is less than 11%. The probability that it deviates from the mean by more than 4 times its variance is less than 6%, and the probability that it deviates by more than 10 times its variance is less than 1%. Thus, outlying values can be detected and flagged as being erroneous using statistical tests.

Other types of data validity checks are related to the relationships that exist between the different fields of the management information record. Inter-field validation can be done by techniques such as the following:

Reasonableness Constraint Definition: A reasonableness constraint is a relationship that should hold among the different fields of the management information. As an example, assume that the management information consists of three fields: a start time, a end time, and a count of packets observed at the router between the start time and the end time. From the description of the management information, one can make the reasonable assumption that the start time should always be less than the end time. Any record that does not satisfy this constraint is likely to be in error. Reasonableness constraints of similar nature can be defined over the various fields of a record and evaluated as new instances of management information records are retrieved. The reasonableness constraints can also cover statistics regarding the averages or other metrics of records seen recently.

Use of Distance Metrics: One effective way to determine if a specific record is reasonable is by defining a distance metric among two records. A distance metric among two records is defined by taking the difference in the fields of two records, and then combining them to come up with a Euclidean distance metric. If it is possible to define such a distance metric, then the metric can be used to determine if a new record deviates significantly from the set of records that have been seen previously.

Error Patterns and Validity Patterns: An error pattern is a relationship among different fields in a record that should never be seen. A validity pattern is a relationship among different fields in a record that should always be seen. If a new record does not show the defined set of validity patterns, or shows an error pattern, it is not valid.

The next stage in the data cleansing process is the hardest one, determining what to do with the information in the records that are found to be invalid. One possibility in some cases is to drop the entire record. If management information is collected relatively frequently, and one is expecting the information in the records to be updated relatively soon, then dropping records found in error

would be a perfectly acceptable solution. However, in other cases where the management information may be less frequent, dropping the management information may not be the best solution.

In most cases, the corrective action to be taken when a piece of data is found to be invalid depends on the specific type of management that is done. In some cases, it may be appropriate to reuse the last valid management information that was received in place of the invalid information. In some rare cases, it may be possible to correct the data so that it can be made valid.

The data cleansing process often makes use of rules or policies that can be used to perform all the steps outlined above in a coherent manner. The rules or policies are expressions of format if <condition> then <action>, where the condition is a Boolean logic expression defined over the different fields of the management information record, and the action provides the modification to the management data record that needs to be made. A modification may change the values of some fields or it may require that the record be dropped.

The specifications of these rules can be used to perform most of the steps in the data cleansing process described above.

5.4.3 Data Format Conversion

One of the main functions in the preprocessing step is to convert all management information retrieved from various sources into a single format which will be used to store the information in the database. Different types of databases may be used for monitoring different aspects of the system, e.g., the performance management database may use a different format than a fault management database.

For each database, a variety of device types, applications, and management protocols (CIM, SNMP, proprietary commands, diverse log formats) will need to be supported. Due to the differences in the commands and standards, and the information available in the standards, there will be differences in the format of management information that will be coming through.

The best way to deal with the different formats of incoming data is to convert them into a common format to be stored into the management database. The number of management databases will depend on the specific needs of an environment, but it would not be uncommon to find a different database for performance management, problem management, security management reports, customer problem tickets, accounting records, and configuration management. For the sake of an example, let us assume that each of the above types of databases have been developed with a canonical database schema.

The task then is to convert management information which are records in many different formats to management information into a single format. The simplest approach in many instances is to write a transformer software for each different format where the transformer format converts the incoming data

format (specific MIB, a CIM model, or a XML input, or a log file format) into the format of records selected for the database. Each of the input formats consists of records with a set of different fields.

The generic problem of format conversion can be stated as the following. Consider N input streams of management information, with the ith input stream consisting of records consisting of Ki fields, labeled ${}^{i}L_1$, ${}^{i}L_2$, $-{}^{i}L_{Ki}$, and an output stream consisting of records with M fields labeled $O_1, \ldots O_M$, we need to find a way to convert each record in each input stream into a different record in the output stream. An example is illustrated in Fig. 5.3.

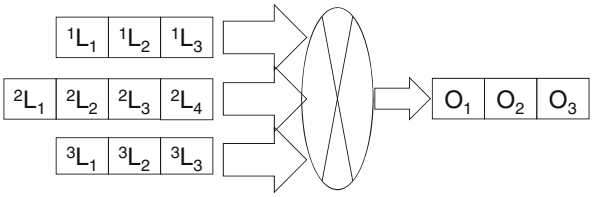


Fig. 5.3 Generic format conversion problem

One approach is to write N software modules, each module translating a record in one of the incoming formats to the format required of the output format. However, with the large number of devices that need to be supported in any environment, the number of data converters that are needed in this approach can increase rapidly.

An alternative approach is to use a generic adapter. Generic adapters can be used when there is a significant amount of shared logic among input formats and rely on using different configuration to customize the shared logic to different input formats. As an example, two of the three input streams in Fig. 5.2 consist of records with three fields. Assuming that the output stream is simply creating new labels on the fields of the records, without making any changes, and one of the fields in the second input stream is redundant, a piece of software that is simply relabeling each of the incoming field would be sufficient to do the conversion into a common format. For each of the input formats, a different transformation will be applied. An example could be

$$\begin{aligned}
&\text{Input Stream 1} : {}^{1}L_1 \rightarrow O_1, {}^{i}L_2 \rightarrow O_2, {}^{i}L_3 \rightarrow O_3\\
&\text{Input Stream 2} : {}^{2}L_3 \rightarrow O_1, {}^{2}L_1 \rightarrow O_2, {}^{2}L_4 \rightarrow O_3\\
&\text{Input Stream 3} : {}^{3}L_3 \rightarrow O_1, {}^{3}L_1 \rightarrow O_2, {}^{3}L_2 \rightarrow O_3.
\end{aligned}$$

The generic software module runs with three input transformations to apply and one trades off the number of software modules with the number of different configurations of the software modules.

In addition to simple relabeling, one can use generic adapters with more sophisticated input configurations. A generic adapter can be based on rules defined matching regular expressions in the input fields to output specific values in the output fields.

If the input format happens to be a standard representation of data, e.g., XML, then tools like XSLT which provide the ability to transform incoming data in a generic manner can be used to perform generic adaptations. XSLT takes a set of format conversion rules and transforms an input XML file into an output XML or other format and allows invocation of external commands that can store the output into a database. Although existing system management information is not commonly input into the XML format, as technologies like web services gain wider adoption, such a transformation may become a common way for transforming management information into a generic format.

5.5 Management Database

The job of the management database is to store the management information into a standard canonical format. As mentioned previously, there will be many instances of a management database, each instance focusing on a different function which is being managed.

While the nature and contents of the databases for each function is discussed in subsequent chapters, there are many aspects of management databases that are common across all of the functions and can be discussed without the specifics of the database schema. These aspects include scalability and data retention requirements of the management database.

The structure of the management database can be seen as consisting of three components – a database component which is usually implemented as a commercial off the self (COTS) database, an access library which is usually specific to the system management vendor, and an information model library. The access library provides access to the database to other components of the monitoring system, which include software components doing both the pre-database processing and those doing the post-database processing.

Most management databases would implement an information model for storing the management information. The information model may be the Common Information Model or its extension, but more commonly is an internal model of the management information that is determined by the system administrators.

The relationship between the different components is illustrated in Fig. 5.4. The relationship assumes that pre-database processing creates the information model objects (prominently write operations), while the post-database processing mostly accesses the information model layer (prominently read operations). However, other approaches, where the post-database processing creates the information model based on raw records created by the pre-database processing are also feasible.

A management database needs to be able to scale in two respects: (i) it should be able to store all the records that will generated over a reasonable life time of operation and (ii) it should be able to efficiently handle the aggregate

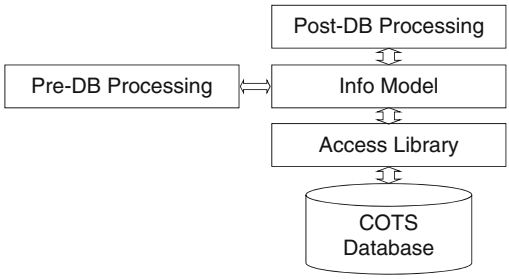


Fig. 5.4 Different logical layers of the management database

throughput of all streams of management information coming for storage into the database.

A unique characteristic of the monitoring function in the systems management database is that its predominant operation consists of adding new records at a regular basis. Furthermore, existing records are rarely updated or deleted (the only exception being where one is using some temporary entries in the management database for data cleansing or data reduction purposes). Some systems management databases would have a slow rate at which the records are added (e.g., database used for configuration management since configuration changes rarely), others would have a moderate rate at which records are added (e.g., fault management – when fault information in a well-designed system should not be too frequent), while yet others would have a very heavy rate at which records will be added (performance management or security management). Nevertheless, most systems management databases will grow in their size over time.

Due to this growth in size, a systems management database needs to be designed so that it will not run out of storage for a reasonable period of time. The first part of the design is relatively simple. If the management database is capable of storing S records and there are k records expected per hour, then the operation period of the database is S/k hours. If we want to design the operation period to be larger than the above, then one will need to use one of the several approaches discussed below.

5.5.1 Partitioned Databases

A partitioned database is one in which a single table of information is split across multiple tables, and information is stored across the different tables depending on the value of a some key fields of the monitored information. As an example, one may choose to define a hash function that maps all the addresses of the devices into a network into a number from 1 to 5. If all database records have a field containing the address of the device that generated the information, the hashed value of the address can be used to select the database

into which the information is stored. Similarly, when retrieving the information from the database, the identity of the device is used to select the database which needs to be accessed. The logic of hashing to the right database can be included in the access library software.

If most of the accesses to the monitored information would tend to follow along the partitioned that is selected, then the partitioning of databases would work well. However, for tasks which require crossing the information in different partitions, one would need to access the information from all the partitions, which can be slow and inefficient.

5.5.2 Rolling Databases

An alternate model for using multiple databases is to partition the database along the temporal axis. In this model, a fixed number of databases are used for storing the management information. Once the storage capacity of the first database is exhausted, information is stored in the second database. The monitoring software progressively uses the next database instance when the previous one has run out of capacity. By the time the last database runs out of capacity, the system would need to have archived over the information from the first databases so that it could be reused again.

Rolling databases provide a fairly good solution for monitoring applications collecting fault, performance, or usage information. Such applications mostly generate time-sequenced records of information, and most analyses are performed on records that are contiguous to each other in terms of time. This sequential nature of the information storage and access makes it appropriate for rolling databases, with the only complexity arising due to the need to work across two databases when the time interval being examined falls partially in one database and partially in another. This complexity is hidden by means of some simple software modules in the access library.

5.5.3 Load-Balanced Databases

An alternative to rolling databases is to have a set of load-balanced databases. While the rolling database provides access to only one database at a time, and thus allows the storage of more records, load-balanced databases provide the ability to access more than one database at the same time.

A load-balanced database system makes sense when the system management data have the characteristics of more frequent reads and less frequent updates. As an example, configuration and topology information of a computer system rarely change, but are required for many different types of analysis engines that are required in various network management tasks and reports. Predominantly read access data can benefit from load balancing.

The load-balanced database is a federation consisting of a load-balancer and several databases. The load balancer acts as a front end forwarding access requests to the databases in the backend. All write requests, i.e., requests that can add or modify records in the database, are sent by the load balancer to a selected database which acts as the master database. The master database records are copied over to the other databases in the system. The read requests for any information are distributed among the different databases that are in the system to balance the load among them.

The overall structure of the system is shown in Fig. 5.5. The access library needs to know which database is the primary (for sending write requests) and which databases are secondary (for sending the read requests to). It could then send the read requests and the write requests to the different set of databases.

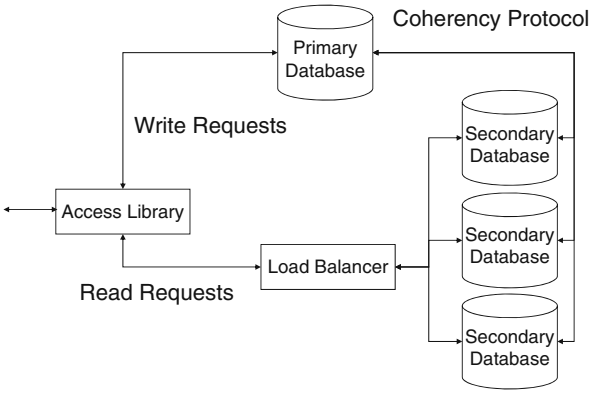


Fig. 5.5 Load-balanced databases

The secondary databases are in effect acting as a cache of the primary database in this model. This leads to a problem of maintaining cache coherency among the different copies on any update requests. There are a few options for ensuring cache coherency. The access library can invoke the write command on all of the secondary databases. Alternatively, the primary database can have a database trigger that will update the contents on each of the secondary databases.

Many commercial databases also provide the option of running multiple instances of the database in a load-balanced mode. In such cases, the access library can be a simple invocation of the database operations since the database will take care of issues related to cache coherency and request routing internally.

5.5.4 Hierarchical Database Federation

In many cases of network and systems management, a hierarchical arrangement can be made among databases storing the monitored management information.

The management database can be viewed as a special partitioning of database information where the partitioning is done on the basis of the geographical distribution of the information. Different databases are used for storing the management information of different areas of a distributed computing system. These databases provide the leaf level (level 0) of the hierarchical management information.

Two or more of the leaf level databases are federated together into a level 1 database. The level 1 database can be used to access any of the level 0 databases using federated database technology. Database federation allows multiple databases to appear as a single database. The level 1 databases can themselves be federated into level 2 databases and so on.

In many cases, the level 1 database need not be a federation of level 0 database, but a new database that aggregates information from the lower level databases. As an example, the level 0 database may collect performance statistics from a region of a network at every 10 min interval, while the level 1 database would have the performance statistics that are averaged over an 1 h interval, but the information is collected from 6 different level 0 databases. In this case, the post-database processing functions would usually monitor the level 1 (or an upper level) database, going to the details of a lower level database only when necessary.

Figure 5.6 illustrates the structure of a hierarchical database system for storing management information.

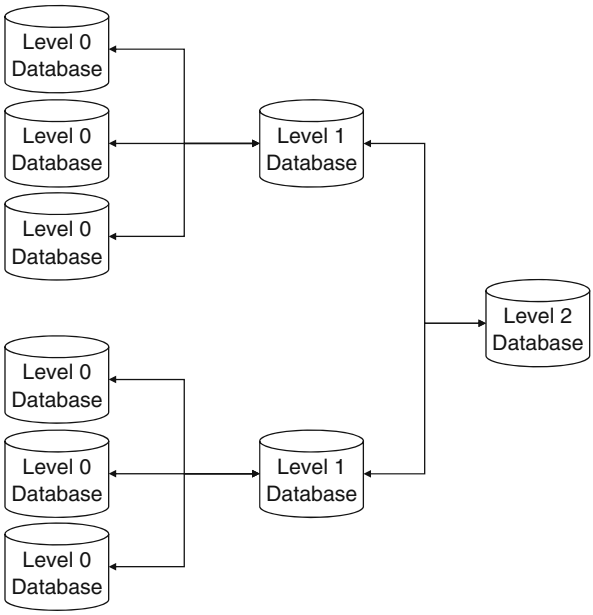


Fig. 5.6 Hierarchical database

5.5.5 *Round-Robin Databases*

Unlike the previous schemes which try to combine multiple relational databases into a federated unit, the round-robin database is a special type of database designed specifically for a special type of data that are encountered commonly in systems management. That special type of data is time-series data, or data that measure the value of a variable over time. Most of network and system performance monitoring data fall into the category of time-series data.

A database that is designed to store information in the manner of a time series can be made more compact and efficient than a general relational database. Furthermore, the database can exploit the fact that the data which is sometime in the past is not necessarily required in full detail and can aggregate the information to save on the amount of storage space required. The tool can discard data that are too old when the database starts to run out of space.

Several implementations of round-robin databases can be found on the Internet as open-source implementations, with most of them implemented with the goal of storing performance management data for systems and network management.

5.6 Summary

In this chapter, we examined the types of management information that are collected at a management station and the different functions that need to be performed on the management data before they are stored into a database. We looked at the various paradigms for collecting network data and issues dealing with the scalability of the management database.

In the next few chapters, we will look at the operations that are performed on the management data stored in the database to perform specific types of management functions.

5.7 Review Questions

1. Why is a database required for storing monitoring information that is collected? Discuss the circumstances under which a system administrator would prefer not to store monitored information in a database.
2. What are the five different components of the generic model for monitoring? What is the key function provided by each of the components.
3. What is the primary technical challenges in the component of data collection?
4. What type of monitoring information can you collect from (a) network devices (b) applications (c) personal computers, and (d) servers.

5. Compare and contrast the benefits and drawbacks of active monitoring and passive monitoring.
6. What are the different techniques that can be used for reducing the volume of monitoring data?
7. Why is data cleansing important for monitoring data? What are some of the ways to perform data cleansing operation?
8. What is the benefit of storing monitored information in a canonical format? What are the drawbacks associated with a canonical format?
9. What is a rolling database? What are its advantages?
10. How does load balancing help in improving the scalability of monitoring databases?
11. For what type of monitoring information is a round-robin database best suited?

Chapter 6
Fault Management

A fault in the computer system is the failure of a component which prevents the computer systems from operating normally. As the computer system operates, it may experience faults due to a variety of reasons. Each fault would generate some type of alerts or error messages to be reported in the monitoring infrastructure. These monitored alert messages will be stored in the management database that is responsible for fault management.

The first section in this chapter discusses a general architecture for fault management. In addition to monitoring for faults, the primary components of a fault management system are the analysis algorithms that process the fault alert data and those that present the fault report to the human administrator. Section 6.2 discusses the different algorithms used for fault management. Section 6.3 discusses some approaches for automating the task of fault management.

6.1 Fault Management Architecture

Any fault management system operates around the two basic abstractions of *root cause* and *symptom*.

Root Cause: A root cause is the occurrence of a specific type of fault. The root cause could be the failure of a node in the network, a disruption in the network, or the faulty configuration of a software application package. The situation is the underlying problem or fault in the network or computer system.

Symptom: A root cause in the computer system is rarely observed directly. The occurrence of any root cause causes different elements in the computer system to generate fault messages and other diagnostic information. These are the symptoms that can be observed directly by the management systems.

The task of the fault management system is to collect the different symptoms from the different element in the computer systems and analyze them to determine the root cause.

D.C. Verma, *Principles of Computer Systems and Network Management*,
DOI 10.1007/978-0-387-89009-8_6, © Springer Science+Business Media, LLC 2009

Due to the wide variety of devices and software packages that can be found in a computer system, and the lack of a widely accepted standard, different devices and software packages generated different types of diagnostic information when a fault occurs. In order to deal with this diversity of error information, fault management systems would define their own canonical models for the fault information. All diagnostic information from different devices will be converted to this canonical model. The process for converting, collecting, and monitoring the variety of diagnostics information into the canonical model is described in the previous chapter on monitoring.

To keep the distinction between the raw diagnostic information and their canonical representation, we define the *symptom* as the information that is represented according to the canonical model, and *events* as raw diagnostic information that is generated by the devices.

Although there is no commonly accepted architecture which is used within the systems management industry or a well-recognized standard for fault management, one can use the generic architecture shown in Fig. 6.1 to study the area of fault management in general.

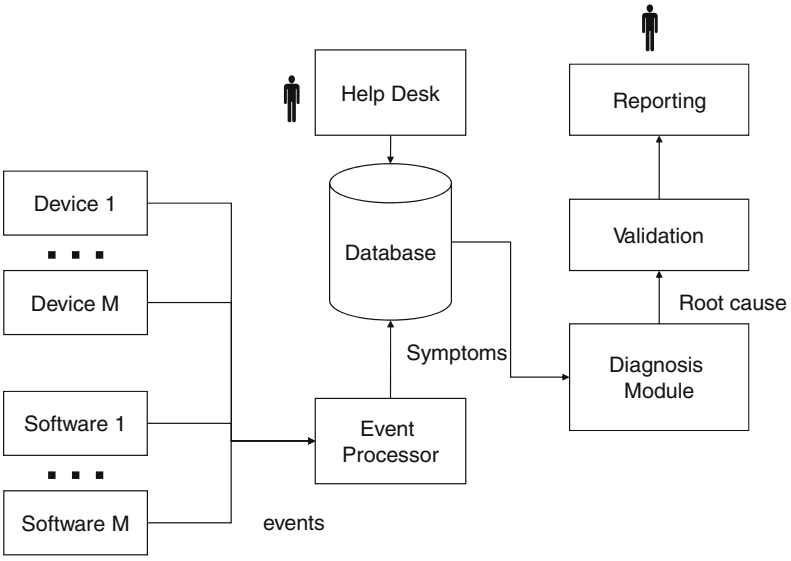


Fig. 6.1 Architecture for fault management

The general architecture for a fault management system has an event-processing infrastructure to monitor the events and convert them into symptoms. The symptoms are usually stored for persistence in a database. The symptoms in the database are analyzed by a diagnosis module that tries to infer the root cause that may have generated those symptoms. A validation step after the diagnosis

performs tests that could verify that the diagnosed root cause has indeed happened. The diagnosed and validated root cause is then reported to the administrator.

Sometimes, the diagnosis module may not be able to determine the root cause completely, and only be able to reduce the volume of symptoms, or advise the administrators about the different possible root causes that may have generated the various symptoms the management system has seen. The administrator can then use the fault correction infrastructure, usually a configuration management system, to take remedial actions to deal with the specific fault.

In addition to the symptoms generated by the devices themselves, problems in operation of the system may be reported by the help desk. The help desk is a mechanism offered by IT departments of many computer system operators so that the users can call and report any problems they may be experiencing. As part of the process, the systems in help desk may feed in symptoms or error event information to the fault management system.

In many cases, it is possible to automate some of the tasks that the administrator does and have an automated system take remedial actions to correct the cause of the fault notifications. Building such an automated system is the goal of autonomic computing technology, described further in Section 6.4.

6.1.1 Common Types of Symptoms

Many components of a computer system can be modeled as operating under a variety of states during their lifetime. A useful state transition model that can be used for understanding the nature of symptoms that are generated in fault management is shown in Fig. 6.2.

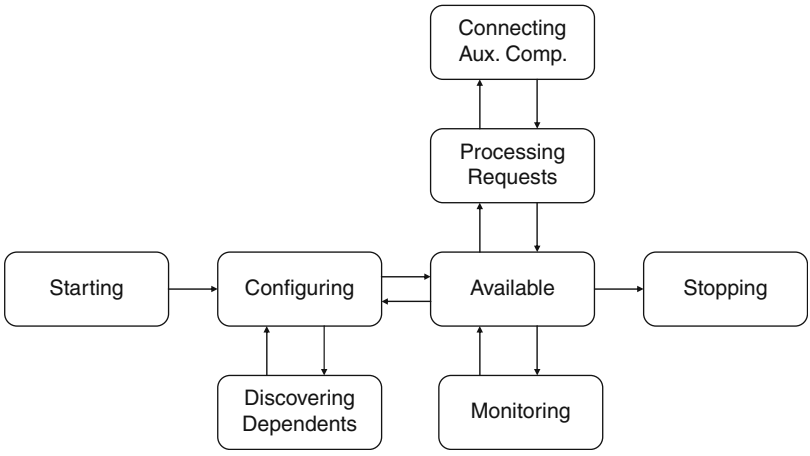


Fig. 6.2 Component state transition model for fault management

A component in the computer system could be a hardware or software component that is operational. As shown in Fig. 6.2, a component begins its operational life by entering the starting stage shown in Fig. 6.2. The component may enter the starting stage because it is powered on or when it is started.

After performing the initial start-up operations, the component will enter the configuring stage, where it attempts to read the configuration information from a file or another location. It then moves into the configuring component.

As part of the configuration step, the component may often need to find other components on which it may be dependent for operation. A device may be dependent on discovery and locating a DHCP server before it can complete its configuration. An application package may need to find other packages and components that it is dependent upon in order to perform its functions. This is shown by the "Discovering Dependents" stage in Fig. 6.2.

After the system has configured itself, it enters the available stage. In the available stage, the system is performing its intended task, whether it is processing web requests, forwarding packets, or completing database transactions.

In the available stage, many systems keep on periodically monitoring other components in the computer system through heart-beat or keep-alive messages. Heart-beats/keep-alive messages are messages exchanged between two components on a periodic basis to check if the other component is alive or not. In most software system implementations, the monitoring may be done concurrently by means of an independent thread within the system that runs periodically. When a component notices that one of its partners that it is supposed to maintain a heart beat with is not responding, it can report that to a fault management system.

Similarly, when a request is received, a component (or a sub-component within the component) will process the request. In order to process the request, the component may need to connect to, or dispatch part of the request processing to an auxiliary component. These mark the processing stage and the connecting stage shown in Fig. 6.2.

As the component enters and leaves the various stages, it generates diagnostic messages that result in different symptoms. Depending on the amount of diagnostic information configured for the component, the system would generate various types of events that are described below. Note that the events were the raw diagnostic information produced by each component, and the same categorization will hold for symptoms that are the events converted to a canonical model for representation.

Starting Events: These events indicate that the component has started.

Stopping Events: These events indicate that the component has stopped.

Configuration Events: These events indicate that the component has configured itself properly on the basis of some configuration parameters that are described, or any error messages in case one or more of the specified configuration parameters could not be supported or were incorrectly formulated.

Dependency Events: These events described the status of other components on which this component is dependent. It may provide information about whether the component with dependency was successfully located and communicated with, or whether that dependent component was not found. As an example, a workstation may report that it failed to reach the DHCP server, or an application server may report that it failed to load a software module which was essential for its operation.

Availability Events: These events describe that the component has moved into the available state and is ready for normal operations.

Request Events: These events describe the status of processing an incoming service request, e.g., whether the request was processed properly, or an error was encountered in processing an event.

Connect Events: These events describe the status of connecting to another auxiliary component when processing a request.

Each of these event types has its corresponding symptom types, which are only the conversion of the events to a canonical representation.

In the database storing the symptoms, each stored symptom would identify its type among the above, the identity of the component that generated the symptom, the status or error information, the identity of other subcomponents that may be involved, etc. One example of a detailed model description for different types of symptoms can be found in the common base event format definition of IBM Autonomic Computing initiative [1, 2].

6.1.2 Common Types of Root Causes

Just like the categorization of the symptoms, many root causes that happen in a failure of the computer system can also be classified into different categories. Some of the typical root causes, which disrupt the normal operation of a computer system, are listed below:

Infrastructure Failure: The operation of a computer system depends on the availability of an infrastructure which includes an operational facility where the computer system is located, a functional power supply to facility, and access to that facility for employees. In case of any outages or disruptions in the facility electric supply, an entire subsystem of the computing environment may disappear. While rare, such outages need to be detected and accounted for.

Component Failure: In contrast with the relative rare failure of the infrastructure, which may cause the failure of a large aggregate of components in a geographical area, the failure of a single component is the more common occurrence in modern computing environments. A failed component may be a device, a communication link, or a software package installed on a device. Components may fail for a variety of reasons,

including power glitches, an accidentally pulled out plug, an incorrect operation by a human operator, failure of hardware due to age, or due to a software bug. A component failure causes symptoms to be generated from any other components that may have been monitoring the failed component with a heart beat.

Component Restart: Many components in a computer system are designed to automatically restart after failing. The time taken by a component to restart varies widely, and the management system may receive various fault notifications while the system is restarting.

Component or System Reconfiguration: Many components and systems are designed with resiliency and reconfiguration in mind. The system would reconfigure itself in order to work around the failures. As an example, when a link fails in the network, the routing protocols automatically try to reconfigure themselves to route packets around the failed link. Such reconfiguration can take some time to stabilize. During the reconfiguration time, the system may generate transient fault events. In some cases, the reconfiguration process does not complete, and the system enters a mode where it is continuously reconfiguring itself. Such a state can cause a more persistent set of fault events which would need administrator intervention to correct.

Component Misconfiguration: Sometimes, a specific component in the system may be misconfigured. A misconfiguration may put the value of a system configuration parameter to be erroneous value which may cause the system not to operate at its optimal point. If a misconfiguration causes the system to not function at all, it can be noticed readily. On the other hand, if the misconfiguration is causing a degradation in performance, it becomes a harder problem to detect. Quite frequently, a misconfiguration will result in error events under some conditions, and not under some other conditions.

Software Bugs: With the ever increasing complexity of software systems in use today, the total number of defects that are present in a computing environment increases despite advances in software engineering paradigms. A bug is an error or defect in the software. A buggy software can cause the failure of the system in various ways, resulting in different types of fault events.

Improper Requests: In some cases, a computer system or a component may fail due to a wrong type of request being dispatched to it. In many cases, an improper request may be generated due to a misconfiguration, e.g., a web server may be sending database transaction requests to the wrong server. In other cases, an improper request may be caused due to external factors, e.g., there may be a sudden increase in the amount of requests being sent to a component due to fluctuations in demand, or someone may be trying to target the system for a denial of service attack. The presence of improper requests will cause fault events to be generated, which need to be given proper attention by the administrator.

Each of the above categories of root causes need to be inferred from the different symptoms that are monitored into the database. In the next section, we look at some of the common algorithms that allow the inference of root cause from the symptoms.

6.2 Fault Diagnosis Algorithms

The effectiveness of fault management is heavily dependent on the algorithms that are available for diagnosing the root cause of the system. In this section, we look at the different algorithms that can be used for this task.

The fault diagnosis problem can be defined as the following. Given a time-ordered sequence of symptoms $s_0, \ldots s_n$ that emanate from different components of a computer system, with the symptom s_k being generated at time t_k determine the set of root causes $c_0, \ldots c_m$ and a set of times $t_0, \ldots t_m$ such that t_k is the latest time that cause c_k may have happened and the set of root causes is the best likely cause of the symptom sequence.

A root cause needs to happen before the first symptom generated by that root cause is observed, but since there is an unpredictable delay between the happening of a root cause and the time when the first symptom is noticed by the management system, the best one can do is to determine the latest time when the root cause may have happened, and sometimes a bound on the earliest time when the root cause may have happened.

In all distributed computing systems, clocks are not synchronized, and thus there is a inherent fuzziness about assigning a time to when an event happens. While an interesting intellectual exercise, the problem of unsynchronized time is rarely an issue in practical systems management. There are two reasons for that, first that the time measured at the management system is usually considered for the majority of the measurements and second since most faults and other type of system management-related events occur at a relatively infrequent stage, an error introduced by lack of perfect synchronization among clocks is relatively insignificant.

In the next sections, we look at some of the techniques that are commonly used to determine the root cause of a failure from the observed symptoms in systems management. These algorithms are also known in literature as problem determination algorithms, fault isolation algorithms, fault localization algorithms, and root cause analysis algorithms. Not all of the methods described below are applied uniformly in practice. Practical systems tend to use diagnosis methods that tend to be simpler and scalable for use in large-scale computer systems, and make fewer assumptions regarding the behavior of the system. The more theoretical approaches, however, allow for better performance or more accurate diagnosis under some set of assumptions.

6.2.1 Topology Analysis Methods

The first approach to determine the root cause of a set of symptoms is to exploit the information about the interconnection among the different components of a computer system. By studying the relationship between the topology of a computer and the different alerts that are being generated, it is possible to determine the root cause generating different alerts.

As an example, let us consider the operation of a network where some applications are keeping track of each other by means of heart-beat messages. Each node reports the failure of the partner if it fails to receive a response from the partner for some fixed number of consecutive heart beats. If multiple pairs report a failure of their partners in about a given range of time period, the management system can compare the paths between the pairs reporting a failure to communicate to determine the node that may have failed and thereby disrupted communication.

One common case of heart beat is within the computer networks where routers keep track of their immediate neighbors by means of heat beats. If several neighbors of a node report that the node is not responding to the heart beat, it is an indication that the node is down.

The use of the graph topology to check for failures can be used for nodes that are maintaining heart-beat messages with others across a large network. In some cases, the heart-beat messages are maintained automatically by the routing protocols and applications that run in the network. As an example, the Border Gateway Protocol (BGP) is used by network operators to connect with other network operators, and all BGP routers within a single network domain would typically exchange heart-beat messages with each other. Message queue middleware such as IBM MQseries also maintain regular heart beat among the different instances of the software. In other cases, the heart-beat monitoring can be done by special monitoring applications whose only task is to send periodic messages to other monitoring boxes within the network. Thus, a network operator may opt to put in a few additional boxes in the network whose primary task is to send probe messages (e.g., using ping messages periodically) to other boxes and raise an alert when the communication to another monitoring device is disrupted.

The general algorithm for detecting the root cause of failure using topology-based schemes can be described as follows. Start with the list of all the node pairs that have reported a failure to communicate with each other. Enumerate the paths, i.e., identify the nodes that lie on the route used to communicate between the two nodes. Compile a list of such nodes and remove from the list any nodes that lie along the path used for communication between two nodes that are still able to successfully communicate with each other. The remaining nodes are the ones that could have potentially failed causing a disruption in the communication.

As an example, let us consider a network of an ISP (Internet Service Provider) who runs four instances of a monitoring service that maintains an internal

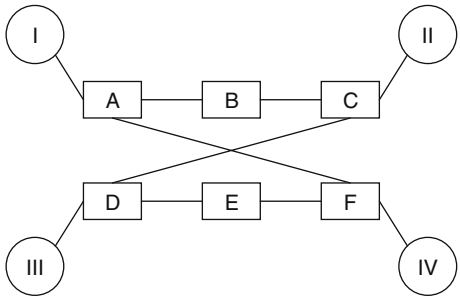


Fig. 6.3 Example of root cause analysis using topology

heart beat with each other and report failures to the management system. If the topology used within the network is as shown in Fig. 6.3, where the circles show the monitoring applications and the squares shown the routers, the management system can use the failure indicators received from the different nodes to identify the router that may have failed.

Let us assume that the management system receives an indication from monitor I that it is unable to communicate with monitor II. Let us assume that the other monitors are able to communicate with each other. The management system can infer from the message notification that monitor I cannot communicate with II that one or more of the routers A, B, C, or monitor II has failed. From the fact that monitors I and IV can communicate with each other, the management system can infer the fact that router A has not failed. Similarly, using the fact that monitor I and II can communicate indicates that monitor II or router C have not failed. This leads to the inference that router B is the one that has failed causing monitors I and II to not be able to communicate with each other.

In a real network, a topology-based analysis of the paths that may have failed may not necessarily result in identification of one node that may have failed, but result in identifying a set of nodes that may have potentially failed. The management system would then try to communicate with each of the nodes in that set to determine the one (or ones) that may have failed.

The topology analysis method can be used not only for networks but for computer systems where the structure of the system can be represented by means of a graph. This can include the detection of faults that occur in the operation of a data center with multiple tiers of applications, or even within a single application where the relationship among different packages can be represented by means of a call graph.

As an example, let us consider an email application that is set up according to the topology shown in Fig. 6.4. An installation uses a spam filter to check for the validity of the email that it receives from external Internet. The span filter accesses a domain name server to validate the address of the sending party and forward validated emails to the SMTP (Simple Mail Transfer Protocol)

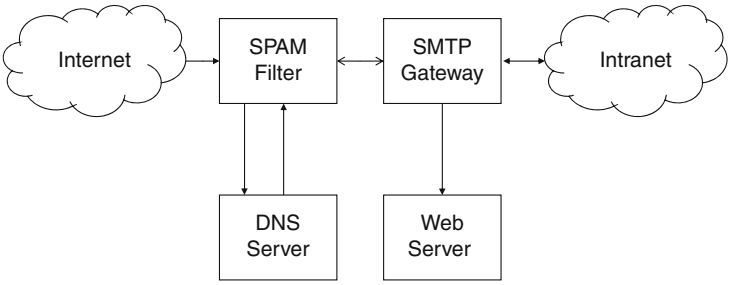


Fig. 6.4 Root cause analysis scenario

gateway that makes the mails available for the users through a web server. The path taken by the mail messages from the Internet to the web server is as shown by arrows in Fig. 6.4.

If a user calls to complain that they are not able to access email sent to them from the Internet, but the users from intranet are able to access the email without a problems, the cause of the failure is in the nodes that lie along the path taken from the Internet to the web server, but not in the path from the Intranet to the web server. The two potential causes of failure then could be the SPAM filter or the DNS server. The management system can check the status of both of these two applications to check which one may be at fault.

As another example of a topology-based approach to identifying the root cause, let us consider the help desk of a cable distribution network. The cable distribution network receives complaints from various subscribes that they are not getting the reception of the television signal. The management system of the cable television can identify the path from the cable head end that transmits the television signals to the different receivers and identify that the bulk of problems are caused by a common node that is shared by all the customers complaining about the problem in the same neighborhood.

Topology-based algorithms provide a good way of identifying potential causes of failures in various systems. They can handle the problems of fairly large networks since graph algorithms that can scale efficiently are well researched in the scientific community and provide an effective means for identifying the root cause or narrowing it down to a small number of potential root causes. They are commonly used as standard practice in several telecommunications network operators and cable network operators.

Topological methods used to identify root cause of faults tend to be highly customized and specific to the network and environment for which they are developed. For a specific computer system environment that is being managed, the operators need to determine the set of root causes that may need to be diagnosed, and then write customized algorithms that are applicable in their environment.

6.2.2 Rule-Based Methods

A rule-based method is an alternate approach to diagnose the root cause from the symptoms that are seen in the fault management database. Unlike the topological approaches, which require customized algorithms for each environment, the rule-based method consists of a software (the expert system) which takes as input a rule base. A rule base is a set of assertions that in the context of root cause identification take the format "if < symptom > then <action>", where the <action> part could either identify a root cause, or cause a derived symptom to be generated. A derived symptom is a symptom that is created synthetically and can be used to derive a root cause eventually.

An example of a simple set of rules that can be used to diagnose some of the common problems that one encounters in troubleshooting Windows XP devices are presented below. The list below shows a subset of rules that looks at the error code (a symptom) that is generated by the device driver and infers the root cause accordingly.

- if error code = 18, then "driver is installed incorrectly"
- if error code = 19, "information in registry is corrupt"
- if error code = 28, "driver does not have required software library"
- if error code = 14, "device needs computer restart"

In the rules given in the example above, no inference of any type is necessary and each error code (symptom) can be mapped directly to a root cause. A set of rules above can be implemented using a table look up instead of using any expert system. A large number of rules used in fault management can indeed be represented as simple table lookups.

In some cases, the rules may require more deduction and logic instead of being a simple lookup as in the example above. As an example, a rule set may look like the following:

- if symptom = AppConfigIniting, then ApplicationInitializing
- if error code = AppConfigReadError and ApplicationInitializing, then ApplicationRestarting
- if error code = AppConfigReadError and ApplicationRestarting, then ApplicationMisConfigured.

The above set of rules can be used to ignore transient errors in the initialization of a application. The root cause that an application has been misconfigured is only diagnosed after two read error messages are obtained when the application is initialized. The first configuration read error message may have been received due to some type of time-out, and the fault management system waits for the application to restart itself and try to reload the configuration again before declaring that it is hung.

Rule-based systems are useful in diagnosing problems and faults in the network and can be customized readily by adding new rules to the existing

body of fault management rules. The main problem with rule-based systems is the task associated with creating and compiling the set of rules. As the number of rules grows larger, managing the rules and drawing inference from theme becomes significantly harder.

6.2.3 Decision Trees

Another method used for diagnosis of faults encountered in the management system is the use of decision trees. A decision tree consists of several nodes which are arranged in a tree structure. At the leaf nodes of the tree, a root cause is identified. At any of the non-leaf nodes, a series of diagnosis tests are performed, and the outcome of the diagnosis test determines which next node of the tree ought to be examined.

A fragment of a decision tree is shown in Fig. 6.5. In the figure, the decision tree initially checks whether the symptom being analyzed refers to one of the three domains of networking, end systems, or storage systems. Depending on the answer to that question, it checks whether the symptom refers to a wired LAN or a wireless LAN. Depending on the outcome, the next set of decisions on the symptoms is made. By progressing further in this manner, the decision tree would result in a root cause.

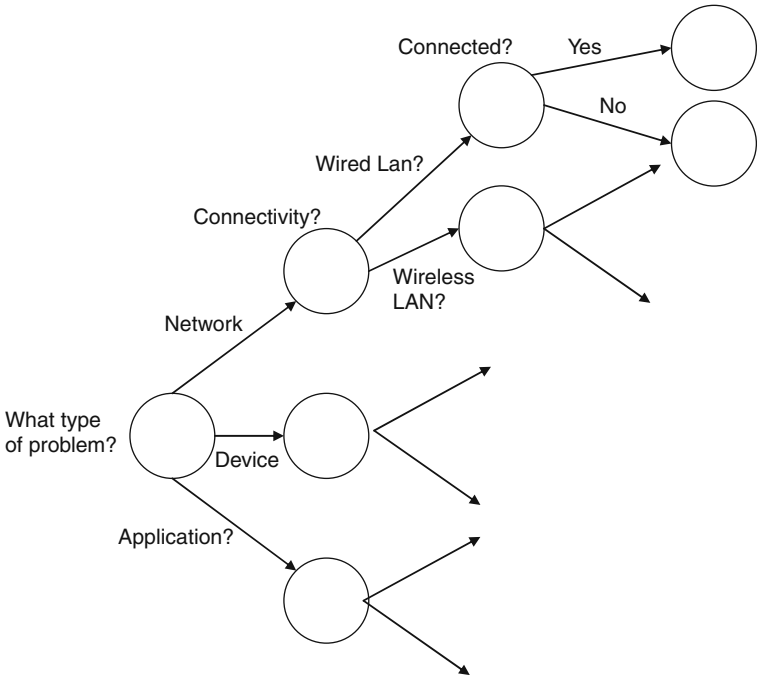


Fig. 6.5 A decision tree

The checks performed at a node in a decision tree need not only look at the symptom that is stored in the database. They can also reach out to the element being managed and perform specific tests that can diagnose the cause of a failure. The decision tree can also be extended to perform corrective actions at a leaf node once the root cause has been identified.

6.2.4 Dependency Graphs

An alternative approach to solve the root-cause diagnosis problem is to create a dependency graph. A dependency graph is a graph that shows the relationship between the root causes and the symptoms that may be generated by a root cause. Given a dependency graph, the problem of root-cause diagnosis can be reduced to that of identifying the root-cause problem which is associated with an identified symptom.

In the simplest case, the dependency graph is going to be a bipartite graph matching a root cause to one or more symptoms. Given a set of symptoms, the root-cause diagnosis problem is that of finding the set of root causes which can completely explain the generation of all of the symptoms. If the dependency graph is such that each symptom is associated with only one root cause, then the diagnosis problem can be solved simply by finding the corresponding root-cause which generates that symptom. Such a simple dependency graph is shown in Fig. 6.6.

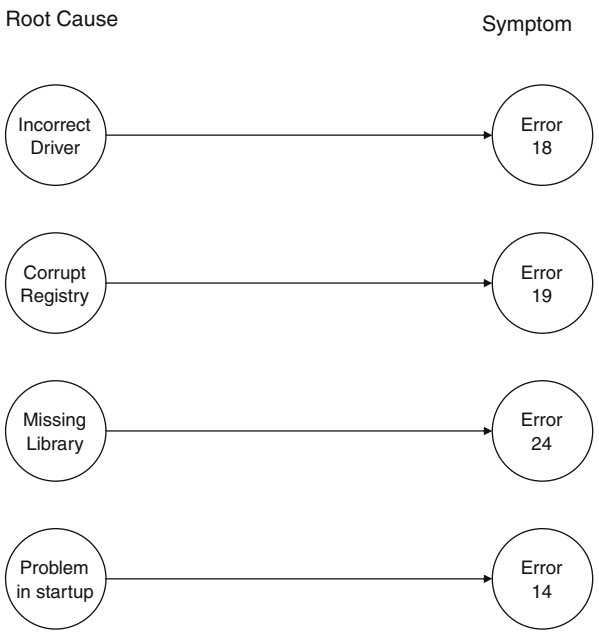


Fig. 6.6 Example of simple dependency graph

However, when a dependency graph is formulated, a symptom may be associated with more than one root cause. A root cause may generate more than one symptom and a symptom may be generated by more than one root cause. Figure 6.7 shows the generic case when a dependency graph does not have a simple relationship between the root causes and the generated symptom. In these cases, the task is to determine the set of root causes which are responsible for generating a set of symptoms that are noticed by the management system. The challenge is that since one symptom can be generated by many different root causes, additional information is needed in order to distinguish the exact root cause which generated a specific symptom.

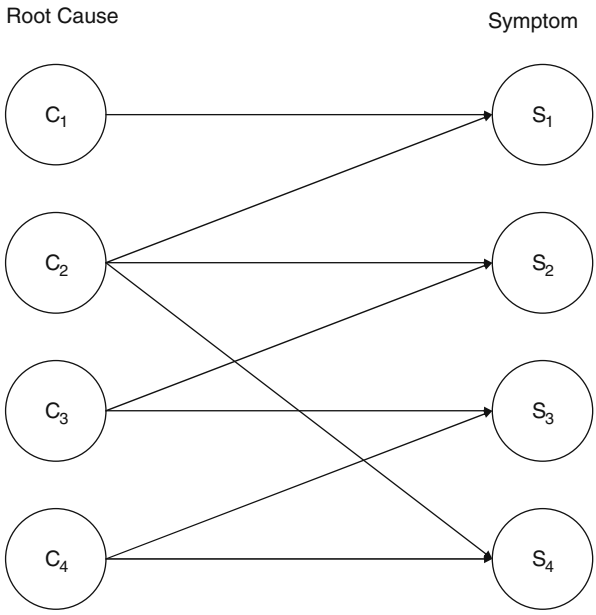


Fig. 6.7 Example dependency graph

If only one symptom is examined at a time, it is hard to determine the root cause when the same symptom may be generated by more than one root cause. However, when looking at a combination of symptoms, it is possible to disambiguate the root cause. Looking at the set of symptoms in Fig. 3.7, we can see that root cause c_1 only causes the occurrence of symptom s_1; root cause c_2 causes an occurrence of symptom s_1; s_2; and s_3; root cause c_3 causes an occurrence of symptoms s_2 and s_3; and root cause c_4 causes an occurrence of symptoms s_3 and s_4. Thus, if we are consider the set of symptoms that are received close to each other, the occurrence of any element of the set $\{ s_1, \{s_1, s_2, s_3\}, \{s_2, s_3\}, \{ s_3, s_4\} \}$ can be mapped exactly to one of the root causes. In effect, the combinations of the different symptoms occurring together are viewed as a derived symptom,

which matches the derived symptom to one of the root causes. If the symptoms are not lost, then all symptoms should be mapped to one of the above groups.

The dependency graph may even be non-bipartite, in the sense that a root cause may not directly generate a symptom, but it may cause the occurrence of some other event in the network which causes a symptom to be generated. In other words, dependency graphs linking root causes and symptoms may have intermediary nodes which are secondary or derived symptoms. If there is a path in the graph from the root cause to the symptom, then the symptom may have been generated from the root cause. We can then convert the dependency graph into a bipartite set, with a root cause connected to a symptom if there is a feasible path in the network from the root cause to that symptom.

The main challenge associated with using a dependency graph is that of generating the dependency graph for the system. Since systems fail in different manners, and software releases in most devices tend to be upgraded fairly frequently, creating and maintaining dependency graphs for a system can be a laborious process. As a result dependency graphs used in practice may be quite incomplete. Another process associated with the use of dependency graph is the fact that the process of collecting symptoms from the computer systems is not error-free, and it is quite likely that some symptoms are lost during the collection process.

In the next section, we look at one technique that is used commercially to deal with the issue of lost symptoms.

6.2.5 Code Book

The code book technique represents the symptoms and root cause bipartite graph as a binary matrix, with the rows representing the root cause and the columns representing the symptom. Equivalently, the rows can be used to represent the symptoms and columns to represent the root cause. Each row of the graph now represents a binary number consisting of 1s and 0s, with a 1 in the location marked by the symptoms that are generated by root cause correspond to that row. Each such binary number is referred to as a code. Each dependency graph would represent a set of valid codes that could be the ones generated due to a root cause, and a set of invalid codes that could not be generated according to the dependency graph. In the example shown in Fig. 6.7, the valid codes represented as s_4,s_3,s_2, and s_1 are 0001, 0111, 0110, and 1100, respectively, for each of the root causes.

The codebook technique looks at the code represented by the symptoms that are received and uses that to determine the root cause. If the code is a valid code, then the root cause corresponding to that code is the one selected. However, due to losses in the network, an invalid code may be received in some instances.

In order to account for the differences, a distance metric is defined on each of the codes. One distance metric can be defined to be the number of positions

which are different in two codes. This distance metric between two binary numbers is known as their Hamming distance. When a new set of symptoms are received, and it does not belong to one of the known codes, a valid code with the smallest Hamming distance to the received code is assumed to be the one that was generated.

As in the case of dependency graphs, the main challenge associated with the codebook technique is the creation and maintenance of the codes.

6.2.6 Knowledge Bases

The proper operation of each of the root cause analysis algorithms described above requires access to some type of knowledge. The knowledge may be the set of rules required to do the diagnosis, the description of a dependency graph, or the description of the codes in the codebook. The main difficulty associated with all of the algorithms was the effort required to maintain that knowledge current. A repository of the knowledge required for any type of operation is known as the knowledge base.

A knowledge-base may be used by an automated algorithm for root-cause analysis or for manual diagnosis of the root cause. The Internet may be viewed as a very convenient knowledge-base for the purpose of root-cause analysis. When a symptom in the form of an error code is seen from any component of a computer system, an Internet search for that error code using any of the leading search engines can identify the root cause and other details about the error, including mechanisms to fix the root cause of the error symptom.

In addition to the Internet, knowledge-bases providing information regarding potential errors related to their products are maintained by the manufacturers of many hardware and software products. The knowledge bases maintained by the manufacturers may be accessible on the Internet, either publicly or limited to registered customers of the products. The knowledge-bases maintained in this manner often allow customers to report the problems they have encountered, the steps used to fix the problem, successful as well as unsuccessful, and thus allow others to benefit from the experience learnt in the past.

Software programs can be used to ease searching through the knowledge-bases of various sites that may be required to be scanned to understand the different symptoms that are encountered by the management system. Such software systems, e.g., the IBM Support Assistant [3], allow users to query knowledge-bases from different locations, as well as enable users to upload information about their specific symptoms and root causes to be shared by other users of the system.

While the most common prevalent forms of knowledge-bases are intended to provide information to the human administrator, similar techniques can be used to provide information sharing for using automated root cause analysis

software modules. In order for such sharing to work, the knowledge-base would need to represent information in a structured manner so that the software modules can interpret and process the knowledge that is stored.

6.2.7 Case-Based Reasoning

One specific approach to root-cause analysis, which is commonly used in human-centric knowledge bases, is using case-based reasoning. The basic idea in case-based reasoning is to solve a problem using solutions used to solve similar problems encountered in the past. If a symptom was diagnosed previously as being generated from some basic root cause, then the same root cause can be assumed to be the cause when the symptom is seen again. Case-based reasoning techniques have been used successfully in customer help desks [4].

In the case of helpdesk support and similar applications where human administrators have the task of determining the root cause of a reported symptom, case-based reasoning relies upon the approach used to solve similar problems before. In helpdesk systems, it is common to create a trouble ticket to make a record of a problem reported by a customer. The trouble ticket is updated to store attempts to resolve the problem, any diagnostic tests that are run for the trouble ticket, and the final diagnosis of the root case and the method used to address the problem. When a new trouble ticket is opened, a case-based reasoner looks through the previous set of trouble tickets to see if a problem of similar nature had arisen previously, and the resulting diagnosis that was used for the same. If a match is found, the help desk staff can attempt the same diagnostic steps on the new trouble ticket as well.

When case-based reasoning is applied to the problem of root-cause analysis, a case may be defined by the set of symptoms that are received from a set of components in the computer system. In addition to the set of symptoms, the nature of type of the component that generated the symptoms are used as other parameters that define the case. In order to use the case-based approach successfully, the results of a human administrator to diagnose the root cause need to be attached to the set of symptoms that are defining the case. An automated system can subsequently look through the different past cases to see if a new set of symptoms can be mapped to the existing case.

The main advantage of a case-based approach is that it allows the knowledge-base to be built iteratively over time as problems are encountered in the operation of a system. The main challenge of case-based approach, when applied in the context of computer administration, is to ensure that the results of a previous investigation are stored and attached with each case that is solved. It usually requires discipline and effort on the part of administrators to keep recording the results of previous solutions in a computerized database and maintaining that discipline in an environment when problems frequently arise is a difficult task.

6.2.8 Other Techniques

Case-based reasoning is a research topic in the broader field of artificial intelligence which can be used to advantage in the domain of fault management. The use of rule-based systems is another such instance. Due to the importance of the subject of fault management and root-cause analysis, many other techniques for fault management based on topics in the field of artificial intelligence and other fields of computer science have been proposed in academic literature. However, due to challenges in the implementation and development of such systems, they have not seen a substantial adoption in practical fault management systems.

One such technique is the use of Bayesian logic to identify the root cause of a symptom. This method is applicable when a symptom may be generated by multiple underlying root causes. The Bayesian logic-based approaches assume that the probability of generating any symptom when any root cause occurs is known. A formula from probability theory, Bayes' formula, can be used to compute the probability that the symptom was generated from any possible root cause. One can now calculate the root cause with the maximal probability of occurrence. The challenge in applying such approach lies in the difficulty of estimating the probabilities of the occurrence of a symptom – which is hard to determine in any practical computer system.

Another technique used for fault management is that of belief propagation networks. A belief is a hypothesis or assertion about a particular root cause occurring, and the techniques use a variety of hypothesis testing schemes to validate the statistical likelihood that a specific root-cause event may have occurred. The difficulty in using such techniques lies in getting sufficient data during the operation of a real system so that the hypothesis testing techniques can be applied.

Other approaches to fault management include training neural networks to diagnose the root cause of failures, the use of regular language generators and parsers, as well as the use of data-mining techniques to determine the events that may be happening with a high probability. Other work on fault management includes schemes to distribute the fault management process so that it can be done in parallel by many distributed fault management systems running in parallel.

An excellent survey of various algorithms to determine the root cause from symptoms using a variety of academic algorithms can be found in [5].

Once the root cause underlying a set of symptoms is identified, the root cause and the symptoms may be reported to the administrators in a variety of manner. In some systems, representing the faults visually on a map of the computer system is used to alert administrators to the occurrence of a fault. In other systems, a scrollable list of high priority alerts is used to show the failure events which may need to be fixed. In operating centers where several administrators are responsible for managing the system, a workload distribution system is used to allocate the failures to specific administrators who are charged with the task

of fixing the problem. Many systems include notification features so that a remote administrator can be emailed, called via an automated dialing system, or receive a text message when a fault happens. Such notification features are used in systems which may encounter a fault while the management staff is away, e.g., during non-work hours.

On the receipt of the notifications, the management staff would identify the root cause (if needed) and take steps to correct the cause of the failure. These may include restarting or rebooting systems, reinstalling a software, or changing the configuration of a machine or software system. All of these steps require a significant amount of manual intervention. In the next section, we look at some of the approaches that are being tried to automate the process of fault diagnosis and recovery in computer systems.

6.3 Self-Healing Systems

A self-healing system is a system that can detect problems that occur within its operation automatically and correct itself in an automated manner. While no computer system can claim to be completely self-healing in all aspects of its operations, there are many instances of self-healing systems which attempt to fix selected problems in an automated manner. In this chapter, we look at some of the techniques and examples where automated systems to diagnose and correct problems have been used. Starting with a generic approach to build a self-healing system, we look at different examples of correcting problems automatically in different domains of a computer system.

6.3.1 Autonomic Computing Architecture and Variations

There are many different frameworks proposed in the current research literature that can be used to design and develop self-managing system. We use one of them, the IBM Autonomic Computing framework, as the example to study. Most of the other self-healing frameworks can be readily mapped to the framework of autonomic computing.

The autonomic computing architecture [6], as shown in Fig. 6.8, divides each self-managing system into an autonomic element and an autonomic manager. The autonomic element consists of sensors which provide information about the current state of the autonomic element that can be read by the manager, and the effectors which are the interfaces on the element that can be invoked. The autonomic manager is responsible for the managing different problems that may arise in the operation of the element.

The autonomic manager itself is composed of four different types of functional elements – a monitoring element, an analysis element, a planning

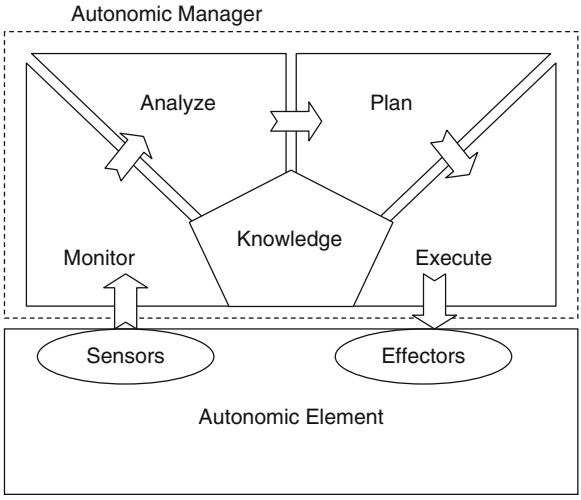


Fig. 6.8 Structure of an autonomic element

element, and an execution element. The operation of each of these elements leverages some knowledge-base which is shared by each of the elements.

The monitoring element is responsible for collecting information from the element being managed and can be viewed as an implementation of the various techniques discussed in Chapter 5.

The analysis element is responsible for understanding whether anything abnormal is happening in the system based on the monitored information and can be viewed as a summary of the techniques described in the previous section of this chapter.

The planning element is responsible for determining the corrective action that can fix the root-cause event in case of any problem, while the execution element is responsible for the invocation of the corrective action. The corrective action in the case of autonomic computing would consist of invoking g one or more of the effector interfaces on the element.

The planning element can use a variety of algorithms to determine the right course of action in order to fix a problem. One simple approach is that of maintaining a knowledge-base which maps each diagnosed root cause to a correction action. An alternative would be to have a rule-based system in which a set of rules define the corrective action to be taken based on the state of the autonomic element, and the root cause that is diagnosed by the analysis element. The decision tree algorithm for root-cause analysis and the knowledge-base approach described in the previous section can also be easily extended to determine the corrective action in addition to the root cause.

The MAPE loop is similar to many other approaches used for decision-making processes that use a control loop to determine their actions. The high-level architecture is similar to those proposed by HP Adaptive Enterprise

initiative as well as the Microsoft Dynamic Systems initiative. Other analogous loops that use similar techniques in other domains are the OODA loop (Observe-Orient-Decide-Act) used in military planning and the PDCA (Plan-Do- Check-Act) concepts used for quality improvement.

As specific instances of autonomic management are realized, the implementations of the different elements in the autonomic loop are customized to deal with specific nuances of the domain in which faults and errors are being managed automatically. In the following sections, we describe some of the prototypes and demonstrations that demonstrate techniques for automated failure handling.

6.3.2 An Example of a Self Healing System

A tiered web server system consists of many different types of computers which work together to provide a highly scalable application to the end users. The tiered web server may encounter a variety of faults and errors, including software packages that are loaded inappropriately, systems that may go down, or whose configuration may be incorrect. As different types of faults may be occurring in the system, a management station can automatically attempt to diagnose the problems and fix them. One of the ways to implement a self-healing system using the architecture discussed in the previous section is described below.

Let us consider an autonomic manager for a tiered web server system. The errors and faults from the different components of the system are captured in the log files produced by the different software packages running in the environment. These log-files are collected and converted to a common format. The collection of the log-files and their conversion constitutes the monitoring component of the MAPE loop. The converted format can be viewed as the generation of the symptoms from the system.

After the logs are converted to a canonical format, the symptoms from different log files are correlated together in the analysis step. In order to correlate the symptoms, each error symptom from one log can be compared to other error symptoms that occur at approximately the same time as those in the other logs. By comparing the fields from different log entries, e.g., the parameters of a URL request at a cache and application servers, and the parameters of a query at the database servers, symptoms that occur for the same request at different logs can be combined together. The collected set of symptoms can then be analyzed to determine the root cause of any errors using one of the methods described earlier in the chapter. This determination forms the analyze step of the MAPE loop.

Once the root cause has been identified, the planning part of the MAPE loop consists of determining an appropriate corrective step. For the common types of root causes that one may encounter in the system shown below, the corrective actions may consist of restarting an application, changing the configuration of an application, or installing an application. The planning module makes the determination based on a set of rules or other planning techniques.

The execution engine is responsible for invoking the decisions of the planning module. However, the execution engine may not be able to execute the decisions immediately. As an example, if the right way to resolve a problem is to restart the database server, and the problem is only affecting a small subset of the requests, a better course of action may be to defer the restart to occur during off-hours. On the other hand, if the fault is affecting the operation of the entire system, then an immediate restart of the system would be warranted.

One way to address such type of decisions is through the specification of policies. Policies are expressions that specify constraints on how a system operates or behaves. They are often specified using the same syntax as that of rules, using a conditional part and an action part that is invoked when the condition is satisfied. In the case of this example, the policy may state that at certain times of day, depending upon the type of symptom received, the corrective action may or may not be performed. The execution engine can check the existence of such policies and take the corrective action when permitted by the policy.

6.4 Avoiding Failures

Although it is desirable to have a good fault management system, as well as systems that can automatically manage the faults as they occur, it is even better to have systems that can prevent or avoid the occurrence of faults. In this section, we look at some of the systems that are designed to prevent the occurrence of faults in a computer system.

In general, several techniques for avoiding failures within systems need to be programmed as an essential attribute of a computer system and can not be influenced by a system administrator or manager. Examples of such techniques include the routing protocols in computer networks which work around link or node failures, adaptive behavior of TCP to work around dropped packets, and the fault-tolerant self-configuration of optical networks such as SONET. A system operator cannot influence upon such techniques over and beyond the features of the product. However, there are some schemes that a system administrator can use to improve the failure avoidance property of a computer system. The administrator can incorporate such techniques either during the planning stage of a computer system, or during the operational stage. In the following subsections, some of these techniques are described.

6.4.1 Redundancy

The most prevalent and common form for avoiding fault is by building in redundancy into the system. A computer system has many components. The function of each component is to process requests that come from the client. After processing, the requests are forwarded to a receiver which could be the

same as the client or another component that needs to perform further operations on the system. To provide redundancy, one or more components of the system are replicated.

The most common form of replication is to have a redundant component as a backup for each of the components in the system. The backup can operate in one of three modes, as a cold standby, a hot standby, or a warm standby. As a cold standby, requests are forwarded to the backup only when the primary component fails. As a hot standby, the requests are sent to both the backup and the primary nodes. Only the primary node forward the modified requests to the next stage of processing, or responds back to the client. The state of the hot standby and the primary are always synchronized. As a warm standby, the requests only go to the primary, who keeps on backing up its state to the backup at periodic intervals. When a primary has multiple backups, the model is also known as the coordinator-cohort model.

In addition to the primary-backup mode of operation, some components may be operating as peers when working in a redundant manner. Each request is processed by all of the peers with some arrangement among the peers so that only one of them will respond back to the client, or forward the request for further processing.

One mechanism is for the request to be broadcast to several peers. Each of the peers processes the requests, but only one of them forward the request to the next component (or the client). The selection of the forwarding peer can be done by means of a hashing function which maps some key identifying attributes of the request to one of the peers in the node. The peers, however, need to be aware of which other peers are operational so that the hashing function can be modified according to the set of peers who are operational. Another alternative is for the peers to select a leader among themselves who is responsible for forwarding the request. This is the same as the coordinator-cohort model where the coordinator is selected dynamically.

Systems that are designed with redundant components can continue to operate even when a component has failed. The fault management system can identify the failed component, but the corrective action of replacing the failed component can be deferred to a convenient maintenance time period.

6.4.2 Independent Monitor

One way for different components of a system to become tolerant is to have an independent monitor. An independent monitor is a system component that periodically checks on the status of all components of a system. If a specific component is down, then the monitor restarts the system or restarts a backup copy of the system.

The monitor is an effective way to avoid the faults that cause a component to stop working, e.g., a process that hangs or crashes in its operation. As long as

the process has not hung up due to a persistent problem, e.g., a configuration issue, the restart of the system would be able to bring the component back to a functional state. The independent monitor would try to restart the failed component a fixed number of times, and if the component keeps on failing (hanging up or crashing) repeatedly, notifies the management system of the failure.

The independent monitor can be used successfully to restart many components of a distributed application. It is quite effective for complex software packages that do not rely strongly on the information maintained by a previous incarnation of the package. Examples of such packages include caches, presentation services, proxies, authentication servers, connection managers, etc. Most applications that provide such functionality can resume correct operation when they are restarted after a crash.

6.4.3 Collaborative Monitoring

A variation of an independent monitor in distributed systems is collaborative monitoring. Consider a distributed computer system which consists of many different components. Each of these components can monitor the other components for failure. The monitoring for failures may be explicit by sending special liveliness messages, or it may be implicit by observing the requests sent by a component to another component, and determining whether a response is obtained in a specific time period.

When a component is determined to have failed, the component noticing the failure can either restart the failed component or more typically notify a controller which has the responsibility of restarting the failed component. The controller approach usually works better since it can remove duplicate requests to restart a failed component.

When monitoring components and restarting them, it is not necessary to wait for a component to fail in order to restart them. The monitor (either the independent monitor or a component collaboratively monitoring another component) can notice a trend of degrading performance (e.g. increased response times) and determine that a failure is likely to have happened. In those cases, a restart can be done even before the component had failed.

A simplistic approach of restarting components before they have failed is that of aged restarts described in the next section.

6.4.4 Aged Restarts

In many application systems, the probability of a hung application or a crash increases as the software keeps on running for a longer period of time. In an application where some latent programming defect causes a slow leak of

memory during operation, the software may become very slow or prone to generate errors after it has been running for a long time. A common method for fixing a malfunctioning computer system is to reboot it and see if that fixed the problem.

If the computer system components being used are susceptible to the aging problem, the one way to avoid faults and failures is to proactively restart the component at periodic intervals. This can be especially effective in environments where multiple instances of a component are used, e.g., a tier of web caches or application servers with a load balancer in front of them. A control component could monitor how long the caches or application servers have been running for, and then restart them one at a time in a staggered fashion to ensure that no instance runs for more that a predetermined amount of time. Since there are multiple instances active at any time, the short interval during which a single instance is rebooted will not be noticeable to the end users, specially if the load balancer in the front is adjusted so as not to route any requests to the affected instance before the restart is done, and the load balancer reconfigured to route the requests back to that instance after it is restarted.

6.5 Summary

In this chapter, we have looked at the different techniques for diagnosing the faults that can arise in systems management. Starting with an abstract architecture for fault management, we have summarized the different algorithms that can be used for understanding the root cause that can lead to generation of different types of faults in the system. We have also discussed the subject of developing self-managing systems as well as the techniques that can be used to build systems that can avoid the occurrence of faults.

While this chapter has focused more on algorithms that can be used practically in large-scale systems, there are several other algorithms that have also been proposed in research literature dealing with fault management. A sampling of the various algorithms can be found in references [7–17].

6.6 Review Questions

1. What is the state transition model for fault management? What general types of log messages are generated in each of the stages?
2. What are the common types of root causes that can cause failures in the computer systems?
3. What are the strengths and weaknesses of topology analysis-based methods for fault analysis in computer systems?
4. What is the nature of rules that are used to analyze faults in rule-based approaches?

5. Describe the dependency graph-based approach to fault management.
6. Many of the fault management approaches require some knowledge base in order to operate properly. What are some of the techniques that can be used in order to create such a knowledge base.
7. Compare and contrast the codebook approach for fault diagnosis with a rule-based approach.
8. Project: Several AI-based algorithms for fault management are described in Section 6.2.8. Discuss the conditions under which these fault management approaches would have an advantage over the non AI approaches such as codebook or rule-based fault diagnosis.
9. What is the autonomic computing architecture for building self-healing systems?
10. What are the issues and challenges in deploying a self-healing system in a production environment?
11. What are some of the approaches that can be used to reduce the chance of a fault happening in the future?

References

1. E. Manoel, M.J. Nielsen, A. Salahshour, S. Sampath, and S. Sudarshanan, Problem Determination using Self-Managing Autonomic Technology, IBM Redbook Number SG-24-6665-00, June 2005.
2. OASIS Web Services Distributed Management Working Group Common Base Event Specification, October 2003.
3. IBM Support Assistant, http://www.ibm.com/software/support/isa/
4. T. Acorn and Walden, S., SMART: Support management automated reasoning technology for Compaq customer service. In Proceedings of the Tenth National Conference Conference on Artificial Intelligence. MIT Press, Cambridge, 1992.
5. M. Steinder and A.S. Sethi, A Survey of fault localization techniques in computer networks, Science of Computer Programming, Special Edition on Topics in System Administration, 53(2): 165–194, November 2004.
6. A. Ganek and T. Corbi, The dawning of the autonomic computing era, Autonomic Computing. IBM Systems Journal, 42(1): 5–18, 2003.
7. A.T. Bouloutas, S.B. Calo, A. Finkel, and I. Katzela, Distributed fault identification in telecommunication networks, Journal of Network and Systems Management, 3(3): 295–312, 1995.
8. S. Brugnoni, R. Manione, E. Montariolo, E. Paschetta, and L. Sisto, An expert system for real time diagnosis of the Italian telecommunications network, In: H.G. Hegering, Y. Yemini (Eds.), Integrated Network Management III, North-Holland, Amsterdam, 1993.
9. G. Forman, M. Jain, J. Martinka, M. Mansouri-Samani, and A. Snoeren, Automated end-to-end system diagnosis of networked printing services using model based reasoning, In: Ninth International Workshop on Distributed Systems: Operations and Management, University of Delaware, Newark, DE, October 1998, pp. 142–154 [87].
10. R.D. Gardner and D.A. Harle, Alarm correlation and network fault resolution using the Kohonen self-organizing map, In: Proceedings of IEEE GLOBECOM, Toronto, Canada, September 1997.
11. P. Hong and P. Sen, Incorporating non-deterministic reasoning in managing heterogeneous network faults, Integrated Network Management II, North-Holland, Amsterdam, 1991, pp. 481–492.

12. C. Joseph, J. Kindrick, K. Muralidhar, and T. Toth-Fejel, MAP fault management expert system, In: B. Meandzija, J. Westcott (Eds.), Integrated Network Management I, North-Holland, Amsterdam, 1989, pp. 627–636 [68].
13. S. Katker, A modeling framework for integrated distributed systems fault management, Proceedings of the IFIP/IEEE International Conference on Distributed Platforms, Dresden, Germany, 1996, pp. 187–198.
14. S. Katker and K. Geihs, A generic model for fault isolation in integrated management systems, Journal of Network and Systems Management, 5(2): 109–130, 1997.
15. I. Katzela and M. Schwartz, Schemes for fault identification in communication networks, IEEE/ACM Transactions on Networking, 3(6): 733–764, 1995.
16. S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo, A coding approach to event correlation, Proceedings of Integrated Network Managemen, Chapman and Hall, London, 1995, pp. 266–277 [86].
17. L. Lewis, A case-based reasoning approach to the resolution of faults in communications networks, In: Proceedings of Integrated Network Management III, North-Holland, Amsterdam, 1993, pp. 671–681 [36].

Chapter 7
Configuration Management

A computer system consists of many components including hardware devices such as servers, routers, switches, and firewalls and software systems such as the operating system and applications. Each such component allows the operator to change its settings to customize its behavior. The sum total of all settings that can be modified and changed by the operator in the computer system is the configuration of the computer system.

Configuration management is the task of managing the configuration of different components in a system so that the system operates seamlessly. In large enterprises, where the configuration management needs to manage the configuration of thousands of devices, effective techniques are needed to make the task of configuration simple.

Each component of a computer system offers a different set of settings that can be modified by an administrator. The first section looks at the set of configuration settings that can be adjusted in different types of computer elements and looks at the different tasks that are associated with configuration management. The next sections discuss the architecture and algorithms associated with each of the different tasks.

7.1 Configuration Management Overview

Configuration management of computer systems can be divided into two primary functions – configuration settings management and software version management. The configuration settings management deals with the task of ensuring that the configuration of the different devices in the computer system is set to have the values that are desired for their proper operation. The software version management deals with the task of ensuring that the different software packages on various devices are maintained at a current version.

Software version management is also referred to as patch management. It is a common practice in software industry to release software changes in major and minor versions. A major version usually adds new features or functions. A

D.C. Verma, *Principles of Computer Systems and Network Management*,
DOI 10.1007/978-0-387-89009-8_7, © Springer Science+Business Media, LLC 2009

minor version usually fixes errors or security vulnerability in the existing major version. The term patch is used to refer to a minor version release.

As an example, let us consider a simple device, the Wi-Fi wireless router which is found ubiquitously in homes and enterprises. In most such devices, one could log into the administrator console of the device and find a variety of parameters that need to be configured for the device to operate properly. At the minimum, the configuration parameters need to be defined for the service set identifier (SSID) – the name that differentiates one wireless network from another, the type of encryption mechanism for communication being used and any associated keys or pass phrases, the network address of the access router and the range of the network addresses to be assigned to devices connecting downlink to the access router, and the type of network connection that is available to uplink to the larger network to which the access router is being configured.

Most wireless routers come with some default values for all of the configuration parameters, and in some cases the default values would be sufficient for the proper use and operation of the device. In a large majority of the cases, some changes to the configuration parameters would be required (e.g., one would want to change the SSID and encryption parameters to a local value). Ensuring that the right configuration settings have been made on the device is the job of configuration management.

Furthermore, like most devices, the wireless router contains a software system which is essential for its proper operation. The manufacturer of the wireless router periodically releases upgrade version for that software system. If there are problems in the proper functioning of the wireless router, e.g., it is not establishing a connection with the newer model of a laptop, the problem may lie in a bug in the software system which can be corrected by upgrading the software to the latest version of the software released by the equipment manufacturer. The job of managing the right software configuration on a device is the task of software release management.

The wireless router is an example of a device where the job of software release management is not very frequent. However, in other types of devices, e.g., on a personal computer, the software upgrades are released much more frequently by the manufacturer of the operating system of the machine. If the personal computer is used to host an application, e.g., a web server, the maker of the web server may also release upgraded versions of the application. The owner of the personal computer has to decide which new versions to install on the machine and when to upgrade from the existing version to a newer version.

As another example, let us consider a server which is installed within an enterprise. In order for the server to work properly, it needs to know different configuration parameters, e.g., have its IP address, have the right configuration, have the right identity of users and passwords established at its location. These are configuration settings that can be found in some configuration file or modified by using a configuration command line script. Furthermore, the operating system on the server may need to be upgraded to satisfy different

types of new patches or upgraded software that the manufacturer of the server may have released. Ensuring that the software is upgraded to the right version for any server is the task of software version management.

In almost any component of a computer system, the challenge of managing the right configuration settings and of managing the software versions across all the devices exists. In a hardware device, e.g., managing a router, wireless access point, or a switch, the normal task consists of setting the configurations of the device, and the software version management task consists of ensuring that the right version of firmware is installed on that device. The software version management task would need to be undertaken less frequently. In other components, where the role of software is more prominent, e.g., in an application such as a web server, the task of software version management may need to be performed more frequently. In such applications, configuration settings management means the right configuration file is created on all installations of the application, while software version management requires that the desired version of the application software is installed. It is quite common for updated versions of the application software to be released. Sometimes, it is desirable to be on the latest version, while at other times it is desirable to be at a previous version in which the application software is known to work properly.

Configuration management consists of three main activities: (i) setting the configurations of devices that are in the computer systems, (ii) discovering the current configuration of devices in the computer system, and (iii) managing the impact that a change in the configuration of devices may cause. Analogous activities are also needed for patch management.

In the remainder of this chapter, we examine the issues involved in configuration management and software version management in more detail.

7.2 Configuration Setting

In this section, we look at the task of setting configuration values on different devices that form part of a distributed computer system. Most of us are likely to have experiences with configuring some type of computer or wireless home gateway. Configuration in these cases requires setting some values or changing the contents of a configuration file on the device.

Each of us probably has some experience in changing or setting the configuration of a computer or a device at our home. Furthermore, all manufacturers are continuously trying to make their devices easier to manage – provided default configuration parameters that would let the device work out of the box and to increase the usability of the devices. Therefore, it may come as a surprise to many that configuration management is a hard task in many large enterprise environments.

The two factors that make configuration management difficult in large enterprises are the (i) number and (ii) diversity of devices and software packages that need to be configured in an enterprise environment.

A given enterprise environment has a large number of devices and software packages. Configuring all of them correctly is a tedious operation, and a few mistakes can occur when a large number of systems are configured. If the configuration process is manual, then mistakes are more likely to occur. Furthermore, each enterprise has many different types of software packages and devices that need to be configured. Since there are no standards about how different systems will be configured, the configuration interfaces offered by different elements are very different, requiring an administrator to learn a variety of configuration interfaces.

Although most enterprises try to reduce the heterogeneity in their computing infrastructure by standardizing on different types of components, complete homogeneity in the computing infrastructure remains an elusive goal. The modern enterprise is always in a continuous state of flux, with mergers, acquisitions, and divesture of different parts of an enterprise occurring at a continuous rate, resulting in a divergence from any IT infrastructure standards that may have been specified. Even if a company were to obtain the difficult goal of standardizing on a single product, e.g., all machines purchased from a single computer supplier and all running a common operating system, it needs to deal with the fact that models of computers change over time and that different versions of operating systems need to be supported. For any business of medium size or larger, obtaining a homogenous computing environment is very difficult.

Let us now look at various techniques that can be used to simplify the configuration of devices in an environment where there are a large number of heterogenous devices.

7.2.1 Reusing Configuration Settings

Although there may be many different devices in an enterprise, the configuration of many of them is likely to be identical or differ only slightly from each other. The first technique to simplify the task of managing configuration of different devices is to reuse the configuration from one device to another.

Let us consider the case of two servers that are running an instance of Apache web server. The two instances of web servers will have virtually similar configuration except for their network address. Thus, an administrator can manually configure one of the web servers, take the set of configuration files that are used for configuring the web server, change all occurrences of the IP address in the configuration files to that of the second server, and copy it over to the second web server.

As an example, let us consider the case of an Apache web server which is installed on a machine with several network interfaces, but expected to be visible only from one of the interfaces available on the machine. Let us further assume that it is configured according to the directives included in the sample

httpd.conf file that is shown below. To keep the discussion simple, we are only providing an excerpt of the configuration file and ignoring many different configuration options that would need to be specified in a real configuration file. The configuration file is a simple structure containing a parameter and its value on each line and some grouping of the parameters together using delimiters in angular brackets like the <IfModule> ... </IfModule> cluster in the example below.

```
ServerRoot "/usr/local/apache"
PidFile logs/httpd.pid

Listen 9.2.2.1:80
Listen 9.2.2.1:443

LoadModule ssl_module modules/mod_ssl.so
LoadModule cgi_module modules/mod_cgi.so
LoadModule mod_load modules/mod_load.so

<IfModule mod_load>
MaxLoad 10 "Access denied, Server load too high"
</IfModule>
DocumentRoot "/usr/local/apache/htdocs"
```

This configuration file is customized to run on a machine which has an interface with an IP address of 9.2.2.1. When the configuration file needs to be copied to another machine, one would need to change the interface address to that of the new machine. Assuming that the web servers store their content in the same location, and are going to use the same modules, and have identical configuration, the same configuration file can be copied over to the second machine – with only the IP address[1] changed between the two files.

When the configuration is set for the first web server, the administrator needs to carefully check that the web server is working properly with the selected configuration setting. Such a validation may be time consuming. However, if the second web server is running on an identical machine, then the tests on validating the configuration of the second machine can be completed much more quickly. Instead of running tests that check the validity of all configuration parameters, tests that check that the address has been modified correctly should suffice.

Although we have given an example for the Apache web server, a similar paradigm can be used for network routers, switches, machine operating system configuration, and a variety of other software systems that one needs to configure.

[1] For Apache httpd.conf, one can also omit the interface address and copy the configuration file across without any changes – assuming that the web-server process is expected to be reachable through each of the network interfaces on the machine, as opposed to a specified one.

A prerequisite to the reuse of configuration settings is the existence of many devices in the computer system whose configurations are relatively similar, and the configuration of one device can be readily copied to the configuration of another device.

7.2.2 Script-Based Configuration Management

Taking a configuration file on one device and modifying it to reflect the name or address of another device saves significant amount of time when compared to doing the configuration afresh. However, the manual editing of the configuration file may still lead to some manual errors, e.g., when the network address needs to be changed in four locations, one may accidentally only change it in three locations or one may make a typing mistake in entering the value.

One method to reduce the error involved in modifying configuration settings when creating configuration settings for different machines that differ in only a few parameters is to use scripts or templates to generate the configuration files. As an example, one could use a template for creating the configuration of the Apache web server discussed in the previous section as shown below:

```
ServerRoot "/usr/local/apache"
PidFile logs/httpd.pid

Listen$IP_ADDR:80
Listen$IP_ADDR:443

LoadModule ssl_module modules/mod_ssl.so
LoadModule cgi_module modules/mod_cgi.so
LoadModule mod_load modules/mod_load.so

<IfModule mod_load>
MaxLoad 10 "Access denied, Server load too high"
</IfModule>
DocumentRoot "/usr/local/apache/htdocs"
```

A template is a configuration file which is created with placeholders for some key values instead of the actual values. In the template shown above, the placeholder $IP_ADDR is used for the network addresses. For each web server, the placeholder is replaced with the actual value, e.g., using a global search and replace utility, and the resulting configuration file used to configure the appropriate web server.

A script is a programmatic way to pick a template and replace the placeholders in the template with the values provided as arguments. As an

example, one can write a simple script in a programming language of one's choice (C, Java, Perl, Python, etc.) which is invoked as

```
configure_web_server 9.2.2.1
```

The script picks up the template stored as a convenient location in the file system, replaces the value of the placeholder $IP_ADDR with the provided argument, takes the resulting configuration file, and uploads it to the right directory on the selected server.

Different scripts can be developed to configure the different types of devices that one may encounter typically. As an example, the operator of a web-hosting site may choose to develop different scripts to add a server to an existing customer, remove a server from an existing customer, add a new customer, or remove an existing customer.

Scripts can also be used to operate on a cluster of devices, e.g., a script can be written to configure both a server and the network firewall which will enable access to that server. If some configuration operations are performed repeatedly in an enterprise, then writing scripts to automate those operations reduces the burden of performing those changes.

In commercial data centers, several scripts are developed for performing common tasks such as installing or uninstalling a specific application, configuring a server, configuring routers, firewalls, and network devices. Scripts are a useful way to automate repetitive tasks and reduce the probability of errors. They are used frequently in data center operations in enterprises and telecommunications system operations.

The use of scripts and templates is associated with one problem – the challenge of managing the large number of scripts that are needed to deal with the operation and maintenance of a complex enterprise environment. Scripts may need to be upgraded as new types of machines are added or the software on one of the machines is changed. Maintaining the scripts when the original author moves to another job is usually difficult.

7.2.3 Model-Based Configuration Management

Model-based configuration management is a way to manage the configuration of devices using abstracted models that can be used to represent the behavior of the different devices.

When developing or designing a new software system, it is common to define a model of the system being developed (e.g., modern day practices call on the use of object-oriented model represented in UML notation). The model is then used as a blueprint to develop the system.

Model-based management extends the same approach to the task of systems and network management [1]. A model of the system is used as the representation of the objectives of the system. The model may be associated with

functional requirements, i.e., the functionality it is required to that deliver, and non-functional requirements (i.e., security, performance, and reliability). In software development, the models can then be converted into running code components that implement the functional and non-functional requirements of the model.

Model-based management uses a similar model of an operational system to perform different types of management functions. Using the model, one can determine the type of configuration each component of a system ought to have. Usually, the model is mapped onto a set of physical resources, and the configuration of each physical resource that can satisfy the functional and non-functional requirements of the model is determined.

Using a model to generate the configuration of a computer system consists of two steps:

(1) In the first step, the model of the application is converted to a physical topology of the system that is being developed. This process is part of the systems planning stage discussed in Chapter 2.
(2) In the second step, the configuration parameters of the different devices in the physical topology of the system are determined and configured using an appropriate set of scripts.

The first step, converting a user model to the right physical topology, is beyond the scope of systems management and its rightful place is in the field of software engineering. As a brief description, the UML model of the application is converted into a logical topology of the application that needs to be deployed. Then the logical topology is mapped onto the physical infrastructure of a data center, i.e., one determines the servers and devices where the logical components of the topology are to be installed and the functionality required of each of the devices. Interested readers can find a detailed exposition of the steps in [2].

Let us take a look at how step 2 can be implemented to configure the system automatically. Since the physical topology of the infrastructure is developed by means of software driven by higher level models, the type of configuration that is needed at each of the component in the physical topology is described in detail by the generated topology. For a component that is a physical device (e.g., a router, a firewall, or a server), the function to be performed by the desired functionality can be determined from the model describing the physical topology. If we have scripts written to provide that functionality to be supported on the specific device that is present in the environment, that script can be invoked and configure the device in an appropriate manner.

As an example, let us consider the case where we are trying to configure a data center network so as to enable the operation of a web server. The model of the physical topology that has been developed consists of a firewall that exports port 80 (the traditional port for http) and port 443 (the traditional port for https, i.e., http with the secure socket layer) connected to a web server installed with a web-server software. Let us say that the model needs to be implemented

in an environment which consists of a Cisco ASA 5505 and an Apache web server running on an IBM p5 560Q server with AIX operating system.

If there is a library of the scripts available for configuring the different devices in the environment, then the right script to configure the firewall and to configure the Apache software on the p5 server can be determined from that library. The additional parameter which will be needed to run the scripts is the network address of the firewall and the server, which is specified as part of the physical topology. Thus, the right scripts to be invoked on each of the devices selected and then the right applications and devices are to be configured correctly.

Model-based configuration management can be viewed as an abstraction layer that is implemented atop scripts to reduce the errors and inconsistencies that may occur due to the use of incorrect scripts. It also alleviates the complexity of managing scripts since scripts are now derived on the basis of higher level models describing the function and operation of the software system.

Model-based management can be used for management functions other than configuration management, e.g., they can be used as analytic models to assess the overall performance and security requirements of a computer system. Many commercial systems, such as Microsoft DSI and IBM Rational, provide support for model-based configuration management.

7.2.4 Configuration Workflows

In the configuration of a more complex environment, multiple scripts may need to be invoked, and these scripts may need to be invoked in a specific order. One could either encode the sequence of different scripts in another larger script or use the concept of a workflow to define the composition of scripts.

A workflow is a sequence of operations that need to be performed in order to achieve a goal. The operations, in general, may represent human actions, computer programs, or a description of any type of activity. In the context of computer systems configuration, the operation in a workflow would be a script that is invoked on a target machine.

The workflow allows the composition of different scripts in a more flexible manner than writing scripts. The workflows can be composed from the underlying scripts without using a programming or scripting language, and many system administrators will find them as a more convenient means for combining scripts.

Another advantage of the workflow mechanism is that it makes the market viable for configuration management products that enable automation of configuration operations. In general, the scripts written for automation of configuration operations in any enterprise are highly specific to the nuances of the IT infrastructure of that enterprise. This makes it hard for an independent systems management software vendor to provide a convenient product to use for

configuration management. However, if the software vendor provides a library of scripts to configure common types of applications and machines, and the customization to a specific customer is done by means of a workflow, a single product can be tailored more easily for use within many different environments.

Several software configuration products provide support for workflows for assembling different scripts together to create automation packages.

7.2.5 Simplifying Configuration Through Higher Abstractions

One common technique used to simplify the configuration and provisioning of computers systems is through the definition of higher level abstractions that are provided to drive the underlying configuration of the different devices that are within the computing environment.

The definition of the higher level abstraction is dependent on the specific technology or system that is being supported. As an example, let us consider the configuration of the security parameters of an enterprise network, which is managing its security using a combination of packet filtering firewalls, packet level encryption using IP security protocol, and remote access to other sites using a remote terminal protocol like Citrix operating over a secure transport protocol like SSL or TLS.

The IT infrastructure consists of the various firewalls, Citrix servers, and encryption boxes. One way to manage the configuration would be to determine the right configuration to support the access needs of the different users. That method of configuration has the advantage of being hands-on and providing a fine granularity of visibility and access to the administrators. However, that also requires the maintenance of the mapping of different user's access needs to the configuration of the different devices.

An alternate approach for managing the configuration of the devices is to define the configuration by means of higher level abstractions. A higher level abstraction for the case of security configuration would be to define the concepts of extended intranet, extranets, and remote users. An extended intranet is the collection of different networks that are within the control of the enterprise, and the firewalls need to be configured so that packets can securely transfer between the different component networks. An extranet is a set of networks or systems not under the control of the enterprise, but may belong to an external partner, who needs access to a subset of the enterprise's intranet. And remote users connect to the enterprise intranet using a remote access system such as Citrix.

Instead of trying to configure the individual devices, the configuration of the system can be represented in terms of the three abstractions outlined above. Changes are specified at the level of higher level abstraction, and the changes are implemented using automation scripts or workflows to drive the configuration of the lower level devices. As an example, when a new user is added for

remote access, the workflow scripts configure the Citrix servers and the firewalls to enable access from the new user to a set of devices. When a new network is added to the intranet, the workflow configures the firewalls at all of the remaining networks to enable secure access to the newly added network, in addition to configuring the firewall at the new site.

Managing the configuration by means of higher level abstractions has several advantages – it reduces the complexity of managing lower level devices, it promotes automation, and it is more efficient than managing the lower level configuration. The only drawback in defining higher level abstractions is that it increases the distance between the administrator and the devices, which may cause delays in diagnosis of the configuration problem in the case where the automation workflow has an error in it. On the balance, the advantages of configuring at the higher level of abstraction outweigh the difficulties of diagnosing the rare workflow error.

7.2.6 Policy-Based Configuration Management

Policy-based configuration management is a general technique for defining higher level abstractions in many different areas for the purpose of configuration management.

A policy is a constraint or a guideline on the operation of a computer system. It is often expressed in the form of a condition-action syntax (if the specified condition becomes true, then take the action) or in the form of a event-condition-action. (On the occurrence of a specific event, if the condition is true then take the action.) The use of policies for defining the configuration of various devices has been proposed for fields such as networking [3], security [4], and storage area networks [5].

In policy-based configuration management, higher level abstractions are defined for a domain and then translated to lower level configuration of the different devices that are found in the system. Two variations of using policies for simplified configurations have been proposed in literature – one in which policies are used to define the high-level configuration and the other in which policies are used as a mechanism to derive configuration parameters from higher level abstractions.

When policies are specified as the higher level mechanism for configuration, the entire system's operation is defined by means of higher level policies. As an example, the security configuration of an enterprise intranet may be defined by means of a set of role-based access control policies. Each user in an enterprise is assigned one or more roles, and all resources are grouped into a hierarchical order. The role-based access control policies defined which users (as defined by their roles) are allowed to have access to specific resources or resource groups. Through a refinement process, the high-level policies are mapped down to the exact configuration of the various devices that are needed to support the required access controls.

In the other approach to use policies, the higher level abstractions are defined so as to be decoupled from the actual operations needed to support those abstractions. As an example, for security purposes, a higher level abstraction may define the concept of intranets, extranets, and remote access users which are somewhat removed from the mechanism needed to support them (IP packet encryption, transport level security, etc.). A set of policy rules are used to transform or change these requirements into the configuration of the exact access control into the security gateways and firewalls.

In both of these approaches, the *if condition then action* paradigm acts as a general framework to simplify configuration and move it to a higher level of functionality – which attempts to ease the task of the system administrator.

7.3 Configuration Discovery and Change Control

Defining the configuration of devices when they are installed initially or need to have a change in their configuration is only one part of the overall challenge of configuration management. Another important aspect of configuration management is the task of discovering the configuration of the current devices in an enterprise.

The task of discovering configuration is a special case of the discovery process described in Chapter 4 and the monitoring process described in Chapter 5. In Chapter 4, different aspects related to discovering the devices present in the network were described, and Chapter 5 describes obtaining the different types of metrics available at the device and store them into a management database.

The management database into which all of the configuration information related to the different devices in the computer system is collected is called CMDB – or the configuration management database. The configuration management database contains information about the configuration of different devices in the enterprise.

The concept of CMDB has been given a lot of impetus by the introduction of IT Infrastructure Library or ITIL. ITIL [6] is a framework of best practices designed with the objective of managing the different types of services provided in computing environments. One aspect of that framework is the creation of a CMDB which tracks the configuration of various IT systems in an enterprise. In this book, we will only discuss aspects of CMDB related to maintaining the configuration data – although the focus of ITIL is on specification of a set of processes dealing with enterprise operations.[2]

[2] Strictly speaking, ITIL CMDB stores dependencies about services and processes into a central repository in addition to the configuration items of different devices. A repository containing only configuration information does not strictly conform to the definition of CMDB, but the term CMDB is loosely used in industry to refer to such repositories. In this book, we are going with the looser definition used in the industry.

There are two characteristics of configuration management database that differ from other types of management information. The first one is the fact that the configuration of most devices is relatively static, i.e., it does not change very frequently. The other fact is that the configuration of different devices is represented in very different manners – the configuration files of a mainframe server and the configuration parameters of a Ethernet switch have little in common.

The challenge of different types of configuration among devices is addressed by structuring the CMDB. The fact that configurations do not change rapidly allows the addressing of diversity in configuration information by creating a federated CMDB rather than a single dedicated repository. These two approaches to address the CMDB are discussed further below.

7.3.1 Structure of the CMDB

The configuration management database consists of a collection of different types of information related to the configuration of the various devices. In most implementations of the CMDB, some common mechanism for creating a common schema for all types of configuration information is defined. That common representation is referred to as a configuration item. In other words, a CMDB is a collection of different types of configuration items. Each configuration item would typically consist of four types of attributes:

Core attributes: These attributes include the identity, name, time of creation, time of last update, etc., attributes that are required of all configuration items in the database. These would typically include the identity of the device to which the configuration item belongs, the class of device, and a status information. The status information may indicate whether the device is active, inactive, or in any other state.

Capability attributes: The capability attributes refer to those attributes of a configuration item which may be needed in order to develop or support any management applications that may be implemented on top of the configuration database. As an example, if one is to implement an application that maps the location of each device to its physical location, then a capability attribute may record the room number to which a device belongs.

Control attributes: The context attributes provide an audit trail regarding the entity that created the configuration item (e.g., the software that created the configuration item) as well as the time-stamps and details of any changes to that configuration item.

Context attributes: These are additional attributes that may provide some context about the configuration item, e.g., they may record the number of processors of a machine or the name of the administrator of the machine. The context attributes may be used optionally by an administrator to get a better understanding of the type and usage of the device to which the configuration item belongs.

In addition to storing information about the physical devices, some CMDBs may store information about the end-to-end service which the different items are providing. In addition to the configuration items dealing with the physical devices, configuration items of the overall service or application supported by the IT infrastructure may also be represented in the CMDB.

7.3.2 Federated CMDB

A federated CMDB provides the concept of a single centralized repository of all configuration information, even though the actual configuration data may be distributed across multiple different machines with very different schemas describing the information they contain. The concept behind the federation of the CMDB is to allow an integration of configuration databases that may exist across different organizations or across difference types of components.

As an example, a CMDB may be created by integration of three types of configuration databases, the first dealing with the configuration of network devices, the second dealing with configuration of personal computers and applications on those computers, and the third dealing with servers and applications running on the servers. These three sets of configuration information would have a very different type of design and may have a different type of structure. Furthermore, they may be populated by a different set of discovery tools.

In the federated CMDB, a database would provide a virtualized view of the entire configuration with only a top view abstraction of configuration items and pointers to entries in the individual databases. The federated CMDB may contain the relationship between the different configuration items, e.g., an application installed on a personal computer may be the client component of an application installed on the server. Thus, the federated CMDB becomes an aggregation point for the three individual configuration databases. The fact that the database is not a single one, but spread across multiple databases, can be hidden by the database implementation (many commercial databases offer support for such federation), or it may be hidden by an access library software which all management applications written on top of the CMDB use.

The federated CMDB has many advantages – it allows existing configuration databases to be reused, it allows for a more scalable operation, and it allows for a single integrated view of the configuration of the entire enterprise.

Once a CMDB is in place, different types of management applications dealing with the configuration attributes can be developed. In the next section, we examine some of the typical applications that are developed for the management of configuration on top of the configuration management database.

7.3.3 Dependency Discovery

An important aspect of populating the configuration management database is the discovery and population of dependencies that exist among different

configuration items associated with different devices. The configuration items are associated with different components in the network, e.g., servers, routers, client workstations, and applications. It is relatively easy for a discovery tool to obtain the configuration information from a single device. As described in Chapter 4, the system would discover the different devices in the environment using a variety of techniques. Once the presence of a device or application package is known, it is relatively easy for a management system to query the device and obtain all appropriate configuration items.

The one complication in reading the configuration items from any device is the difference in protocols supported by a device for configuration item. As discussed in Chapter 3, different standards exist for obtaining different types of configuration information from different devices. Most network devices would export their configuration information using SNMP protocols using the MIB convention to represent the structure of the configuration data and would also provide a command line interface to retrieve the information manually or using an interactive scripting language. Many servers and client workstations would support configuration information retrieval using the DMTF CIM model and protocols. Applications typically do not have a standard convention for representing their configuration, but they invariably store the configuration in files and one could retrieve the right configuration files by remotely logging into the machine using a protocol such as telnet or secure shell. Applications installed on Windows operating system can store their configuration in files, or in the windows registry, depending on their design.

Discovery tools are designed so that they can discover the various configuration items available in a variety of formats and convert them to a canonical representation in the CMDB. The next step in the CMDB is the discovery of dependencies among the different configuration items in the CMDB. The techniques described below can be used to discover some of the dependencies.

7.3.3.1 Application Dependencies

An application software may depend on other application packages if it calls the software package or uses its interfaces. Application dependencies can be determined by observing the sequences of procedure calls that are invoked by the application. There are a variety of ways by which such an observation of the procedure call invocations can be made.

In some programming environments such as Java, it is possible to insert interceptor routines [10] which can track the invocation of the different procedure calls. These procedure invocation traces can be collected and analyzed to determine which software package is invoking another software package through its API. The sequence of invocation is referred to as the call graph for an application.

Interceptor routines can also be inserted into environments that are non-object oriented, although the interception method is more intrusive in those cases. One approach to introduce interceptors is through the so-called “shim”

layers. A shim layer is a library that provides the same interface as another software library underneath it and is used in many instances to provide additional functionality or to track calls to the library underneath.

As an example, the network driver interface specification (NDIS) mechanism within Windows or Linux is often used to insert networking shims. NDIS allows for invocation of multiple drivers in a chained manner and can be used to insert a virtual driver in between different normal protocol processing functions to perform other operations. As an example, the common protocol layering has one NDIS driver performing the processing of layer 3 protocol (IP processing) which invokes through the NDIS interface the processing of next layer or protocol (the Ethernet protocol processing). One could insert a virtual NDIS driver between the two layers to perform various operations, e.g., collect statistics about packets leaving the machine. Interceptors can also be placed with other interfaces that are present in the system.

Using an interceptor, or other techniques for monitoring network traffic, we can collect information about which packets are flowing among machines. We can map the port numbers used in packets to the applications that are running on the communicating machines. One way to do this mapping is to see which process is using the port assigned to the machine at that time (which can be obtained from the standard network monitoring commands on the machine) and then determining which executable was used to start the process (information available from the operating system of the machine). The call graph determining which application is calling another application can be used to build the dependency relationship among different applications. The application that is using the same port for communicating to different other applications is the server application, and the other party is the client applications. Client applications depend on the server applications for their correct operation.

7.3.3.2 Network Dependencies

A dependency on the configuration of different devices to permit correct operation can also be determined from an analysis of the network topology that is present in an environment. In a data center network, servers are used to host applications that are accessible by clients running outside the data center. Each server depends on access to an external network in order to enable the applications to be accessible.

By analyzing the topology of the network, one could determine the path between the access point of the external network to the data center and each server. The configuration of the different devices on the path must permit some of the traffic from the clients outside the data center to access the applications. If there is any device restricting traffic flow along the path, e.g., a firewall or a VLAN (Virtual local area network) switch, then the configuration of that device must permit such communication.

If we look at the application running on the server, then we can see that there is a dependency between the configuration of the application and the configuration of the device. The correct operation of the application depends on the proper configuration of the devices between the access point and the server on which the application is running.

Several kind of dependencies among the configuration of the devices can be determined by an analysis of the network if we know the connectivity that is required among different edges of the network. In a telecommunications network, if one knows the locations of the different customer access points and the list of access points that need to communicate to each other, the dependency of customer communication on the intervening devices can be determined by means of topology analysis.

7.3.3.3 Service Dependencies

A service is a combination of applications that is used in concert to provide some function to a user of the IT infrastructure. As an example, a collection of web-caching proxies, web servers, directory systems, and database systems can be used in concert by a bank to provide an online account access service.

A service therefore depends on many different applications that are running inside the IT infrastructure. However, the service is an abstract concept which cannot be determined or discovered automatically using a discovery tool.

However, if a manual description of a service is provided, then that description can include the list of applications that are required for providing the service. Such a list can then be augmented by discovering other applications that are required for successful operation of the listed applications (using techniques described in Section 7.3.3.1) as well as the network devices which are required for each of the applications (using techniques described in Section 7.3.3.2). This results in the determination of all the components of the system on which the offered service depends.

7.4 Configuration Management Applications

The configuration management database provides an enterprise with a view of the configuration of the entire IT infrastructure. Once the configuration information is collected, the information can be used to develop management applications that will ensure the smooth operation of the computer systems. In this section, we look at some of the typical configuration management applications that may be run on top of a configuration management database.

7.4.1 Configuration Validation

Configuration validation is the task of ensuring that the different devices in the computer system are configured properly. When the different configuration

items are stored within the database, it is possible to validate both the configuration of individual devices, and the configuration of related pair of devices.

One generic way to validate the configuration of devices is to express the constraints the configuration ought to satisfy. Constraints can be specified on the configuration of a single device or configuration of two devices. An example of the configuration of a single device is the constraint that all IP addresses in an enterprise have a given prefix (e.g., all machines of IBM have IP addresses of format 9.x.x.x). An example of the configuration related to two devices can be seen in an application system which consists of an application server connected to a database server. A requirement on the configuration could be that the number of threads that are configured to perform concurrent operations on the database server be no less than the number of concurrent threads that are configured to run on the database system.

For configuration validation, the constraints are encoded into a computer readable format, e.g., they may be represented as a set of XML statements or some other format. The configuration of the different devices in the configuration management database is then compared against the specified rules. Any violations are either reported to a customer or they can be automatically corrected in some cases by reconfiguring the devices.

Configuration validation systems of the type described here have been used in many contexts, e.g., to check the configuration of IP networks [7], to check the configuration of storage area networks [8], and to check the configuration of web-based applications [9].

7.4.2 What-If Analysis

One use of the configuration management database is the ability to do what-if analysis. The what-if analysis allows an administrator to determine the impact of making any configuration change in the system. When the configuration of a system changes, the change in configuration can impact other systems. The set of systems whose configuration change is impacted can be identified by means of the dependencies among configuration items that are described in the system.

As an example, let us consider a data center in which there are multiple applications that depend on the existence of a directory server in order to perform their user authentication service. Let us assume that the LDAP server needs to shut down for 4 h in order to perform some repairs. An analysis of the configuration database dependencies will indicate the applications which will be impacted by the non-availability of the server.

Similarly, the set of applications whose configuration would need to be potentially modified if the LDAP server configuration is changed can also be determined by means of the dependencies stored in the configuration database. The set of services which can be impacted by the failure of a component within the data center can also be readily determined by an analysis of such dependencies.

7.4.3 Configuration Cloning

Another advantage of having a configuration management database which stores the different configurations in a single location is that it easily allows the cloning of the configuration of one device to another. When the configuration of a source device is being cloned to a destination device, the steps involved are copying the configuration of the source device, modifying any device-specific parameters such as device address, and then changing the configuration of the destination device.

In many cases configuration cloning can be done without any changes required, e.g., consider the case of a load-balanced application. Such an application usually consists of two or more servers running that application, and requests for application access are spread among the devices using a load-balancer device. Suppose an additional server is needed to increase the capacity of the load-balanced application. The configuration of any of the servers can be cloned to derive the configuration of the new application without any changes. The only change needed is the adjustment of the load-balanced configuration for it to incorporate the new server added to the system.

7.5 Patch Management

In addition to maintaining the configuration of the different servers and software packages in an IT environment, one also needs to ensure that the software version is maintained up to date in the environment. All software manufacturers periodically release new versions (or patches) of their software, and the existing infrastructure may need to be upgraded to the newer release.

The general practice of patch management requires following a process life cycle consisting of four stages:

- *Patch Identification*: This stage includes the determination of available upgrades to existing devices that may need to be installed.
- *Patch Assessment*: This stage is the task of determining the importance and criticality that any new patch be applied.
- *Patch Testing*: This stage involves checking whether the installation of the new patches will impact system operation.
- *Patch Installation*: This stage does the actual installation of the patches and updating the software of existing applications and devices.

Each of these stages is described in the sections below.

7.5.1 Patch Identification

Patch identification is the task of determining which of the different software packages installed on the different devices in a computing environment have

newer software releases that are available from the manufacturers. The task involves obtaining the version of the software that is installed on the current machine and comparing that to the latest version of the software that is available from the manufacturer.

The task of obtaining the current software version that is installed in the different devices can be simplified significantly if there is a configuration management database which records the version of the different pieces of software that is installed on all of the machines. If there is no configuration management database, the current version needs to be retrieved as part of the discovery process for the different devices. These versions then need to be compared against the latest version that may be available from the manufacturer of the software.

The software manufacturers usually publish information about the latest version of the software they support on their web sites. Although there is no industry standard on how the information will be made available, automated scripts directed at obtaining the latest software version of specific application packages can be readily developed. The task then is comparing the latest version of the software to the installed version.

7.5.2 Patch Assessment

Even though the latest version of the software may be available, it does not necessarily mean that the software on existing systems need to be updated to the latest one. The installation of a new version of the software may break some existing applications. Also, a newer version may result in a degraded performance if the new version includes additional functionality that may not be needed for existing applications. Sometimes, an upgrade in the software version may necessitate an upgrade of the hardware, and the costs of such an upgrade need to be taken into consideration. On the other hand, a new version which reduces critical security vulnerabilities in the existing software may need to be applied immediately.

Patch assessment is the process of determining if a new version of software ought to be installed. It is hard to find a general tool which provides guidance on patch assessment – since the needs and requirements of different enterprises tend to be quite different. Therefore, patch assessment is usually done by the administrative staff that need to make a determination on which patch needs to be applied and the impact of making the patch or not making the patches.

When making an assessment as to whether the patch ought to be applied or not, the following aspects of applying the patches need to be taken into consideration:

- *Security Vulnerabilities*: Does applying the patch fix a hole in the security of the IT infrastructure? If a patch released by a software manufacturer fixes a security flaw that may impact operations, then the patch is probably worth

applying. On the other hand, if the system is being operated in a manner which renders that vulnerability ineffective in some other way, e.g., that application is installed on only an isolated machine which only connects to trusted computing systems, then one may want to skip the patch.

- *Compatibility*: Does the patch interfere with any of the critical applications that are already running on the system? If the patch adversely impacts the operation of the system, then it should not be installed. On the other hand, some patches fix performance flaws in existing software and can improve the performance of the existing applications. These issues need to be taken into account when deciding whether the new patch ought to be applied.
- *Functionality Supported:* Does the patch include features and functions that are relevant to the operation of the business and is worth the trouble of updating? If an enterprise is based completely on the Lotus Notes mail system, and an upgrade to Internet Explorer which provides enhanced support for Outlook, which provides same functionality as Notes, is available, an enterprise may decide not to apply that patch.
- *Operational Impact*: In general, applying any patch would require a disruption in the operation of the computer systems. The amount of time that will be lost due to the maintenance needs to be taken into account when deciding to assess a patch. The next available time-window when the update can be applied also needs to be taken into consideration.
- *Cost Considerations*: How much would the application of the patch cost in terms of labors and lost productivity? Does it require an upgrade of the hardware to maintain the required performance? A patch installation that may cost a lot to perform may be skipped, even if other considerations require it to be selected.

As can be seen, many of these considerations are soft issues dealing with the nature of the business the IT infrastructure is supporting, and it is hard to give a generic recipe for determining when a patch ought to be applied. Once a decision to apply a patch is made, the next step is to test the patch for proper operation.

7.5.3 Patch Testing

One of the most crucial aspects in the operation of a software patch is the evaluation of the impact of the new patch on existing systems and applications. The evaluation of this impact is made by testing the upgrade on a copy of the real system and performing a variety of tests to ensure that the patch does not cause any existing applications to break.

The extent of testing to be performed depends on the nature of the patch and the criticality of the application which is being upgraded or patched. For upgrades of firmware or operating system, one would first apply that patch to a single device and check that the device continues to work as intended after the

patch is applied. After that one would test for the impact of that upgrade on the operation of the existing applications that run on the system.

In enterprises which are running critical applications, patch testing is a serious process which requires setting up a realistic replica of the running application of a separate set of hardware systems, apply the patches to the replica, and run a variety of tests to validate that the test system is performing as desired. In many cases, one would take the requests that are coming into the real system, run them on the replica set up for testing, and validate that the responses and behavior of the test replica are similar to that of the original system.

In many enterprises, regression tests are established for each operational system. The regression tests include a set of tests that need to be completed successfully by the test replica before a patch is declared to be safe. Several test management systems exist which can automate the execution of the regression tests and enable system administrators to validate that a new patch is safe to apply.

Such test management systems can sometimes be integrated with patch installation systems described in the next chapter to provide a common way to handle testing and deployment [10].

If the result from testing of the system operation is favorable, i.e., the patch upgrade does not impact the operation of the system, then the patch can be applied to operational systems using the patch installation techniques described in the next section.

7.5.4 Patch Installation

Patch installation is the process of performing the actual upgrade of the software on an operational system. In most cases, the installation of the patch requires that the operational system be shut down for maintenance, although there has been the emergence of the capability to do hot patching for several applications and operating system. In hot patching, the application software has the ability to be upgraded while it is running, without requiring the shutdown of a system for maintenance purposes.

The maintenance downtime is usually scheduled on weekends or nights when the system is not likely to be heavily used. It is common to schedule many patches or upgrades on the systems to be applied during a single maintenance window. In that regard, one needs to be careful about installing patches in a safe order.

When multiple software patches are applied, they may need to be applied in a specific order. One patch may require the existence of another patch, e.g., a new patch to an application may require a minimum level of the operating system, and one needs to ensure that the operating system is at the desired level. Before a patch can be applied, all the prerequisite patches need to be determined and applied.

Fortunately, several tools existing on the market for software configuration management can arrange the upgrades so that they are done in the right order [11]. These tools would usually embody the concept of a dependency graph, i.e., they would allow the administrator to specify the prerequisites of each patch installation. From the existing configuration of the current software, the system would then determine the sequence of patches that needs to be applied so that the desired revision level can be attained on the software. Installation scripts then perform the required upgrades in the correct order on the system.

Once the installation is complete, it is usually tested for correct operation before making the system back live.

In systems that cannot afford to have a large maintenance downtime, one could establish an entire replica of an operational system with the required patch upgrades and have it installed in Paralel with the older operational system. Then the traffic is switched from the old system to the new system by reconfiguring a network device in front of the operational system. Such an operation can be performed with a relatively low or no downtime.

As an example of a system with no downtime, consider a load balancer with three web servers hosted behind it. When an upgrade is to be performed on the web servers, a fourth web server can be added with the new patches and one of the existing web servers taken out of operation. The decommissioned web server is then upgraded and brought on live, decommissioning one of the original servers, which is then upgraded. Finally, the last original server can be decommissioned. The switching from one set of servers to a different set can be done by reconfiguring the load balancer.

7.6 Summary

In this chapter, we have reviewed the basic concepts behind managing the configuration of different types of devices in an IT infrastructure. Configuration management has two major components, managing the correct settings of configuration parameters of the difference components of the IT infrastructure and managing the right version on software for each of the components, the latter being patch management.

Configuration parameter settings management includes the task of setting the parameters and the task of monitoring the current configuration settings. Configuration parameters can be set by various techniques such as templates, predefined scripts, model-based management, and policy-based management. Configuration monitoring can be done effectively by creating a configuration management database which provides a consolidated view of the different configuration settings. Configuration management database is usually implemented in a federated manner and enables the development of several applications such as configuration validation, dependency analysis, what-if analysis, and configuration cloning.

Patch management consists of a continuous four-stage process which consists of identifying the patches that may be applied, assessing which of the patches need to be applied, testing for the impact of the patch on operations, and the actual installation and deployment of the patch.

In the next chapter, we look at the subject of performance management.

7.7 Review Questions

1. What is configuration management?
2. How does software version management for systems management differ from software version control during software development?
3. What are the factors that make configuration management hard in large IT environments?
4. What are some of the techniques used to simplify the task of configuration management in large environments?
5. Compare model-based management to configuration management technique using templates. What are the relative strengths and weaknesses of both approach?
6. Describe policy-based management. How does it simplify the task of configuration management?
7. What is a configuration management database? How can it be used?
8. How can dependencies be discovered in a configuration management database?
9. What are the different aspects of patch management that need to be considered to manage software versions in an IT infrastructure?
10. Discuss some of the conditions under which it would be better for a computing infrastructure to remain at older versions of its software instead of upgrading to the latest version.

References

1. R. Shomaly, A model based approach to network, service and customer management systems: Object oriented technology, BT Technology Journal, 11(3): 123–130, November 1993.
2. T. Eilam et al., Managing the configuration complexity of distributed applications in internet data centers, IEEE Communications Magazine, March 2006.
3. J. Strassner, Policy-based Network Management: Solutions for Next Generation, Morgan Kaufmann, August 2003.
4. S. Jajodia et al., A unified framework for enforcing multiple access control policies. In: Proceedings of the ACM International SIGMOD Conference on Management of Data, May 1997.
5. D. Agrawal, K-W. Lee, and J. Lobo, Policy-based management of networked computing systems. IEEE Communications, 43(10):69–75, 2005.
6. UK Office of Government Commerce, The Introduction to the ITIL Service LifeCycle Book, August 2007.

7. R. Talpade, and J. Singh, Compliance assessment of IP networks: A necessity today, Information Systems Control Journal, 4: 51–53, April 2007.
8. D. Agrawal et al., Policy based validation of SAN configuration, Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, June 2004.
9. Y. Li et al., Modeling and Verifying Configuration in Service Deployment, IEEE International Conference on Services Computing, Chicago, IL, September 2006.
10. W. De Pauw and J. Vlissides, Visualizing object-oriented programs with Jinsight, Lecture Notes in Computer Science, In: Lecture Notes on Computer Science, ECOOP'98 Workshop Reader, July 1998.
11. O. Crameri et al., Staged deployment in mirage, an integrated software upgrade testing and distribution system. ACM SIGOPS Operating System Review, 41(6): 221–236, December 2007.

Chapter 8
Performance and Accounting Management

The performance of a computer system is a measurement of how well the system is able to respond to the workload that is being put onto it. The performance management of a system is the set of tasks that ensures that the system performance is satisfactory for its intended purpose.

In order for a system has good performance, one needs to use deployment time techniques as well as operation time techniques. The deployment time techniques are concerned with the designing the system so that it meets the workload that is anticipated. The operational time techniques deal with the issue of monitoring the system and reporting on its performance.

Deployment time techniques, which help in designing a system which will have a desired level of performance, were discussed in Chapter 2. A system will typically be designed to provide a desired level of performance. As discussed in that chapter, performance is characterized by measures of workload, throughput, response time, capacity, and utilization. The focus of this chapter is on performance management during the operation of the system.

During the operation of the system, the tasks involved in performance management include those of monitoring and reporting the observed performance of the system, determining and correcting the causes for degraded performance, and estimating future performance problems that may arise. This chapter focuses on these aspects of performance management.

In the beginning of this chapter, we discuss the need for operation time performance management, the different approaches for operational performance management in systems, followed by a discussion of the techniques for performance monitoring, performance troubleshooting, and capacity planning.

Accounting management is closely related to performance management and deals with the issue of determining how much of the resources are used by different users of the system in order to do accounting and billing. The last section of this chapter discusses issues in accounting management.

D.C. Verma, *Principles of Computer Systems and Network Management*,
DOI 10.1007/978-0-387-89009-8_8, © Springer Science+Business Media, LLC 2009

8.1 Need for Operation Time Performance Management

Since most systems are designed with some targeted level of performance in mind, that does not obviate the need for performance monitoring and performance management during the operation of a computer system. There are many reasons why performance management during operational stage of a computer system's life cycle remains a critical part of the overall systems management.

Despite the best models and designs that one can develop at the design time of the system, any performance planning at the design time is subject to various simplifying assumptions that one needs to make for the analysis of performance. These assumptions are frequently not satisfied by the real workload that is experienced within the system. As an example, it is quite common to assume Poisson distributions to characterize workload in networks, but it has been demonstrated that the workload in real networks never displays such ideal behavior [1]. Therefore the observed performance of the system in operation can deviate significantly from the design point of the system.

Since the end goal of any computer system is to provide a service to its users at an acceptable performance, monitoring the actual performance of the system is a good indicator of how well the system is doing. Continuous monitoring and observation of system performance is a good practice that can alert the administrator of any potential problems in system performance and allow one to take proactive steps to correct the problem before complaints start arriving at the help desk.

In many environments, monitoring of the system performance may be driven by the business objectives of the organization. If the computer system is used to provide services to a set of customers, there typically might be service level agreements (SLAs) in place requiring that the system continue to operate at a desired level of performance. Violation of the service levels may require giving customer credits and can cause monetary loss. Thus, an active approach to monitoring the operational performance of the system helps in tracking how well the SLAs are being satisfied and reduce the chances of monetary loss due to failure to satisfy the SLAs that may have been put into effect. Finally, performance monitoring needs to be performed in order to obtain the required information for billing and accounting purposes.

Because of these reasons, performance monitoring remains one of the most important aspects of overall systems management.

8.2 Approaches for Performance Management

The end goal for performance management is to ensure that the performance of the IT infrastructure is acceptable. In general, the performance of any computer system is determined by its capacity and its workload. As mentioned

in Chapter 2, workload measures the amount of work the system is being asked to perform, and capacity is the total amount of workload the system can support.

When the amount of workload in the system is significantly lower than its capacity, the performance would typically be quite good. However, there are cases where the workload may come close to the capacity of the system, or in some cases exceed the capacity of the system. This may happen because the workload gradually grows over time, there are periodic variations in the workload, or there is a sudden surge in the workload due to any reason. When workload exceeds or is close to the actual capacity of any system, most performance metrics are quite bad.

There are three approaches to manage performance under the conditions of heavy load. The first approach is to avoid the occurrence of such a situation. This is often done by designing the system so that its workload always remains a manageable fraction of its capacity. Also referred to as over-provisioning, this approach tries to operate the system in a state so that the probability of system encountering a heavy load is negligible. In any type of computer system, if the total workload on the system is a small fraction of its capacity, e.g., below 40% of capacity, the probability of poor performance is negligibly small.

After a system has been operational for some time, it is not uncommon to encounter some situations where the system experiences congestion, i.e., periods where the workload approaches a higher fraction of the system's capacity. When the workload is close to or exceeds the capacity of any computer system, the probability of good performance is negligibly small. Congestions and overload situations occur because the system workload grows gradually over time. Therefore, each operational system needs to be monitored continuously to check whether it is approaching its capacity, and an upgrade in the design of the system is required.

When a system is operating close to or beyond its capacity, good performance cannot be provided to all of the users. However, it is possible to provide good performance to some of the users at the expense of other users. The second approach to managing performance is to divide the users of the computer system into multiple categories – providing a different level of performance to the users in different categories. This is the fundamental concept behind offering different classes of service in different environments.

The third approach to managing performance is to provide good performance for specific individuals. In this approach, using a finer grain for providing performance assurance levels, a select set of individuals are assured of good performance. Performance is assured for each individual user, not for a category of user. This type of performance assurance works best for ensuring the performance of an individual user, but it comes at the expense of other users of the system.

Depending on the context in which a computer system is being operated, either of the above three approaches can be used to manage performance seen by individual users. In environments where bandwidth and computing resources are plentiful, one would tend to use the over-provisioning approach. In environments where bandwidth is relatively scarce, and some users have a critical need for better performance, the second approach of providing service differentiation would be appropriate. When the computing environment has a time-critical application or a very important user, the third approach may be used.

Most commercial environments tend to use the over-provisioning approach for managing performance since bandwidth and computing resources are relatively abundant. In a military environment, where bandwidth may be limited and soldiers, support personnel, and commanding officers have different criticality, the service differentiation approach may be the better one to use, and if one needs to carry a message from the Chief Executive Officer to all of the employees, then assuring individual performance to such message transmission may be the preferred approach.

In either of the approaches, the system would need to monitor and report the performance of the system as collected from the different components.

8.3 Performance Monitoring and Reporting

The architecture used for performance monitoring is identical to the model discussed in Chapter 4. The metrics and information available from the different devices in the network are collected from the different devices within the network, and then these metrics are analyzed to prepare the performance reports of the operation of the computer system.

Performance monitoring and reporting can be done either in real time or at periodic intervals. The operator of a system would typically like to see the performance metrics being displayed on a computer screen automatically, with colors or warning events identifying the any components where there may be a performance impact. At the same time, an overall performance report of how the entire system is performing may need to be prepared on a regular basis (e.g., once every day or once every month) which can be used to summarize the overall performance of the computer system to a customer or management in an enterprise.

There are two main challenges in performance monitoring and reporting, the first being the sheer volume of performance monitoring data that is available, and the second being the fact that the performance metrics for which reports need to be generated need not correspond exactly with the performance metrics that are captured by the raw data. The performance data need to be converted into the appropriate metrics that are to be included into the reports.

8.3.1 Performance Metrics

A computer system consists of many components and devices. Each of the components and devices contains many types of performance metrics. The following performance metrics are typically available from the different components of a computer system:

- *Workload*: The amount of work per unit of time encountered by the component. Workload may be characterized as requests per second or in terms of inbound bandwidth such as packets per second.
- *Response*: The amount of time it takes for the component to respond to a request.
- *Loss Rate*: The fraction of requests that do not receive a response, or receive an erroneous response.
- *Throughput*: The amount of work per unit of time that receives a normal response through the component. Sometimes it is also referred to as goodput.
- *Utilization*: A measurement of how busy the system is, what fraction it is busy in any given amount of time.

The exact meaning of the metrics above would differ slightly for different components and devices.

If we look at an interface of a network router that is forwarding IP packets, the workload is the number of IP packets that are received on the interface; the response time is the amount of time it takes for the packet to be forwarded to next interface; the loss rate is the number of packets that are discarded at the interface for any reason, the throughput is the number of packets that are successfully forwarded, and the utilization can be measured by the ratio of the bytes per second received on the interface to the maximum possible rate of the interface. Most of these metrics are normally collected by the software of an IP router and are available as entries in an SNMP MIB. Similar metrics can be defined at the level of overall IP router where the metrics from individual interfaces are summed together.

If we consider a live application such as a web server, then the workload is the number of web requests that are received at the web server; the response time is the average amount of time it takes to process the request and send the response back to the web server; the loss rate is the number of requests that receive an error response; the throughput is the number of web requests that are sent to a normal response, and the utilization is the fraction of time the application is busy. Web servers use the HTTP protocol and the protocol sends a response code of 200 for successful responses (more strictly speaking in range of 200–299 with various differences in meaning) and other codes for unsuccessful or retargeted responses. Most web servers also generate logs of their requests, and by observing the amount of the response code over any period of time, one can obtain an estimate of these metrics. The utilization of a web server is not directly

reported in the logs, but can be approximated by the utilization of the server the web server is running on. An alternative way to obtain these performance metrics would be to have the web server be periodically polled by a client machine and measure the collected responses.

Each of these metrics fluctuates rapidly over time, and the number of packets processed by an IP router in a 5 min interval could be quite different than the number of packets processed by the same router in the immediately following interval. Therefore, the volume of the performance metrics that is collected and stored into the management database rapidly increases. The management of scalability of performance data is one of the key challenges in performance management of any large enterprise.

8.3.2 Addressing Scalability Issues

One of the primary challenges associated with the volume of performance data is the scalability of the management database. There are two primary challenges that need to be addressed – first there must be a scalable management database that can handle the volume of performance data that is coming in, and second we need to develop schemes to reduce the volume of performance metrics to the extent possible.

Different approaches to make the management database more scalable have been discussed in Chapter 4. Among the various techniques, one approach that is very useful for storing performance data is that of round-robin database. The round-robin database contains enough storage to hold a fixed number of records and rolls over overwriting the oldest records when there is no more space to write any more records.

One way to understand the operation of the round-robin database is consider the fact that the performance metrics that are collected from a time series. A time series is a sequence of measurements that is made at different instances of time. If the performance metrics are collected at regular periodic intervals, one can characterize the set of metrics as a series $x_1, x_2, x_3, \ldots$ where x_i is the set of metrics collected in the ith period. The round-robin database is simply a way to retain the most recent N entries in the time series where the value of N is determined by the size of the round-robin database. If the current time interval is t, then only the set of metrics generated at time $x_{t-N}, \ldots . x_t$ are stored.

Because the performance data can be represented by the time series, one can try to reduce the size of the time series by increasing the interval at which the performance metrics are polled, or by aggregating the time series records. One way to aggregate would be to replace each set of K records by means of the average of those records. As shown in Fig. 8.1 below, this reduces the amount of information that needs to be stored by a factor of K. Some additional information about the original set of K metrics can be retained by maintaining other attributes such as the standard deviation that

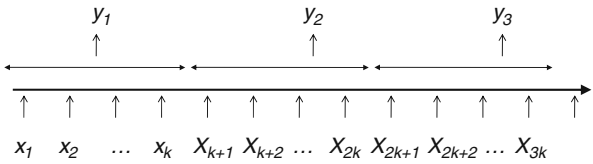


Fig. 8.1 Aggregation of performance data

characterize the spread of the different metrics among the K sample that are selected.

Viewing the performance data as a time series allows one to use other techniques for summarizing time series to compress the metrics that are collected from the performance data. These compressions would usually be done for storage and archival of the data. In order to manage the volume of performance data, it is common to use a hierarchical scheme to store the performance metrics collected by an enterprise.

Figure 8.2 illustrates the scheme for managing performance data in an hypothetical enterprise. The enterprise collects performance data at a frequency of every 5 s from the different devices, and it creates them in a daily metrics database. Let us consider only one performance data – the metrics for server utilization that are collected from any given server every 5 s. The daily metrics database is implemented as a round-robin database that is capable of storing about 6 months of performance data. The daily performance data is maintained in full detail for a month so that any analysis can be performed in detailed data for the purpose of performance troubleshooting. Every week, the daily performance data are compressed and stored into a long-term metrics database. The long-term metrics database contains aggregated metrics that are summarized from the daily performance log. Every 6 months, the information maintained in the long-term metrics database is aggregated and stored into a historical metrics database.

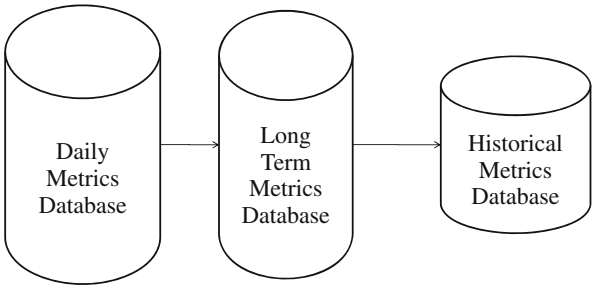


Fig. 8.2 Data aggregation in an enterprise

The size of the performance metrics contained in the aggregated metrics can be reduced significantly by taking advantage of the time-series nature of the performance metric. One can store more information that can be potentially

useful in the future by observing the fact that the time series of performance metric has periodic variations which can be maintained as data are aggregated.

The simplest data aggregation would be to take averages over some intervals, e.g., once every hour when aggregating the performance data from the daily metrics database to the long-term metrics database, and store the mean and variance of the server utilization. In that case, the mean and variance of every 720 records in the daily metrics database is stored in the long-term metrics database. Similarly, the mean and variance for the utilization on a daily basis can be used to store in the historical metrics database, combining the information of 24 records in the long-term metrics database into the historical metrics database.

The simple aggregation method is effective and can meet the needs for an enterprise to look back at the historical metrics to determine long-term capacity planning issues when it only needs to rely on the daily mean information. However, if the enterprise wants to look for the daily variation in its workload from the historical metrics database, it may find that it does not have that information stored to perform that level of analysis.

One option for the enterprise would be to compress all of the information in the daily database store and store it in the long-term metrics database, e.g., by extracting the entire performance records for an hour, compressing it into a zipped file, and then storing that as a record in the long-term metrics database. However, storing this much data may soon start to grow significantly in size, and it is not clear if storing that data serves any useful purpose.

There is a trade-off between the amount of data that needs to be stored and the type of analysis that can be performed on the historically stored information. If the intent of the historically stored information, as is often the case, to determine future capacity planning needs, then one may decide that it is important to store not only the mean and the variance in the historical information for each day but also information that could represent the daily variation in the utilization. One could do it by storing the maximum and minimum utilization observed (for any 5 s interval) as additional fields for the hourly aggregated information in the long-term performance metrics database. Similarly, the maximum and minimum utilization observed during the day for each of the records stored in the long-term metrics database can be maintained in the historical metrics database. This provides more information for subsequent use in capacity planning.

8.3.3 Error Handling and Data Cleansing

An approach of visualizing the performance metrics as time series also provides us with simple techniques to deal with the case of errors in the collected information. Errors in the collected information may happen due to many causes – the software collecting the data may crash and not provide the desired

result for some time period, the network path from the monitoring point to the managed device may be disrupted and the data may not be available, or the device may have failed or decommissioned for maintenance.

In terms of the time-series view of the performance metrics, we have some set of measurements x_1, x_2,...x_K, available, but the next metric point x_{K+1} is missing. We could either skip over the missing information, not storing the record corresponding to that time in the management database or try to obtain an estimate for that.

For some performance metrics, the right estimate may be to repeat the last value as the representative measure, i.e., use the assumption that $x_{K+1} = x_K$. This estimate would work for those performance metrics that tend to change relatively slowly, e.g., the number of threads listening on a port for a server-side application, or the number of servers assigned to a load balancer which dynamically adjusts the number of servers depending on the load. These metrics change relatively slowly compared to the frequency at which they are monitored, and thus repeating the last estimate may be a reasonable estimate.other cases, an extrapolation from the past few values can be used to fill in the missing gaps. A common method to perform the extrapolation is to use linear extrapolation using the values in the recent past. As an example, if the information is being collected at regular intervals, one estimate can be obtained as

$$x_{K+1} = 2x_K - x_{K-1}.$$

If the metrics are not being collected at regular intervals, then a weighted estimate based on the time between the collection points can be used.

The extrapolation provided above only used last two measurements to extrapolate the missing measurement. A more sophisticated approach would be to take some last M measurements, obtain the best-fit line for those measurements, and then extrapolate the missing data from those measurements.

Extrapolation from the past measurements is a good way to estimate metrics that tend to vary rapidly when compared to the periodicity of measurements, such as the utilization or request rate of different devices.

In some environments, there is a delay between the generation of a measurement and when it would be used. As an example, the measurement data may be collected every 5 min, but it will only be used to generate a report at the end of the day. One could then defer the determination of missing data for some time until the next few measurements are received. In those cases, estimation of the missing data can use not only the information obtained before the missing data but also information obtained after the missing data. We could then obtain a better estimate of the missing data by interpolating between the values measured before and after the missing data. Under these conditions, we know the values x_1, x_2, ..., x_K, and the values x_{K+2}, ..., available, but are missing the value x_{K+2}.

A simple interpolation scheme would be

$$x_{K+1} = (x_K + x_{K+1})/2.$$

If the metrics are not being collected at regular intervals, then a weighted estimate based on the time between the collection points can be used. As in the case of extrapolation from the past, the interpolation could also use more than one point on either side and try to do a best-fit linear estimation.

Although it is possible to use non-linear estimating functions in both interpolation and extrapolation cases, they may not be required in the majority of performance measurement cases. Non-linear estimates are computationally more difficult to compute, and if the performance metrics are obtained reasonably frequently may not provide too much of additional accuracy above the linear estimates for most performance management applications.

In many cases, the performance measurement data that is obtained from a reading is not a single value but a record containing of multiple values. As an example, when one reads the performance metrics from a service, one would read the CPU utilization, memory used, disk queue length, etc., at the same time. When the entire record is missing from the data, each attribute of the record can be estimated using the techniques described above. However, sometimes a few fields of a record may be missing, while other fields may be present. As an example, a reading may report the CPU utilization, but may be missing the disk queue length used if there was a problem in reading the disk queue. An estimate of the missing fields can be determined by comparing the correlation among the fields that are present in the partially filled record and the completed records that have been seen in the time series. One could obtain the best-fit line which predicts the missing attributes as a function of the attributes that are present in the record on the basis of the historical records that have been collected, and then use that to estimate the missing values.

8.3.4 Metric Composition

Although many different types of performance metrics can be collected automatically from the different devices in the network, these metrics are not necessarily the ones that the operator of a computer system may want to observe on an ongoing basis. In most of the cases, the information that needs to be reported consists of a summary metrics that is composed from many different individual metrics.

As an example, let us consider the case of the service provider who is hosting paid web services to a customer. The service provider wants to be able to monitor the responsiveness of the web site and wants to maintain the availability and the responsiveness of the metric to be at an acceptable level. The service provider can do this using three alternative methods: (i) by hiring an external web site testing service which can provide reports on

the performance of the web site, (ii) by running a testing application to monitor the web site at a local monitoring site, or (iii) by estimating the performance of the web site application using schemes that monitor only local information available from the monitoring metrics available at the local routers and web sites.

One system implementing scheme (iii) using the metrics available at the web server is described in [2], and a similar system can be developed using information available at a network device from packet headers [3]. The system monitors the metrics available at the local network devices and the web server. Using instrumentation available at the web server, they obtain the round-trip time of a TCP connection established for the web server, a metric that the TCP protocol computes during its normal course of operation, and observe the start and end of a transmission. By adding the estimate of the round trip to the duration of the session, a reasonable estimate of the time spent per connection is obtained. The information can then be aggregated according to the addresses of the different clients, or any other aggregation scheme that is desired.

In these systems, as in many other cases, the information that the administrator is interested in viewing is very different than the information that is available from the statistics collected from the different devices. The information stored in the performance management database is the information collected from the devices. Generally, two approaches can be used to obtain the report required for the desirable metrics from the management database – (i) writing SQL queries against the database or (ii) processing composition operations to convert the metrics.

To illustrate the above examples, let us assume that we are collecting the information available from a system like [2] containing the IP address of a client, the URL accessed, the round-trip time and the duration of the session, and storing it in a performance management database. The report that is desired is the client time performance seen from a customer's intranet whose IP subnet address is averaged on a daily basis.

SQL or simple query language is the existing standard for obtaining information from relational databases. Queries against the database are used to retrieve the metrics required for the task, and then combined to obtain the desired result. In the example given above, a SQL query would be written to retrieve all the records with IP addresses in the specified customer's intranet within the last 24 h and to average them. This information can then be included in the daily report provided for the customer.

In the conversion approach, a second database will be created to store the computed metrics. A SQL query, similar to the previous one, will be written to obtain information from the performance management database and the right metrics computed. Then, another SQL data manipulation operation will be used to store it in a composed metrics database where the records consist of the customer identity and the average value. In essence,

the time series stored in one database is converted into another time series stored in another database.

8.3.5 Performance Monitoring Approaches

Because of the importance of monitoring system performance, many different approaches have been developed for monitoring the performance of different applications, systems, and networks. In this section, we briefly look at some of the techniques used for monitoring the performance of different networks, servers, and applications. These techniques are used to populate the performance management database.

8.3.5.1 Networks

The primary architectures used for monitoring the performance of computer networks include SNMP-based approaches, analysis of network flows, and the usage of the ping tool.

SNMP-Based Approaches: The SNMP protocol and the various MIB specifications associated with it provide a variety of information about the local load, utilization, and packets flowing through the network. The use of SNMP to collect different performance metrics provides a convenient scheme to collect the different information from many different network devices in a standard manner. The monitoring system consists of the SNMP manager which polls the information from the different devices and populates the performance management database.

Flow Monitors: A flow in a network is a stream of packets going between a specified pair of source and destination machines (or networks of machines). Most router vendors support the capture of flows in the network, and their retrieval through a management API. There is also an IETF specification [4] for real-time flow monitoring which allows such flow information to be captured through an SNMP-based approach. For each flow, the number of packets transmitted can be recorded. By exploiting information about the different protocol headers present in the flows, analysis tools that can analyze the performance of specific applications can be written.

Ping Measurements: Many network monitoring schemes are based on the use of the ping tool, a software utility that allows a machine to send a packet to a remote machine and can record the round-trip time for the same. A management station can also request a machine to send a ping request to another machine and record the resulting round-trip time. Many network performance management systems rely on

having a set of machines that are pinging other machines in order to obtain an estimate of different types of delays within the network.

8.3.5.2 Servers

Most server operating systems maintain a continuous track of different performance metrics. Therefore, the performance monitoring scheme for servers deals primarily with the method used to retrieve those metrics from the server. The two primary methods are agent-based and agent-less collection of performance metrics.

In the agent-based approach, a software agent is installed on the server that is being monitored. The agent software collects the performance metrics, performs any aggregation of metrics, and sends the collected metrics to the performance management system. The advantage of having an agent is that agents can use more efficient protocols to transmit the information and perform a variety of preprocessing at the server machine. The disadvantages are that the agents consume server cycles (only an issue if the server is overloaded) and the installation and upgrade of right agents is additional work for configuration management. The exception is when standard agents come pre-installed into the system.

In the agent-less approach, agent software is not installed on the servers, but a remote console logs into the server using the proper credentials and runs a script to collect the required information. The advantage of the agent-less approach is that there is no software installation required on any of the servers. That complexity is traded off against the need to maintain a secure set of credentials that allow the automated scripts to logon to the servers securely. The only drawback in agent-less collection of performance metrics is that all collection needs to be done at periodic intervals, – without a software agent one cannot write a preprocessing software which will send out an alert every time a system's utilization exceeds a given threshold.

8.3.5.3 Applications

Applications are the key components in any computer systems whose performance needs to be managed actively. All other components (servers, clients, networks, etc.) are but supporting infrastructure to run applications with a good performance. Therefore application monitoring is one of the most important aspects of systems performance monitoring.

An application typically consists of a server-side application that are accessed by clients, e.g., a web server accessed by browsers (clients), or a database system accessed by software components requiring access to the data. We look at monitoring the performance that clients experience in accessing the servers. In a few rare cases, the client software can report on the end-to-end performance of the applications (e.g., mainframe terminal emulation software) and that information can be readily collected. However, it is not adopted

in most other clients and is not a feature of the web browser, which is the client of choice in many applications. The techniques that a system administrator can use in these cases include

Active Monitoring: In many types of applications, it is possible to have some artificial clients whose only role is to monitor the performance of the server application. The monitoring clients are written so as to invoke requests that do not change the status of the information maintained at the server-side application. For some applications, such as web sites, most requests are reading of information and there are many requests the monitoring client can invoke without impacting the server. For some other applications, e.g., a bank's account management system, adding and deleting money into individual accounts can have significant implications. In these cases, the monitoring clients may access a synthetic account – set up explicitly for monitoring without any real funds, or performance a series of requests whose net effect cancels out (e.g., a debit of 5 cents on an account followed by a credit of 5 cents).

Active monitoring clients are offered as a service by some companies for monitoring the performance of various web sites, or they can be deployed by an enterprise internally. Active monitoring clients can report the end-to-end performance as experienced by an actual client co-located with them, but may need to be limited to a subset of requests to not interfere with the proper operation of the application.

Proxy-Based Monitoring: A proxy is an intermediary application running in between the client and the server instances, appearing to be a server to the client, and a client to the server. Proxies are commonly used for many different applications and provide functions such as caching of content for scalability or managing security credentials. However, proxies also provide a convenient way for monitoring the performance of the application as experienced by the clients.

A proxy can track the start and the end time of every transaction that is made to the client, and then record the response time, the identity of the client, and the bandwidth exchanged for a transaction between the client and the server. The set of such records can then be processed to obtain a variety of performance metrics.

ARM Correlators: Many industrial applications are implemented in a distributed manner and encompass many different instances of server applications that are invoked sequentially or in another manner. As an example, a check that is presented electronically for a bank may need to be passed through a variety of applications, first through a check validating application, then through a recording application, then through a system that checks for availability of money in the accounts, then through a clearing house for balancing across accounts, etc. In these cases, the performance of an application needs to be tracked and identified across all of the application instances.

ARM is a standard developed by the Open Group [5], a consortium of companies that standardizes on APIs, with the primary focus on Unix systems. It enables applications to provide instrumentation information using a standard interface. The ARM API [6] provides calls that can mark the beginning of a transaction, establish checkpoints in that transaction, mark the end of the transaction, and carry the correlator across different instances of an application. This allows linking of nested transactions. The user of ARM across several applications can enable the tracking of any performance problem in the entire distributed application system. Applications that make calls to the ARM API can be analyzed for their availability and response time by any management tool which understands the ARM specifications.

Network Flow Analysis: An alternative way of monitoring application response time is by capturing and analyzing packets in the network that are traveling between the clients and the servers. It is easy to capture traces of packets generated by an application in the network, or even on the server running the application using normal commands available on the systems. The latter is not advisable since it can interfere with the performance of the actual application.

Once the data are captured, it can be analyzed to track the performance of the application. Knowledge of the protocol used between the client and the server of the application enables one to reconstruct the performance metrics such as throughput, response time, or error rates fairly effective. There is usually a lag in processing and analyzing the data, so the approach works better for an off line analysis of performance, rather than a real-time tracking of application performance.

8.3.6 Performance Reporting and Visualization

In many enterprises, standard operating practices require periodic reporting of the performance of the computer system. Such a periodic reporting may also be a requirement in some business agreements, e.g., service level agreements typically require the reporting of the performance of a supported system at regular periods.

The generation of these reports is usually done using a template-based approach. In the template-based approach, the bulk of the report is in a standard format with some place-holders for the monitored value of the performance metrics that need to be reported. When a new report needs to be generated, the current set of performance metrics are inserted into a copy of the template to make the latest report.

In addition to the reports that are generated, one would also need to provide a visual means for the operator of a system to get an understanding of how the system is performing. In large computer systems, it is traditional to have a

console at the management center where the operators can obtain summary information about the performance of the system. The goal of performance visualization is to enable the operator of the system to obtain a summary of the system performance by means of a quick look at the console.

The following approaches are the common ones used to provide visual indication of the performance of a computer system:

Event Scrolls: In this model, the visual information shown on the console are events alerting the administrators about the potential performance problems that may be occurring in the system. The events are shown as a scrolling list represented to the administrator. The performance problems may be defined as the case of the performance of a system component exceeding specific thresholds, e.g., an alert could be shown as a line on the screen listing a network access point which is dropping too many packets. The number and size of alerts shown on the console are an indication of how the system is performing. When no alerts are shown, the system is performing well. In other words – No News is Good News.

A typical example of an event scroll is shown in Fig. 8.3. Each line in the scroll shows an event identity – a unique identifier for the event, and the node which is affected by it. The third entry in each line indicates the severity level of the event, many administrators classify events into severity levels with the level number indicating how critical the event is. In the example shown below, the severity is more critical as the number decreases, and the administrator has selected to display only events with severity 1 (most critical) and 2 (second most critical) to show in the event scroll. The event identifiers are not continuous because of this selection process applied. The last column describes the type of event using codes such as TH_EXC (Threshold Exceeded) and UT_EXC (Utilization Exceeded). Generally, each company would have their own set of event codes, to which the administrators rapidly get used to, and use as a quick way to identify the type of problem that is happening.

Id	Node	Severity	Type
2987	9.2.6.8	1	TH_EXC
2995	9.12.5.6	2	UT_EXC
2998	9.4.6.5	2	UT_EXC
3006	9.5.7.9	1	IF_DOWN

Fig. 8.3 Example event scroll

While performance alerts are a good mechanism to keep the operator informed about potential performance problems, it does not readily alert the

operator in a situation where no alerts may be reported due to a flaw in the management system itself. Management software, like any other software, is apt to fail at times, and there will not be any indication of an alert when the software reporting the alert itself fails. Additional information, e.g., the time when the alert list was refreshed needs to be displayed on the console to alert the administrator of potential problems in visualization.

Map of System with Color Codes: In some cases, the visualization of performance is provided by means of a map of the computer system that is deployed and using a color-coded scheme to represent the performance of the system. A color-coded scheme may show green on sections of the map where performance is good, red on sections of the map where performance is bad, and yellow where performance may tend to become bad. The definition of good and bad performance can be defined by means of thresholds, e.g., performance is good when the system response time is below a first threshold value, red when it exceeds a second threshold value, and yellow in between the two threshold values.

A sample map showing a map-based approach for performance visualization is shown in Fig. 8.4. The map is shown for a hypothetical company operating a cross-country network in the United States. The performance levels when good may be shown by a green status indicator at the location, while poor performance levels may be shown with a red indicator at the screen. In the figure, the red indicator is shown as the grey circle, while the green indicators are the grey circle with an X. The performance at one of the access points in the south-eastern region appears to be degrading and needs attention of the administrator.

Fig. 8.4 Example performance map

A map of the system is a good representation of distributed environments where the system consists of many different nodes, and the nodes can be mapped onto a geographical environment. Telecommunication companies, which operator large networks, employ this representation in their operation centers. A quick look at the map can inform the administrator about the overall health of the system, and also identify any locations within the computer system which may need attention.

Summary Charts with Color Codes: In some cases, a good visualization of performance is provided by means of summary charts. A summary chart may be a line graph indicating the performance of the system as a function of time. When the performance metric jumps, that may be an indication of a situation requiring operator action. For performance metrics where one just needs to maintain a current indicator value, one may use a gauge to visually represent the performance metric. Another type of summary chart may provide a pie chart of the number of system components with different levels of performance, providing visual summary of the overall system health to the administrator. Figure 8.5 illustrates a display which shows a line graph, a gauge, and a pie chart.

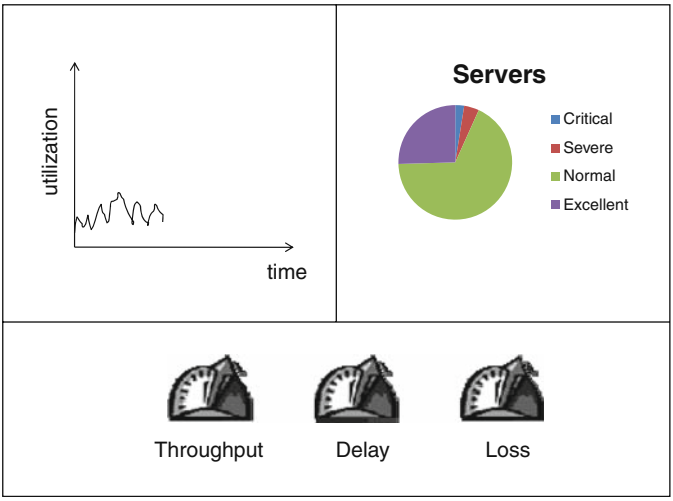


Fig. 8.5 Example of summary charts

The goal of the different visual representations is to alert the operators to the existence of potential performance problems, so that remedial actions to correct the causes of the performance problems can be taken.

8.4 Performance TroubleShooting

Performance troubleshooting consists of two parts: (i) identifying when there may be a potential problem with performance and (ii) taking remedial steps to correct the cause of that performance problem.

In many respects, performance problems can be considered as a special type of fault, and the techniques used for fault management as described in Chapter 6 used for dealing with the root causes of the fault. However, there is one key difference between performance problems and faults as described in Chapter 6. Performance problems generally do not cause the system to stop functioning, and as such are harder to detect than hard faults which cause the system to stop functioning.

We look first at some techniques to identify performance problems, followed by the techniques that can be used to correct potential causes of the performance problems.

8.4.1 Detecting Performance Problems

There are many techniques that can detect if a system's performance is not acceptable. These techniques will usually be applied to process the performance monitoring information that has been collected in the performance management database.

8.4.1.1 Thresholds

A common way to detect performance problems is to define thresholds for acceptable performance. As an example, let us assume that the performance metric being monitored is the end-to-end response time of an application as measured by a monitoring client or composed from metrics available at different networking devices and servers. One could define a threshold for operation that will be considered normal and acceptable for the service being offered. Depending on the performance metric being observed, the threshold could be a higher threshold or a lower threshold. As an example, if the response time were being observed, one would define an upper bound on acceptable response time. On the other hand, if one were observing loss rate as a metric, one would define a lower bound on the acceptable loss rate as the threshold.

A threshold is established based on a variety of conditions, taking into account factors such as the acceptable levels of performance in the system, the type of performance that needs to be supported under any business agreements, and the statistical property of the performance metric. The thresholds that depend on the statistical property of the performance metric can only be truly determined after the system has been operational for some time. The thresholds

that depend on business agreements are usually fixed and need to be determined a priori.

As an example, let us consider a network operator which has provided a service level agreement to its customers that the end-to-end delay within the network between any two points in the same country will be not more than 30 ms. The service level agreement may require refunds or credits to the customers if the performance falls below that level. In order to avoid suffering the economic penalty, the administrators would define a performance threshold at two-thirds of the agreed-upon value during operation, and set of threshold to define a performance problem whenever the same-country delay exceeds 20 ms. This gives the business some time to address the cause of the performance problem when the performance level crosses some threshold.

When the system has been operational for some time, more accurate values of the threshold can be obtained. Let us consider the time-series $x_1, x_2, \ldots x_K$ created by the performance metric. The historical data would provide two metrics, the mean μ and the variance σ for the data that are collected using the time series. A reasonable threshold to use for the threshold would be $\mu + 3\sigma$ when an upper bound is desired and $\mu - 3\sigma$ when a lower bound is desired.

In the case of the network given as an example above, the network provider may observe that their average latency within a single country is 10 ms and has a variance of 2 ms. In that case, the administrator may want to set the upper threshold to be 16 ms to flag a performance problem. That provides the operator with more time to react to a potential performance problem than the previously established threshold of 20 ms.

Performance metrics in any company show cyclical behavior, varying differently during different times of the day. The cycles occur at different scales, e.g., daily, monthly, and annually. In the daily cycle, most computer systems used in a company for business experience more workload during regular work hours, decreases a little during lunch hour, and decreases substantially after work hours. Computer systems used for entertainment show a reverse trend in usage, with a surge during lunch hours at work, another surge after work till late night period, and low usage during nighttime and normal work hours. On the monthly cycle, computer systems used for accounting applications may see a surge in workload toward the end of the month when monthly bookkeeping needs to be completed. Similarly, due to vacations taken by different people, the amount of workload on any company system changes from month to month.

When computing the thresholds, one may choose to establish different thresholds for different periods, taking into account the different cyclical variations on the workload of the system. Setting different thresholds for different times of day would flag out any potential performance problems that can then be investigated for potential root cause and a solution.

8.4.1.2 Statistical Abnormality

The time-series data that are generated by any performance metric has some statistical properties that can be estimated to remain relatively constant under operating conditions [7]. However, when a situation arises which can cause a performance challenge to happen, the statistical properties will change and a performance problem can be flagged by looking for such abnormalities. The time series can be passed through a system that compares the statistics and looks for statistical aberrations [8, 9], automating a substantial part of the analysis.

In any given environment, a different set of performance metrics may be the best one to use in order to determine that a performance problem situation exists [10]. In many IT systems, server utilization metrics are used as indicators of performance problems. Other systems may use response time as a metric, or the amount of offered load on the system.

8.4.1.3 Help Desk Reports

A definite indication of performance problems in the computer systems are reports and calls to the help desk. The help desk is the location where users of the computer system call when they are experiencing problems, such as system running slow, or not being able to access the applications properly. When the volume of calls reporting performance problems increases on the help desk, that is, a certain indication of a performance problem that is affecting the system.

It must be pointed out that the use of help-desk reports to identify the performance problems has one significant drawback. A customer who has called the help desk has already experienced bad performance. The system management software should enable the detection of performance problems before the customer had a chance to call the help desk.

Other methods discussed earlier in this section enable the detection of such performance problems before they could be noticed by the customers. However, help-desk complaints provide a way to catch the few cases in which the performance problems are not readily captured by the statistical abnormality detection or threshold schemes.

8.4.2 Correcting Performance Problems

Once a performance problem has been identified, it needs to be remedied. In order to do that, the root cause of performance problems needs to be identified and appropriate action taken to manage the problem.

When performance problems are encountered, the first step to address them is to determine the root cause. Once the root cause is determined, then the performance problem can be addressed. Some of the common causes of performance problems and ways to address them are listed below:

8.4.2.1 Misconfiguration

A common cause of performance problems is improper configuration at one of the devices in the computer system. The improper configuration is not something which causes a device or software package to work incorrectly, it simply makes it slow. As an example, many multi-threaded applications are designed with a thread responsible for handling a single incoming request. If the number of threads configured at the application is smaller than the average number of pending requests, then the application will experience large delays even if the actual machine running on the server is relatively underutilized.

Interactions among the configuration of two applications can also cause a degradation of performance. As an example, let us consider a web application server which is accessing a database server. Assume that an authentication system invokes an LDAP directory to validate user credentials. There are several LDAP directory systems installed in the company, and for convenience sake, they are partitioned so that they redirect the authentication system to whichever directory contains the credentials of the user. An authentication system configuration should list the directory server which is most likely to have the credentials of the user that is being validated. Otherwise, an authentication will take two interactions with the LDAP servers to validate any credentials, while a reconfigured system would only take one interaction, speeding up the performance. The configuration of the authentication system needs to take into account the design of the directory infrastrcuture.

When any persistent performance problem is encountered, checking the configuration of the system is likely to identify any potential configuration issues, which can then be corrected.

8.4.2.2 System Changes

A common cause of misconfigurations leading to performance problems are any changes in system configuration made in the recent past. A change in system configuration may have resulted in an inadvertent configuration mistake which causes the system to work poorly.

In general, system changes have an impact on the performance metrics that is being measured. The impact of a system upgrade is usually expected to improve the performance. However, when the performance is impacted adversely, the change in configuration can often be identified as the culprit.

Rolling back the change which caused the performance degradation is often a good way to address such performance problems.

8.4.2.3 Workload Growth

When a system is designed, it is designed with a specific workload level in mind. As time passes, the workload on the system is likely to change, mostly increasing for most successful applications and systems. As the workload grows, one or

more components in the system may get overloaded, reaching the limit of the workload that they can handle. A gradual increase in the sustained workload of the system would require the change in the design of the system.

Usually finding the component that is overloaded and increasing its capacity works well to handle the growth in the workload.

8.4.2.4 Workload Surge

The rise of the Internet has seen many applications and systems running and accessed via the web with a very large number of potential clients. Any web site can be accessed by a large number of users, although there are few visitors for most of the Internet-connected systems. A phenomenon amplified by the existence of many potential clients is that of a sudden workload surge, or flash crowds. A flash crowd is a phenomenon when workload on a site increases very rapidly in a very short amount of time. It may happen when a relatively unknown web site may be mentioned in a popular television or radio show, causing an increase in interest among several users who may be unaware of its existence.

A flash crowd requires additional capacity at very short notice. Some of the large hosting providers maintain a pool of free servers that can be rapidly commissioned into providing additional capacity for existing or new customers when their workload increases. It is useful to have such capacity in all types of installations. For smaller installations, where having a spare pool may be too expensive, arrangements can be made with providers who can dynamically provide additional capacity at short notice. Several Internet-based service providers are able to offload the traffic and host them from their servers.

8.5 Capacity Planning

The task of capacity planning is to determine when specific components in a computer system would need to be upgraded, based on the current estimates of the workload that is coming into the system. As workload grows on a system, one needs to know when to install additional capacity into the system in order to design a better computer system.

The first step in capacity planning is to clean out the time-series data on performance metrics that are collected. The cleanup stage involves removing the data from the time-series that shows statistical abnormality (as discussed in Section 8.4.1) and filling in missing data (as discussed in Section 8.3.3). Once the data have been cleansed, one can look at using the performance metrics to determine when additional capacity may be needed into the system.

The task of capacity planning can be viewed as that of estimating when the workload of a system will exceed its capacity. The workload is characterized by the time series which consists of the performance metric. The capacity is usually

a threshold bound which the workload should not exceed. The following are some examples of capacity planning in different types of computer systems:

- A branch office connects to the Internet with an OC-3 link (155 Mbps). The network bandwidth utilized over the course of the last year is known. Good performance requires that the maximum utilization of the network link does not exceed 80% of link capacity (i.e., the total bandwidth used does not exceed 125 Mbps). We need to estimate when the link will need to be upgraded to a higher capacity one.
- A web site operates using a load balancer and three servers to host the current site. The response time for the server needs to be maintained below 300 ms. The statistics for response time over the last 2 years are known. We need to estimate when the web site will need more than three servers to maintain its response time below the threshold.
- A server used to host banking transactions is operated so that its average utilization over any 1 minute period does not exceed 70%. Statistics for the last year of the operation of the server, enumerating the utilization per minute are known. Estimate the time when the server will need to be upgraded.

In all of the above examples, a time series of a performance metric is known, and we need to estimate when the time series is going to exceed a threshold. In order to make this determination, we need to find out what the growth rate of the time series is going to be.

There is one key difference between the extrapolation of the time series for long-term capacity prediction and for the extrapolation/interpolation of the missing data in the time series. In the case of extrapolation for missing data, the value of the time series being estimated was very close in time to the values for which the time series is known. At these small scales of time, one could make the assumption that the time series was approximately linear and use linear regression to find the missing data. However, in the case of forecasting for capacity planning purposes, the interval over which the capacity needs to be predicted is much larger, at the scale of months or years where the linear approximation would not hold true.

The approaches used for estimating the rate of growth revolve around the scheme of building a model that can predict how the time series is evolving. A model is a relationship that can predict the value of the next element in the time series given the past history of the time series. Some of the models that can be used to predict the future of the time series are described below.

8.5.1 Simple Estimation

In the simple estimation method, we try to find a growth curve (most commonly linear) that best fits the data that are available for the performance metrics. As

an example, let us consider the branch office with an OC-3 link example that was provided above. We could take the average of all the 1 minute usage in an hour, and then pick the hour with the busiest average as representative of the bandwidth needs for that day. Then we take the average of the daily needs as the representative utilization for a month.

Using the monthly utilization data resulting above, we try to estimate the function that predicts that monthly estimate. Let us say the function turns out to be:

$$\rho = 0.25 + 0.01t,$$

where ρ is the utilization and t is time as measured in months. Let us say that we are currently at $t = 40$. We can estimate that the time when ρ becomes 0.8 will be at $t = 55$, or 15 months in the future. Thus, the link will need to be upgraded in about 15 months to a higher speed link in order to not experience performance problems.

A similar method of estimation can be used for the other two examples illustrated above. The simple methods provide a means for a rough estimation of the time frame by which capacity upgrade is needed.

Once the metrics are obtained, more sophisticated models can be used to estimate the time series and the seasonal variation that is experienced in that time series. Then, the model can be used to estimate when the time series will cross a threshold. Two useful models for building the time series are described in the next sections. A more thorough discussion of the principles of time series analysis can be found in [11–14].

8.5.2 ARIMA Models

ARIMA is a short name for autoregressive integrated moving average model. ARIMA models assume that relationship between the elements of a time series consists of three parts: (1) an autoregressive part, (2) a contribution from a moving average, and (3) a part involving the derivatives with respect to time of the time series.

The autoregressive part of the model states that the value of an element in the time series can be predicted by a linear weighted combination of some past p members of the time-series. In other words,

$$x_t = \alpha_{t-1}.x_{t-1} + \alpha_{t-2}.x_{t-2} + \alpha_{t-3}.x_{t-3} \ldots + \alpha_{t-p}.x_{t-p}$$

where the various α_i are constants. If p terms from the past are used in the model, the model is said to have the degree p.

The moving average part of the model states that an adjustment needs to be made to the autoregressive part due to the errors that may have been made in

the estimation of previous elements of a time series. Since our prediction is being made for a long interval into the future, an estimate of the error that may creep in needs to be taken into account. The adjustment made to the predicted value using this component is given by

$$\Delta x_t = \beta_{t-1}.x_{t-1} + \beta_{t-2}.x_{t-2} + \beta_{-3}.x_{t-3} \ldots + \beta_{t-q}.x_{t-q},$$

where the various β_i are constants. If q terms from the past are used in the moving average model, the moving average model is said to have the degree q.

A combination of the two leads us to an ARMA [p,q] model or an Autoregressive Moving Average model which states that

$$x_t = \alpha_{t-1}.x_{t-1} + \alpha_{t-2}.x_{t-2} \ldots + \alpha_{t-p}.x_{t-p} - (\beta_{t-1}.x_{t-1} + \beta_{t-2}.x_{t-2} \ldots + \beta_{t-q}.x_{t-q}),$$

where p and q are the parameters of the model.

The derivative of a time series is the time series that results from taking the difference between two consecutive time-series values and dividing them by the time elapsed between them. If we assume that the time interval between consecutive measurements is 1, then the derivative time series is obtained by taking the difference between consecutive members, i.e., the series $x_1, x_2, x_3, \ldots$ will have a derivative series given by the terms $x'_1, x'_2, x'_3, \ldots$ with the relationship

$$x'_k = x_{k+1} - x_k$$

ARIMA models are defined using three parameters, p, q, and d. In the ARIMA model, the time series is differentiated d times, and then the best-fit ARMA (p,q) model is used to estimate the data, and the resulting ARMA model integrated back d times to obtain a model for the original time series.

Different software packages are available that simplify the task of applying ARIMA models. In order to fit an ARIMA model, we first need to determine the parameters p, q, and d that are appropriate for fitting the data that is available. The choice of the parameters is usually done by fitting the model on a smaller subset of the total available time series, and finding the smallest values of p, d, and q for which the goodness of fit is acceptable. Once the values of p, q, and d are determined, the model is used to predict the time series in the future, and the time when the threshold is expected to be crossed determined.

8.5.3 Seasonal Decomposition

In most performance metrics that are collected over a long period of time, it is common to find seasonal or daily cycles in the performance data. In general, the time series reflecting performance data can be broken down into the following components: (a) a trend which shows the increase of decrease in the average

value of the time series, (b) cyclical variations representing changes in the performance metrics in daily, monthly or annual cycles, and (c) random fluctuations in the value.

One would like to know seasonal variations in order to understand the variation of system performance. An approach to determine the cyclical and seasonal variation is to use the Fourier transform of a time series. For any function, the Fourier representation of a function is its representation as a sum of multiple sine and cosine functions, each sine/cosine function occurring at regular periods. The Fourier representation of a time-series is of the format:

$$x_k = A_0 + A_1 \sin(k) + A_2 \sin(2k) + \ldots A_M \sin(Mk) + B_1 \cos(k) + B_2 \cos(2k) + \ldots B_M \cos(Mk),$$

where M, k, and the various A_i and B_i are constants.

Given a time series in the regular format, our first task would be to identify the various constants which provide the Fourier representation of that series. The software to do such conversion is readily available from many sources. Then, we identify those sinusoidal terms (sins or cosines) that are large compared to others. These sinusoidal components represent the seasonal cyclic variations of the time series.

As an example, let us assume that we have a performance metric which is measured every 5 min given by

$$x_k = 5 + 20 \; sin\,(60k)$$

and the other terms are negligible. Then the performance metric has a regular cyclical period occurring every 5 hours.

Once the seasonal components have been identified, capacity planning removes the cyclical components and estimates the trend in the growth of the overall system utilization. If the main dominant cycles are known, then one could average the metric over the period of the dominant cycle. This averaging eliminates the cyclical variations. Then, one can fit a linear estimation of how the workload is growing through the cycles and determine when one would run out of capacity. In the above example, the capacity planning exercise would average the performance metric over the period of 5 hours and estimate the growth of the metric across the 5 hours period.

8.6 Accounting Management

Accounting management for computer systems is largely related to the question of billing and chargeback in companies. In environments where such an accounting is not a significant issue, this aspect of computer management may not be as dominant. As an example, if a company does not implement

any chargeback scheme, it may opt not to have an accounting management system in its operations center.

A computer system is designed to support several users. At the end of a period of usage, e.g., end of each month, each user may need a report of the various usage of the system they have made. Typically, the report on the usage of the resources is provided to a billing or accounting department, which is then responsible for creating a chargeback or bill for the user. Billing and charging practices are not within the scope of systems management, but determining the resource usage by each customer that provides the information needed for billing is considered part of systems management.

Some examples of various aspects of accounting management:

- A company provides Voice services to its customers. At the end of each month, a listing of numbers called and the minutes used by each of calls needs to be generated for each customer.
- The IT organization in a company provides a network linking different site facilities. At the end of each month, each site provides a summary of the bandwidth that was consumed inbound and outbound by the site.
- The IT organization in a company operates e-mail services for different departments. At the end of each quarter, it sends a report listing the total number of email users in each department, the total number of email messages exchanged, and the amount of disk space the users are taking on the email server.

Accounting management thus consists of identifying the users and determining the resources consumed by each of the users. To a large degree, accounting management consists of producing reports from the performance data such as throughput or transactions log that are available at a device. The only additional complexity is identifying which of the different transactions or requests belong to each of the user.

The simplest approach to address such identification is that of assigning a dedicated device to each of the users. As an example, when Internet access service is provided, each customer gets a dedicated access box (a cable modem or an ISDN router). When a customer gets a voice service, he or she gets a dedicated phone number attached to a specific phone (for cellular phones) or to an access box. Then, the transactions associated from that dedicated device can be readily associated with the customer. In any environment when the logs or performance metrics collected from a device provide details about a user, the collected logs and metrics provide a clean way of reporting resource usage by each customer.

In other cases, the access device may be shared among different users. In some cases, the access devices may be used only by one user at a time, but different users may access the device at different times. An example of this is provided by the infrastructure for mobile employees supported by many companies. Instead of providing fixed offices, companies provide a facility by which an employee can come to a building and obtain access to a phone and network

access to the company's intranet by authenticating themselves of one of many "mobile" offices. This allows a traveling employee to be able to access services can as print, fax, telephone, and email at any location as if they were at their original office using the same numbers. A mobile office may be used only by one employee at a time, but it can be used by different employees belonging to different departments at different times.

In the case of such devices, their resource usage logs need to be examined to determine which user was accessing the device at various times. Some companies would produce a report on a department-wide usage, rather than (or in addition to) a report per employee, and they need to map the employee to a department which is readily done by means of organization charts and directory services on the company. Then, the resource utilization reports can be generated as required.

The last case is that of devices which are shared by several users concurrently, e.g., a wireless access point of a company. In these cases, the logs reporting the sign-on of a user to the access point needs to be collected and then analyzed to determine which user was active. Each line in the logs of access is analyzed and associated with the user and the department that used the resource, so that the proper resource usage report can be generated.

Accounting management is more critical for resources which are expensive and whose usage needs to be monitored, or costs distributed to different departments. As the price of IT equipment and infrastructure falls, some of the companies are moving away from a detailed report on the usage of cheaper resources.

8.7 Summary

In this chapter, we looked at the topics related to performance management and accounting management. Performance management consists of three primary functions – monitoring the performance of systems to produce any require reports, identifying when performance is becoming unacceptable and resolving performance problems, and predictive capacity planning to avoid performance problems in the future. We looked at the different techniques used to perform each of these three functions.

Associated with performance monitoring is the issue of accounting management, which consists of generating resource usage reports for the different devices. These resource usage reports are typically used for billing different customers for their usage.

In the next chapter, we look at the subject of security management.

8.8 Review Questions

1. Why is performance management important in an IT environment?
2. Discuss the advantages and disadvantages of using over-provisioning as a way to manage performance of computer systems and networks.

3. What are some of the approaches to manage performance when a computer system is operating over or close to its capacity?
4. Why is the round-robin database suitable for storing performance metrics? What advantages does it offer over a traditional relational database?
5. How can missing performance data be interpolated using the time-series model.
6. Describe the different techniques that can be used to monitor the performance of (i) networks, (ii) applications, and (iii) workstations and servers.
7. Why is proper visualization an important aspect of performance management in large computing environments?
8. Why are performance problems harder to detect and diagnose than hard faults in a computer system?
9. Why are data extrapolation methods used for capacity planning significantly different than methods used for data cleansing?
10. What is the ARIMA model, and how can it be used to determine the capacity needs of a system?

References

1. V. Paxson and S. Floyd, Wide-area trac: The failure of Poisson modeling, IEEE/ACM Transactions on Networking, 3(3):226–244, June 1995.
2. D. Olshefski, J. Nieh, and D. Agrawal, Inferring client response time at the web server, Proceedings of the 2002 ACM SIGMETRICS Conference, Marina Del Ray, California, pp. 160–171, June 2002.
3. F.D. Smith, F.H. Campos, K. Jeffay, and D. Ott, What TCP/IP protocol headers can tell us about the web. ACM SIGMETRICS Performance Evaluation Review, 29(1):245–256, 2001.
4. RMON Reference.
5. Computer Measurement Group – ARM Working Group, URL http://cmg.org/regions/cmgarmw/index.html
6. M. Johnson, The Application Response Measurement API Version 2, Available at URL http://cmg.org/regions/cmgarmw/marcarm.html.
7. P. Hoogenboom and J. Lepreau, Computer system performance problem detection using time series models, In: Proceedings of the USENIX Summer 1993 Technical Conference, Cincinnati, Ohio, June 21–25, 1993.
8. I. Turbin, Capturing workload pathology by statistical exception detection system, Proceedings of the Computer Measurement Group, 2007.
9. R. Kaminski, Automating process and workload pathology detection, Proceedings of the Computer Measurement Group, 2003.
10. J. Hellerstein, An approach to selecting metrics for detecting performance problems in information systems, Proceedings of the 1996 ACM SIGMETRICS Conference, Philadelphia, PA, May 1996.
11. G.E.P. Box, G.M. Jenkins, and G.C. Reinsel, Time series analysis, Forecasting and Control, 3rd ed. Prentice Hall, Englewood Clifs, NJ, 1994.
12. P.J. Brockwell, and R.A. Davis, Introduction to Time Series and Forecasting, 2nd ed. Springer-Verlag, New York, 2002.
13. J.D. Hamilton, Time Series Analysis, Princeton University Press, Princeton, NJ, 1994.
14. A First Course on Time Series Analysis, http://statistik.mathematik.uni-wuerzburg.de/timeseries/

Chapter 9
Security Management

The goal of systems management is to try to keep computer systems and networks running properly and flawlessly. In order to do so, computer systems must be accessible to the people who are authorized to use them, and not be accessible to anyone who is not authorized to use them. An unauthorized person can cause several problems in the operation of a system, including denying access to legitimate users, disrupting the operation of the system, or cause the system to behave in ways that could be detrimental to the users. As a result, security management is a key aspect of any type of computer systems management.

Information security management can be divided into the five aspects of authentication, confidentiality, integrity, non-repudiation, and availability [1]. Authentication is the job of ensuring that any user or program accessing the information is identified correctly. Confidentiality means that information should be visible only to authorized users. Integrity refers to ensuring that information or systems containing the information are not corrupted. Non-repudiation means the source of any information should not be able to deny responsibility for creating the information. Availability is the task to ensure timely and reliable access to services for those authorized to use them. Each of these aspects is applicable in the context of computer systems and networks management.

Each aspect of information security has a specific meaning when applied in the context of computer systems and network management. Authentication translates to the proper management, configuration, and operation of systems and software used to validate users trying to access a computer system. Confidentiality consists of managing access controls so that only authorized users can access a computer system, and that data in transit or stored on the system does not get exposed to unauthorized users. Maintaining the integrity of a computer system requires validating that no unauthorized software is installed on the computer systems. Such unauthorized software includes malware, viruses, and Trojan horses. Non-repudiation means the ability to have sufficient information to track the system or person at fault for any security-related incident. Availability in computer systems from a security perspective involves preventing denial of service attacks on the system. Such attacks prevent legitimate

D.C. Verma, *Principles of Computer Systems and Network Management*,
DOI 10.1007/978-0-387-89009-8_9, © Springer Science+Business Media, LLC 2009

users from accessing the system, e.g., ensuring that the system operates in the presence of a denial of service attack.

There is an important caveat that needs to be made regarding computer security. There is no known meaningful way to guarantee absolute security in any computer system. The only way to absolutely guarantee computer security is to shut down all equipment, lock them in a vault, and throw the key away. However, that does render the computer system rather useless. Any scheme to manage the security of a computer system is a trade-off between usability of the system and the preventing potential damage by an attacker. Any security mechanism in a computer system has an associated degradation in ease of use, increase in cost, and decrease in performance. Furthermore, there is a continuous escalation in the capabilities of intruders, attackers, and other evil entities that try to attack any computer system, and security mechanisms continually need to evolve to address such an escalation. The set of security measures to be put into place and the amount of risk to take are business decisions that need to be taken in the context of the specific environment in which a computer system is being used.

We begin this chapter with a discussion of some common principles and technologies that are used for managing the various aspects of security in all kinds of computer systems. This is followed by individual sections on issues involved in managing the security of personal computers, networks, and servers. The last section in this chapter discusses the operational principles and common tasks involved in managing the security of computer systems.

9.1 General Techniques

In this section, we look at some of the common technologies and techniques that are used to manage security in different types of computer systems. Many of the techniques described in this section are based on the principles of computer cryptography, so it is worthwhile to quickly look at the basic underlying concepts of cryptography. After a quick introduction to cryptographic principles, we examine generic techniques for authentication, access control, integrity, and prevention of denial of service attacks in computer systems.

9.1.1 Cryptography and Key Management

In its basic form, cryptography converts a piece of information that can be interpreted by anybody (a plain text) into a piece of information that is mangled (encrypted text) in a way so that it is accessible and interpreted to only a limited set of users. Such users typically are able to interpret the encrypted text by converting it back to plain text using a privileged piece of information (keys). Different cryptographic approaches have different

assumptions about key creation and sharing, and use different algorithms to convert back and forth between encrypted text and plain text. A brief synopsis is provided in this section; a more detailed discussion can be found in several other references [2–4].

Two broad classes of cryptographic schemes are those based on private keys and public keys. In private key cryptography, the same key is used for the two operations of converting plain text to encrypted text and vice versa. If the same key is not used for both operations, the key for one operation can be easily obtained if the key for the other one is known. The keys are kept as secret among the different parties that want to communicate in a secret manner. The digital encryption standard or DES [5] is an example of a private key cryptography algorithm.

In public key cryptography, the privileged information consists of two related keys, one which is kept secret (private key) and the other which can be made public (public key). The public key can be readily determined if one knows the private key, but the private key cannot be easily determined for knowledge of the public key. Information encrypted using the public key can be decrypted only using the private key, and similarly information encrypted using the private key can only be decrypted using the public key. If information is to be transmitted securely from a sender A to a receiver B, A can use B's public key to encrypt the message and only B can decrypt it. A special case of public key cryptography is identity-based cryptography in which the public key of an individual is a part of the identity (name or address) of the individual.

The steps by which an entity A sends a secret message to another entity B using private key and public key techniques is shown in Fig. 9.1. As shown, the basic steps in converting the message into an encrypted message which is transmitted over an insecure channel and decrypted by the receiver are the same. Only different keys are used for the encryption and decryption process.

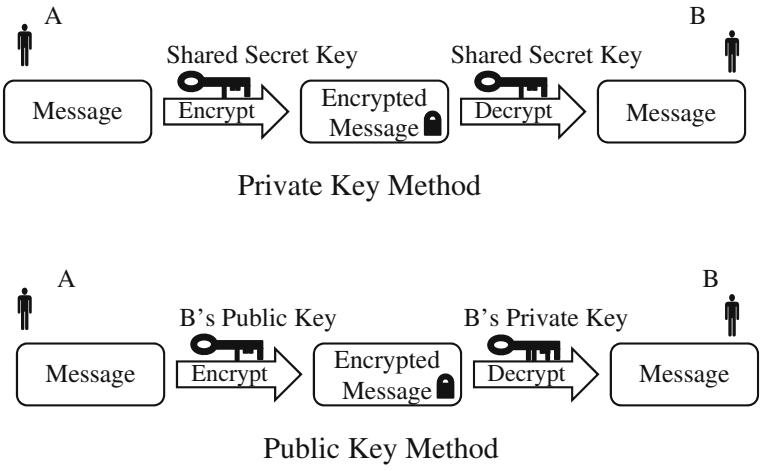


Fig. 9.1 Message transmission using private and public keys

When either private keys or public keys are used with a large number of entities, key management becomes an issue which needs to be addressed. Consider a set of devices that are using private keys to secure communication among each other, e.g., a group of handheld radios used by military, firemen, or policemen. Each device needs to have the private keys for each other device which it may need to communicate with. If there are N devices, each device needs to have $(N-1)$ secret keys if each pair of communicating devices needs to have their communication secured. Since devices may have limited memory, they may not be able to store all of the keys for a large number of devices.

A key server provides a convenient way to manage keys for such communication. The key server is trusted by all devices to provide the right keys to them. Each device needs only to have a key which enables it to communicate to the key server. The device can then obtain the keys for any other device it needs to communicate with by querying the key server for the same.

In practice, a common mode of operation is for the key server to preprogram the keys on a device that enables it to communicate with other devices for a particular operation. Private keys and preprogramming of keys are used frequently in communications using handheld radios for public safety officials (e.g., firefighters or police) and military personnel, since the extent and number of devices one typically communicates in these contexts are usually limited and can be predetermined in advance. The communications between the key server and devices need to be done securely; typically using physical proximity for the first time a device is given its keys. Subsequent keys can be provided over the air as long as the device is able to communicate securely with the key server.

The key management process for private keys using a key manager is shown in Fig. 9.2. Device A wants to communicate with devices B and C. It sends a list of devices it wants to communicate with the key manager. This communication is protected using the key that A is programmed to use with the key manager. The key manager creates keys that enable devices to communicate pairwise, key AB for A and B to communicate together and key AC for A and C to communicate together. The key manager also needs to send the key AC to device C and the key AB to device B. This requires that the devices B and C be in contact with the key server so that they can receive the new keys.

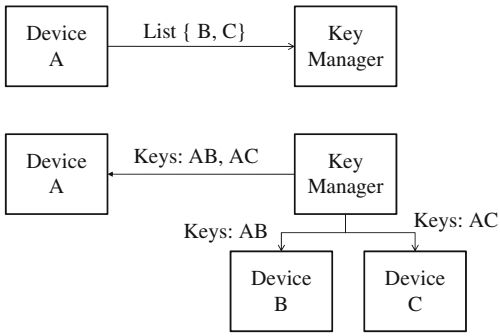


Fig. 9.2 Key management for private keys

A challenge arises in management of private keys if the key manager is not in contact with devices B and C at the time it gives out keys to device A. One solution to this problem is to give out a fixed number of keys to each device in advance. The device which is being programmed is given an index to the key list of the other device. The steps required for managing keys using this approach is shown in Fig. 9.3. Each device which is programmed by the key manager is given a list of hundred keys. When device B communicates with key manager, it gets the hundred keys even if it is not programmed to talk to any other device yet. When device A will contact the key manager with a request to issue keys to communicate with device B, device A would be told that it can use key index 53 to communicate with device B along with the corresponding keys. When A would initially establish contact with device B, it would identify that it wants to use the 53rd key stored in the pre-issued list of B to communicate with it. Having done that initial negotiation, A and B can communicate with each other.

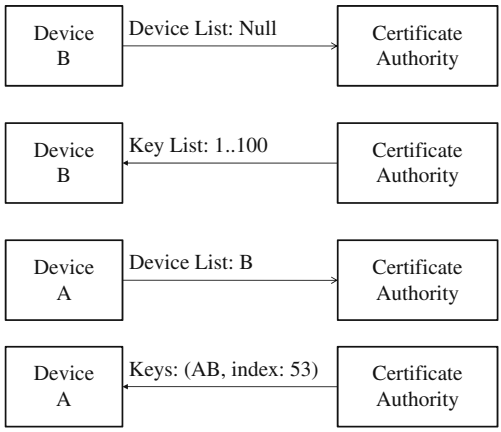


Fig. 9.3 Key management for private keys

In communications that happen on the Internet, public keys are more commonly used. Public key schemes require an infrastructure by which the public keys of parties involved in a communication are provided to other members involved in the communication. As in the case of private key communications, such an exchange requires obtaining the public keys from a trusted entity. The trusted entity in public key communications is called a certificate authority. The job of the certificate authority is to issue certificates, which are digital documents that provide the public key of a computer (or another entity). The certificate is signed using the private key of the certificate authority, whose public key is widely disseminated and known to all communicating parties. Any receiver can validate the authenticity of the signed certificate using the public key of the certificate authority. It does need to check back with the certificate authority in case the certificate has been revoked after issuance.

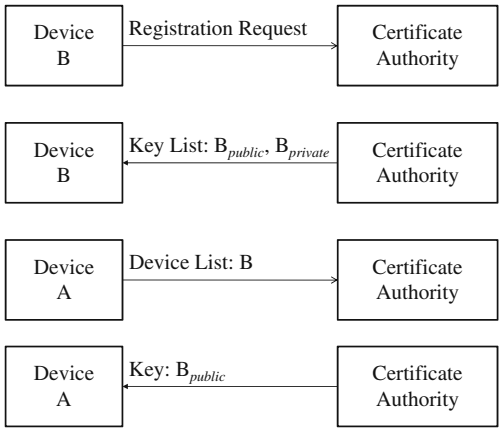


Fig. 9.4 Key management for public keys

The process of key management for public keys is shown in Fig. 9.4. Device B registers with the certificate authority to obtain a pair of private key and public keys for itself. When device A wants to communicate with device B, it queries the certificate authority for the public key of B. Device A is given that public key using a certificate signed by the certificate authority.

In addition to the certificate authority, public key infrastructure would also include other servers known as the registration authorities and validation authorities. These are the two functions a certificate authority performs, and the validation and registration authorities are in effect proxies acting on behalf of a certificate authority for a more scalable operation. Furthermore, there can be multiple certificate authorities, one certificate authority corresponding to various portions of the network.

There are simpler models for managing keys than using a hierarchy of certificate authorities (the normal PKI or public key infrastructure [6]), but they make assumptions about simpler trust relationships existing among communicating parties. These include mechanisms where a certificate can be self-signed by an entity without requiring a certificate authority. Such mechanisms are used in protocols such as Pretty Good Privacy [7].

In any computer system, the systems administrator has to determine what type of cryptographic schemes are to be used, and how the appropriate keys or certificate ought to be managed. Assuming that the certificates or keys are managed well, other types of security functions can be performed with an increase sense of security.

9.1.2 Authentication

Authentication is the task of identifying that a user (either human or software entity) is indeed who he, she, or it claims to be. Almost all aspects of authentication include a user trying to access a system, and the user being challenged to prove that it is indeed who it purports to be.

The general process of authentication has two concepts – a user identity and a set of credentials. The user claims to have a certain identity and presents a set of credentials to support that claim. The system that is authenticating the user validates that the credentials belong to the user with the specified identity.

Authentication is required for access to physical systems as well as computer systems. In an office, employees may be given identity cards which are issued by an appropriate authority, and can be used to authenticate the identity of anyone trying to access the premises. The credentials are the identity card, and the identity of the user is the face which is usually compared against the photo on the identity card.

In addition to identity cards, another common method to authenticate humans in physical offices is by means of biometric information. The credentials are the biometric information associated with a user, e.g., the fingerprints, a scan of the eyes, or some set of ratios of the facial features which can uniquely be associated with a person. A person trying to access a facility provides the name and the biometric credential (e.g., the fingerprint or eye scan) and the system validates it against the credential stored in its database. In some cases, the authentication system may just match the credentials against a set of stored biometric data and automatically infer the user name from them if it matches.

The more common case of authentication that a system administrator has to deal with is virtual access of computer systems by users, who may be either local or remote. In this case, the identity of the user is a user name that is provided for access, and the credentials are an additional piece of information provided. The most common type of credential that is used is a password [8]. The underlying assumption is that passwords are kept private and known only to the user with the specific identity. By comparing the identity of the user and the password that is stored there, the user can be authenticated.

When using passwords for authentication, several issues need to be considered. The set of all passwords need to be accessible to the validation system, but exposing all the passwords to any single individual increases the probability of breach. Therefore, the typical paradigm is to store passwords in the authentication system using a one-way function. A one-way function is a program that converts a password into a mangled version easily, but it is very hard to retrieve the original password from the mangled version. The passwords are checked by converting them to the mangled version and checking for equality of the mangled versions. The method for securing passwords in this manner can also be used with other types of credentials.

In place of passwords, other types of credentials can be used. One example of such a credential is a shared secret like the password which can only be known to the user and the validation system. Another type of authentication can be done using public key cryptography. If each user has a public key and a private key, then the user just has to show that it knows the private key. The validation system only needs to store the public keys, which can be dealt without much security restrictions. When a user tries to log on with a stated identity, the validation system can pick a random message, encrypt it with the public key to

the specified identity, and send it to the user. The user is then required to decrypt the message and send back the original random message. Since only the user is able to do the decryption, this scheme serves to authenticate the identity of the user.

Secret credentials such as password or even the private key in a public key-based system are subject to compromise over long term. It is common practice in many enterprises to require the credentials to be changed at regular intervals to prevent potential compromise. The frequency at which the credentials are required to be changed ought to be selected so that it does not become too onerous on the actual users.

9.1.3 Confidentiality/Access Control

Access control is a counterpart of the authorization step in computer security. While authorization results in validating the identity of an entity in the computer system, authorization is the process of determining what the level of access that authorized user ought to be given within the system.

Authorization can be viewed as a mapping between the different users of a computer system and the resources that exist in the computer system. Corresponding to each pair of user and resource, either the user is authorized to access the resource to perform a particular operation (e.g., read, write, or delete) or it is not authorized to do so. One can define an access control matrix as a matrix with resources enumerated along the rows and users enumerated along the columns, and each entry representing the permissions granted to a user on a specific resource. Authorization and access controls mechanism provide a way to succinctly represent the entries in the access control matrix.

When the set of authorized users are enumerated for each resource, the resource is said to have associated with it an access control list. When the set of authorized resources are enumerated for each user, the representation is called a capability. A capability for a user may be signed or certified by a trusted party and that allows a user to present the capabilities at any time.

One common way, at least in many military domains, to succinctly represent access controls and capabilities is by means of clearance levels. In this representation, levels of access are defined, e.g., levels 1 through 4. Each resource is categorized as belonging to one level, and each user is categorized as belonging to one level. The access control list for any resource consists of the users who are categorized at levels lower than the level of a resource. The capability of a user consists of all resources at a level higher than his/her categorization. Thus, level 2 users can access all resources at category 2, 3, and 4, while a level 2 resource can be accessed by any user at level 1 or 2.

A more generalized form of having multiple levels of security is to consider a label that is assigned to each of the users, and another set of labels that is assigned to each of the resources. One can then define a table between the user

labels and the resource labels indicating whether a user with that label is allowed to access a resource with a specific label or not. As a generalization, more than one label can be allowed to be associated with a single user.

One specific example of access control using labels is role-based access control. In this case, a role is a label assigned to a user. A user can have more than one role, with each role indicating a function the user may need to perform (e.g., an employee, a human resources person, a computer administrator). The labels associated with resources are usually to group them into several categories, usually in a grouping of some nature. Role-based access control specifies which resource a user with an assigned role is allowed to access.

Another commonly used technique to control access in many areas is the use of a black list or a white list. A black list is a list of users that are not allowed to access the resource. A white list is a list of users that are allowed to access a resource. When a black list is used, users are allowed access to the resource by default unless they happen to be blacklisted. In a white list, the default is to deny access to the resource to all but the users enumerated in the white list. Black lists and white lists are used in many computer applications such as in network firewalls and control of access when surfing the Internet.

9.1.4 Integrity

The job of integrity of a computer system is ensuring that the set of applications and components installed in it are the ones which were originally intended in its design. Integrity may be compromised in various ways. On a server or workstation, unwanted software may be installed either by a malware like a virus or by a user in violation of standing policy. On a network, new devices may be added which should not be there. The techniques for managing integrity consist of detecting and preventing unauthorized devices or software in the system.

One way to ensure system integrity is to prevent the installation of any new software or devices in the system, except when done by a limited set of authorized people. Accordingly, some companies only allow software on their laptops to be installed by a set of administrators, with users having little ability to change the configuration or software on the devices. In some environments, such a measure may work. In other environments, such a restriction may be viewed as being draconian. In either of the two environments, one needs to continuously detect if system integrity has been compromised. Once detected, system integrity can be restored by removing the unwanted software.

There are three general techniques used for validating system integrity. One technique is to check system integrity against a set of predefined validation policies, the other is to detect anomalies in the system and the third is to check for the presence of unwanted elements.

The validation of system integrity against a set of predefined policies works in conjunction with a configuration management or discovery system. In most

environments, there will be a configuration management infrastructure to discover hardware device and software packages that are present in the computer system and obtaining the configuration of all discovered elements. Validation policies would indicate which configurations are allowed and which are not. Checking the configuration data, stored in the configuration management database, or retrieved on the fly from a device being validated can indicate the presence of new software or hardware, changes in the configuration of existing hardware and software, and any violations against policies can be flagged. As an example, configuration monitoring software can detect all wireless network identifiers (e.g., the service set identifier or SSID for 802.11 a.k.a. traditional home Wi-Fi networks) and associated wireless access points detected at any laptop and report them to a management server. The management server can then check if any unauthorized SSIDs or wireless access points are present in the system.

Anomaly detection checks for compromise of system integrity by assuming that regular systems have some common characteristic of operation. The common characteristic of operation may include an expected amount of traffic generated by the system, or the expected processor utilization on the system, an expected ratio between number of read requests and write requests in a system, an expected ratio of packets received using the TCP protocol versus the number of packets received using the UDP protocols, or a variety of other measurements that are defined in the specific context of how a system is being used. The anomalies are detected by means of software analysis packages that look at the monitored data. Any anomalous behavior detected needs to be further analyzed to see whether it represents a breach of any system's integrity. As an example, if one were normally expecting to see no packets in the network on network port 4374 (a randomly selected unused port), and one notices a machine sending several packets on that port, it may indicate a breach of system integrity. Further investigation needs to be done to check whether those packets were generated by a malware (definite breach of system integrity), or a user running a normal application software on a different port of the machine (which may or may not be a breach depending on the usage expected of that machine).

The final technique for validating system integrity can be viewed upon as an extension of the black-list scheme for authorization. In this scheme, a signature is computed for each unwanted component in the black list. The signature could be a hash computed over the binaries that represent the package, a characterization of the network traffic the software or device would generate, a summary of processes that the unwanted software would spawn, or some other attribute. A signature is any pattern or characteristics of the unwanted software component which can be observed to identify its existence. Monitoring software would compute the signatures for all software or hardware components that are available and compare those signatures against the black list. Several common antivirus software packages on personal computers operate on this principle. Signatures of common viruses are maintained in a master list and files on the

personal computers are compared against those signatures. Some software packages may compare calls made to operating systems to compute the signatures as opposed to scanning the files.

On servers or computers where the set of installed applications is limited to a few authorized ones, a signature-based white list scheme can be used to detect violations of system integrity. The signatures of all authorized software are maintained in a white list. Monitoring software can then compute the signatures of all software that is running or installed in the system and compare it against the white list. Any software that is not contained in the white list indicates a potential violation of computer integrity.

9.1.5 Non-Repudiation

In a general context, non-repudiation refers to the task of ensuring that a party in a dispute cannot deny an action that it has taken or done. In courts and legal disputes, non-repudiation is generally done by producing written documents containing the signature of the parties involved in the dispute. In the context of digital communications, non-repudiation involves ensuring that someone who sent a file (or other binary data) cannot deny that the file originated from the sender. This is usually done by means of a digital signature.

In the context of computer systems management, non-repudiation takes on another meaning. In any computer systems, there will be violations of integrity, access control, and policies that are put forth for the operation of the computer systems. Non-repudiation is the task of finding out the person or machine that may have done the violation.

The generic approach for non-repudiation in computer systems management is by maintaining proper logs in the systems that can track any actions or changes. The logs are used to create audit trails that record actions that are taken, the user who took that action, the time of the action, and any other relevant parameters of the action.

Audit logs need to be processed by automated tools which can weed out the logs in which nothing of interest happens. They can then be reviewed periodically by a human administrator as a safeguard. The more common practice, however, is to use audit trails to determine the root cause of a problem that may have happened in a system. As an example, if the configuration of a router is changed leading to some portions of the network becoming inaccessible, audit trails can identify the change which caused the problem and the user who may have caused the problem in the first place.

9.1.6 Availability

In the context of computer systems management, availability from a security perspective implies prevention against denial of service attacks. A denial of service attack is an attempt to deny access to computers or networks by

overwhelming them with what appears to be legitimate traffic. A prerequisite to maintain availability is to maintain system integrity. If a system is compromised with malware, the malware may prevent access to the computer system by overwhelming its resources or by changing authentication credentials for users.

One can attempt to detect denial of service attacks by trying to identify an abnormality in the patterns of incoming traffic that is different from the normal traffic. Although several algorithms have been proposed for such detection, the error rate is still relatively high for them to be used in practice. Therefore, most commercial installations would need to depend on additional capacity in order to ensure that the system is available in the presence of denial of service attacks.

The general approach to handling denial of service attacks is to ensure that there is sufficient capacity in the system to handle any potential denial of service attack. If there are sufficient machines and sufficient bandwidth in the network, then denial of service attacks, including all but the most widespread distributed denial of service attacks can be handled without impairing the performance seen by regular users. In many type of environments, where the number of users is limited, such sizing can be done to ensure availability. An example would be any servers or applications designed to run in an enterprise intranet. Since the enterprise intranet has a finite number of possible users, the servers can be provided sufficient capacity to prevent any denial of service attack even if significant portions of the clients were infected with malware launching such an attack.

Such a sizing may not always be possible in the case of servers or systems connected to the Internet, where a distributed denial of service attack can involve a much larger number of clients. Providing sufficient capacity may be too expensive in these cases. In those cases where the capacity is getting overwhelmed at one location, an alternate approach is to avail capacity that is dynamically available at other locations. Several service providers can offer capacity dynamically at new locations, and they can provide an adequate answer to a denial of service attack.

Taking these various general techniques into account, let us now examine the different aspects of security management that need to be enforced in the context of specific types of computer systems.

9.2 Security Management for Personal Computers

Personal computers include desktops, laptops, and netbooks that are used by individuals for their computing needs. Security management of personal computers can be viewed along the aspects of authentication, confidentiality, integrity, non-repudiation, and availability. In the contexts of personal computers, these aspects can be interpreted as following:

- *Authentication*: Ensuring that only the properly authorized persons are able to access the personal computers.
- *Confidentiality*: Ensuring that the data and applications on the personal computers are only available to the authorized people.
- *Integrity*: Ensuring that the personal computers are not compromised due to existed malware or viruses.
- *Non-Repudiation*: Keeping track of the changes made on the configuration and installation of the personal computers.
- *Availability*: Ensuring that the personal computers data and applications are available to the user in the presence of faults and failures.

The techniques for managing these various aspects will be similar to the general techniques we discussed earlier. However, in the context of personal computers, some specific issues need to be considered. Authentication on personal computers is managed by means of passwords, but these ought to be augmented with techniques that allow people to recover and reset forgotten passwords. Confidentiality requires proper management of data that are stored on the personal computers. Integrity translates to appropriate measures to protect against viruses and malware. Non-repudiation can be handled in most cases by maintaining audit trails as described in the previous section. Availability of information in personal computers needs to be dealt by having proper procedures for data backup and recovery. Another aspect of availability in personal computers, which also overlaps with aspect of integrity is maintaining the software on personal computers so that it is up to date with regards to security threats and reduces the probability of the personal computer being compromised and rendered unavailable.

These specific aspects related to personal computers security management are discussed in this section. These include the topics of data protection, malware protection, patch management, and data backup. Some of these issues will be of shared concerns with servers and server-based applications.

9.2.1 Data Protection

Personal computers such as laptops, notebooks, and PDAs can be lost with an alarming degree of ease due to their portability, and the fact that many people travel with them. Therefore, adequate steps must be taken to prevent such data from falling into wrong hands where they may be abused. For personal computers belonging to enterprises, the loss of data may compromise business plans, trade secrets, and intellectual property. Even for individuals, loss of personal computers may compromise sensitive information such as tax returns, social security numbers, and other personal information which may lead to identity theft and associated problems.

One common method to protect data on the personal computers would be to keep them on hard drives that are only accessible when a password for the disk

is typed. Unless the password is known, the disk is not usable, and the only recourse for someone with the disk is to reformat the entire disk. In that case, the data on the disk are destroyed and no longer accessible to the other users. The danger of course is that if the disk password is forgotten, the information on the disk becomes impossible to retrieve.

One possible approach to deal with the problem of lost passwords is to store passwords in a repository from where they can be accessed when forgotten. However, that is likely to create more security vulnerabilities since such a store becomes a very attractive target to break. A better option would be to use data backup and recovery mechanisms to periodically take a copy of the data into a more stable backup system. The backed up information can then be restored on a new disk (or the reformatted version of an existing disk) which is used to replace the original old disk.

Disk password protection or disk content encryption provides for storing secure information at a disk-wide level. Encryption of information can also be done at other levels, e.g., at the file system level by having an encrypted file system, or even at the level of individual files by protecting the contents of an individual file. The level at which the information is chosen to be encrypted provides constraints on who can access the system and how much support can be provided to recover lost passwords. With encrypted file systems, the encryption keys are stored at the level of the operating system, and these keys can be stored in an enterprise system from whether they are retrieved automatically – thus enabling more robust protection against lost passwords. File level encryption would require programs that are enabled to use protected files and are most useful when exchanging information through files. The passwords required to decrypt the encrypted files can be exchanged through phone calls or other mechanisms.

Another mechanism that can be used to protect personal computers and other devices are Internet-based remote destruct buttons. When a personal computer or device such as a laptop, PDA, or cell phone is lost but is able to connect to the Internet, it can send a signal which activates software that destroys all data on the machine. The software is always running on the personal computer, and users can activate the destruct button by reporting that their devices are lost to a registration server. The prerequisite for this operation is that the device be able to access the Internet when it is booted up after being lost or stolen.

9.2.2 Malware Protection

Malware on person computers takes the forms of many different viruses, Trojan horses, and other types of malicious programs that may be installed on one's computer system inadvertently. Malware protection requires a variety of techniques to be implemented on machines. These include the installation of

antivirus software, and also installing policies regarding the access of different applications that may need to access the external network from the computer system.

An antivirus software is a program that checks for the existence of undesirable software on the personal computer. An antivirus software may use techniques such as signature analysis or abnormality detection to identify that a virus may be present on the system. Generally, the antivirus software would need to update the signatures available for checking the existence of viruses at a regular frequency. Therefore, the configuration of the personal computer needs to allow such updates to happen.

Many personal computers also run software which acts like a personal firewall for the machine. The personal firewall regulates which applications on the computer can access the external world, and which external applications can connect to the applications running on the personal computer. As a general rule, no application running on the personal computer should be allowed to accept connections from the outside world, most applications on the personal computer acts as clients and establish connections to servers running on the outside. However, some legacy applications (such as ftp) require that the personal computer accept a connection from the outside world in order to work properly. In some cases, applications may need to be run to accept connections on personal computers as well. In order to enable the personal computer to perform its functions, the policies on what is permitted and what is not permitted to connect to the outside world needs to be established.

The system administrator in an enterprise where many personal computers need to operate smoothly needs to determine a way to figure out setting the right policies on each machine. Generally, users need some flexibility in modifying the policies while the administrator needs to set a default policy which is required to ease the task of any desired applications running on the personal computer. Most personal firewall software packages allow the enterprise policies to be downloaded from a central enterprise location (which is configured as part of the default configuration of the machine in the enterprise). The personal firewall can upload and refresh the policies as they change from the central locations. The enterprise policies can be augmented by additional policies that the user may want to specify for allowing access to some applications.

9.2.3 Patch Management

An important aspect of managing availability of personal computers is managing the software available on those systems. The task of managing the software configuration on personal computers is patch management. Patch management for the general purpose of configuration management has been discussed in Section 7.5.

In addition to being a component of configuration management on personal computers, patch management plays an important part in managing the security of personal computers. Many security vulnerabilities arise due to exploitation of known bugs in the operating systems or common applications running on a personal computer. Thus, maintaining an up-to-date patch for personal computers is an essential aspect of maintaining the integrity of personal computers.

Patch management is also important for the security of servers and applications running on servers.

9.2.4 Data Backup and Recovery

Personal computers tend to be relatively fragile. Data and applications stored on personal computers may be lost due to loss or theft of the device, or due to a failure or the device. Sometimes, information stored on devices may be deleted accidentally. In order to ensure that such a loss of data is not irreversible, backup and recovery solutions need to be implemented to ensure the availability of information in every computing environment.

The data backup and recovery system usually consists of an application that runs periodically on the personal computer. It tracks any files that have been modified since the last backup and then stores a copy of those files on a backup server. In order to preserve space on the backup server, backups can be done in one of two modes, a full backup or an incremental backup. The full backup keeps a snapshot of all the data stored on the computer and stores it in the backup server in a compressed format. An incremental backup only stores changes from the file as created in the previous complete backups. A system administrator can determine the frequency of backups, and the ratios of incremental backup for each full backup in most versions of the software.

It is quite common to store metadata associated with a backup at the server. The metadata may contain information such as the time a file was backed up, whether it was an incremental backup or a full backup, and if an incremental backup the reference to the full backup from which changes to the file are maintained. During each backup session, the backup application would contact the server, find out the changes from the file since its last backup, and send the modified files to the storage server if desired.

The recovery application allows the user to restore any data from one of the previous backups. In the case of accidental loss of data, the recovery application can create the lost version from the backup data. In the case of a misplaced machine or a failed disk, the contents of the old disk can be recreated on a new machine or a disk from the backup server.

In most environments multiple personal computers are backed up at a single backup server. For scalability reasons, the backup server may consist of a cluster of backup machines rather than a single machine. Since the backup

server has to handle storage needs of a large number of clients, it is usually implemented as a cluster of high-end servers running with a large amount of storage space, obtained from a mixture of tape drives, storage area network devices, and using network-attached storage.

Large enterprises may have several backup sites, with each backup site supporting a subset of client machines. The partitioning of client machines among different backup sites is determined primarily by the proximity of a client to the backup site. In many cases, the backup sites act as peers to each other and providing mirroring capability to provide disaster recovery support for storage backup functions.

Maintaining the data backup and restoration service is an essential part of systems management and operations in any enterprise. It is crucial for maintaining availability in the presence of failures and loss of machines.

Data backup also needs to be done for computer servers and applications that are installed on servers. Furthermore, such servers need to have mechanisms that can recreate the applications installed on the servers in case of a failure, not just the data.

9.3 Security Management for Computer Servers

Computer servers are machines that frequently run unattended and host applications such as web sites, e-mail exchange servers, printing services, and other applications. As with other systems, we will view security management of servers and applications running on the servers along the aspects of authentication, confidentiality, integrity, non-repudiation, and availability. In the contexts of servers and server-based applications, these aspects can be interpreted as following:

- *Authentication*: Ensuring that only the properly authorized persons are able to access the servers. An additional complexity arises in case of servers due to the large number of passwords required by different applications.
- *Confidentiality*: Ensuring that the data and applications on the servers and applications are only available to the authorized people.
- *Integrity*: Ensuring that the servers are not compromised or attacked.
- *Non-Repudiation*: Keeping track of the changes made on the configuration and installation of the servers.
- *Availability*: Ensuring that the server and its applications available to the user in the presence of faults and failures.

Some of the issues in security management of servers are similar to the security management for personal computers. They are both end points of a communication and run a wide variety of applications. However, since servers and server-based applications need to support many different clients, some additional concerns need to be taken into account for servers.

Authentication on servers is managed by means of passwords, but these ought to be augmented with techniques that allow people to recover and reset forgotten passwords. Techniques to manage and simplify passwords including single sign-on systems need to be implemented in any environment where there are more than one servers or applications. Confidentiality requires proper management of data that is stored on the servers. Furthermore, since many clients need to access a server application, secure access protocols need to be supported on the servers. Integrity translates to appropriate measures to protect against inappropriate configuration updates on servers. Non-repudiation can be handled in most cases by maintaining audit trails as described in the previous section. Availability of servers needs to be dealt by protecting it against denial of service attacks as discussed in Section 9.1.

These specific aspects related to personal computers' security management are discussed in this section. These include the topics of password management, single sign-on, and secure access protocols that need to be supported on the different server applications.

9.3.1 Password Management

In a typical enterprise, there are many servers and many applications running on those servers. Most employees have a need to access many of those servers and applications. However, passwords may be forgotten readily because there are so many of them. Even if there is a single password required to access all the server applications, an employee may forget that password too. In almost all large computing environments, provisions need to be made for people forgetting their passwords and needed to retrieve them or more commonly reset them.

The management of passwords requires the establishment of a system that can reset the passwords of users. The reset password typically has a short life span, requiring the user to reset the password to a new one after logging in the first time using the first password. The password for an application can frequently be reset in a self-service mode by users accessing a web-based application. In these cases, the web-based application would authenticate the user by asking it for some other information, e.g., its employee identifier number and last name, or any other combination of information that is likely to be known only to the user. The reset password can then be sent back to the user via e-mail or as a voice prompt on the user's phone mail, or any other way that the user may access it conveniently. It is common practice not to provide the e-mail directly on the web interface, but using some other means which is likely to be accessible only to the user.

In server-based applications, a common danger is that of a hacker getting access to the password of the users. A common mode of the attack is that of a hacker trying to run through various combinations of a password. In order to encounter these and similar attacks, different types of password policies are

frequently enforced. Examples of these policies include instructions on minimum length of the password, the mandatory use of numbers or non-alphabet symbols in the password, and the maximum lifetime on the validity of a password. These policies help in reducing the risk of a password being compromised.

When establishing the policies that are to be used for managing passwords, caution must be taken to achieve an appropriate trade-off between usability and security. If passwords are made too complex, then users would have problems remembering them and are more likely to write them down, resulting in a larger risk of security compromises. If passwords are made to change too frequently, people have a tendency to forget the passwords, and thus are more likely to call the helpdesk to reset the password, increasing the cost of password management and lost productivity of the users.

One of the challenges in many enterprises is the sheer number of applications and servers whose access needs to be password controlled. In order to alleviate the problems of an exploding numbers of passwords, techniques that can allow the use of a single password to access many applications are needed. The techniques, known collectively as single sign-on techniques, are described in the next section.

9.3.2 Single Sign-On

The management of single sign-on [9] infrastructure is one of the most important aspects of managing security in any enterprise. It is one of the techniques that can reduce the amount of effort and time required to access enterprise systems. A well-implemented single sign-on system can reduce the number of calls placed to helpdesk to manage forgotten or lost passwords.

A single sign-on system allows users to access multiple applications on servers using a single password. Each of these applications may require different types of passwords to access them. The following are the three general approaches that are used to implement single sign-on systems.

Authentication Server: The user authenticates himself to a central authentication server. The authentication server provides a set of credentials or ticket to the user. The user presents this ticket to the different applications which validate the ticket. Different types of tickets can be provided to access different types of applications.

Authentication Proxy: The user accesses the applications through an authentication proxy. The proxy has a safe store that contains the passwords and potentially different user IDs required to access different applications. Once the proxy has authenticated the user, it can retrieve the right combination of user ID and password from the safe store and present it to the target application. The safe store may be an encrypted file stored at the proxy, or it may be a smart card that the user can provide to a proxy running on the local machine.

Shared Credentials: Applications are written so that the user uses a single authentication system. When any user accesses an application, the application validates the credential using this common authentication system. A typical implementation of the common authentication system may store all credentials in a directory server accessed using the LDAP protocol.

The systems administrator needs to select one of these methods to implement the single sign-on in the enterprise. The shared credential methods can only be used with applications that are developed in-house or which can be used with an in-house authentication proxy. The authentication proxy can be used with applications that use the same protocol – which is convenient in current times when many applications use the HTTP protocol as the front-end interface for the users. The authentication server approach is supported by protocols such as Kerberos [10].

The use of single sign-on system needs to be complemented by a well-implement password management system.

9.3.3 Secure Access Protocols

When managing servers and applications, the servers need to be accessed securely by management applications. Similarly, in order to authenticate clients, secure access protocols needs to be implemented to safeguard the communications between the different clients and the server.

Applications running on a server can be accessed using different protocols, some of which are secure and others are not secure. A secure protocol would encrypt the information that is flowing between the communicating parties. There are a variety of secure access protocols available for server-based applications, and the systems administrator needs to select which protocol to use for a specific application.

Modern encryption protocols use a combination of public key cryptography and private key cryptography. Encryption and decryption operations performed by public key algorithms are slower than the equivalent operations using private key algorithms. However, the task of establishing the keys required to communicate with an unknown person is easier using the public key cryptography schemes. One can communicate with a named party as long as the public key of the person is known, while the other option would require the knowledge of a mutually shared secret key. As a result, modern encryption protocols are designed in two phases. In the first phase, public key cryptography is used to establish a secret channel of communication. A client may use the public key of the server or server application to initiate the establishment. Over this channel, the client and server negotiate a secret key they will use for data communication. Then, in the second phase of the communication, the secret key using private key algorithms is used to encrypt and decrypt the messages. In

the first phase, the duration over which the secret key will be used is also negotiated, either in terms of a bound number of exchanged bytes or a bound of the maximum life of the secret key. After the time is exhausted, the first phase communication renegotiates a second secret key.

In order for the process to work efficiently, the system administrator must configure the right public key certificates for the protocol, and configure the appropriate policies for duration of the shared secrets used in the communication.

The three common security access protocols used by server-based applications are SSL (secure sockets layer) [11], IPsec (Internet Protocol Security) [12], and SSH (secure shell) [13]. Each of these protocols has its pros and cons.

IPsec is a protocol that can encrypt (or just authenticate) any machine at the network (IP) layer of TCP/IP protocol suite. Thus, IPsec can be used to secure the communication between any two machines running the IP protocol. SSL is a protocol implemented at the application layer, and can be used to authenticate applications that are explicitly written to run on this protocol. IPsec usually has difficulties traversing boundaries where network address translators are used – since it authenticates packets at the network address. SSL – the secure sockets layer (or more properly TLS – transport layer security), on the other hand, encrypts at the application level and thus can easily traverse address translator and other network security devices. However, since the network packets are not secure, an interceptor can extract more information about the communication, e.g., the application using the SSL protocol by examining the transport and network headers.

One common use of secure access protocols is to remotely access corporate applications and networks in a secure manner. If the applications being accessed are going to be accessed through a web-browser, then the SSL-based access has many advantages. It can traverse firewalls, does not require a client software on the remote machine used to access the server applications, and is not impacted by network address translator. However, if general class of applications have to be supported, then IPsec-based access may be more advantageous.

SSH or secure shell is an application that allows a user to remotely open a command line prompt on the server, and invoke user commands. It provides a convenient method to access for system administration tasks.

9.4 Security Management for Computer Networks

If we consider the same five aspects of authentication, confidentiality, integrity, non-repudiation, and availability in the contexts of a computer communication network, we can interpret them as follows:

- *Authentication*: Ensuring that only the properly authorized devices are able to connect to the network.
- *Confidentiality*: Ensuring that the data on the network is encrypted.

- *Integrity*: Ensuring that the network is not compromised by presence of packets interceptors or other malicious devices.
- *Non-Repudiation*: Keeping track of the changes made on the information sent out on the network.
- *Availability*: Ensuring that the network connectivity is maintained in the presence of attackers.

The techniques for managing these various aspects have a lot of commonality with the techniques already discussed above. However, in the context of computer networks, some specific issues need to be considered. Authentication on personal computers is managed by means of protocols that only allow authorized devices to connect to the network. Confidentiality requires the user of secure access protocols that we have already discussed. Integrity translates to appropriate measures to validate network configuration and detecting intruders on the network. Non-repudiation can be handled in most cases by maintaining audit trails as described in the previous section. Availability of network in the presence of attackers requires techniques to limit the bandwidth that may be used by malicious applications on the network.

The techniques used for enforcing these aspects in computer networks that have not already been discussed include the user of firewalls, intrusion detection/prevention systems, and honeypots.

9.4.1 Firewalls

A firewall [14] is a network device that sits on the boundary between two networks with a different level of trusts and controls the traffic flowing between these two networks. A common use of the firewall is to protect the boundary between the public Internet and a company's intranet.

Firewalls may implement a variety of techniques to inspect and filter out traffic that is not desirable. Packet filtering firewalls look at the network and transport headers of the inbound and outbound packets and use those fields to decide which packets to let through and which ones to filter out. Stateful packet filters track the connection that packets belong to and filter out those connections. Application level gateways are proxies for applications that can filter out the content of specific types of applications. Applications that are commonly used have application level gateways specially created for those applications. Examples include filters for e-mail, web proxies that filter out dangerous attachments and VoIP security gateways.

Firewalls form an important line of defense against the most common types of attacks that an external attacker may launch on the network. They can be used to close off access to any but selected applications running on designated ports. Application level gateways can scan the traffic to remove malicious traffic. Thus, firewalls provide the basic level of defense in more networked environments. Firewalls are used in many locations, e.g., between a home

network and the network services provider, between enterprise customers' campus networks and their network services providers, or between different branches of an enterprise network.

Firewalls can be used to create different security zones inside a network. The rules at firewalls prevent users in one zone from accessing the computers in other zones. As an example, Fig. 9.5 shows how firewalls can be used to separate out different customers in a shared hosting data center. The firewall in front of each customer network prevents traffic from other customer networks, while permitting traffic from the management zone and the Internet to flow through. The access restrictions on those firewalls are shown in the figure.

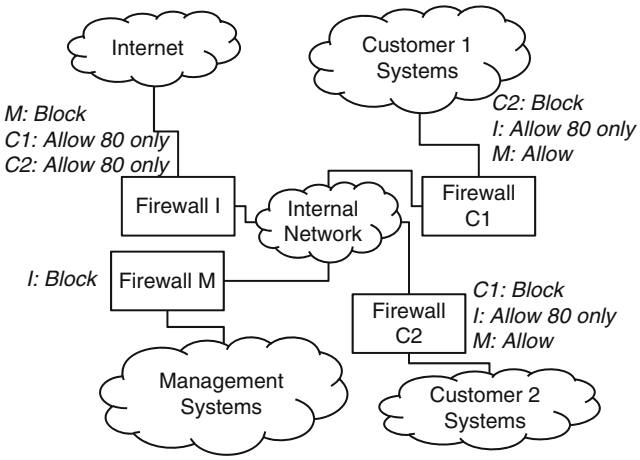


Fig. 9.5 Firewalls in shared hosting data center

The firewall I which is facing the Internet blocks all traffic to the management network. It only allows traffic on port 80 to customer 1 and 2 network. The firewall facing the management system blocks all traffic from the Internet, but allows traffic from the customer segments. The customer facing firewalls block all but port 80 traffic from the Internet and block traffic from each other. These firewalls taken together implement the isolation zone of the shared hosting data center.

The system administrator needs to be aware of the location of the firewalls in the network, ensure the existence of a firewall everyplace networks with different trustworthiness come into contact and configure them correctly to ensure that the security and trust requirements are properly satisfied.

9.4.2 Intrusion Detection/Prevention Systems

An intrusion detection system (IDS) [15] is a device which inspects the packets entering a network to determine if there are attackers trying to attack a system. The intrusion detection system can be installed by an enterprise to detect

attacks on its intranet, by the operator of a data center or a data center to detect attacks on its servers, or by a networking services provider to observe any attacks on its customers.

An intrusion prevention system (IPS) is an intrusion detection system which is also capable of filtering or dropping some types of traffic. The intrusion prevention system detects whether there is suspected malicious traffic within the network, and if so, drops the packets that are suspected as being malicious.

Both IDS and IPS systems detect intrusions by using a combination or signature analysis or abnormality detection. In the case of signature analysis, known types of network attacks are characterized. A network attack may be characterized by means of packets seen on a specific port, by means of a sequence of packets showing up on the port, or by the structure of the contents of some of the payload that is used to launch the attacks on the network. A network intrusion is frequently an attempt to gain access to the computers by an outsider. The intrusion attack would typically involve an attempt to scan the network to identify any available servers or computers, and then try to probe the available computer for any potential vulnerabilities. When a vulnerable computer is identified, the attacker may try to gain access to the computer by launching specific types of attacks, e.g., trying to execute a program which may overrun a buffer and obtain access to the privilege mode of the computer.

In each of the stages of these attacks, the outsider trying to attack the system will follow known patterns of attacks. One way to scan the network would be done by sending a connection request to all possible addresses that may be in the adjacent space (perhaps with some randomization). Thus, if too many connection requests to different local addresses are obtained as originating from the same remote address, one may identify an attempt to scan the network. The probing of vulnerabilities similarly tries to establish a connection to potentially vulnerable server applications listening for outside requests. The sequence of such probes can be used as a signature for detecting an attack in this stage. Packets that actually exploit the vulnerability can be identified from their payload since they have to be crafted precisely to take over the system. Thus, attack patterns that are known to have been used previously can be characterized, and their signature patterns can be searched for in the network traffic passing through an IDS/IPS device.

It is not uncommon to have intrusion attacks be launched by script kiddies – people who pick up programs that are available on the Internet to launch attacks on computer systems, and try them out. The person running the program himself/herself need not be very sophisticated. Signature-based intrusion detection systems can be highly effective against such script kiddies.

On the other hand, sophisticated attackers would tend to develop new methods for scanning the network, probing for vulnerabilities and exploiting the vulnerabilities than are known to the security experts. Such attacks cannot be detected with signatures, and abnormality detection is used to develop methods to identify and detect new types of attacks. In this approach, a baseline is taken of traffic that is normally operational on the network. The baseline is

assumed to be free of any intrusion attempt. Ongoing traffic is compared to the baseline to identify any unusual traffic patterns, e.g., too many attempts to connect to a server on a strange port number. These unusual attempts are then examined for potential intrusions. An IPS system may block traffic from machines that are the source of the abnormal traffic.

The main issue in IDS systems using either of the two approaches is that of false positives. Because of the statistical nature of the traffic and the high volumes that are involved, detecting signatures of known attacks or detecting abnormalities in traffic end up flagging potential intrusions even when the traffic originating is due to normal users. In many cases, the reported intrusions need to be examined manually, which adds delays. The uncertainty also results in legitimate traffic being blocked by IPS systems, which is highly undesirable in most circumstances. One approach to address the problems of false positives is the development of the honeypot concept, which is described in the next section.

9.4.3 Honeypots

A honeypot [16] is a machine with some known vulnerability that is put out deliberately on the network in order to attract an attacker. The honeypot is used to observe the type of attack that is launched by the intruder, which can be used to develop signatures for preventing the attack or learning about the unknown vulnerabilities in a system which an attacker may be trying to exploit.

Honeypots may be placed using Internet addresses which are not likely to be used by normal users. As an example, an address close to that of a legitimate server, but not registered with the domain name service may be used for a honeypot. The only user likely to access such as machine is an attacker scanning for available machines.

Honeypots can be created to provide various levels of deception for the attacker. The honeypot software needs to be authentic enough to give a false sense of security to the attacker, but still leave enough controls so that the honeypot can be turned off in case an attacker is able to compromise the system and use it to launch more attacks.

Honeypots do not suffer from the problem of false positives that is common in IDS/IPS systems and are valuable tools for the system administrator to monitor any suspicious activities. A honeypot is more likely to catch an insider [17] trying to attack the network than other approaches used for protecting the network.

9.5 Operational Issues

In addition to the techniques for managing the security of the computing infrastructure, there are several operational issues that a system administrator needs to handle in order to properly provide a secure environment. These

operational aspects include provisions for physical security, setting and enforcing security policies, regulatory compliance, and auditing. These aspects are discussed in this section. A detailed discussion of the operational issues and best practices for security management can be found in [18].

9.5.1 Physical Security

Physical security [19] is an important aspect of managing the overall security of any computer system. If an attacker can get access to a computer room, they can easily disrupt the operation of the system, steal information physically from printed material, or access unauthorized systems. At the very least, an intruder with physical access to the system can destroy or shut down the system disrupting system availability.

All computer servers and networking equipment need to be operated in a secure location. The location can be physically secured by means of a lock and key, or for shared access, secured using badges or other types of access control mechanisms. For more sensitive and critical parts of the computer system, more sophisticated access control methods, e.g., biometric access control can be used. These methods would use fingerprints or eye-iris scans to identify a person and allow access to the secure area.

Physical security includes techniques that can prevent access to a computer system by means of physical proximity, including the use of smart cards or biometric devices to access computers, and shielding a computer device against hackers who try to read electromagnetic radiation to break security keys. Another important part of physical security is providing adequate protection against infrastructure outages. These include proper care to prevent physical damage, e.g., due to some object falling on the computer servers, as well as protecting against electric surges and outages. Surge protectors and backup power supplies are an important component of a secure operation of the system.

A weak link in physical security is the disposal of old computer equipment. One can occasionally come across news articles where sensitive data are found on computers sold on the Internet or in second-hand shops. In order to prevent the loss of sensitive information from old equipment, all information must be removed from old devices. The disk drives ought to be reformatted to ensure that none of the old information remains inadvertently on the old equipment. Similarly, any equipment that is being transported needs to be tracked and data on it ought to be protected, e.g., using passwords or encryption. This prevents data from inadvertently falling into wrong hands if the package is misdelivered.

9.5.2 Security Policies

Proper security management often requires the specification of security policies [20]. A security policy is a high-level directive or guidance provided to the

system management staff or the IT department which provides high-level guidance on the operation of the system. The security policies of any organization may typically be provided in a document expressed in English or another natural language. Quite frequently, these policies may be specified so that they comply with the regulations or laws of the land. Some laws like HIPAA (Health Information Portability and Accounting Act) in the United States require hospitals and physicians to provide strict controls on who may access the information stored in different computer systems. Other businesses have their own regulations and laws which would impact the security of different systems. In almost any type of commercial business, some information needs to be restricted, e.g., credit card numbers of customers, or salaries of employees. Access to systems that contain such restricted information needs to be restricted as well.

Other types of security policies may deal with issues related to installation and configuration of the computer systems and networks. A policy may require that certain type of security software be installed on all personal computers, or that some types of intrusion detection systems be installed on the network, or it may require different parts of computer networks and systems to be logically segregated from each other. Security policies may also define roles and access capabilities associated with each role. Other policies specify constraints on passwords, e.g., passwords cannot be a dictionary word, must contain a non-numeric character, or have a minimum length.

Security policies may be used in various ways in the context of any particular installation of a computer system. In the most basic use-case scenario, the security policies may act as a rule book for the system administrator. Before making any changes in the configuration of any system, the administrator needs to check the rule book to validate manually that no policy is being violated. In such case of manual validation, the system administrator simply needs to read the specified policy and determine whether he/she is in compliance.

In some cases, the policies may be supported by configuration management systems which can flag out if any configuration change by an administrator can possibly violate one of the security policies. In order to make such an automated check, the policies need to be translated from natural language to some structured format which can be interpreted by a software system. The software can then check for potential violation of policies expressed in that manner.

In case a violation of policies may trigger other policies specified either in human-readable or machine-readable format. These policies may deal with aspects such as the process for reporting violations, the rules for escalating information about such violations through the IT management, or the corrective actions to be taken when specific types of violations may have occurred.

Another way to use policies represented in a computer-readable manner is to check for potential violation of security policies in the configuration information of a computer system or network. The process of configuration validation is similar to that described in (Section 7.4.1).

Another use-case for security policies is the deployment of a policy-based management system. In a policy-based management system, policies are

specified at a high level and then translated through a series of steps into the configuration of the computer system which is in conformance with the specified security policies. Using such tools, the configuration of any system would always be compliant with the specified policies.

9.5.3 Auditing

An audit is an important aspect of maintaining security in computer systems. Audits generally look at a snapshot of the computer system and validate whether the system is conforming to the security policies that may have been specified for the system. An audit can also take a physical inventory of the systems that are present in the environment, and validate that against the set of systems that are expected to be present.

Audits are frequently performed as a set of security questionnaires that need to be filled out and validated. The auditor may be an external person, or someone internally from the system management organization. The auditor would check on whether proper procedures are being followed, and examine the configurations of different systems and the network to ensure that the proper policies are being enforced.

In some cases, audits can be performed in an automated manner. As an example, security policies related to installation of mandatory software (e.g., antivirus) or strong passwords can be audited automatically. Such automated audits take the form of software that is installed on computer systems and run periodically. Automated audits can also be performed on the configuration of networked devices. The process and approach for automated audits is identical to the configuration validation approach described in Section 7.4.1.

A different way to perform an audit for security of systems is to hire white hat hackers. The white hat hackers try to compromise the security of the system without the knowledge of the system administrator, using a variety of attacks, technical as well as non-technical. The success of the white hat hacker in compromising the security of the system would identify potential vulnerabilities in the system, while the failure of the hacker to intrude into the system provides a better assurance that the system security is being managed properly.

Audits are a useful tool in managing the security of any computer system in any context, and provide a good insight into the quality of security management that exists in the environment.

9.6 Summary

In this chapter, we have looked at the different aspects of security management in computer systems. Computer systems security can be analyzed along the five aspects of authentication, confidentiality, integrity, non-repudiation, and

availability. The applications of these aspects have slightly different contexts in personal computers, servers, and networks, but they all rely on a common set of general techniques. The common techniques include cryptography, authentication techniques, access control mechanisms, anomaly detection, and signature analysis. Security management for personal computers includes topics such as data protection using encryption, protection against malware, patch management, and data backup techniques. Security management of servers requires additional aspects of password management, single sign-on systems, and secure access mechanisms. In computer networks, additional security issues include the use of firewalls, intrusion detection systems, and honeypots. In addition to the technology for managing security, one needs to consider physical aspects such as enforcement of security policies and audits.

9.7 Review Questions

1. Discuss the similarities and differences in the general approaches used for security management and systems management.
2. Almost all of the techniques presented for protecting personal computers are also application to protecting computer servers and server-based applications. What are the differences in the design of such systems that will come into affect when you consider servers instead of personal computers?
3. In Chapter 1, we had discussed three computing environments, a data center, an enterprise, and a communicatons network provider. What are the different security challenges that each of these environments may face?
4. Find out the security policies that are in effect for the computing systems and networks in your university. Determine which subset of those security policies can be automated using software assets and which will be hard to automate.
5. Honeypots provide an attractive way to learn about attacks launched on the computers, but also subject the enterprise to many risks. What are some of the risks a data center deploying honeypots may run into?
6. There is a trade-off between the usability of computer systems and the security that can be provided to users. Discuss the differences between the system usability and risks between two enterprises, one of whom requires all employees to change passwords every 3 months and the other requires all employees to change passwords every 6 months.
7. What are the two basic types of techniques used for detecting viruses in personal computers? Discuss the similarities and differences between the two basic types of techniques used for intrusion detection in networks.
8. What are the different ways to use security policies in the management and administration of a computer system?

References

1. W. Maconachy, C. Schou, D. Ragsdale, and D. Welch, A model for information assurance: *An integrated approach*, Proceedings IEEE Workshop on Information Assurance and Security, West Point, NY, June 2001.
2. W. Stallings, Cryptography and Network Security, Prentice Hall, Saddle River, NJ, 2005.
3. B. Schneier, Applied Cryptography, Wiley, New York, 1996.
4. J. Hoffstein, J. Pipher, and J. Silverman, An Introduction to Mathematical Cryptography, Springer Verlag, New York, 2008.
5. Data Encryption Standard. Federal Information Processing Standards Publication 46. National Bureau of Standards, U.S. Department of Commerce, 1977
6. C. Adams and S. Lloyd, Understanding PKI: Concepts, Standards and Deployment Considerations, Addision Wesley, Boston, 2002.
7. P. Zimmerman, The Official PGP User's Guide, MIT Press, Cambridge, MA, 1995.
8. L. Lamport, Password Authentication with insecure communication, Communications of the ACM, 24(11): 770–772, November 1981.
9. A. Pashalidis and C. Mitchell, A Taxonomy of Single Sign-on Systems, Lecture notes in Computer Science, Springer Verlaf, volume 2727, January, 2003.
10. J. Steiner, C. Neuman, and J. Schiller. Kerberos: An authentication service for open network systems. In: USENIX Winter Conference, February 1988.
11. S. Thomas, SSL & TLS Essentials, Wiley, New York, February 2000.
12. N. Doraswamy and D. Harkins, IPSec, Prentice Hall, Saddle River, NJ, 2003.
13. D. Barrett, R. Silverman, and R. Byrnes, SSH, the secure shell, o'reilly media, California, CA, May 2005.
14. W. Cheswick, S. Bellovin, and A. Rubin, Firewalls and Internet Security, Addison-Wesley, Boston, 2003.
15. R. Barnard, Intrusion Detection Systems, Butterworth-Heinemann, Boston, 1988.
16. L. Spitzner, Honeypots: Tracking Hacker, Addison Wesley, Saddle River, NJ 2002.
17. L. Spitzner, Honeypots: Catching the Insider threat, Computer Security Applications Conference, Las Vegas, December 2003.
18. M. Swanson and B. Guttman, Generally accepted principles and practices for securing information technology systems, NIST Publication 800-14, September 1996.
19. S. Weingart, Physical security devices for computer subsystems: A survey of attacks and defenses, Lecture Notes in Computer Science Volume 1965, Springer, January 2000.
20. M. Hughes and R. Stanton, Winning security policy acceptance, Computer Fraud & Security, 2006(5): 17–19, May 2006.

Chapter 10
Advanced Topics

There are several subjects that are related to systems management which a student of the subject ought to be aware of, because they are closely related, or advanced subjects in systems management. In this chapter, we provide a brief review of those subjects.

10.1 Process Management

While we had discussed the technology and systems required for different aspects of systems management, an important aspect of system management also relates to the processes and procedures that are followed in order to have computer systems operate without any problems. A process is defined as the set of procedures that are followed in order to perform any task related to IT systems.

As an example, let us consider the procedure for obtaining and installing a server in a data center. The request needs to be obtained with proper authorization, and an IT architect needs to determine the set of software that needs to be installed on the new server. The network administrators need to be contacted to create the right network connectivity for the server, a system administrator needs to install set of software determined by the architect, test the software for correct operation, and then make the server live. These steps involve different people and organizations within a company. Not following the steps of the procedures may result in problems when a new system is configured. However, because most of the steps in the process involve human interaction and operation, one cannot easily automate the processes using software tools.

Process management deals with the task of defining the processes used in system management (or in general any IT processes), and putting them into a manner so that they can be used in a uniform manner across an enterprise. They can be viewed as a collection of a set of best practices that should be used for performing the different tasks associated with systems management. A collection of such best practices is referred to as IT Service Management or ITSM [1] .

D.C. Verma, *Principles of Computer Systems and Network Management*,
DOI 10.1007/978-0-387-89009-8_10, © Springer Science+Business Media, LLC 2009

The most common versions of ITSM are based around IT Infrastructure Library (ITIL). ITIL [2] provides a systematic approach identifying and describing processes related to the provisioning and management of IT services, from inception through design, implementation, operation, and continual improvement. The set of ITIL version 3 specifications is divided into five subsets dealing with service strategy, service design, service transition, service operation, and continual service improvement.

Service strategy deals with processes needed for identifying new IT services including their technical and financial aspects. Service design deals with processes related to the tasks of availability management, capacity management, security management, etc., that will be needed during the design stage of a computer system's life cycle. Service transition defines the processes related to changes in the services, such as change management, managing different releases of the system, and configuration management. Service operation describes processes required for managing problems and requests that arise during the operation stage of the life cycle. Continual service improvement deals with processes involved in monitoring and managing quality of service of IT services.

ITIL and ITSM usually tend to specify processes at a rather high level in order for them to be applicable across a large set of usage conditions. These specifications need to be customized and refined into a more detailed set of process specifications which are specific to the operation of a company's system management environment. Thus, the ITIL specifications provide the starting point of process management, rather than a complete specification for a set of detailed processes.

Establishing a set of processes which can be used for various aspects of systems management is a useful practice that can be very useful in those aspects of systems management which require coordination of actions of multiple administrators or coordination across the different departments of an organization.

10.2 Helpdesk Systems

Each computer system has one or more users, and even in the best-run computer systems, the users will end up with some problems related to system operation. A help desk [3,4] is a mechanism to provide help to the users who encounter such problems.

A help desk can be divided into four major components: (i) an interface through which the users can reach a helpdesk operator; (ii) a ticket management database which keeps track of problems encountered by the helpdesk staff; (iii) a knowledge base which is used to store information that the helpdesk staff can use to address and resolve the problems that are encountered; and (iv) analysis tools which can look at the history of the help desk and update/modify the knowledge base.

In traditional help desks, the primary interface for users to reach the helpdesk operator was through a phone line. Currently, several other interfaces are also commonly used: users can reach the helpdesk operator through e-mail, a user can report their problems using a web-based interface, and in many cases the user can establish an instant messaging session with one of the helpdesk staff to communicate regarding their problem. Phone and instant messaging allows the user to interact in real time with helpdesk staff, while e-mail/web-based interface problems are addressed by helpdesk operators in a non-interactive manner. In a web-based interface, it is common for the help desk to provide much of the information to the user so that the user can resolve the problem on his/her own without needing to contact the helpdesk staff.

Interfaces that are based purely on web or e-mail-based interactions are cheaper for the company operating the service, but typically have a large lag in responses from the help desk. Phones and instant messaging interactions have a more interactive dialog between the users and the helpdesk staff. Another aspect deals with the cost of the help desk. Phone-based helpdesk access tends to be more expensive than the other mechanisms, and several companies tend to outsource those operations to some part of the globe where it can be done relatively cheaply.

Each request for user help that is received at a help desk is usually logged into a ticket system. A ticket is a record of a complaint received at the help desk, and records information such as the problem encountered, the conditions under which the problem was encountered, and the attempts made to resolve it with the resulting outcome. If the problem cannot be resolved immediately, the ticket acts as the store which records the status of the problem, and the attempts to resolve it at subsequent times.

A help desk is typically organized around levels, with each level involving people who are more knowledgeable about the computer system to which the problem may resort to. The initial contact with the help desk is handled by representatives who would try some routine steps to address and resolve the problem. If the level 1 representative cannot address the problem to the customer's satisfaction, it is passed on to a level 2 representative, who would have more technical depth to address the problem. The process continues until the problem is resolved, or the maximum level of help available is exhausted for the problem.

At each level of the help desk, a knowledge base serves as a repository of the information regarding the typical problems encountered by the users, and typical solutions used to address them. A knowledge base may contain previous instances of a similar problem, and the different steps needed to resolve them. It may also contain instructions describing the sequence of tests to be performed on any problem, and advice on actions to be taken based on the sequence of the tests.

The knowledge base may be created in a static manner or be updated dynamically by means of analysis tools. The analysis tools look at the history of the information maintained in the trouble tickets, and mine them to obtain

with better guidance for the future. Case-based reasoning is a common tool applied to generate knowledge bases from a history of reported problems.

The knowledge base developed by the help desk and such analytic tools are also of tremendous help to the system administrator for operational system management. The information in that knowledge base can be used for proactive diagnosis of faults, performance and configuration problems before they are reported by a user.

10.3 Web, Web 2.0, and Management

A good resource for all aspects of system management today is the Internet. On the Internet, which happens to be updated continuously by a large number of people, it is quite common for system administrators and others to post information about the problems they have experienced in the operation of their systems, attempts made to resolve them, things that worked, and things that did not work.

The same resource can be quite useful for any system administrator who has the daunting task of diagnosing the cause of a failure, understand the causes of poor performance. A search on any of the major Internet search engines regarding the problem one is encountering is likely to result in pointers to various pages which described how that problem may be solved, or at the very least, the attempts that were made to solve the problem. Frequently, the results may also indicate other changes to system configuration or management which can improve the quality of management provided to the computer system.

The one challenge with information on the web is it has a relatively high amount of noise. If one is searching for something very specific, e.g., an error code encountered in a fault event, the search is more likely to provide useful information. On the other hand, if one searches for a more general problem such as a machine running slow without any specific error code, several links may show up that may not provide useful information. Furthermore, the information on the web is not authoritative, so caution must be exercised in following it.

Web 2.0 [5] refers to tools available on the Internet which enable collaboration among different users on the Internet dispersed across several geographies. The term is used to include wikis – web sites where visitors can edit as well as view the content, social networking sites, and other Internet-based collaboration tools. Such collaboration sites focused on systems management provide a more focused area to search for related information, and may be more effective than a broad search on the Internet, and show results that are more relevant and somewhat more authoritative than a generic Internet search.

As more and more collaboration sites focusing on systems and network management start taking shape, one would expect them to start becoming a repository of the knowledge required to manage and operate computer systems and networks effectively.

10.4 Summary

This chapter has looked at some of the areas that are related to systems management and provided a brief summary of them. Most of these areas impact systems management through all of the life cycle of a computer system. They have an impact and influence on the techniques and systems that will be used for systems management.

Like all other fields, the boundaries of what gets defined as systems management are somewhat flexible. Security management, which once used to be part of systems management, has grown large enough to become a field in its own rights. As new technologies emerge and reshape computing systems, some of the related fields we have discussed in this chapter may end up getting absorbed into systems management. It is also possible that some fields considered part of systems management, e.g., fault management, may grow large enough to become considered a field of computer science in its own rights.

After reading the various chapters of the book, you should have gained an overall insight of the different aspects of systems management and be ready to take on the many challenges that you may face one day managing computer systems.

10.5 Review Questions

1. In this book, we have followed an FCAP model for studying system management. An alternate approach to study system management would be to study it as a collection of various processes required for infrastructure management. Discuss the advantages and disadvantages of both the approaches.
2. How does a helpdesk system impact the operations of the systems management team and vice versa?
3. Discuss the advantages and disadvantages of relying on web-based information for system management purposes.

References

1. E. Rozemeijer, J. Van Bon, and T. Verheijen, Frameworks for IT Management: A Pocket Guide, Van Haren Publishing, 2007.
2. Foundations of IT Service Management based on ITIL V3, Inform-IT, Van Haren Publishing, 2007.
3. D. McBride, A Guide to Help Desk Technology, Tools & Techniques, Course Technology, 2000.
4. B. Czegel, Running an Effective Help Desk, John Wiley & Sons, 1998.
5. D. Nickull, D. Hinchcliffe, and J. Governor, Web 2.0 Architectures, O'Reilly, 2009.

Index