

Warehouse-Scale Architectures for Utility Computing

The datacenter is the computer.

Luiz André Barroso, Google (2007)

A hundred years ago, companies stopped generating their own power with steam engines and dynamos and plugged into the newly built electric grid. The cheap power pumped out by electric utilities didn't just change how businesses operate. It set off a chain reaction of economic and social transformations that brought the modern world into existence. Today, a similar revolution is under way. Hooked up to the Internet's global computing grid, massive information-processing plants have begun pumping data and software code into our homes and businesses. This time, it's computing that's turning into a utility.

Nicholas Carr, *The Big Switch: Rewiring the World, from Edison to Google* (2008)

6.1 Introduction

Anyone can build a fast CPU. The trick is to build a fast system.

Seymour Cray, Considered the father of the supercomputer

The warehouse-scale computer (WSC) is the foundation of Internet services billions of people use every day: search, social networking, online maps, video streaming, online shopping, online gaming, software-as-a-service, and so on. The emergence of the smartphone as a ubiquitously available and always on thin client device necessitated the creation of WSCs as a scalable, reliable, and cost-effective back-end platform for these tremendously popular Internet services. It also led to the emergence of cloud computing, a utility-based way of consuming computing hardware, software, and even data that is accessible to anyone with a credit card.

Although WSCs appear to be just large clusters, their architecture and operation are quite different, as we will see. To achieve the scale of more than 50,000 servers per WSC in a cost-effective and performant manner, all aspects of a WSC are codesigned, including hardware, software, networking, and even the power distribution and cooling infrastructure in the buildings that house the WSC. At this leading edge, the WSC is the modern descendant of the supercomputer—making Seymour Cray the godfather of today’s WSC architects. His supercomputers were also vertically codesigned to support high-performance computations (HPC) for scientists and engineers that could be done nowhere else. However, Cray’s computers were so expensive that only a few companies could afford them. Today’s WSCs act as one giant machine that costs hundreds of millions of dollars but are surprisingly cost-effective in supporting a wide variety of computations for a broad set of users. Thus WSCs arguably play a more important role for society today than Cray’s supercomputers did in the past.

WSC architects share many goals and requirements with server architects:

- *Cost-performance*—Work done per dollar is critical in part because of the scale. Reducing the costs of a collection of WSCs by few percent could save tens of millions of dollars.
- *Energy efficiency*—Except for the photons that leave WSCs, they are essentially closed systems, with almost all the energy consumed turned into heat that must be removed. Thus peak power and consumed power drive the cost of power distribution and cooling systems. They also determine the amount of hardware a WSC can host, hence its peak performance and user capacity. The majority of the infrastructure costs of building a WSC go toward power and cooling. Moreover, energy efficiency is an important part of environmental stewardship.
- *Dependability via redundancy*—The long-running nature of Internet services and their 24/7 usage mean that the hardware and software in a WSC must collectively provide at least 99.99% (called “four nines”) of availability; that is, services must be down less than 1 hour per year. Redundancy is the key

to dependability for both WSCs and servers. Although server architects often utilize more hardware at higher costs to reach high availability, WSC architects rely instead on numerous cost-effective servers connected by a network and redundancy managed by software. In addition to local redundancy inside a WSC, an organization needs redundant WSCs to mask events that can take out whole WSCs. Indeed, although every cloud service needs to be available at least 99.99% of the time, the dependability of a full Internet company like Amazon, Google, or Microsoft needs to be even higher. If one of these companies was completely offline for 1 hour per year—that is, 99.99% availability—that would be front page news. Multiple WSCs have the added benefit of reducing latency for services that are widely deployed.

- *Network I/O*—Server architects must provide a good network connection to the external world, and WSC architects must also. An additional challenge for WSC architects is to provide a high-bandwidth and often custom internal network needed to implement distributed and reliable services within a WSC and to keep data consistent between multiple WSCs.

Not surprisingly, there are also characteristics *not* shared with server architecture:

- *Wide variety of workloads*—Through cloud computing, WSCs host workloads of all types and scales. Some of these workloads include traditional server applications such as single-node databases and scientific computations. However, WSCs also host the largest distributed workloads including interactive services such as social networking and search, video streaming platforms, and massively parallel batch programs to calculate metadata useful to such services. Machine learning (ML) computations represent an increasing percentage of WSC compute cycles for both serving (inference) and batch (training) workloads.
- *Ample parallelism*—A concern for a server architect is whether the applications in the targeted marketplace have enough concurrency to justify the amount of parallel hardware and whether the cost is too high for the communication hardware needed to exploit this parallelism. A WSC architect has no such concern. First, there is ample *process-level parallelism* as more users and workloads move to cloud platforms. Second, batch applications benefit from *data-level parallelism* as they typically operate on a large number of distinct datasets that require independent processing. This is similar to the *data-level parallelism* we saw in [Chapter 4](#), this time applied to massive datasets in storage instead of smaller datasets in memory. Hence data-level parallelism can be exploited using multiple machines in a distributed manner. Finally, interactive Internet services are used by thousands to millions of independent users. Since reads and writes across users are seldom dependent, interactive services benefit from *request-level parallelism*, as many independent requests can proceed in parallel with little need for communication or synchronization.

- *Operational costs count*—Server architects typically ignore operational costs of a server, assuming that they pale in comparison to purchase costs. WSCs have a massive scale and longer lifetimes—the building and electrical and cooling infrastructure are often amortized over 10–20 years. So the operational costs can quickly add up: energy, power distribution, and cooling can represent more than 30% of the costs of a WSC over 10 years.
- *Location counts*—To build a WSC, the first step is building a warehouse. One question is where? Real estate agents emphasize location, but the location for a WSC means access to inexpensive electricity, access to water or moderate temperature air for cooling, proximity to Internet backbone optical fibers, people nearby to work in the WSC, and low risk from environmental disasters, such as earthquakes, floods, and hurricanes. A more obvious concern is just the cost of the land, including enough space to grow the WSC. For companies with many WSCs, another concern is finding a place geographically near a current or future population of Internet users, to reduce access latency over Internet services. Other factors include taxes, property costs, social issues (people sometimes want a facility in their country), data sovereignty (some jurisdictions require local hosting), cost and reliability of networking, cost and source of power (e.g., hydroelectric versus coal), weather (cooler is cheaper), and overall Internet connectivity.
- *Computing efficiently across all levels of utilization*—Server architects usually design their systems for peak performance within a cost budget and worry about power only to make sure they don't exceed the cooling capacity of their enclosure. As we will see in [Section 6.7](#), WSC servers are rarely fully utilized for a variety of reasons. High utilization is challenging as it can lead to higher latencies observed by external users or interference between the workloads sharing the compute, storage, and network components of the WSC. Given that operational costs count, WSC servers need to compute efficiently across all levels of utilization: low, medium, and high.
- *Scale and the opportunities/problems associated with scale*—Often extreme computers are extremely expensive because they require custom hardware, and yet the cost of customization cannot be effectively amortized since few extreme computers are made. However, when purchasing thousands of servers at a time, there *are* great volume discounts. WSCs are so massive internally that there is economy of scale even if there are not many WSCs. These economies of scale led to commercial cloud computing because the lower per-unit costs of a WSC meant that companies could rent servers at a profit below what it costs outsiders to do so themselves. The flip side of 100,000 servers is failures. [Figure 6.1](#) shows outages and anomalies for a small cluster of 2,400 servers. Even if a server had a mean time to failure (MTTF) of an amazing 25 years (200,000 h), the WSC architect would need to design for five server failures a day. [Figure 6.1](#) lists the annualized disk failure rate as 2%–10%.

Approx. number events in 1st year	Cause	Consequence
1 or 2	Power utility failures	Lose power to whole WSC; doesn't bring down WSC if UPS and generators work (generators work about 99% of time).
4	Cluster upgrades	Planned outage to upgrade infrastructure, many times for evolving networking needs such as recabling, to switch firmware upgrades, and so on. There are about nine planned cluster outages for every unplanned outage.
1000s	Hard-drive failures	2%–10% annual disk failure rate (Pinheiro et al., 2007)
	Slow disks	Still operate, but run $10\times$ to $20\times$ more slowly
	Bad memories	One uncorrectable DRAM error per year (Schroeder et al., 2009)
	Misconfigured machines	Configuration led to $\sim 30\%$ of service disruptions (Barroso and Hölzle, 2009)
	Flaky machines	1% of servers reboot more than once a week (Barroso and Hölzle, 2009)
5000	Individual server crashes	Machine reboot; typically takes about 5 min (caused by problems in software or hardware).

Figure 6.1 List of outages and anomalies with the approximate frequencies of occurrences in the first year of a new cluster of 2,400 servers. Based on Barroso, L.A., 2010. Warehouse Scale Computing [keynote address]. In Proceedings of ACM SIGMOD, June 8–10, 2010, Indianapolis, IN.

Given two disks per server and an annual failure rate of 4%, with 100,000 servers the WSC architect should expect to see one disk fail per *hour*. However, software failures vastly outnumber hardware failures, as Figure 6.1 shows, so the system design must be resilient to server crashes caused by software bugs, which would happen even more frequently than disk failures. With tens of thousands of servers in these very large facilities, WSC operators become very good at changing disks, so the cost of disk failure is much lower for a WSC than a small data center. The same applies to DRAMs. Plausibly, WSCs could use even less reliable components if cheaper ones were available.

Example Calculate the availability of a service running on the 2,400 servers in Figure 6.1. Unlike a service in a real WSC, in this example the service cannot tolerate hardware or software failures. Assume that the time to reboot software is 5 min and the time to repair hardware is 1 h.

Answer We can estimate service availability by calculating the time of outages because of failures of each component. We'll conservatively take the lowest number in each category in Figure 6.1 and split the 1,000 outages evenly between four components. We ignore slow disks—the fifth component of the 1,000 outages—because

they hurt performance but not availability, and power utility failures, because the uninterruptible power supply (UPS) system hides 99% of them.

$$\begin{aligned}\text{Hours Outage}_{\text{service}} &= (4 + 250 + 250 + 250) \times 1 \text{ h} + (250 + 5000) \times 5 \text{ min} \\ &= 754 + 438 = 1192 \text{ h}\end{aligned}$$

Since there are 365×24 or 8760 h in a year, availability is

$$\text{Availability}_{\text{system}} = \left(\frac{8760 - 1192}{8760} \right) = \frac{7568}{8760} = 86\%$$

Without software redundancy to mask the many outages, a service on those 2400 servers would be down on average one day a week—zero “nines”—which is far below the 99.99% availability is the goal of WSCs.

Computer clusters are the forerunners of WSCs. Clusters are collections of independent computers that are connected using local area networks (LANs) and switches. Clusters became popular in the late 1990s for scientific computing and then later for Internet services as they were more cost-effective than shared-memory multiprocessors for workloads that did not require intensive communication. (Shared-memory multiprocessors were the forerunners of the multicore computers discussed in [Chapter 5](#).) One view of WSCs is that they are just the logical evolution from clusters of hundreds of servers to tens of thousands of servers.

However, the operators of traditional clusters generally collect machines and third-party software from many parts of an organization and run them centrally. Their main focus tends to be consolidation of the many services onto fewer machines, which are isolated from each other to protect sensitive information. Hence the facilities that house these clusters tend to have a great deal of hardware and software heterogeneity to serve their varied customers inside an organization. WSC architects customize third-party software or build their own, and WSCs have much more homogeneous hardware. Overall, the WSC goal is to make the hardware/software in the warehouse act like a single computer that typically runs a variety of applications. Often the biggest cost in a conventional cluster is the people to maintain it, whereas in a well-designed WSC, the server hardware is the greatest cost, and people costs shift from the topmost to the bottommost. Since clusters typically also don't have the scale of a WSC, they don't get the economic benefits of the scale previously mentioned.

WSCs are also similar to high-performance computing systems in scale and cost—there are HPC designs with a million processors that cost hundreds of millions of dollars. HPC systems historically have been homogeneous architectures with more powerful processors and much lower-latency networks between the nodes than are found in WSCs because the HPC applications are more interdependent and communicate more frequently. HPC systems are optimized to handle a few long-running jobs that keep the servers fully utilized, even for

weeks at a time. In contrast, WSCs are architected to support a wide variety of workloads, including many interactive services that have variable load. The utilization of servers in WSCs ranges between 10% and 80% and varies throughout each day. WSCs exhibit higher hardware heterogeneity as they are populated gradually, and multiple hardware generations are introduced to better serve different workloads. Distributed storage is a main component of WSC architecture as WSC workloads are more storage and I/O intensive than scientific computations. Fault-tolerance techniques that are common in HPC systems, such as checkpoint and restore, are impractical for many WSCs workloads. Nevertheless, the recent introduction of domain-specific accelerators (DSAs) for ML in WSCs, including the networking necessary to run long training workloads across hundreds to thousands of accelerators, is drawing WSCs and HPCs architectures closer. Big portions of recent WSCs are designed to host ML supercomputers.

6.2

Cloud Computing: The Return of Utility Computing

If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility.... The computer utility could become the basis of a new and important industry.

John McCarthy, MIT centennial celebration (1961)

Utility computing goes back to commercial timesharing systems and even batch processing systems of the 1960s and 1970s, where companies only paid for a terminal and a phone line and then were billed based on how much computing they used. Many efforts since the end of timesharing have tried to offer such pay-as-you-go services, but they were often met with failure. The change that motivated cloud computing, the modern and successful version of utility computing, is the huge popularity of Internet services starting in the 2000s. Online services for business, entertainment, or education are accessed primarily through popular thin clients, either browser applications on personal computers or interface applications on smartphones. The application logic and data management for Internet services runs on back-end servers that must be highly available, sufficiently fast, and scalable to meet the evolving requirements of these services. Cloud computing allows Internet service providers of all types and sizes to rent the resources needed at the time for their service back-ends. Cloud providers maintain multiple WSCs in multiple geographic regions to support their clients.

Cloud computing offers significant advantages to its users:

- *Low barrier to entry:* Anyone with a credit card can launch a small virtual machine with a couple of cores and start using it within a few seconds at the cost

of a few USD cents per hour with no long-term commitment. In contrast, depending on your starting point, procuring the compute, storage, and networking equipment for a privately owned back-end can take weeks, cost thousands of US dollars, and require long-term contracts for space and operations support.

- *Pay as you go (scale)*: Cloud users can quickly scale up or down the number and type of virtual machines they use to closely match the load and nature of their services. In most cloud models, users are billed for the resources they use at the granularity of seconds. In contrast, a privately owned back-end will constantly be oversized (too expensive) or undersized (insufficient) as load varies. Pay-as-you-go is particularly important for startup companies, whose product may become popular overnight. Being able to use more resources and provide good quality of service to their growing customer base without a huge upfront IT spent is transformational for their business model.
- *Low operational costs*: Operational costs, such as rent for space, insurance, and the salary of human operators, are a significant contributor to the overall cost of privately owned infrastructure. A service provider with 10 servers must still employ a computer administrator with sufficient expertise in security, networking, software configuration, and reliability in order to operate their service. Cloud computing reduces operational costs significantly. The cloud provider is responsible for many operational tasks such as basic networking, security, and reliability. Operational costs are amortized across thousands of users and servers and reduced through automation and codesign, as we will discuss later in the chapter.
- *Better reliability and security*: In addition to lower cost, most cloud users enjoy significantly better reliability and security on the cloud. Small service vendors do not have or cannot afford the expertise necessary for reliable and secure operation of privately owned infrastructure. Cloud providers have this expertise and offer security and reliability features either built into their basic offerings (e.g., automatically recover and restart a failed VM) or pay-as-you-go features (e.g., automatic replication for disaster recovery). However, the shared nature of cloud infrastructure introduces new security challenges that cloud providers must address.
- *Use of latest technology*: Cloud providers are incentivized to quickly procure and deploy the latest hardware or software technology to improve efficiency or gain a competitive advantage over competitors. Due to their volume, cloud providers often have preferential access to new technology, such as the latest chips. Cloud providers are also developing some of the latest hardware and software technology themselves. Cloud users benefit from using the latest technology even if they only operate a few virtual machines.

Cloud providers can operate a profitable business due to the following reasons:

- *Economies of scale*: Large cloud providers such as AWS, Microsoft Azure, Google Cloud Platform, Alibaba Cloud, and Tencent Cloud operate millions of server machines across all locations. Hence they can benefit from economies of scale in multiple ways. For example, they get significant volume discounts when they purchase components, such as chips. They can easily amortize the R&D cost for optimized supply chains, new automation tools, custom chips for their use cases, novel distributed systems, or WSC codesign that reduces their capital or operational expenses in the long term.
- *Efficiencies from codesign*: As we will see in this chapter, all aspects of WSCs including chips, systems, software, cooling, and power delivery are code-designed to significantly improve various efficiency metrics such as reduce power consumption or reduce cloud management overheads.
- *High utilization through multiplexing*: Cloud providers multiplex users and workloads to achieve significantly higher utilization of their infrastructure than any cloud user alone would be able to achieve. This allows them to better amortize the capital expenses that dominate their overall costs such as the purchase cost of compute, storage, and networking hardware. Later in this chapter, we will discuss some of the challenges of multiplexing workloads in WSCs and some of the solutions developed by cloud providers.
- *Higher-level services*: Cloud platforms started by renting out raw cycles on slices or sets of servers. Over the years, cloud providers started offering numerous higher-level services such as databases as a service, ML as a service, or geodistributed content-delivery networks (CDNs). These higher-level services are priced at a significant premium over the cost of the raw computing cycles they consume as they reduce the amount of software that clients must deploy and operate themselves.

Cloud providers can make a significant profit while delivering cost savings to customers because they can operate IT infrastructure much cheaper than their customers would on their own. Based on a study in 2006 that compared a WSC with a traditional IT data center with only 1,000 servers, Hamilton (2010) reported the following advantages:

- *5.7 times reduction in storage costs*—At the time, it cost the WSC \$4.6 per GB per year for disk storage versus \$26 per GB for the data center.
- *7.1 times reduction in administrative costs*—The ratio of servers per administrator was over 1000 for the WSC versus just 140 for the data center.
- *7.3 times reduction in networking costs*—Internet bandwidth cost the WSC \$13 per Mbit/s/month at the time versus \$95 for the data center. Not surprisingly, one can negotiate a much better price per Mbit/s by ordering 1,000 Mbit/s than by ordering 10 Mbit/s.

Cloud computing has made the benefits of WSCs available to everyone. Cloud computing offers cost associativity with the illusion of infinite scalability at no extra cost to the user: 1,000 servers for 1 hour cost no more than 1 server for 1,000 hours. It is up to the cloud provider to ensure that there are enough servers, storage, and Internet bandwidth available to meet the demand. The previously mentioned optimized supply chain, which drops time-to-delivery to a week for new computers, is a considerable aid in providing that illusion without bankrupting the provider. This transfer of risks, cost associativity, pay-as-you-go pricing, and better reliability and security is a powerful argument for companies of varying sizes to use cloud computing instead of their own, on-premise systems.

At this point, cloud computing is extremely successful and growing in popularity. In 2021 global end-user spending on public clouds was estimated at over \$400 billion (Gartner 2022). Future growth rate is expected to be close to 20% for the upcoming years; hence it is projected that by 2025, spending on cloud services will surpass spending on traditional IT. In other words, cloud computing is already and will likely remain a major driver for computer architecture.

Cloud Computing Models

Figure 6.2 presents the three main models for cloud computing services. The difference between these models lies in which infrastructure components are managed by the cloud provider rather than the cloud user:

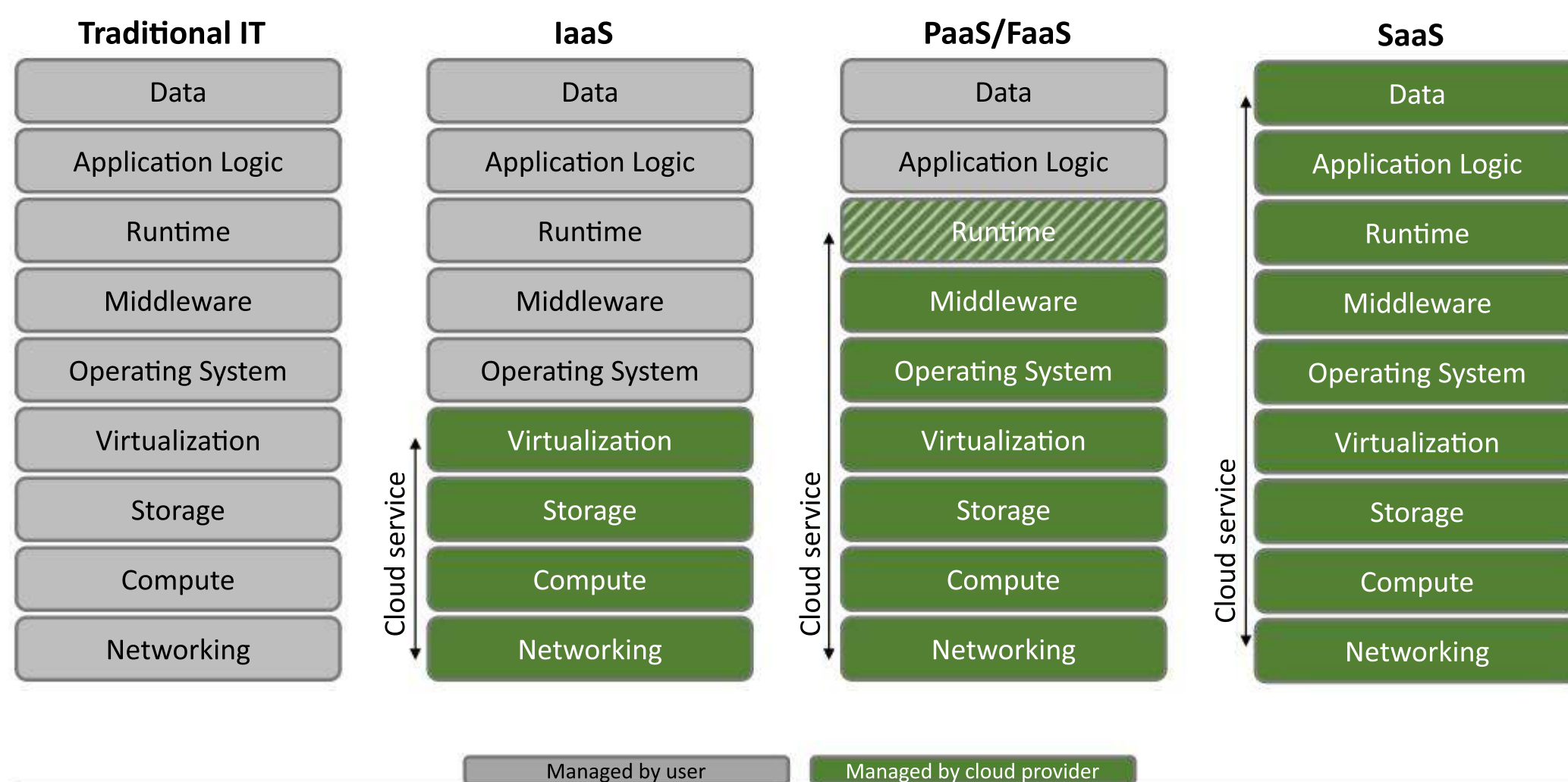


Figure 6.2 The three primary cloud computing models (IaaS, PaaS, and SaaS), compared to the technology stack for traditional enterprise IT. In traditional IT, all hardware and software layers of the stack are managed by the user. Cloud models transfer the responsibility for operating an increasing number of these layers to the cloud provider. The PaaS and FaaS models are similar in terms of management responsibility but differ in how and when application logic executes.

- *Infrastructure as a Service (IaaS)*: In this model the cloud provider offers users with virtualized resources for compute, storage, and networking. These resources can be flexibly sized (e.g., 1–256 cores and 0.5–1024 GB for virtual machines) and are functionally equivalent to physical hardware resources. However, they are implemented by slicing physical hardware using the virtualization technology we will discuss in [Section 6.3](#). On top of this virtual hardware, users launch their preferred software stack, including an operating system, one or more middleware and runtime frameworks, and one or more applications. Pricing is set based on the size of compute resources and the time they are used. Storage resources are priced based on capacity and provisioned throughput. Network bandwidth is typically linked to the size of compute resources — bigger virtual machines are provisioned with more network bandwidth.
- *Platform as a Service (PaaS)*: In this model the cloud provider manages all aspects of the operating system, the middleware, and often the runtime framework used to run user code. For example, the Elastic MapReduce service in AWS manages Hadoop clusters that users can program for distributed analytics jobs. The Elastic Kubernetes Service in AWS manages clusters running the Kubernetes orchestration framework so that users can run containerized applications. PaaS services are priced similarly to IaaS services, based on the time usage and size of virtual machines, with an added premium for the management of the additional software components.
- *Function as a Service (FaaS)*: This model, often called *Serverless Computing*, is similar to PaaS, as the cloud provider manages the operating system and an application runtime such as a container execution framework or a managed language runtime system. In PaaS, the user code executes as a long running service that can receive and process multiple external requests. In FaaS, the user code executes in a short-lived function that is launched to respond to a single external event or a small batch of events. FaaS allows users to quickly scale their cloud footprint from zero to thousands, thus optimizing their cost and performance as their application load varies. This makes FaaS very attractive for applications with high load variance, such as Internet-of-Things (IoT) applications and interactive analytics workloads.
- *Software as a Service (SaaS)*: In this model the provider offers a full application that users can access through a client device. The provider manages all aspects of the application, including its core logic and the user data it operates on. Many cloud clients use IaaS and PaaS resources to run applications that are offered as SaaS services to their consumer or enterprise customers. Cloud providers also offer some SaaS services, such as email and cloud-based productivity suites. Pricing is set based on the number of users, the volume of data, and other application-specific features and metrics.

IaaS, the original cloud model, has been gradually enriched with features that simplify the management of virtual machines. For example, providers alleviate

OS installation and updates by curating images of stable versions of popular operating systems that the user can boot automatically on their virtual machine. The number of virtual machines used by a distributed application can be automatically increased (*scaled out*) or decreased (*scaled in*) based on machine-level or application-level metrics tracked by the cloud provider's infrastructure. Users can configure reliability options such as periodic snapshots of their virtual machine's state and the use of active/passive or active/active replication for fault recovery. Finally, users can configure virtual machines to be spread across racks, independent clusters, or multiple geographic regions for better fault tolerance. These features are also available in the PaaS model.

Cloud security is an important concern as physical machines may be shared across multiple users under all models. The cornerstone of cloud security is hardware-assisted virtualization technology that provides memory isolation between concurrently running virtual machines (see [Section 6.3](#)). Network virtualization, which is also supported by hardware in network interface chips (see [Section 6.5](#)), allows cloud providers and users to selectively enable network connections within the WSC or across the Internet. Storage services offer encryption at rest and in transit. Data in memory are also encrypted in instances supported by the latest hardware. Cloud providers monitor the software they manage and update it to address security vulnerabilities when needed. Finally, cloud providers offer identity management services, which allow their enterprise clients to authenticate and authorize their own employees or clients for access control.

Nevertheless, there are use cases that are particularly sensitive to the security risks of cloud multitenancy, such as applications that operate on financial or healthcare data. The discovery of microarchitecture side-channel attacks that can potentially bypass memory isolation mechanisms has also generated additional security concerns. Cloud providers currently offer two solutions to mitigate these issues. The first solution is the availability of *trusted execution environments (TEEs) using hardware enclaves* such as Intel SGX and Arm TrustZone. Hardware enclaves encrypt portions of memory and protect the transition in and out of functions allowed to process the encrypted data. They essentially provide an additional protection layer in case an attacker bypasses the operating system and virtualization mechanisms for memory isolation or in case a rogue employee of the cloud provider can probe the memory bus of a shared machine. Enclaves are increasingly used to implement TEEs and higher-level confidential computing frameworks. The technology is evolving fast to meet emerging requirements, reduce overheads, extend applicability, and address concerns that microarchitecture attacks can also bypass enclave protection.

The second solution is a *bare-metal* cloud model, which gives the user access to a full physical machine in the cloud WSC. The machine is not shared and does not run a virtualization software stack. The cloud provider manages just the network in order to secure the bare-metal machine from other cloud users and vice versa. Network isolation is typically implemented using the network switches or by introducing an additional bump-in-the-wire between the bare-metal machine

and the data center network fabric. The bare-metal model also allows an expert user to further fine-tune everything about their software stack on the machine, without concerns about virtualization overheads.

Whether cloud clients use PaaS containers, FaaS functions, or their favorite software stack on IaaS virtual machines or bare-metal machines, they can read and write data from distributed storage services managed by the cloud provider. These services use a variety of interfaces (block, file system, or key-value store) and are optimized and priced for different speeds (Flash-based, disk-based, or optimized for archival). The *separation of compute and storage* in the cloud is a fundamental concept. It allows cloud users to easily scale their compute resources, flexibly switch between the available cloud models, and change how applications are structured, while data interfaces are stable and logically separate.

A Closer Look at IaaS Cloud Services

IaaS is the foundation that all other cloud models and services are built upon. IaaS relies on virtualization technology to slice a physical machine and run on it multiple virtual machine instances. Starting with physical machines of different configurations, cloud providers create a wide variety of IaaS instance types and sizes. [Figure 6.3](#) presents the main characteristics of a few instance types from AWS EC2 service, their IaaS offering first launched in 2006. We present 19 out of the more than 600 instance types and sizes across four families of EC2 instances: *general-purpose* instances for various workloads; *compute-optimized* instances for compute intensive workloads like gaming or media transcoding; *memory-optimized* instances for high-performance databases; *storage-optimized* instances for distributed file systems and workloads like MapReduce; and *accelerated computing* instances that provide access to devices like GPUs, FPGAs, and ML accelerators. By picking the right instance families, types, and sizes, users can right-size the performance and cost of their virtual infrastructure to their application requirements.

IaaS instances can be purchased in four different ways:

- *On-demand instances*: this option allows users to launch instances and pay by the second of usage.
- *Reserved instances and saving plans*: these options provide a significant discount over on-demand instances but require a long-term usage commitment either for an individual instance or a collection of instances.
- *Spot instances*: this option allows users to rent currently unallocated compute capacity at a significant discount. However, spot instances may be preempted if the resources are needed for other types of instances.
- *Dedicated hosts and instances*: these options allow users to ensure their instances run on hardware that is not shared with another user.

Users may select reserved instances for long-running workloads, while using on-demand or spot instances for irregular load.

	Instance	vCPUs	Memory (GiB)	Storage (GB)	Network (Gbps)	EBS (Gbps)	Accelerators	Hourly Cost
General purpose	t4g.nano	2	0.5	EBS-only	≤5	≤2.1		\$0.0042
	t4g.large	2	8	EBS-only	≤5	≤2.7		\$0.0672
	m6i.8xlarge	32	128	EBS-only	12.5	10		\$1.5360
	m6id.8xlarge	32	128	1x 1900 NVMe SSD	12.5	10		\$1.8984
	m6id.32xlarge	128	512	4x 1900 NVMe SSD	50	50		\$7.5936
	m6id.metal	128	512	4x 1900 NVMe SSD	50	40		\$7.5936
Compute optimized	c6in.8xlarge	32	64	EBS-only	50	20		\$1.8144
	c6in.32xlarge	128	256	EBS-only	200	80		\$7.2576
Memory optimized	r6i.large	2	16	EBS-only	≤12.5	≤10		\$0.1260
	r6i.32xlarge	128	1,024	EBS-only	50	40		\$8.0640
	r6i.metal	128	1,024	EBS-only	50	40		\$8,0640
Storage optimized	h1.16xlarge	64	256	8x 2000 HDD	25			\$3.7440
	im4gn.large	2	8	1 x 937 Nitro SSD	≤25	≤9.5		\$0.1819
	im4gn.16xlarge	64	256	4x 7500 Nitro SSD	100	38		\$5.8206
Accelerated computing	p4de.24xlarge	96	1152	8x 1000 NVMe SSD	400	19	8x A100 GPUs 320GB HBM2 600GB/s NVSwitch	\$40.9657
	trn1.32xlarge	128	512	4x 2000 NVMe SSD	8 x 100	80	16x Trainium chip 512GB HBM2	\$21.5000
	inf1.xlarge	4	8	EBS only	≤25	≤4.75	1x Inferentia chip	\$0.0280
	inf1.24xlarge	16	96	EBS only	25	4.75	16x Inferentia chip with direct interconnect	\$4.7210
	f1.16xlarge	64	976	4x 940 NVMe SSD	25	14	8x Xilinx VU9p FPGAs	\$13.2000

Figure 6.3 Cost and key characteristics of example instances on AWS EC2. The hourly cost is for on-demand instances in the Virginia region in the United States (usa-east-1) in December 2022. Note that the two bare-metal instances are priced identically to the equivalent VM instances (m6id.metal and m6id.32xlarge, r6i.metal and r6i.32xlarge). We review the technology that enables VM and bare-metal instances to cost and perform similarly in Section 6.8.

AWS offers multiple options for durable storage that are decoupled from the EC2 instances. AWS S3 is a simple object store that implements distributed storage with eventual consistency for a wide variety of workloads within WSCs and across the Internet. It provides multiple data tiers: *standard* for actively accessed data, *infrequent* for infrequently used data, *glacier* for backup data that may need to be quickly restored, and *glacier deep archive* for digital preservation data. Each tier level represents a different design point in the performance vs cost tradeoff for storage systems. AWS Elastic Block Storage (EBS) implements high-performance, distributed block storage, based on either solid-state drives (SSD tiers) or hard disk drives (HDD tiers). AWS Elastic File Storage (EFS) supports full file-system semantics similar to NFS. It is fully managed by AWS and its storage capacity is elastic, growing and shrinking automatically as files are created or erased. Finally, AWS FSx is an HPC file system that supports the interfaces of the

Lustre, ZFS, ONTAP, and Windows File Server file systems. Durable storage is priced based on capacity, provisioned throughput, and bursting throughput.

Cloud Metrics and WSC Architecture Drivers

Before we analyze WSC efficiency in [Section 6.7](#), it is worth understanding some of the key business metrics of cloud computing that directly impact how WSCs are optimized.

Utility computing is a viable business if the provider can consistently build and operate the shared infrastructure at a lower cost than what the clients are willing to pay for using it. This makes *total cost of ownership (TCO)* the primary metric for WSCs. TCO includes both one-time *capital expenses*, such as buying buildings and servers, as well as recurring *operational expenses*, such as paying for electricity and personnel salaries. Next, revenue from shared infrastructure is directly linked to *high utilization*. Hence WSCs hardware and software must be designed to mitigate the challenges of tightly packing workloads on shared servers, such as security and performance isolation issues.

The *architectural flexibility* to quickly deploy new hardware and software components is also important. For instance, a new ML accelerator chip may provide a 10x cost-performance advantage for customer workloads. The ability to integrate this chip into the cloud infrastructure within weeks of its release, instead of months later, can have a big impact on the cloud provider's profit and the customer's benefit. Customers also expect cloud services to be *highly reliable and available*. This requirement motivates key aspects of cloud architecture such as multiple geographic locations, multiple independent WSCs within each location, distributed architecture within each WSC, and separation of compute and storage. It also suggests that the occasional maintenance of cloud hardware and software must be mostly transparent to cloud users. While some client workloads are distributed applications that can handle an occasional instance failure, many client workloads are traditional enterprise applications that rely on the high reliability of individual machines. Cloud providers aim for *hitless upgrades* of their software so that client instances can keep running while they update the software that manages virtual machines, switches, or storage in a WSC.

This section focused mostly on *public cloud* frameworks that host client workloads, such as those operated by companies like Amazon, Microsoft, Google, Alibaba, Oracle, and Tencent. There are also a few companies with big computing needs that build and operate *private clouds* for their own workloads and Internet services. For example, Google pioneered a lot of the hardware and software technology for WSCs in the 2000s in support of its private cloud for massive workloads like web search, online maps, YouTube, and advertisement serving. Meta is also operating a big private cloud for social networking services like Facebook and Instagram. Tight control of all workloads allows a company to simplify certain aspects of its cloud framework. For example, if all workloads are designed to be distributed, it is not as important to support hitless upgrades for individual servers. Process-level and container-level isolation is sufficient for security, eliminating the

need for virtual machines. Similarly, if all programs are based on the same remote procedure call (RPC) library, this library can implement security features that would otherwise require network virtualization. Nevertheless, WSCs for private and public clouds are sufficiently similar, especially in their hardware architecture, that we will not differentiate between the two further in this chapter.

6.3 Hardware Support for Virtualization

A virtual machine is taken to be an efficient, isolated duplicate of the real machine. We explain these notions through the idea of a virtual machine monitor (VMM)... a VMM has three essential characteristics. First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.

Gerald Popek and Robert Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM* (July 1974).

Virtualization technology was first developed in the late 1960s and remained an important part of mainframe computing over the years. Although largely ignored in the domain of single-user and server computers in the 1980s and 1990s, virtualization gained popularity in the 2000s. The initial success was allowing developers to run multiple operating systems on a single machine (e.g., multiple variations of the Windows OS) to test their application code under different scenarios without buying multiple physical machines. Soon, the focus shifted toward consolidating machines in enterprise machine rooms, either to increase utilization or to get old software stacks running on new hardware that the software stack does not directly support. These enterprise use cases inspired the revival of utility computing at WSC scales, with virtualization as the core enabling technology for cloud computing.

Virtualization involves the use of additional layers of hardware or software in order to create one or more virtual copies of the underlying resource being virtualized. Starting with a *physical machine*, virtualization allows us to create multiple *virtual machines (VMs)*, where the instruction set presented by the VMs and the underlying hardware match. Such VMs are called (operating) *system virtual machines* and IBM VM/370, VMware ESX Server, and Xen are examples. Starting with a different definition of the term machine, one can construct alternative types of VMs. For example, language-based virtual machines, such as the Java VM and JavaScript engines, define as machine the user-level application binary interface (ABI). We will focus exclusively on system VMs that present the illusion that the users of a VM have an entire computer to themselves and can run their own software stack including an operating system. A single computer runs multiple VMs and can support a number of different operating systems (OSes). Without

virtualization, a single OS owns all the hardware resources; with virtualization, multiple OSes all share the hardware resources.

The software layer that manages VMs is called a *virtual machine monitor* (VMM) or *hypervisor*. The underlying hardware platform that is virtualized, the physical machine, is called the *host*, and its resources are shared among the *guest* VMs. The hypervisor is similar to the lower half of an operating system as it manages how physical resources are mapped into virtual resources. Hypervisors use a combination of three key techniques for resource management: *time sharing*, *partitioning*, and *emulation*. For example, a CPU core can be time-shared between VMs, similarly to how an OS time-shares a core between processes. In a multicore chip, cores can also be partitioned across VMs. Physical memory is partitioned with each VM given access to a subset of pages at any point in time. I/O hardware is often emulated. I/O calls by a VM are mediated by the hypervisor, which accesses I/O devices and returns the results to the VM. The core isolation portion of a hypervisor can be roughly 10,000 lines of code. The hypervisor is much smaller than a traditional OS as it does not include upper-half functionality, such as graphics and user interfaces, high-level APIs, and libraries. As we will discuss, there is now significant hardware support for hypervisors in all popular instruction set architectures (ISAs). Moreover, cloud providers are gradually off-loading the hypervisor software to specialized hardware, such as a SmartNIC, freeing cycles on the host CPU cores for client VMs.

Virtualization enables features that are transformational for utility computing:

- *Aggregation*: Virtualization allows multiple VMs with distinct software stacks to share a host. Hence a cloud provider can deploy the physical machines with best TCO, typically large servers with many cores, and aggregate on each machine multiple client VMs of varying sizes.
- *Better fault tolerance, recovery, and maintenance*: Virtualization replaces a physical machine owned by the client with a VM that runs on a cloud server. A VM encapsulates all the state of a machine and its software stack as a collection of bits. These bits can be manipulated and moved. For example, cloud providers can periodically save the VM state on durable storage to provide *VM snapshots* for fault recovery. They can copy the VM state from one host to another (*VM live migration*) to enable hitless maintenance of the original host or to support active/passive fault tolerance.
- *Better scheduling*: Since client VMs can be moved between hosts, the cloud provider can occasionally attempt to pack running VMs into a small number of hosts with high utilization. The freed hosts can be placed in low-power modes to save energy or made available for large workloads that work best when they use exclusive or neighboring hosts. The cloud provider can also move VMs to better load balance the WSC network and its storage system or to space apart VMs that experience performance interference.
- *Software flexibility and isolation*: VMs provide an abstraction that can run the complete software stack, including old operating systems such as DOS or

customized operating systems. A typical deployment might have some VMs running legacy OSes, many running the current stable OS release, and a few testing the next OS release. All these stacks can co-exist without complicating the WSC operation. Crashes, bugs, or security incidents in one VM are isolated from other VMs that run on the same host.

What Makes an Architecture Virtualizable?

A hypervisor must meet three conditions as it virtualizes the underlying hardware: *safety*, *equivalency*, and *efficiency*. Safety requires that the hypervisor is isolated from VM guests, and VM guests are isolated from each other. Guest VMs should not be able to directly change allocation of physical resources. Equivalency requires that the software running on a guest VM should behave exactly as if it were running on the native hardware. The outputs of any calculations should be identical, including any side effects, excluding differences caused by performance dependencies or the availability of resources such as the amount of physical memory each VM guest is allocated. Efficiency suggests that the performance of the software running in a guest VM should be as close as possible to that when running on native hardware. The limitations of the fixed hardware resources that are shared or partitioned across VMs, as well as the overheads of multiplexing or emulating resources, can lead to lower performance.

Let's first focus on how hardware supports the safety and equivalency requirements. To virtualize a processor, the hypervisor must control just about everything—access to privileged state, address translation, I/O, exceptions, and interrupts—even though the guest VMs currently running are temporarily using them, either to run OS code or user code. For example, in the case of a timer interrupt, the hypervisor must suspend the currently running guest VM, save its state, handle the interrupt, determine which guest VM to run next, and then load its state. Guest VMs that rely on a timer interrupt are provided by the hypervisor with a virtual timer to program and an emulated timer interrupt. To be in charge, the hypervisor must run at a higher privilege level than both the user and OS code in any VM and be able to intercept any guest operations that manipulate the physical hardware resources, such as cores, memory, or interrupts.

In 1974 Gerald Popek and Robert Goldberg formally defined the requirements that allow an ISA and its various implementations to be virtualizable (Popek & Goldberg 1974). Their famous theorem states that it is possible to construct a hypervisor if:

- the ISA defines at least two *processor modes*, a system mode and a user mode; and
- the set of *sensitive instructions* that can observe or update system state is a subset of the set of *privileged instructions* that are available only when the processor operates in the system mode.

System state refers to all configuration registers and memory structures that reflect or manage features such as the processor state, interrupts, virtual memory (MMU), security, and resource allocation. If, when executed in user mode, all instructions that can observe or update such state generate an exception that places the processor in the system mode, the hypervisor running alone in system mode is in full control of the system. If code running in a guest VM, such as an operating system, attempts to execute any sensitive instruction, the hypervisor will have a chance to intervene and manage or virtualize physical resources as needed.

In the late 1990s, when virtualization was becoming popular again, the x86 ISA did not meet the requirements of the Popek and Goldberg theorem. Specifically, there were 17 instructions that were sensitive but not privileged (Robin & Irvine 2000). For example, the `push` instruction allowed a guest operating system running in ring 1 or 3, two x86 modes for user code, to observe the value of register `%CS` that among other things describes the current processor mode. Hence a guest OS could observe it is not running in kernel mode (x86 ring 0). Nevertheless, early hypervisors were able to virtualize x86 systems using two approaches. The VMware Workstation relied on *dynamic binary translation (DBT)* to run guest VMs with unmodified operating systems. As the hypervisor loads pages of code for a VM, it scans for the problematic instructions and replaces them with equivalent, safe, and performant code sequences (Bugnion et al 2012). The Xen Hypervisor relied on *paravirtualization*, where the guest operating system is modified to eliminate problematic instructions and interact efficiently with the underlying hypervisor (Barham et al., 2003). While the hardware support for virtualization we discuss below made the x86 architecture compliant with Popek and Goldberg's theorem, DBT and paravirtualization can still be useful with addressing performance shortcomings in virtualized environments.

Architectural Support for Virtualization

Architectural support for virtualization was motivated by three factors. First, architects wanted to ensure that all sensitive instructions are privileged and eliminate the need for DBT and paravirtualization in the design of a hypervisor. Second, in a system with two operating modes, user and system, the hypervisor must run in the system mode. This leads to what is known as *mode compression* (or ring compression in x86), where the guest OS and applications must share the user mode which reduces the isolation between them. Finally, mode compression also leads to performance loss because of the excessive number of transitions between the guest VM and the hypervisor, as unmodified operating systems tend to frequently access system state through sensitive instructions. At this point, all major instruction sets include architectural support for virtualization. We will describe briefly the x86 approach and highlight main differences in Arm and RISC-V.

The VT-x extensions to x86 ISA address these issues by defining a new operating mode for the hypervisor, called the *root mode*, that is orthogonal to existing

x86 modes as shown in Figure 6.4. VM guests operate in *nonroot mode*, with the OS running in ring 0 (the system mode in x86) and user applications running in ring 3 (the user mode in x86). The additional and orthogonal mode eliminates mode compression without changing the hardware view for existing operating systems. VT-x also encapsulates all the state of a virtual machine in a physical memory structure called the *virtual machine control structure (VMCS)*. The VMCS includes all the configuration registers available to the OS in nonroot, ring 0 mode. The VMCS allows the hypervisor to manipulate the state of VMs (save, restore, read, and write) and schedule VMs for execution in a manner similar to how an OS manages user processes and their state.

When the hypervisor is ready to launch or resume the virtual machine described by a VMCS, it uses the *vmlaunch* or *vmresume* instructions. These instructions atomically copy the contents of the VMCS structure from memory into processor registers and transition the processor to the non-root mode. The VM will start executing, transitioning between its OS (ring 0) and applications (ring 3) as needed. In most cases, the OS can read and write its configuration registers using the privileged instructions in x86 without causing a transfer of control to the hypervisor. The processor will switch to the root mode and invoke the hypervisor in the following *vmexit* events: 1) if the VM code attempts to access a root-mode sensitive state (e.g., read a CPU model specific register) or use a root-mode privileged instruction (e.g., *vmlaunch*), 2) if the processor receives an external interrupt or an interrupt that cannot be handled by the VM (e.g., a reset condition), and 3) if the VM makes a call into the hypervisor using the *vmcall* instruction.

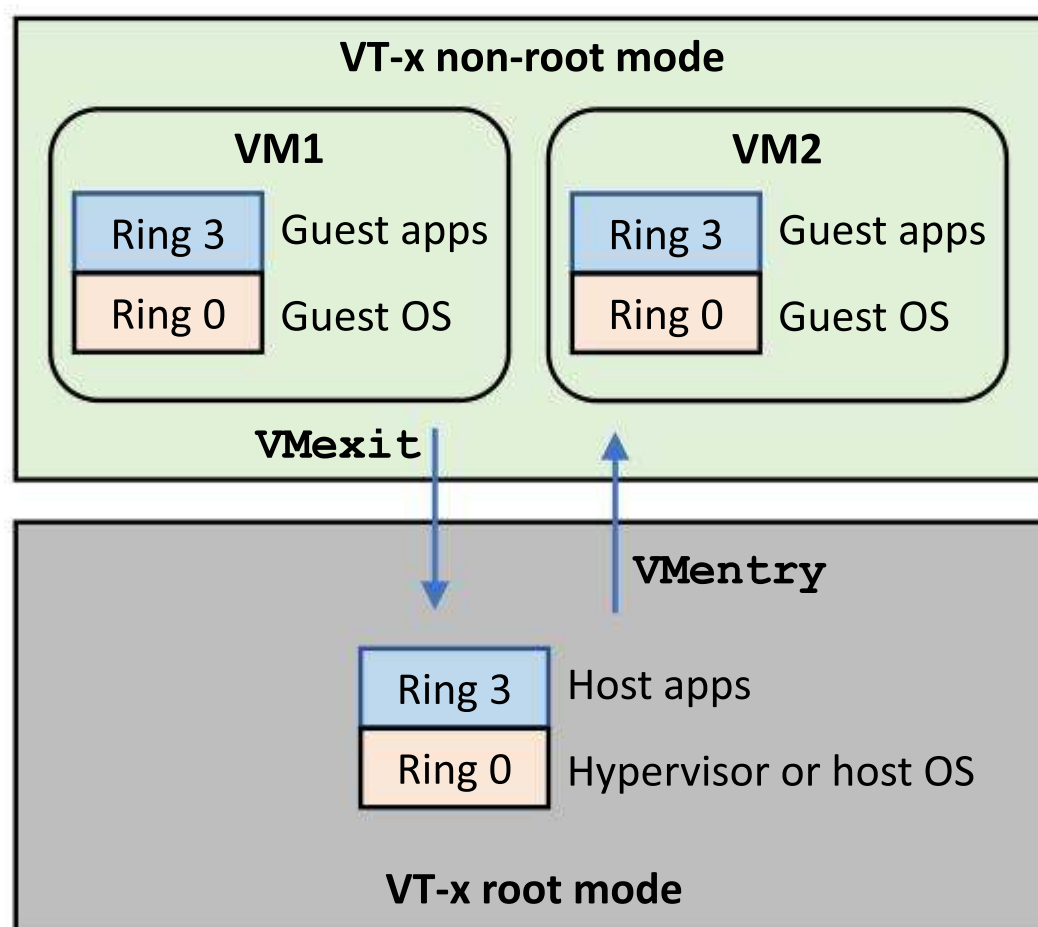


Figure 6.4 The root and nonroot operating modes in the VT-x extension for the x86 architecture. Guest OS and applications run in nonroot mode, using rings 0 and 3. WSCs typically use type-1 hypervisors that boot directly on physical hardware and run in root mode, ring 0. When virtualization is used on a personal computer, there is typically a host OS that runs in root mode, ring 0. The type-2 hypervisor runs as a host application in root mode, ring 3.

The *vmcall* instruction is equivalent to the *syscall* instruction used by a user process to call into the operating system.

The VT-x extensions meet the safety and equivalency requirements of Popek and Goldberg's theorem, albeit in a nondirect way. All instructions that can observe or modify state in root mode are root-mode privileged and will cause a *vmexit* event. However, many instructions that operate on sensitive state in the nonroot mode will execute without a *vmexit* event as the hardware guarantees that they operate on duplicate state that is exclusive to the specific VM. The latter reduces the number of transitions between the hypervisor and guest VMs and leads to better performance.

Similar to VT-x, the RISC-V ISA H extension defines the H mode for the hypervisor that is orthogonal to the S (system) and U (user) modes. The Arm ISA introduced virtualization support through the new EL2 execution mode, which is strictly more privileged than that user (EL0) and system (EL1) modes in Arm, but not orthogonal to them. Arm has an additional EL3 mode defined by the TrustZone security extensions. EL3 is orthogonal to all other modes and allows a secure monitor to split the system resources between a trusted and a non-trusted execution environment. The trusted side of the TrustZone system can be used to implement a software-only trusted platform module (TPM).

Memory Virtualization

The way memory is virtualized is critical to the safety and performance characteristics of hypervisors and virtual machines. Before we discuss how it is done, let's first define the terms that will help us understand the differences and interactions between the virtual memory we discussed in [Chapter 2](#) and memory virtualization for VMs. The actual semiconductor memory available in the system, typically built with DRAM chips, will be referred to as the *host physical memory (HPM)*. The hypervisor allocates portions of HPM to different VMs and ensures that VMs cannot read or write each other's allocation unless explicitly allowed. The guest OS in each VM manages *guest physical memory (GPM)*. The GPM is a linear address space. The guest OS has no visibility into how the hypervisor uses portions of the HPM to support the GPM. It is also possible that infrequently used portions of the GPM are temporarily stored in secondary storage through paging. Finally, each process running in the VM has its own *guest virtual memory (GVM)* address space that is managed by the guest OS and implemented on top of the GPM. In a virtualized system there is a single HPM address space, one GPM per VM, and one GVM per guest process in each VM.

Prior to ISA support for virtualization, hypervisors managed memory for unmodified guests using *shadow page tables*. The page table for a process maps GVM to GPM and is managed by the guest OS. The hypervisor would intercept all memory management operations by the OS, such as MMU and TLB management instructions, and construct a shadow page table that maps GVM to the HPM. The shadow page tables are used to refill the TLB structures and provide translations for the GVM addresses accessed by load and store instructions during

program execution. During shadow table management, the hypervisor decides how much memory each VM can use and whether a GVM page should be backed by an HPM page or by secondary storage. Shadow page tables rely on DBT or paravirtualization to intercept all memory management events by the OS and to reduce the number of times the hypervisor is invoked. Shadow paging is likely the most complex component of a hypervisor.

The VT-x extensions for x86 defined hardware support for *extended page tables (EPTs)* and eliminated the need for shadow page tables in software. The MMU and TLB instructions and logic were extended to understand the structure of both the OS-managed page tables that map GVM to GPM addresses and the hypervisor-managed page table that maps GPM to HPM addresses. On a load or store instruction, the hardware will use the TLB to quickly translate from GVM to HPM addresses. If there is a miss, the hardware-based page table walker will use both the GVM→GPM and the GPM→HPM page tables to identify the proper translation and refill the TLB, often without invoking the guest OS or the hypervisor. The guest OS will be invoked if there is a page fault during a GVM→GPM translation, while the hypervisor will be invoked if there is a page fault during GPM→HPM (translation).

The page table walk with EPT is complex because both tables (GVM→GPM, GPM→HPM) are multilevel structures and GPM addresses cannot be used before undergoing translation to HPM addresses. [Figure 6.5](#) shows the number of steps involved in translating 64-bit addresses in x86. If there is a TLB hit, the address translation completes quickly without any memory accesses for both virtualized and nonvirtualized systems. If there is a TLB miss in a nonvirtualized system, the hardware walker starts at the address pointed by base register CR3 and performs four levels of page table lookups. It accesses tables PML4, PDPT, PD, and PT using different portions of the virtual address as the index for each table lookup. The cost of translation is 4 memory accesses, one per table lookup. For a virtualized system, the cost is 24 memory accesses. The value of register CR3 and the outputs of the 4 table lookups (PML4, PDPT, PD, and PT) are GPM addresses that need to be translated using the hypervisor-maintained table for GPM to HPM translation, which is also a 4-level page table. Hence there are $5 * 4$ additional lookup steps, for a total of 24 memory accesses. The 6x increase in the cost of page table walks when virtualization is enabled puts additional pressure for high hit rate in TLBs. It also motivates the use of larger pages in order to increase the coverage of the TLB and reduce the number of page table walking steps. Appendix L covers in depth the key optimizations for memory translation such as multilevel page tables, support for multiple page sizes, and page table caching.

Arm and RISC-V define support for memory virtualization that is similar to the x86 EPTs.

Example Assume a nonvirtualized x86 system that, for typical workloads, has a baseline average memory access time (AMAT) of 2.2 cycles due to cache misses for instructions and data and L1 TLB misses but not including L2 TLB misses. Assume

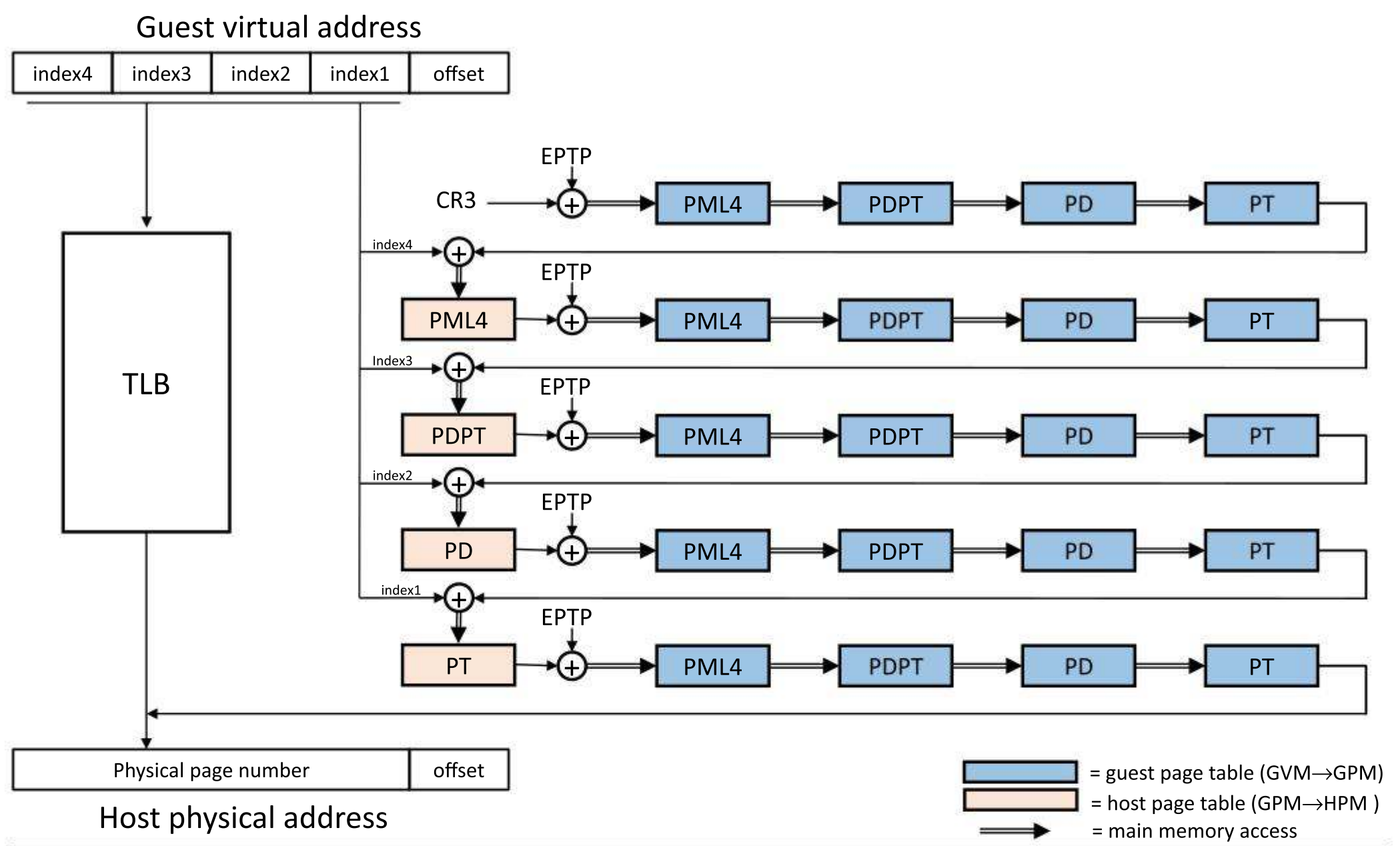


Figure 6.5 Address translation with extended page tables for x86. CR3 is the base address for the GVM→GPM page table; a GPM address itself. EPTP is the base address for the GPM→HPM page table; an HPM address itself. The translation process invoked on a TLB miss requires 24 main memory accesses.

the L2 TLB miss rate is 0.2%. Assume each page table memory access takes 50 cycles (a roughly equal mix of L2, L3, and main memory accesses). What is the overall AMAT for this system? If virtualization is turned on, what should the L2 TLB miss rate be reduced to keep the AMAT the same? If you can only optimize the L2 TLB miss rate to 0.005%, what should be the target latency for the virtualized 4-level page table walk to keep the AMAT same as the nonvirtualized system?

Answer For the nonvirtualized system, the overall AMAT includes both the baseline AMAT and the impact of L2 TLB misses, including 4 memory accesses per page table walk:

$$\begin{aligned}
 \text{AMAT} &= \text{Baseline AMAT} + \text{L2 TLB miss penalty} \\
 &= \text{Baseline AMAT} + \text{L2 TLB miss rate} \times \text{page table access cost} \\
 &= 2.2 + 0.002 \times 50 \times 4 = 2.2 + 0.4 = 2.6 \text{ cycles}
 \end{aligned}$$

As we can see, even a low L2 TLB miss rate can have a significant performance impact due to the high cost of page table walks. As mentioned in [Chapter 2](#),

hardware-based page table walkers can handle multiple outstanding TLB misses, reducing some of the overhead.

For the virtualized system, the page table cost is 6 times higher as we saw in [Figure 6.5](#). To yield the same AMAT of 2.6, the L2 TLB miss rate must be $0.2\% / 6 = 0.033\%$.

We can reduce the impact of virtualization on the AMAT by reducing the TLB miss rate using larger L2 TLBs and by using larger page sizes. For example, both Intel and AMD support 2MiB pages. If the L2 TLB miss rate is reduced to 0.05%, the page table walk time that leads to AMAT of 2.6 is

$$\text{AMAT} = 2.6$$

$$2.2 + 0.0005 \times 24 \times \text{page table access cost} = 2.6$$

$$\text{Page table access cost} = 0.4 / (0.0005 \times 24) = 33.33 \text{ cycles}$$

One way to reduce the page table access cost is to introduce a special MMU cache that holds page table entries. Rather than hold sequential addresses, the MMU cache keeps a multilevel page table entry, allowing a single access to return several linked page table entries. See Appendix L for more details.

I/O Virtualization

The baseline approach to virtualizing I/O systems, such as storage or networking hardware, is to have the hypervisor intercept all interactions between VMs and I/O hardware. This can be achieved through *vmexit* events on programmed I/O (PIO) instructions, on loads/stores to memory mapped I/O (MMIO) addresses, and on I/O interrupts. Alternatively, DBT or paravirtualization can introduce *vmcalls* into the hypervisor for I/O. There are multiple advantages to the hypervisor interposing on all I/O operations. First, the hypervisor can translate from the I/O interface assumed by the guest OS to the one of the available hardware. This allows old VMs to run using emulated I/O devices. It also allows unmodified VMs to benefit from new hardware features that their older device drivers cannot configure and exploit. Second, since the hypervisor can observe and control the state and status of all I/O operations, it is possible to move a VM between physical machines without requiring a VM reboot. The hypervisor can quiesce all I/O operations, pause the VM, transfer all the bits that capture its state to another machine (its registers and memory and the state tracked by the hypervisor), and resume its execution. As we discussed earlier, this *live migration* operation enables key benefits for cloud computing. Finally, interposition allows the hypervisor to apply a policy on how multiple VMs can safely share the capabilities of an I/O device, e.g., moderate that network bandwidth each VM can generate.

However, interposition introduces latency and throughput overheads that can be significant for I/O-intensive applications that rely on SSDs or high-speed network interfaces (NICs). For instance, a 200 Gbps NIC can deliver up to

400 million Ethernet packets per second, making the cost of interposing on all packets quite high. This problem has motivated architectural support for I/O virtualization so that VMs can directly and safely access I/O devices. Ideally, we would like to observe the following behavior for an I/O operation such as sending a network packet. First, the guest OS or a user process can directly program the I/O device using MMIO instructions. Next, the I/O device's DMA engine copies the data to be transmitted directly from VM memory without help from the hypervisor. When the packet transmission completes, the I/O device delivers an interrupt directly into the guest VM without help from the hypervisor.

We have already discussed support for the first I/O step. The EPT allows us to map the HPM addresses for I/O devices into the GPM or GVM address space of a guest. We will now focus on supporting the next two steps, DMA and interrupt virtualization, by reviewing the VT-d (IOMMU) extensions for x86. Arm and RISC-V define similar mechanisms. An additional requirement is to allow more than one VM to directly and safely access an I/O device. We will discuss the SRIOV solution for PCIe-based I/O devices in [Section 6.5](#).

The IOMMU unit defined by VT-d includes two components: one for *DMA remapping (DMAR)* and one for *interrupt remapping (IR)*. The DMAR component allows software running within a VM to program DMA engines in I/O devices using GPM or GVM addresses. When the DMA engine issues a memory read or write access, the device identity is used to look up a context identifier. For devices connected through the PCIe interface, the device identity includes a bus number (8b), a device number (5b), and a function number (3b). The context identifier includes a pointer to the page tables that should be used to translate the memory address (e.g., from GPM to HPM). Using these page tables, address translation occurs as usual. The IOMMU includes a TLB that caches recently used translations to accelerate the process. In fact, early IOMMU designs did not include a hardware page table walker. On a TLB miss, they would invoke the hypervisor to translate the address used by the DMA engine to HPM and update the TLB for further use. The TLB design for IOMMUs is quite challenging. In addition to the requirement for a low miss rate, which puts pressure on capacity, the TLB must have sufficient throughput to support one or more high-performance I/O devices, e.g., a 200 Gbps generating DMA accesses for up to 400 M packets per second per direction.

The IR component works in the following way. To generate a message-signaled interrupt, an I/O device sends a message to the advanced programmable interrupt controller (APIC) of one of the cores in the system. With VT-d, every VM is associated with a virtual APIC, which is a memory structure identified by the corresponding VMCS. When a device generates an interrupt under VT-d, the IR uses the I/O device identifier to look up the information about the VM (VMCS) that will receive the interrupt and how should the interrupt message be rewritten. The virtual APIC in the corresponding VMCS is updated to capture the pending interrupt. The virtual APIC also notifies the core that is currently running the VM and should consider delivering the interrupt. The physical

APIC uses the state of the core and the state of the VM to decide if this interrupt can be raised immediately or if it should wait. For example, the guest OS may be currently handling a higher-priority interrupt that should complete before further interrupts are triggered. When the interrupt is raised, the guest OS interrupt handler for the right interrupt vector is invoked without any intervention by the hypervisor. This *exitless* way of delivering interrupts to running VMs is known as *posted interrupts*. Posted interrupts can also be used for interprocess interrupts (IPIs) between two VMs or two processes in the same VM.

An Example Hypervisor: The Xen Project

The Xen hypervisor is an open-source project that was first introduced in 2003 and was used as the initial hypervisor in AWS. It is a bare-metal or type-1 hypervisor, which means that the hypervisor is the first software that boots on the machine. In contrast, type-2 hypervisors, such as the VMware Workstation, first boot a base operating system; the hypervisor is a user-level process that relies on services of this host OS to run additional VMs on the same hardware.

Xen introduced paravirtualization to address the safety and performance challenges of virtualizing x86 systems prior to the introduction of the VT-x and VT-d extensions (Barham et al 2003). The Xen developers modified Linux, changing about 3,000 lines or about 1% of the x86-specific code. These changes eliminated the use of sensitive but not privileged instructions and substituted code that led to big inefficiencies under virtualization. For example, to avoid flushing the TLB associated with shadow page tables, Xen maps itself into the upper 64 MiB of the address space of each VM. Xen allows the guest OS to allocate pages, checking only to be sure the guest OS does not violate protection restrictions. To protect the guest OS from the user programs in the VM, the original Xen took advantage of the three protection levels available in the x86. The Xen hypervisor runs at the highest privilege level (0), the guest OS runs at the next level (1), and the applications run at the lowest privilege level (3). Most OSes for the x86 architecture run everything at privilege levels 0 or 3. None of these changes affect the ABIs of Linux that VM workloads rely on. As of 2011, Linux supports Xen paravirtualization out of the box.

Xen calls VMs “*domains*.” An initial, privileged domain called *Dom0* boots first and manages the regular VMs, called *guest domains*. Dom0 includes the drivers for hardware I/O devices (back-end drivers). Guests use drivers for virtual disks and virtual network devices (front-end drivers), which call into Dom0 to complete I/O requests. It is also possible to disaggregate some of the Dom0 functionality using *driver domains*, which are unprivileged VMs that are responsible for a particular I/O device and its back-end driver. Driver domains avoid having Dom0 as the system bottleneck and improve its reliability — a driver crash does not affect the Dom0 operation.

Over the years, Xen evolved to utilize architectural support for virtualization. Starting in 2006, Xen uses VT-x to support unmodified guest OSes such as Microsoft Windows. VT-d was later utilized to improve I/O performance and

security. For example, driver domains utilize the IOMMU to protect other domains from back-end driver bugs and exploits. Xen supports lightweight VMs that use a mix of hardware-assisted virtualization and paravirtualization to improve VM performance and decrease its memory footprint. The smaller VMs are, the more VMs with low compute requirements can fit within the memory of a server machine in a cloud WSC.

Confidential Computing and Secure Enclaves

With the broad adoption of public cloud systems, there have been concerns about the security of data for critical applications such as financial or medical workloads. The hardware-assisted memory isolation between processes (virtual memory) or VMs (EPTs) can be bypassed by exploiting bugs in the application, the guest OS, or the hypervisor itself. Cloud computing broadens the attack surface as cloud infrastructure is public and the attacker may be running in the same WSC and even the same machine (colocated VMs). Many cloud workloads implement Internet services and hence allow malicious users to interact with them and find opportunities for code injections or other security exploits. Finally, some cloud users may not fully trust the cloud provider, either because of the possibility of rogue employees or because they directly compete with one of the cloud provider's products.

These challenges are motivating the development of confidential computing platforms, where the platform provides strong guarantees about the confidentiality and integrity of data while running on public clouds and implementing Internet services. Such platforms can be built using emerging cryptographic techniques such as homomorphic encryption, multiparty computations, and zero-knowledge proofs. Since the computational overheads of these cryptographic techniques are still significant, CPU vendors have introduced hardware support for *trusted execution*, such as the Intel software guard extensions (SGX). Although the topic of finer-grained protection has been around for decades, it has gotten little commercial traction before because of the high overheads and because other solutions that are more efficient and less intrusive have been acceptable. The rise of cyberattacks and the amount of confidential information online have led to a reexamination of techniques for improving such fine-grained security.

SGX and similar extensions in other ISAs allow user programs to create *secure enclaves*. The portions of code and data placed in the enclave are always encrypted with a key provided by the user code. They are decrypted only while in use within the CPU chip. Programs can call functions within the enclave through a narrow interface using special instructions that switch the processor to the secure enclave mode. Enclaves do not handle interrupts, page faults, or any *vmexit* events. On such events, the processor first saves the program state in encrypted memory, overwrites registers with synthetic values, exits secure mode, and invokes prespecified handlers for interrupt handling. Another notable feature of SGX is that when the enclave is initialized with code and data, it

generates a cryptographic hash of its state that an external attestation service can check to ensure that neither the code nor the data have been previously modified.

Enclaves allow for confidential execution of certain functions while only trusting the CPU vendor. Because the enclave is always encrypted, standard hypervisor or OS operations for virtual memory or I/O can access the enclave (e.g., to move a page) but cannot extract any plaintext information. For an enclave to work, all the code and all the data required for the function must be part of the enclave. Like Intel's SGX, IBM and AMD's recent processors support on-the-fly main memory encryption.

6.4

Computer Architecture of Warehouse-Scale Computers

A WSC consists of 50,000–100,000 compute servers and storage blocks connected through a hierarchical network. Compute, storage, and networking equipment are placed in racks, along with power delivery equipment such as AC to DC power converters and backup batteries. [Figure 6.6](#) shows an example WSC rack

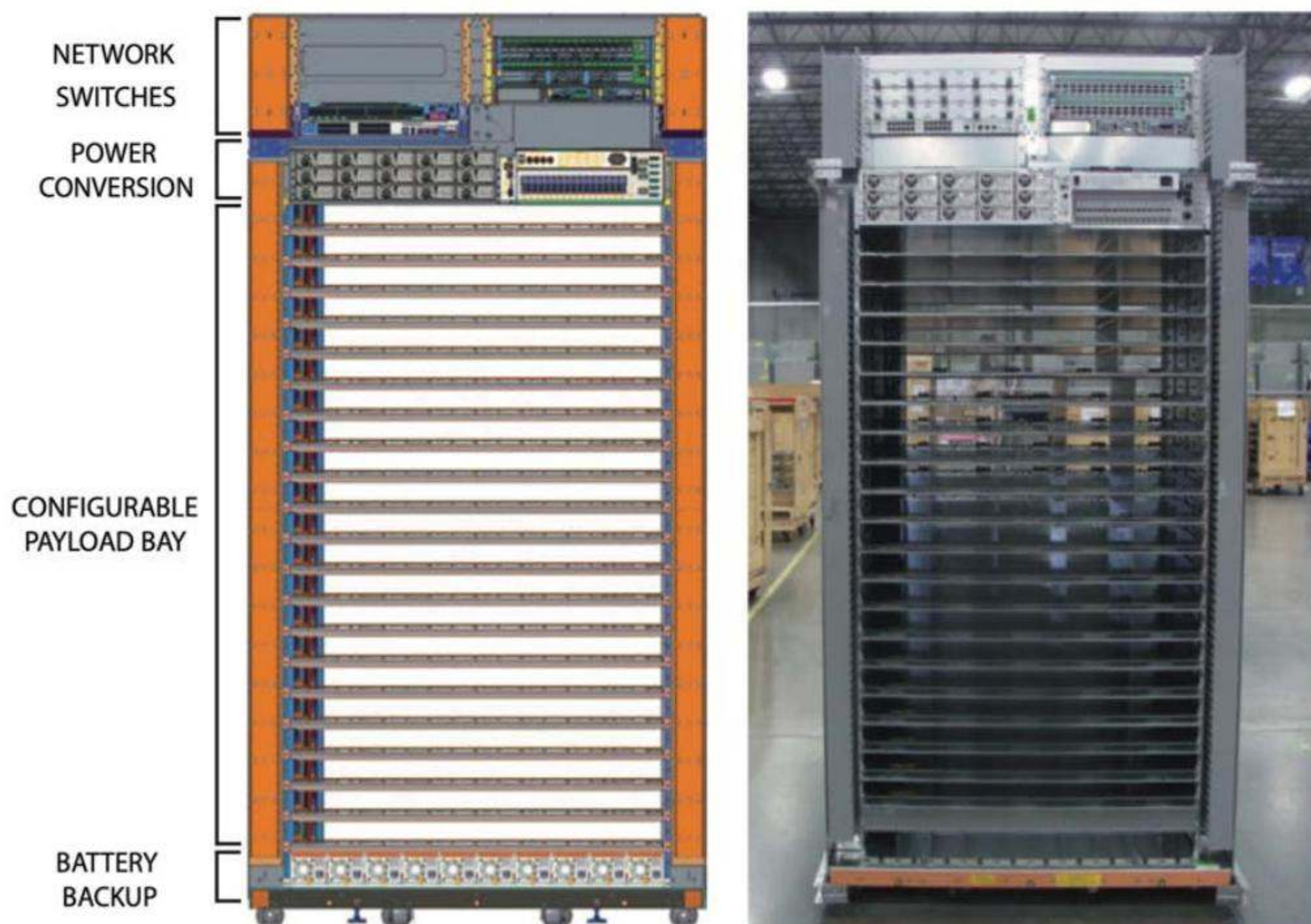


Figure 6.6 A Google machine rack that supports servers, storage, and networking equipment. From ([Barroso et al., 2018](#)), Figure 3.3.

from Google. Rack dimensions vary significantly, with traditional racks sized 19 inches wide by 48 inches deep, while several recent designs are shallower and wider to accommodate multiple servers in each horizontal tray. Rack height is typically 70–84 inches and is reported in *rack units* (1 RU = 1.75 inches = 4.44 cm). The number of servers per rack is also limited by the available power (e.g., 10–30 KW per rack), cooling and airflow constraints, and the bandwidth available to connect the rack with the network fabric. Racks are arranged in rows within warehouse building as shown in Figure 6.7. Figure 6.8 shows a



Figure 6.7 Row of server racks in a Google WSC. Although hard to see, there are bus ducts above the racks that distributed 400 V power to the servers. This picture shows a cold aisle that operators use to service the equipment.



Figure 6.8 The Google data center facility in Eemshaven, Netherlands. In addition to the multiple, multistory buildings that host the IT equipment in racks, the picture shows the transformers, switchgear, and generators needed to operate a WSC.

WSC facility with multiple warehouse-sized buildings that hosts the racks and the electrical and cooling equipment necessary to operate them.

In this section, we will describe the compute, storage, and networking components of a WSC. We highlight a few examples from the *Open Compute Project* (OCP, <https://www.opencompute.org/>), which maintains detailed specifications for hardware components designed by WSC operators such as Meta, Google, Microsoft, and Alibaba. In Section 6.6, we will describe the power delivery and cooling equipment that are critical to the WSC operation and are codesigned with the electronics components.

WSC Compute

While WSCs are relatively homogeneous compared to traditional enterprise data centers, they still host a variety of server configurations. The primary motivation for heterogeneity is to better match the hardware capabilities and cost to the characteristics of key WSC workloads. Figure 6.9 summarizes the seven server types used in Meta’s WSCs in 2017 and the workloads that matched each server configuration. Racks may include a single server type or multiple server types, as long as they meet the overall power delivery and cooling constraints. Hence, some racks may not be fully populated. WSCs host multiple generations of server and rack types as hardware is gradually installed and replaced over the lifetime of the warehouse.

Figure 6.10 shows a 1RU 2-socket server. This is a common server configuration in WSCs as it can be configured with the right compute and memory capacity to handle a wide range of applications. Since the fixed cost of the server (metal, board, fans, NIC) is amortized across two CPU sockets, this server configuration is cost-effective and works well even when the server is sliced to run multiple small VMs. There are server designs that further improve cost and modularity. Figure 6.11 presents the Meta Yosemite multi-node server design. The 4U chassis includes 3 sleds that can host 1–4 server blades of different configurations. In the base configuration, the sled includes four 1-socket servers and a small board at the top for the baseboard management controller (BMC) and the NIC that are shared

	Type I	Type III	Type IV	Type V	Type VI	Type VII	Type VIII
Use-case	Web serving	Databases	Data analytics	Photos	Multiple services	Cold storage	Disaggregated Flash
CPU	4x Xeon D	2x Xeon E5	2x Xeon E5	2x Xeon E5	2x Xeon E5	2x Xeon E5	2x Xeon E5
Memory	32GB	256GB	128GB	32GB	256GB	256GB	32GB
Storage	1x 500GB SATA	2x 3.2TB PCIe Flash	15x 5TB PCIe Flash	30x 4TB SAS HDD	1x 1.8TB PCIe Flash	240x 4TB SATA HDD	15x 5TB SATA/PCIe Flash

Figure 6.9 The seven server types used in Meta’s WSCs as presented in the 2017 OCP Summit (<https://engineering.fb.com/2017/03/20/networking-traffic/ocp-summit-2017-facebook-news-recap/>). The type I server is essentially an enclosure with 4 single-socket servers that share a network interface. The remaining types are 2-socket servers with varying CPU, memory, and storage configurations. Types VI, V, VII, and VIII couple a 2-socket server with one or more storage trays.

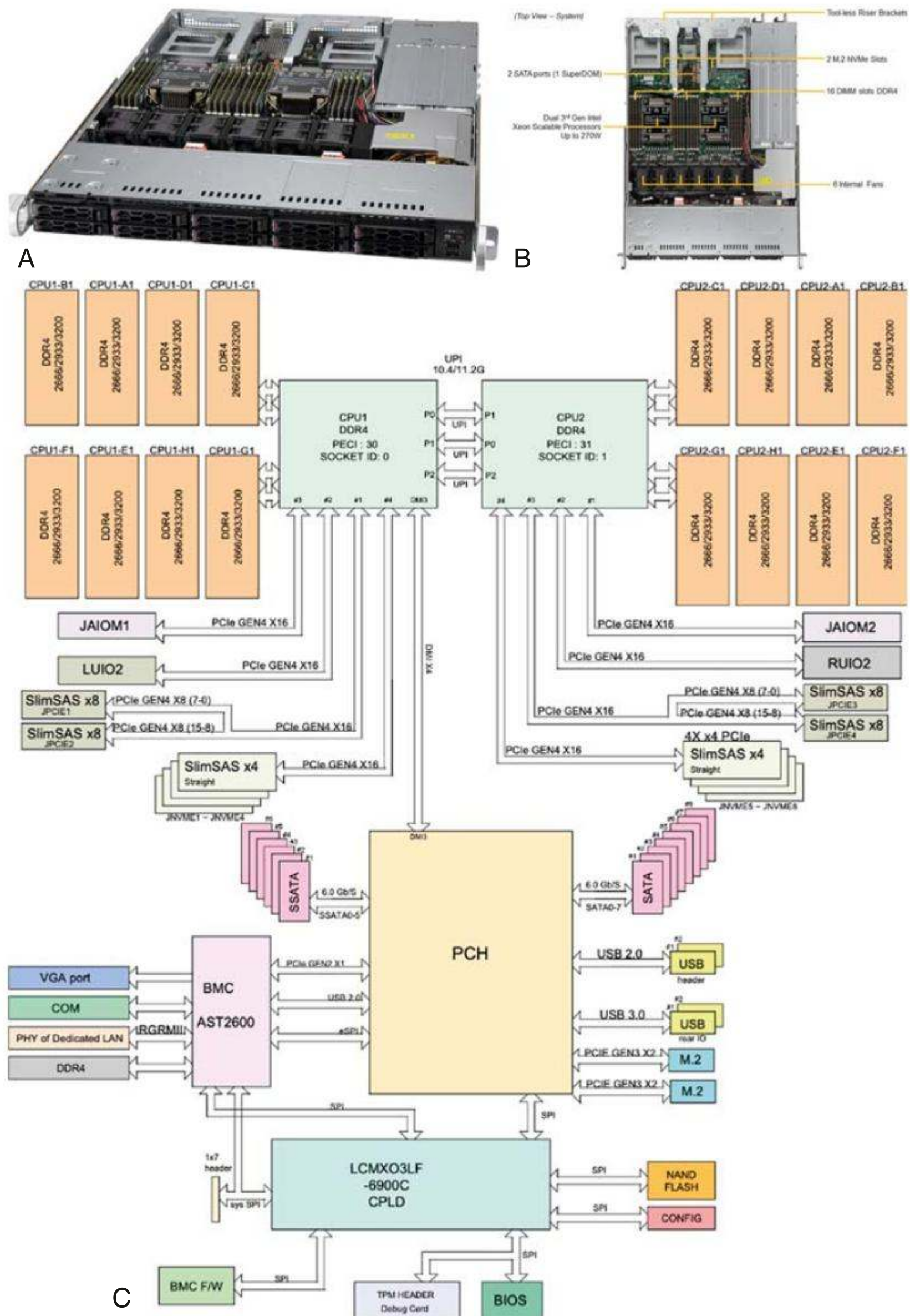


Figure 6.10 Front view, top view, and block diagram of a SuperMicro SuperServer (SYS-120C-TN10R) that is based on OCP 3.0 design concepts. The server can host 2 third-generation Intel Xeon scalable processors (up to 40 SMT cores per socket), 16 DDR4 memory modules (up to 4 TB capacity), 10 2.5-inch storage drives (NVMe, SAS, or SATA), and 2 M.2 NVMe drives. Other I/O devices, such as network interfaces or disk enclosures, can be connected through the two x16 PCIe4 ports. The server dimensions in inches are 17.2W × 23.5D × 1.7D. The server can host two redundant 800W 120V DC power supplies for enterprise deployments. In a WSC deployment, a server will likely receive DC power from a rack-level power module. The detailed specification is available at <https://www.supermicro.com/en/products/system/cloud/1u/sys-120c-tn10r>.

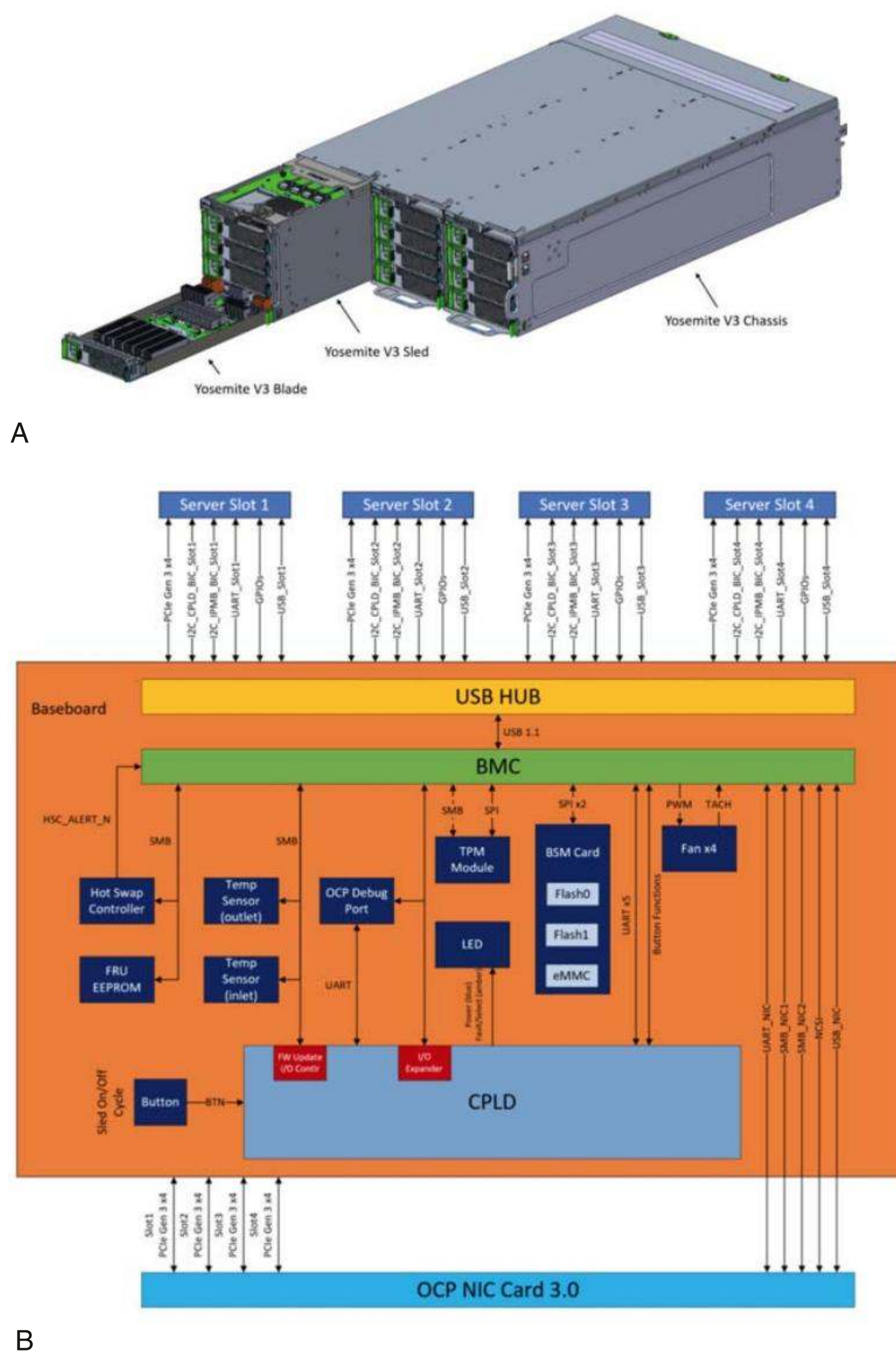


Figure 6.11 The Meta Yosemite v3 chassis and platform block diagram. Each 3RU chassis hosts 3 vertical sleds with 4 blades per sled. There are various blade designs for 1-socket servers, Flash expansion boards, or accelerator boards. A fully populated OCP Openrack v2 can fit 8 chassis or 96 1-socket servers. All Yosemite blades are front serviceable without interrupting the operation of the remaining blades. The detailed specification is available at <https://www.opencompute.org/documents/ocp-yosemite-v3-platform-design-specification-1v16-pdf>.

across all 4 blades. The BMC manages secure boot, fans, voltage regulators, and various system sensors. The NIC includes a 4x PCIe switch that allows it to present each server with a physically separate interface to the NIC's resources. This configuration is cost-effective for workloads that have ample session-level parallelism and small memory requirements, such as front-end webserver, scale-out microservices, and small to medium VMs. These workloads achieve the same throughput with two 1-socket servers as with one 2-socket servers. Hence there is no benefit from using the more expensive CPU chips that go into 2-socket servers. The Yosemite design allows multiple 1-CPU servers to amortize the cost of the BMC and NIC. The Yosemite sled can also host other blade configurations, including blades with shared storage or shared domain-specific accelerators (DSAs).

Small DSAs, such as chips for ML inference or video transcoding, can be deployed as PCIe or M.2 peripheral boards within the servers discussed above. However, the importance, scale, and computational intensity of training and serving large ML models, such as large language models (LLMs), has led to the deployment of specialized servers for DSAs. These servers host 4–16 high-end DSA chips, with each chip consuming 800 W or more. Each server connects to three networks: 1) the ordinary data center network that connects them to distributed storage and the public Internet, 2) a high-bandwidth network that directly connects the DSAs within a server, and 3) a separate interserver interconnect that supports distributed training and inference across multiple servers. These servers are the building blocks for the DSA supercomputers we discuss in [Chapter 7](#).

[Figure 6.12](#) presents the NVIDIA DGX H100 server for ML training with GPUs. The 3U server includes 8 H100 Tensor Core GPUs, each with 80 GB of HBM memory and capable of up to 8 PFLOPS for FP8 operations or 2 PFLOPS for BF16 or FP16 operations. The 8 GPUs communicate directly through the NVLink interconnect using 4 switch chips at the peak bandwidth of 900 GB/s per GPU (18 links per GPU at 50 GB/s per link). The four switches expose externally 72 additional links in order to build larger systems. DGX pod systems with up to 32 DGX servers (256 GPUs) can be connected using an additional layer of NVLink switches. Larger systems connect DGX pods in an Infiniband network using one 400 Gbps CX7 NICs per GPU. The GPUs are managed by two x86 CPUs with up to 2 TB of additional memory and separate NICs to connect to the ordinary WSC network. A single DGX server consumes up to 10 KW. Similar GPU servers are common in public cloud and private WSCs. In [Chapter 7](#) we take an end-to-end look at system design for DSAs.

Beyond the wide adoption of DSAs, there are two significant trends for WSC compute. After two decades of x86 monopoly, servers with Arm CPUs are now available by all major cloud providers. AWS and Alibaba have already developed their own Arm CPU chips, taking advantage of their high volumes and creating the opportunity for further optimizations for their WSC environment. Contrary to what was expected a decade ago, these Arm CPUs include powerful OOO cores that are competitive with their x86 counterparts (see pitfall in [Section 6.9](#)).

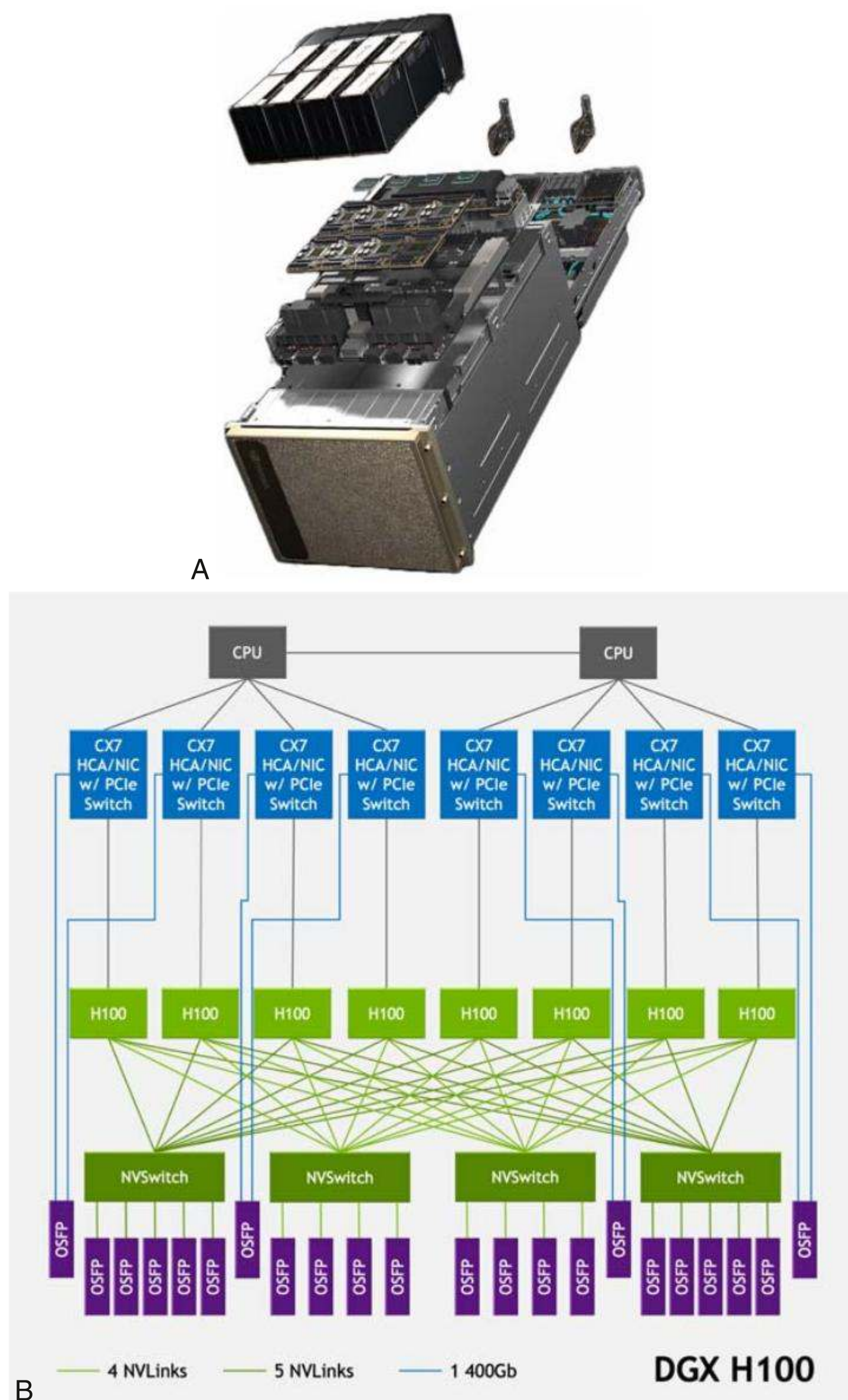


Figure 6.12 The picture and block diagram of the NVIDIA DGX H100 server for GPU-based ML training. Each H100 GPU connects to the 4 NVSwitch chips using 18 links in total. It also connects to a dedicated CX7 Infiniband NIC over a PCIe x16 link. The NIC also functions as a PCIe switch to connect the GPU to one of the two x86 CPUs. The block diagram does not include the 1 or 2 additional NICs used to connect the x86 CPUs to the data center network or the PCIe switches needed to connect the CPUs to local storage such as NVMe drives.

A major change is also coming to main memory systems. The number of cores in CPU chips is rising quickly as semiconductor scaling continues. Moreover, CPU vendors are copackaging multiple chiplets to create large CPU sockets with 64 cores or more that consume up to 400 W each and require significant memory bandwidth. Current CPUs integrate memory controllers and communicate to DRAM modules directly through DDR channels. Each DDR pin is limited to 3.2–8.4 Gb/s per pin, as it is difficult to design faster transceivers in DRAM technology. In contrast, the logic technology used for CPU manufacturing allows for transceivers that operate at 64 Gb/s (PCIe) or 100 Gb/s (Ethernet) per pin. There is an emerging trend to move some or all of the DRAM controllers into a separate buffer chip, as was the case until the mid-2000s, using PCIe-speed links to connect CPUs to buffers. The buffer chips will connect to a large number of DDR channels to implement a high-capacity and high-bandwidth memory system. Similar to GPUs and other DSAs, CPU packages may also integrate some amount of stacked HBM DRAM. The trend for off-chip memory controllers, promoted by the Compute Express Link (CXL) consortium (<https://www.computeexpresslink.org/>), opens several additional opportunities in memory systems design. CXL buffers simplify the deployment of new, nonvolatile memory technologies. They also enable sharing DRAM capacity between a small number of servers (Li et al 2023), similar to how the Yosemite sled shares the NIC across four servers. Switched CXL fabrics can allow for larger-scale memory disaggregation.

WSC Storage

Several WSC servers include local storage, such as 1–8 HDDs or Flash SSDs. Local storage is necessary for enterprise workloads that expect to run on a server with local devices or need higher storage throughput than what can be provisioned per server over the WSC network. For example, 8 NVMe-based SSDs have peak read bandwidth of 400 Gbps. VMs using local storage must be aware that the data is lost if the disks or the server fails. If local storage is used for non-ephemeral data, the VM software is responsible for persistence, for example, by copying data across multiple instances. Data in local storage is encrypted using client-provided keys so that no data leaks as the physical hardware is used by different VMs over time.

Most WSC workloads store persistent data in distributed storage systems. Cloud providers offer a variety of storage frameworks with block, file, or object interfaces, backed by HDDs or SSDs at various performance and price points. Distributed storage systems offer several advantages. The physical separation of compute hardware from persistent storage simplifies management. VMs can move or fail and restart but still connect to the same storage volumes or file systems over the network. Responsibility for persistence, failure and disaster recovery, and scaling the allocated storage capacity and throughput is transferred to the cloud provider.

The hardware components to implement distributed storage are disk arrays known as *just a bunch of disks* (JBOD). Figure 6.13 describes the Meta Grand

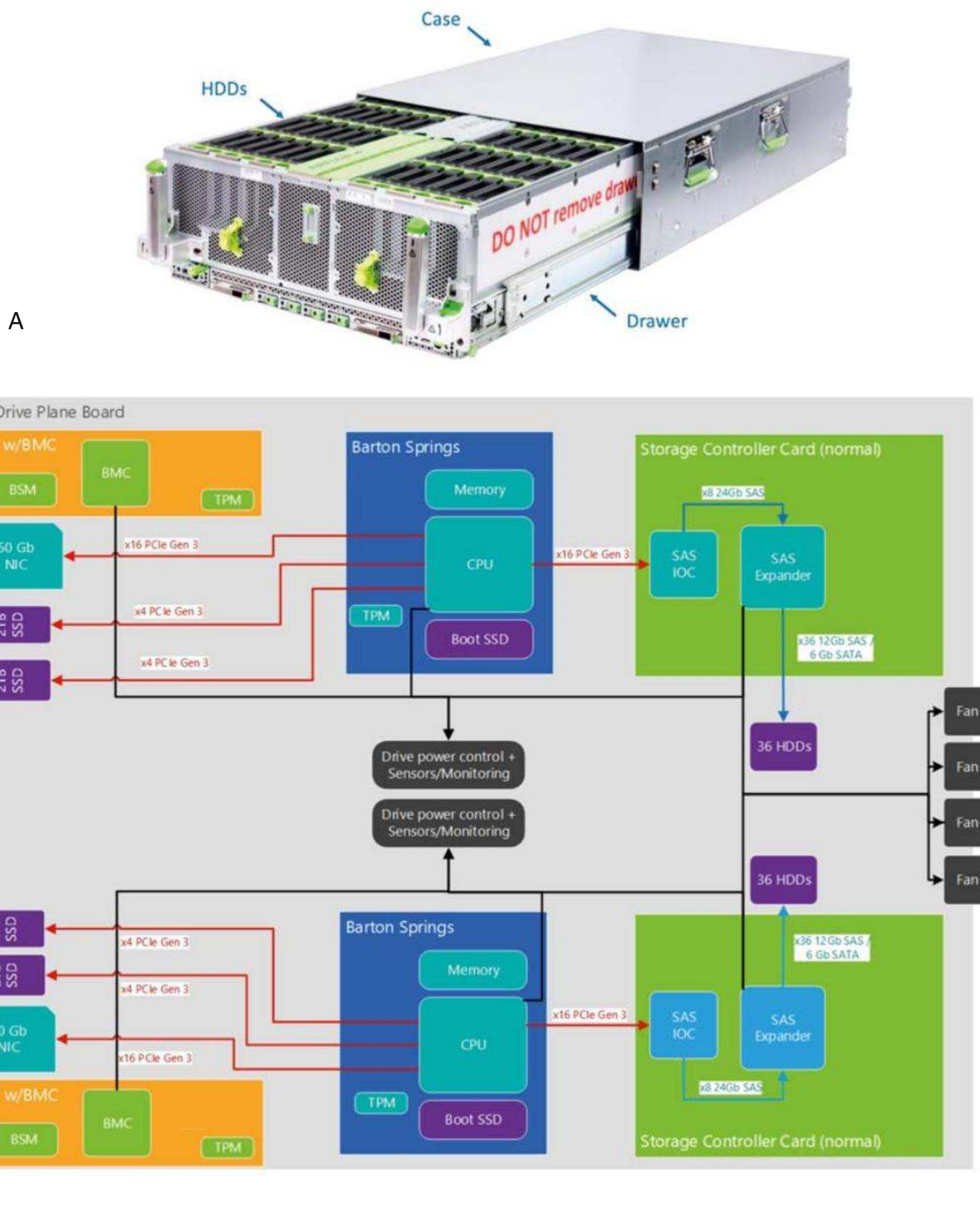


Figure 6.13 The picture and block diagram of the Meta Grand Canyon disk storage system. The block diagram shows a configuration with two 1-socket servers integrated into the storage system to act as head nodes. JBOD configurations would not include any servers and would use mini-SAS cables to connect to 1 or 2 servers in the rack. The detailed specification is available at <https://www.opencompute.org/documents/grand-canyon-storage-system-specification-v1-0-pdf>.

Canyon JBOD array that hosts 72 HDDs in a 4U enclosure. At 16 TB per HDD, this is a petascale storage device. Figure 6.14 describes the Meta Lightning JBOF array that includes two independent Flash trays. Each tray has 15 slots, and each slot can host either 2 SSDs in the M.2 form factor (2.5 inches) or 1 SSD in the U.2 form factor (3.5 inches). JBOD and JBOF arrays include the devices needed for power and signal integrity, telemetry, and management, including hot-plug support. However, they do not implement any reliability schemes such as replication

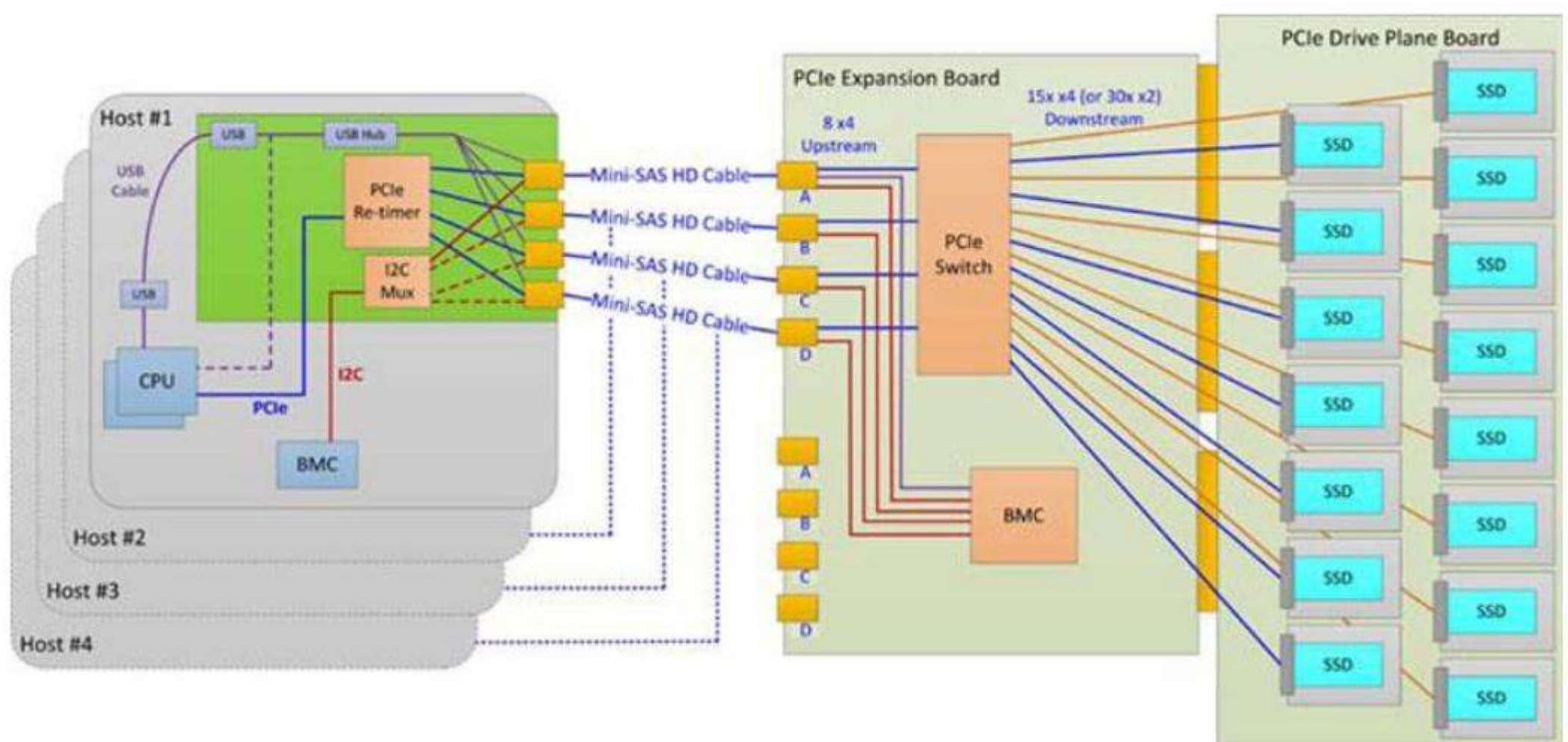


Figure 6.14 The picture and block diagram of the Meta Lightning NVMe JBOF system. The picture shows an enclosure with two Lightning trays. The block diagram shows how each Lightning tray can connect to up to 4 servers using mini-SAS cables. The detailed specification is available at <https://www.opencompute.org/documents/facebook-lightning-hardware-specification>.

or RAID codes within the array. These arrays connect to the servers above or below them in the rack using PCIe cables, often referred to as the *head nodes*. The WSC storage software running in these servers implements the distributed storage functionality, including replication or erasure coding across racks, rows, and clusters that can recover the data in the presence of a wide range of failures in the WSC: disk failures, enclosure failures, server hardware or software failures, rack failures, or network failures. Whether a disk failed or the network link that serves a JBOD array failed, the distributed storage software will recover the data the same way if the replicas or code blocks are distributed across the whole WSC. The total number of replicas or code blocks is also lower when multiple failure cases are addressed with a unified mechanism.

WSC architects can tune the deployment of storage arrays and optimize them for various use cases ranging from fast storage for hot data to low-cost archival storage. First, they can deploy different types of HDDs and SSDs, optimized for throughput or capacity. Next, they can tune the number of head nodes, the bandwidth of the connection between the array and the head nodes, and the network bandwidth allocated to connect to the WSC network. For instance, each Lightning JBOF tray can be served by a single head node through one x8 PCIe connection or 4 nodes using 4 PCIe connections. Finally, they can tune the fault-tolerance scheme used to trade off speed for storage overheads. A 3x replication scheme can tolerate 2 disk failures and has instantaneous recovery, but comes at 200% storage overhead. In contrast, Reed Solomon codes can reduce the overhead of fault tolerance but introduce recovery latency. For example, an RS(9,6) code uses 3 extra blocks for every group of 6 blocks. Its storage overhead is 50% and it requires 6 blocks to be read to reconstruct any failed block. An RS(14,10) code has 40% overhead but requires 10 blocks to be read to reconstruct any failed block.

WSC Networking

WSC networking traffic has been growing rapidly as both the number of WSC nodes and the compute or storage capabilities of each individual node have been increasing. [Figure 6.15](#) shows that network traffic between servers in Google WSCs increased by 235x within just 10 years. Clearly, without great care, the WSC network can easily become a performance or cost bottleneck.

The ideal WSC network is a perfect crossbar switch that can connect 50–100 K nodes at *full bisection bandwidth*: all pairs of two nodes can communicate concurrently at the full speed of their NICs (e.g., 100 Gbps). Moreover, the WSC network must be resilient: servers should be able to communicate at high bandwidth despite the occasional failures of networking links and switches in a large system. Unfortunately, this ideal switch is impractical as its cost for n ports (nodes) grows as n^2 . In the early 2010s, it became clear that no enterprise networking solution could meet the scale and cost requirements of WSCs. As a result, cloud providers developed a new breed of networks for WSCs that

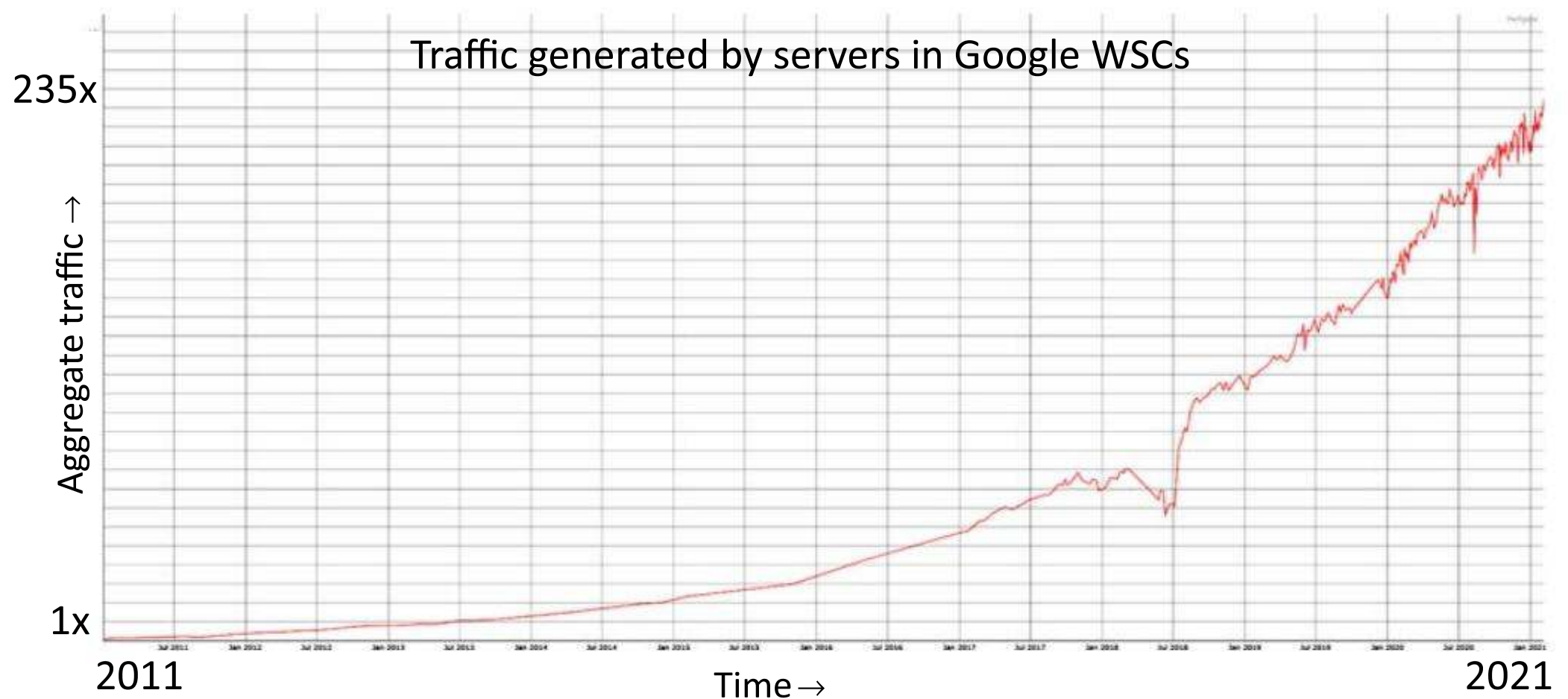


Figure 6.15 Network traffic from all servers in Google’s WSCs over 10 years (2011–2021).

combines ideas from supercomputing (topologies) and commodity networking (switch chips and transport protocols).

Modern WSC networks use multistage topologies, such as the Clos topology invented for circuit-switched, telephony networks and named after the telecommunications expert who formalized its properties (Clos, 1953). The multiple stages of switches allow this topology to approximate the connectivity of a large crossbar at a significantly lower cost. [Figure 6.16](#) shows the generic structure of a Clos network in a WSC. The first switch stage is the *top-of-the-rack (TOR) or L1 switches* that exist on every rack. Each TOR switch uses its downlinks to connect to servers, while its uplinks connect to a group of *leaf or L2 switches*. Each leaf switch connects to all *spine or L3 switches*. This modular topology scales to tens of thousands of nodes by adding leaf groups. The topology can be extended using additional levels, following the leaf-to-spine connectivity pattern. Appendix L covers in depth interconnection networks, including topologies and switch microarchitecture.

The Clos topology is cost-effective because switch systems at all three levels are built using the same commodity switch chips. [Figure 6.17](#) shows the switches used in Google’s Jupiter interconnect, their fifth generation of custom WSC network, first deployed in 2012 (Singh et al, 2015). TOR, leaf, and spine switches are implemented using one or more boards with four 16-port crossbar chips. At its maximum size, the Jupiter bisection bandwidth is an impressive 1.3 Pbit (10^{15}) per second. Note that the whole Internet might have a bisection bandwidth of just 0.2 Pbit/s; one reason is that Jupiter was built for a high bisection bandwidth, but the Internet was not. [Figure 6.18](#) reviews how the switches at each level evolved over the five generations of custom WSC networks at Google.

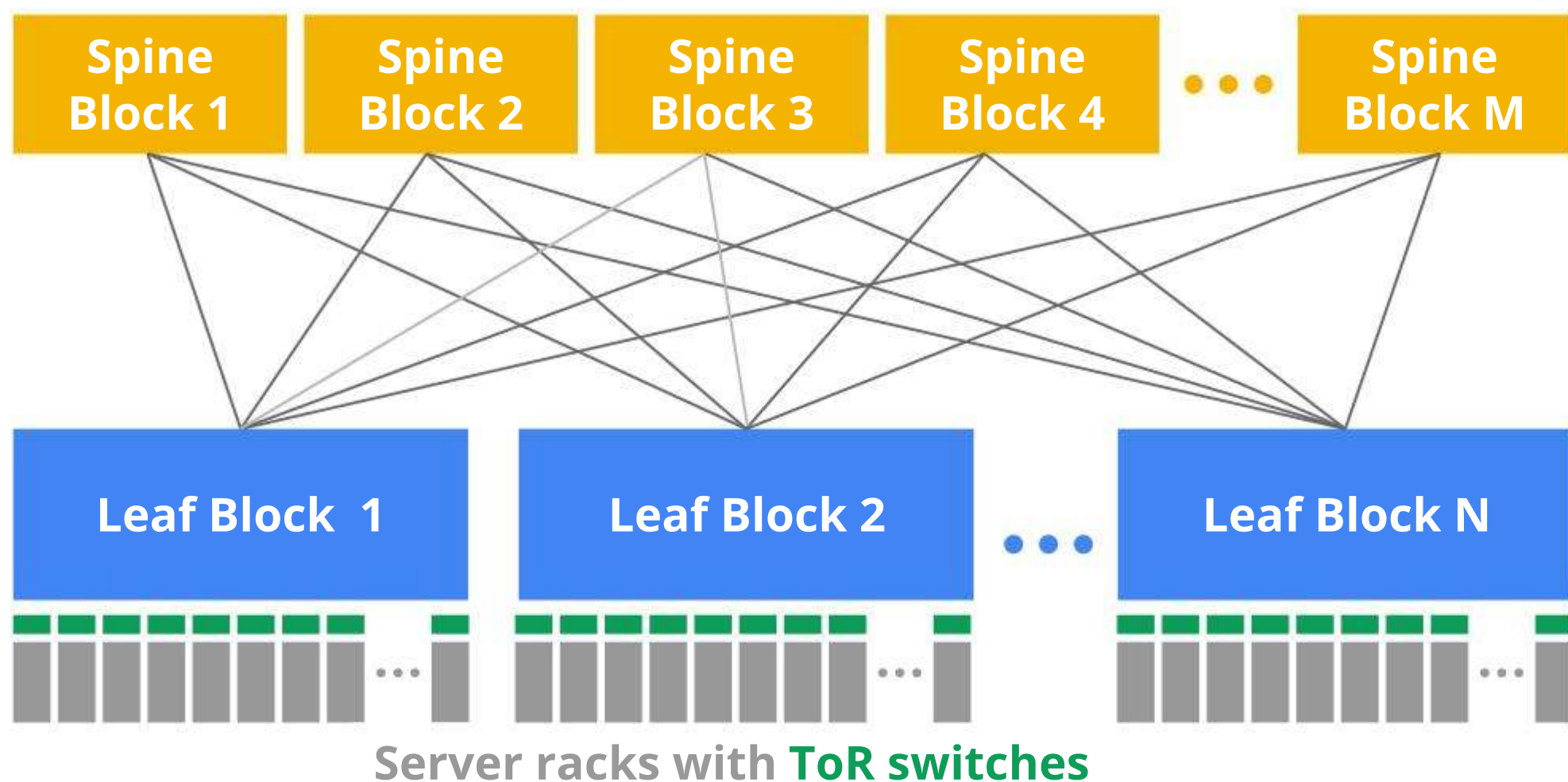
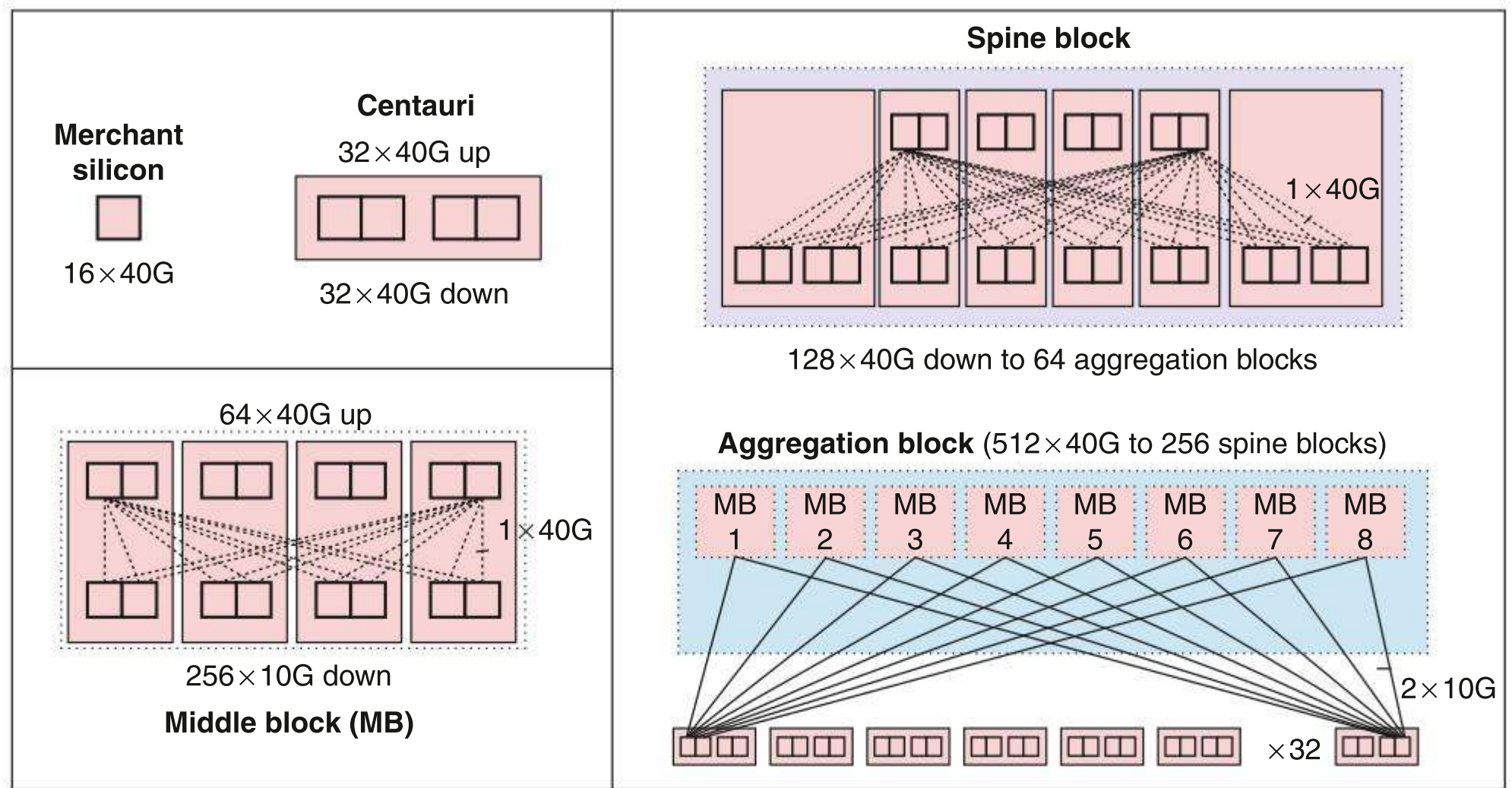


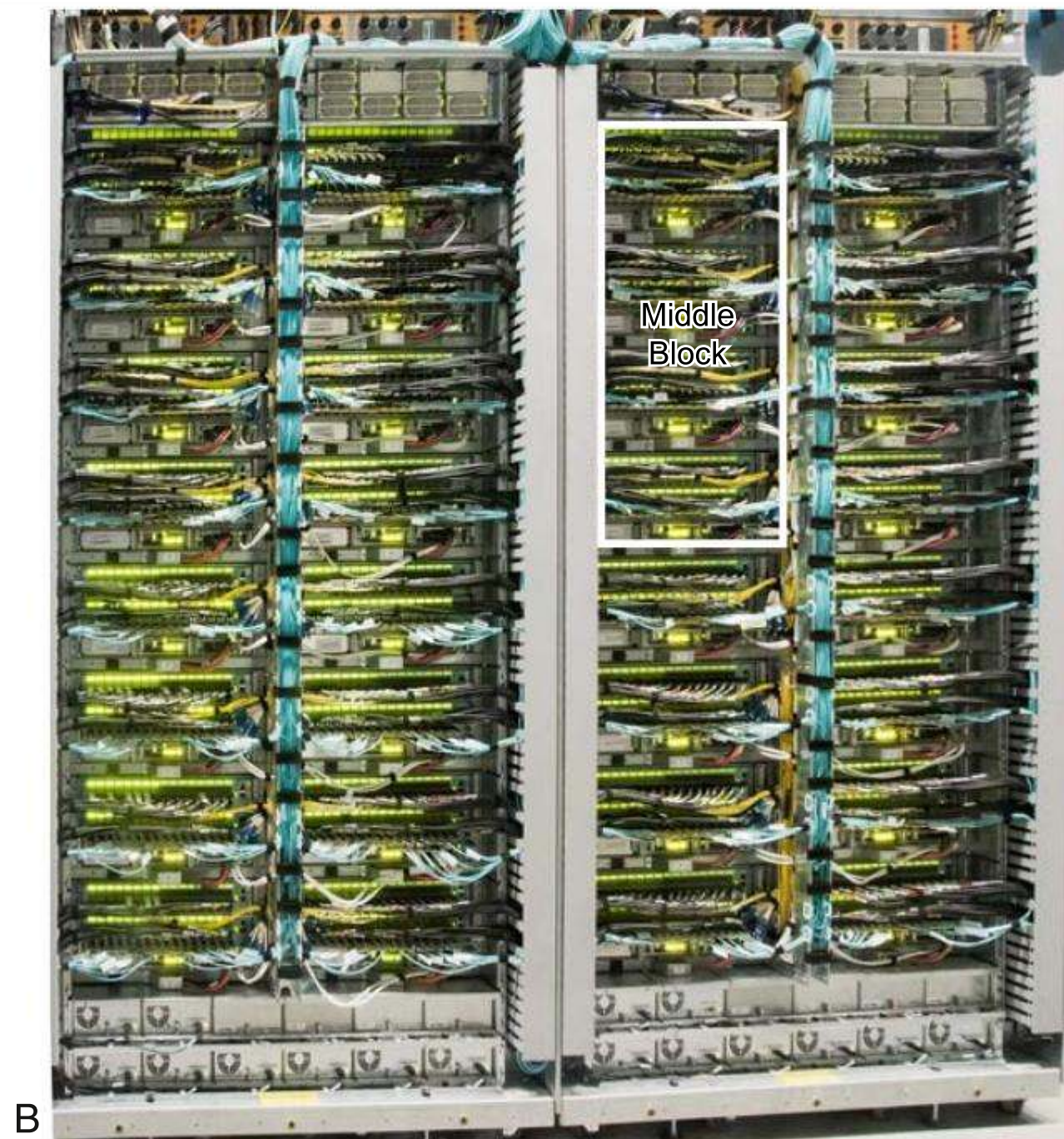
Figure 6.16 A typical Clos network has three logical stages containing crossbar switches: top-of-the-rack (L1), leaf (L2), and spine (L3). Each input to a TOR switch can be routed to any output of any spine switch. The availability of multiple paths to connect any two servers allows traffic scheduling to mitigate congestion in spine and leaf switches.

Clos topologies typically do not offer full bisection bandwidth. In the Jupiter configuration described above, each TOR switch has a 3:1 ratio between down-link and uplink bandwidth, known as a 3:1 *oversubscription ratio*. While any two servers can communicate at the top speed of their links (40 Gbps), if all servers within a rack attempt to transmit to servers in other racks their effective bandwidth will be 1/3. Effective bandwidth is also limited by congestion in leaf and spine switches as various communication paths in the WSC overlap. WSC operators minimize the impact of oversubscription and congestion using various techniques. They colocate within each rack workload that is unlikely to concurrently transmit at full speed toward the network core. They attempt to schedule the processes of distributed workloads as closely as possible so that most of their traffic only goes across one TOR or one leaf switch. Unfortunately, fault tolerance suggests that these processes should be spread out evenly across the WSC to ensure that any link or rack failure has minimal impact on their availability. WSC operators also randomize the path through the multipath Clos network used by any connection in order to spread traffic as evenly as possible and avoid congestion in the core.

The Clos implementation in [Figure 6.17](#) is inherently fault tolerant. Any two TOR switches can communicate through multiple paths through leaf and spine blocks. No single link or chip failure in leaf or spine blocks can disconnect a TOR switch. In some cloud deployments racks include a pair of redundant



A



B

Figure 6.17 The building blocks for Google’s Jupiter Clos network from Figure 13 (a) and 14 (b) in Singh et al. (2015). Figure (a) shows the commodity switch chip for Jupiter, which is a 16-port crossbar chip with 40 Gbps links. The key building block is the 4U Centauri enclosure with 4 crossbar chips and all 64 ports accessible through its front panel. Centauri is used as the TOR (L1) switch, with one possible configuration using 48 downlinks to the servers and 16 uplinks to the network core. The leaf (L2) switch, also known as middle box at Google, consists of four Centauri enclosures in a 2-stage configuration with 64 uplinks and 64 downlinks. Figure (b) shows four middle boxes housed in a rack. The leaf or aggregation block includes 8 leaf switches with 512 downlinks connecting to 32 TOR switches and 512 uplinks connecting to the 256 spine switches. Each spine (L3) switch uses 6 Centauri enclosures in a 2-level topology that offers 128 downlinks to connect with 64 leaf blocks.

TOR switches to shield non—fault-tolerant applications from TOR switch failures or switch maintenance times. Similar Clos topologies can be designed using newer commodity switch chips. For example, the Broadcom Tomahawk 5 chip released in 2022 can function as a 64-port crossbar for 800-Gbps links, 128-port crossbar for 400-Gbps links, or 256-port crossbar for 200-Gbps links. These options provide additional flexibility on how these chips are used. WSC architects may use the higher link speeds within the core of the topology, while within racks they may use 200-Gbps per link in order to maximize the number of server ports in the system.

Example What is the maximum number of servers that the Jupiter topology can support using the building blocks in [Figure 6.17](#)? Assume every server connects to the TOR switch using one 40 Gbps link and that there is a 3-to-1 oversubscription ratio at the TOR level. Also assume that we use a redundant pair of 40 Gbps links to connect each TOR to each middle box, and each aggregation block to each spine. What would be the capacity if we use a 1-to-1 oversubscription ratio at the TOR level?

Answer Let's review the building blocks in [Figure 6.17](#):

- The spine block (L3) exposes 128 40 Gbps downlinks to aggregation blocks. Since each aggregation block connects to every spine block using a pair of redundant links, the maximum number of aggregation blocks in the system is $128/2 = 64$.
- The TOR switch (L1) has 64 40 Gbps ports. A configuration with 3-to-1 oversubscription has 48 downlinks toward servers and 16 uplinks toward the aggregation blocks. These 16 uplinks are sufficient to connect to 8 middle boxes per aggregation block using a pair of redundant links.
- At the aggregation block (L2) level, each middle box has 64 downlinks toward TOR switches. Hence each middle box can connect to 32 TOR switches.

The maximum number of servers in the system is:

$$\begin{aligned}
 &64 \text{ aggregation blocks} \times 32 \text{ TOR switches/aggregation block} \\
 &\quad \times 48 \text{ servers/TOR switch} \\
 &= 98,304 \text{ servers}
 \end{aligned}$$

In practice, the server count is lower as some aggregation blocks are used for external connections to other WSC clusters and the public Internet. However, it is possible to split each 40 Gbps link into four 10 Gbps links, which allows for more servers at lower bandwidth per server.

With a 1-to-1 oversubscription ratio, each TOR has 32 uplinks and 32 downlinks. Assuming the same aggregation block design, each aggregation block

can now accommodate 16 TOR switches, with 4 links connecting each TOR switch to each middle box. The maximum number of servers in the system is:

$$\begin{aligned}
 &64 \text{ aggregation blocks} \times 16 \text{ TOR switches/aggregation block} \\
 &\quad \times 32 \text{ servers/TOR switch} \\
 &= 32,768 \text{ servers}
 \end{aligned}$$

To accommodate more servers, we would have to use more aggregation blocks and a bigger spine block. While oversubscription can lead to congestion when a large number of servers within a rack transmits at high speed, it greatly reduces the cost of the WSC network required to connect a given number of servers.

Data center generation switch	First deployed	Merchant silicon	Top of rack (ToR) switch config	Edge aggregation block	Spine block	Fabric speed	Host speed	Bisection BW
Four-Post CRs	2004	Vendor	48×1 Gbps	—	—	10 Gbps	1 Gbps	2 Tbps
Firehose 1.0	2005	8×10 Gbps 4×10 Gbps (ToR)	2×10 Gbps up 24×1 Gbps down	$2 \times 32 \times 10$ Gbps	32×10 Gbps	10 Gbps	1 Gbps	10 Tbps
Firehose 1.1	2006	8×10 Gbps	4×10 Gbps up 48×1 Gbps down	64×10 Gbps	32×10 Gbps	10 Gbps	1 Gbps	10 Tbps
Watchtower	2008	16×10 Gbps	4×10 Gbps up 48×1 Gbps down	$4 \times 128 \times 10$ Gbps	128×10 Gbps	10 Gbps	$n \times 1$ Gbps	82 Tbps
Saturn	2009	24×10 Gbps	24×10 Gbps	$4 \times 288 \times 10$ Gbps	288×10 Gbps	10 Gbps	$n \times 10$ Gbps	207 Tbps
Jupiter	2012	16×40 Gbps	16×40 Gbps	$8 \times 128 \times 40$ Gbps	128×40 Gbps	10/40 Gbps	$n \times 10$ Gbps/ $n \times 40$ Gbps	1300 Tbps

Figure 6.18 Six generations of network switches deployed at Google WSCs (Singh et al., 2015). The Four-Post CRs used commercial 512-port, 1 Gbit/s Ethernet switches and 48-port, 1 Gbit/s Ethernet top-of-the-rack (ToR) switches, which allowed for 20,000 servers in the WSC. The remaining generations used custom switch designs. The goal of Firehose 1.0 was to deliver 1 Gbps of nonblocking bisection bandwidth to each of 10,000 servers, but it ran into problems with the low connectivity of the ToR switch that caused problems when links failed. Firehose 1.1 was the first custom-designed switch with better connectivity in the ToR switch. Watchtower and Saturn followed in the same footsteps but used new, faster merchant switch chips. Jupiter used 40 Gbps links and switches to deliver more than 1 Pbit/s of bisection bandwidth.

WSC networks include a large number of links; hence cable cost and cabling complexity are significant concerns. At present, links between servers and TOR switches are typically bundles of copper cables. For example, a 200-Gbps link may be 4 cables each operating at 50-Gbps or 2 cables each operating at 100-Gbps. At short distances (1–3 m) and link speeds of 100–400 Gbps, copper cables are practical and cost-effective. The links between L1, L2, and L3 switches are significantly longer (10–100 s of meters) and run at higher speeds, making optical cables the preferred solution. Optical cables are often pre-bundled together to reduce the deployment complexity in a WSC. As servers with GPUs or other DSAs require increasing amounts of bandwidth, copper cables for the server to TOR links are becoming increasingly challenging. We may soon see some of the following technologies in wide deployment in data centers: active copper cables that include repeaters to extend the reach and bandwidth of copper cables or optical transceivers that are copackaged within the compute sockets in order to decrease the cost and power consumption of optical transceivers for the servers to TOR links.

Outfitting a WSC with 100 K servers takes months to years and, once full, the WSC is incrementally upgraded with the latest hardware available. Leaf switches can be added and replaced as needed without affecting the rest of the WSC network. Spine switches, on the other hand, introduce a difficult tradeoff. If spine switches are added incrementally as the WSC is populated, one must perform significant rewiring between leaf and spine switches with each addition, introducing a long period of degraded performance. If all spine switches are installed from the beginning, all wires for future leaf switches can be placed in advance. However, this also means that spine switches will use the oldest technology in the data center. If the Clos network initially uses 100 Gbps links, upgrading some leaf switches to use 200 Gbps links will provide no bandwidth improvements until spine switches are also upgraded. This trade-off has motivated WSC operators to introduce a layer of *optical circuit switches (OCSs)* between the leaf and the spine switches. OCSs are microelectromechanical systems (MEMSs) that can be reprogrammed within milliseconds to change the connections between input and output fibers. OCSs are rate agnostic as they simply reflect light and do not perform packet-level decisions. [Figure 6.19](#) shows how Google used an OCS layer to facilitate the incremental deployment of both leaf and spine blocks in the Jupiter fabric, allowing at each step for the best possible topology for the available leaf switches (Poutievski et al, 2022). All wiring is incremental as leaf and spine switches are added over time.

Google engineers took this idea further, using OCSs as a full replacement for spine switches. Leaf switches connect only to OCSs, which also have direct connections to each other. [Figure 6.20](#) shows how Google programs the connectivity within the OCSs to configure the desired topology and connectivity rates between leaf switches (Poutievski et al, 2022). Note that a pair of new leaf switches with 200 Gbps links can now communicate directly at full speed, even though communication to all other, older leaf switches must still be derated to 100 Gbps. This is

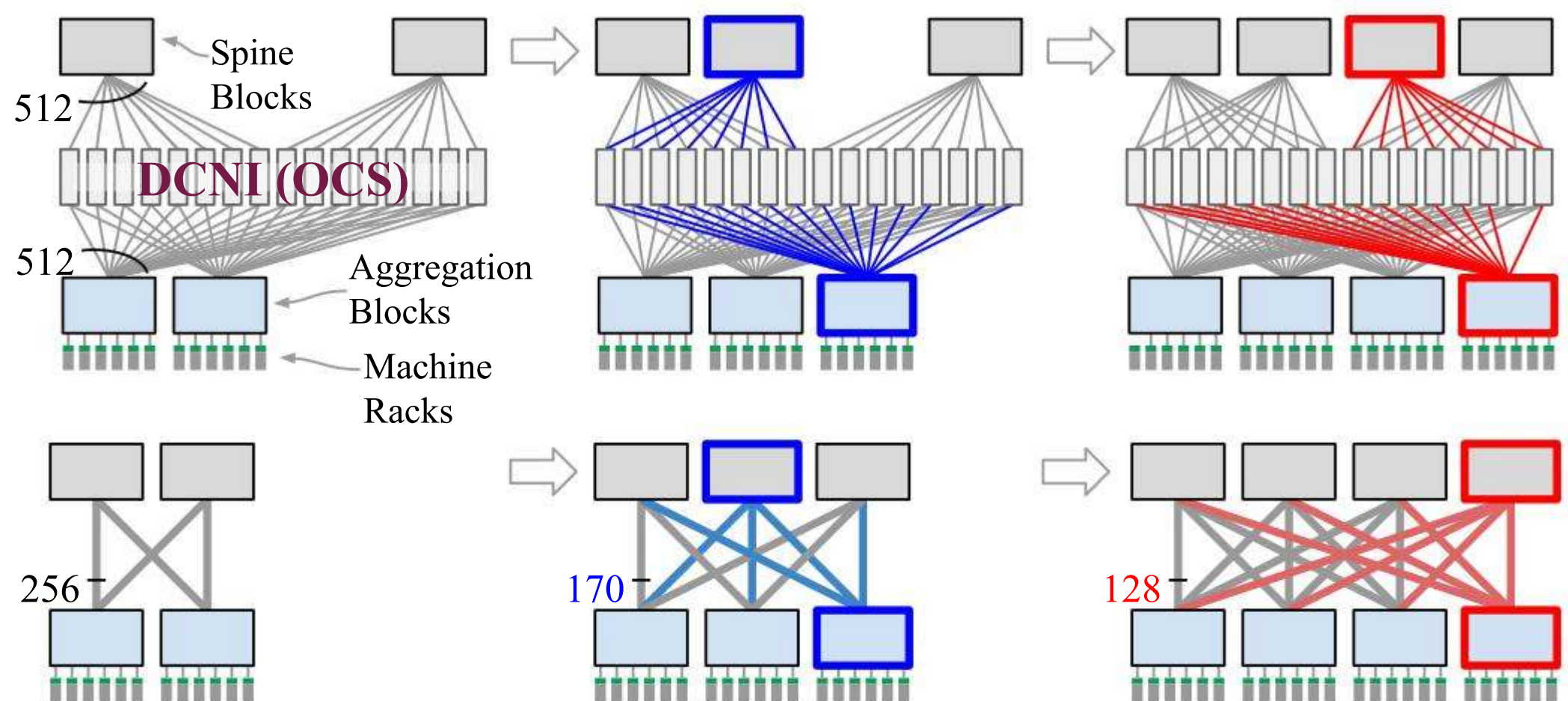


Figure 6.19 The use of optical circuit switches in Google's WSC networks from Figure 2 in (Poutievski et al, 2022). The top diagrams show how an optical switching layer (DCNI) enables incremental expansion of the Jupiter topology from two to four spine blocks while achieving full burst bandwidth among the aggregation (leaf) blocks. The evolution of the logical topology is shown in the bottom diagrams. The DCNI layer is implemented using optical circuit switches (OCSs) and allows incremental rewiring of the fabric as new blocks are added. The newly introduced leaf and spine blocks can use the latest available link speeds, while any communication path that includes an older switch will use the lower link speeds.

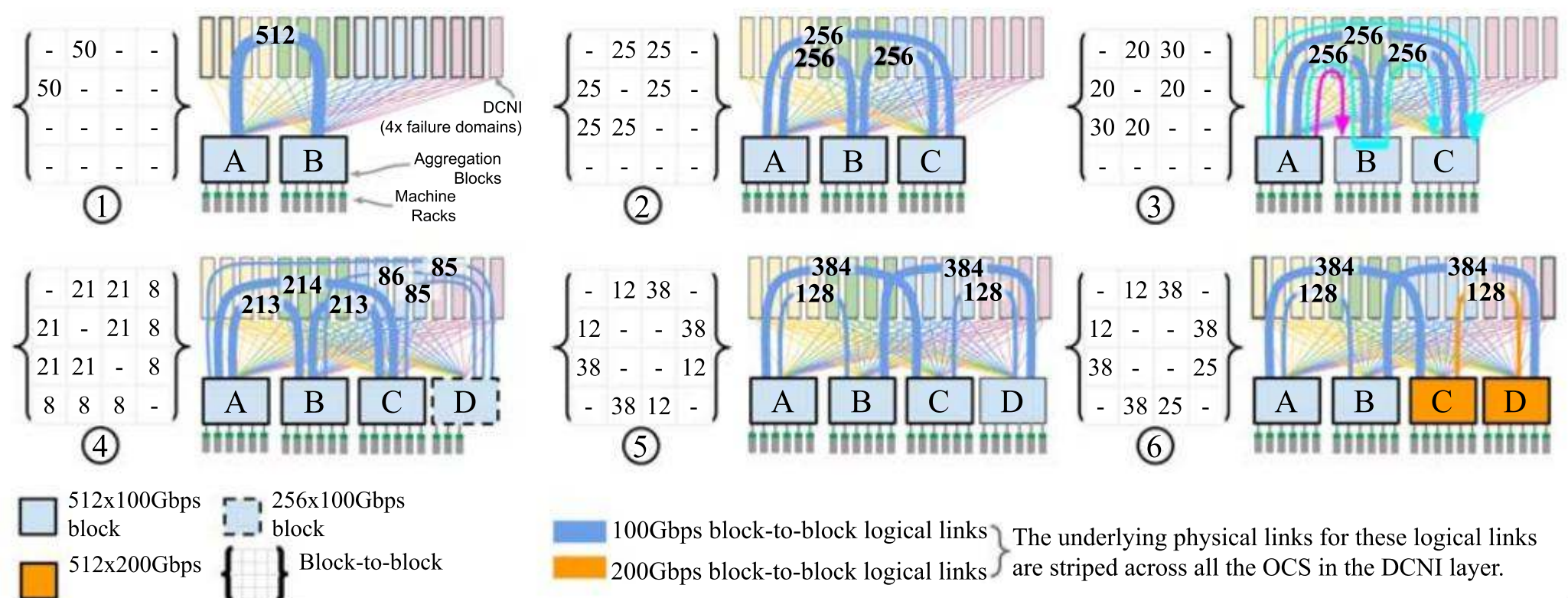


Figure 6.20 The use of optical circuit switches as spine switch replacement in Google's WSCs from Figure 5 in (Poutievski et al, 2022). In addition to enabling incremental deployment, the OCSs allow for direct-connect topologies that are dynamically configured based on traffic patterns. (1) Initially, Aggregation Blocks A, B are added with 512 uplinks each. (2) Block C is added with 512 uplinks. Each block has 50T outgoing demand, uniformly distributed across other blocks. Topology engineering forms a uniform mesh topology to match the uniform steady-state demand matrix. (3) Traffic Engineering (TE) adjusts Weighted Cost Multipath (WCMP) weights based on finer-grained version of demand in (2): A sends all traffic (20T) to B directly (pink) and splits traffic to C (cyan) 5:1 (25T:5T) between direct and indirect paths (via B). (4) Block D is added with 256 uplinks (only a subset of machine racks in D are populated). (5) Block D is augmented to 512 uplinks. (6) Blocks C, D are refreshed to 200 G link speed.

possible because, unlike spine switches, OCSs are rate agnostic; they merely reflect light. At a high level, the spine layer in a Clos topology provides a way to spread and balance traffic across all links in the leaf block that contains the receiving node. This spreading can handle many traffic patterns well, even worst-case all-to-all permutations, at the cost of a large number of optical transceivers for the spine switches. Google engineers observed that their network traffic patterns are relatively predictable and far from the worse-case permutation patterns that the spine is designed for. Instead of spreading traffic, the OCS layer allows them to do *topology engineering*, programming the connectivity and amount of bandwidth between leaf switches that are needed given current application patterns on a daily basis.

WSC networks necessitated the development of control plane systems that can handle configuration, routing, management, and real-time control of all network devices. Unlike the Internet that relies on distributed and independent management, WSC control plane takes advantage of the logically centralized view of the network state to optimize for performance, scalability, safety, and liveness beyond what is possible by considering one network connection at the time (Ferguson et al 2021). The control plane decisions are actuated by the CPU-based controllers associated with each switch device at all levels of the network. WSC networks have also led to rapid evolution in the transport protocols used by communicating applications. Since WSC networks are homogeneous and centrally controlled systems, it is possible to extract congestion and telemetry information from switches or accurately measure round-trip times in order to build better congestion control algorithms. It is also possible to deploy *remote direct memory access (RDMA)* protocols for low latency and low overhead communication.

The Programmer's View of a WSC

The WSC architecture has implications for workloads of all types. Even single VM workloads need to understand the implications of accessing storage from remote servers. Distributed or *cloud-native* workloads must be optimized to the performance and reliability characteristics of large machine counts and networks in a WSC.

Programmers understand the WSC memory hierarchy. [Figure 6.21](#) shows the latency, bandwidth, and capacity of memory hierarchy inside a WSC, and [Figure 6.22](#) shows the same data visually. These figures are based on the following assumptions:

- Combined, the two sockets in the server have 16 memory channels populated with 32 GB DDR4-3200 memory modules. The total memory capacity is 512 GB, the access latency is 100 ns, and peak memory bandwidth across all channels is 409.6 GB/s.

	Local	Rack	WSC Row
DRAM latency (μ s)	0.1	100.0	200.0
Flash latency (μ s)	100.0	200.0	300.0
Disk latency (μ s)	5,000.0	5,500.0	6,000.0
DRAM bandwidth (MB/s)	409,600	12,500	2,500
Flash bandwidth (MB/s)	6,900	12,500	2,500
Disk bandwidth (MB/s)	300	12,500	2,500
DRAM capacity (GB)	512	24,500	735,000
Flash capacity (GB)	3,840	184,320	5,500,000
Disk capacity (GB)	20,000	960,000	29,800,000

Figure 6.21 Latency, bandwidth, and capacity of the memory hierarchy of a WSC. Figure 6.22 plots this same information.

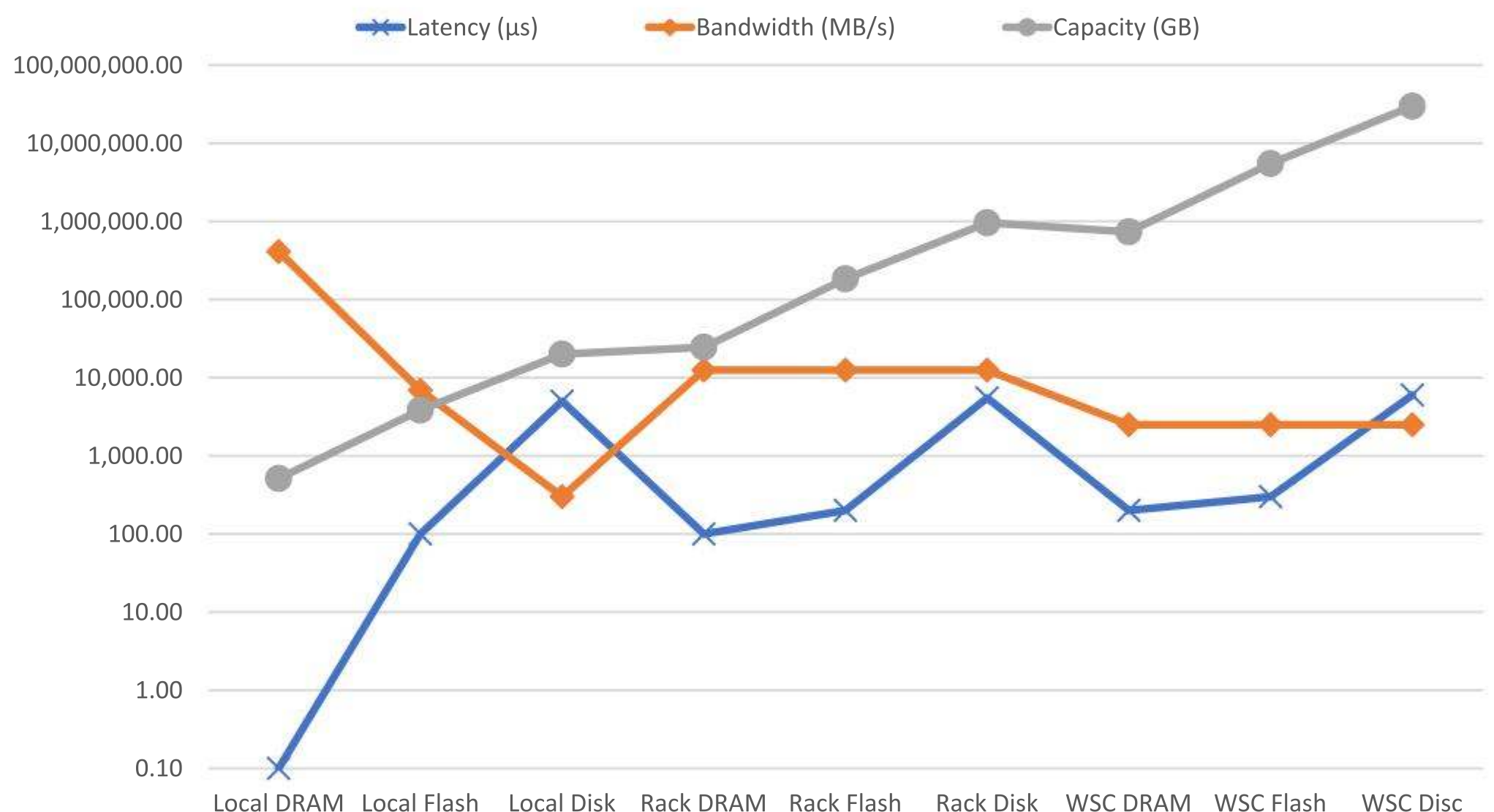


Figure 6.22 Graph of latency, bandwidth, and capacity of the memory hierarchy of a WSC for data in Figure 6.21.

- The server contains one 3.84 TB NVMe Flash drive with maximum read bandwidth of 6.9 GB/s and unloaded read latency of 100 μ s. Finally, the server includes a 20 TB hard disk that offers a 5 ms average access time and sustained transfer rate of 300 MB/s. The two CPU sockets share one 100 Gbps Ethernet NIC.
- In this example a rack includes 48 servers. NICs, switches, and networking software increase access to DRAM and Flash accesses by 100 μ s and to disk by 500 μ s. The 100 Gbps NICs limit the bandwidth to remote receives

to 12.5 GB/s, but data can be sourced from multiple devices on the rack. The total storage capacity of the rack is roughly 24.5 TB of DRAM, 184.3 TB of Flash, and 960 TB of disk storage.

- The WSC row is 30 racks, so storage capacity of an array goes up by a factor of 30: 735 TB of DRAM, 5.5 PB of Flash, and 29.8 PB of disk. The networking gear in the multistage network and potential congestion increase latency to DRAM within an array by 200 μ s, by 300 μ s for Flash, and by 1 ms for disk. Oversubscription limits the bandwidth available to each server for the remote access to DRAM, array Flash, or array disk to 20 Gbps or 2.5 GB/s.

Figures 6.21 and 6.22 show that for hard disks and Flash devices the overall latency does not increase significantly when switching from local to remote accesses as the device latency is significantly higher than the additional network latency. If data are sourced from multiple remote disks, remote disk bandwidth is significantly higher than local disk bandwidth. Flash bandwidth is higher within a rack and lower across the row. These observations have motivated the separation of compute from storage in WSCs, which has great advantages for scalability and reliability. Many distributed applications initiate stateless compute tasks that are scheduled across the WSC and rely on distributed (remote) storage for the input and outputs of these tasks. Compute is easy to scale. The distributed storage system uses *sharding* or *partitioning* of data across storage nodes to achieve the capacity, bandwidth, and reliability needed. Both compute and storage can scale independently and benefit from parallelism in a large WSC.

In contrast, Figures 6.21 and 6.22 show that networking overheads dramatically increase the latency to remote DRAM, within a rack or within a row. Still, both remote DRAM and remote Flash are more than 10 times faster to access than a local disk. The network collapses the difference in bandwidth between rack DRAM, Flash, and disk and between array DRAM, Flash, and disk. These observations have motivated many WSC applications to use distributed key-value stores (KVS), based on DRAM or Flash, for fast access to massive amounts of data. For example, the Facebook social network used the DRAM-based, Memcache KVS to serve hundreds of millions per user each day at interactive latencies (Nishtala et al., 2013).

The dramatic latency increase from local to remote DRAM is motivating the deployment of new WSC technologies. RDMA is networking technology that eliminates the overheads and unpredictability of networking software. Most of the network overheads are either due to congestion in the fabric due to oversubscription or due to the latency of networking software. The network switches themselves have cut-through latency of 200–500 ns per hop. Using networking protocols such as Infiniband and RoCE that are implemented in the NIC hardware, RDMA can reduce remote memory access latency to a few microseconds. The emerging CXL switched fabrics are also promising to allow fast memory sharing (disaggregation) within WSCs.

Example What is the average memory latency assuming that 90% of accesses are local to the server, 9% are outside the server but within the rack, and 1% are outside the rack but within the row?

Answer The AMAT is

$$(90\% \times 0.1) + (9\% \times 100) + (1\% \times 200) = 0.09 + 9 + 2 = 11.09 \mu\text{s}$$

or a factor of 100 slowdown versus 100% local accesses. Clearly, locality of access within a server is vital for WSC performance.

Example How long does it take to transfer 1000 MB between disks within the server, between servers in the rack, and between servers in different racks in the row? How much faster is it to transfer 1000 MB between DRAM in the three cases?

Answer A 1000 MB transfer between disks takes

$$\text{Within server} = 1000/300 = 3.3 \text{ s}$$

$$\text{Within rack} = 1000/300 = 3.3 \text{ s}$$

$$\text{Within row} = 1000/300 = 3.3 \text{ s}$$

In the rack and row transfers the bottleneck for bandwidth is the disks and not the network.

A memory-to-memory block transfer takes

$$\text{Within server} = 1000/409,600 = 0.024 \text{ s}$$

$$\text{Within rack} = 1000/12,500 = 0.08 \text{ s}$$

$$\text{Within row} = 1000/2,500 = 0.4 \text{ s}$$

Thus, for large block transfers outside a single server, it doesn't matter whether the data are in memory or on disk. Given the network constraints, what matters is the degree of parallelism used to best utilize the network. Multiple remote disks are competitive to remote memory for large transfers.

6.5

The Architecture of High-Performance I/O Devices

We will now take a closer look at the architecture of NICs and Flash storage devices, two key I/O components for WSC networking and storage, respectively. Many of the hardware and software optimizations used to improve their bandwidth and latency are applicable to other I/O devices connected to CPU hosts or directly to DSAs.

Network Interfaces

Figure 6.23 shows the block diagram of a simple network interface (NIC) with 2 Ethernet ports, each running at 100 Gbps. The NIC includes an Ethernet medium access control (MAC) block and a set of transmit/receive queues for each port. The MAC block implements in hardware the physical layer (encoding and signaling) and the data link layer (low-level bit transmission/reception) of the network stack. All remaining layers can be implemented in software on the host, including the IP and TCP protocols commonly used for networking across the Internet. As we discuss next, modern NICs include full or partial accelerators for some of these software tasks. Software interacts with the NIC through an Ethernet device driver. The transmit and receive queues decouple the MAC layer and the software stack in both directions. At a minimum, they need to store two Ethernet packets each, where a packet can be up to 9,000 bytes long. In practice, NIC on-chip buffers are larger.

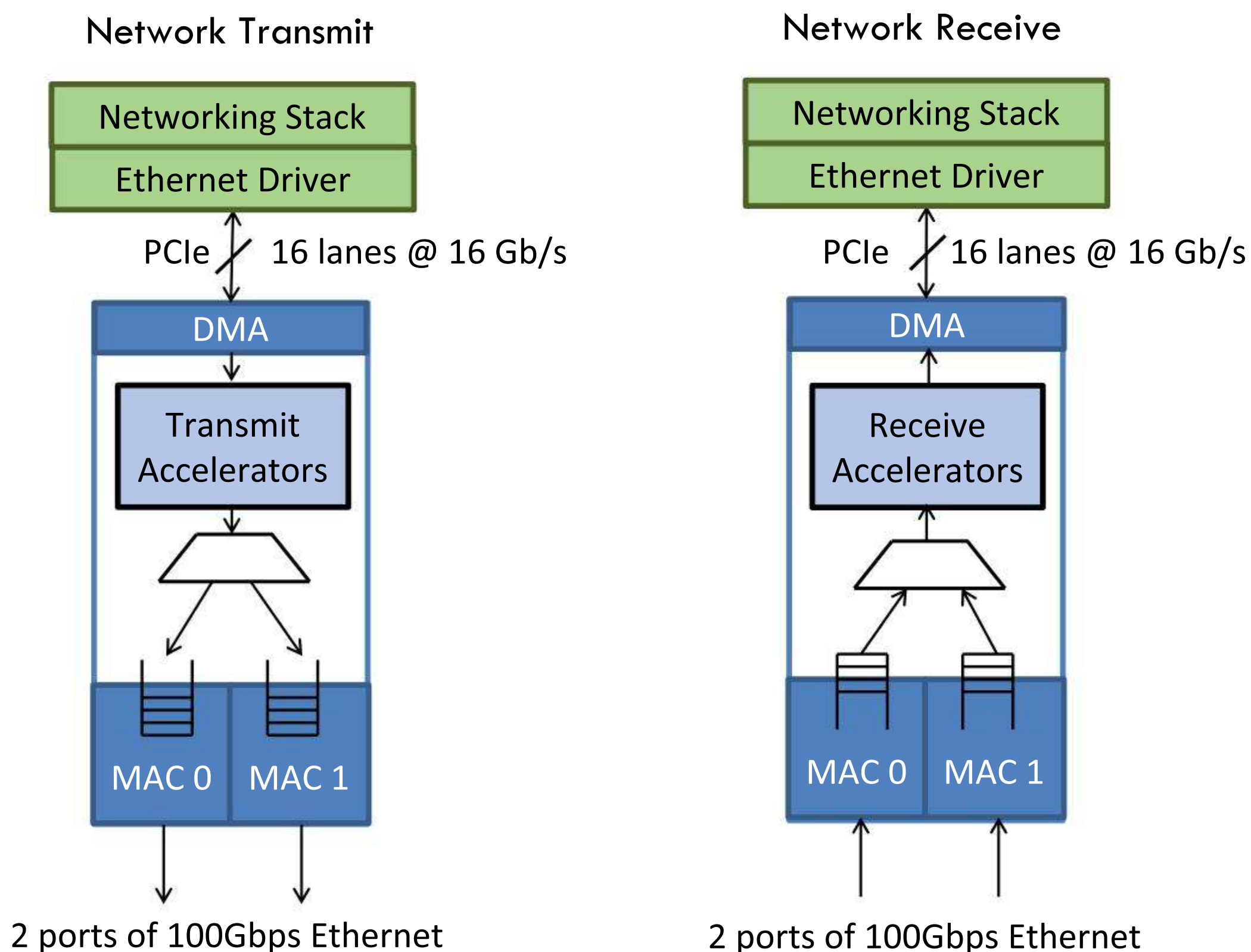


Figure 6.23 A simplified view of the transmit and receive paths of a network interface (NIC) with 2 Ethernet ports.

The following steps are involved with transmitting a packet over the network. The user-level software prepares the outgoing data in a memory buffer and uses a system call like *write()* or *send()* to transmit the data over a preestablished connection, such as a TCP/IP socket. The OS likely copies the data into its own memory buffer and executes a series of networking protocols such as the TCP and IP. Each protocol appends header and trailer information such as source and destination addresses, sequence numbers, and error-correcting information. When the packet is ready for transmission, the OS device driver will communicate with the NIC using two FIFO data structures: the *send queue* (SQ) and the *completion queue* (CQ). Both are implemented as circular buffers in host memory using head and tail pointers. The OS driver writes a descriptor into the SQ that includes the packet address in memory and its destination Ethernet port and then increments the SQ tail pointer. Next, the driver notifies the NIC by ringing a *doorbell*, which is done by writing to a NIC register that is memory mapped to the host's physical address space. At this point, the NIC accesses the first packet descriptor in the SQ, increments the head pointer, and configures a direct memory access (DMA) engine to copy the packet from the host's memory into the transmit buffer for the selected Ethernet port. The Ethernet MAC drains its buffer and transmits the packet over the link at a speed negotiated with the upstream Ethernet switch, for example at 100 Gbps. When transmission is complete, the NIC writes the packet descriptor in the CQ, increments the CQ tail pointer, and generates an interrupt for the host OS. When the interrupt is served, the OS may decide to deallocate the memory buffer for the packet. For some protocols, such as TCP, the OS may wait for a reception acknowledgment from the recipient before the buffer is deallocated. If there is a transmission failure, the packet may be retransmitted.

The following steps are involved with receiving a packet over the network. The NIC and the OS driver communicate using a pair of FIFO data structures, the *receive queue* (RQ) and the *completion queue* (CQ). Both are implemented as circular buffers in host memory using head and tail pointers. The receive queue is periodically populated with the addresses of memory buffers allocated by the OS for packet reception. When a packet arrives, the NIC reads a buffer address from the RQ and increments the RQ head pointer. Next, it programs the DMA engine to copy the packet from the NIC receive queue to the OS buffer. When the DMA transfer completes, the NIC frees the space in the NIC receive queue. It also writes a packet descriptor in the CQ and updates the CQ tail pointer. Finally, it generates an interrupt for the host OS. The OS will at some point process the packet using a sequence of protocols, such as IP and TCP, ensure the packet is correctly received using error codes, and remove all headers and trailers. Next, it copies the data to a user-level buffer for the application that is listening to this connection. Finally, the OS deallocates or recycles its own buffers used by the packet and potentially generates an acknowledgment message that notifies the sender that the packet has been received with or without errors.

The simplified transmit and receive processes just summarized have several inefficiencies that lead to low bandwidth and high latency networking. The first

challenge is the significant CPU overhead for network protocol processing. NICs partially alleviate this challenge by including what are known as *stateless offloads*, which are hardware accelerators for network protocol tasks. The most common offloads are those for generating error detection codes at transmission and checking them at reception for common network protocols, such as Ethernet, IP, TCP, and UDP. The error codes are generated and checked by the NIC on the fly as a packet is transmitted or received and communicated to the host OS. NICs also include *large segment offloads*, which provide logic that, at the sender, splits a large message into the smaller packets supported by the network and, at the receiver, reassembles the large message from the smaller packets, even if they arrive out of order. Hence transmitting and receiving a 1 MB chunk involves the networking protocols 1 time instead of $1 \text{ MB}/9000\text{B} = 112$ times. Large segment offloads require larger buffers on the NIC and are complicated as the Ethernet packets may be reordered or occasionally dropped as they cross the WSC network fabric.

A second challenge is the overhead of interrupt processing. A 100 Gbps NIC receiving minimum-sized Ethernet packets (64B) can generate 200 M interrupts per second. To avoid overwhelming the CPU, NICs implement *interrupt moderation*, where the rate of interrupts is capped, and each interrupt may signal the arrival of multiple packets. NICs also include interrupt steering logic that selects which CPU core should receive the interrupt. A simple option is to interrupt a single core for all packets so that the core has warm instruction and data caches for network processing. At high networking rates, this core will become the bottleneck, so NICs also implement mechanisms that can spread the interrupts across multiple cores by hashing the connection information for each packet (*receive side scaling*). This hashing ensures that a single core will receive all interrupts for one connection and synchronization overheads will be minimized. It is also important to note that once the OS is done with the network processing, an additional interrupt may be needed to schedule the application process that is waiting for the incoming packet. Hence, some NICs include logic that tracks which core has recently generated send system calls for each active connection. When a packet arrives for the same connection, they interrupt the associated core. If the core is still running the application that established this network connection, the additional interrupt is not needed. The OS and applications can avoid interrupts altogether by polling the NIC for incoming network packets. Some operating systems use a mixed approach: when a NIC interrupt causes them to run code from their networking subsystem, they stay in polling mode for a while to amortize the interrupt cost.

A third challenge involves memory and caching overheads. Sending and receiving packets requires multiple copies between NICs, OS buffers, and application buffers. In an unoptimized system the memory bandwidth consumed is 5–8 times higher than the network bandwidth. A first optimization is *data direct I/O (DDIO)*, where incoming packets are placed directly in the last-level cache of the CPU. If packets are processed quickly, DDIO speeds up network processing

and reduces the memory bandwidth used. If not, the packets are gradually evicted to main memory. A second optimization is *header splitting*, where the packet metadata are placed in a separate buffer from the packet data. Splitting improves locality in network processing as the OS typically handles only the headers, while the data are processed by the applications themselves. It is also necessary for *zero-copy networking*, where the payload is copied directly from the NIC into an application buffer, eliminating redundant copies through OS buffers. Zero-copy networking requires that the receiving application has preposted the addresses of buffers where it can receive incoming data. Zero-copy is complicated in the presence of packet reorderings and drops.

The majority of NICs used in WSCs are Ethernet devices that implement several of the optimizations discussed previously. In the last few years, there have been two emerging trends for NICs. First, several clusters optimized for ML use *Infiniband (IB)* or *RDMA over Converged Ethernet (RoCE)* networking devices. RoCE is an implementation of the InfiniBand protocol over an Ethernet fabric. IB and RoCE offload all network protocol processing to the NIC and implement zero-copy networking to application buffers on CPU memory or GPU memory. The second trend is the deployment of SmartNICs, which integrate programmable cores that can offload a wide range of tasks, including network protocol processing, remote storage protocol processing, and various hypervisor tasks. NICs used in public cloud frameworks also include hardware blocks for network virtualization, that is, programmable logic that can add and remove the layers of headers needed to support private virtual networks and enforce security rules in a shared WSC (Firestone et al., 2018).

NVMe Flash Drives

Nonvolatile Memory Express (NVMe) drives are widely used in WSCs to implement storage systems that are optimized for high throughput and low latency. Hard disk drives are still popular for storage systems optimized for low cost per stored bit.

Figure 6.24 shows the organization of an NVMe drive. The key component is a package of vertically stacked Flash memory chips (100+ chips per stack) that acts as a single, high-density chip. Multiples of such packages are organized in channels for parallelism. Each channel can support one access at a time, but the device controller can overlap accesses across channels. The access granularity for a package is a *page* of 1–8 KB, including some spare storage for error coding. Reading a page takes a few tens of microseconds. However, writing a page is both slower and more complex. Pages are organized in *blocks* of 64–256 pages. The contents of a block are first reset using an *erase* operation that takes several milliseconds. Then, pages can be written in sequential order (0, 1, 2, ...) at the cost of one or two milliseconds per page write. The low speed of the write process is due to the nature of the Flash memory cell (state captured as charge on a floating gate), the frequent use of multilevel cells (MLCs) to increase Flash capacity, and the structure of a Flash cell array (NAND arrangement). Finally, Flash blocks have

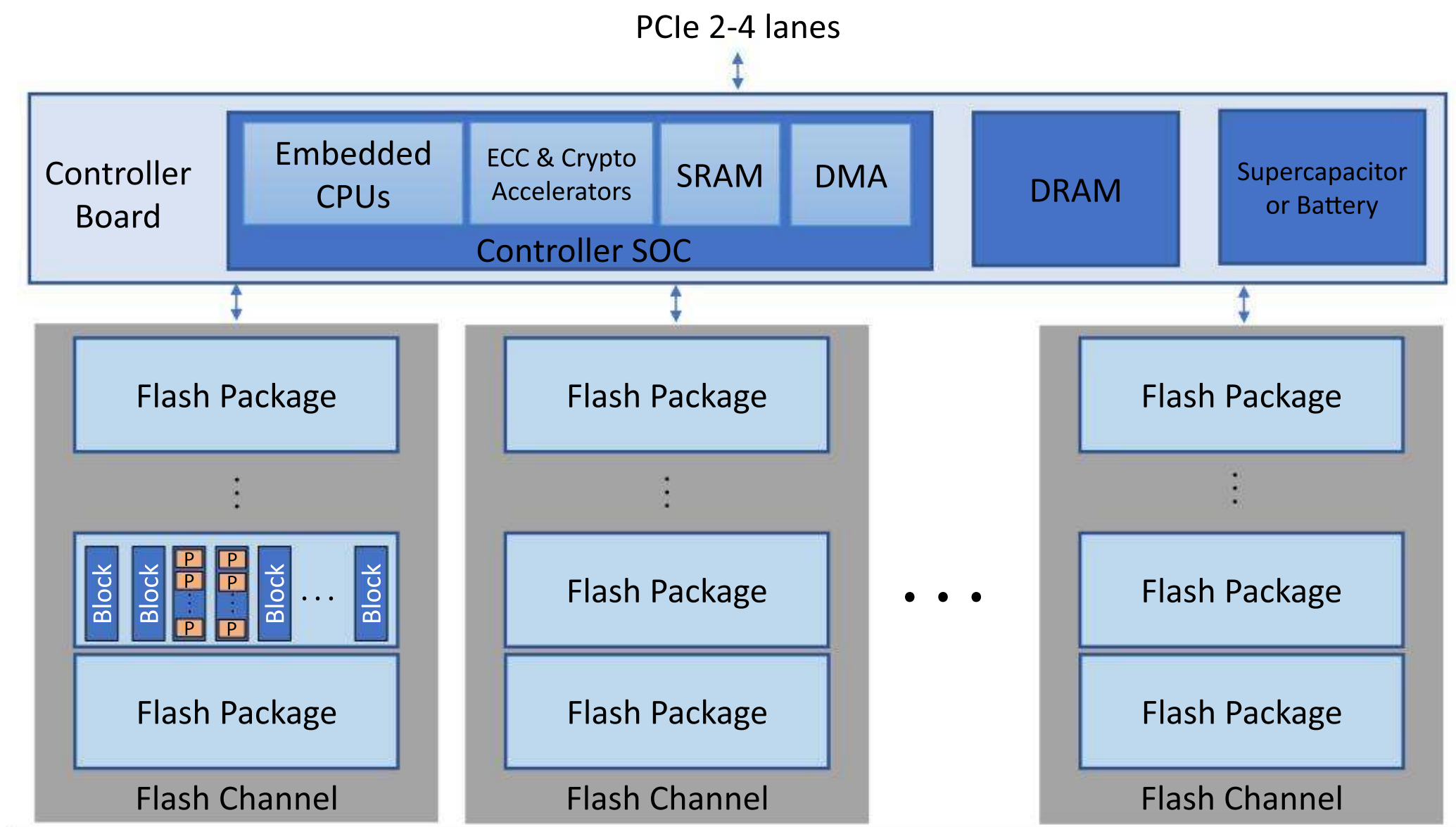


Figure 6.24 The block diagram of an NVMe Flash device with a PCIe interface. NVMe devices are built at varying capacities, interface speeds, and form factors (M.2, U.2, AIC, etc.). There are also NVMe drives that use lower-bandwidth disk drive interfaces like SAS and SATA.

limited *endurance*. Each block can be erased 100 K to 1 M times before it no longer functions properly as memory.

These device characteristics make it difficult for operating systems to use Flash devices directly. Hence, the NVMe controller implements a *Flash translation layer (FTL)* that translates block storage read and write accesses to Flash read, erase, and write operations and evenly spreads the *wear-out* from writes across all available blocks. Figure 6.25 shows the key data structures that the FTL maintains. The first structure is a map that associates *logical block addresses (LBAs)* visible to the host OS to Flash device addresses including the channel, package, block, and page numbers. The map is accessed on every read to identify the physical page to access. The second structure tracks the state of every block on the device, including its health, if it is erased, and the number of valid pages it stores. Finally, there is a pointer to the current *write point* that identifies the block that currently receives writes and the next available page within this block. On write accesses to a logical block, the controller writes the new data in at the current write point and updates the map structure to indicate the new association between a logical block and a physical page. Finally, it updates the information for the Flash block that is used to store the data for this logical block by decrementing the count of valid

Block Map		Block Information Table				
LBA	Physical Page Address <Channel, Package, Block, Page>	Block	Erased	Erase Count	Valid Page Count	Bad Block Indicator
0	<2, 4, 59, 3>	0	False	15	15	False
1	<2, 5, 12, 8>	1	True	33	0	False
...	...	...	...	...	...	...
8k	<4, 0, 33, 129>	2	False	1	4	False

Write point	
	Physical Page Address <Channel, Package, Block, Page>
Write point	<3, 1, 12, 24>

Figure 6.25 The key data structures for the FTL in a Flash NVMe device. All data structures are stored in DRAM memory on the NVMe device and periodically saved in Flash memory for nonvolatility. The block information table may include additional information for each block.

pages. The controller evens wear-out by selecting as the next write point a Flash block with a low erase count.

In the background, the NVMe controller periodically scans the block info table and decides whether it should garbage collect any blocks with low valid page counts. Any valid pages from a garbage collected block will be copied to the write point and the empty block will be scheduled for erasing. While this compaction step is critical for utilizing all available capacity, it leads to a write amplification problem that increases wear-out and delays reads blocked behind these additional writes. There are several tradeoffs and different algorithms tuning garbage collection and write point selection for various access patterns and objectives. The FTL is typically implemented as firmware on one or more embedded cores in the NVMe device. The data structures in [Figure 6.25](#) are stored in DRAM on the Flash board and are periodically saved on Flash memory for nonvolatility. NVMe devices typically include a supercapacitor so that the controller can quickly save all data structures to Flash when power is lost. The FTL also implements scheduling logic for pending accesses and DRAM-based caching of frequently accessed data.

The interface between NVMe devices and the software running on the host CPU is similar to that of NIC devices. It is implemented over the PCIe interface and uses command and completion queue pairs. Read and write commands issued to an NVMe device have variable latency and may complete out-of-order. Commands may be queued and scheduled by the controller. Some read commands complete quickly, while others may be delayed for milliseconds because of pending writes or garbage collection tasks in the corresponding Flash blocks,

packages, or channels. The typical characteristics of an NVMe device are capacity of 0.5–10 TB, unloaded read latency of 50–100 microseconds, throughput of 1,000,000–2,000,000 operations per second, and power consumption of 5–15 W. For comparison, a typical disk has capacity of 1–25 TB, unloaded read latency of 5000–10,000 microseconds, throughput of 100–200 operations per second, and power consumption of 5–10 W.

I/O Device Virtualization

As we described in [Section 6.3](#), I/O devices can be virtualized by having the hypervisor intercept all I/O requests by guest VMs, either through paravirtualization or by trapping on loads and stores to memory-mapped I/O addresses. High-performance I/O devices such as NICs and NVMe drives are now designed to allow direct I/O access by guest VMs without the hypervisor’s help.

The key technique from the device side is virtualization of the PCIe interface. Following the *single-rooted I/O virtualization (SR-IOV)* extensions to the PCIe standard, I/O devices implement multiple interface functions. The single *physical function (PF)* includes the registers necessary to manage all functionality of the I/O device. In a nonvirtualized system, the single OS runs a device driver that uses the PF for I/O operations. In a virtualized system, the hypervisor uses the PF and maps one of the many *virtual function (VF)* interfaces implemented by the device into the guest physical address of each VM guest for its OS to use. Each VF includes all the registers necessary to implement the basic operations of the device, send and receive packets for NICs, and read and write blocks for NVMe drives. The VF interfaces do not expose management features such as enabling/disabling VFs, configuring the storage block ranges each VF can access, or setting the fraction of the network bandwidth each VF can use.

From the host side, the key technique for direct VM access to I/O devices is the IOMMU, which we described in [Section 6.3](#). The IOMMU is configured by the hypervisor to translate the DMA address used for each PF/VF in the I/O devices and properly remap interrupts so that a guest VM can directly access VFs.

Within each PF/VF, an I/O device typically supports multiple queue pairs for networking or storage accesses. These queue pairs can be used to further separate and isolate I/O requests from multiple applications within a guest VM. For networking, for example, using a separate queue pair for each application allows the OS to control how much bandwidth each application will use and potentially deliver incoming data directly into the application’s virtual address space.

6.6

WSC Power Distribution and Cooling

The critical power of a state-of-the-art WSC can exceed 100 MW, which is roughly the power consumption of 75,000 houses in the United States. Hence it is important to understand how power is delivered and how heat is removed from WSC facilities. As it turns out, optimizing power delivery and cooling

systems and codesigning them with the WSC hardware and software has been an important factor in making WSCs cost-effective, despite their massive size.

The simple but powerful metric to evaluate the efficiency of the WSC infrastructure for power delivery and cooling is called *power utilization effectiveness* (or *PUE*):

$$\text{PUE} = (\text{Total facility power}) / (\text{IT equipment power})$$

IT equipment power is the power consumption by the racks that host compute, storage, and networking equipment. The total facility power includes the power delivery and cooling overheads: power delivery introduces losses due to conversions, while cooling consumes power in equipment like air conditioning units. PUE is greater than or equal to 1, and the bigger the PUE, the less efficient the WSC power delivery and cooling is.

The Uptime Institute has been tracking the average PUE for large-scale facilities for fifteen years (Uptime Institute, 2022). In 2007, the average PUE was 2.5, which means that for every 1 W delivered to IT equipment, power delivery and cooling required 1.5 W of overhead, with cooling being responsible for 2/3 or more of this overhead. The high overheads are problematic in three ways. First, they increase the operational costs for the WSC facility, specifically the electricity bill. Infrastructure costs for power distribution and cooling are also the majority of the construction costs of a WSC facility. Second, power overheads limit the WSC scale and the amount of IT equipment it can host given the overall power available to the facility. Finally, energy waste is bad for the environment, especially if the power generation sources are not renewable (see [Section 7.8](#) in the next chapter). By 2021, the average PUE was reported to be as low as 1.57. The largest WSCs are significantly more efficient. [Figure 6.26](#) shows the average



Figure 6.26 The PUE data for all Google WSCs, as reported at <https://www.google.com/about/datacenters/efficiency/>.

PUE for Google WSCs, which is roughly at 1.10 since 2020. In the rest of this section, we will review how power delivery and cooling have been gradually optimized down to a 10% overhead. The PUE is not constant and varies for each WSC throughout the year as the environmental conditions around it change and impact the efficiency of its cooling system.

WSC Power Delivery

Figure 6.27 presents an example of WSC power delivery infrastructure and the many conversions between the utility power lines and the servers. For redundancy, there are typically a couple of external power sources that deliver energy over high-voltage lines (115 KV AC). Utility substations reduce voltage to between 10 and 50 KV AC for distribution within the site. Next, several unit substations reduce voltage down to 400–500 V AC and introduce a type of *uninterruptible power supply (UPS)* to ensure that a fraction of the WSC can continue operating even when external power is lost. The UPS system typically includes a set of diesel-powered generators that can provide power within 10–15 seconds after external power loss. It also includes some form of energy storage (electrical, mechanical, or chemical) to cover the time it takes for generators to turn on. The UPS

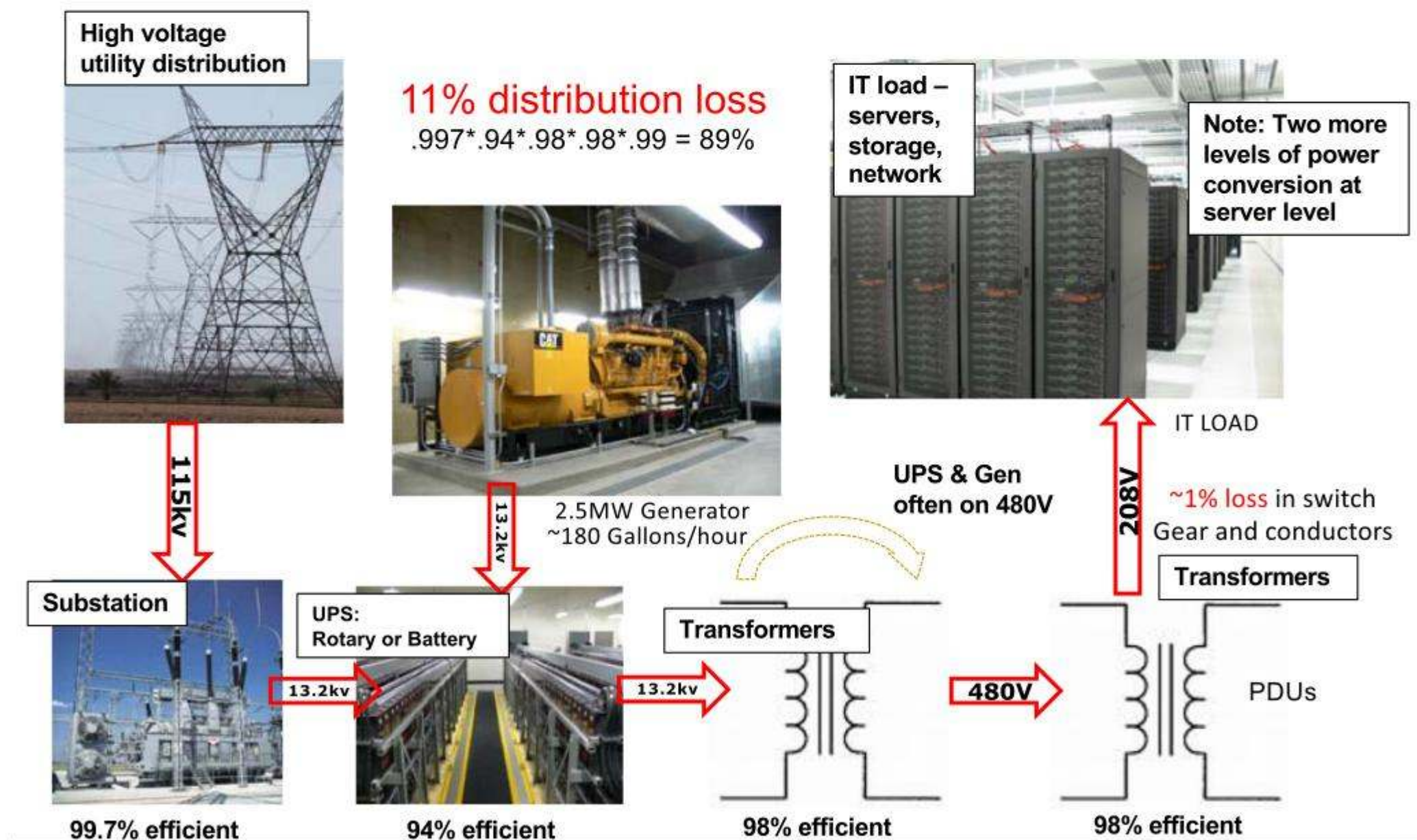


Figure 6.27 Power delivery in a WSC, including backup generators and UPS systems. From Hamilton, J., 2010. Cloud computing economies of scale. In: Paper Presented at the AWS Workshop on Genomics and Cloud Computing, June 8, 2010, Seattle, WA. http://mvdirona.com/jrh/TalksAndPapers/JamesHamilton_GenomicsCloud20100608.pdf.

will also smooth the power feed, removing spikes and sags. Each unit substation feeds a number of *power distribution units (PDUs)* that step down voltage to the 110 V AC (North America) or 220 V AC (Europe) that the IT hardware accepts. Each PDU can support 100–200 KW of power consumption and includes multiple circuit breakers, each supporting roughly 6 KW so that a short in one server will have a small blast radius. The power generators and UPS units are deployed with $N + 1$ or $N + 2$ redundancy with connectivity that allows all PDUs to function even if one generator or UPS unit is offline due to failure or maintenance.

Each voltage conversion just discussed introduces losses. The UPS system is often the biggest challenge, with 94% to 97% efficiency (3% to 6% losses), as it requires a double conversion, AC-DC and DC-AC, in order to connect power storage devices such as batteries. A conventional design may have total losses of up to 11% or efficiency of 89%. Unfortunately, there are further losses within each IT equipment that are not accounted for by the PUE metrics. A typical server includes a power supply that converts 110 V or 220 V AC to 12 V DC. Additional voltage regulators will step voltage down to what semiconductor chips need, which may be 1.1 V DC for CPUs. The overall losses within a server can be 10% as well. This loss has motivated some to suggest the use of server-level power usage effectiveness (sPUE) in addition to the PUE metric.

The key to limiting losses in power delivery is to reduce the number of voltage conversions. A key improvement has been the introduction of distributed UPS functionality to eliminate the double conversion for the centralized UPS connected at the substation level. A common design is to introduce an array of batteries within each rack as a local UPS for all the equipment in the rack. This UPS arrangement has 99% efficiency. It also introduces additional fault tolerance as an individual battery may fail while the array remains functional. Additional improvements can be achieved by switching to DC power distribution. For example, substations output 380 V DC and rack-level power supplies can convert voltage down to 12–48 V for distribution to the IT equipment. DC distribution within the rack also eliminates the need for a double conversion for the rack-level UPS (battery array).

WSC Cooling

WSC equipment generates considerable heat that is removed through hierarchies of closed and open cooling loops that eventually transfer heat outside the WSC facility. [Figure 6.28](#) shows the full cooling loop including the classical raised-floor design for cooling and the systems needed beyond the IT equipment room, such as the heat exchangers and cooling towers. A *computer room air-conditioning (CRAC)* unit cools the air in the server room. Cold air is distributed in the facility using the underfloor area between the slab floor and raised tiles supported by a steel structure. Cold air is released through perforated tiles in the *cold aisles* of the facility. The room is organized in alternating aisles of cold air and hot air by orienting servers in opposite directions in alternating rows of racks so that hot exhaust blows in alternating directions. The fans in the servers help cool air

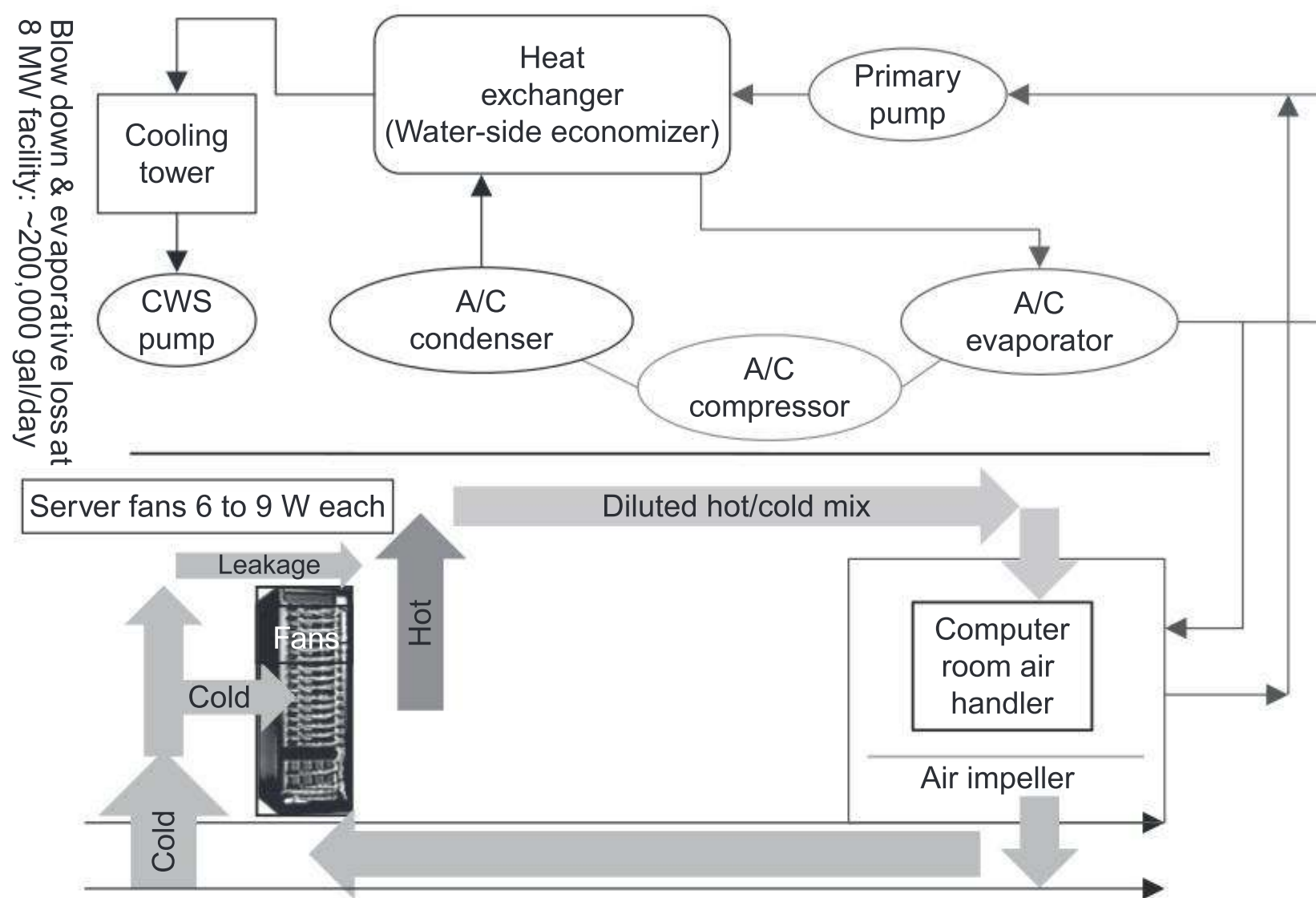


Figure 6.28 Mechanical design for WSC cooling systems. From Hamilton, J., 2010. Cloud computing economies of scale. In: Paper Presented at the AWS Workshop on Genomics and Cloud Computing, June 8, 2010, Seattle, WA. http://mvdirona.com/jrh/TalksAndPapers/JamesHamilton_GenomicsCloud20100608.pdf.

flow over chips and heatsinks, removing the heat towards the *hot aisles*. The hot air rises to the top and flows toward the intakes of the CRAC unit, where it is cooled again using chilled water, similar to how a refrigerator removes heat by releasing it outside the refrigerator. As a liquid absorbs heat, it evaporates. Conversely, when a liquid releases heat, it condenses. Air conditioners pump the liquid into coils under low pressure to evaporate and absorb heat, which is then sent to an external condenser, where it is released. Thus, in a CRAC unit, fans push warm air past a set of coils filled with cold water, and a pump moves the warmed water to the chillers to be cooled down.

This CRAC-based design is particularly inefficient and was the primary contributor to the high PUE in early data centers (>2.5). Over the past 15 years, a series of steps have drastically improved WSC cooling overheads down to 10% to 20%. One of the easiest but important ways to improve efficiency is to run the IT equipment at higher temperatures so that the air does not need to be cooled as much. WSCs raised the cold aisle temperature from less than 68°F (20°C) in traditional IT facilities to 80°F (27°C). While IT equipment can operate at even higher temperatures, it can affect the frequency of errors and overall equipment lifespan. Another simple optimization was to physically separate hot and cold aisles to avoid losses due to hot and cold air mixing toward the top of racks. [Figure 6.29](#) shows the isolated warm aisle in a Google WSC. Next, WSC



Figure 6.29 Hot aisle in a Google data center, which is clearly not designed to accommodate people.

architects holistically optimized airflow. In conventional data centers, each server relies on internal fans to ensure a sufficient flow of cool air over the hot chips to maintain their temperature. These mechanical fans are one of the weakest components in servers; for example, the MTBF of fans is 150,000 hours versus 1,200,000 hours for disks. They also consume significant power themselves, 6–9 W each. Google, for instance, designed the server fans to work synergistically with dozens of giant fans in the room ([Figure 6.30](#)) to ensure airflow for the whole room. This division of labor means the small server fans use as little power as possible while delivering maximum performance at the worst-case power and ambient conditions. The large fans are controlled using air pressure as the control variable. The fan speeds are adjusted to maintain a minimum pressure difference between the hot and cold aisles.

The next set of optimizations targeted the need for mechanical refrigeration in air-conditioning chillers. The basic idea is to get almost free cooling using low-temperature air or water available close to the WSC facility. Air-economization involves using large fans to push cold outside air into the facility. The air is first filtered to avoid contamination. Hot air is expelled outside the WSC building. Water-economization exploits cold water near the data center, such as a lake, river, or sea. The chiller is replaced with a heat exchange system between the water loop that cools air in the WSC and the external source of cold water ([Figure 6.31](#)). Air- and water-economization may completely replace mechanical refrigeration or reduce the need to use chillers in just a few warm days of the year. Hence the PUE may vary over time as shown in [Figure 6.26](#).



Figure 6.30 The cool air blows into the room containing the aisles of servers in a Google WSC. The hot air goes through large vents into the ceilings, where it is cooled before returning to these fans.



Figure 6.31 Steam rising from the cooling towers that transfer heat to the air from the water used to cool equipment.

Over time, the power consumption of processing chips has steadily increased, with the latest CPUs already at 400 W and GPUs at 800 W. Racks with such equipment are difficult to cool with air, leading to the wider use of liquid cooling. In [Chapter 7](#) we will see examples of ML accelerators that rely on closed-loop, liquid cooling straight to the processing chips. WSC operators are also investigating submerging IT equipment into special coolants in order to improve heat

removal and allow for faster operation, higher clock frequency, or more parallel processing units per chip (Jalili et al, 2021).

6.7

The Cost and Efficiency of Warehouse-scale Computing

Now that we have seen the architecture of WSCs, including their mechanical components, we can discuss their cost and why they are considered cost-effective despite being very expensive facilities.

Cost of a WSC

As mentioned in the introduction, WSC designers worry about the cost to operate as well as the cost to build the WSC. Accounting labels the former costs as *operational expenditures (OPEX)* and the latter costs as *capital expenditures (CAPEX)*.

To put the cost of energy into perspective, Hamilton (2010) did a case study to estimate the costs of a WSC. He determined that the CAPEX of an 8 MW facility was \$88 million and that the roughly 46,000 servers and corresponding networking equipment added another \$79 million to the CAPEX for the WSC. [Figure 6.32](#) shows the rest of the assumptions for the case study.

Hamilton's study works out to \$11/watt for the building, power, and cooling. Barroso et al. (2018) reported consistent results for several cases, with the cost at \$9–\$13/watt.

We can convert CAPEX into OPEX by a cost of capital conversion, assuming 5% borrowing cost, which is a standard convention in US accounting rules. That is, we can just amortize CAPEX as a fixed amount each month for the effective life of the equipment. [Figure 6.33](#) breaks down the monthly OPEX for Hamilton's case study. Note that the amortization rates differ significantly for his case study, from 10 years for the facility to 4 years for the networking equipment and 3 years for the servers. Thus the WSC facility lasts a decade, but the servers are replaced every 3 years and the networking equipment every 4 years. By amortizing the CAPEX, Hamilton came up with a monthly OPEX, including accounting for the cost of borrowing money (5% annually) to pay for the WSC. At \$3.8 million, the monthly OPEX is about 2% of the CAPEX (or 24% annually).

This figure allows us to calculate a handy guideline to keep in mind when making decisions about which components to use when being concerned about energy. The fully burdened cost of a watt per year in a WSC, including the cost of amortizing the power and cooling infrastructure, is

$$\begin{aligned} & \frac{\text{Monthly cost of infrastructure} + \text{monthly cost of power}}{\text{Facility size in watts}} \times 12 \\ &= \frac{\$765 \text{ K} + \$475 \text{ K}}{8 \text{ M}} \times 12 = \$1.86 \end{aligned}$$

Size of facility (critical load watts)	8,000,000
Average power usage (%)	80%
Power usage effectiveness	1.45
Cost of power (\$/kWh)	\$0.07
% Power and cooling infrastructure (% of total facility cost)	82%
CAPEX for facility (not including IT equipment)	\$88,000,000
Number of servers	45,978
Cost/server	\$1450
CAPEX for servers	\$66,700,000
Number of rack switches	1150
Cost/rack switch	\$4800
Number of array switches	22
Cost/array switch	\$300,000
Number of layer 3 switches	2
Cost/layer 3 switch	\$500,000
Number of border routers	2
Cost/border router	\$144,800
CAPEX for networking gear	\$12,810,000
Total CAPEX for WSC	\$167,510,000
Server amortization time	3 years
Networking amortization time	4 years
Facilities amortization time	10 years
Annual cost of money	5%

Figure 6.32 Case study for a WSC, rounded to nearest \$5000. Internet bandwidth costs vary by application, so they are not included here. The remaining 18% of the CAPEX for the facility includes buying the property and the cost of construction of the building. We added people costs for security and facilities management in Figure 6.14, which were not part of the case study. Note that Hamilton's estimates were done before he joined Amazon, and they are not based on the WSC of a particular company. Based on Hamilton, J., 2010. Cloud computing economies of scale. In: Paper Presented at the AWS Workshop on Genomics and Cloud Computing, June 8, 2010, Seattle, WA. http://mvdirona.com/jrh/TalksAndPapers/JamesHamilton_GenomicsCloud20100608.pdf.

The cost is roughly \$2 per watt-year. From a short-term economic point of view, reducing costs by saving energy should not result in spending more than \$2 per watt-year (see Section 6.9). From a long-term environmental point of view, we might be able to justify much higher costs to save energy (see Section 7.8).

Note that in Figure 6.33, more than a third of OPEX is related to power, with that category trending up while server costs are trending down over time. The networking equipment is significant at 8% of total OPEX and 19% of the server CAPEX, and networking equipment is not trending down as quickly as servers are, perhaps because of the continuing demand for higher network bandwidth. This difference is especially true for the switches in the networking hierarchy

Expense (% total)	Category	Monthly cost	Percent monthly cost
Amortized CAPEX (85%)	Servers	\$2,000,000	53%
	Networking equipment	\$290,000	8%
	Power and cooling infrastructure	\$765,000	20%
	Other infrastructure	\$170,000	4%
OPEX (15%)	Monthly power use	\$475,000	13%
	Monthly people salaries and benefits	\$85,000	2%
	Total OPEX	\$3,800,000	100%

Figure 6.33 Monthly OPEX for Figure 6.32, rounded to the nearest \$5000. Note that the 3-year amortization of servers means purchasing new servers every 3 years, whereas the facility is amortized for 10 years. Thus the amortized capital costs for servers are about three times more than for the facility. People costs include three security guard positions continuously for 24 h a day, 365 days a year, at \$20 per hour per person, and one facilities person for 24 h a day, 365 days a year, at \$30 per hour. Benefits are 30% of salaries. This calculation does not include the cost of network bandwidth to the Internet because it varies by application or vendor maintenance fees because they vary by equipment and by negotiations.

above the rack, which represent most of the networking costs. People costs for security and facilities management are just 2% of OPEX. Dividing the OPEX in [Figure 6.33](#) by the number of servers and hours per month, the cost is about \$0.11 per server per hour.

Barroso et al., (2018) evaluated CAPEX and OPEX in terms of cost per watt per month. Thus, if a 12-MW WSC is depreciated over 12 years, the depreciation cost is \$0.08 per watt per month. They assumed the company got the capital for the WSC by taking out a loan at 8% annually—corporate loans are typically between 7% and 12%—and the interest payments added another \$0.05, giving a total of \$0.13 per watt per month. They factored in the cost of servers similarly. A 500-watt server that cost \$4000 was \$8 per watt, and the 4-year depreciation was \$0.17 per watt per month. An 8% interest on a loan for the servers added \$0.02. They estimated networking at \$0.03 per watt per month. They reported that the typical OPEX cost for multiple MW WSCs varied from \$0.02–\$0.08 per watt per month. The grand total was \$0.37–\$0.43 per watt per month. For an 8-MW WSC, the monthly cost minus the cost of electricity is about \$3.0 million to \$3.5 million. If we subtract the monthly power use from Hamilton’s calculation, his estimate of the monthly rate will be \$3.3 million. Given the different approaches to predicting costs, these estimates are remarkably consistent.

Example The cost of electricity varies by region in the United States from \$0.03–\$0.15 per kilowatt-hour. What is the impact on hourly server costs of these two extreme rates?

Answer We multiply the critical load of 8 MW by the average PUE from [Figure 6.32](#) (second row) to calculate the average power usage:

$$8 \times 1.45 \times 80\% = 9.28 \text{ Megawatts}$$

The monthly cost for power then goes from \$475,000 in Figure 6.33 to \$205,000 at \$0.03 per kilowatt-hour and to \$1,015,000 at \$0.15 per kilowatt-hour. These changes in electricity cost alter the hourly server costs from \$0.11–\$0.10 and \$0.13, respectively.

Example What would happen to monthly costs if the amortization times were all made to be the same—say, 5 years? How would that change the hourly cost per server?

Answer The spreadsheet is available online at <http://mvdirona.com/jrh/TalksAndPapers/PerspectivesDataCenterCostAndPower.xls>. Changing the amortization time to 5 years changes the first four rows of Figure 6.33 to

Servers	\$1,260,000	37%
Networking equipment	\$242,000	7%
Power and cooling infrastructure	\$1,115,000	33%
Other infrastructure	\$245,000	7%

and the total monthly OPEX is \$3,422,000. If we replaced everything every 5 years, the cost would be \$0.103 per server hour, with more of the amortized costs now being for the facility rather than the servers, as in Figure 6.33.

The rate of about \$0.10 per server per hour can be much less than the cost for many companies that own and operate their own (smaller) conventional data centers. The cost advantage of WSCs led large Internet companies to offer computing as a utility where, like electricity, you pay only for what you use.

Over the years, WSCs have increased in size significantly, with the large facilities now consuming over 100 MW. The methods for TCO calculation and conclusions presented above still apply. Modern WSCs cost \$7–\$10 per Watt to build. As shown in an example study in Figure 6.34, the TCO of a fully populated WSC is dominated by capital expenses, in particular, the cost of compute/networking/storage equipment (60%–65%) and the facility cost (10%). Power and PUE overheads are significantly lower (5%–3%, respectively).

Efficiency Through Power Provisioning

Even at \$7 per W, a 100 MW WSC costs \$700 M to build. A first way to improve the WSC cost-effectiveness is to amortize this cost by hosting as many servers as possible. Surprisingly, it is not obvious how many servers a WSC can support after subtracting the overhead for power distribution and cooling. The *nameplate*

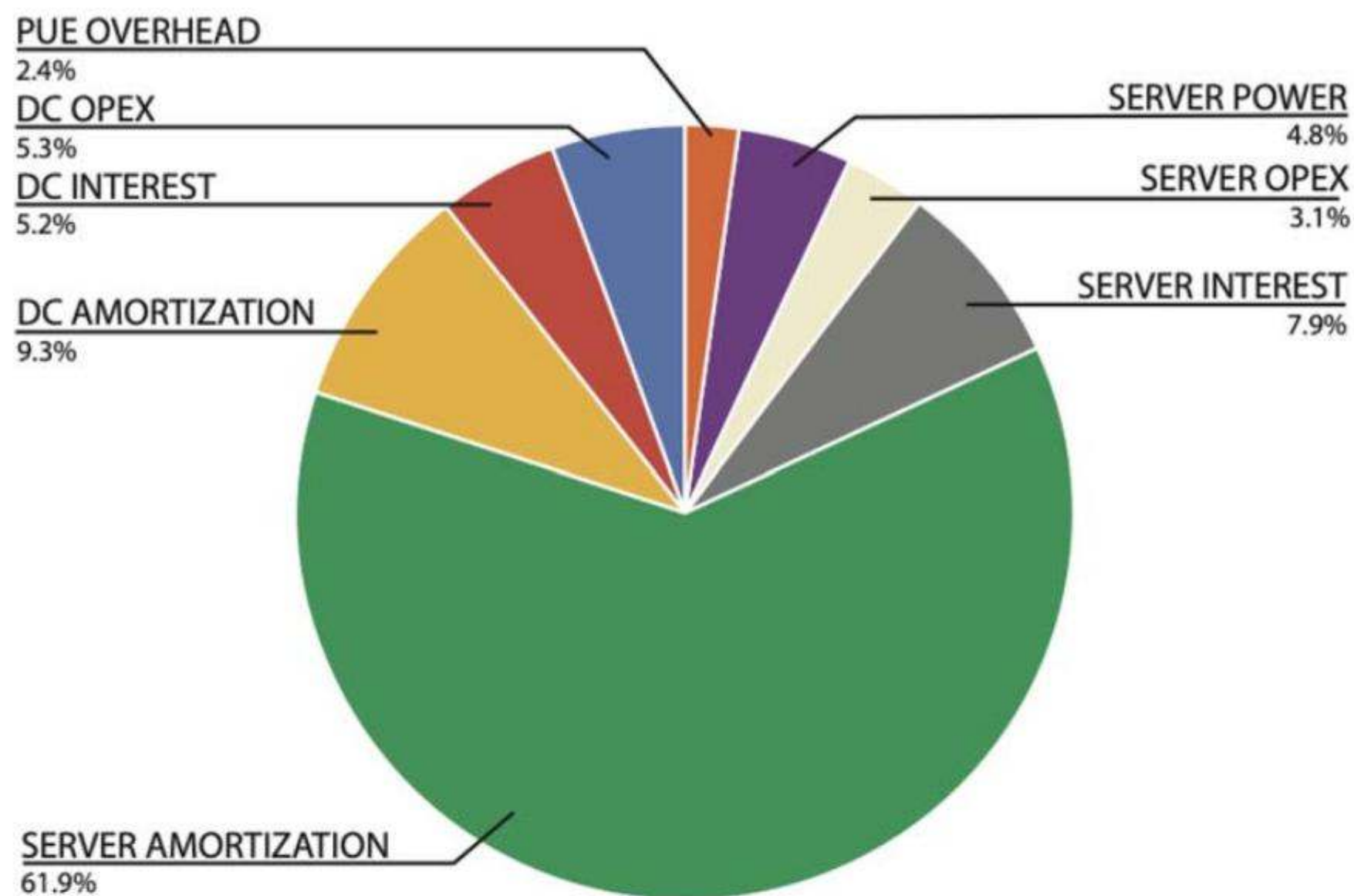


Figure 6.34 The breakdown of the yearly WSC TCO from Figure 6.1 in Barroso et al., (2018). The TCO refers to a WSC that hosts 2-socket servers that draw 340 W and cost \$5 K in 2018. The cost of electricity is assumed to be 6.7 cents/kWh. The cost of data center construction is \$10/W, amortized over 20 years. The servers are financed with an 8% interest-only loan. Operational costs are \$0.04/W per month and average PUE is 1.5.

power rating from the server manufacturer is always conservative: it is the maximum power a server can ever draw if all its components reach their maximum power consumption concurrently. The first step then is to measure a single server under a variety of workloads to be deployed in the WSC, ranging from power-virus benchmarks to more realistic workloads. Networking is typically about 5% of power consumption, so it can be ignored at first. These measurements lead to estimates for the *actual peak power* of servers, which is often 60% lower than the nameplate power, and allows for a significant increase in the number of servers deployed in the WSC.

Fan et al., (2007) showcased that provisioning based on the actual peak power of servers is again too conservative. They found that there is a significant gap between what thousands of servers could theoretically do, in the worst case, and what they will do in practice, since no real workloads will keep thousands of servers all simultaneously at their peaks. They found that they could safely oversubscribe the number of servers by as much as 40% based on the measured peak power of a single server. This kind of power oversubscription is now common in WSCs. However, it requires an extensive monitoring framework to detect correlated peaks across the servers within a rack, behind a PDU, or the WSC as a whole. The following are techniques that can be used to mitigate the problem

of temporarily exceeding the rack's, PDU's or WSC's power delivery capabilities: a) reduce the clock frequency and hence the power consumption of some servers; b) temporarily draw additional power from the batteries included in every rack; c) pause or deschedule lower priority tasks; and d) move tasks between racks, PDUs, or WSCs in order to balance power needs and avoid correlated spikes. If these techniques are used rarely, the impact on workload performance and batteries' lifetime is small.

Making the most of every Watt available in a WSC is also a key reason for developing DSAs for key WSC workloads such as ML. As we describe in [Chapter 7](#), DSAs' performance per watt is two orders of magnitude higher than that of general-purpose CPUs. DSAs significantly reduce the need to build new WSCs to meet the performance requirements of their workloads. The savings easily make up for the cost to develop the hardware and software for a large custom ASIC, which was \$50–\$100 M in 2022.

Efficiency Through Higher Utilization

Since servers' CAPEX dominates TCO, keeping these servers utilized most of the time is important for overall efficiency. WSCs provide utility computing either for the public (cloud computing) or for the workloads of a single large company (private cloud). Multiplexing users and workloads on a pool of servers results in higher utilization as various peaks and lows in workload demands average out. Nevertheless, it is difficult to keep a WSC at near full utilization for long periods of time. [Figure 6.35](#) shows the average utilization distribution for two Google clusters over a period of 3 months. Cluster (a) is used for mixed workloads including several user-facing services. These services exhibit diurnal patterns with high activity during business hours or important events and low activity during nights or holidays. Hence utilization varies between 10% and 50%. Cluster (b) is used for large, continuous batch workloads which exhibit low variation in their demands over long periods of time. Hence utilization varies between 50% and 85%.

Variability in user load is a major but not the only reason for medium to low utilization in many WSCs. It is also difficult for developers to size the resources needed for their applications, as well as the number and type of instances. Developers tend to oversize their workloads to sidestep issues such as performance variability due to the hardware heterogeneity in WSCs and the occasional interference from coscheduled workloads (Delimitrou & Kozyrakis 2014). For example, a workload that uses most cores in a CPU is less likely to suffer due to cache misses induced by a memory-intensive workload that is scheduled on the same machine. Most developers have limited visibility and understanding of performance issues; hence supersizing their resource reservations is a practical approach for them.

Over the years, several techniques have been developed to help increase utilization. These techniques include tools that automatically size and scale the

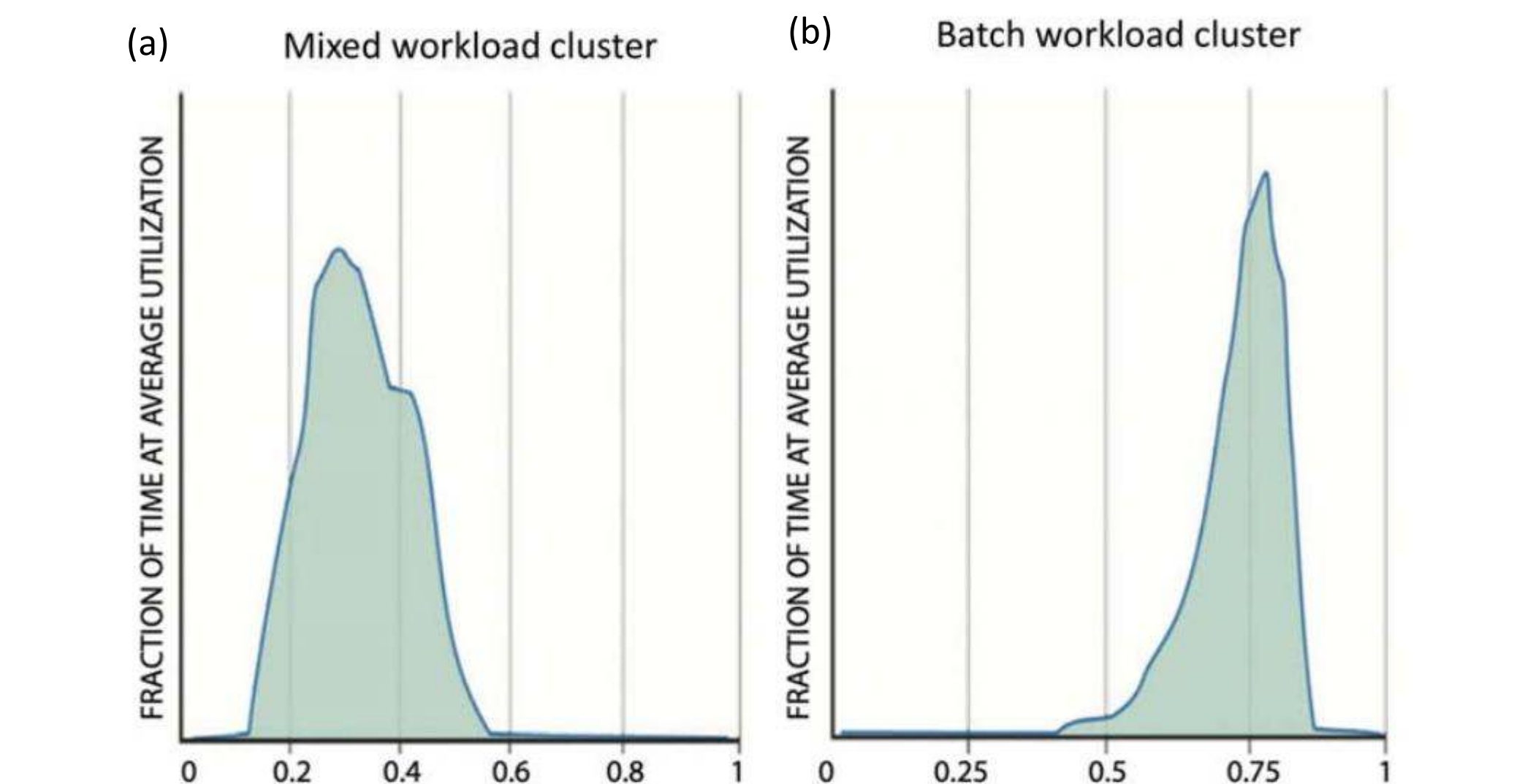


Figure 6.35 The average CPU utilization of two Google clusters with over 20 K servers each during the period January to March 2013 from Figure 5.5 in Barroso et al (2018). Cluster (a) runs mixed workloads, including both online services and batch applications. Cluster (b) runs continuous batch workloads.

resources given to cloud workloads based on actual needs (right sizing and autoscaling); schedulers that predict or track interference between workloads and adjust assignments to reduce it (interference-aware scheduling); and schedulers that allow low-priority tasks, such as preemptable spot instances running batch analytics, to safely run on reserved but currently idling resources (resource harvesting). ML and online learning techniques are frequently used to deal with the large design space that is difficult to explore safely and efficiently using heuristics. At the hardware level, there is increased focus on performance isolation for coscheduled workloads, using techniques such as cache and memory bandwidth partitioning (Lo et al., 2016) and quality-of-service (QoS) techniques for networking and storage systems.

Efficiency Through Higher Performance

Since performance per dollar is the ultimate metric, we still need to measure performance. The performance of distributed workloads can be impacted by several aspects including bandwidth drops and latency increases depending on the distance to the data. In a WSC, the DRAM bandwidth within a server is 40 times greater than within a rack, which in turn is 4–10 times greater than within the WSC; see Figure 6.21. Thus there is another kind of locality to consider in the placement of data and programs within a WSC.

Developers of Internet services are also concerned with latency because latency is visible to users. Users' satisfaction and productivity are tied to response time of a service. Several studies from the timesharing days report that user productivity is inversely proportional to time for an interaction, which was typically broken down into human entry time, system response time, and time for the person to think about the response before hitting the next entry (Doherty and Thadhani, 1982). The results of experiments showed that cutting system response time by 30% shaved the time of an interaction by 70% (Brady, 1986). This implausible result is explained by human nature: people need less time to think when given a faster response, as they are less likely to get distracted and remain "on a roll."

Figure 6.36 shows the results of a more recent experiment for the Bing search engine, where delays of 50–2000 ms were inserted at the search server (Schurman and Brutlag, 2009). As expected from previous studies, time to next click roughly doubled the delay; that is, a 200 ms delay at the server led to a 500 ms increase in time to next click. Revenue dropped linearly with increasing delay, as did user satisfaction. A separate study on the Google search engine found that these effects lingered long after the 4-week experiment ended. Five weeks later, there were 0.1% fewer searchers per day for users who experienced 200 ms delays, and there were 0.2% fewer searches by users who experienced 400 ms delays. Given the amount of money made in search, even such small changes are disconcerting. In fact, the results were so negative that they ended the experiment prematurely.

Because of this extreme concern with satisfaction of all users of an Internet service, performance goals are typically specified so that a high percentage of requests are below a latency threshold, rather than just offering a target for the average latency. Such threshold goals are called *service-level objectives (SLOs)*. An SLO might be that 99% of requests must be below 100 ms. Thus, the designers of Amazon's Dynamo key-value storage system decided that for services to offer good latency on top of Dynamo, their storage system had to deliver on its latency goal 99.9% of the time (DeCandia et al., 2007). For example, one improvement of Dynamo helped the 99.9th percentile much more than the average case, which reflects their priorities.

Server delay (ms)	Increased time to next click (ms)	Queries/user	Any clicks/user	User satisfaction	Revenue/user
50	—	—	—	—	—
200	500	—	−0.3%	−0.4%	—
500	1200	—	−1.0%	−0.9%	−1.2%
1000	1900	−0.7%	−1.9%	−1.6%	−2.8%
2000	3100	−1.8%	−4.4%	−3.8%	−4.3%

Figure 6.36 Negative impact of delays at the Bing search server on user behavior (Schurman and Brutlag, 2009).

Dean and Barroso (2013) proposed the term *tail tolerant* to describe systems designed to meet such goals:

Just as fault-tolerant computing aims to create a reliable whole out of less-reliable parts, large online services need to create a predictably responsive whole out of less-predictable parts.

The causes of unpredictability include contention for shared resources (caches, networks, etc.), queuing, variable microprocessor performance because of optimizations like Turbo mode and energy-saving techniques like DVFS, software garbage collection, and many more. Google concluded that instead of trying to prevent such variability in a WSC, it made more sense to develop tail-tolerant techniques to mask or work around temporary latency spikes. For example, fine-grained load balancing can quickly move small amounts of work between servers to reduce queuing delays.

Efficiency Through Energy Proportionality

As mentioned in [Section 6.6](#), although PUE measures the energy efficiency of a WSC, it has nothing to say about what goes on inside the IT equipment. Another source of electrical inefficiency is the power supply *inside* the server, which converts input of high AC or DC voltage to the lower voltages that chips and disks use. In 2007, many power supplies were 60%–80% efficient, hence there were greater losses inside the server than there were going through the many steps and voltage changes from the high-voltage lines at the utility tower to supply the low-voltage lines at the server. One reason was that the power supply was often oversized in watts for what was on the motherboard. Moreover, such power supplies were typically at their worst efficiency at 25% load or less, even though, as [Figure 6.35](#) shows, many WSC servers operate in that range. Computer motherboards also have voltage regulator modules (VRMs), and they can have relatively low efficiency as well.

Barroso and Hölzle (2007) argued that the goal for the whole server should be *energy proportionality*; that is, servers should consume energy in proportion to the amount of work performed. Fifteen years later, we’ve gotten close but have not hit that ideal goal. For example, the best-rated SPECpower servers in [Chapter 1](#) still use about 20% of the full power when idle and almost 50% of full power at just 20% load. That represents huge progress since 2007, when an idle computer used 60% of full power and 70% at a 20% load, but there is still room to improve.

Systems software has been traditionally designed to use all an available resource if it potentially improves performance, without concern for the energy implications. For example, operating systems use all of the memory for program data or for file caches, although much of the data will likely never be used. Recently, WSC software architects started optimizing for energy usage and proportionality. For example, Lo et al., (2014) designed a dynamic controller that performs fine-grain power management so that a collection of servers barely

meets the SLO requirements of an online service. This leads to improved energy proportionality, with savings of 20%–30% when servers run at 10%–60% utilization.

6.8

Putting It All Together: Custom Silicon in the AWS Cloud

Launched in 2006, *Amazon Web Services (AWS)* was the first provider of IaaS cloud instances. As of the beginning of 2023, AWS was the largest public cloud provider. It offers more than 200 cloud services and operates with a run rate revenue of \$80 billion. AWS owns cloud infrastructure across the world, with 30 launched geographic regions and 410 points of presence (edge locations and caches). In these 30 regions AWS operates 96 availability zones, where each availability zone consists of one or more discrete WSCs, each with redundant power, networking, and connectivity, housed in separate facilities.

AWS has been developing hardware and software technology for its cloud infrastructure with the high-level goal to reduce its TCO and improve the performance and cost for customer workloads. In addition to the efficiency optimizations discussed in [Section 6.7](#), recent developments in AWS cloud systems have been aiming at the following objectives:

- *Eliminate virtualization overheads.* This optimization reduces infrastructure costs and allows customers to use all available compute cycles for their applications. It also leads to a significant reduction in energy usage and the emitted and embodied carbon emissions.
- *Reduce the time and cost to deploy and maintain new technology.* New technologies, such as new CPUs, accelerators, or new DRAM or Flash chips, provide major improvements in cost and performance for both the cloud operator and its customers. To derive the biggest possible benefit, it is important to reduce deployment time from months to weeks and reduce maintenance downtime that is visible to customer workloads to practically zero.
- *Increase availability and reliability.* As cloud facilities are becoming critical utilities, similar to electricity and telecommunications systems, they cannot just go down.
- *Expand use cases and target markets.* For instance, AWS designed the Outpost racks that enable a pool of AWS compute and storage units to be safely placed in customer or third-party sites, supporting applications that require on-premise or edge deployment for latency, security, or regulatory reasons.

Custom silicon has been playing an increasing role in the AWS infrastructure over the past decade. In 2015 AWS acquired Annapurna Labs, a fabless semiconductor company that has since developed a series of chips and systems optimized for cloud computing infrastructure. AWS has sufficient volume to develop its

own chips and achieve the following benefits over using commodity chips in its WSC systems:

- *Vertical innovation:* In a horizontal IT ecosystem chip vendors, hardware vendors, and software vendors optimize at the boundaries of what they deliver. By designing its own chips, AWS can innovate vertically, for instance, by adding chip features that provide savings at the system or WSC level.
- *Specialization:* AWS chip designs prioritize the features that make a big end-to-end difference for its use cases. In contrast, a commodity chip must be optimized for the use cases of multiple potential customers. The long list of features leads to design compromises, as well as increased design time, silicon area, and power consumption.
- *Development speed:* Software teams typically start developing for a new commodity chip once it is deployed. By controlling both silicon and software design cycles, AWS can develop the two in parallel and ensure that the software is ready as soon as the hardware is deployed. This coordination allows AWS to get the most value from each new chip.
- *Security:* Custom chips allow AWS to design the entire system holistically to reduce the attack surface. AWS uses its own chips to provide a hardware root of trust, formally verifies the firmware that runs on them, and uses them to limit the interactions within a server to a narrow set of secure APIs.
- *Lower cost:* Commodity chips are priced at 20%–100% above their cost. At AWS volumes, not paying for the chip vendor’s profit margin leads to major savings and justifies the cost of in-house chips development.

By early 2023, AWS had deployed the following custom chips in its WSCs:

- *Nitro:* a family of chips designed to offload and accelerate hypervisor and I/O tasks such as VM networking, VM storage, and RDMA.
- *Nitro security:* a small chip that is deployed on AWS motherboards to support active security monitoring of hardware interfaces and secure firmware upgrades.
- *Graviton:* a line of performance, power, and cost-efficient CPUs for cloud workloads.
- *Trainium* and *Inferentia:* a family of DSAs for ML training and inference. We discuss architecture of DSA chips for ML in [Chapter 7](#).

The Nitro System

The hypervisor is a significant source of performance overheads for cloud workloads as it mediates every I/O operation. For networking, for example, the

hypervisor must implement the encapsulation/decapsulation necessary for virtual private networking (VPC), apply security rules and rate limiters, and implement routing. Given the speed of modern I/O devices, such as 200 Gbps Ethernet NICs and 5 Gbps NVMe drives, and the high number of guest VMs a modern server can host, the hypervisor tasks can occupy several cores in the host CPU. If these cores are reserved for the hypervisor, the effective capacity of hosts drops significantly. If the cores are not reserved, guest workloads may experience significant performance unpredictability (jitter) as the collective I/O traffic varies over time. Finally, a complex hypervisor running on the same CPU as the guest VMs creates security risks.

The goal of the Nitro project has been to offload nearly all hypervisor tasks to custom AWS silicon, leaving all CPU cycles to VM workloads. AWS gradually executed toward this goal since 2013. By 2017, it replaced its original Xen-based hypervisor with the minimal Nitro hypervisor. This KVM-based hypervisor allocates cores, memory, and interrupts by configuring CPU features like VT-x and VT-d at VM launch or VM resizing events. With CPUs that support posted interrupts, the hypervisor is out of the way for all other events, including the frequent I/O traffic. The limited scope of the Nitro supervisor makes its CPU footprint negligible and reduces its interface with the VM to improve security.

The remaining hypervisor tasks are implemented in a family of Nitro cards based on custom AWS silicon:

- *The Elastic Network Adapter (ENA) Controller*: This card is a PCIe Ethernet NIC that utilizes SR-IOV to give VM guests direct access to ENA virtual functions. The ENA interface uses submission and completion queues as we discussed in [Section 6.5](#). The NIC implements the VPC data-plane tasks, such as encapsulation/decapsulation, security rules, rate limiters, routing, time synchronization, and traffic encryption.
- *The Elastic Fabric Adapter (EFA) Controller*: This card is a PCIe NIC optimized for data-intensive ML and HPC applications. Similar to RoCE and IB NICs, EFA supports OS bypass by implementing a full network transport protocol in the NIC. The AWS SRD protocol provides a reliable datagram service that exploits the multipath nature of the WSC network fabric to achieve high throughput and low latency (Shalev et al., 2020). Applications use the EFA through popular communication frameworks such as the OFI Libfabric or the NVIDIA Collective Communications Library (NCCL). The NIC supports zero-copy networking for GPUs and other DSAs, as well as all baseline VPC data-plane tasks.
- *The Elastic Block Storage (EBS) Controller*: The PCIe EBS controller exposes to VM guests a collection of NVMe virtual functions that allow them to access block storage volumes over the network. Hence VMs can dynamically add or remove volumes and resize volumes. VMs can also turn off for a while or migrate to a new machine and reconnect to their persistent EBS volume. The controller implements all system software necessary for remote storage

access over the network, including zero-copy, data integrity checks, barriers and snapshots, and the cyphers for encrypted storage volumes.

- *The Instance Storage Controller:* This PCIe card exposes to VM guests a collection of NVMe virtual functions that allow them to access local Flash and hard disks. The controller implements transparent encryption. When a VM terminates the encryption key is destroyed, making the ephemeral data on the disks inaccessible. The controller provides limiters for shared disks and implements integrity checks, snapshots, and drive monitoring. For Flash storage, the controller also implements the FTL logic (Nitro-SSD).
- *The Nitro Controller Card:* This card interfaces with the EC2 control plane through passive API endpoints. It also coordinates all other Nitro cards and the Nitro hypervisor to implement resource assignment and instance life cycle management.

AWS has implemented five generations of chips for Nitro cards thus far. All chips contain Arm cores so that their functionality can evolve. Within each generation, there are multiple boards and sometimes multiple chip versions to support specific uses. Each typically has a dedicated *stock keeping unit (SKU)*, for instance, a dedicated SKU for EFA networking, a dedicated SKU for EBS, a mainstream “all-in-one” SKU that brings all functionality in one chip for high-volume server configurations, and a simple NIC SKU that has no Nitro functionality. Nitro cards are designed to allow for *hitless upgrades*, where the card firmware can be updated without stopping the VMs running on the CPU host.

Figure 6.37 shows how the Nitro cards and hypervisor work together to launch a VM instance. Upon receiving a customer request, the EC2 control plane will identify a server that can host the request. First, the EC2 control plane sends a command to the Nitro Controller on the server to launch the VM (step 1). Next, the Nitro Controller allocates specific cores and memory regions for this VM (step 2) and asks the hypervisor to configure the CPU for this allocation (step 3). The Nitro controller maps an EBS volume that contains the boot image for the VM (step 4) and makes it available to host through an NVMe interface (step 5). It also sets up all other EFA, ENA, EBS, and local instance storage devices associated with this VM (step 6). Finally, it makes the instance runnable so that it can now boot (step 7). Once the VM is running, the hypervisor executes only if the core, memory, or device allocation for this VM changes or if the VM makes a hypervisor call itself. With the hypervisor basically out of the way, the performance between a VM-based and a bare-metal instance is virtually identical as shown in Figure 6.38.

The Nitro system accelerates the deployment of new technology. Nearly all of the virtualization tasks are implemented in Nitro cards; hence new CPUs, DSAs, and storage systems can be deployed, and new operating systems can be supported without significant efforts in porting or extending hypervisors. Running a server in bare-metal configuration is identical to starting a VM without a hypervisor running on the host CPU. Instance and server management is uniform

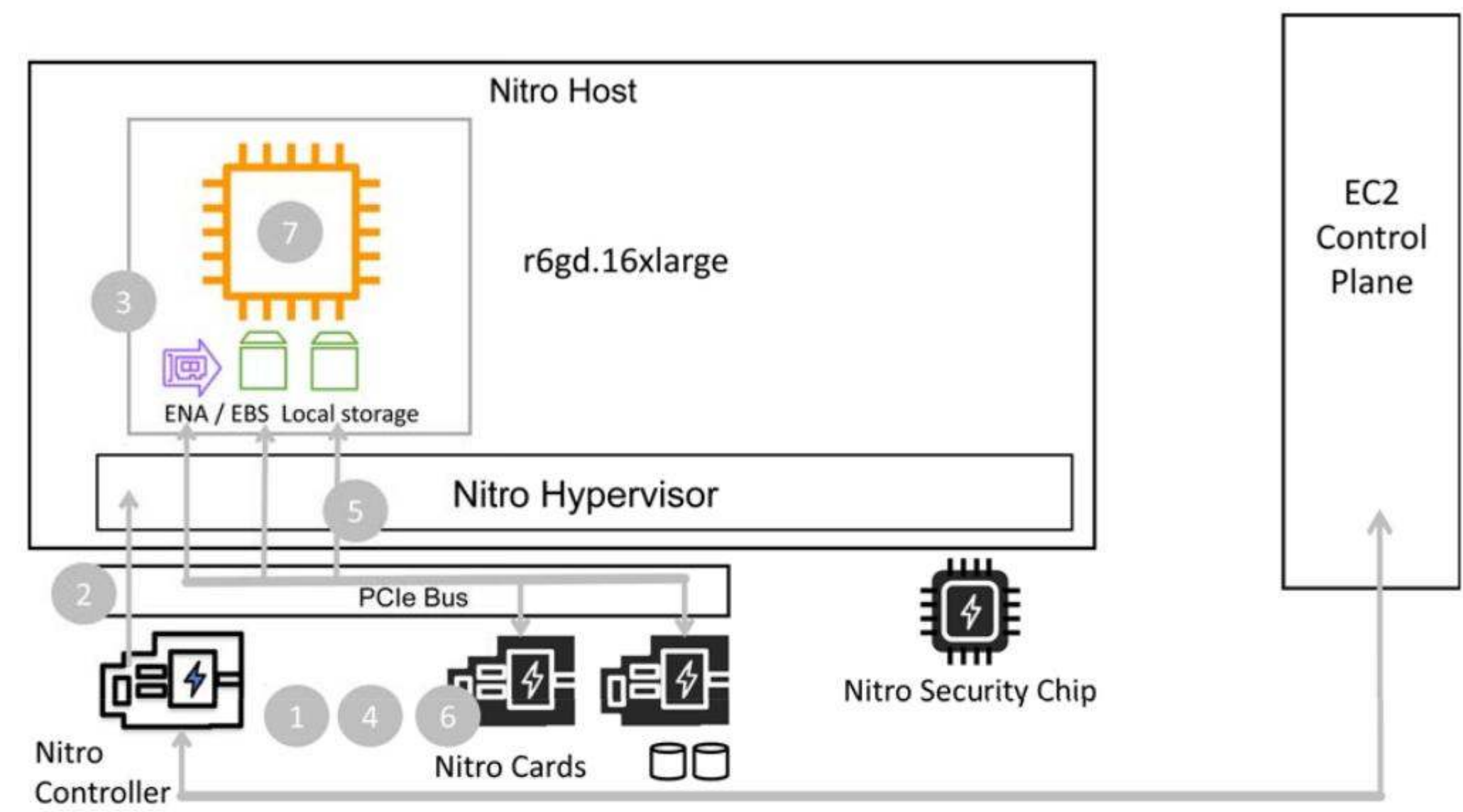


Figure 6.37 The launch process for VM instances on an AWS Nitro system. The Nitro host includes one or more CPUs with directly attached memory. All local storage devices, disk or Flash, are accessed through a Nitro Instance Controller card. Remote storage is accessed through a Nitro EBS card. Networking is provided by Nitro ENA and EFA cards. The OS running on the VM instance includes device drivers that access these devices over conventional PCIe and NVMe interfaces.

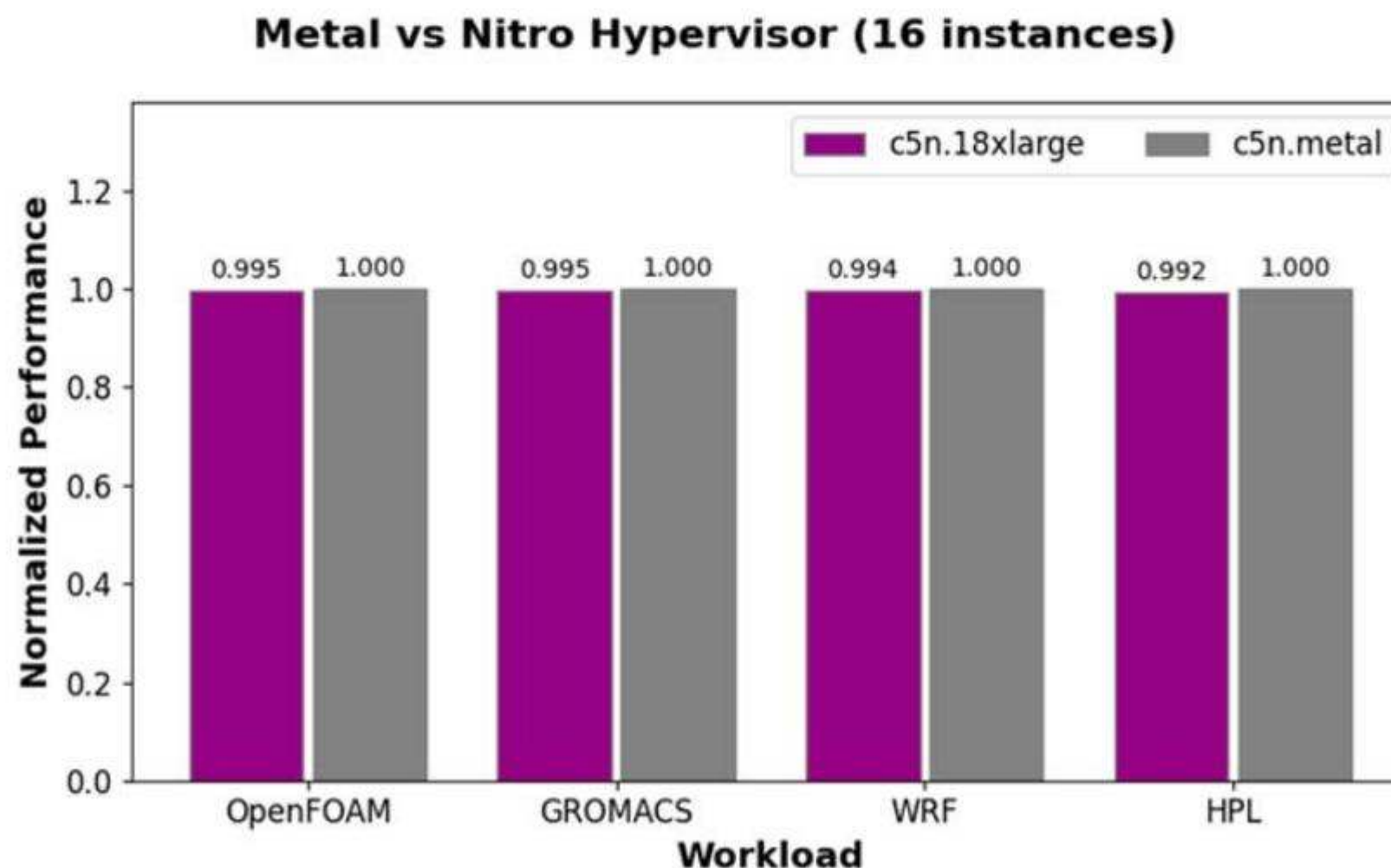


Figure 6.38 Normalized performance of a VM and a bare-metal instance on compute-intensive HPC benchmarks. The `c5n.18xlarge` and `c5n.metal` instances use the same hardware: a 2-socket Intel Xeon Platinum 8000 system with 18 cores per socket with a 100 Gbps EFA adapter. The only difference is that the `c5n.18xlarge` uses the Nitro hypervisor. For the HPC workloads used, the two instances perform within 1% of each other. The workload details are available at <https://aws.amazon.com/blogs/hpc/bare-metal-performance-with-the-aws-nitro-system/>.

through the Nitro cards. Prior to the Nitro system, AWS offered roughly 70 EC2 instances. In early 2023, AWS offers more than 600 instances with a variety of CPUs (Intel and AMD x86, AWS and Apple Arm), accelerators (GPUs, ML DSAs, Video DSAs, FPGAs), storage and networking options, and operating systems (Linux, Windows, Unix, macOS).

Nitro Security

The Nitro Security chip is a small chip placed in the server's main board so that it can monitor all operations to local nonvolatile storage devices through interfaces such as SPI or I2C buses to detect anomalies. This device delivers and attests to the board's firmware. The Security chip is also placed between the BMC and CPU for x86-based hosts. The Security chip plays two important roles. First, it prohibits any software running on the CPU from updating any of the firmware, such as the BMC firmware. Firmware updates are delivered through Nitro Controller and no firmware is stored on the board. This eliminates zero-day and supply chain attacks on storage devices such as SPI Flash. Second, it assists the Nitro Controller with the secure boot and attestation process.

When the boot process starts, the Security chip holds the server's CPUs and BMC in reset until the Nitro Controller has completed its own secure boot process and server authentication. The Nitro Controller uses a temper-resistant *trusted platform module (TPM)* to cryptographically measure and verify the integrity and validity of its own boot ROM. The TPM stores keys that have been used to digitally sign a set of known good measurements. If its boot ROM is valid, the Nitro Controller can extend a chain of trust by measuring and verifying the firmware used in subsequent steps of the boot process, as well as the software images used during the server's operation. If invalid firmware is detected, the server is immediately removed from service and will not host customer workloads until the problem is addressed. AWS has formally verified the memory safety properties of the early-stage boot code in the Nitro system (Cook et al., 2021). When the Nitro system completes its secure boot, it then downloads and attests the firmware in the server's main board. At this point, the Security chip allows the CPU and BMC to boot.

The Nitro secure boot process secures bare-metal servers as the OS running on the CPU cannot modify any of the BIOS, BMC, or any other firmware in the server. It also blocks the deployment, by mistake or by malicious intent, of any firmware that has not been properly signed and released through a production deployment process. Finally, it enables Nitro-based systems to be deployed outside of AWS facilities, as is the case with Outpost racks and edge deployments. AWS can use a remote attestation process to ensure that the Nitro servers are running an approved set of firmware and software images before they are allowed to run customer workloads and connect to services running in AWS data centers.

Beyond secure boot, another important security feature of the Nitro system is that it enables cloud management without direct access to user data by AWS

employees. Local storage and main memory are encrypted using VM-specific keys that are destroyed when the VM terminates. Persistent storage volumes can be encrypted using customer-provided keys. All network traffic between AWS data centers is transparently encrypted at the physical layer. All traffic within a VPC is transparently encrypted in a Nitro card for most EC2 instance types. All AWS management APIs are encrypted, authenticated, authorized, and logged.

The Graviton CPU Chips

AWS developed the Graviton Arm-based CPUs with the goal to provide the best possible price performance for EC2 instances and drive more energy-efficient servers. Graviton1 was the first Arm CPU to power IaaS instances in a large cloud provider (2018), establishing that the Arm hardware and software ecosystem can support cloud applications at scale.

Figure 6.39 presents the key features of three Graviton generations. The per-core performance has been steadily increasing using the OOO and SIMD

	Graviton 1	Graviton 2	Graviton 3(E)
Year announced	2018	2019	2021
Process technology	16nm	7nm	7nm
Transistor Count (B)	5	30	55
Instruction Set	ARMv8.0a	ARMv8.2a + fp16, rcpc, dotprod, crypto	ARMv8.4 + sve, rng, bf16, int8, crypto
Clock Frequency	2.3GHz	2.5GHz	2.6GHz
Number of cores	16	64	64
Core Type	Cortex-A72	Neoverse-N1	Neoverse-V1
Fetch/Decode/Issue Width	3 / 3 / 5	4-8 / 4 / 8	8 / 5-8 / 15
Functional Units (Int, FP/SIMD, LD/ST, BR)	3/2/2/1	3/2/2/1	4/4/5/2
SIMD FUs	2x 128b (Neon)	2x 128b (Neon)	4x 128b (Neon) 2x 256 (SVE)
ROB Size	128	128	256
Per Core L1I/L1D/L2	48KB/32KB/--	64KB/64KB/1MB	64KB/64KB/1MB
LLC Size	8MB	32MB	32MB
DRAM Type	DDR4-2633	DDR4-3200	DDR5-4800
DRAM Controllers	2	8	8
DRAM Bandwidth	40GB/s	200GB/s	300GB/s
DRAM Encryption	No	Yes	Yes
PCIe Type	Gen4	Gen4	Gen5
PCIe Lanes	32	48	64

Figure 6.39 The main characteristics of three generations of AWS Graviton CPUs. For Graviton3, the migration to the 5 nm process to save power was balanced against the maturity of the 7 nm process. Since the Graviton3 design is inherently low power, the 7 nm process was more cost-effective for the 2 years that the chip was in mass production. The Graviton3 last-level cache (LLC) is relatively small because the chip has plenty of DRAM bandwidth and latency at load is well capped. Moreover, the high DRAM bandwidth allows sophisticated prefetch engines to be particularly effective.

techniques we discussed in [Chapters 3 and 4](#). The Graviton2 core is roughly 2x faster than the Graviton1 core, while the Graviton 3 core improves integer and FP performance by another 30% and 60%, respectively. Memory bandwidth has increased from 50–300 GB/s per chip to support the higher per-core performance and the 4x increase in the core count over the years.

[Figure 6.40](#) presents the architecture of the Graviton3 chip. The package consists of 7 chiplets. The main chiplet contains the 64 cores and their caches connected with a mesh interconnect with 2 TB/s bisection bandwidth. The 2 chiplets on the left and 2 chiplets on the right contain memory controllers and off-load functions, while the bottom 2 chiplets contain the PCIe interfaces and other I/O functions. Several aspects of Graviton3 were optimized for cloud deployments. First, it uses a non-SMT core to avoid the performance interference between two threads sharing an SMT core and increase the efficiency of the L1 caches, branch predictors, and TLBs. Second, the CPU does not support 2-socket server configurations. This choice limits the number of cores a single VM can use but eliminates the need for NUMA optimizations in the software stack and reduces cost and power dissipation. Similar to the Meta Yosemite server discussed in [Section 6.3](#), the Graviton3 chassis amortizes system-level costs by including three 1-socket servers that share Nitro cards, BMC, and the power supply. Third, Graviton3 uses DDR5-4800 memory. The 50% memory bandwidth increase allows for the relatively small L3 cache and is critical for performance when cloud instances utilize a large number of cores. Finally, Graviton3 implements several security features including always-on main memory encryption. The CPU also implements the Arm pointer authentication extensions and a true random number generator for the Arm TRNG instruction. Graviton3 supports posted interrupts so that I/O interrupts can be delivered directly to VM guests without hypervisor intervention.

Even though AWS produces its own CPUs, it still deploys x86 CPUs and Apple’s CPUs. AWS serves millions of customers with diverse software compatibility constraints and performance requirements. One CPU may work best for workload with large code footprints and random memory accesses, while another

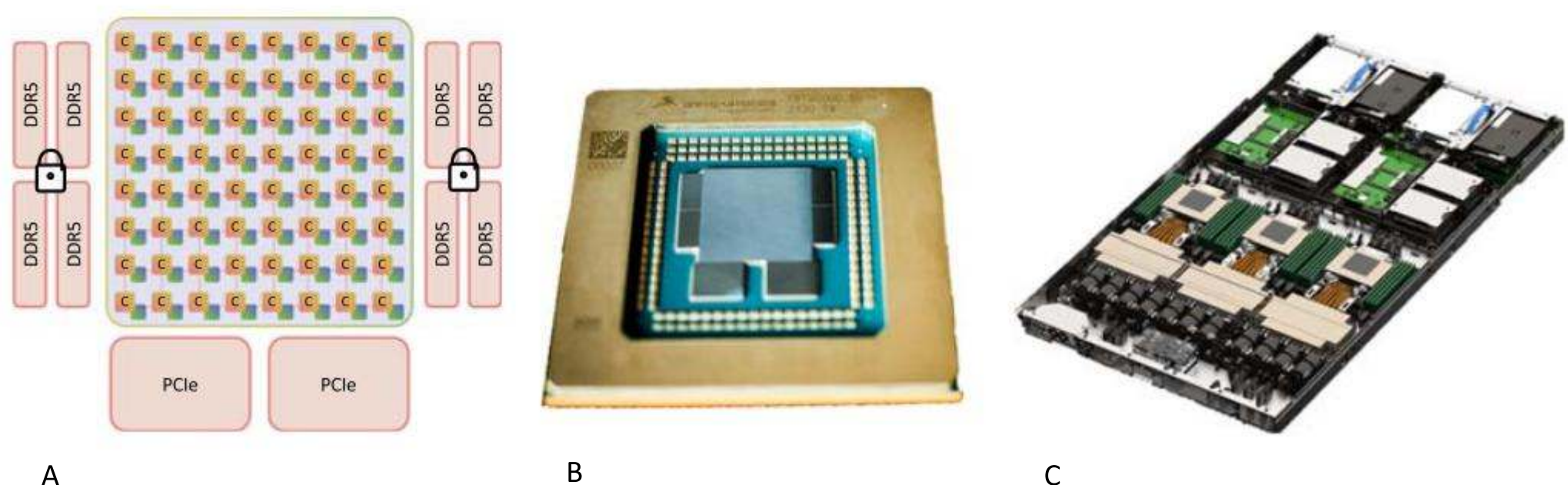


Figure 6.40 The block diagram (a), die photo (b), and server chassis (c) for the Graviton3 CPU. The server chassis packs three one-socket Graviton3 servers that share the Nitro card and power infrastructure.

CPU may work best for I/O-intensive workloads. AWS provides a wide range of instances with different CPU architectures, clock frequencies, network, and storage configurations, allowing customers to pick the optimal one for their workload. In terms of sizing its own CPU, AWS aims to optimize cost performance. Some CPUs reach their *thermal design power (TDP)* limit with as little as half their cores fully loaded. Hence the cloud vendor must either leave the rest of the cores idle (higher cost) or operate all cores at lower clock frequency or with throttling mechanisms enabled (lower performance). AWS looks at CPU power and performance/power at the 50th, 90th, and 99th percentiles of load distribution in its fleet and favors designs that have lower power consumption (normalized to performance) at the 50th and 90th percentiles. The Graviton CPUs are designed to be energy proportional, consuming about 50% of their TDP at 50% of load. AWS also aims at end-to-end efficiency by cooptimizing the CPUs, networks, and storage systems. When applications achieve the best possible performance on each instance, they require fewer instances for a given load, which translates to cost savings for the customer, TCO savings for the cloud provider, and a reduction in emitted and embodied carbon emissions.

CPU Efficiency for Cloud Computing

To understand the tradeoffs in CPU choice for cloud computing, we will look at the estimated performance for the SPECCPU2017 benchmark suite for six AWS compute instances available in Q4 of 2022. [Figure 6.41](#) presents the configuration of these instances and the CPU chip they use. All instances occupy a full physical server, either a single socket or a dual socket machine. The six CPUs in these instances represent two CPU generations from Intel, AMD, and AWS, with significant differences in the number of OOO cores, memory hierarchy, and power consumption.

[Figure 6.42](#) shows the SPECCPU2017 performance estimates when using a single virtual core (vCPU) in each instance. For CPU microarchitectures that support SMT execution (AMD Milan and both Intel CPUs), this is equivalent to measuring a core's performance in single-threaded mode. As expected, the instances based on 2021 chips from Intel, AMD, and AWS (m6i, m6a, and c7g) outperform those with 2019 chips from the same vendor (c5, c5a, and c6g, respectively). All CPU vendors are gradually improving the ILP capabilities of their processors, use longer vector instructions, and in some cases increase the clock frequency. The m6i instance with the Intel Ice Lake CPU performs best for integer workloads, while the m6a instance with the AMD Milan CPU performs best for floating-point workloads. Nevertheless, the best-performing core for an individual benchmark can vary significantly. Each of the three 2021 cores outperforms the other two by more than 25% for benchmarks `520.omnetpp` (Intel Ice Lake), `519.lbr` (AMD Milan), and `549.fotonik3d` (AWS Graviton3). This variety illustrates the advantage of offering CPU choice in cloud platforms. Customers can

Instance Information	Instance	c5.metal	c5a.24xlarge	c6g.metal	m6i.32xlarge	m6a.48xlarge	c7g.16xlarge
	CPU chip sockets	2	1	1	2	2	1
	Total cores per instance	48	48	64	64	96	64
	Total vCPUs per instance	96	96	64	128	192	64
CPU Chip Information	CPU chip	Intel Xeon 8275L	AMD EPYC 7R32	AWS Graviton2	Intel Xeon 8375C	AMD EPYC 7R13	AWS Graviton3
	Architecture	Cascade Lake	Rome	Graviton2	Ice Lake	Milan	Graviton3
	Year introduced	2019	2019	2019	2021	2021	2021
	Technology	14nm	7nm	7nm	10nm	7nm	7nm
	Cores per chip	24	48	64	32	48	64
	SMT threads per core	2	2	1	2	2	1
	All-core turbo frequency (GHz)	3.00	2.80	2.50	3.50	2.90	2.60
	Max single-core turbo frequency (GHz)	3.90	3.30	2.50	3.50	3.60	2.60
	L3 cache size (MB)	36	192	32	54	256	32
	DRAM channels per socket	6	8	8	8	8	8
	DRAM Type	DDR4-2933	DDR4-3200	DDR4-3200	DDR4-3200	DDR4-3200	DDR5-4800
	DRAM Bandwidth per socket (GB/s)	141	205	205	205	205	300
	Estimated power consumption per socket (W)	240	280	130	280	300	170

Figure 6.41 The configuration of 6 AWS instances and the CPU chips they use. These 6 instances were available in Q4 of 2022. All instances occupy a full server with one or two CPU sockets. The servers that support these instances are custom designs by AWS. The power consumption listed is the estimated consumption for each socket when running the SPECCPU2017 benchmarks on all cores. The all-core turbo frequency is the maximum clock frequency that each server can sustain when all cores in all sockets are running the SPECCPU2017 benchmarks. The single-core turbo frequency is the maximum clock frequency each server can sustain when a single core runs the SPECCPU2017 benchmarks while all other cores are idle.

benchmark their specific workload across different instances and select the one with hardware that best matches their workload characteristics.

Figure 6.43 shows the SPECCPU2017 performance estimates when using all virtual cores available in each instance. For all integer benchmarks, the m6a

		c5.metal	c5a.24xlarge	c6g.metal	m6i.32xlarge	m6a.48xlarge	c7g.16xlarge
Integer	500.perlbench_r	5.88	4.27	3.68	6.55	5.02	5.09
	502.gcc_r	6.79	6.01	4.58	7.68	7.07	7.27
	505.mcf_r	5.63	5.55	4.95	5.90	6.70	7.15
	520.omnetpp_r	5.95	3.49	3.81	6.76	5.07	5.37
	523.xalancbmk_r	6.54	6.04	4.40	6.76	8.30	6.64
	525.x264_r	9.49	9.14	7.35	10.39	10.85	11.76
	531.deepsjeng_r	4.27	4.07	3.92	4.33	5.10	4.80
	541.leela_r	4.00	3.56	3.54	4.13	4.32	4.23
	548.exchange2_r	11.2	10.15	8.05	11.90	11.83	11.58
	557.xz_r	3.65	2.88	2.58	3.78	3.67	3.25
Floating-point	503.bwaves_r	36.49	61.14	35.68	45.33	74.52	74.39
	507.cactuBSSN_r	5.44	6.31	4.75	6.24	7.68	8.78
	508.namd_r	3.91	4.03	3.30	4.38	4.74	4.98
	510.parest_r	6.44	7.43	5.64	6.68	8.86	8.71
	511.povray_r	6.53	5.73	4.46	7.50	6.98	6.41
	519.lbm_r	3.93	6.21	3.62	4.71	7.55	6.00
	521.wrf_r	9.05	9.92	6.42	9.95	12.2	9.65
	526.blender_r	5.76	5.49	4.41	6.71	6.83	6.36
	527.cam4_r	7.41	8.25	5.12	8.33	9.85	8.16
	538.imagick_r	7.20	6.94	5.68	7.86	8.46	9.60
	544.nab_r	6.20	5.97	4.51	6.96	7.75	6.47
	549.fotonik3d_r	10.22	16.27	10.28	12.31	18.09	24.46
	554.roms_r	6.43	9.42	4.78	7.30	10.83	8.02
	Geomean INT	5.98	5.09	4.44	6.39	6.33	6.21
	Geomean FP	7.22	8.48	5.84	8.23	10.22	9.69

Figure 6.42 The performance estimates for the SPECCPU2017 benchmarks running on a single virtual CPU of the 6 AWS instances. These performance estimates using AWS instances do not represent official SPECCPU2017 for the corresponding CPU chips. For CPUs with SMT cores, these results were measured using one core running in single-threaded mode. All binaries were compiled using GCC10. We highlight with boldface text the instance that achieves the best performance for each benchmark.

		c5.metal	c5a.24xlarge	c6g.metal	m6i.32xlarge	m6a.48xlarge	c7g.16xlarge
Integer	500.perlbench_r	230	226	198	333	478	272
	502.gcc_r	233	199	127	330	436	166
	505.mcf_r	151	124	106	236	261	204
	520.omnetpp_r	146	126	104	200	249	124
	523.xalancbmk_r	227	224	177	327	540	259
	525.x264_r	442	487	440	625	1041	722
	531.deepsjeng_r	226	255	237	315	572	299
	541.leela_r	232	259	222	337	570	270
	548.exchange2_r	511	593	502	828	1214	738
	557.xz_r	166	164	119	223	346	140
Floating-point	503.bwaves_r	486	285.92	360	716	597	630
	507.cactuBSSN_r	213	183.81	171	305	384	332
	508.namd_r	190	224.05	204	276	462	315
	510.parest_r	127	91.15	98	200	264	183
	511.povray_r	258	313.94	276	401	641	407
	519.lbm_r	54	32.55	36	75	71	63
	521.wrf_r	223	157.71	163	344	327	268
	526.blender_r	265	285.33	239	392	632	331
	527.cam4_r	296	271.04	227	410	661	368
	538.imagick_r	321	416.57	341	441	892	611
	544.nab_r	349	376.27	279	526	820	409
	549.fotonik3d_r	143	95.36	107	209	195	183
	554.roms_r	96	69.3	58	150	138	118
	Geomean INT	236	234	192	340	503	266
	Geomean FP	202	174	164	298	380	276

Figure 6.43 The performance estimates for the SPEC CPU2017 benchmarks running all virtual CPUs of the 6 AWS instances. These performance estimates using AWS instances do not represent official SPEC CPU2017 for the corresponding CPU chips. For CPUs with SMT cores, these results were measured while running two SMT threads per core. All binaries were compiled using GCC10. We highlight with boldface text the instance that achieves the best performance for each benchmark.

instance delivers the highest throughput as it supports the largest number of concurrently executing vCPUs, 192 vCPUs on 96 AMD Milan cores with 2 threads per core. For the floating-point benchmarks, the m6i instance, with 128 vCPUs running on 64 Intel Ice Lake cores, is competitive to m6a for one-third of the benchmarks. This suggests that thread count is not the only factor to consider for high throughput computing. System architecture issues such as caches, on-chip interconnect, and power management are important as well.

Absolute performance on a single machine is not the only efficiency metric that cloud providers or customers value. A cloud provider will likely optimize for performance per Watt, with the goal to serve as many customers as possible within the constrained power budget of a WSC. A cloud customer will likely consider performance per cost, looking for the most cost-effective way to achieve a certain performance level. Similarly, a cloud customer may consider performance per virtual CPU, which is the unit of scaling in cloud instances. [Figure 6.44](#) compares the m6i, m6a, and c7g instances for these metrics for the integer and floating-point benchmarks in the SPECCPU2017 suite. The c7g instance based on the Graviton3 CPU provides a significant advantage, 2x to 3x, in terms of performance per watt and performance per cost, the two metrics it was optimized for. This advantage is important for scale-out workloads, either customer applications or cloud services such as databases and analytics as a service, that use tens to

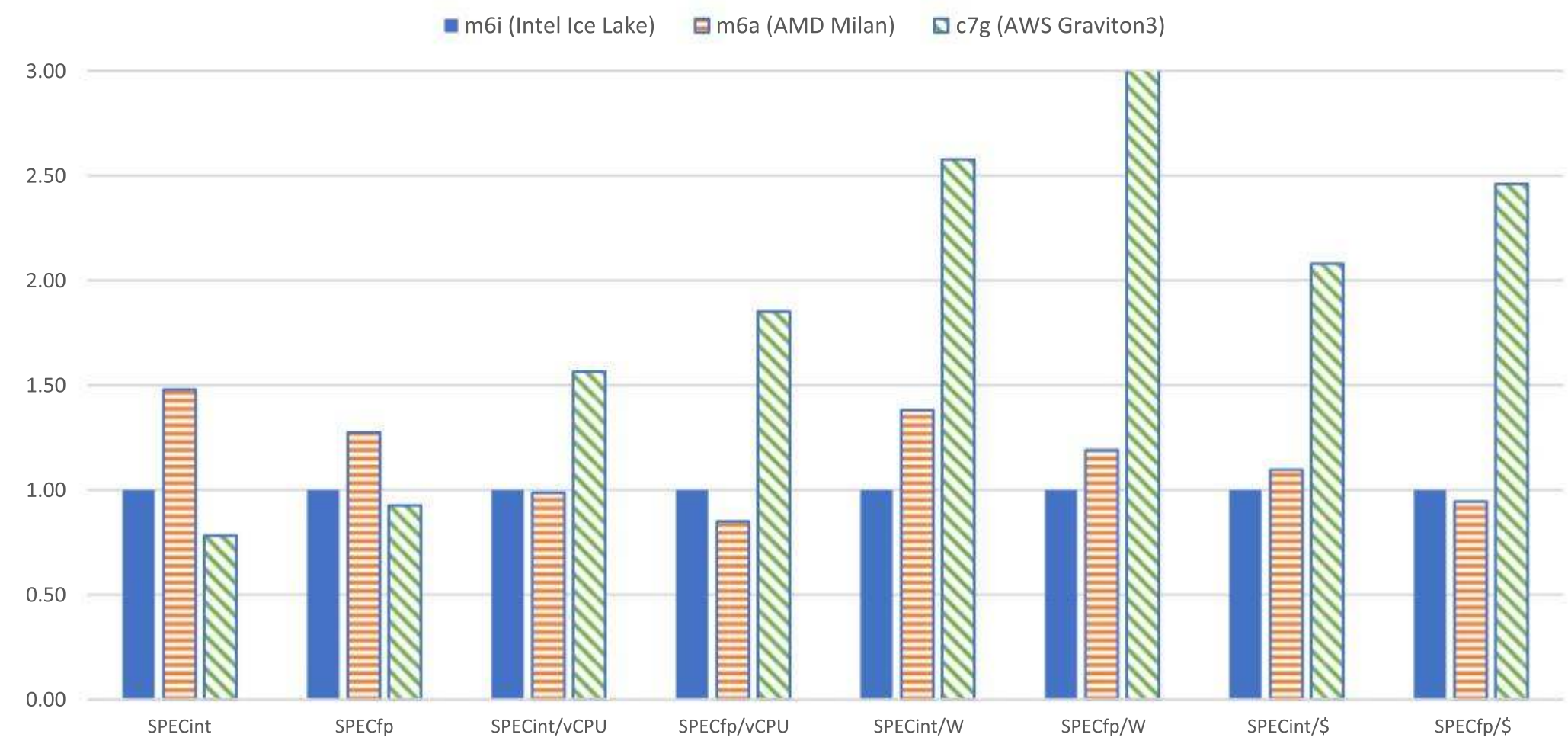


Figure 6.44 Comparison of the m6i, m6a, and c7g instances for the SPECCPU2017 integer and floating-point benchmarks. All metrics are normalized to the m6i instance. For performance, we use the full-system geometric means of SPECCPU2017 integer and floating-point benchmarks from Figure 6.43. For power consumption, we use the per-socket estimates from Figure 6.41. For instance cost, we use the AWS hourly cost for on-demand instances in December 2022.

thousands of instances. It also highlights that cloud providers can get significant end-to-end benefits by building their own CPU chips, even if they start with existing hardware IP (commercial Arm cores in this case).

6.9 Fallacies and Pitfalls

Despite WSC being just 25 years old, WSC architects like those at Google have already uncovered many pitfalls and fallacies about WSCs, often the hard way. As we said in the introduction, WSC architects are today's Seymour Crays.

Fallacy *Cloud computing providers are losing money.*

When AWS was announced, a popular question about cloud computing was whether it was profitable at the low prices at the time. AWS has grown so large that it must be recorded separately in Amazon's quarterly reports. To the surprise of some, AWS has proved to be the most profitable portion of the company. In 2021 AWS had \$62 billion in revenue and an operating profit of \$18.5B, whereas all other Amazon businesses had \$407 billion in revenue and an operating profit of just \$6.5 billion. AWS had \$12.2 billion in revenue for 2016, with an operating margin of 25%, whereas Amazon's retail operations had an operating margin of less than 3%. AWS is consistently responsible for three-fourths of Amazon's profits.

Pitfall. *Focusing on average performance instead of 99th percentile performance.*

Pitfaai. As Dean and Barroso (2013) observed, developers of WSC services worry about the tail more than they care about the mean. If some customers get terrible performance, that experience can drive them away to a competitor, and they'll never return.

Pitfall. *Using too wimpy a processor when trying to improve WSC cost-performance.*

Pitfaiv. Amdahl's law still applies to WSC. There will be some serial work for each request and that can increase request latency if this work runs on a slow server (Hölzle, 2010; Lim et al., 2008). If the serial work increases latency, then the cost of using a wimpy processor must include the software development costs to optimize the code to return it to the lower latency. The larger number of threads of many slow servers can also be more difficult to schedule and load balance, and thus the variability in thread performance can lead to longer latencies. When required to wait for the longest task, a 1-in-1000 chance of bad scheduling is probably not an issue with 10 tasks, but problematic with 1000 tasks.

Pitfav. A decade ago, some expected that custom CPUs will feature a larger count than simpler processor cores than the latest OOO core. This expectation turned out to be false. Cloud providers continue to deploy the latest x86 chips with the latest

OOO core. Some providers have built custom CPU chips using OOO Arm cores that are designed to compete with x86 OOO cores in single-thread performance, such as the AWS Graviton CPUs. For single-machine workloads that move to the cloud, OOO cores are an obvious choice. Even for distributed workloads that have ample request-level parallelism, OOO cores allow for good throughput and latency without significant effort from developers to optimize their code. Despite their inefficiencies and slow rate of improvement OOO cores are still very important for WSC systems. Interestingly, high-end OO cores are now common in smartphones again, in big-little arrangements with simpler cores.

- Pitfavi.** Many smaller servers can also lead to lower utilization because it's clearly easier to schedule fewer things. Finally, even some parallel algorithms get less efficient when the problem is partitioned too finely. The Google rule of thumb is to use the low-end range of server-class computers (Barroso and Hölzle, 2009).
- Pitfavii** As a concrete example, Reddi et al., (2010) compared embedded microprocessors (Atom) and server microprocessors (Nehalem Xeon) running the Bing search engine. They found that the latency of a query was about three times longer on Atom than on Xeon. Moreover, the Xeon was more robust. As load increases on Xeon, quality of service degrades gradually and modestly. The Atom design quickly violates its quality-of-service target as it tries to absorb additional load. Although the Atom design is more energy-efficient, the response time affects revenue, and the revenue loss is likely much greater than the cost savings of less energy. Energy-efficient designs that cannot match the response-time goals are unlikely to be deployed; we'll see another version of this pitfall lesson in the next chapter ([Section 7.9](#)).
- Pitfaviii** This behavior translates directly into search quality. Given the importance of latency to the user, as [Figure 6.37](#) suggests, the Bing search engine uses multiple strategies to refine search results if the query latency has not yet exceeded a cutoff latency. The lower latency of the larger Xeon nodes means that they can spend more time refining search results. Thus, even when the Atom had almost no load, it gave worse answers in 1% of the queries than Xeon. At normal loads, 2% of the answers were worse. Kanev et al., (2015) have more recent, yet consistent, results.
- Pitfall.** *Inconsistent measure of PUE by different companies.*
- Pitfax.** Google's PUE measurements start from the power before it reaches the substation. Some measure at the entrance to the WSC, which skips voltage step-downs that represent a 6% loss. There will also be different results depending on the season of the year if the WSC relies on the atmosphere to help cool the system. Finally, some report the design goal of the WSC instead of measuring the resulting system. The most conservative and best PUE measurement is a running average of the past 12 months of the measured PUE, starting from the feed of the utility.
- Fallacy** *Capital costs of the WSC facility are higher than for the servers that it houses.*

Although a quick look at [Figure 6.33](#) might lead one to that conclusion, that quick glimpse ignores the length of amortization for each part of the full WSC. However, the facility lasts up to 20 years, whereas the servers need to be repurchased every 3 or 4 years. Using the amortization times in [Figure 6.33](#) of 10 years and 3 years, respectively, the capital expenditures over a decade are \$72 million for the facility and $3.3 \times \$67$ million, or \$221 million, for servers. Thus the capital costs for servers in a WSC over a decade are a factor of three higher than for the WSC facility. The TCO breakdown in [Figure 3.34](#) reinforces this point.

- Pitfall.** *Trying to save power with inactive low power modes versus active low power modes.*
- Pitfaii.** [Figure 6.35](#) shows that the average utilization of servers in general clusters can be between 10% and 50%. Given the concern about operational costs of a WSC from [Section 6.7](#), one would think low power modes would be a huge help.
- Pitfaiii.** As [Chapter 1](#) mentions, DRAMs or disks cannot be accessed in these *inactive low power modes*, so they must be returned to fully active mode to read or write, no matter how low the rate. The pitfall is that the time and energy required to return to fully active mode make inactive low power modes less attractive. [Figure 6.35](#) shows that almost all servers average at least 10% utilization, so long periods of low activity might be expected, but not long periods of inactivity (Lo et al., 2014).
- Pitfaiv.** In contrast, processors still run in lower power modes at a small multiple of the regular rate, so *active low power modes* are much easier to use. Note that the time to move to fully active mode for processors is also measured in microseconds, so active low power modes also address the latency concerns about low power modes.
- Fallacy** *Given improvements in DRAM dependability and the fault tolerance of WSC systems software, there is no need to spend extra for ECC memory in a WSC.*

Because ECC adds 8 bits to every 64 bits of DRAM, potentially a ninth of the DRAM costs could be saved by eliminating error-correcting code (ECC), especially since measurements of DRAM have claimed failure rates of 1000–5000 FIT (failures per billion hours of operation) per megabit (Tezzaron Semiconductor, 2004).

Schroeder et al., (2009) studied measurements of the DRAMs with ECC protection at the majority of Google's WSCs, which was surely many hundreds of thousands of servers, over a 2.5-year period. They found 15–25 times higher FIT rates than had been published, or 25,000–70,000 failures per megabit. Failures affected more than 8% of DIMMs, and the average DIMM had 4000 correctable errors and 0.2 uncorrectable errors per year. Measured at the server, about a third experienced DRAM errors each year, with an average of 22,000 correctable errors and 1 uncorrectable error per year. That is, for one-third of the servers, one memory error was corrected every 2.5 h. Note that these systems used the more powerful Chipkill

codes rather than the simpler SECDED codes. If the easier scheme had been used, the uncorrectable error rates would have been 4–10 times higher.

In a WSC that had only parity error protection the servers would have to reboot for each memory parity error. If the reboot time were 5 min, one-third of the machines would spend 20% of their time rebooting! Such behavior would lower the performance of the expensive facility by about 6%. Moreover, these systems would suffer many uncorrectable errors without operators being notified that they occurred.

In the early years Google used DRAM that did not even have parity protection. In 2000, during testing before shipping the next release of the search index, it started suggesting random documents in response to test queries (Barroso and Hölzle, 2009). The reason was a stuck-at-zero fault in some DRAMs, which corrupted the new index. Google added consistency checks to detect such errors in the future. As WSC grew in size and as ECC DIMMs became more affordable, ECC became the standard in Google WSCs. ECC has the added benefit of making it much easier to find broken DIMMs during repair.

Such data suggest why the Fermi GPU ([Chapter 4](#)) adds ECC to its memory, where its predecessors didn't even have parity protection. Moreover, these FIT rates cast doubts on efforts to use the Intel Atom processor in a WSC—because of its improved power efficiency—since the chipset did not support ECC DRAM.

Pitfall. *Coping effectively with microsecond delays as opposed to nanosecond or millisecond delays.*

Pitfallii. Barroso et al., (2017) point out that modern computer systems make it easy for programmers to mitigate latencies in the nanosecond and millisecond timescales (such as cache and DRAM accesses at tens of nanoseconds and disk accesses at a few milliseconds) but that such systems significantly lack support for microsecond-scale events. Programmers get a synchronous interface to the memory hierarchy, with hardware doing heroic work so that such accesses appear consistent and coherent ([Chapter 2](#)). Operating systems offer programmers a similar synchronous interface for a disk read, with many lines of OS code enabling the safe switching to another process while waiting for the disk and then returning again to the original process when the data is ready. We need new mechanisms to cope with the microsecond delays of memory technologies like Flash or the fast network interfaces like 200 Gbit/s Ethernet.

Fallacy *Turning off hardware during periods of low activity improves cost-performance of a WSC.*

[Figure 6.33](#) shows that the cost of amortizing the power distribution and cooling infrastructure is 50% higher than the entire monthly power bill. Thus, although it certainly would save some money to compact workloads and turn off idle machines, even if half the power were saved, the monthly operational bill would be reduced only by 7%. There would also be practical problems to overcome because the extensive WSC monitoring infrastructure depends on being able to poke equipment and see it respond. Another advantage of energy proportionality

and active low power modes is that they are compatible with the WSC monitoring infrastructure, which allows a single operator to be responsible for more than 1000 servers. Note also that preventive maintenance is one of the important tasks that take place during idle time.

Fe The conventional WSC wisdom is to run other valuable tasks during periods of little activity to recoup the investment in power distribution and cooling. A prime example is the batch MapReduce jobs that create indices for search. Another example of getting value from meager utilization is spot pricing on AWS. AWS users who are flexible about when their tasks are run can save up to a factor of four for computation by letting AWS schedule the tasks more flexibly using spot instances, such as when the WSC would otherwise have low utilization.

Fallacy *Flash devices are only cost-effective for warm data that are accessed frequently.*

Fe The cost per bit for Flash is an order of magnitude higher than that of hard disks. Nevertheless, Flash can be cost-effective for some storage systems for cold data. Consider, for instance, the storage system for the photos posted on social networks. Most older photos will rarely get accessed. However, when a user requests an older photo, they expect interactive speeds. A cold storage system that uses hard disks would have to keep the disks spinning to meet SLO latency at the cost of roughly 5 W per disk. A Flash-based cold storage system can idle more efficiently because it has no moving parts and semiconductor chips can exist low power modes faster. The cost of energy may offset the additional cost for the Flash devices.

6.9

Concluding Remarks

Inheriting the title of building the world's biggest computers, computer architects of WSCs are designing a large part of the future IT that supports cloud computing and all the online services that we access through our cellphones. We believe that WSCs are changing the goals and principles of systems design, just as the needs of mobile clients and IoT are changing the goals and principles of microprocessor design. Both are revolutionizing the hardware and software industry, as well. Performance per dollar and performance per joule drive both mobile client hardware and the WSC hardware, and parallelism and domain-specific accelerators are key to delivering on those sets of goals. Architects are playing a vital role in both halves of this exciting future world.

Looking forward, there are exciting opportunities and critical challenges for WSCs. As more workloads migrate to the cloud, there is continuous investment in WSC technology, including custom silicon and custom software stacks for a wide range of WSC tasks. The use of ML techniques to design, optimize, and manage WSC hardware and software is showing promising early results. On the other hand, the challenges of semiconductor technology persist. While

transistor feature size is still scaling, the complexity and cost of semiconductor manufacturing are high, and the energy benefits are decreasing due to the lack of Dennard scaling (see [Chapter 1](#)). There are similar technology challenges for memory and copper link scaling. At the same time, WSC architects have already removed a lot of the low-hanging inefficiencies across the WSC stack. Hence the field will likely need to look for a new set of innovations in the computer architecture of the chips that populate the WSC and the codesigned software stack for them. The DSAs discussed in the next chapter, chiplets, 3D integration, and codesigned optics are some of the hardware developments that can bring sustained improvements to WSCs. Finally, the massive energy requirements of WSCs make it essential to reduce and, if possible, neutralize the environmental impact of these systems. In [Chapter 7](#) we review how operators of state-of-the-art WSCs steer computation to locations with very clean energy, which can reduce carbon dioxide emissions by factors of 5–10. Nevertheless, despite the environmental challenges of WSCs, moving computation from client devices to the WSCs uses nasty-to-produce silicon better. The utilization of silicon in clients such as cellphones is notoriously low.

6.10

Historical Perspectives and References

Section M.8 (available online) covers the development of clusters that were the foundation of WSC and of utility computing. Readers interested in learning more should start with Barroso et al., (2018) and the blog postings of James Hamilton at <http://perspectives.mvdirona.com>, plus his talks at the annual Amazon re:Invent conference.

Case Studies and Exercises by Parthasarathy Ranganathan

Case Study 1: Total Cost of Ownership Influencing Warehouse-Scale Computer Design Decisions

Concepts illustrated by this case study

- Total Cost of Ownership (TCO)
- Influence of Server Cost and Power on the Entire WSC
- Benefits and Drawbacks of Low-Power Servers

- 6.1 Total cost of ownership is an important metric for measuring the effectiveness of a warehouse-scale computer (WSC). TCO includes both the CAPEX and OPEX described in [Section 6.7](#), and reflects the ownership cost of the entire data center to achieve a certain level of performance. When considering different servers, networks, and storage architectures, TCO is often the most important comparison metric used by data center owners to decide which options are best; however,

TCO is a multidimensional computation that takes into account many different factors. The goal of this case study is to take a detailed look into WSCs, to see how different architectures influence TCO, and to understand how TCO drives operator decisions. This case study will use the numbers from [Figures 6.32 and 6.33](#) and [Section 6.7](#) and assumes that the described WSC achieves the operator's target level of performance. TCO is often used to compare different server options that have multiple dimensions. The exercises in this case study examine how such comparisons are made in the context of WSCs and the complexity involved in making the decisions.

[5/5/10] < 6.7 > In this chapter data-level parallelism has been discussed as a way for WSCs to achieve high performance on large problems. Conceivably, even greater performance can be obtained by using high-end servers. However, higher-performance servers often come with a nonlinear price increase.

- a. [5] < 6.7 > Assuming servers that are 10% faster at the same utilization but are 20% more expensive, what is the CAPEX for the WSC?
- b. [5] < 6.7 > If those servers also use 15% more power, what is the OPEX of the WSC?
- c. [10] < 6.7 > Given the speed improvement and power increase, what must the cost of the new servers be to be comparable to the original cluster? (*Hint: Based on this TCO model, you may have to change the critical load of the facility.*)

6.2 [5/10] < 6.7, 6.9 > To achieve a lower OPEX, one appealing alternative is to use low-power versions of servers to reduce the total electricity required to run the servers. However, similar to high-end servers, low-power versions of high-end components also have nonlinear trade-offs.

- a. [5] < 6.7, 6.9 > If low-power server options offered 15% lower power at the same performance but are 20% more expensive, are they a good trade-off?
- b. [10] < 6.7, 6.9 > At what cost do the servers become comparable to the original cluster? What if the price of electricity doubles?

6.3 [5/10/15] < 6.7 > Servers that have different operating modes offer opportunities for dynamically running different configurations in the cluster to match workload usage. Use the data in [Figure 6.45](#) for the power/performance modes for a given low-power server.

- a. [5] < 6.7 > If a server operator decided to save power costs by running all servers at medium performance, how many servers would be needed to achieve the same level of performance?
- b. [10] < 6.7 > What are the CAPEX and OPEX of such a configuration?
- c. [15] < 6.7 > If there was an alternative where you could purchase a server that is 20% cheaper but $x\%$ slower and uses $y\%$ less power, find the performance-power curve that provides a TCO comparable to the baseline server.

Mode	Performance	Power
High	100%	100%
Medium	75%	60%
Low	59%	38%

Figure 6.45 Power-performance modes for low-power servers.

- 6.4 [Discussion] < 6.7 > Discuss the trade-offs and benefits of the two options in Exercise 6.3, assuming a constant workload being run on the servers.
- 6.5 [Discussion] < 6.1, 6.7 > Unlike high-performance computing (HPC) clusters, WSCs often experience significant workload fluctuation throughout the day. Discuss the trade-offs and benefits of the two options in Exercise 6.3, this time assuming a workload that varies.
- 6.6 [Discussion] < 6.7 > The TCO model presented so far abstracts away a significant amount of lower-level details. Discuss the impact of these abstractions on the overall accuracy of the TCO model. When are these abstractions safe to make? In what cases would greater detail provide significantly different answers?

Case Study 2: Resource Allocation in WSCs and TCO

Concepts illustrated by this case study

- Server and Power Provisioning Within a WSC
 - Time Variance of Workloads
 - Effects of Variance on TCO
- 6.7 Some of the key challenges to deploying efficient WSCs are provisioning resources properly and utilizing them to their fullest capacity. This problem is complex due to the size of WSCs as well as the potential variance of the workloads being run. The exercises in this case study show how different uses of resources can affect TCO. Assume data from [Figures 6.32 and 6.33](#) as appropriate.

[5/5/10] < 6.7 > One of the challenges in provisioning a WSC is determining the proper power load, given the facility size. As described in the chapter, nameplate power is often a peak value that is rarely encountered.

 - a. [5] < 6.7 > Estimate how the per-server TCO changes if the nameplate server power is 200 W and the cost is \$3000.
 - b. [5] < 6.7 > Also consider a higher power but cheaper server option whose power is 300 W and costs \$2000.
 - c. [10] < 6.7 > How does the per-server TCO change if the actual average power usage of the servers is only 70% of the nameplate power?
 - 6.8 [15/10] < 6.7 > One assumption in the TCO model is that the critical load of the facility is fixed, and the amount of servers fits that critical load. In reality, due to

the variations of server power based on load, the critical power used by a facility can vary at any given time. Operators must initially provision the data center based on its critical power resources and an estimate of how much power is used by the data center components.

- a. [15] < 6.7 > Extend the TCO model to initially provision a WSC based on a server with a nameplate power of 300 W but also calculate the actual monthly critical power used and TCO assuming the server averages 40% utilization and so consumes only 225 W. How much capacity is left unused?
 - b. [10] < 6.7 > Repeat this exercise with a 500-W server that averages 20% utilization and consumes only 300 W.
- 6.9 [10] < 6.2, 6.7 > WSCs are often used in an interactive manner with end users, as mentioned in [Section 6.7](#). This interactive usage often leads to time-of-day fluctuations, with peaks correlating to specific time periods. For example, for Netflix rentals there is a peak during the evening periods of 8–10 p.m.; the entirety of these time-of-day effects is significant. Compare the per-server TCO of a data center with a capacity to match the utilization at 4 a.m. compared to 9 p.m.
- 6.10 [Discussion/15] < 6.2, 6.7 > Discuss some options to better utilize the excess servers during the off-peak hours or find ways to save costs. Given the interactive nature of WSCs, what are some of the challenges to aggressively reducing power usage?
- 6.11 [Discussion/25] < 6.7, 6.9 > Propose one possible way to improve TCO by focusing on reducing server power. What are the challenges to evaluating your proposal? Estimate the TCO improvements based on your proposal. What are some advantages and drawbacks?

Exercises

- 6.12 [10/10/10] < 6.1 > One of the important enablers of WSC is ample request-level parallelism, in contrast to instruction- or thread-level parallelism. This question explores the implication of different types of parallelism on computer architecture and system design.
- a. [10] < 6.1 > Discuss scenarios where improving the instruction- or thread-level parallelism would provide greater benefits than those achievable through request-level parallelism.
 - b. [10] < 6.1 > What are the software design implications of increasing request-level parallelism?
 - c. [10] < 6.1 > What are the potential drawbacks of increasing request-level parallelism?
- 6.13 [Discussion/15/15] < 6.3, 6.4 > When a cloud computing service provider receives jobs consisting of multiple virtual machines (VMs) (e.g., a MapReduce

job), many scheduling options exist. The VMs can be scheduled in a round-robin manner to spread across all available processors and servers, or they can be consolidated to use as few processors as possible. Using these scheduling options, if a job with 24 VMs was submitted and 30 processors were available in the cloud (each able to run up to 3 VMs), round-robin would use 24 processors, while consolidated scheduling would use 8 processors. The scheduler can also find available processor cores at different scopes: socket, server, rack, and an array of racks.

- a. [Discussion] < 6.4 > Assuming that the submitted jobs are all compute-heavy workloads, possibly with different memory bandwidth requirements, what are the pros and cons of round-robin versus consolidated scheduling in terms of power and cooling costs, performance, and reliability?
- b. [15] < 6.4 > Assuming that the submitted jobs are all I/O-heavy workloads, what are the pros and cons of round-robin versus consolidated scheduling, at different scopes?
- c. [15] < 6.4 > Assuming that the submitted jobs are network-heavy workloads, what are the pros and cons of round-robin versus consolidated scheduling, at different scopes?

6.14 [15/15/10/10] < 6.4 > Distributed computing frameworks like MapReduce enable large amounts of parallelism by having data-independent tasks run on multiple nodes, often using commodity hardware; however, there are limits to the level of parallelism. For example, for redundancy MapReduce will write data blocks to multiple nodes, consuming disk and, potentially, network bandwidth. Assume a total dataset size of 300 GB, a network bandwidth of 1 Gb/s, a 10 s/GB map rate, and a 20 s/GB reduce rate. Also assume that 30% of the data must be read from remote nodes, and each output file is written to two other nodes for redundancy. Use [Figure 6.21](#) for all other parameters.

- a. [15] < 6.4 > Assume that all nodes are in the same rack. What is the expected runtime with 5 nodes? 10 nodes? 100 nodes? 1000 nodes? Discuss the bottlenecks at each node size.
- b. [15] < 6.4 > Assume that there are 40 nodes per rack and that any remote read/write has an equal chance of going to any node. What is the expected runtime at 100 nodes? 1000 nodes?
- c. [10] < 6.4 > An important consideration is minimizing data movement as much as possible. Given the significant slowdown of going from local to rack to array accesses, software must be strongly optimized to maximize locality. Assume that there are 40 nodes per rack, and 1000 nodes are used in the MapReduce job. What is the runtime if remote accesses are within the same rack 20% of the time? 50% of the time? 80% of the time?

6.15 [20/20/10/20/20/20] < 6.1, 6.4 > WSC programmers often use data replication to overcome failures in the software. The Hadoop distributed file system (HDFS), for

example, employs three-way replication (one local copy, one remote copy in the rack, and one remote copy in a separate rack), but it's worth examining when such replication is needed.

- a. [20] < 6.1 > Let us assume that Hadoop clusters are relatively small, with 10 nodes or less, and with dataset sizes of 10 TB or less. Using the failure frequency data in [Figure 6.1](#), what kind of availability does a 10-node Hadoop cluster have with one-, two-, and three-way replications?
 - b. [20] < 6.1 > Assuming the failure data in [Figure 6.1](#) and a 1000-node Hadoop cluster, what kind of availability does it have with one-, two-, and three-way replications? What can you reason about the benefits of replication, at scale?
 - c. [10] < 6.1, 6.4 > The relative overhead of replication varies with the amount of data written per local compute hour. Calculate the amount of extra I/O traffic and network traffic (within and across rack) for a 1000-node Hadoop job that sorts 1 PB of data, where the intermediate results for data shuffling are written to the HDFS.
 - d. [20] < 6.1, 6.4 > Using [Figure 6.21](#), calculate the time overhead for two- and three-way replications. Using the failure rates shown in [Figure 6.1](#), compare the expected execution times for no replication versus two- and three-way replications.
 - e. [20] < 6.1, 6.4 > Now consider a database system applying replication on logs, assuming each transaction on average accesses the hard disk once and generates 1 KB of log data. Calculate the time overhead for two- and three-way replications. What if the transaction is executed in-memory and takes 10 μ s?
 - f. [20] < 6.1, 6.4 > Now consider a different database system that requires two network round trips for two-phase commitment (for ACID consistency). What is the time overhead for maintaining consistency as well as replications?
- 6.16 [15/15/20/Discussion] < 6.1, 6.9 > Although request-level parallelism allows many machines to work on a single problem in parallel, thereby achieving greater overall performance, one of the challenges is how to avoid dividing the problem too finely. If we look at this problem in the context of service-level agreements (SLAs), using smaller problem sizes through greater partitioning can require increased effort to achieve the target SLA. Assume an SLA of 95% of queries respond at 0.5 seconds or faster, and a parallel architecture similar to MapReduce that can launch multiple redundant jobs to achieve the same result. For the following questions, assume the query-response time curve shown in [Figure 6.46](#). The curve shows the latency of response, based on the number of queries per second, for a baseline server as well as a “small” server that uses a slower processor model.
- a. [15] < 6.1, 6.9 > How many servers are required to achieve this SLA, assuming the query-response time curve shown in [Figure 6.46](#) and the WSC receiving 30,000 queries per second? How many “small” servers are required

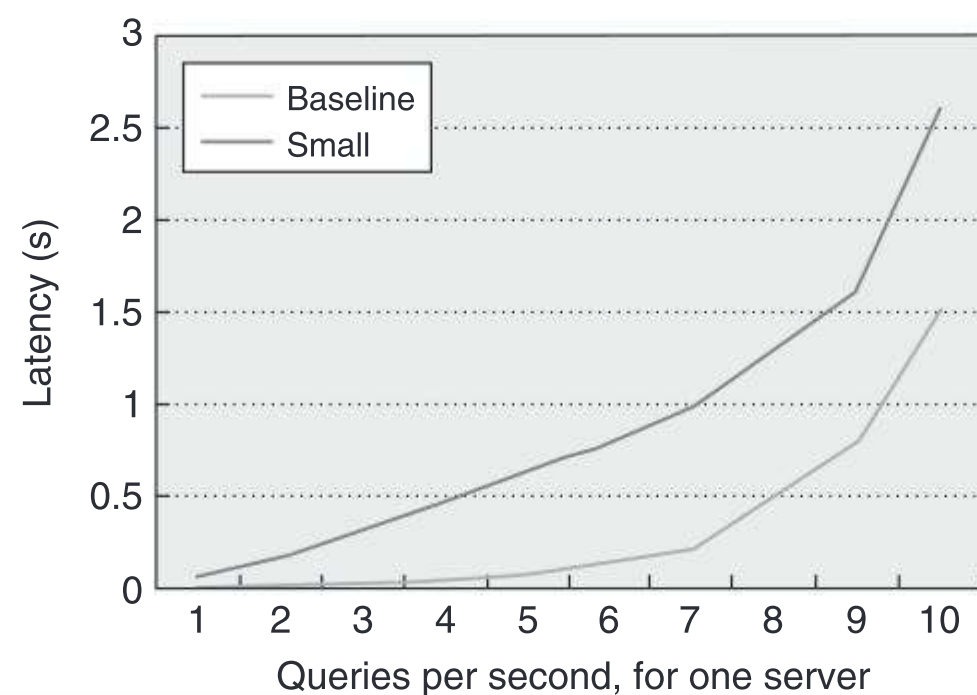


Figure 6.46 Query-response time curve.

to achieve this SLA, given this response-time probability curve? Looking only at server costs, how much cheaper must the “small” servers be than the normal servers to achieve a cost advantage for the target SLA?

- b. [15] < 6.1, 6.9 > Often, “small” servers are also less reliable due to cheaper components. Using the numbers from [Figure 6.1](#), assume that the number of events due to flaky machines and bad memories increases by 30%. How many “small” servers are required now? How much cheaper must those servers be than the standard servers?
- c. [20] < 6.1, 6.9 > Now assume a batch processing environment. The “small” servers provide 30% of the overall performance of the regular servers. Still assuming the reliability numbers from Exercise 6.15 part (b), how many “small” nodes are required to provide the same expected throughput of a 2400-node array of standard servers, assuming perfect linear scaling of performance to node size and an average task length of 10 min per node? What if the scaling is 85%? 60%?
- d. [Discussion] < 6.1, 6.9 > Often the scaling is not a linear function but instead a logarithmic function. A natural response may be instead to purchase larger nodes that have more computational power per node to minimize the array size. Discuss some of the trade-offs with this architecture.

6.17 [10/10/15/Discussion] < 6.4, 6.9 > Let us consider the inclusion of nonvolatile flash memory in the memory hierarchy, either through solid-state disks (SSDs) or PCI Express-attached cards. Typical SSDs have a bandwidth of 250 MB/s and latency of 75 μ s, whereas PCIe cards have a bandwidth of 600 MB/s and latency of 35 μ s.

- a. [10] Take [Figure 6.22](#) and include these points in the local server hierarchy. Assuming that identical performance scaling factors like DRAM are accessed

at different hierarchy levels, how do these flash memory devices compare when accessed across the rack? Across the array?

- b. [10] Discuss some software-based optimizations that can utilize the new level of the memory hierarchy.
 - c. [15] As discussed in “Fallacies and Pitfalls” (Section 6.9), replacing all disks with SSDs is not necessarily a cost-effective strategy. Consider a WSC operator that uses it to provide cloud services. Discuss some scenarios where using SSDs or other flash memory would make sense.
 - d. [Discussion] Discuss if there is an opportunity for other layers in the memory hierarchy - slower memory (e.g. disaggregated memory) or slower flash (more cost-optimized technologies).
- 6.18 [20/20/Discussion] < 6.4 > *Memory Hierarchy*: Caching is heavily used in some WSC designs to reduce latency, and there are multiple caching options to satisfy varying access patterns and requirements.
- a. [20] Let’s consider the design options for streaming rich media from the Web (e.g., Netflix or YouTube). First we need to estimate the number of videos, number of encode formats per video, and concurrent viewing users. Assume a streaming video provider that has 12,000 titles for online streaming, with each title having at least four encode formats (at 500, 1000, 1600, and 2200 kbps). Let’s also assume that there are 100,000 concurrent viewers for the entire site, and an average video is 75 min long (accounting for both 30-min shows and 2-h videos). Estimate the total storage capacity, I/O and network bandwidths, and video streaming—related computation requirements.
 - b. [20] What are the access patterns and reference locality characteristics per user, per video, and across all videos? (*Hint*: Random versus sequential, good versus poor temporal and spatial locality, relatively small versus large working set size.)
 - c. [Discussion] What movie storage options exist by using DRAM, SSD, and hard drives? Compare them in performance and TCO. Would alternate memory technologies like those in Problem 6.17(d) be useful?
- 6.19 [20/Discussion/Discussion/Discussion] < 6.4 > Consider a social networking website with 100 million active users posting updates about themselves (in text and pictures) as well as browsing and interacting with updates in their social networks. To provide low latency, Facebook and many other websites use *memcached* as a caching layer before the back-end storage/database tiers. Assume that at any given time the average user is browsing megabytes of content, and on any given day the average user uploads megabytes of content.
- a. [20] For the social networking website discussed here, how much DRAM is needed to host its working set? Using servers each having 96 GB DRAM, estimate how many local versus remote memory accesses are needed to generate a user’s home page.

- b. [Discussion] Now consider two candidate *memcached* server designs, one using performance-optimized higher-energy cores and the other using smaller energy-efficient cores (for example, E-cores and P-cores announced by Intel and AMD). Given that *memcached* requires large physical memory but has low CPU utilization, what are the pros and cons of these two designs?
- c. [Discussion] Today's tight coupling between memory modules and processors often requires an increase in CPU socket count in order to provide large memory support. List other designs to provide large physical memory without proportionally increasing the number of sockets in a server. Compare them based on performance, power, costs, and reliability.
- d. [Discussion] The same user's information can be stored in both the *memcached* and storage servers, and such servers can be physically hosted in different ways. Discuss the pros and cons of the following server layout in the WSC: (1) *memcached* collocated on the same storage server, (2) *memcached* and storage servers on separate nodes in the same rack, or (3) *memcached* servers on the same racks and storage servers collocated on separate racks.

6.20 [5/5/10/10/Discussion/Discussion/Discussion] < 6.2, 6.4, 6.7 > *Data center Networking*: Distributed frameworks (like MapReduce) and warehouse-scale computers are a powerful combination to tackle large-scale data processing. For this problem, we will assume we sort one petabyte (1 PB) of records in 6 hours using 4000 servers and 48,000 hard drives.

- a. [5] Derive disk bandwidth from [Figure 6.21](#) and associated text. How many seconds does it take to read the data into main memory and write the sorted results back?
- b. [5] Assuming each server has two 1 Gb/s Ethernet network interface cards (NICs) and the WSC switch infrastructure is oversubscribed by a factor of 4, how many seconds does it take to shuffle the entire dataset across 4000 servers?
- c. [10] Assuming network transfer is the performance bottleneck for petabyte sort, can you estimate what oversubscription ratio Google has in its data center?
- d. [10] Now let's examine the benefits of having 10 Gb/s Ethernet without oversubscription—for example, using a 48-port 10 Gb/s Ethernet (this was used by the 2010 Indy sort benchmark winner TritonSort). How long does it take to shuffle 1 PB of data?
- e. [Discussion] Compare the two approaches here: (1) the massive scale-out approach with high network oversubscription ratio, and (2) a relatively small-scale system with a high-bandwidth network. What are their potential bottlenecks? What are their advantages and disadvantages, in terms of scalability and TCO?
- f. [Discussion] Sort and many important scientific computing workloads are communication-heavy, while many other workloads are not. List three

example workloads that do not benefit from high-speed networking. What EC2 instances would you recommend to use for these two classes of workloads?

- g. [Discussion] Look up the various benchmarks in www.sortbenchmark.org and recent winners in each category. How do these results match the insights from the discussion in part (e)? How does the cloud instance used for the most recent winner of CloudSort compare with your answer in part (f)?

- 6.21 [10/25/Discussion] < 6.4, 6.7 > Because of the massive scale of WSCs, it is very important to properly allocate network resources based on the workloads that are expected to be run. Different allocations can have significant impacts on both the performance and total cost of ownership.
- a. [10] Using the numbers in the spreadsheet detailed in Figure 6.32, what is the oversubscription ratio at each access-layer switch? What is the impact on TCO if the oversubscription ratio is cut in half? What if it is doubled?
- b. [25] Reducing the oversubscription ratio can potentially improve performance if a workload is network limited. Assume a computational job that uses 120 servers and reads 5 TB of data. Assume the same ratio of read/intermediate/output data as in Figure 6.47, Sep-09, and use Figure 6.21 to define the bandwidths of the memory hierarchy. When reading data, assume that 50% of data is read from remote disks; of that, 80% is read from within the rack, and 20% is read from within the array. For intermediate data and output data, assume that 30% of the data uses remote disks; of that, 90% is within the rack and 10% is within the array. What is the overall performance improvement when reducing

Month	Number of MapReduce Jobs	Average completion time (s)	Average no. servers per job	Avg. no. cores per server	CPU core years	Input data (PB)	Intermediate data (PB)	Output data (PB)
Sep-16	95,775,891	331	130	2.4	311,691	11,553	4095	6982
Sep-15	115,375,750	231	120	2.7	272,322	8307	3980	5801
Sep-14	55,913,646	412	142	1.9	200,778	5989	2530	3951
Sep-13	28,328,775	469	137	1.4	81,992	2579	1193	1684
Sep-12	15,662,118	480	142	1.8	60,987	2171	818	874
Sep-11	7,961,481	499	147	2.2	40,993	1162	276	333
Sep-10	5,207,069	714	164	1.6	30,262	573	139	37
Sep-09	4,114,919	515	156	3.2	33,582	548	118	99
Sep-07	2,217,000	395	394	1.0	11,081	394	34	14
Mar-06	171,000	874	268	1.6	2002	51	7	3
Aug-04	29,000	634	157	1.9	217	3.2	0.7	0.2

Figure 6.47 Monthly MapReduce usage at Google from 2004 to 2016. Over 12 years, the number of MapReduce jobs increased by a factor of 3300. Updated from [Dean, J., 2009].

- the oversubscription ratio by half? What is the performance if the oversubscription ratio is doubled? Calculate the TCO in each case.
- c. [Discussion] How do you see emerging technology trends affecting the design of future WSCs? Some examples include the trend to more and more cores per system (including using chiplets), as well as higher bandwidth and improved energy efficiency from optics.

- 6.22 [5/15/15/20/25/Discussion/Discussion] < 6.2 > *Realizing the Capability of the Cloud*: Imagine you are the site operation and infrastructure manager of the Alexa.com top site and are considering using Amazon Web Services (AWS). What factors do you need to consider when determining whether to migrate to AWS? What services and instance types could you use, and how much cost could you save? You can use Alexa and site traffic information (e.g., Wikipedia provides page view stats) to estimate the amount of traffic received by a top site, or you can take concrete examples from the Web, such as the following example: <http://2bits.com/sites/2bits.com/files/drupal-single-server-2.8-million-page-views-a-day.pdf>. The slides describe an Alexa #3400 site that receives 2.8 million page views per day, using a single server. The server has two quad-core Xeon 2.5 GHz processors with 8 GB DRAM and three 15 K RPM SAS hard drives in a RAID1 configuration, and it costs about \$400 per month. The site uses caching heavily, and the CPU utilization ranges from 50% to 250% (roughly 0.5–2.5 cores busy).
- [5] Looking at the available EC2 instances (<http://aws.amazon.com/ec2/instance-types/>), what instance types match or exceed the current server configuration?
 - [15] Looking at the EC2 pricing information (<http://aws.amazon.com/ec2/pricing/>), select the most cost-efficient EC2 instances (combinations allowed) to host the site on AWS. What is the monthly cost for EC2?
 - [15] Now add the costs for IP address and network traffic to the equation, and suppose the site transfers 100 GB/day in and out on the Internet. What is the monthly cost for the site now?
 - [20] AWS also offers a microinstance for free for 1 year to new customers and 15 GB bandwidth each for traffic going in and out across AWS. Based on your estimation of peak and average traffic from your department's Web server, can you host it for free on AWS?
 - [25] Based on the service characteristics, if a much larger site like Netflix.com migrates its streaming and encoding infrastructure to AWS, what AWS services could be used by Netflix and for what purposes?
 - [Discussion] Look at similar offerings from other cloud providers (Google, Microsoft, Alibaba, etc.). How do the answers to parts (a)–(e) change?
 - [Discussion] “Serverless computing” allows you to build and run higher-level applications and services without thinking about specific servers. Examples include AWS Lambda, Google Cloud Functions, Microsoft Azure Functions,

etc. Continuing to wear your site operation and infrastructure manager hat, when would you consider serverless computing?

- 6.23 [Discussion/Discussion/20/20/Discussion] < 6.7, 6.9 > [Figure 6.36](#) shows the impact of user-perceived response time on revenue and motivates the need to achieve high throughput while maintaining low latency.
- [Discussion] Taking Web search as an example, what are the possible ways of reducing query latency?
 - [Discussion] What monitoring statistics can you collect to help understand where time is spent? How do you plan to implement such a monitoring tool?
 - [20] Assuming that the number of disk accesses per query follows a normal distribution, with an average of 2 and standard deviation of 3, what kind of disk access latency is needed to satisfy a latency SLA of 0.1 s for 95% of the queries?
 - [20] In-memory caching can reduce the frequencies of long-latency events (e.g., accessing hard drives). Assuming a steady-state hit rate of 40%, hit latency of 0.05 s, and miss latency of 0.2 s, does caching help meet a latency SLA of 0.1 s for 95% of the queries?
 - [Discussion] When can cached content become stale or even inconsistent? How often can this happen? How can you detect and invalidate such content?
- 6.24 [15/15/20/Discussion] < 6.6, 6.7 > The efficiency of typical power supply units (PSUs) varies as the load changes; for example, PSU efficiency can be about 80%—40% load (e.g., output 40 W from a 100-W PSU), 75% when the load is between 20% and 40%, and 65% when the load is below 20%.
- [15] Assume a power-proportional server whose actual power is proportional to CPU utilization, with a utilization curve as shown in [Figure 6.48](#). What is the average PSU efficiency?
 - [15] Suppose the server employs $2N$ redundancy for PSUs (i.e., doubles the number of PSUs) to ensure stable power when one PSU fails. What is the average PSU efficiency?
 - [20] Blade server vendors use a shared pool of PSUs not only to provide redundancy but also to dynamically match the number of PSUs to the server's actual power consumption. Assume a blade enclosure that uses up to six PSUs for a total of 16 servers. In this case what is the average PSU efficiency for the enclosure for a server with the same utilization curve?
 - [Discussion] Consider the impact of the different efficiency numbers in the context of the broader TCO discussions in [Figures 6.32 and 6.33](#): how do the different designs impact the total TCO? Given these, how would you optimize designs for future warehouse-scale computers?

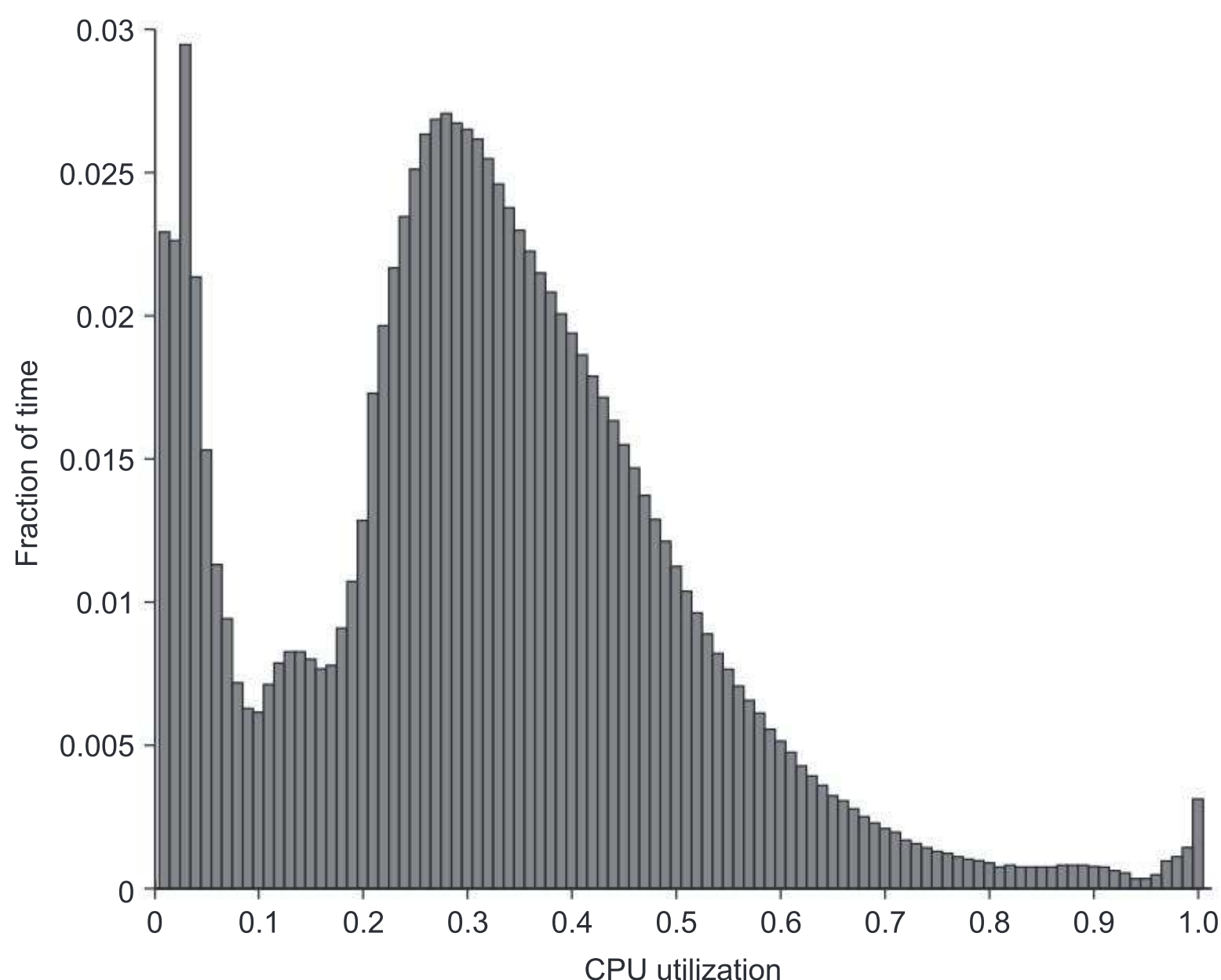


Figure 6.48 Average CPU utilization of more than 5000 servers during a 6-month period at Google. Servers are rarely completely idle or fully utilized, instead operating most of the time at between 10% and 50% of their maximum utilization. The third column from the right in Figure 6.50 calculates percentages plus or minus 5% to come up with the weightings; thus 1.2% for the 90% row means that 1.2% of servers were between 85% and 95% utilized. From Figure 1 in [Barroso and Hölzle, 2007].

- 6.25 [5/Discussion/10/15/Discussion/Discussion/Discussion] < 6.6, 6.9 > *Power stranding* is a term used to refer to power capacity that is provisioned but not used in a data center. Consider the data presented in Figure 6.49 [Fan, Weber, and Barroso, 2007] for different groups of machines. (Note that what this paper calls a “cluster” is what we have referred to as an “array” in this chapter.)
- [5] What is the stranded power at (1) the rack level, (2) the power distribution unit level, and (3) the array (cluster) level? What are the trends with oversubscription of power capacity at larger groups of machines?
 - [Discussion] What do you think causes the differences between power stranding at different groups of machines?
 - [10] Consider an array-level collection of machines where the total machines never use more than 72% of the aggregate power (this is sometimes also referred to as the ratio between the peak-of-sum and sum-of-peaks usage). Using the cost model in the case study, compute the cost savings from comparing a data center provisioned for peak capacity and one provisioned for actual use.

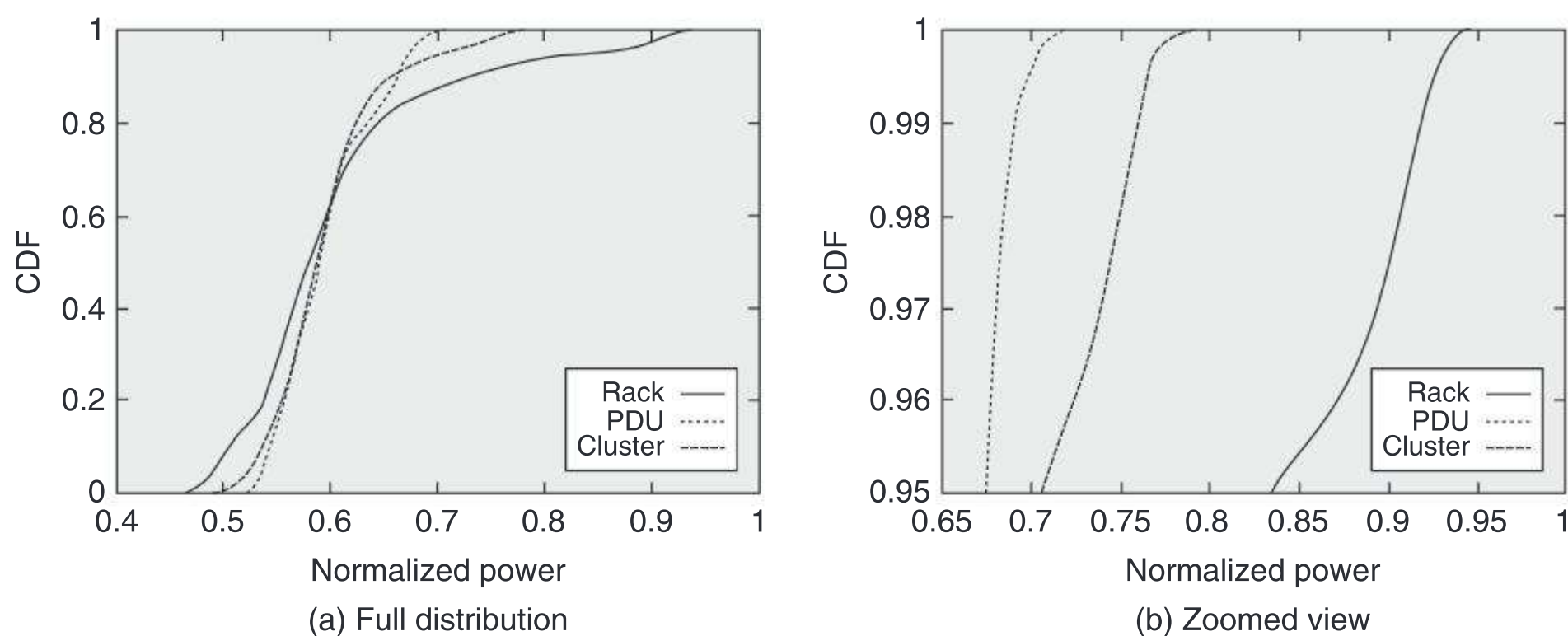


Figure 6.49 Cumulative distribution function (CDF) of a real data center.

- d. [15] Assume that the data center designer chose to include additional servers at the array level to take advantage of the stranded power. Using the example configuration and assumptions in part (a), compute how many more servers can now be included in the warehouse-scale computer for the same total power provisioning.
 - e. [Discussion] What is needed to make the optimization of part (d) work in a real-world deployment? (*Hint*: Think about what needs to happen to cap power in the rare case when all the servers in the array are used at peak power.)
 - f. [Discussion] Two kinds of policies can be envisioned to manage power caps [Ranganathan et al., 2006]: (1) preemptive policies where power budgets are predetermined (“don’t assume you can use more power; ask before you do!”) or (2) reactive policies where power budgets are throttled in the event of a power budget violation (“use as much power as needed until told you can’t!”). Discuss the trade-offs between these approaches and when you would use each type.
 - g. [Discussion] What happens to the total stranded power if systems become more energy proportional?
- 6.26 [5/20/Discussion] < 6.6, 6.7 > [Section 6.6](#) discussed the use of per-rack batteries as local UPS. One can take this idea further and use per-server batteries. Let us examine the consequences of this design.
- a. [5] Assume that the use of a battery as a mini-server-level UPS is 99.99% efficient and eliminates the need for a facility-wide UPS that is only 92% efficient. Assume that substation switching is 99.7% efficient and that the efficiency for the PDU, step-down stages, and other electrical breakers are 98%, 98%, and 99%, respectively. Calculate the overall power infrastructure efficiency improvements from using a per-server battery backup.

- b. [20] Assume that the UPS is 10% of the cost of the IT equipment. Using the rest of the assumptions from the cost model in the case study, what is the break-even point for the costs of the battery (as a fraction of the cost of a single server) at which the total cost of ownership for a battery-based solution is better than that for a facility-wide UPS?
 - c. [Discussion] What are the other trade-offs between these two approaches? In particular, how do you think the manageability and failure model will change across these two different designs?

- 6.27 [5/5/Discussion] < 6.6, 6.7 > For this exercise, consider a simplified equation for the total operational power of a WSC as follows:
 - a. [5] Assume an 8 MW data center at 80% power usage, electricity costs of \$0.10 per kilowatt-hour, and a cooling-inefficiency multiplier of 0.8. Compare the cost savings from (1) an optimization that improves cooling efficiency by 20% and (2) an optimization that improves the energy efficiency of the IT equipment by 20%.
 - b. [5] What is the percentage improvement in IT equipment energy efficiency needed to match the cost savings from a 20% improvement in cooling efficiency?
 - c. [Discussion] What conclusions can you draw about the relative importance of optimizations that focus on server energy efficiency and cooling energy efficiency?

- 6.28 [5/5/Discussion] < 6.6, 6.7 > As discussed in this chapter, pieces of the cooling equipment in WSCs can themselves consume a lot of energy. Cooling costs can be lowered by proactively managing temperature. Temperature-aware workload placement is one optimization that has been proposed to manage temperature to reduce cooling costs. The idea is to identify the cooling profile of a given room and map the hotter systems to the cooler spots so that at the WSC level the requirements for overall cooling are reduced.
 - a. [5] The coefficient of performance (COP) of a computer room air conditioning unit (CRAC) is a measure of its efficiency and is defined as the ratio of heat removed (Q) to the amount of work necessary (W) to remove that heat. The COP of a CRAC unit increases with the temperature of the air the CRAC unit pushes into the plenum. If air returns to the CRAC unit at 20°C and we remove 10 KW of heat with a COP of 1.9, how much energy do we expend in the CRAC unit? If it takes a COP of 3.1 to cool the same volume of air, but the air is returned at 25°C, how much energy do we now expend in the CRAC unit?
 - b. [5] Assume a workload distribution algorithm is able to match the hot workloads well with the cool spots to allow the CRAC unit to be run at a higher temperature to improve cooling efficiencies like in the prior exercise. What is the power savings between the two cases described previously?

- c. [Discussion] Given the scale of WSC systems, power management can be a complex, multifaceted problem. Optimizations to improve energy efficiency can be implemented in hardware and in software, at the system level, and at the cluster level for the IT equipment or the cooling equipment, etc. It is important to consider these interactions when designing an overall energy-efficiency solution for the WSC. Consider a consolidation algorithm that looks at server utilization and consolidates different workload classes on the same server to increase server utilization (this can potentially have the server operating at a higher energy efficiency if the system is not energy proportional). How would this optimization interact with a concurrent algorithm that tried to use different power states (see ACPI, Advanced Configuration Power Interface, for some examples)? What other examples can you think of where multiple optimizations can potentially conflict with one another in a WSC? How would you solve this problem?

6.29 [5/10/15/20] < 6.7 > Energy proportionality (sometimes also referred to as energy scale-down) is the attribute of the system to consume no power when idle, but more importantly, gradually consume more power *in proportion* to the activity level and work done. In this exercise we will examine the sensitivity of energy consumption to different energy proportionality models. In the exercises below, unless otherwise mentioned, use the data in [Figure 6.50](#) as the default.

- a. [5] A simple way to reason about energy proportionality is to assume linearity between activity and power usage. Using just the peak power and idle power

Load	Performance	Watts	SPEC weightings	Weighted performance	Weighted watts	Figure 6.3 weightings	Weighted performance	Weighted watts
100%	2,889,020	662	9.09%	262,638	60	0.80%	22,206	5
90%	2,611,130	617	9.09%	237,375	56	1.20%	31,756	8
80%	2,319,900	576	9.09%	210,900	52	1.50%	35,889	9
70%	2,031,260	533	9.09%	184,660	48	2.10%	42,491	11
60%	1,740,980	490	9.09%	158,271	45	5.10%	88,082	25
50%	1,448,810	451	9.09%	131,710	41	11.50%	166,335	52
40%	1,159,760	416	9.09%	105,433	38	19.10%	221,165	79
30%	869,077	382	9.09%	79,007	35	24.60%	213,929	94
20%	581,126	351	9.09%	52,830	32	15.30%	88,769	54
10%	290,762	308	9.09%	26,433	28	8.00%	23,198	25
0%	0	181	9.09%	0	16	10.90%	0	20
Total	15,941,825	4967		1,449,257	452		933,820	380
				ssj_ops/W	3210		ssj_ops/W	2454

Figure 6.50 SPECpower result using the weightings from Figure 6.48 instead of even weightings.

- data from [Figure 6.50](#) and a linear interpolation, plot the energy-efficiency trends across varying utilizations. (Energy efficiency is expressed as performance per watt.) What happens if idle power (at 0% activity) is half of what is assumed in [Figure 6.50](#)? What happens if idle power is zero?
- b. [10] Plot the energy-efficiency trends across varying activities, but use the data from column 3 of [Figure 6.50](#) for power variation. Plot the energy efficiency assuming that the idle power (alone) is half of what is assumed in [Figure 6.50](#). Compare these plots with the linear model in the previous exercise. What conclusions can you draw about the consequences of focusing purely on idle power alone?
 - c. [15] Assume the system utilization mix in column 7 of [Figure 6.50](#). For simplicity, assume a discrete distribution across 1000 servers, with 109 servers at 0% utilization, 80 servers at 10% utilization, etc. Compute the total performance and total energy for this workload mix using the assumptions in part (a) and part (b).
 - d. [20] One could potentially design a system that has a sublinear power versus load relationship in the region of load levels between 0% and 50%. This would have an energy-efficiency curve that peaks at lower utilizations (at the expense of higher utilizations). Create a new version of column 3 from [Figure 6.50](#) that shows such an energy-efficiency curve. Assume the system utilization mix in column 7 of [Figure 6.50](#). For simplicity, assume a discrete distribution across 1000 servers, with 109 servers at 0% utilization, 80 servers at 10% utilization, etc. Compute the total performance and total energy for this workload mix.
- 6.30 [15/20/20] < 6.7 > This exercise illustrates the interactions of energy proportionality models with optimizations such as server consolidation and energy-efficient server designs. Consider the scenarios shown in [Figures 6.51 and 6.52](#).
- a. [15] Consider two servers with the power distributions shown in [Figure 6.51](#): case A (the server considered in [Figure 6.50](#)) and case B (a less energy-proportional but more energy-efficient server than case A). Assume the system utilization mix in column 7 of [Figure 6.50](#). For simplicity, assume a discrete distribution across 1000 servers, with 109 servers at 0% utilization, 80 servers at 10% utilization, etc., as shown in row 1 of [Figure 6.52](#). Assume performance variation based on column 2 of [Figure 6.50](#). Compare the total performance and total energy for this workload mix for the two server types.

Activity (%)	0	10	20	30	40	50	60	70	80	90	100
Power, case A (W)	181	308	351	382	416	451	490	533	576	617	662
Power, case B (W)	250	275	325	340	395	405	415	425	440	445	450

Figure 6.51 Power distribution for two servers.

- b. [20] Consider a cluster of 1000 servers with data similar to the data shown in [Figure 6.50](#) (and summarized in the first rows of [Figures 6.51 and 6.52](#)). What are the total performance and total energy for the workload mix with these assumptions? Now assume that we were able to consolidate the workloads to model the distribution shown in case C (second row of [Figure 6.52](#)). What are the total performance and total energy now? How does the total energy compare with a system that has a linear energy-proportional model with idle power of zero watts and peak power of 662 W?
- c. [20] Repeat part (b), but with the power model of server B, and compare with the results of part (a).
- 6.31 [10/Discussion] < 6.7 > *System-Level Energy Proportionality Trends*: Consider the following breakdowns of the power consumption of a server:
- CPU, 50%; memory, 23%; disks, 11%; networking/other, 16%
 - CPU, 33%; memory, 30%; disks, 10%; networking/other, 27%
- a. [10] Assume a dynamic power range of $3.0 \times$ for the CPU (i.e., the power consumption of the CPU at idle is one-third that of its power consumption at peak). Assume that the dynamic range of the memory systems, disks, and the networking/other categories above are, respectively, $2.0 \times$, $1.3 \times$, and $1.2 \times$. What is the overall dynamic range for the total system for the two cases?
- b. [Discussion] What can you learn from the results of part (a)? How would we achieve better energy proportionality at the system level? (*Hint*: Energy proportionality at a system level cannot be achieved through CPU optimizations alone but instead requires improvement across all components.)
- 6.32 [30] < 6.7 > Pitt Turner IV et al., [2008] presented a good overview of data center tier classifications. Tier classifications define site infrastructure performance. For simplicity, consider the key differences as shown in [Figure 6.53](#) (adapted from Pitt Turner IV et al., [2008]). Using the TCO model in the case study as a guiding framework, compare the cost implications of the different tiers shown.
- 6.33 [Discussion] < 6.7 > Based on the observations in [Figures 6.36 and 6.32](#), what can you say qualitatively about the trade-offs between revenue loss from downtime and costs incurred for uptime?

Activity (%)	0	10	20	30	40	50	60	70	80	90	100
No. servers, case A and B	109	80	153	246	191	115	51	21	15	12	8
No. servers, case C	504	6	8	11	26	57	95	123	76	40	54

Figure 6.52 Utilization distributions across cluster, without and with consolidation.

Tier 1	Single path for power and cooling distributions, without redundant components	99.0%
Tier 2	$(N + 1)$ redundancy = two power and cooling distribution paths	99.7%
Tier 3	$(N + 2)$ redundancy = three power and cooling distribution paths for uptime even during maintenance	99.98%
Tier 4	Two active power and cooling distribution paths, with redundant components in each path, to tolerate any single equipment failure without impacting the load	99.995%

Figure 6.53 Overview of data center tier classifications. Adapted from [Pitt Turner IV et al., 2008].

- 6.34 [15/Discussion] < 6.6 > Some recent studies have defined a metric called TPUE, which stands for “true PUE” or “total PUE.” TPUE is defined as $PUE * SPUE$. PUE, the power utilization effectiveness, is defined in [Section 6.6](#) as the ratio of the total facility power over the total IT equipment power. SPUE, or server PUE, is a new metric analogous to PUE but instead applied to computing equipment. SPUE is defined as the ratio of total server input power to its useful power, where useful power is defined as the power consumed by the electronic components directly involved in the computation: motherboard, disks, CPUs, DRAM, I/O cards, and so on. In other words, the SPUE metric captures inefficiencies associated with the power supplies, voltage regulators, and fans housed on a server.
- [15] Consider a design that uses a higher supply temperature for the computer room air conditioning (CRAC) units. The efficiency of the CRAC unit is approximately a quadratic function of the temperature, and this design therefore improves the overall PUE, let’s assume, by 7%. (Assume baseline PUE of 1.7.) However, the higher temperature at the server level triggers the on-board fan controller to operate the fan at much higher speeds. The fan power is a cubic function of speed, and the increased fan speed leads to a degradation of SPUE. Assume a fan power model: where ns is the normalized fan speed = fan speed in rpm/18,000 and a baseline server power of 350 W. Compute the SPUE if the fan speed increases from (1) 10,000–12,500 rpm and (2) 10,000–18,000 rpm. Compare the PUE and TPUE in both these cases. (For simplicity, ignore the inefficiencies with power delivery in the SPUE model.)
 - [Discussion] Part (a) illustrates that, while PUE is an excellent metric to capture the overhead of the facility, it does not capture the inefficiencies within the IT equipment itself. Can you identify another design where changes to the TPUE are potentially lower than the changes to traditional PUE? (*Hint*: See Exercise 6.26.)
- 6.35 [Discussion/30/Discussion] < 6.7 > Two benchmarks provide a good starting point for energy-efficiency accounting in servers: the SPECpower_ssj2008

benchmark (available at http://www.spec.org/power_ssj2008/) and the JouleSort metric (available at <http://sortbenchmark.org/>).

- a. [Discussion] Look up the descriptions of the two benchmarks. How are they similar? How are they different? What would you do to improve these benchmarks to better address the goal of improving WSC energy efficiency?
 - b. [30] JouleSort measures the total system energy to perform an out-of-core sort and attempts to derive a metric that enables the comparison of systems ranging from embedded devices to supercomputers. Look up the description of the JouleSort metric at <http://sortbenchmark.org>. Download a publicly available version of the sort algorithm and run it on different classes of machines—a laptop, a PC, a mobile phone, etc.—or with different configurations. What can you learn from the JouleSort ratings for different setups?
 - c. [Discussion] Consider the system with the best JouleSort rating from your experiments above. How would you improve the energy efficiency? For example, try rewriting the sort code to improve the JouleSort rating. What does running sort in the cloud do to energy efficiency?
- 6.36 [10/10/15/Discussion] < 6.1, 6.4 > [Figure 6.1](#) is a listing of outages in an array of servers. When dealing with the large scale of WSCs, it is important to balance cluster design and software architectures to achieve the required uptime without incurring significant costs. This question explores the implications of achieving availability through hardware only.
- a. [10] Assuming that an operator wishes to achieve 95% availability through server hardware improvements alone, how many events of each type would have to be reduced? For now, assume that individual server crashes are completely handled through redundant machines.
 - b. [10] How does the answer to part (a) change if the individual server crashes are handled by redundancy 50% of the time? 20% of the time? None of the time?
 - c. [15] Discuss the importance of software redundancy to achieving a high level of availability. If a WSC operator considered buying machines that were cheaper but 10% less reliable, what implications would this have on the software architecture? What are the challenges associated with software redundancy?
 - d. [Discussion] Discuss the importance of eventual consistency in how warehouse-scale computers can scale.
- 6.37 [15] < 6.1, 6.9 > Look up the current prices of standard DDR4 DRAM versus DDR4 DRAM that has error-correcting code (ECC). What is the increase in price per bit for achieving the higher reliability that ECC provides? Using the DRAM prices alone, and the data provided in Section 6.9, what is the uptime per dollar of a WSC with non-ECC versus ECC DRAM?

- 6.38 [5/Discussion/Discussion] < 6.1, 6.9 > *WSC Reliability and Manageability Concerns*:
- [5] Consider a cluster of servers costing \$2000 each. Assuming an annual failure rate of 5%, an average of an hour of service time per repair, and replacement parts requiring 10% of the system cost per failure, what is the annual maintenance cost per server? Assume an hourly rate of \$100 per hour for a service technician.
 - [Discussion] Comment on the differences between this manageability model versus that in a traditional enterprise data center with a large number of small- or medium-sized applications, each running on its own dedicated hardware infrastructure.
 - [Discussion] Discuss the trade-offs in having heterogeneous machines in a warehouse-scale computer.
- 6.39 < 6.1, 6.7 > [5/5/10/Discussion] Imagine you have created a web service that runs very well (responds within 100 ms latency) 99% of the time and has performance issues 1% of the time (maybe the CPU went into a lower power state and the response took 1000 ms, etc.).
- [5] Your service grows popular, and you now have 100 servers and your computation has to touch all these servers to handle the user request. What is the percentage of time your query is likely to have a slow response time, across 100 servers?
 - [5] Instead of “two nines” (99%) single server latency SLA, how many “nines” do we need to have for the single server latency SLA so that the cluster latency SLA has bad latencies only 10% of the time or lower?
 - [10] How do the answers to parts (a) and (b) change if we have 2000 servers?
 - [Discussion] [Section 6.7](#) (page 531) discusses “tail-tolerant” designs. What kind of design optimizations would you need to make in your web service (*Hint*: Look at the “Tail at Scale” paper from Dean and Barroso [2013])?
- 6.40 <6.3, 6.2> [Discussion/Discussion/5/5/5] Modern virtualization platforms/hypervisors support live migration, the ability to move an entire virtual machine between physical hosts. Live migration provides numerous benefits to the cloud provider and customers, including hitless system maintenance and improvements to TCO enabled by more efficient packing of workloads onto physical hosts. Suppose we have two machines, A and B, each with 12 cores and 256 GB of DRAM. Initially, machine A is running two VMs, VM0 using 6 cores and 24 GB of DRAM and VM1 using 4 cores and 48 GB of DRAM. Machine B is running a single VM, VM2 consuming 4 cores and 128 GB of DRAM. Suppose VM1 is shut down by the user. Based on previous experience with this customer’s VMs, we expect VM0 and VM2 to continue running for the foreseeable future. Suppose that we have a simple VM migration system that 1) pauses the VM, 2) transfers CPU/memory state over the network, and 3) resumes the VM. Assume that migration time will be dominated by transferring memory contents.

- a. [Discussion] When migrating VMs, what resources do we need to account for beyond CPU/memory state? How would you expect these resources to be managed if the goal is to minimize migration time?
 - b. [Discussion] Suppose we want to consolidate VM0 and VM2 onto a single physical host. What metrics might we look at to decide which VM to migrate?
 - c. [5] Assume our simple migration system and suppose that we have guaranteed six nines of uptime to our VM customers. If we will only migrate a VM at most once per day and this is the only likely source of downtime, how long can we spend in the migration process?
 - d. [5] Suppose our system has decided that VM2 should be migrated to machine A. What is the minimum required bandwidth between A and B to migrate the VM without violating our uptime guarantee, again assuming one migration per day?
 - e. [5] Given a 200 Gb/s network, how many times could VM0 be migrated per day without violating an uptime guarantee of five nines?
- 6.41 <6.3, 6.2> [Discussion] Containerization is another popular isolation mechanism, in addition to process isolation, virtual machines, physical isolation, and others.
- a. [Discussion] Look up popular container management tools. Is isolation touted as the main use case?
 - b. [Discussion] Compare and contrast containerization against the other isolation mechanisms listed above. Discuss along various axes, including performance overheads, isolation capabilities, and hardware requirements.
 - c. [Discussion] What kinds of WSC operators might rely on each kind of isolation? Provide an example use case for each isolation technique in a WSC.
 - d. [Discussion] Isolation mechanisms are never perfect. Find and explain some examples of these mechanisms being defeated in production systems.
- 6.42 <6.5> [30/10] Today's WSCs offer a plethora of I/O mechanisms and accelerators with various cost/performance profiles. This wide variety is also available to cloud users, with pricing listed on cloud provider websites.
- a. [30] Plot the per-hour/per-GB/per Gb/s/etc. costs for the following resources across different cloud providers. Try to isolate the costs of each by comparing/contrasting instances of various types.
 - i. Networking (varying bandwidth)
 - ii. DRAM (varying capacity and standard)
 - iii. NVMe Flash (varying IOPS, bandwidth)
 - iv. Disk (varying IOPS, bandwidth)
 - v. GPUs (varying OPs/s)
 - vi. FPGAs
 - vii. ML Accelerators (varying OPs/s)

- b. [10] Collect the retail costs of similar devices. If the cloud provider is purchasing the devices at these prices, how long do they have to rent them out to break even?

6.43 < 6.8 > [Discussion/5/5/5/Discussion] FPGAs provide the ability to deploy new specialized hardware designs in hours rather than months or years, at the expense of power and performance when compared to an ASIC. In this question we will consider the tradeoffs of deploying specialized hardware designs on FPGAs vs. ASICs. Suppose you are a cloud operator with the ability to purchase one of the following accelerators to deploy in your fleet: (1) FlexAcc, an FPGA-based accelerator that can be modified and re-programmed quickly to maintain a constant speedup of 16x over software on a 32-core CPU, and (2) FastAcc, an ASIC-based accelerator that cannot be modified but is 1024x faster than software on a 32-core CPU, consumes 1/8th the power of FlexAcc, and consumes 1/32nd the power of a CPU. Suppose that FlexAcc costs the same as FastAcc to purchase.

- a. [Discussion] What are some application domains or libraries that might map better to the FlexAcc form factor? What about the FastAcc form factor?
- b. [5] Suppose that with each new software release, FastAcc becomes half as effective (power consumption is constant). After how many software releases are FlexAcc and FastAcc equal in terms of instantaneous performance per watt?
- c. [5] Suppose a 32-core CPU costs \$2048 and power costs \$0.125/kWh. Assuming that both the CPU and FastAcc will be deployed for 2 years without any new software releases. What is the maximum purchase price of FastAcc to break even against the CPU?
- d. [5] Redo part c, but for FlexAcc.
- e. [Discussion] We have implied in a-d that the ASIC form factor accelerators are wholly unable to evolve with software; however, this is not the case. What are some examples of programmable ASICs? How do they maintain flexibility?

6.44 <6.8> [Discussion/Discussion/240/5/5/5]: Reproducing open-source custom silicon for WSCs. Karandikar et al. released an open-source hardware accelerator design for Protocol Buffers, a serialization framework commonly used for inter-service communication and persistent storage in warehouse-scale computers.

- a. [Discussion] What motivated the design of this accelerator? Find some references that discuss the CPU utilization of serialization/deserialization across different cloud providers and at different points in time.
- b. [Discussion] How should an accelerator like this be integrated into a system (near-core, CXL, PCIe, etc.)? Discuss the tradeoffs from both a performance and integration cost perspective.
- c. [240] Follow the artifact appendix in Karandikar et al., to run the Google fleet-derived protocol buffers benchmarks on the open-source implementation of the protocol buffers accelerator. Access to AWS is required.

- d. [5] This paper not only proposes new hardware for WSCs but also makes use of specialization in WSCs to enable its evaluation by harnessing cloud FPGAs for fast hardware simulation using FireSim. Compute the cost of using the required FPGAs in a pay-as-you-go fashion (on EC2 F1) vs. purchasing a single FPGA (Xilinx Alveo U250) outright.
- e. [5] How many times would you need to run the benchmark to break even if you purchased an FPGA? What if you account for the cost of the server that hosts the FPGA?
- f. [5] Redo the calculations in the previous sections, assuming that you purchased enough FPGAs (eight) to match the parallelism of running on multiple FPGAs in the cloud as encoded in the artifact.

References

- Barroso, L.A., Holzle, U., Ranganathan, P., 2018. *The Datacenter as a Computer: Designing Warehouse-Scale Machines*, third ed. Morgan & Claypool Publishers.
- Bugnion, E., Devine, S., Rosenblum, M., Sugerman, J., Wang, E.Y., November 2012. Bringing Virtualization to the x86 Architecture with the Original Vmware Workstation., 30 ACM Transactions of Computing Systems, p. 4. <https://doi.org/10.1145/2382553.2382554>.
- Cook, B., Khazem, K., Kroening, D., Tasiran, S., Tautschnig, M., Tuttle, M.R., 2021. Model checking boot code from AWS data centers. *Form. Methods Syst. Des.* 57 (1), 34–52. <https://doi.org/10.1007/s10703-020-00344-2>.
- Delimitrou, C., Kozyrakis, C., 2014. Quasar: resource-efficient and QoS-aware cluster management. In: *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems (ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, pp. 127–144. <https://doi.org/10.1145/2541940.2541941>.
- Ferguson, A.D., Gribble, S.D., Hong, C.-Y., Edwin Killian, C., Mohsin, W., Mühe, H., Ong, J., Poutievski, L., Singh, A., Vicisano, L., Alimi, R., Chen, S.S., Conley, M., Mandal, S., Nagaraj, K., Bollineni, K.N., Sabaa, A., Zhang, S., Zhu, M., Amin, V., April 12-14, 2021. Orion: Google's Software-Defined Networking Control Plane. In: *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation*.
- Firestone, D., Putnam, A., Mundkur, S., Chiou, D., Dabagh, A., Andrewartha, M., Angepat, H., Bhanu, V., Caulfield, A., Chung, E., Chandrappa, H.K., Chaturmohta, S., Humphrey, M., Lavier, J., Lam, N., Liu, F., Ovtcharov, K., Padhye, J., Popuri, G., Raindel, S., Sapre, T., Shaw, M., Gabriel, S., Sivakumar, M., Srivastava, N., Verma, A., Zuhair, Q., Deepak, B., Doug, B., Vaid, K., Maltz, D.A., Greenberg, A., 2018. Azure accelerated networking: SmartNICs in the public cloud. In: *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (NSDI'18)*. USENIX Association, USA, pp. 51–64.
- Forecast: Public Cloud Services, Worldwide, 2020-2026, 3Q22 Update.” <https://www.gartner.com/document/4019167>
- Jalili, M., Manousakis, I., Goiri, Í., Misra, P.A., Raniwala, A., Alissa, H., Ramakrishnan, B., Tuma, P., Belady, C., Fontoura, M., Bianchini, R., 2021. Cost-efficient overclocking in immersion-cooled datacenters. In: *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA '21)*. IEEE Press, pp. 623–636. <https://doi.org/10.1109/ISCA52012.2021.00055>.
- Karandikar, S., Leary, C., Kennelly, C., Zhao, J., Parimi, D., Nikolic, B., Asanovic, K., Ranganathan, P., 2021. A Hardware Accelerator for Protocol Buffers. In: *MwICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO '21*). <https://doi.org/10.1145/3466752.3480051>.
- Li, H., Berger, D.S., Novakovic, S., Hsu, L., Ernst, D., Zardoshti, P., Shah, M., Rajadnya, S., Lee, S., Agarwal, I., Hill, M.D., Fontoura, M., Bianchini, R., March 2023. Pond: CXL-Based

- Memory Pooling Systems for Cloud Platforms. In: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). Vancouver, BC Canada.
- Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P., Christos, K., 2016. Improving Resource Efficiency at Scale with Heracles. *ACM Trans. Comput. Syst.* 34 (2), 33. <https://doi.org/10.1145/2882783>. *Article 6*.
- Nishtala, R., Fugal, H., Grimm, S., Kwiatkowski, M., Lee, H., Li, H.C., McElroy, R., Paleczny, M., Peek, D., Paul, S., Stafford, D., Tung, T., Venkataramani, V., 2013. Scaling Memcache at Facebook. In: Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation (nsdi'13). USENIX Association, USA, pp. 385–398.
- Popek, G.J., Goldberg, R.P., July 1974. Formal requirements for virtualizable third generation architectures. *Communications of the ACM* 17 (7), 412–421. <https://doi.org/10.1145/361011.361073>.
- Poutievski, L., Mashayekhi, O., Ong, J., Singh, A., Tariq, M., Wang, R., Zhang, J., Beauregard, V., Conner, P., Gribble, S., Kapoor, R., Kratzer, S., Li, N., Liu, H., Nagaraj, K., Ornstein, J., Sawhney, S., Urata, R., Vicisano, L., Yasumura, K., Zhang, S., Zhou, J., Amin, V., 2022. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In: Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22). Association for Computing Machinery, New York, NY, USA, pp. 66–85. <https://doi.org/10.1145/3544216.3544265>.
- Shalev, L., Ayoub, H., Bshara, N., Sabbag, E., Nov.-Dec. 2020. A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC. *IEEE Micro* vol. 40 (6), 67–73. <https://doi.org/10.1109/MM.2020.3016891>.
- Turner IV, Pitt, W., Seader, J.H k, Brill, K.G., 2008. Tier classifications define site infrastructure performance. White Paper, The Uptime Institute.
- Uptime Institute Global Data Center Survey Results 2022, <https://uptimeinstitute.com/resources/research-and-reports/uptime-institute-global-data-center-survey-results-2022>

This page intentionally left blank

7.1	Introduction	586
7.2	Guidelines for DSAs	588
7.3	Example Domain: Deep Neural Networks	590
7.4	Google's Tensor Processing Unit v4 and v4 lite, Data Center DNN Accelerators	602
7.5	The NVIDIA A100 and T4 GPUs, Graphics, and DNN Accelerators for the Data Center	609
7.6	Graphcore IPU Bow, a Data Center Accelerator for Training	616
7.7	The Samsung Neural Processing Unit (NPU), a Smartphone Inference Accelerator	618
7.8	Cross-Cutting Issues	621
7.9	Putting It All Together: Comparing DNN Accelerators	625
7.10	Fallacies and Pitfalls	629
7.11	Concluding Remarks	632
7.12	Historical Perspectives and References	633
	Case Studies and Exercises by Cliff Young	633
	References	648