

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.

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

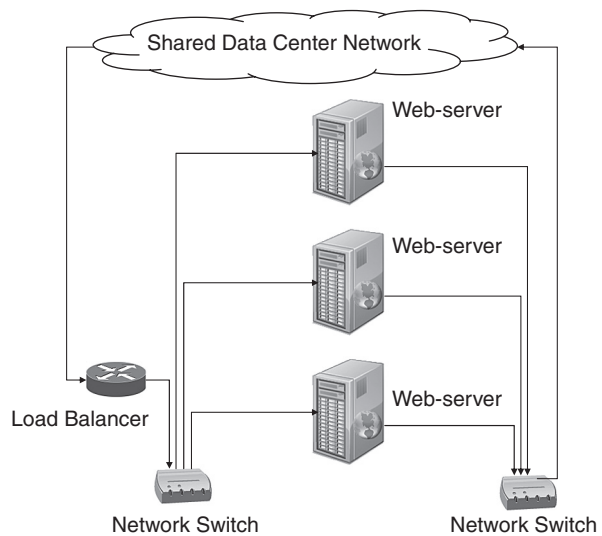


Fig. 2.1 Model of a web site

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 use for its web site?

Answer: The average number of active requests we would expect in the system would be $250 \text{ ms} \times 100 \text{ request/s}$ 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 k th

component is C_k requests per second, then each resource on the average visits the k th 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 $50 \cdot (40 + 3 \cdot 10)$ 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 or 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.

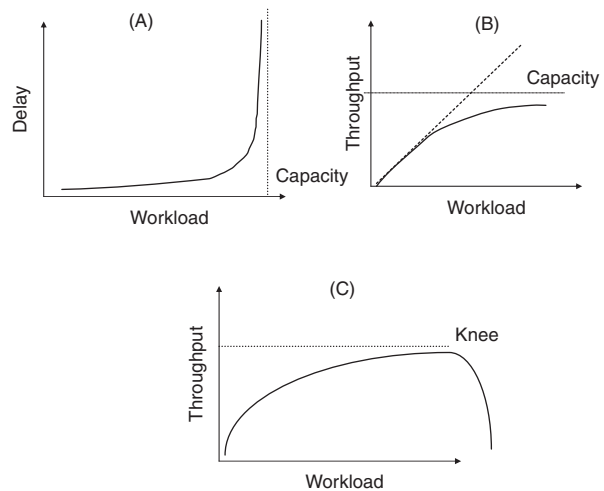


Fig. 2.2 Canonical delay and throughput curves

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 Mbps. 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 Mbps.

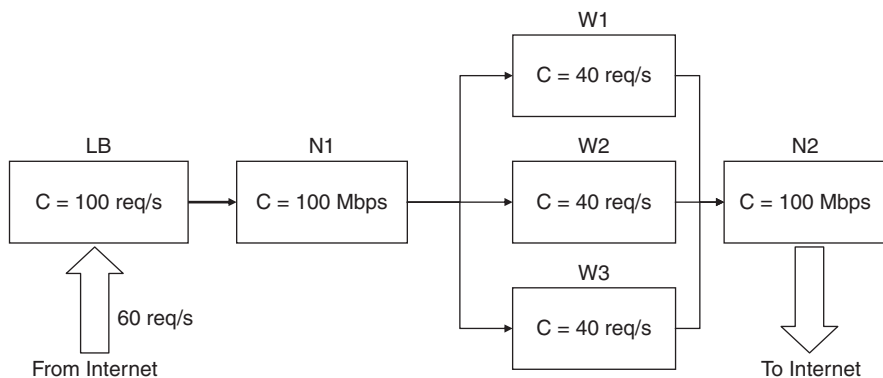


Fig. 2.3 Example of performance analysis

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 $F(t)$ 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 i th web server, and links L_{2i} connect the i th web server with network switch N2.

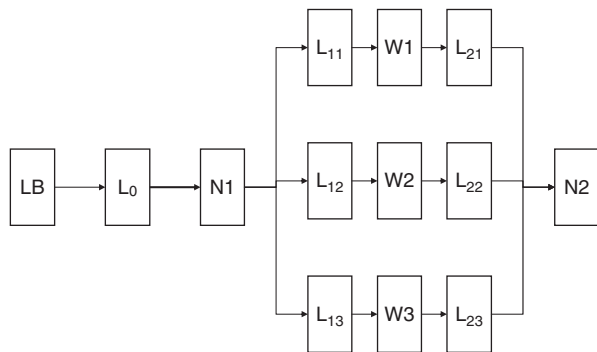


Fig. 2.4 Reliability block diagram of the web site

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

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

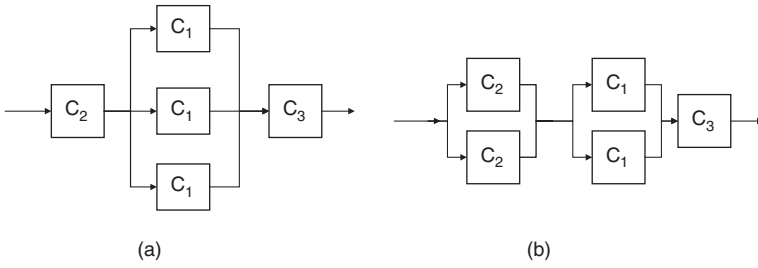


Fig. 2.5 Comparison of relative reliability

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.

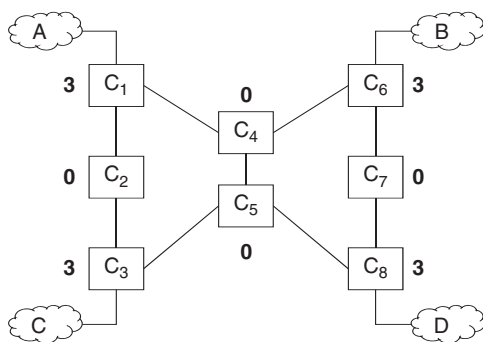


Fig. 2.6 Criticality of routers in a network

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.

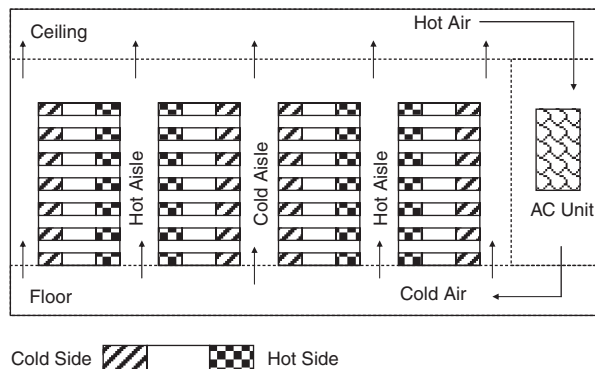
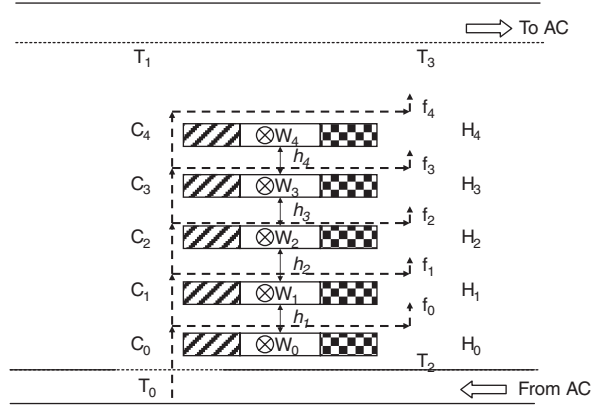


Fig. 2.7 Hot and cold aisles in a typical data center

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 i th 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 i th equipment is W_i and the distance between the i th and $i+1$ th equipment is given by h_i . The temperature in the cold aisle after the i th equipment is C_i and the temperature in the hot aisle after the i th 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 $\sum f_i$ is less than F , we can obtain the temperature on the cold aisle of the side by the approximation

$$C_0 = T_0,$$

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

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 in 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.

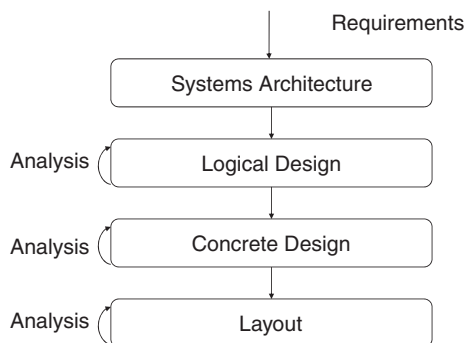


Fig. 2.9 Systems planning process

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.

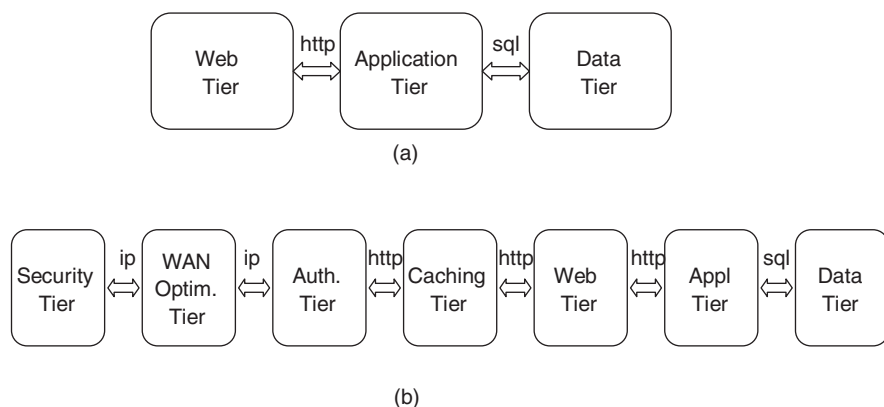


Fig. 2.10 Tiered architectures

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).

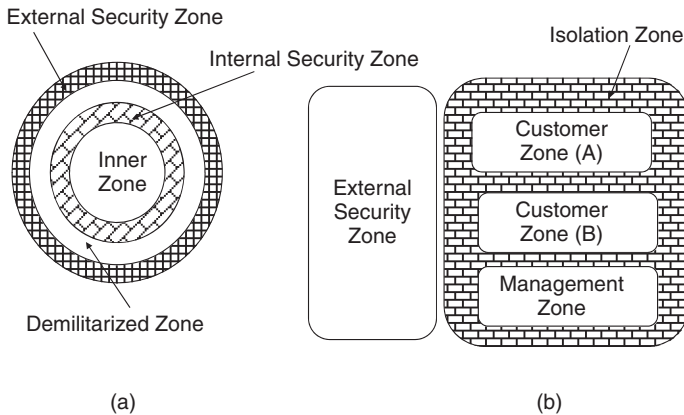


Fig. 2.11 Tiered architectures

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.

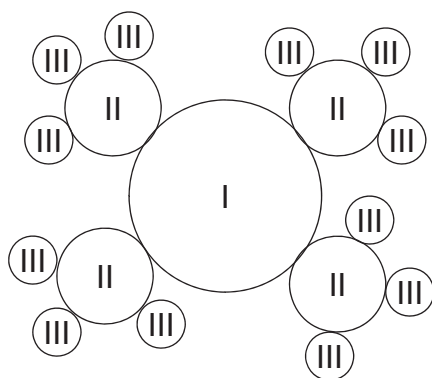


Fig. 2.12 Hierarchical architecture

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 inter-connection 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. Bertsekas 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, *Computer 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.



<http://www.springer.com/978-0-387-89008-1>

Principles of Computer Systems and Network
Management

Verma, D.C.

2009, XV, 260 p., Hardcover

ISBN: 978-0-387-89008-1