The Practice of System and Network Administration: Volume 1: DevOps and other Best Practices for Enterprise IT, 3rd Edition

Table of Contents

Chapter 13. Server Hardware Strategies

Another group of machines in your organization is the server fleet. This population of machines is used to provide application services, web services, databases, batch computation, back-office processing, and so on.

A single machine might provide one service or many. For example, a single server might be a dedicated file server, or it might be a file server, a DNS server, and a wiki server while performing many other functions. Alternatively, some services might be too large to fit on a single machine. Instead, the work of the service may be distributed over many machines. For example, Google's Gmail service is distributed over thousands of machines, each doing a small fraction of the work.

By definition a server has dependents, usually many. A single server may have hundreds of clients relying on it. A web server may have thousands of users depending on it. Contrast this to a laptop or desktop, which generally has just a single user.

In a model that has one big server, or just a few, any investment made in a server is amortized over all the dependents or users. We justify any additional expense to improve performance or increase reliability by looking at the dependents' needs. In this model servers are also expected to last many years, so paying for spare capacity or expandability is an investment in extending its life span.

By comparison, when we have hundreds or thousands of servers, saving a little money on a per-machine basis saves a lot of money overall. If we are buying 100 machines, and can save a few dollars on each one, those savings add up quickly.

This chapter's advice applies to a typical enterprise organization that mostly purchases off-the-shelf hardware and software, with a limited number of home-grown applications. Volume 2 of this book series is more specifically geared toward web-based applications and large clusters of machines.

There are many strategies for providing server resources. Most organizations use a mix of these strategies.

The three most common strategies are:

- **All eggs in one basket:** One machine used for many purposes
- **Beautiful snowflakes:** Many machines, each uniquely configured
- **Buy in bulk, allocate fractions:** Large machines partitioned into many smaller virtual machines using virtualization or containers
- In addition, there are variations and alternatives:
- **Grid computing:** Many machines managed one as unit
- **Blade servers:** A hardware architecture that places many machines in one chassis
- **Cloud-based compute services:** Renting use of someone else's servers
- **Software as a service (SaaS):** Web-hosted applications
- **Server appliances:** Purpose-built devices, each providing a different service

The rest of this chapter describes these strategies and provides advice on how to manage them.

What Is a Server?

The term “server” is overloaded and ambiguous. It means different things in different contexts. The phrase “web server” might refer to a host being used to provide a web site (a machine) or the software that implements the HTTP protocol (Apache HTTPD). Unless specified, this book uses the term “server” to mean a machine. It refers to a “service” as the entire hardware/software combination that provides the service users receive. For example, an email server (the hardware) runs MS Exchange (the software) to provide the email service for the department.

Note that Volume 2 of this book series deals with this ambiguity differently. It uses the term “server” to mean the server of a client–server software arrangement—for example, an Apache HTTPD server. When talking about the hardware it uses the word “machine.”

13.1 All Eggs in One Basket

The most basic strategy is to purchase a single server and use it for many services. For example, a single machine might serve as the department’s DNS server, DHCP server, email server, and web server. This is the “all eggs in one basket” approach. We don’t recommend this approach. Nevertheless, we see it often enough that we felt we should explain how to make the best of it.

If you are going to put all your eggs in one basket, make sure you have a really, really, really strong basket. Any hardware problems the machine has will affect many services. Buy top-of-the-line hardware and a model that has plenty of expansion slots. You’ll want this machine to last a long time.

In this setting it becomes critical to ensure data integrity. Hard disks fail. Expecting them to never fail is irrational. Therefore RAID should be used so that a single disk can fail and the system will continue to run. Our general rule of thumb is to use a 2-disk mirror (RAID 1) for the boot disk. Additional storage should be RAID 5, 6, or 10 as needed. RAID 0 offers zero data integrity protection (the “zero” in the name reflects this). Obviously, different applications demand different configurations but this fits most situations. Putting plain disks without RAID 1 or higher on a server like this is asking for trouble. Any disk malfunction will result in a bad day as you restore from backups and deal with any data loss. When RAID was new, it was expensive and rare. Now it is inexpensive and should be the default for all systems that need reliability.

Data integrity also requires regular data backups to tape or another system. These backups should be tested periodically. As explained in Section 43.3.3, RAID is not a backup strategy. If this system dies in a fire, RAID will not protect the data.

If a server is expected to last a long time, it will likely need to be expanded. Over time most systems will need to handle more users, more data, or more applications. Additional slots for memory, interfaces, and hard drive bays make it easy to upgrade the system without replacing it. Upgradable hardware is more expensive than fixed-configuration equivalents, but that extra expense is an insurance policy against more costly upgrades later. It is much less expensive to extend the life of a machine via an incremental upgrade such as adding RAM than to do a wholesale replacement of one machine with another.

Forklift Upgrades

The term **forklift upgrade** is industry slang for a wholesale replacement. In such a situation you are removing one machine with a metaphorical forklift and dropping a replacement in its place.

Another problem with the “one basket” strategy is that it makes OS upgrades very difficult and risky. Upgrading the operating system is an all-or-nothing endeavor. The OS cannot be upgraded or patched until all the services are verified to work on the new version. One application may hold back all the others.

Upgrading individual applications also becomes more perilous. For example, what if upgrading one application installs the newest version of a shared library, but another application works only with the older version? Now you have one application that isn’t working and the only way to fix it is to downgrade it, which will make the other application fail. This Catch-22 is known as **dependency hell.**

Often we do not discover these incompatibilities until after the newer software is installed or the operating system is upgraded. Some technologies can be applied to help in this situation. For example, some disk storage systems have the ability to take a snapshot of the disk. If an upgrade reveals incompatibility or other problems, we can revert to a previous snapshot. RAID 1 mirrors and file systems such as ZFS and Btrfs have snapshotting capabilities.

Upgrading a single machine with many applications is complex and is the focus of Chapter 33, “Server Upgrades.”

The more applications a big server is running, the more difficult it becomes to schedule downtime for hardware upgrades. Very large systems permit hardware upgrades to be performed while they remain running, but this is rare for components such as RAM and CPUs. We can delay the need for downtime by buying extra capacity at the start, but this usually leads to more dependencies on the machine, which in turn exacerbates the scheduling problem we originally tried to solve.

13.2 Beautiful Snowflakes

A better strategy is to use a separate machine for each service. In this model we purchase servers as they are needed, ordering the exact model and configuration that is right for the application.

Each machine is sized for the desired application: RAM, disk, number and speeds of NICs, and enough extra capacity, or expansion slots, for projected growth during the expected life of the machine. Vendors can compete to provide their best machine that meets these specifications.

The benefit of this strategy is that the machine is the best possible choice that meets the requirements. The downside is that the result is a fleet of unique machines. Each is a beautiful, special little snowflake.

While snowflakes are beautiful, nobody enjoys a blizzard. Each new system adds administrative overhead proportionally. For example, it would be a considerable burden if each new server required learning an entirely new RAID storage subsystem. Each one would require learning how to configure it, replace disks, upgrade the firmware, and so on. If, instead, the IT organization standardized on a particular RAID product, each new machine would simply benefit from what was learned earlier.

13.2.1 Asset Tracking

When managing many unique machines it becomes increasingly important to maintain an inventory of machines. The inventory should document technical information such as the operating system and hardware parameters such as amount of RAM, type of CPU, and so on. It should also track the machine owner (either a person or a department), the primary contact (useful when the machine goes haywire), and the services being used. When possible, this information should be collected through automated means; this makes it less likely the information will become outdated. If automated collection is not possible, an automated annual review, requiring affirmations from the various contacts, can detect obsolete information that needs to be updated.

A variety of inventory applications are available, many of which are free or low cost. Having even a simple inventory system is better than none. A spreadsheet is better than keeping this information in your head. A database is better than a spreadsheet.

13.2.2 Reducing Variations

Always be on the lookout for opportunities to reduce the number of variations in platforms or technologies being supported. Discourage gratuitous variations by taking advantage of the fact that people lean toward defaults. Make right easy: Make sure that the *lazy path* is the path you want people to take. Select a default hardware vendor, model, and operating system and make it super easy to order. Provide automated OS installation, configuration, and updates. Provide a wiki with information about recommended models and options, sales contact information, and assistance. We applaud you for being able to keep all that information in your head, but putting it on a wiki helps the entire organization stick to standards. That helps you in the long term.

The techniques discussed in Part II, "Workstation Fleet Management," for managing variation also apply to servers. In particular, adopt a policy of supporting a limited number of generations of hardware. When a new generation of hardware is introduced, the oldest generation is eliminated. This practice also works for limiting the number of OS revisions in use.

Google's SysOps Death Squads

Google's SysOps (internal IT) organization had a policy of supporting at most two Linux releases at any given time, and a limited number of hardware generations. For a new OS release or hardware model to be introduced into the ecosystem, the oldest one had to be eliminated.

Eliminating the old generation didn't happen automatically. People were not continuing to use it because of laziness; there was always some deep technical reason, dependency, lack of resources, or political issue preventing the change. Finding and eliminating the stragglers took focused effort.

To accelerate the process, a project manager and a few SAs would volunteer to form a team that would work to eliminate the stragglers. These teams were called "death squads," an insensitive name that could have been invented only by people too young to remember the politics of Central America in the late twentieth century.

The team would start by publishing a schedule: the date the replacement was available, the date the old systems would no longer receive updates, and the date the old systems would be disconnected from the network. There were no exceptions. Management had little sympathy for whining. Employees sometimes complained the first time they experienced this forced upgrade process but soon they learned that restricting technologies to a small number of generations was a major part of the Google operational way and that services had to be built with the expectation of being easy to upgrade or rebuild.
The team would identify machines that still used the deprecated technology. This would be published as a shared spreadsheet, viewable and editable by all. Columns would be filled in to indicate the status: none, owner contacted, owner responded, work in progress, work complete. As items were complete, they would be marked green. The system was very transparent. Everyone in the company could see the list of machines, who had stepped up to take ownership of each, and their status. It was also very visible if a team was not taking action.
The death squad would work with all the teams involved and offer support and assistance. Some teams already had plans in place. Others needed a helping hand. Some needed cajoling and pressure from management. The death squad would collaborate and assist until the older technology was eliminated.
This system was used often and was a key tactic in preventing Google's very large fleet from devolving into a chaotic mess of costly snowflakes.

13.2.3 Global Optimization

While it sounds efficient to customize each machine to the exact needs of the service it provides, the result tends to be an unmanageable mess.
This is a classic example of a local optimization that results in a global de-optimization. For example, if all server hardware is from one vendor, adding a single machine from a different vendor requires learning a new firmware patching system, investing in a new set of spare parts, learning a new customer support procedure, and so on. The added work for the IT team may not outweigh the benefits gained from using the new vendor.
The one-off hardware might be rationalized by the customer stating that spares aren't required, that firmware upgrades can be skipped, or that the machine will be self-supported by the customer. These answers are nice in theory, but usually lead to even more work for the IT department. When there is a hardware problem, the IT department will be blamed for any difficulties. Sadly the IT department must usually say no, which makes this group look inflexible.

The Unique Storage Device

A university professor in the mathematics department purchased a custom storage system against the advice of the IT department. He rationalized the exception by promising to support it himself.
During a software upgrade the system stopped working. The professor was unable to fix the problem, and other professors who had come to rely on the system were dead in the water.
After a month of downtime, the other professors complained to the department chair, who complained to the IT department. While the department chair understood the

agreement in place, he'd hear none of it. This was an emergency and IT's support was needed. The IT group knew that once they got involved, any data loss would be blamed on them. Their lack of experience with the device increased that risk.
The end of this story is somewhat anticlimactic. While the IT group was establishing an action plan in cooperation with the vendor, the professor went rogue and booted the storage server with a rescue disk, wiped and reinstalled the system, and got it working again. Only then did he concede that there was no irreplaceable data on the system.
Unfortunately, this experience taught the professor that he could ignore IT's hardware guidance since he'd get support when he needed it, and that he didn't really need it since he fixed the problem himself.
The other faculty who had come to depend on this professor's system realized the value of the support provided by the IT group. While the previous system was "free," losing access for a month was an unacceptable loss of research time. The potential for catastrophic data loss was not worth the savings.
What the IT department learned was that there was no such thing as self-support. In the future, they developed ways to be involved earlier on in purchasing processes so that they could suggest alternatives before decisions had been made in the minds of their customers.

13.3 Buy in Bulk, Allocate Fractions

The next strategy is to buy computing resources in bulk and allocate fractions of it as needed.
One way to do this is through virtualization. That is, an organization purchases large physical servers and divides them up for use by customers by creating individual virtual machines (VMs). A virtualization cluster can grow by adding more physical hardware as more capacity is needed.
Buying an individual machine has a large overhead. It must be specified, ordered, approved, received, rack mounted, and prepared for use. It can take weeks. Virtual machines, by comparison, can be created in minutes. A virtualization cluster can be controlled via a portal or API calls, so automating processes is easy.
VMs can also be resized. You can add RAM, vCPUs, and disk space to a VM via an API call instead of a visit to the datacenter. If customers request more memory and you add it using a management app on your iPhone while sitting on the beach, they will think you are doing some kind of magic.
Virtualization improves computing efficiency. Physical machines today are so powerful that applications often do not need the full resources of a single machine. The excess capacity is called **stranded capacity** because it is unusable in its current form. Sharing a large physical machine's power among many smaller virtual machines helps reduce stranded capacity, without getting into the "all eggs in one basket" trap.
Stranded capacity could also be mitigated by running multiple services on the same machine. However, virtualization provides better isolation than simple multitasking. The benefits of isolation include:

- **Independence:** Each VM can run a different operating system. On a single physical host there could be a mix of VMs running a variety of Microsoft Windows releases, Linux releases, and so on.

- **Resource isolation:** The disk and RAM allocated to a VM are committed to that VM and not shared. Processes running on one VM can't access the resources of another VM. In fact, programs running on a VM have little or no awareness that they are running on VMs, sharing a larger physical machine.
- **Granular security:** A person with root access on one VM does not automatically have privileged access on another VM. Suppose you had five services, each run by a different team. If each service was on its own VM, each team could have administrator or root access for its VM without affecting the security of the other VMs. If all five services were running on one machine, anyone needing root or administrator access would have privileged access for all five services.
- **Reduced dependency hell:** Each machine has its own operating system and system libraries, so they can be upgraded independently.

13.3.1 VM Management

Like other strategies, keeping a good inventory of VMs is important. In fact, it is more important since you can't walk into a datacenter and point at the machine. The cluster management software will keep an inventory of which VMs exist, but you need to maintain an inventory of who owns each VM and its purpose.

Some clusters are tightly controlled, only permitting the IT team to create VMs with the care and planning reminiscent of the laborious process previously used for physical servers. Other clusters are general-purpose compute farms providing the ability for customers to request new machines on demand. In this case, it is important to provide a self-service way for customers to create new machines. Since the process can be fully automated via the API, not providing a self-service portal or command-line tool for creating VMs simply creates more work for you and delays VM creation until you are available to process the request. Being able to receive a new machine when the SAs are unavailable or busy reduces the friction in getting the compute resources customers need. In addition to creating VMs, users should be able to reboot and delete their own VMs.

There should be limits in place so that customers can't overload the system by creating too many VMs. Typically limits are based on existing resources, daily limits, or per-department allocations.

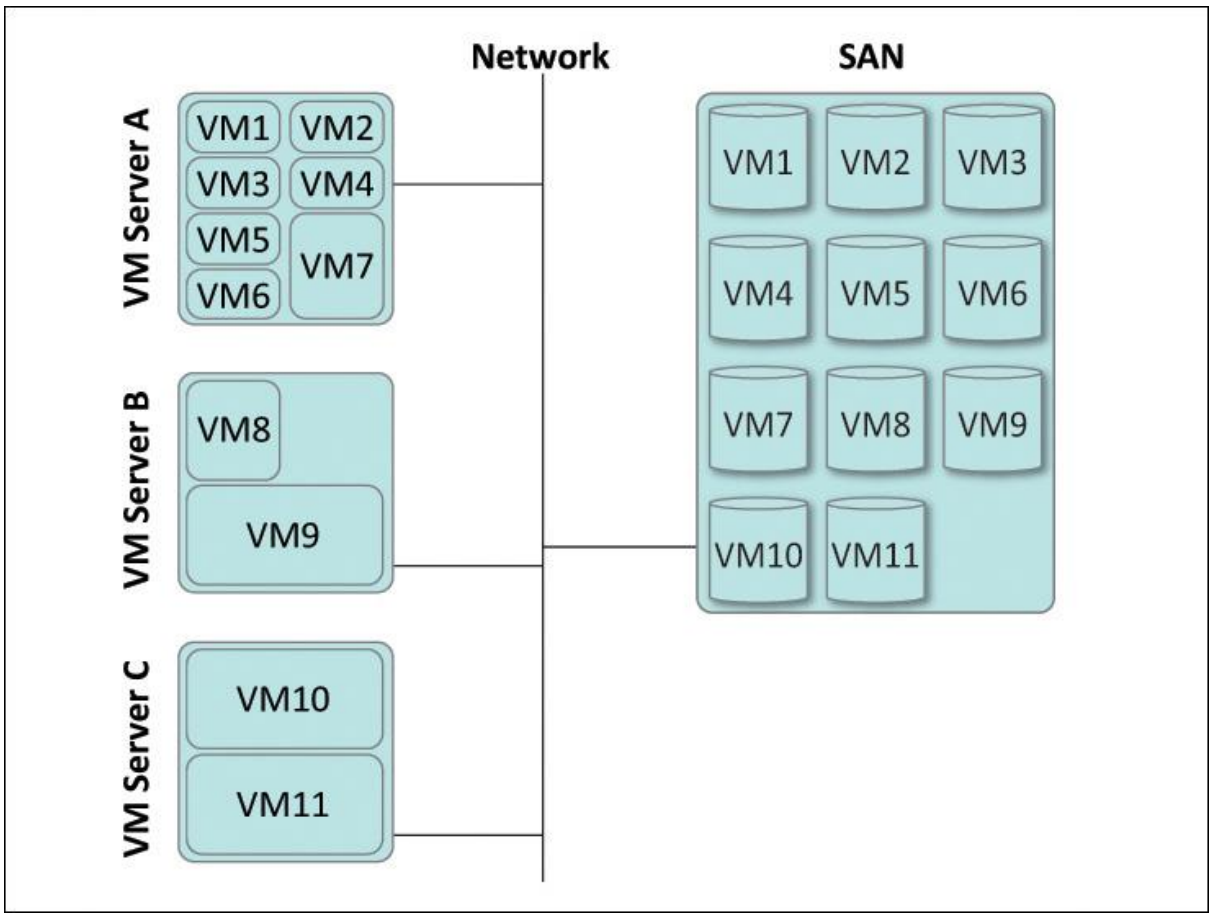
Users can become confused if you permit them to select any amount of disk space and RAM. They often do not know what is required or reasonable. One strategy is to simply offer reasonable defaults for each OS type. Another strategy is to offer a few options: small, medium, large, and custom.

As in the other strategies, it is important to limit the amount of variation. Apply the techniques described previously in this chapter and in Part II, "Workstation Fleet Management." In particular, adopt a policy of supporting a limited number of OS versions at any given time. For a new version to be introduced, the oldest generation is eliminated.

13.3.2 Live Migration

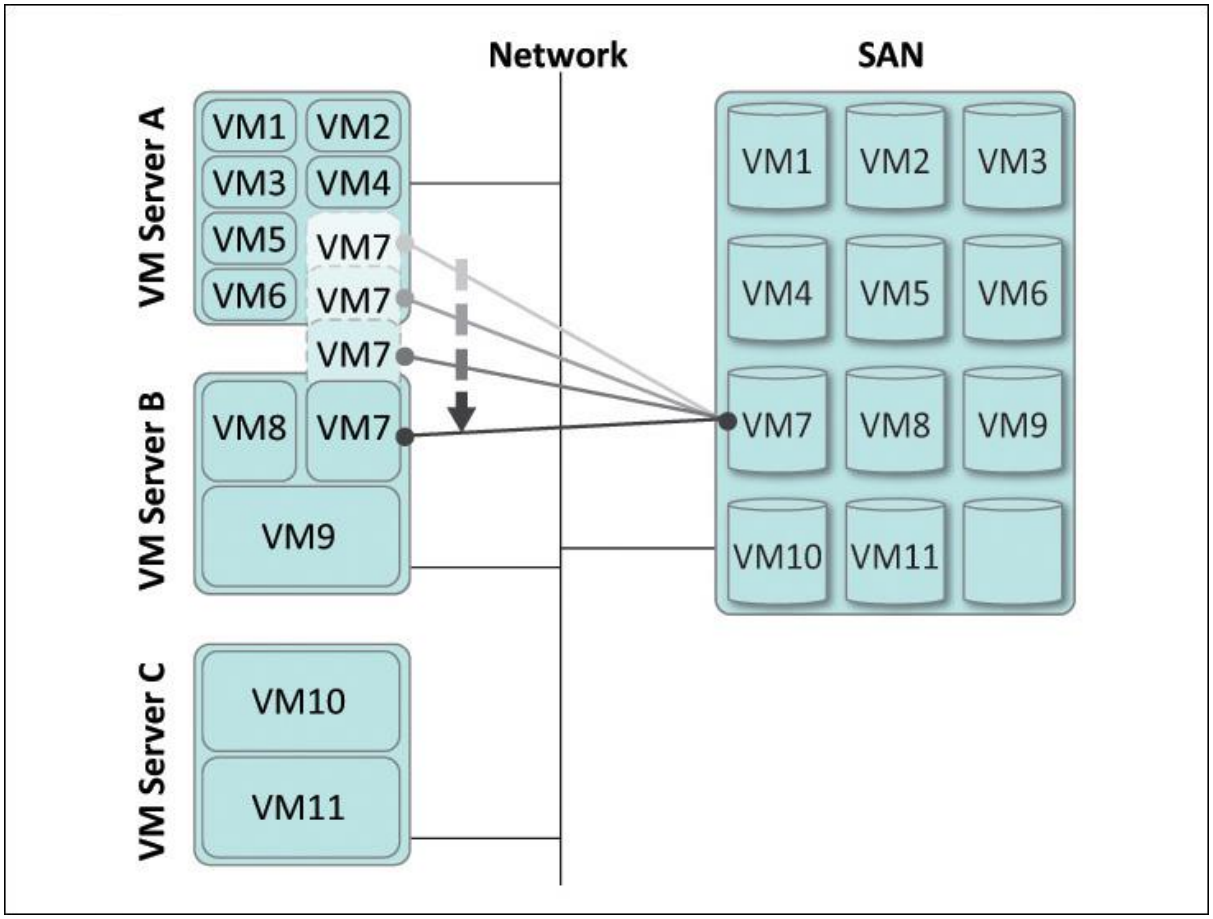
Most virtual machine cluster management systems permit live migration of VMs, which means a VM can be moved from one physical host to another while it is running. Aside

from a brief performance reduction during the transition, the users of the VM do not even know they're being moved.
Live migration makes management easier. It can be used to rebalance a cluster, moving VMs off overloaded physical machines to others that are less loaded. It also lets you work around hardware problems. If a physical machine is having a hardware problem, its VMs can be evacuated to another physical machine. The owners of the VM can be blissfully unaware of the problem. They simply benefit from the excellent uptime.
The architecture of a typical virtualization cluster includes many physical machines that share a SAN for storage of the VM's disks. By having the storage external to any particular machine, the VMs can be easily migrated between physical machines.
Shared storage is depicted in Figure 13.1. The VMs run on the VM servers and the disk volumes they use are stored on the SAN. The VM servers have little disk space of their own.



Three VM (Virtual Machine) servers VM server A, VM Server B, and VM Server C are shown on the left and SAN (Storage Area Network) is shown on the right. VM Server A consist of seven blocks labeled, VM1, VM2, VM3, VM4, VM5, VM6, and VM7. Virtual Server B consists of two blocks labeled, VM8 and VM9, and Virtual Server C shows two blocks labeled, VM10 and VM11. The Storage Area Network shows eleven blocks labeled, VM1, VM2, VM3, VM4, VM5, VM6, VM7, VM8, VM9, VM10, and VM11. A network connects VM server A, VM Server B, and VM Server C, and the network is connected to the Storage Area Network.
Figure 13.1: VM servers with shared storage

In Figure 13.2 we see VM7 is being migrated from VM Server A to VM Server B. Because VM7's disk image is stored on the SAN, it is accessible from both VM servers as the migration process proceeds. The migration process simply needs to copy the RAM and CPU state between A and B, which is a quick operation. If the entire disk image had to be copied as well, the migration could take hours.



Three VM (Virtual Machine) servers VM server A, VM Server B, and VM Server C are shown on the left and SAN (Storage Area Network) is shown on the right. VM Server A consist of seven blocks labeled, VM1, VM2, VM3, VM4, VM5, VM7, VM6, VM7, VM7, empty, VM7, VM8. Virtual Server B consists of two blocks labeled, VM8 and VM9, and Virtual Server C shows two blocks labeled, VM10 and VM11. The Storage Area Network shows eleven blocks labeled, VM1, VM2, VM3, VM4, VM5, VM6, VM7, VM8, VM9, VM10, and VM11. A network connects VM server A, VM Server B, and VM Server C, and the network is connected to the Storage Area Network. VM7 in the Storage Area Network is connected to four VM7s of VM Server A.

Figure 13.2: Migrating VM7 between servers

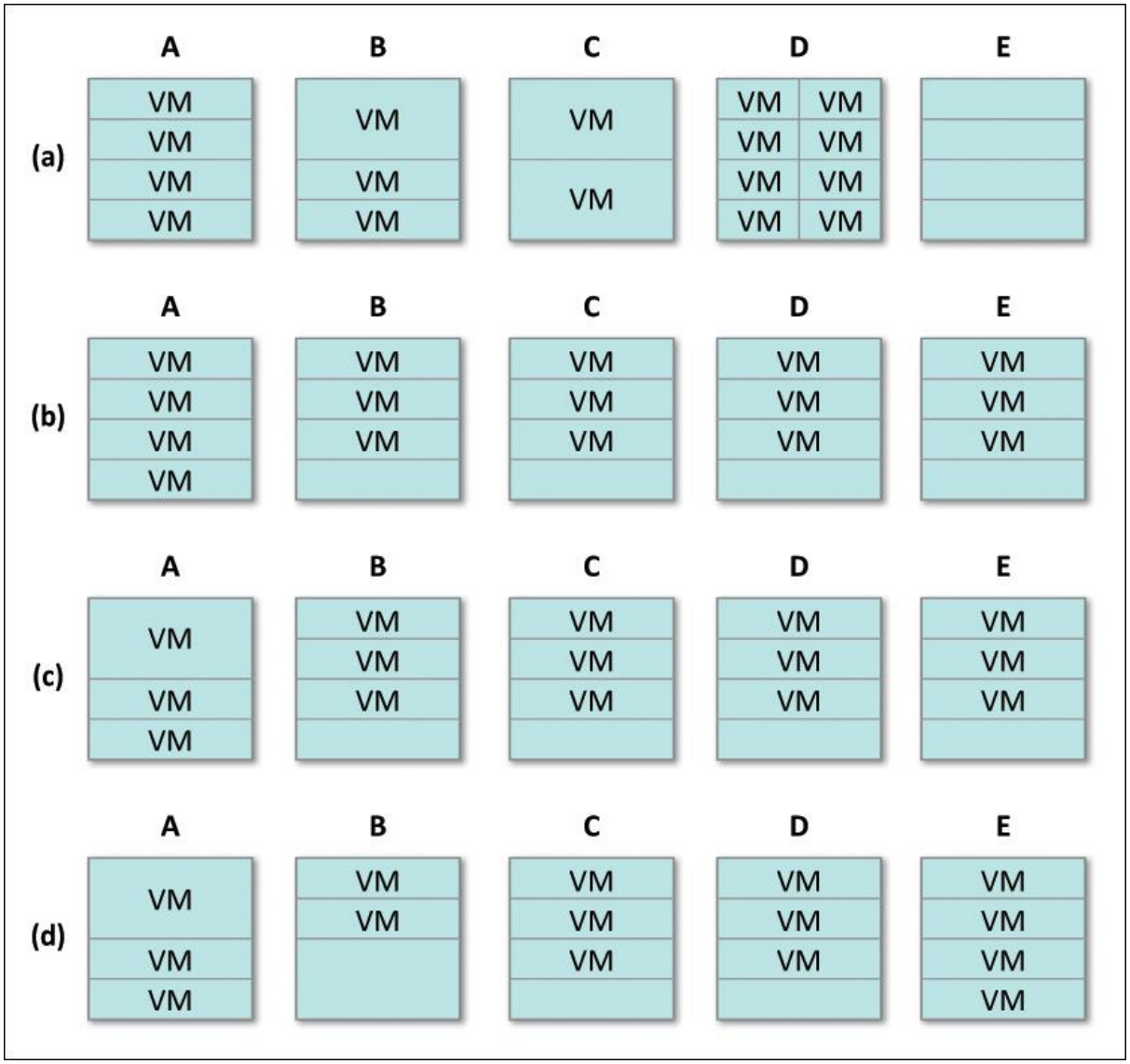
13.3.3 VM Packing

While VMs can reduce the amount of stranded compute capacity, they do not eliminate it. VMs cannot span physical machines. As a consequence, we often get into situations where the remaining RAM on a physical machine is not enough for a new VM.

The best way to avoid this is to create VMs that are standard sizes that pack nicely. For example, we might define the small configuration such that exactly eight VMs fit on a physical machine with no remaining stranded space. We might define the medium configuration to fit four VMs per physical machine, and the large configuration to fit two VMs per physical machine. In other words, the sizes are $\frac{1}{8}$, $\frac{1}{4}$, and $\frac{1}{2}$. Since the denominators are powers of 2, combinations of small, medium, and large configurations will pack nicely.

13.3.4 Spare Capacity for Maintenance

If a physical machine needs to be taken down for repairs, there has to be a place where the VMs can be migrated if you are to avoid downtime. Therefore you must always reserve capacity equivalent to one or more physical servers. Reserving capacity equivalent to one physical server is called *N* + 1 redundancy. You may need *N* + 2 or higher redundancy if there is a likelihood that multiple physical machines will need maintenance at the same time. For example, a network with hundreds of physical machines is likely to have many physical machines out for repair at any given time.
One strategy is to keep one physical machine entirely idle so that, when needed, the VMs from the machine to be repaired can all be migrated to this machine. This scheme is depicted in Figure 13.3(a). When physical machine A, B, C, or D needs maintenance, its VMs are transferred to machine E. Any one machine can be down at a time. It doesn't matter which combination of VMs are on the machine, as long as the spare is as large as any of the other machines.



In machine A, column A shows a block with four rows labeled VM, column B shows three rows labeled VM, column C shows two rows labeled VM, column D shows eight cells labeled VM, and column E shows four blank rows. In machine B, column A shows a block with four rows labeled VM, columns B, C, D, and E have four rows in which the first three rows are labeled VM. In machine c, column A has three rows labeled VM, columns B, C, D, and E have four rows in which the first three rows are labeled VM. In machine d, column A has three rows labeled VM, column B has three rows in which two rows are labeled VM, columns C and D have four rows in which three rows are labeled VM, column E has four rows labeled E.
Figure 13.3: VM packing

Of course, this arrangement means that the spare machine is entirely unused. This seems like a waste since that capacity could be used to alleviate I/O pressure such as contention for network bandwidth, disk I/O, or other bottlenecks within the cluster.
Another strategy is to distribute the spare capacity around the cluster so that the individual VMs can share the extra I/O bandwidth. When a machine needs to be evacuated for repairs, the VMs are moved into the spare capacity that has been spread around the cluster.

This scheme is depicted in Figure 13.3(b). If machine A requires maintenance, its VMs can be moved to B, C, D, and E. Similar steps can be followed for the other machines. Unfortunately, this causes a new problem: There might not be a single machine with enough spare capacity to accept a VM that needs to be evacuated from a failing physical host. This situation is depicted in Figure 13.3(c). If machine A needs maintenance, there is no single machine with enough capacity to receive the largest of its VMs. VMs cannot straddle two physical machines. This is not a problem for machines B through E, whose VMs can be evacuated to the remaining space.
If machine A needed to be evacuated, first we would need to consolidate free space onto a single physical machine so that the larger VM has a place to move to. This is known as a Towers of Hanoi situation, since moves must be done recursively to enable other moves. These additional VM moves make the evacuation process both more complex and longer. The additional VM migrations affect VMs that would otherwise not have been involved. They now must suffer the temporary performance reduction that happens during migration, plus the risk that the migration will fail and require a reboot.
Figure 13.3(d) depicts the same quantity and sizes of VMs packed to permit any single machine to be evacuated for maintenance.
Some VM cluster management systems include a tool that will calculate the minimum number of VM moves to evacuate a physical machine. Other systems simply refuse to create a new VM if it will create a situation where the cluster is no longer $N + 1$ redundant. Either way, virtual clusters should be monitored for loss of $N + 1$ redundancy, so that you are aware when the situation has happened and it can be corrected proactively.

13.3.5 Unified VM/Non-VM Management

Most sites end up with two entirely different ways to request, allocate, and track VMs and non-VMs. It can be beneficial to have one system that manages both. Some cluster management systems will manage a pool of bare-metal machines using the same API as VMs. Creating a machine simply allocates an unused machine from the pool. Deleting a machine marks the machine for reuse.
Another way to achieve this is to make everything a VM, even if that means offering an an extra large size, which is a VM that fills the entire physical machine: $\frac{1}{1}$. While such machines will have a slight performance reduction due to the VM overhead, unifying all machine management within one process benefits customers, who now have to learn only one system, and makes management easier.

13.3.6 Containers

Containers are another virtualization technique. They provide isolation at the process level instead of the machine level. While a VM is a machine that shares physical hardware with other VMs, each container is a group of processes that run in isolation on the same machine. All of the containers run under the same operating system, but each container is self-contained as far as the files it uses. Therefore there is no dependency hell.
Containers are much lighter weight and permit more services to be packed on fewer machines. Docker, Mesos, and Kubernetes are popular systems for managing large

numbers of containers. They all use the same container format, which means once a container is created, it can be used on a desktop, server, or huge farm of servers.
A typical containerization cluster has many physical machines running a stripped-down version of a server OS. Since each container is self-contained, the OS does not need much installed other than the kernel, the management framework, and the files required to boot the system. CoreOS and Red Hat Atomic are two such stripped-down operating systems.
Pros and cons of virtualization and containerization, as well as technical details of how they work, can be be found in Volume 2, Chapter 3, “Selecting a Service Platform,” of this book series.
Self-service container administration is made possible by Kubernetes and other container management systems. Customers provide the container specification and the management system finds an available machine and spins up the container.
One bad situation people get into with containers is a lack of reproducibility. After using the system for a while, there comes a day when a container needs to be rebuilt from scratch to upgrade a library or other file. Suddenly the team realizes no one is around who remembers how the container was made. The way to prevent this is to make the container’s creation similar to compiling software: A written description of what is to be in the container is passed through automation that reads the description and outputs the container. The description is tracked in source code control like any other source code. Containers should be built using whatever continuous integration (CI) system is used for building other software in your organization.

13.4 Grid Computing

Grid computing takes many similar machines and manages them as a single unit. For example, four racks of 40 servers each would form a grid of 160 machines. Each one is configured exactly alike—same hardware and software.
To use the grid, a customer specifies how many machines are needed and which software package to run. The grid management system allocates the right number of machines, installs the software on them, and runs the software. When the computation is done, the results are uploaded to a repository and the software is de-installed.
A big part of grid management software is the scheduling algorithm. Your job is held until the number of machines requested are available. When you request many machines, that day may never come. You would be very disappointed if your job, which was expected to run for an hour, took a week to start.
To start a big request, one has to stall small requests until enough machines are available, which wastes resources for everyone. Suppose your job required 100 machines. The scheduler would stall any new jobs until 100 machines are idle. Suppose at first there are 50 machines idle. An hour later, another job has completed and now there are 75 machines idle. A day later a few more jobs have completed and there are 99 machines idle. Finally another job completes and there are 100 free machines, enough for your job to run. During the time leading up to when your job could run, many machines were sitting idle. In fact, more than 75 compute-days were lost in this example.
The scheduler must be very smart and mix small and big requests to both minimize wait time and maximize utilization. A simple algorithm is to allocate half the machines for big jobs. They will stay busy if there is always a big job ready to run. Allocate the

other half of the machines for smaller jobs; many small and medium jobs will pass through those machines.
More sophisticated algorithms are invented and tested all the time. At Google the scheduling algorithms have become so complex that a simulator was invented to make it possible to experiment with new algorithms without having to put them into production.
Typically grids are very controlled systems. All allocations are done though the grid management and scheduling system. Each machine runs the same OS configuration. When entropy is detected, a machine is simply wiped and reloaded.
Because many machines are being purchased, shaving a few dollars off the cost of each machine pays big dividends. For example, a video card is not needed since these machines will not be used for interactive computing. For a 1,000-machine cluster, this can save thousands of dollars. RAID cards, fancy plastic front plates with awesome LED displays, redundant power supplies, and other add-ons are eliminated to save a few dollars here and there. This multiplies out to thousands or millions of dollars overall.
Data integrity systems such as RAID cards are generally not installed in grid hardware because the batch-oriented nature of jobs means that if a machine dies, the batch can be rerun.
While one wants a grid that is reliable, compute efficiency is more important. Efficiency is measured in dollars per transaction. Rather than examining the initial purchase price, the total cost of ownership (TCO) is considered. For example, ARM chips are less expensive than x86 Intel chips, but they are slower processors. Therefore you need more of them to do the same job. Is it better to have 1,000 high-powered Intel-based machines or 2,000 ARM-based machines? The math can be fairly complex. The ARM chips use less power but there are more of them. The amount of power used is related directly to how much cooling is needed. The additional machines require more space in the datacenter. What if the additional machines would result in a need to build more datacenters? TCO involves taking all of these factors into account during the evaluation.
Grid computing has other constraints. Often there is more bandwidth between machines on the same rack, and less bandwidth between racks. Therefore some jobs will execute faster if rack locality (putting all related processes in one rack) can be achieved.
Grid computing is more efficient than virtualization because it eliminates the virtualization overhead, which is typically a 5 to 10 percent reduction in performance.
Grids are easier to manage because what is done for one machine is done for all machines. They are fungible units—each one can substitute for the others. If one machine dies, the scheduler can replace it with another machine in the grid.
Case Study: Disposable Servers
Like many web sites, Yahoo! builds mammoth clusters of low-cost 1U PC servers. Racks are packed with as many servers as possible, with dozens or hundreds configured to provide each service required. Yahoo! once reported that when a unit died, it was more economical to power it off and leave it in the rack rather than repair the unit. Removing dead units might accidentally cause an outage if other cables were loosened in the process. Eventually the machine would be reaped and many dead machines would be repaired en masse.

13.5 Blade Servers

There is a lot of overhead in installing individual machines. Each machine needs to be racked, connected to power, networked, and so on. Blade servers reduce this overhead by providing many servers in one chassis.
A blade server has many individual slots that take motherboards, called **blades**, that contain either a computer or storage. Each blade can be installed quickly and easily because you simply slide a card into a slot. There are no cables to connect; the blade's connector conveys power, networking, and I/O connectivity. Additional capacity is added by installing more blades, or replacing older blades with newer, higher-capacity models.
Blades can be used to implement many of the strategies listed previously in this chapter. Computers within the blade chassis can be allocated individually, or used to create a virtualization cluster, or used to create a grid.
Another benefit of blade systems is that they are software configurable. Assigning a disk to a particular computer or a computer to a particular network is done through a management console or API. Does a particular blade computer need an additional hard drive? A few clicks and it is connected to storage from another blade. No trip to the datacenter is required.
Blade systems are most cost-effective when the chassis lasts many years, enabling you to upgrade the blades for the duration of its lifetime. This amortizes the cost of the chassis over many generations of blades. The worst-case scenario is that shortly after investing in a chassis, a newer, incompatible chassis model becomes available. When you buy a blade-based system, you are betting that the chassis will last a long time and that new blades will be available for its entire lifespan. If you lose the bet, a forklift upgrade can be very expensive.

Grid Uniformity

A division of a large multinational company was planning on replacing its aging multi-CPU server with large grid computing environment, implemented as a farm of blade servers. The application would be recoded so that instead of using multiple processes on a single machine, it would use processes spread over the blade farm. Each blade would be one node of a vast compute farm to which jobs could be submitted, with the results consolidated on a controlling server.
This had wonderful scalability, since a new blade could be added to the farm and be usable within minutes. No direct user logins were needed, and no SA work would be needed beyond replacing faulty hardware and managing which blades were assigned to which applications. To this end, the SAs engineered a tightly locked-down minimal-access solution that could be deployed in minutes. Hundreds of blades were purchased and installed, ready to be purposed as the customer required.
The problem came when application developers found themselves unable to manage their application. They couldn't debug issues without direct access. They demanded shell access. They required additional packages. They stored unique state on each machine, so automated builds were no longer viable. All of a sudden, the SAs found themselves managing 500 individual servers rather than one unified blade farm. Other divisions had also signed up for the service and made the same demands.
A number of things could have prevented this problem. Management should have required more discipline. Once the developers started requesting access,

management should have set limits that would have prevented the system from devolving into hundreds of custom machines. Deployment of software into production should have been automated using a continuous integration/continuous deployment (CI/CD) system. If the only way to put things in production is by automated deployment, then repeatability can be more easily achieved. There should have been separate development and production environments, with fewer access restrictions in the development environment. More attention to detail at the requirements-gathering stage might have foreseen the need for developer access, which would have led to the requirement for and funding of a development environment.

13.6 Cloud-Based Compute Services

Another strategy is to not own any machines at all, but rather to rent capacity on someone else's system. Such cloud-based computing lets you benefit from the economies of scale that large warehouse-size datacenters can provide, without the expense or expertise to run them. Examples of cloud-based compute services include Amazon AWS, Microsoft Azure, and Google Compute Engine.

13.6.1 What Is the Cloud?

Cloud computing is a marketing term fraught with confusing interpretations. Marketing people will gladly tell you that the cloud solves all problems, as if there are no problems left in the world now that the cloud is here.
There are three common definitions for **the cloud**, each coming from different communities:

- **Consumers:** When a typical consumer uses the term the cloud, they mean putting their data on a web-based platform. The primary benefit is that this data becomes accessible from anywhere. For example, consumers might have all their music stored in the cloud; as a result their music can be played on any device that has Internet access.
- **Business people:** Typically business people think of the cloud as some kind of rented computing infrastructure that is elastic. That is, they can allocate one or thousands of machines; use them for a day, week, or year; and give them back when they are done. They like the fact that this infrastructure is a payas-you-go and on-demand system. The on-demand nature is the most exciting because they won't have to deal with IT departments that could take months to deliver a single new machine, or simply reject their request. Now with a credit card, they have a partner that always says yes.
- **IT professionals:** When all the hype is removed (and there is a lot of hype), cloud computing comes down to someone else maintaining hardware and networks so that customers can focus on higher-level abstractions such as the operating system and applications. It requires software that is built differently and new operational methods. IT professionals shift from being the experts in how to install and set up computers to being the experts who understand the full stack and become valued for their architectural expertise, especially regarding how the underlying infrastructure affects performance and reliability of the application, and how to improve both.

This book uses a combination of the last two definitions. The on-demand nature of the cloud benefits us by providing an elastic computing environment: the ability to rapidly and dynamically grow and shrink. As IT professionals, it changes our role to focus on performance and reliability, and be the architecture experts.

13.6.2 Cloud Computing's Cost Benefits

Over the years computer hardware has grown less expensive due to Moore's law and other factors. Conversely, the operation of computers has become increasingly expensive. The increased operational cost diminishes and often eliminates the cost reductions.

The one place where operational cost has been going down instead of up is in large grids or clusters. There the entire infrastructure stack of hardware, software, and operations can be vertically integrated to achieve exceptional cost savings at scale.

Another way to think about this issue is that of all the previously discussed strategies, the "buy in bulk, allocate fractions" strategy described earlier is generally the most economical and proves the most flexible. Cloud-based compute services take that strategy to a larger scale than most companies can achieve on their own, which enables these smaller companies take advantage of these economics.

As these cost trends continue, it will become difficult for companies to justify maintaining their own computers. Cloud computing will go from the exception that few companies are able to take advantage of, to the default. Companies that must maintain their own hardware will be an exception and will be at an economic disadvantage.

Adoption of cloud computing is also driven by another cost: opportunity cost. Opportunity cost is the revenue lost due to missed opportunities. If a company sees an opportunity but the competition beats them to it, that could be millions of potential dollars lost.

Some companies miss opportunities because they cannot hire people fast enough to staff a project. Other companies do not even attempt to go after certain opportunities because they know they will not be able to ramp up the IT resources fast enough to address the opportunity. Companies have missed multi-million-dollar opportunities because an IT department spent an extra month to negotiate a slightly lower price from the vendor.

If cloud computing enables a company to spin up new machines in minutes, without any operations staff, the ability to address opportunities is greatly enhanced. At many companies it takes months, and sometimes an entire year, from the initial request to realizing a working server in production. The actual installation of the server may be a few hours, but it is surrounded by months of budget approvals, bureaucratic approvals, security reviews, and IT managers who think it is their job to always say "no."

We are optimistic about cloud computing because it enables new applications that just can't be done any other way. Kevin McEntee, vice-president at Netflix, put it succinctly: "You can't put a price on nimble." In his 2012 talk at re:Invent, McEntee gave examples of how the elastic ability of cloud computing enabled his company to process video in new ways. He extolled the value of elastic computing's ability to let Netflix jump on opportunities that enabled it to be a part of the iPad launch and the Apple TV launch, and to quickly launch new, large libraries of videos to the public. Netflix has been able to make business deals that its competition couldn't. "If we were still building out datacenters in the old model we would not have been able to jump on those opportunities" (McEntee 2012).

There are legal and technical challenges to putting your data on other people's hardware. That said, do not assume that HIPAA compliance and similar requirements automatically disqualify you from using cloud-based services. Cloud vendors have a variety of compliant options and ways of managing risk. A teleconference between your legal compliance department and the provider's sales team may lead to surprising results.

It is also possible to have an in-house cloud environment. With this approach, a department runs an in-house cloud and bills departments for use. This is common in industries that must meet strict HIPAA compliance requirements but want the economic benefits of massive scale.

Volume 2 of this book series, especially Chapter 3, "Selecting a Service Platform," contains advice about selecting and managing a cloud platform.

13.6.3 Software as a Service

Software as a service (SaaS) means using web-based applications. Many small companies have no servers at all. They are able to meet all their application needs with web-based offerings. For example, they host their web site using a hosting service, they use a web-based payroll provider, they share files using Dropbox, they use Salesforce for customer relationship management and sales process management, and they use Google Apps for Work for word processing, spreadsheets, and presentations. If they use Chromebooks, iPads, or other fixed-configuration web-browsing devices, they don't need traditional computers and can eliminate a lot of the traditional enterprise IT infrastructure that most companies require.

This strategy was impossible until the early 2010s. Since then, ubiquitous fast Internet connections, HTML5's ability to create interactive applications, and better security features have made this possible.

When a company adopts such a strategy, the role of the IT department becomes that of an IT coordinator and integrator. Rather than running clients and services, someone is needed to coordinate vendor relationships, introduce new products into the company, provide training, and be the first stop for support before the provider is contacted directly. Technical work becomes focused on high-level roles such as software development for integrating the tools, plus low-level roles such as device support and repair management.

13.7 Server Appliances

An **appliance** is a device designed specifically for a particular task. Toasters make toast. Blenders blend. One could do these things using general-purpose devices, but there are benefits to using a device designed to do one task very well.

The computer world also has appliances: file server appliances, web server appliances, email appliances, DNS/DHCP appliances, and so on. The first appliance was the dedicated network router. Some scoffed, "Who would spend all that money on a device that just sits there and pushes packets when we can easily add extra interfaces to our VAX and do the same thing?" It turned out that quite a lot of people would. It became obvious that a box dedicated to doing a single task, and doing it well, was in many cases more valuable than a general-purpose computer that could do many tasks. And, heck, it also meant that you could reboot the VAX without taking down the network for everyone else.

A server appliance brings years of experience together in one box. Architecting a server is difficult. The physical hardware for a server has all the requirements listed earlier in this chapter, as well as the system engineering and performance tuning that only a highly experienced expert can do. The software required to provide a service often involves assembling various packages, gluing them together, and providing a single, unified administration system for it all. It's a lot of work! Appliances do all this for you right out of the box.

Although a senior SA can engineer a system dedicated to file service or email out of a general-purpose server, purchasing an appliance can free the SA to focus on other tasks. Every appliance purchased results in one less system to engineer from scratch, plus access to vendor support in case of an outage. Appliances also let organizations without that particular expertise gain access to well-designed systems.

The other benefit of appliances is that they often have features that can't be found elsewhere. Competition drives the vendors to add new features, increase performance, and improve reliability. For example, NetApp Filers have tunable file system snapshots that allow end users to "cd back in time," thus eliminating many requests for file restores.

13.8 Hybrid Strategies

Most organizations actually employ a number of different strategies. Typically a small organization has a few snowflakes and possibly a few eggs in one basket. Medium-size organizations usually have a virtualization cluster plus a few snowflakes for situations where virtualization would not work. Large organizations have a little of everything.

Use what is most appropriate. Small companies often can't justify a grid. For large companies it is best to build a private, in-house cloud. If a company needs to get started quickly, cloud computing permits it to spin up new machines without the delay of specifying, purchasing, and installing machines.

Often one platform is selected as the default and exceptions require approval. Many IT departments provide VMs as the default, and bare-metal requests require proof that the application is incompatible with the virtualization system. Some companies have a "cloud first" or "cloud only" strategy.

Depending on the company and its culture, different defaults may exist. The default should be what's most efficient, not what's least expensive.

The only strategy we recommend against is an organization trying to deploy all of these strategies simultaneously. It is not possible to be good at all strategies. Pick a few, or one, and get very good at it. This will be better than providing mediocre service because you are spread too thin.

13.9 Summary

Servers are the hardware used to provide services, such as file service, mail service, applications, and so on. There are three general strategies to manage servers.

All eggs in one basket has one machine that is used for many purposes, such as a departmental server that provides DNS, email, web, and file services. This puts many critical services in one place, which is risky.

With beautiful snowflakes, there are many machines, each configured differently. Each machine is configured exactly as needed for its purpose, which sounds optimal but is

a management burden. It becomes important to manage variations, reducing the number of types of things that are managed. We can do this many ways, including adopting a policy of eliminating one generation of products before adopting a new one. Buy in bulk, allocate fractions uses virtualization or containers to achieve economies of scale in hardware management but provides a variety of machine sizes to users. Virtualization provides the ability to migrate VMs between physical machines, packing them for efficiency or to move VMs away from failing hardware. Packing VMs must be done in a way that maintains $N + 1$ redundancy.

There are several related strategies. Grid computing provides hundreds or thousands of machines managed as one large computer for large computational tasks. Blade servers make hardware operations more efficient by providing individual units of computer power or storage in a special form factor. Cloud-based computing rents time on other people's server farms, enabling one to acquire additional compute resources dynamically. Software as a service (SaaS) eliminates the need for infrastructure by relying on web-based applications. Server appliances eliminate the need for local engineering knowledge by providing premade, preconfigured hardware solutions.

Organizations use a mixture of these strategies. Most have one primary strategy and use the others for specific instances. For example, it is very common to see a company use virtualization as the default, physical machines as the exception, and SaaS for designated applications.

Chapter 14. Server Hardware Features

Server hardware is designed with a bias toward performance and remote manageability. It doesn't earn the "server" designation just because it can be mounted in a rack. Server hardware differs from workstation hardware both in form factor and in the options available at time of purchase. It is also different in how we configure it and manage it.

The previous chapter discussed several strategies for providing server capacity to our customers. Most of those strategies involved buying and maintaining hardware rather than using cloud-based services. This chapter is about high-level issues related to server hardware. The next chapter will cover more low-level technical issues.

Server hardware generally falls into one of two major categories: **enterprise** and **cluster**.

Enterprise server hardware prioritizes the needs of commercial applications that enterprises tend to use: business applications, email servers, file servers, and so on. These applications tend to assume data integrity and resiliency issues are handled at the hardware layer. This is a polite way of saying they assume these responsibilities are someone else's problem. As a result, these models emphasize reliability, high availability, and performance. Their marketing emphasizes reliability worth paying extra for.

Cluster server hardware prioritizes the needs of cluster or distributed computing applications such as Hadoop, Cassandra, web server farms, and so on. These applications work around hardware failures through software-based resiliency techniques such as service replication with automatic failover. They don't require expensive hardware reliability options such as RAID cards. These models are stripped down to the bare essentials: fast CPUs, network connections, and RAM. Their marketing emphasizes their excellent total cost of ownership: initial purchase price plus lifetime power, cooling, and other operational costs.

This chapter focuses primarily on enterprise servers. Cluster management is the focus of Volume 2 of this book series, especially Chapter 3, "Selecting a Service Platform."

14.1 Workstations Versus Servers

Workstations and servers are fundamentally different. The differences stem from the ways that each is used, which gives us a different set of requirements for each. For example, servers have higher uptime requirements than workstations, because many people rely on them. Servers have higher data integrity requirements than workstations, because core enterprise data is stored and processed there. Servers have higher CPU and memory requirements than workstations, because they are used for complex data processing, or to provide a service for many people.

As a result, server hardware is different from workstation hardware. Server operating systems are different from workstation operating systems. Also, we manage servers

differently, from patching to configuration. We will examine those differences in more detail.

14.1.1 Server Hardware Design Differences

Server hardware is designed with different priorities than workstation hardware. Workstations are designed for an individual user to sit in front of, and interact with directly. They need a display and user input devices, such as a keyboard, mouse, touchscreen, and microphone. Servers are designed to be housed in datacenters and accessed remotely by many users, usually indirectly through the services that they supply. Some of the characteristics that differ for servers versus workstations are profiled here:

- **More CPU performance**: Servers tend to have faster CPUs and more of them. CPUs may be available in a variety of speeds. Typically the fastest revision of a CPU is disproportionately more expensive—this is a surcharge for being on the cutting edge. Such an extra cost can be more easily justified on a server that is supporting hundreds of customers, or is hosting many virtual machines, or is expected to last many years.

- **High-performance I/O:** Servers tend to have more I/O capacity. I/O requirements are often proportional to the number of clients being served. This necessitates faster I/O subsystems. Faster I/O may be achieved by using the latest, fastest technologies that aren't available elsewhere yet. Connections to storage may be multiple dedicated paths, one for each disk, rather than a shared channel. The path between devices is highly optimized so that there are no bottlenecks even at full utilization. This requires special care when designing the system and choosing tradeoffs that favor performance over cost.

- **Expandability**: Servers usually have more physical space inside for hard drives and more slots for cards and CPUs. Adding more slots and the connecting circuitry is expensive. Therefore consumer hardware manufacturers tend to eliminate them to lower the initial purchase price, whereas they are easily justified on server hardware.

- **Upgrade options**: Servers are designed for growth. They generally have the ability to add CPUs or replace individual CPUs with faster ones. Typically server CPUs reside on separate cards within the chassis, or are placed in removable sockets on the system board for ease of replacement.

- **Rack mountable**: Generic servers and server appliances should be rack-mountable. Servers should be rack-mounted, not stacked or put on shelves. Whereas desktop hardware may have a pretty, molded plastic case in the shape of a gumdrop, a server should be rectangular and designed for efficient space utilization in a rack. Any covers that need to be removed to do repairs should be removable while the host is still rack-mounted. More importantly, the server should be engineered for cooling and ventilation in a rack-mounted setting. A system that has only side cooling vents will not maintain its temperature as well in a rack as one that vents front to back. Having the word

server included in a product name is not sufficient; care must be taken to make sure that it fits in the space allocated.

- **Front and rear access**: A rack-mounted host is easier to repair or perform maintenance on if tasks can be done while it remains in the rack. Such tasks must be performed without access to the sides of the machine. All cables should be on the back, and all drive bays should be on the front. Some DVDROM bays open on the side, indicating that the host wasn't designed with racks in mind. Power switches should be accessible but not easy to accidentally bump. Some systems, often network equipment, require only front access. This means that the device can be placed "butt in" in a cramped closet and still be serviceable. Some hosts require that the external plastic case be removed to successfully mount the device in a standard rack. Be sure to verify that this does not interfere with cooling or functionality.

- **High-availability options**: Server hardware should include various high-availability options, such as dual power supplies, RAID, multiple network connections, and hot-swap components. RAM should be error correcting, not just error checking.

- **Remote management:** Server hardware should be capable of being remotely managed. For Unix servers and network equipment, this means serial port consoles. For modern PC hardware, it means out-of-band (OOB) management, such as Integrated Lights-Out (iLO) or Intelligent Platform Management Interface (IPMI) for remote console, power, and baseboard management. In the past IP-KVM and remote power control systems added remote control capabilities to a system after the fact, but it is less complex and costly to have such facilities integrated into the server themselves. Integrated temperature sensors and other hardware monitoring features are also standard on server hardware today.

Don't Run Services on Desktop Hardware

It is tempting to purchase desktop hardware for use as servers. Doing so is short-sighted.

The initial purchase price is often lower, but the longer-term costs will be higher. What is the cost of lost productivity due to slow performance? What is the cost of a multi-hour outage due to a failed hard drive on a system without RAID?

Desktop hardware is not as reliable. It tends to be sold with consumer-grade hard drives and RAM, and lacks reliability enhancements such as ECC RAM or RAID storage. It usually has limited I/O capacity, leading to unsatisfactory performance. It does not mount in a rack, which seems fine at first, but after a few machines have accumulated becomes more difficult to manage. While one can purchase shelves and other devices to make desktop towers fit in a rack, they do not pack as efficiently as rackable server hardware.

Desktop hardware does not include remote management facilities, which means problem diagnostics and anything requiring console access will require physically

visiting the machine. Walking to the machine delays when such technical work starts and takes you away from your desk—neither of which is an efficient way of working.

14.1.2 Server OS and Management Differences

How we configure servers is also different. Servers use a different OS and their configurations are managed differently.

Server hardware runs a server OS. Microsoft Windows Server Edition includes additional software for providing services such as ActiveDirectory. It also is tuned for server operation instead of interactive performance: Various defaults are changed to provide better performance for long-running applications. Workstation editions are tuned with a bias for best interactive performance.

In the Linux world, and more recently with Windows Server Core, the server edition is stripped down to the bare essentials, eliminating even the windowing and graphic subsystems. Functionality is added as needed by installing the appropriate package.

A system is more reliable and secure when it is smaller. Not having a GUI greatly reduces the memory footprint and not activating any video drivers avoids a major source of bugs and crashes. Debugging is easier because what hasn't been installed can't get in the way. Maintenance is easier because there are fewer software packages to be upgraded. Security is improved because a package that hasn't been installed is one fewer source of vulnerabilities.

Often companies have a policy of installing security patches within a certain number of days after the vendor has released the patch. For many servers, this means a constant flow of upgrades and reboots to patch software that isn't even used. By being disciplined and not installing unnecessary software, patch and reboot cycles will be reduced.

Server OSs are often patched on a different schedule. While workstations are updated frequently, often with user approval, servers are patched on a schedule that matches the requirements of the application. That might mean weekly during off-hours, or monthly during carefully announced maintenance windows.

The configuration of servers is also different. Configuration management systems are used to configure and update the system, as was discussed in Chapter 4, "Infrastructure as Code." Network configuration is generally done by hardcoding the configuration, as described in Section 5.4, though DHCP INFORM is often used to automatically update network settings.

Top Five Server Configuration Mistakes

A number of common configuration mistakes may occur when people forget to clearly differentiate between workstations and servers.

1. Using the workstation edition of the OS, because it happens to be what was installed

2. Using desktop hardware for a server, because it was “free”

3. Using someone’s desktop for a shared function, because they’ve promised never to power off their machine

4. Running on a machine under someone’s desk, because if the service is successful, we’ll move it to the computer room later

5. Forgetting to label the machine or to add it to the inventory system

14.2 Server Reliability

All devices fail. We need to accept that fact, and prepare for failure. Because servers need to be more reliable, and have higher uptime than workstations, one of the ways we prepare for equipment failure is to buy server hardware with additional features for reliability and data integrity. When evaluating server hardware, it is important to understand these features, and to know which questions to pose to the vendors. In addition, servers should be housed in a restricted-access, climate-controlled environment, with protected power—in other words, a computer room or datacenter.

14.2.1 Levels of Redundancy

Servers often have internal redundancy such that one part can fail and the system keeps running. For example, there may be two power supplies in the system. Either can fail and the system keeps running. The term N + 1 redundancy is used when we wish to indicate that there is enough spare capacity for one failure. N + 2 redundancy would mean there is enough spare capacity for two failed power supplies.

Having two power supplies does not automatically create N + 1 redundancy. What if the system is loaded with enough power-hungry cards and components that two power supplies are required just to provide enough watts to run the system? It is possible for this system to be N + 0 (no spare capacity) even though there are two power supplies. In such a case, a third power supply would be required to be N + 1.

Just because the power supplies are N + 1 redundant, it doesn’t mean the system can’t fail. What if the CPU dies or a hard disk fails? For this reason, CPUs, storage systems, and other components may have N + 1 redundancy options, too.

When a salesperson says that a product has no single point of failure, or that the system is N + 1 redundant, a good question to ask is “Which parts aren’t N + 1 redundant?” Every system has some failure point. That’s just physics. It is your job to know what it is. That’s system administration.

14.2.2 Data Integrity

Another aspect of server reliability is data integrity. How do we know if the data stored on the machine will be available and valid in the future? Hard disks and SSDs eventually wear out and die. In turn, we must have a plan to deal with this eventuality. Not having a plan is as irrational as assuming our storage systems will last forever.

RAID

Enterprise applications are typically written assuming that storage is perfect—that it is always available and never fails. While this is an irrational assumption, we can make a reasonable simulation of this requirement by using RAID (Redundant Array of Independent Disks) levels 1 and higher. When a disk fails, the system keeps running and we have time to replace the disk.

- RAID 1 stores all data twice, once on each disk of a two-disk mirror. If one disk fails, the system can simply rely on the remaining disk. The failed disk is replaced and the data from the surviving disk is mirrored to the new disk, bringing the system back to full resiliency. As long as this happens before the remaining disk also fails, we have a system of constantly available storage. If the second disk fails before restoration is complete, we lose all data and must restore data from backups. However, the probability of a two-disk failure is small if we act quickly.

- RAID levels 2 and higher are similar but create redundancy in ways that are more efficient, have better performance, or can survive two disk failures.

- RAID 0 is unlike other RAID levels in that it does not aid data integrity. It is actually less resilient to failure, but provides better performance. For the purpose of this chapter, unless specified otherwise, when we refer to RAID we mean levels 1 and higher.

Chapters 43, “Data Storage,” discusses the RAID levels in detail, including how to select which RAID level to use, and the performance implications of each. That said, for a general enterprise server a two-disk RAID 1 mirror is the minimum for a small server, and RAID 5 or 6 is the minimum for a server with a handful of disks.

The brilliance of RAID is that it decouples component failure from service failure. Before RAID, if a disk died, the service died: Dead disk equals outage. You would spend the day replacing the failed disk, restoring data from backups, and apologizing to your customers. It could take hours or days to bring the service back. With RAID 1 and higher, the system can survive a single disk failure. Now a dead disk equals potentially lower performance, but the system keeps running. People would rather have a slow system than no system. With RAID, customers are often unaware that anything has failed.

Non-RAID Approaches

The alternative to RAID is to assume data integrity is handled elsewhere. Plain disks are used with no RAID protection and failures are handled at another layer of the design. For example, suppose a group of redundant web servers all contain the same data, which they receive from a master. The web servers themselves do not need RAID because if a disk fails, the web server is simply shut down for repairs and the

traffic is divided among the remaining web servers. After the server hardware is repaired, the data is recopied from the master.

Another example of handling data integrity elsewhere is a distributed storage system such as Google's GFS, Hadoop's HDFS, Cassandra, and others. These systems store copies of data on many hosts, essentially creating RAID-like storage that is distributed among many machines, not just many disks.

A strategy you should avoid is to rely on backups. With this strategy, you copy all data to tape or disk periodically and, if a disk fails, you take the service down while the data is restored from the backup copy. This strategy includes a built-in outage, even if you could restore the data very fast. Backups are important and needed for other reasons—for example, to keep a copy off-site in case the building burns down. RAID is not a backup strategy. RAID and backups are both needed; they are complimentary. Backups will be discussed in greater detail in Chapter 44, "Backup and Restore."

14.2.3 Hot-Swap Components

Server hardware has many redundant components, and these components should be hot-swappable. Hot-swap refers to the ability to add, remove, and replace a component while the system is running.

Normally, parts should be removed and replaced only when the system is powered off. Being able to hot-swap components is like being able to change a tire while the car is driving down a highway. It's great not to have to stop a service to fix components that fail frequently, such as fans, disks, and power supplies.

Hot-swappable components increase the cost of a system. This additional cost is justified when it eliminates downtimes for expansion or repairs.

The more quickly a failed component can be replaced, the better. RAID systems usually run more slowly until a failed component has been replaced and the RAID set has been rebuilt. More importantly, while the system is not fully redundant, you are at risk of a second disk failing and losing all data.

When a vendor makes a claim of hot-swappability, always ask two questions: Which parts aren't hot-swappable? In what way, and for how long, is service interrupted when the parts are being hot-swapped? Some network devices have hot-swappable interface cards, but the CPU is not hot-swappable. Others claim hot-swap capability but do a full system reset after any device is added. This reset can take seconds or minutes. Some disk subsystems must pause the I/O system for as long as 20 seconds when a drive is replaced. Others run with seriously degraded performance for many hours while the data is rebuilt onto the replacement disk. Be sure that you understand the ramifications of component failure. Don't assume that hot-swap parts make outages disappear: They simply reduce the outage.

Vendors should—but often don't—label components as to whether they are hot-swappable. If the vendor doesn't provide labels, you should create your own.

Hot-Plug Versus Hot-Swap

Be mindful of components that are labeled hot-plug. This means that it is electrically safe for the part to be replaced while the system is running, but the part may not be recognized until the next reboot. Even worse, perhaps the part can be plugged in while the system is running, but the system will immediately reboot to recognize the part. This is very different from being hot-swappable.

Tom once created a major, but short-lived, outage when he plugged a new 24-port FastEthernet card into a network chassis. He had been told that the cards were hot-pluggable and had assumed that the vendor meant the same thing as hot-swap. Once the board was plugged in, the entire system reset. This was the core switch for his server room and most of the networks in his division. Ouch!

You can imagine the heated exchange when Tom called the vendor to complain. The vendor countered that if the installer had to power off the unit, plug the card in, and then turn power back on, the outage would be significantly longer. Hot-plug was an improvement.

To prevent problems in the future, Tom put a big sign on the device that said, “Warning: Plugging in new cards reboots system. Vendor thinks this is a good thing.”

14.2.4 Servers Should Be in Computer Rooms

Servers should be installed in an environment with proper power, fire protection, networking, temperature and humidity control, and physical security. That means a computer room or datacenter—not a spare cubicle or closet.

The two most important points are power and cooling. For reliable operations, servers need power that is reliable and clean. Servers are designed to run at a particular operating temperature, usually around 10 to 35°C (50 to 95°F). Cooling is required to remove the heat they generate.

A rack of equipment standing in the middle of an office is a disaster waiting to happen. It is sharing a circuit not designed for the draw of a rack of power-hungry servers. The cooling in an office generally turns off on nights and weekends. It is not physically secure, and can be accidentally or maliciously interfered with.

It is a good idea to reserve the physical space for a server when it is being purchased. This is a safeguard against having the space double-booked. One can reserve the space using a high-tech datacenter inventory system, or a simple spreadsheet. Taping a paper sign in the appropriate rack is low-tech but sufficient.

After assembling the hardware, it is best to mount it in the rack immediately before installing the OS and other software. Consider the following phenomenon: A new server is assembled in someone’s office and the OS and applications loaded onto it. As the applications are brought up, some trial users are made aware of the service. Soon the server is in heavy use before it is intended to be, and it is still in someone’s office without the proper protections of a machine room. Now the people using the

server will be disturbed by the downtime required when it is moved into the machine room. The way to prevent this situation is to mount the server in its final location as soon as it is assembled. It also reduces the risk that you may lose the rack-mounting hardware. An even worse outcome is to move the machine into the server room, discover you don't have a network cable of sufficient length, and have to move it back. Scheduling a second downtime is embarrassingly unprofessional.

Field offices aren't always large enough to have machine rooms, and some entire companies aren't large enough to have datacenters. However, field offices should at least have a designated room or closet with the bare minimum capabilities: physical security, UPS (many small ones if not one large one), and proper cooling. A telecom closet with good cooling and a door that can be locked is better than having your company's payroll installed on a server sitting under someone's desk. Inexpensive cooling solutions, some of which eliminate the need for drainage by reevaporating any water they collect and exhausting it out the exhaust air vent, are now commonplace.

14.3 Remotely Managing Servers

Servers need to be maintained remotely. Remote management means that it should be possible to do all system administration tasks involving the machine from a remote location, except physical labor such as adding and removing physical hardware. It should be possible to remotely access the machine's console and, optionally, have remote control of the power switch.

Remote management systems have the benefit of permitting you to operate a system's console when the machine is in a bad state or otherwise disconnected from the network. For example, certain things can be done only while a machine is booting, such as pressing a key sequence to activate a basic BIOS configuration menu. Remote console systems also let you simulate the funny key sequences that have special significance when typed at the console—for example, CTRL-ALT-DEL on PC hardware and L1-A on Sun hardware.

Remote access is important because servers are often housed in machine rooms located around the world. Requiring hands-on access for daily tasks is inefficient. You shouldn't have to fly to Europe every time you want to see console diagnostics, reboot the system, or reload the OS. Even if the server is down the hall in a server closet, having remote access eliminates time-wasting visits to the machine, and enables you to manage the server while out of the office or from home.

Security implications must be considered when you have a remote console. Often, host security strategies depend on the consoles being behind a locked door. Remote access breaks this strategy. Therefore, console systems should have properly considered authentication and privacy systems. For example, you might permit access to the console system only via an encrypted channel, such as SSH, and insist on authentication by a one-time password system, such as a handheld authenticator.

14.3.1 Integrated Out-of-Band Management

Modern servers have remote management capabilities built-in. Such systems are generically called out-of-band (OOB) management, and have an Ethernet port as an interface. Other terms you may hear (some of which are proprietary technologies) are lights-out management (LOM), Integrated Lights-Out (iLO), Intelligent Platform Management Interface (IPMI), remote insight board (RIB), or Remote Insight Light-Out Edition (RILOE).

Once an IP address is assigned to the OOB interface, one can connect over the network to a remote management subsystem that provides access to the console. OOB solutions usually just require a browser to access the console. However, some require a custom client for access to the full capabilities, so it is worth checking this point before buying a system.

These remote management systems are isolated enough from the main system that they are accessible even when the main system is down or powered off. This feature is often the differentiator between a vendor's server-class machines and other offerings.

Bear in mind that many of these built-in systems have been found to suffer from traditional security holes, such as buffer overruns. It is not wise to assume that setting a password on the OOB interface and using SSL is sufficient protection for such access to your servers. Assume that access to the OOB interface is equivalent to physical access to the machine, and establish additional security measures accordingly. For example, put OOB interfaces on a dedicated protected network. Allow access to this network only via a web proxy with certificate-based authentication, and appropriate authorization.

14.3.2 Non-integrated Out-of-Band Management

Third-party products can add some OOB functionality to systems that lack built-in remote management. These products typically focus on either remote power cycles or remote console access.

Remote Power Cycle

One can gain the ability to remotely power a machine off and on using a switched power distribution unit (PDU). A PDU is the fancy name for a power strip. A switched PDU can individually turn on or off each power receptacle. For example, if the machine has crashed in a way that can be resolved only by turning the power off and back on, one remotely connects to the switched PDU and issues the command to cycle the power of its jack.

Remote Console with IP-KVM

One can gain the ability to remotely access the console of a machine using an IPKVM switch. A KVM switch is a device that lets many machines share a single keyboard, video screen, and mouse (KVM). For example, you might be able to fit three servers and three consoles into a single rack. With a KVM switch, you need only a single keyboard, monitor, and mouse for the rack. Now more servers can fit in that rack. You

can save even more room by having one KVM switch per row of racks or one for the entire datacenter.

An IP-KVM is a KVM switch that can be accessed remotely. This eliminates the need for any monitors, keyboards, or mice in the computer room. You simply run a client on your workstation that connects to the IP-KVM. As you type and move your mouse, the client emulates these actions on the machine via the KVM mechanism.

Remote Console with Serial Consoles

Network equipment, appliances, and older servers have serial consoles. They have no video, keyboard, or mouse connector. In the old days one would attach a physical VT-100 terminal to the serial console port of each device.

What Is a VT-100 Terminal?

Younger readers may think of a VT-100 terminal only as a software package that interprets ASCII codes to display text, or a feature of a telnet or ssh package. Those software packages are emulating actual devices that used to cost hundreds of dollars each. One would be purchased for each server to be its console. Before PCs had VT-100 emulators, a mainframe would have dozens or hundreds of serial ports, each one connected to a physical terminal.

Having a large VT-100 terminal for each server would consume a lot of space in the machine room. Often there would be a long table with dozens of terminals. In the 1990s it became popular to replace physical terminals with a terminal console concentrator. This device is one box with many serial ports. The serial cables that previously connected to a physical VT-100 terminal instead plug into one of the ports on this box. If you need to connect to the serial console of a device, you SSH or TELNET to a particular port on the concentrator, and it connects you to the console of the server.

For sites with many servers, network equipment, and appliances that are connected this way, it is common to have console server software, such as Conserver (http://www.conserver.com), to manage the serial consoles. This software gives SAs a way to connect to a console by name, rather than having to remember which server is attached to which concentrator on which port. It also stays connected to the serial consoles, preventing other access and logging all console output, giving SAs the ability to review the history of what happened to a machine. SAs connect and authenticate to the console server software, allowing fine-grained control of who is authorized to connect to which consoles. Also, serial console concentrators usually have the option to require authentication—for example, using RADIUS or TACACS—before allowing someone to connect to a console. However, this access control tends to permit access to all consoles or to none, rather than giving each SA access only to the consoles he or she needs.

Remote console access is discussed further in Section 26.5.

Monitor All Serial Ports

Once Tom noticed that an unlabeled and supposedly unused port on a device looked like a serial port. The device was from a new company and Tom was one of its first beta customers. He connected the mystery serial port to his console and occasionally saw status messages being output.

Months went by before the device started having a problem. He noticed that when the problem happened, a strange message appeared on the console. This was the company's secret debugging system! When he reported the problem to the vendor, he included a cut-and-paste of the message he was receiving on the serial port. The company responded, "Hey! You aren't supposed to connect to that port!" Later, the company's employees admitted that the message had indeed helped them to debug the problem.

After this, Tom adopted a policy of always monitoring all serial ports.

14.4 Separate Administrative Networks

It is common for servers to have a separate NIC that is connected to a dedicated administrative network. Separating administrative traffic from normal service traffic has a number of benefits.

For servers, the primary benefit is often to isolate disruptive traffic. Backups, for example, consume a large amount of bandwidth and can interfere with normal service traffic. Sometimes servers will also have a dedicated backup interface that is connected to a high-speed network dedicated to backups.

In addition, the administrative network is often more stable and static, while the service network is more dynamic. For example, the service NIC of a server might reconfigure itself frequently as part of a load-balancing and failover mechanism. Monitoring and control of such mechanisms therefore is often done via the administrative NIC of a server so that it doesn't step on its own tail.

The administrative network often has more restrictive firewall policies than the service network, since it has more critical systems. The administrative network is often where one connects other administrative devices and control systems. For example, UPSs, PDUs, IP-KVMs, temperature and environmental monitors, and other devices now connect to the network via Ethernet for remote access, control, and monitoring. The administrative network is also where a server's iLO/IPMI interfaces connect.

This separate network should be engineered to use separate equipment so that it will not be affected by outages in the main network. Often this network is engineered very simply, with old-fashioned managed switches, no VLANs, and a simple topology. This network then provides a way for SAs to get to machines and network devices during a network outage.

14.5 Maintenance Contracts and Spare Parts

When purchasing a server, consider how repairs will be handled. All machines eventually break. Expecting them not to break is irrational. To be rational we must plan for the eventual hardware failure.

A maintenance contract is an agreement with a vendor that it will fix or replace hardware and provide other support for a certain duration of time. Maintenance contracts are like an insurance policy; you pay and hope you don't need it, and so does the vendor.

14.5.1 Vendor SLA

Vendors tend to have a variety of maintenance contract options, with different service level agreements (SLAs). For example, maintenance contracts may offer to ship a replacement for a bad part with a 4-hour response time, a 12-hour response time, or next-day options. Sometimes the options include 24/7 response, but other times just specify some number of business days. Other options include having the customer purchase a kit of spare parts and receive replacements after a spare part gets used. Usually you have to install the replacement part, though vendors offer on-site repairs for a higher price. Sometimes the service offerings vary by country or region, depending on where the vendor has local presence.

Response time is often tricky to calculate. While you might think a 4-hour response time contract means that a hard drive that fails at 7 AM will be replaced by 11 AM, the truth is quite different. For most vendors the clock starts ticking not when the part fails, but when the vendor has confirmed that the part is dead. Suppose a machine fails at 7 AM, and you don't notice it is down until you arrive at the office at 9 AM. You know it is down because you are greeted by an angry mob. It may then take 30 minutes to get through to tech support and convince them that, yes, the machine is on the contract. Running diagnostics may require an hour. It may be 11 AM before the vendor has agreed to send the replacement part. At that point you may learn that the spare part depot in your city has the part and a courier is on the way. Or, you may discover that the spare part is not available and must be shipped overnight from the factory. Obviously, this process can be improved: The outage should have been detected by the monitoring system, not by the angry mob. The point here, however, is that the definition of the 4-hour response time is defined by the contract, not by the clock on the wall.

The faster the response time, the more expensive the maintenance contract. Select the most appropriate option for the type of server. If a server is noncritical, the default maintenance contract may be sufficient. Replacement parts are ordered as needed under this arrangement, perhaps with rush shipping. It may require an open purchase order in place to expedite the ordering.

Ironically, a highly critical server can be on the lowest-level maintenance contract. If it is truly impossible to live without this server, it should be part of a mirrored pair that automatically fails over to a hot spare if there is trouble, or at least a cold spare should be sitting beside it ready to be activated. Therefore such a system does not require an expensive, 4-hour-response-time maintenance plan. A parts replacement plan may be sufficient.

For some applications, a critical server may be too big and expensive to have a replica. In this case, a 4-hour-response-time contract is warranted. Some vendors can guarantee that spares will be kept in a depot in the same city. The SLA for the service provided by the machine must include that for major repairs, downtime may be multiple hours.

Some applications can use inexpensive servers that are replicated many times—for example, a web farm of ten machines, each performing the same function. The system has enough spare capacity that any one machine can be down and the system will be fine. In this case, a next-day or two-day response time is reasonable because the likelihood that a second machine will die in a two-day period is small. Where this model can be applied, it is the most cost-effective.

14.5.2 Spare Parts

In a large environment, the cost of many maintenance contracts may be more expensive than staffing your own repair department. This in-house department can receive training and certification on the hardware and maintain its own inventory of spares.

Some vendors have a self-support plan where the customer purchases a spares kit and the maintenance contract is simply an expedited replacement plan for any spares that are consumed. This arrangement is typically used in high-end servers and appliances where the customer only has a few, critical units from that vendor. The spares kit is less expensive than buying a second machine, often called a cold spare because it is left powered off (cold) until needed. A hardware spares kit is less expensive because it does not require a software license, and because it has fewer parts. It does not include things that typically can't break, such as the chassis itself. It needs to include only one of any given part: If the original system has four CPUs, the kit needs to contain only one. Similarly, a single spares kit should be sharable among all machines it applies to. One needs to get additional spares kits only as new models are introduced into the fleet.

On-Site Spares and Staff

In the 1990s Bell Labs had so many machines from Sun Microsystems in its Murray Hill building that it negotiated to have the regional Sun maintenance person's office be in its building. Bell Labs provided the office space for free, and in exchange received many intangible benefits. Response time to the company's calls was faster since there was a good chance the Sun worker was already in the building. Since his spare time was spent at Bell Labs, he was often around to help out with issues outside of the contract, such as installing new machines. His office also became a regional spare parts depot. Common parts were in the building, reducing the time to repair.

14.5.3 Tracking Service Contracts

Sometimes during an outage we discover that the machine is not on the service contract. The solution usually involves talking to a salesperson, who will often have

the machine repaired on good faith that it will be added to the contract immediately or retroactively. There are a few different strategies for improving on this approach.

It is good practice to write purchase orders for service contracts for 10 percent more than the quoted price of the contract, so that the vendor can grow the monthly charges as new machines are added to the contract.

Put machines on the contract at the time of purchase. Make sure there is a choke-point that purchases pass through where someone makes sure that any purchase without an accompanying service contract is rejected. The person assigned this task might be someone in the purchasing department, or the salesperson whom the vendor has assigned to your account. Provide guidelines for helping the person select the appropriate service level, or have a default.

Often the first 12 months of service are built into the cost of the device, and one must remember to add the service contract after that time. Prearrange that the device will be added automatically at that time.

Some vendors require you to purchase your maintenance contract in advance. For example, they may offer service for three years and require all three years to be purchased when the machine is acquired. In this case, it is important to track when the contract expires so that an extension can be negotiated, the machine can be replaced, or the machine will be designated for use by noncritical services.

Lastly, track the service contracts as part of the machine inventory. This way you can run reports to find machines missing from the contract.

The other side of this potentially problematic process is forgetting to remove machines from the service contract when they are decommissioned. It is also good practice to review the service contract annually to ensure that new servers were added and retired servers were deleted. Strata once saved a client several times the cost of her consulting services by reviewing a vendor service contract that was several years out-of-date.

Again, tracking service contracts as part of inventory is important to make this happen.

14.5.4 Cross-Shipping

Something else that’s good to see in maintenance contracts is the ability to have replacement parts cross-shipped.

When a server has hardware problems and replacement parts are needed, some vendors require the old, broken part to be returned to them. This makes sense if the replacement is being done at no charge as part of a warranty or service contract. The returned part has value since it can be repaired and returned to service with the next customer that requires that part. Also, without such a return, a customer could simply be requesting part after part, possibly selling them for profit.

Some vendors will not ship the replacement part until they receive the broken part. This practice significantly increases the time to repair. Better vendors will ship the

replacement immediately and expect you to return the broken part within a certain amount of time. This practice is called cross-shipping; the parts cross paths as they are delivered.

Cross-shipping is generally included in a maintenance contract. If not, a credit card or open PO might be required. Be wary of vendors claiming to sell servers that don't offer cross-shipping under any circumstances. Such vendors aren't taking the term server very seriously.

14.6 Selecting Vendors with Server Experience

Some vendors have years of experience designing servers, and it shows. They build hardware with the server-related features listed earlier, as well as include little extras that one can learn only from years of market experience.

From time to time companies with a history of making consumer products jump into the server market and their foray is a disaster. They take their existing models and slap a "server" name onto it. The product does not integrate with enterprise systems such as network directory services. Management of the device is manual and cumbersome, lacking the ability to manage many servers at once, and with no kind of remote management. There are no maintenance contracts available other than what was offered with the vendor's consumer products. No on-site support or parts by mail means in the event of a hardware failure, the entire system will have to be packed up and carried to a local electronics store. Technical support is provided via phone to people accustomed to dealing with consumer-grade products and includes only consumer-grade responses. The support personnel can't handle highly complex technical issues, nor can they offer solutions beyond "wipe and reload"—an option that may be fine for a laptop, but is insufficient for fixing a business's primary server.

Select vendors that are known for building reliable hardware. Some vendors cut corners by using consumer-grade parts; others use premium-quality parts.

The difference between consumer- and premium-grade parts is often one of testing. RAM manufacturers do not have one assembly line that makes consumer-grade RAM chips, and another that makes premium-grade RAM chips. They have one factory that makes RAM chips. If a chip passes all the quality assurance tests, it is sold as a premium-grade product. If it passes most of the quality assurance tests, it is sold as a consumer-grade product at a lower price. If it doesn't pass enough tests, it is sold as scrap.

The MIL-SPEC Scam

Some vendors make a big deal out of the fact that they use MIL-SPEC parts. The implication is that these parts are higher quality. That is misleading.

MIL-SPEC is the U.S. military's requirements specifications for parts. MIL-SPEC does not require parts be of high quality; rather, it specifies the required quality and tolerances for deviation. In other words, what is on the label has to be true. What MIL-

SPEC really means is that the manufacturer is more certain that the item being purchased will be the quality expected, whether that is high or low quality.

That said, manufacturing is improved because the vendor can spend less time worrying about the variations in the parts being used, which allows it to focus on overall quality.

14.7 Summary

- This chapter discussed what makes server hardware different from workstation hardware. It is not just the fact that the hardware fits in a rack or has “server” in its name.

- Server hardware is engineered with a bias toward performance. Enterprise hardware tends to emphasize reliability through hardware, improving reliability so that applications do not need to concern themselves with handling hardware failures as much. Cluster hardware tends to emphasize reliability through software, reducing the cost by eschewing expensive RAID controllers and other hardware.

- Server hardware is different in that is tends to emphasize CPU and I/O performance. It is more expandable and upgradable. Maintenance is made easier by physical design, such as no side access required, and remote management features such as IPMI.

- Server hardware is configured differently. Operating systems have special server editions that add server features and remove client features.

- N + 1 redundancy means that any one part of a system can fail and the system keeps running, although performance may be degraded. This is a common feature of storage, power, and network designs.

- Servers should be housed in server rooms with proper cooling and power. They should be remotely managed, so that maintenance doesn’t require a walk to the machine room or a flight to a datacenter.

- Servers generally have maintenance contracts that include a defined response time and the ability to cross-ship spare parts.

Chapter 15. Server Hardware Specifications

This chapter is about the decisions one makes when purchasing an individual server. Designing hardware is all about making different tradeoffs. Exactly what vendors offer changes from month to month and from year to year. Nevertheless, certain principles remain constant.

Vendors have different product lines. Each is biased toward cost or performance in different ways. If we can understand what went into the designer's decisions, we can be smarter customers.

Within a product line there are many hardware choices: CPUs, RAM, storage, and so on. Fundamentally we should begin with the application requirements in mind and use those requirements to guide each decision, including the amount of RAM, the amount of storage, the number and types of CPUs, and so on. Even if we are adding general-purpose compute power to a virtualization cluster before we know which actual applications will be running, the requirements can be specified in general terms of what virtualization clusters need.

Bear in mind that with each different type of model you purchase, the support overhead becomes higher. For most companies, it is better to settle on a couple of standard server models than to try to optimize each server for the particular application that will be running on it. We do not want to end up with a blizzard of beautiful snowflakes.

Server performance is very complex, but in general an application is usually waiting for one of four things: disk, CPU, memory, or network. If we had perfect knowledge of our system, it would have just enough of all four without any wasted capacity. Unused capacity is money spent but not used. However, the reality is that we don't have perfect knowledge of our systems, and we can't purchase these resources in highly granular increments. Nor would we want to: Spare capacity is needed for the occasional bursts of activity, plus it offers room for future growth.

Instead, we usually generalize the application as being bound by one or two of the four limiting factors and purchase a server that is optimized in that aspect. Database software is often limited by the speed of disk I/O, so we start our purchasing decision-making process by looking at models with the fastest disk subsystem. One application may require large amounts of RAM, all other factors being inconsequential. Another application may be a CPU hog, churning on data for hours before spitting out a small result.

15.1 Models and Product Lines

Most vendors have multiple product lines. They are similar to the workstation product lines described in Section 6.1.3: one that emphasizes lowest initial cost or purchase price, one that emphasizes lowest total cost of ownership (TCO), and another that emphasizes high performance.

The line that emphasizes initial purchase price is often called the "value line" and trades expandability to achieve a lower initial purchase price. Like its workstation

analog, this line is appropriate only when saving money now is more important than long-term manageability. It might be appropriate for a small company with a few employees that does not expect to grow. For example, a small law practice might have one lawyer and one legal secretary—the same structure it has had for years and will have in the future.

The line that emphasizes TCO has models that may cost more initially, but save money in the long term. These machines have higher performance and additional expansion capabilities that prevent obsolescence. They also have management features that become more important when one has many machines, especially if they are remote. This line is appropriate for a company large enough to have dedicated IT staff who maintain the fleet as a unified whole.

The line that emphasizes performance provides access to the latest technologies, such as next-generation CPUs, or amounts of RAM that seem ludicrous by today's standard but will be normal expectations in a few years. Vendors also have specialty product lines for vertical markets, such as high-end graphics, numerically intensive computing, carrier-grade equipment (suitable for use in telecom environments), and so on. You pay a premium for early access to such technology. This is appropriate for demanding applications such as genomics, large database applications, numerical simulations, and other applications that require high performance.

15.2 Server Hardware Details

There are many hardware options available. While the specific sizes, quantities, and speeds might change over the years, some of the basic decisions stay the same. To make the best purchasing decision, you need to understand the technical details of each component, its limitations, and how the components interact with one another.

15.2.1 CPUs

The most fundamental decision in selecting a server is the CPU—in particular, the choice of a few high-speed cores versus many slow cores. Depending on the architecture of the software that will run on the machine, this decision can determine whether your service is fast and efficient or underutilizes the capacity of the machine.

In the old days, typical computers had a single CPU and we could count on vendors offering faster CPUs each year. Moore's law predicts that CPU performance will approximately double every 18 months, permitting either the same CPU to be purchased for half the price, or a CPU that is twice as fast to be purchased for the same price.

More recently, this trend has reached certain physical limits. With no other option available, CPU manufacturers started putting multiple CPU cores on the same chip. Instead of a CPU that is twice as fast, you can get a single CPU with two CPU **cores**, each able to run a program or execution thread in parallel.

In aggregate, the computing power has doubled, but any single-threaded program will run more slowly. There is a difference between being able to run one program twice as fast and being able to run two programs at the same time.

Multiple CPUs and Cores

Your purchasing choice is often between a few fast cores or many slower cores. For example, when this book was being written, a Dell R730 server could be outfitted with the CPU options listed in Table 15.1.

Table 15.1: Intel Xeon Processor E5-2600 v3 Product Family

Processor Model	CPU Cores	CPU Frequency	Aggregate Speed (Cores × Speed)	Retail Price (Dell, Aug. 2015)
E5-2699 v3	**18**	2.3 GHz	**41.4 GHz**	$8,900
E5-2698 v3	16	2.3 GHz	36.8 GHz	$7,040
E5-2697 v3	14	2.6 GHz	36.4 GHz	$5,850
E5-2690 v3	12	2.6 GHz	31.2 GHz	$4,560
E5-2660 v3	10	2.6 GHz	26.0 GHz	$3,130
E5-2667 v3	8	3.2 GHz	25.6 GHz	$4,490
E5-2643 v3	6	3.4 GHz	20.4 GHz	$3,360
E5-2637 v3	**4**	**3.5 GHz**	14.0 GHz	$2,160

If an application could utilize all of the cores, the fastest result would be achieved with the processor that has the most aggregate horsepower. In Table 15.1, the model E5-2699 v3, with 18 cores at 2.3 GHz each, has has nearly three times as much aggregate speed as the model with the fastest individual cores.

In contrast, if the application was single-threaded, the fastest result would come from using the model with the fastest core—in this example, the 3.5 GHz cores on the E5-2637 v3. It has single-CPU performance more than 50 percent faster than the E5-2699 v3. While this model has four cores, the application would use only one of them. The other three would either be idle or be used by other processes on the machine.

A server might also have multiple CPU sockets; a low-end server might have a single socket; a high-end server might have 12 or more. Most computers are limited in that each CPU must be of the same kind and speed. One can add capacity only by adding matching CPUs to empty sockets, or replacing all the CPUs with faster ones. The Dell R730 has two sockets; the second can be empty or be filled with an exact match of the first CPU module.

Multitasking and Multithreading

With a single CPU, an operating system can seem to run multiple programs at the same time, but this multitasking is only an illusion. A CPU can really run only one program at a time. Since each program is sharing the CPU, each program runs for a few milliseconds, and then the OS puts it to sleep and runs another program. Each time one process is put to sleep so that another can run is called a **context switch**. This cycle, when repeated, permits many programs to share a CPU. The illusion is complete.

If two programs share the same CPU and require 10 seconds to run each, the total run time will be about 20 seconds. If one program pauses to wait for a disk read or other I/O, the other program will have full use of the CPU. This can reduce that 20 seconds significantly.

With multiple CPUs each program can be assigned to a different CPU and they actually do run in parallel. In the earlier example, the total run time for the two programs, each using its own CPU, will be about 10 seconds as expected, depending on overhead and other factors. If there are more processes than CPUs, sharing begins again.

Programs can also spawn multiple threads of execution. Some web server software is designed to create one thread for each incoming web request. The OS spreads these threads over all the available CPUs—a practice that better utilizes the multiple CPUs.

However, if two or more threads need access to the same resource (for example, the same variable), it is risky for them to do so at the same time. If one is changing the variable, the other may read it in a half-changed state. To eliminate such conflicts, a program locks the variable, accesses it, and then unlocks the variable. Only one thread can hold a lock at a time; the others wait until the lock is released. Threaded programs can spend a lot of time waiting on locks, which reduces efficiency. For example, a program with eight threads running on an eight CPU machine may utilize less than the full resources of the machine.

Threading is explained in greater detail in Volume 2 of this book series in Chapter 5, “Design Patterns for Scaling.” We also recommend that you search for articles on the web about Amdahl’s law, which is a more formal way of determining whether an application would benefit from increased parallelization.

Example Applications

In this section we discuss various applications and identify whether they would benefit from having a greater number of cores overall versus a smaller number of faster cores.

A single-threaded legacy application would run fastest on a single fast core. Any additional CPUs would go mostly unused. We say “mostly” because there most certainly will be auxiliary programs running concurrently: monitoring tasks, backups, periodic maintenance tasks, cron jobs, and and so on.

High-throughput web server software systems usually manage a pool of threads, assigning each incoming HTTP request to a different thread. Such software will get the most aggregate throughput on a machine with many cores, even if they are relatively slow. Conversely, some varieties of web server software are engineered to provide high throughput on a single fast core. Check the software’s documentation.

Graphics computation may or may not benefit from many CPUs. There are too many factors to establish a general rule. One should check with the developers to learn how the software was designed, then verify their statements by running your own timing experiments or benchmarks. For example, one application might take advantage of parallelism inherent to the calculations being performed and be designed to use many

CPUs; it would benefit from the aggregate performance. Another application may be single-threaded and run best on a single fast CPU. But what if this single-threaded application is used many times over? For example, suppose the application was used to render each frame of a film. If unrelated frames can be rendered in parallel, the total throughput may be optimized with many cores. Again, benchmarks will reveal what is best.

A virtualization cluster generally runs better with many cores. VMs can be allocated dedicated cores or they can share cores. Dedicating a core to a VM guarantees a certain performance level for the VM. However, what makes VMs more efficient is to let the virtual machine manager (VMM) dynamically allocate cores to VMs. The amount of CPU power a VM needs changes over time, and the VMM dynamically schedules resources to make the best use of the system. Suppose a VM is allocated four virtual CPUs (vCPUs) but is using only two of them. The VMM can notice this and allocate the extra CPUs to another VM that is in need. Later, if the first VM starts running heavier tasks, it might receive more CPU time. Allocating more resources than exist is called **oversubscription**. For example, allocating 16 vCPUs to 8 real CPUs is a 2:1 oversubscription. This may be sufficient for applications such as VDI, where the typical VM spends most of its time idle. In contrast, a CPU-bound numerical analysis application will suffer if it shares CPUs.

If the virtualization cluster has mixed use, a hybrid solution may be best. For example, suppose the cluster is used for 100 VDI users (light-weight users who can share CPUs) plus dozens of application servers (heavy-weight use that needs dedicated, high-speed CPUs). The physical machines that make up the cluster should include a combination of many-core and faster single-core configurations.

Other CPU Types

The most commonly used family of CPUs is the x86-64 architecture, but other systems do exist. The popularity of the Sun Microsystems SPARC architecture is waning but the ARM architecture is on the rise.

The ARM architecture is particularly interesting because it has extremely good power efficiency. Because of this, the ARM chips dominate the smartphone market. However, since the 64-bit ARM architecture was announced, servers based on ARM have become a possibility.

ARM is incompatible with x86, so it doesn’t run operating systems or software that were not specifically ported to the ARM architecture. As of this writing, Microsoft Windows is not available for ARM. Sites that have experimented with ARM chips use Linux and compile their own software.

On the one hand, ARM servers tend to have lower performance than x86 chips, so you need more of them to make up the disparity. On the other hand, they use less energy, and may be a net win overall. It depends on the application. Studies have benchmarked the two options to determine which is most cost-effective when considering all the costs of ownership: initial cost, energy consumed by the chips, energy consumed for cooling, floor space consumed, and so on. For large-scale web

applications, ARM-based servers often win. Over time this once-niche architecture will likely grow to be a dominant player.

15.2.2 Memory

Another critical component to consider in server selection is memory. Random access memory (RAM) is where running programs and their data are stored. A server needs to have enough RAM to support the applications that are running, plus any RAM used by the OS and any support programs that are also running.

Until the mid-1990s CPUs and RAM were about the same speed. The RAM system could feed a CPU with data at full rate. In the mid-1990s this changed. CPU speed outpaced RAM speed. CPUs were starved for data because the RAM subsystem could not keep up.

At the same time the entire system got so fast that the latency (the time it takes information to travel from one place to another) between the CPU and the RAM started to dominate system performance. Latency can't be improved because it is limited by the speed of light, or electrons, and nobody has figured out to make either of those move faster. Therefore, as CPU and RAM speed improved but latency stayed the same, performance suffered because the CPU could not be fed data fast enough.

To improve this situation, CPU designers created a number of solutions such as various caching architectures and NUMA.

Caches

One way CPU designers overcome the problem of latency is through caching. A cache stores frequently used or soon-to-be used data close to the CPU for faster access. It uses a small amount of high-speed (expensive) RAM to improve the performance of the main system's lower-speed (less expensive) RAM.

In an enterprise environment, with shrink-wrap software, the main thing to know about the CPU caches is the cache per core ratio. The more cache there is per core, the better the performance.

To understand how caches improve performance, it is necessary to look into some of the details of how computers operate at the most basic level. A block of memory is called a page. The pages of memory in active use by a process are called the working set, or the process's memory footprint. For best performance, the working set of a process must fit entirely in the cache. Otherwise, the cache is always evicting pages to bring in new ones.

Modern processors have multiple levels of caches, called Levels 1, 2, 3, and 4, or L1, L2, L3, and L4, respectively. Advances in CPU development can result in additional cache levels.

The L1 cache is on the core itself. It is dedicated to a particular core, and becomes the core workspace. It is usually very small (a few kilobytes). There is typically very little variation in L1 cache sizes among different CPUs.

The L2 cache is near the core, often shared among cores that are on the same chip. The L1 cache feeds from the L2 cache. It is larger than the L1 cache but smaller than the L3 cache. L2 cache sizes vary slightly between CPUs. The variation may be important to developers who are optimizing code for a particular CPU, but not so much for enterprise SAs who have to run shrink-wrap code.

The L3 cache is shared among all CPUs of the machine, and often with other functions that are part of the chip package. Where an L4 cache exists, it may take on this role, leaving the L3 cache to be dedicated to the CPUs. The L3 cache can have segments dedicated to particular CPUs, graphics processing units (GPUs), or other functions. The L2 cache feeds from the L3 cache. The L3 cache is larger than the L2 cache, and there is significant variation between processors in the size of the L3 cache. This is the number that you typically see quoted in the CPU specifications. What you want to look at here is the amount of cache per CPU core.

The L4 cache, for CPUs that have it, is also incorporated into the CPU package. It feeds the L3 cache. It is larger and slower than the L3 cache. Currently, L4 caches are quite new and rare, but we expect that situation to change rapidly.

Caches improve performance but add complexity. When two cores access the same area of RAM, copies of data may appear in two different caches. If one core writes to its RAM, the data in the other cache is invalid and must be removed so that the other core is not accessing obsolete data. Doing this operation accurately without reducing performance is very complex.

In a multitasking environment, anytime the OS switches from one process to another (a context switch), the L1 cache is unlikely to have the data needed by the new process. The program will run slowly until the appropriate data is paged in. A large L2 or L3 cache helps this happen faster. If these caches are large enough, it is possible that they might contain the working sets of two different processes. A similar situation occurs when two VMs share the same CPU. If switched between two processes or VMs rapidly, the value of the cache may be negated unless the working set of each fits in the L3 cache.

This kind of performance issue may not surface until you begin testing real workloads. When performing such tests, try running the same workload with larger and larger amounts of data. You will eventually see performance stall or plateau when the amount of data reaches a certain size. This is an indicator that the working set has exceeded the size of the cache. Use this information to tune the system so that the working set stays within the cache. Developers can use this information to restructure their code to have a smaller footprint. For example, the information could be read by row instead of by column.

Over time, caches get larger and more sophisticated. By the time this book is printed, new generations of chips will undoubtedly be shipping with larger, more sophisticated caches. More levels may be added, some may be removed, and which levels are shared or not shared by cores may change.

NUMA

Another way that CPU designers have improved RAM latency is by introducing Non-Uniform Memory Architecture (NUMA) RAM systems.

In the old days, all CPUs had equal access to all RAM in the system. The speed at which a CPU could access any page of RAM was the same, or uniform. RAM was one big pool to which everyone had equal access.

To improve RAM latency, chip designers created a new RAM architecture. Each partition, or bank, of RAM would be very close to a particular CPU. The other CPUs could access the bank of another CPU, but such access would be slower or at higher latency. Access speed to RAM was no longer uniform; it depended on which bank of RAM was being accessed. There would always be a bank of RAM that was more local, faster to access, for each particular CPU.

Designers modified operating systems to take advantage of this improved access speed. The OS would coordinate between the CPU scheduler and the virtual memory system such that a process running on a particular CPU had its memory stored in that CPU's memory bank. When everything came together, performance was stellar. In fact, this development gave Silicon Graphics (SGI) a huge advantage in the scientific computing arena in the mid-1990s.

This kind of process scheduling is quite complex. In the past, if a CPU was idle, moving a process to it was easy. Now, to optimize performance, all of its data has to be copied to the receiving CPU's bank, too. Moving a process from one bank to another is very expensive. Avoiding moves, especially for complex workloads, is as difficult as predicting the future.

Once an obscure feature of high-end systems, NUMA is now commonplace in server hardware and server operating systems. Both Windows Server and Linux optimize for it.

When specifying the amount of RAM for a new server, one must take into account whether NUMA is in use. The processes that will be running on each core must fit within the bank size. Usually all banks must be the same size, and there is a minimum size for each. Therefore, an 8-core server that requires at least 16 GB of RAM for each core cannot be configured with less than 128 GB of RAM. If the processes that will run on each CPU require more than 16 GB of RAM, performance will suffer as the OS tries to move processes between CPUs, or simply gives up and permits a CPU to run a program from a distant bank.

Swap Space

Swap space is a feature of an OS's virtual memory system whereby if the system runs out of RAM, the OS can overflow to disk. The disk space used for this is usually a preallocated partition or file known as swap space. When the system runs out of RAM, the least recently used pages of RAM are copied to swap space and those pages are freed for use by programs that actively need RAM. The next time that swapped-out

page of RAM is touched, the OS copies it from disk into memory, possibly expelling some other page of memory to disk.

Disks are approximately 100 times slower than RAM, so the process of pausing a program to bring a swapped page in from disk is a performance nightmare. However, swap space is like an auto's safety air-bag: You never plan to use it, but it is good to have during an emergency.

Swap space is not a purchase-time option. Normal disk space is used. Reconfiguring a machine to use more or less swap space can be done dynamically in most operating systems.

Most servers don't use virtual memory because performance while swap space is used is terrible. However, an application crashing because the system ran out of memory is even worse. It is a good idea to allocate a small amount of swap space to handle the rare situations where swap space will prevent a crash.

Swap Space at Stack Overflow

Stack Overflow's default policy for Linux and Microsoft Windows servers is to allocate 4 GB of swap space, but to treat actual use of the swap space as an error condition. The company's monitoring systems raise an alert if swap space is used. These alerts are analyzed to determine if this was a one-time thing or a warning sign that the machine needs more RAM.

15.2.3 Network Interfaces

Servers often have multiple network interface cards (NICs) because they are connected to different networks, or because they are ganged or grouped together for increased bandwidth or resiliency. A server needs network bandwidth proportional to the number of clients accessing it. A server with many clients needs more bandwidth than one with fewer clients, assuming all clients generate the same amount of bandwidth. A server with 100 active clients, each with 1 Gbps network connections, could in theory receive 100 Gbps of traffic, but most likely there will be other bottlenecks in the network that prevent this situation from happening. That said, in general servers require faster network connections than the clients they serve.

We already discussed the concept of connecting servers to separate service, backup, and administrative networks in Section 14.4. Servers may also have separate fiber-optic interfaces for attaching to dedicated high-speed storage networks, both to speed up data access and to avoid having high-volume data access interfere with the service traffic.

Sometimes multiple NICs are grouped together to provide more bandwidth or more resiliency. For example, two 10 Gbps NICs might connect to the same switch, providing 20 Gbps aggregate bandwidth. This practice is often called bonding, or teaming, the available bandwidth. Alternatively, the same two NICs might each be connected to different switches so that if either switch fails, traffic still flows. This improves the resiliency, or survivability, of the system.

The number, speed, and configuration of NICs required depend on the functions performed by the server and the architecture of your datacenter network.

15.2.4 Disks: Hardware Versus Software RAID

Disks, particularly those with moving parts, are the most failure-prone components in most systems. In the previous chapter we discussed using RAID for data integrity. RAID can be achieved through hardware or software. You need to understand the difference before making a purchasing decision.

Software RAID

Software RAID is part of the operating system. This setup provides RAID via device drivers. The benefit of this approach is that it is inexpensive, because it is included in the price of the operating system.

The downside is that it is usually not as fast because it is running on the same CPUs as the rest of the system. RAID Levels 5 and 6 require many CPU cycles to calculate parity.

Software RAID often has fewer features. For example, it may support only RAID 1 mirrors. Older RAID software did not support RAID on the boot disk, because of the Catch-22 involved in loading the RAID drivers from the boot disk. This problem has been fixed since Windows Server 2008R2 was launched and in most Linux distributions since 2013.

Software RAID on the boot disk brings a certain amount of complexity when a disk fails. Generally the system must boot off half of the RAID mirror to get started. If that is the disk that failed, the BIOS may not be smart enough to try the second disk. Typically you have to manually reconfigure the boot drive to be the remaining good disk.

For this reason, it is a good idea to test the failure and recovery situations before the system goes into production. Keep good notes on what worked and didn’t work, and what was required to recover the system after a disk failed. It is better to know the limits of the system ahead of time than to discover them during an outage.

Hardware RAID

With hardware RAID, RAID is implemented in a hardware device, usually a card that is plugged into one of the expansion slots of the server. The hard disks plug into this device, and it manages the disks as one or more RAID groups. The server sees each RAID group as a hard disk. It has no idea that the hard disk presented to it is actually a bunch of disks in a RAID configuration. That is all done behind the scenes.

Hardware RAID usually provides higher performance and more features than software RAID. The typical hardware RAID controllers implement a wide range of RAID levels. They usually have dedicated hardware that performs parity calculations.

The downside of this approach is the higher cost. RAID controllers can be very expensive. For a simple system that does not demand extreme performance but simply needs the basic protection of a mirrored disk, hardware RAID may be overkill.

The term “hardware RAID” is slightly misleading. A RAID controller has plenty of software: It is just running on the controller itself. Because it is a self-contained system, it does not hog the computer’s CPU and memory as software RAID does. The hardware itself is specially designed for performance. For example, each disk usually has its own data path so that multiple disks running at the same time do not compete for resources. Such optimizations cannot be guaranteed in software RAID on a standard motherboard.

Some motherboards have integrated RAID controllers. These are often simple RAID controllers that provide RAID 1 mirroring only. This is sufficient for many applications.

Previously RAID was an expensive, luxurious option that only the most high-end applications could afford. Now it is inexpensive and commonplace.

Even Mirrored Disks Need Backups

A large e-commerce site used RAID 1 to duplicate the disks in its primary database server. Database corruption problems started to appear during peak usage times. Neither the database vendor nor the OS vendor was willing to accept responsibility. The SAs ultimately needed to get a memory dump from the system as the corruption was happening so that they could track down who was truly to blame.

Unknown to the SAs, the OS was using a signed integer rather than an unsigned one for a memory pointer. When the memory dump started, it reached the point at which the memory pointer became negative and started overwriting other partitions on the system disk. The RAID system faithfully copied the corruption onto the mirror, making it useless.

This OS software error caused a very long, expensive, and well-publicized outage that cost the company millions in lost transactions and dramatically lowered the price of its stock. The lesson learned here is that mirroring is quite useful, but never underestimate the utility of a good backup for getting back to a known good state.

15.2.5 Power Supplies

The second-most failure-prone component in a server is the power supply. It is very common to have *N* + 1 redundant power supplies so that if one fails, the system can keep running.

Each power supply provides a certain amount of power, measured in watts. Generally a second power supply is enough to provide *N* + 1 redundancy, but sometimes a third or fourth is required for large servers or network equipment with many fiber interfaces. The vendor can advise on how many power supplies a particular configuration requires both at a minimum and to provide *N* + 1 redundancy.

Each power supply should also have a separate power cord. Operationally speaking, the most common power problem is a power cord being accidentally pulled out of its socket. Yet, some vendors inexplicably provide dual power supplies with a single power cord. Such vendors demonstrate ignorance of this basic operational issue.

Each power supply should draw power from a different source: a separate circuit or uninterruptible power supply (UPS). Generally each power distribution unit (PDU) in a datacenter is its own circuit, so plugging each power cord into a different PDU assures two power sources.

Another reason for separate power cords is that they permit the following trick: Sometimes a device must be moved to a different power strip, UPS, or circuit. In this situation, separate power cords allow the device to move to the new power source one cord at a time, leaving the system up during the entire transition.

Separate Power Cords Save the Day

Tom once had a scheduled power outage for a UPS that powered an entire machine room. However, one router absolutely could not lose power, because it was critical to systems outside the machine room. This dependency was discovered only moments before the scheduled power outage.

Luckily, there was a wall socket near the router that would be unaffected by the outage. It had been installed for lights and other devices that did not require UPS support. While it wasn't UPS protected or an independent, clean circuit, some power was better than no power.

Shortly before the planned datacenter outage, Tom moved one of the router's power cords to the wall socket. During the outage the router ran on a single power supply. The day was saved.

15.3 Things to Leave Out

What you don't want on a server is anything you won't use. This includes fancy video cards, keyboards, monitors, and mice. Servers do not display graphics locally, so fancy video graphics cards are a waste. As discussed previously, monitors and keyboards take up precious datacenter room. A KVM permits better use of space.

A mobile phone or laptop is part computing device, part fashion statement. It really does matter what color it is. Servers, in contrast, should be used and not seen. They should be in datacenters where people generally won't see them. Servers don't need beautifully designed front panels and cases that look like they should be on display at the Museum of Modern Art. Cases should be designed to provide proper airflow at the lowest price possible. Beautiful, fancy cases are expensive to design and manufacture, yet the only value they provide is to impress you during the sales presentation. In a perfect world, vendors would offer a discount to customers who forego any purely decorative plastic bezel that attaches to the front of servers. Sadly, they don't.

If you look at pictures of Google datacenters, you'll see rack after rack of servers that are cost reduced to an extreme. No pretty front bezel. In fact, the original Google servers had no case at all. They consisted of bare motherboards that sat on metal shelves, separated only by a layer of insulation. You can see one such rack at the Computer History Museum in Mountain View, California. A little money saved on each server multiplies into big savings when you have many servers.

15.4 Summary

When you are purchasing a server, there are many options and many choices to be made. We begin by considering the application requirements and base our decisions on them.

Vendors offer different product lines, some focusing on lowest initial purchase price, lowest total cost of ownership (TCO), or highest performance. The lowest TCO is achieved by including facilities that reduce the cost of IT management, mostly by supporting remote management and features that enable the fleet to be managed as a whole instead of as individual hosts.

There are many types and models of CPUs. The Intel x86-64 is the most common today. Most servers have the option of a few fast cores or many slower cores; each is appropriate for different applications. Applications that take advantage of many cores include threaded applications, virtualization, and programs that are designed to distribute work over many cores. Applications that work better with fewer fast cores include single-threaded legacy applications and certain mathematical computation systems.

Servers need a sufficient amount of RAM to run their applications. Modern CPUs improve RAM speed through caches and NUMA. Caches may be on the core (L1), between cores (L2), or on the motherboard (L3), although such distinctions are changing over time. NUMA places certain banks of RAM near certain CPUs; the operating system then attempts to optimize performance by placing programs running on a particular CPU in the appropriate RAM banks.

Servers usually need reliable, high-speed networking. The server needs network bandwidth proportional to the number of clients that access it. A server may use faster NICs or bond many slower NICs together so that they act as one larger channel. Multiple NICs can be used to improve resilience by connecting each to a different switch, thereby creating a system that can survive failed switches.

Different servers have different storage requirements. Disks fail, so servers should use RAID to ensure that they can survive a failed disk. The exception is servers that store caches of data that can be found elsewhere. Hardware RAID means the RAID processing is done on a separate controller; it is generally faster and has more features. With software RAID, the RAID processing is part of the operating system; it is generally slower but has the benefit of being included in the cost of the operating system.