CHAPTER 8

Closing Remarks

Pervasive Internet access, the proliferation of mobile devices, and the rapid growth of cloud computing have all enabled an ever growing number of applications to move to a web services delivery model ("to the cloud"). In this model, the massive amounts of well-connected processing and storage resources in large datacenters can be efficiently amortized across a large user population and multiple ubiquitous workloads. These datacenters are quite different from traditional colocation or hosting facilities of earlier times, constituting a new class of large-scale computers. The software in these computers is built from several individual programs that interact together to implement complex Internet services, and may be designed and maintained by different teams of engineers, perhaps even across organizational and company boundaries. The data volume manipulated by such computers can range from hundreds to thousands of terabytes, with service-level requirements for high availability, high throughput, and low latency often requiring replication of the baseline data set. Applications of this scale do not run on a single server or even on a rack of servers. They require clusters of many hundreds or thousands of individual servers, with their corresponding storage and networking subsystems, power distribution and conditioning equipment, and cooling infrastructure.

Our central point is simple: this computing platform cannot be viewed simply as a miscellaneous collection of co-located machines. Large portions of the hardware and software resources in these datacenters must work in concert to deliver good levels of Internet service performance, something that can only be achieved by a holistic approach to their design and deployment. In other words, we must treat the datacenter itself as one massive computer. The enclosure for this computer bears little resemblance to a pizza box or a refrigerator, the images chosen to describe servers in past decades. Instead it looks more like a building or warehouse—computer architecture meets traditional (building) architecture—a *warehouse-scale computer* (*WSC*).

Hardware and software architects need to understand the characteristics of this class of computing systems so that they can continue to design and program today's WSCs. WSCs are built from a relatively homogeneous collection of components (servers, storage, and networks) and use a common software management and scheduling infrastructure across all computing nodes to orchestrate resource usage among multiple workloads. In the remainder of this chapter, we summarize the main characteristics of WSC systems described in previous sections and list some important challenges and trends.

L. A. Barroso et al., *The Datacenter as a Computer*

8.1 HARDWARE

The building blocks of choice for WSCs are commodity server-class machines, consumer- or enterprise-grade storage devices, and Ethernet-based networking fabrics. Driven by the purchasing volume of hundreds of millions of consumers and small businesses, commodity components benefit from manufacturing economies of scale and therefore present significantly better price/performance ratios than their corresponding high-end counterparts. In addition, Internet and cloud applications tend to exhibit large amounts of easily exploitable parallelism, making the peak performance of an individual server less important than the aggregate throughput of a collection of servers.

The higher reliability of high-end equipment is less important in this domain because a fault-tolerant software layer is required to provision a dependable Internet service regardless of hardware quality. Even if we used highly reliable servers, clusters with tens of thousands of systems will experience failures too frequently for software to assume fault-free operation. Moreover, large and complex Internet services are often composed of multiple software modules or layers that are not bug-free and can fail at even higher rates than hardware components.

Given the baseline reliability of WSC components and the large number of servers used by a typical workload, there are likely no useful intervals of fault-free operation: we must assume that the system is operating in a state of near-continuous recovery. This state is especially challenging for online services that need to remain available every minute of every day. For example, it is impossible to use the recovery model common to many HPC clusters, which pause an entire cluster workload upon an individual node failure and restart the whole computation from an earlier checkpoint. Consequently, WSC applications must work around failed servers in software, either at the application level or (preferably) via functionality provided via middleware, such as a provisioning system for virtual machines that restarts a failed VM on spare nodes. Despite the attractiveness of low-end, moderately reliable server building blocks for WSCs, high-performance, high-availability components still have value in this class of systems. For example, fractions of a workload (such as SQL databases) may benefit from higher-end SMP servers with their larger interconnect bandwidth. However, highly parallel workloads and fault-tolerant software infrastructures effectively broaden the space of building blocks available to WSC designers, allowing lower end options to work very well for many applications.

The performance of the networking fabric and the storage subsystem can be more relevant to WSC programmers than CPU and DRAM subsystems, unlike what is more typical in smaller scale systems. The relatively high costs (per gigabyte) of DRAM or FLASH storage make them prohibitively expensive for large data sets or infrequently accessed data; therefore, disk drives are still used heavily. The increasing gap in performance between DRAM and disks, and the growing imbalance between throughput and capacity of modern disk drives makes the storage subsystem a common performance bottleneck in large-scale systems, motivating broader use of Flash, and potentially

new non-volatile memory technologies like ZNAND [Sam17] or 3D Xpoint [3DX]. The use of many small-scale servers demands networking fabrics with very high port counts and high bisection bandwidth. Because such fabrics are costly today, programmers must be keenly aware of the scarcity of datacenter-level bandwidth when architecting software systems. This results in more complex software solutions, expanded design cycles, and sometimes inefficient use of global resources.

8.2 SOFTWARE

WSCs are more complex programming targets than traditional computing systems because of their immense scale, complexity of their architecture (as seen by the programmer), and the need to tolerate frequent failures.

Internet services must achieve high availability, typically aiming for a target of 99.99% or better (about an hour of downtime per year). As mentioned earlier, achieving fault-free operation on a large collection of hardware and system software is infeasible, therefore warehouse-scale workloads must be designed to gracefully tolerate high numbers of component failures with little or no impact on service-level performance and availability.

This workload differs substantially from that running in traditional HPC datacenters, the traditional users of large-scale cluster computing. Like HPC applications, these workloads require significant CPU resources, but the individual tasks are less synchronized than in typical HPC applications and communicate less intensely. Furthermore, they are much more diverse, unlike HPC applications that exclusively run a single binary on a large number of nodes. Much of the parallelism inherent in this workload is natural and easy to exploit, stemming from the many users concurrently accessing the service or from the parallelism inherent in data mining. Utilization varies, often with a diurnal cycle, and rarely reaches 90% because operators prefer to keep reserve capacity for unexpected load spikes (flash crowds) or to take on the load of a failed cluster elsewhere in the world. In comparison, an HPC application may run at full CPU utilization for days or weeks.

Software development for Internet services also differs from the traditional client/server model in a number of ways. First, typical Internet services exhibit ample parallelism stemming from both data parallelism and request-level parallelism. Typically, the problem is not to find parallelism but to manage and efficiently harness the explicit parallelism that is inherent in the application. Second, WSC software exhibit significant workload churn. Users of Internet services are isolated from the service's implementation details by relatively well-defined and stable high-level APIs (e.g., simple URLs), making it much easier to deploy new software quickly. For example, key pieces of Google's services have release cycles on the order of days, compared to months or years for desktop software products. The datacenter is also a more homogeneous environment than the desktop. Large Internet services operations typically deploy a small number of hardware and system software configurations at any given point in time. Any heterogeneity arises primarily from the incentives to

deploy more cost-efficient components that become available over time. Finally, although it may be reasonable for desktop-class software to assume a fault-free hardware operation for months or years, this is not true for datacenter-level services; Internet services must work in an environment where faults are part of daily life. Ideally, the cluster-level system software should provide a layer that hides most of that complexity from application-level software, although that goal may be difficult to accomplish for all types of applications.

The complexity of the raw WSC hardware as a programming platform can lower programming productivity because every new software product must efficiently handle data distribution, fault detection and recovery, and work around performance discontinuities (such as the DRAM/disk gap and networking fabric topology issues mentioned earlier). Therefore, it is essential to produce software infrastructure modules that hide such complexity and can be reused across a large segment of workloads. Google's MapReduce, GFS, BigTable, and Chubby are examples of the kind of software that enables the efficient use of WSCs as a programming platform. With the introduction of accelerators in the fleet, similar software modules, such as Tensorflow, are needed to hide complexity there as well.

8.3 ECONOMICS AND ENERGY EFFICIENCY

The economics of Internet services demands very cost efficient computing systems, rendering it the primary metric in the design of WSC systems. Cost efficiency must be defined broadly to account for all the significant components of cost including facility capital and operational expenses (which include power provisioning and energy costs), hardware, software, management personnel, and repairs.

Power- and energy-related costs are particularly important for WSCs because of their size. In addition, fixed engineering costs can be amortized over large deployments, and a high degree of automation can lower the cost of managing these systems. As a result, the cost of the WSC "enclosure" itself (the datacenter facility, the power, and cooling infrastructure) can be a large component of its total cost, making it paramount to maximize energy efficiency and facility utilization. For example, intelligent power provisioning strategies such as peak power oversubscription may allow more computing to be deployed in a building.

The utilization characteristics of WSCs, which spend little time fully idle or at very high load levels, require systems and components to be energy efficient across a wide load spectrum, and particularly at low utilization levels. The energy efficiency of servers and WSCs is often overestimated using benchmarks that assume operation peak performance levels. Machines, power conversion systems, and the cooling infrastructure often are much less efficient at the lower activity levels, for example, at 30% of peak utilization, that are typical of production systems. We suggest that *energy proportionality* be added as a design goal for computing components. Ideally, energy-proportional

systems will consume nearly no power when idle (particularly while in active idle states) and gradually consume more power as the activity level increases.

Energy-proportional components could substantially improve energy efficiency of WSCs without impacting the performance, availability, or complexity. Since the publication of the first version of this book, CPUs have improved their energy proportionality significantly while the remaining WSC components have witnessed more modest improvements.

In addition, traditional datacenters themselves are not particularly efficient. A building's power utilization efficiency (PUE) is the ratio of total power consumed divided by useful (server) power; for example, a datacenter with a PUE of 2.0 uses an additional 1 W of power for every watt of server power. Unfortunately, many legacy datacenter facilities run at PUEs of 2 or greater, and PUEs of 1.5 are rare. Clearly, significant opportunities for efficiency improvements exist not just at the server level but also at the building level, as was demonstrated by Google's annualized 1.11 PUE across all its custom-built facilities as of late 2018 [GDCa].

Energy efficiency optimizations naturally produce lower electricity costs. However, power provisioning costs, that is, the cost of building a facility capable of providing and cooling a given level of power, can be even more significant than the electricity costs themselves—in Chapter 6 we showed that datacenter-related costs can constitute more than half of total IT costs in some deployment scenarios. Maximizing the usage of a facility's peak power capacity while simultaneously reducing the risk of exceeding it is a difficult problem but a very important part of managing the costs of any large-scale deployment.

8.4 BUILDING RESPONSIVE LARGE-SCALE SYSTEMS

8.4.1 CONTINUALLY EVOLVING WORKLOADS

In spite of their widespread adoption, in many respects, Internet services are still in their infancy as an application area. New products appear and gain popularity at a very fast pace with some of the services having very different architectural needs than their predecessors. For example, at the time of the first edition of this book, web search was the poster child for internet services. Video sharing on YouTube exploded in popularity in a period of a few months and the needs of such an application were distinct from earlier Web services such as email or search.

More recently, machine learning has exploded in its application across a wide variety of workloads and use-cases. While machine learning is pervasive in multiple web services, some notable recent examples include: more meaning extraction in transitioning from web search to knowledge graphs, automatic image recognition and classification in photo and video sharing applications, and smart reply and automatic composition features in gmail. Beyond web services, machine learning is also transforming entire industries from health care to manufacturing to self-driving cars. Once

again, this has led to a fundamental change in the computation needs for the WSCs that power these workloads. Now with adoption of Cloud Computing, as discussed in Chapter 2, we are in the early stages of yet another evolution in workloads. A particularly challenging consideration is that many parts of WSC designs include components (building, power, cooling) that are expected to last more than a decade to leverage the construction investment. The mismatch between the time scale for radical workload behavior changes and the design and life cycles for WSCs requires creative solutions from both hardware and software systems.

8.4.2 AMDAHL'S CRUEL LAW

Semiconductor trends suggest that future performance gains will continue to be delivered mostly by providing more cores or threads, and not so much by faster CPUs. That means that large-scale systems must continue to extract higher parallel efficiency (or speed-up) to handle larger, more interesting computational problems. This is a challenge today for desktop systems but perhaps not as much for WSCs, given the arguments we have made earlier about the abundance of thread-level parallelism in its universe of workloads. Having said that, even highly parallel systems abide by Amdahl's law, and there may be a point where Amdahl's effects become dominant even in this domain, limiting performance scalability through just parallelism. This point could come earlier; for example, if high-bandwidth, high-port count networking technology continues to be extremely costly with respect to other WSC components.

8.4.3 THE ATTACK OF THE KILLER MICROSECONDS

As introduced by Barroso et al [Bar+17], the "*killer microsecond problem*" arises due to a new breed of low-latency IO devices ranging from datacenter networking to accelerators, to emerging non-volatile memories. These IO devices have latencies on the order of microseconds rather than milliseconds. Existing system optimizations, however, are typically targeted at the nanosecond scale (at the computer architecture level) or millisecond scale (operating systems). Today's hardware and system software make an inadequate platform for microsecond-scale IO, particularly given the tension between the support for synchronous programming models for software productivity, and performance. New microsecond-optimized systems stacks, across hardware and software, are therefore needed. Such optimized designs at the microsecond scale, and corresponding faster IO, can in turn enable a virtuous cycle of new applications that leverage low latency communication, dramatically increasing the effective computing capabilities of WSCs.

8.4.4 TAIL AT SCALE

Similar to how systems need to be designed for fault tolerance, a new design constraint unique to WSCs is the design for *tail tolerance* [DB13]. This addresses the challenges in performance when even infrequent high-latency events (unimportant in moderate-size systems) can come to dominate

overall service performance at the WSC level. As discussed in Chapter 2, for a system with a typical latency of 10 ms, but a 99th percentile of one second, the number of user requests that take more than one second goes from 1% to 63% when scaling from one machine to a cluster of 100 machines! Large online services need to be designed create a predictable and responsive (low latency) whole of out less predictable parts. Some broad principles that have been used in recent WSCs include prioritizing interactive requests, breaking tasks into finer-granularity units that can be interleaved to reduce head-of-line blocking, and managing background and one-off events carefully. Some specific software techniques used in Google systems are discussed in more detail in [DB13] including the use of canary requests and replicated hedge or speculative requests, and putting slow machines on "probation." Recent work on QoS management in WSCs discuss hardware support to improve tail tolerance as well [Mar+11, DK14, Lo+15]. The need for such class of techniques will only continue to be greater as the scale and complexity of WSCs increase.

8.5 LOOKING AHEAD

We are still learning how best to design and use this new class of machines, as they are still relatively nascent (~15 years) compared to traditional systems. Below, we identify some key challenges and opportunities in this space, based on our experience designing and using WSCs.

8.5.1 THE ENDING OF MOORE'S LAW

Overall, the broader computer architecture community faces an important and exciting challenge. As our thirst for computing performance increases, we must continue to find ways to ensure that performance improvements are accompanied by corresponding improvements in energy efficiency and cost efficiency. The former has been achieved in the past due to Dennard Scaling [Den+74]: every 30% reduction in transistor linear dimensions results in twice as many transistors per area and 40% faster circuits, but with a corresponding reduction to supply voltage at the same rate as transistor scaling. Unfortunately, this has become extremely difficult as dimensions approach atomic scales. It is now widely acknowledged that Dennard scaling has stopped in the past decade. This means that any significant improvements in energy efficiency in the foreseeable future are likely to come from architectural techniques instead of fundamental technology scaling.

More recently, we are also seeing trends that classic Moore's law—improvements in cost-efficiency—is also slowing down due to a number of factors spanning both economic considerations (e.g., fabrication costs in the order of billions of dollars) and fundamental physical limits (limits of CMOS scaling). This is a more fundamental disruption to the industry. The challenges are particularly exacerbated by our earlier discussion on evolving workloads and growing demand, with deeper analysis over ever growing volumes of data, new diverse workloads in the cloud, and smarter edge devices. Again, this means that continued improvements in computing performance need to

come from architectural optimizations, for area and resource efficiency, but also around more hardware-software codesign and fundamental new architectural paradigms.

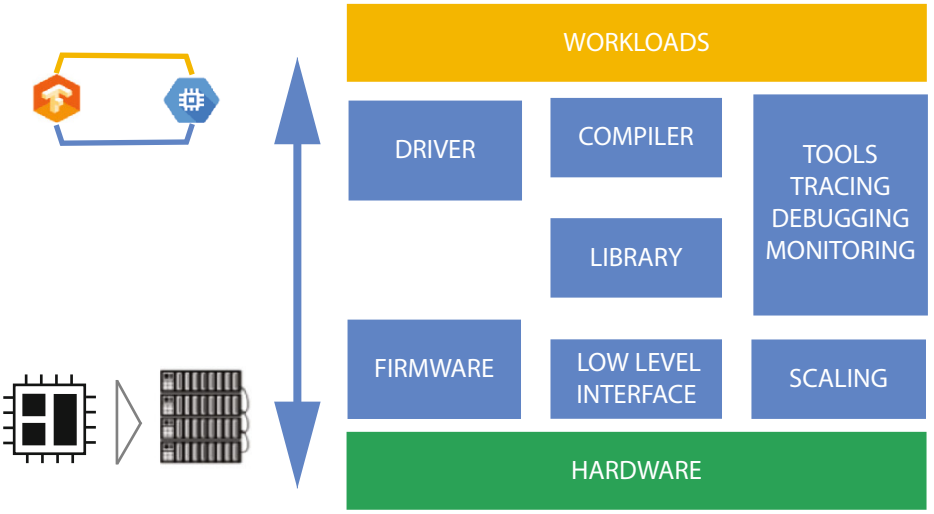

Figure 8.1: Accelerator ecosystems are important.

8.5.2 ACCELERATORS AND FULL SYSTEM DESIGNS

Accelerators are by far the most promising approach to addressing the end of Moore's Law. By tailoring the architecture to the application, we can achieve both improved power and area efficiency. The tradeoff often is that we sacrifice generality and flexibility to increase efficiency for specific types of workloads. In many respects, GPUs were the first and most successful example of the success of this approach, followed by other accelerators discussed in Chapter 3 such as FPGAs deployed by Microsoft and others, and full ASIC solutions such as Google's TPUs.

Accelerators also present a great example of how WSC design decisions need to evolve with changing workload requirements. In the first edition of this book, we argued *against* specialized computing pointing out that the promise of greater efficiency was not worth the tradeoffs of restricting the number of workloads that could benefit from them. However, since then, two key trends have changed that thinking. First, the slowing of Moore's law has made accelerators more appealing compared to general-purpose systems, but second, perhaps more importantly, deep learning models took off in a big way enabling specialized hardware in this space to power a broad spectrum of new machine learning solutions. As a result, earlier in this decade Google began to more broadly deploy GPUs but also initiated a program to build more specialized accelerators.

Accelerators in WSCs are still nascent, but our experience over the past few years have identified some key opportunities. First, hardware is just the proverbial tip of the iceberg (Figure 8.1).

Accelerator design needs to take a holistic view, across hardware and software for sure, but also across large-scale distributed system deployment. Our discussions earlier about the implications of designing at scale for general-purpose systems apply equally to the design of accelerators as well, illustrated by the design of pods of TPUs in Chapter 3. Additionally, similar to our discussions of the software stack in Chapter 2 for traditional systems, it is important to consider the full system stack for accelerators. This includes thinking about the design of accelerators in the context of the supporting interfaces and compilers, but also considering the tool and ecosystem support for tracing, debugging, monitoring, etc. A principled approach to the hardware-software codesign of accelerators is needed, one that carefully addresses problem decomposition, complexity encapsulation, modularity and interfaces, technical debt, and performance. A lot can be learned from our rich experience in software design [Ous18].

Beyond customization for specific workload classes like deep learning, search and video serving, there are also significant opportunities for the "long tail" of workloads. It is notable that nearly one out three compute cycles at Google is attributable to a handful of "datacenter tax" functions that cross-cut all applications. Similarly, the "killer microsecond" opportunities discussed earlier motivate new hardware innovations as well [Bar+17]. Storage, networking, and security are other broad areas where there are other opportunities for acceleration.

8.5.3 SOFTWARE-DEFINED INFRASTRUCTURE

Another promising approach for future WSCs is to embrace software-defined infrastructure. As discussed in prior chapters, WSCs have a lot of, often competing, constraints: how do we design at scale, at low costs, with ease of manageability and deployment, while also achieving high performance and reliability and efficiency? A software-defined infrastructure embraces a modular approach emphasizing efficiency in the design of the individual building blocks and focusing on capability through composability. This allows us to achieve the benefits of volume economics, fungibility, and agility with individual blocks, while allowing specialization, customization, and new capabilities at the broader composition layer.

SDN [Kre+14] is a good example where separating the network control plane (the policy engine) from the forwarding planes allows the the underlying infrastructure to be abstracted, while enabling the control to become more programmable and flexible. SDNs have been widely adopted in the the design of WSCs and in the broader industry [Jai+13, Kol14, Vah17]. We have an opportunity to consider similar software-defined approaches for the broader WSC design. Many ideas discussed in prior chapters including work around QoS management at the platform level [DK14, Lo+15, Mar+11], power management [Rag+08, Wu+16a], automatic memory and storage tier management [RC12], and more broadly disaggregated architectures [Lim+09, Gao+16] all play well into the theme of software-defined infrastructure.

It is also interesting to note that a software-defined infrastructure is the first step to enabling the greater adoption of ML-based automation in WSC designs. Early work in this space such as the use of ML-based models for prefetching [Has+18] or ML-based approaches to power management [EG16] are very promising, and we expect a lot more innovation in this space.

8.5.4 A NEW ERA OF COMPUTER ARCHITECTURE AND WSCS

John Hennessy and David Patterson, titled their 2018 Turing award lecture "A Golden Age for Computer Architecture." We think there is a similar golden age for WSCs coming as well.

Beyond accelerators and software-defined infrastructure, there are a few other exciting trends worth noting. Data is growing much faster than compute. We recently noted [Lot+18] that the pace of bytes of data uploaded to Youtube is exponentially diverging from the pace of traditional compute processing growth (an order of magnitude over the past decade). While there has been significant innovation in the WSC software stack around data storage and processing (for example, in prior chapters, we discussed GFS, Colossus, MapReduce, TensorFlow, etc.), such innovation has been relatively agnostic to the underlying hardware. Emerging new memory technologies such as Intel's 3D-Xpoint [3DX] or Samsung ZNAND [Sam17] present some fundamental technology disruptions in the memory/storage hierarchy. Co-designing new WSC hardware architectures for emerging data storage and processing needs will be an important area.

WSCs and cloud-based computation also offer the potential to reduce the environmental footprint of IT. On the server side, better utilization, lower PUE, and faster introduction of new hardware can significantly reduce the overall energy footprint as workloads move from inefficient on-premise deployments to cloud providers. On the client side, mobile devices don't need to store all data locally or process it on the device, leading to much more energy efficient clients as well.

Recent attacks like Spectre and Meltdown that exploit timing to inappropriately access data (discussed in Chapter 2) point to the growing importance of thinking of security as a first-class design constraint, motivating what some have called Architecture 2.0—a rethink of the hardware software interfaces to protect information [Hen+18]. At a system level, more hardware-software codesign for security is needed. Some examples include Google's Titan root of trust chip [Sav+17] or recent discussions around enclaves (e.g., Intel SGX).

One other opportunity for the broader community to counter the slowing of Moore's law is around *faster* hardware development. Moore's law is often formulated as improved performance over cost *over time* but the time variable does not get as much attention. If we can accelerate the cadence of introducing new hardware innovation in to the market, that can potentially offset slower performance increases per generation (sometimes referred to as optimizing the "area-under-the-curve"). How can we release "early and often" in WSC hardware akin to WSC software? Recent work in this area, for example around plug-and-play chiplets, post-silicon debugging, and ASIC clouds [ERI], as well as community efforts such as the RISC-V foundation (www.riscv.org) present

interesting opportunities in this direction. Additional key challenges are around approaches to test and deploy custom hardware *at scale*. How do we "launch and iterate" at scale for hardware? Some preliminary work in this area include WSMeter [Lee+18] and FireSim [Kar+18] but more work is needed.

Looking further out, the growing proliferation of smart devices and increased computation at the edge motivate a rethink of end-to-end system tradeoffs. Much as WSCs changed how we think of datacenter design by drawing the box at defining the warehouse as a computer, thinking about interconnected WSCs and their relationship with computing at the edge and in the network will be important in the next decade.

8.6 CONCLUSIONS

Computation is moving into the cloud, and thus into WSCs. Software and hardware architects must be aware of the end-to-end systems to design good solutions. We are no longer designing individual "pizza boxes," or single-server applications, and we can no longer ignore the physical and economic mechanisms at play in a warehouse full of computers. At one level, WSCs are simple—just a few thousand servers connected via a LAN. In reality, building a cost-efficient massive-scale computing platform that has the necessary reliability and programmability requirements for the next generation of cloud-computing workloads is as difficult and stimulating a challenge as any other in computer systems today. We hope that this book will help computer scientists and practitioners understand and contribute to this exciting area.