The Datacenter as a Computer

An Introduction to the Design of Warehouse-Scale Machines

Synthesis Lectures on Computer Architecture

The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines
Luiz André Barroso and Urs Hölzle
www.morganclaypool.com

ISBN: 9781598295566 paperback

ISBN: 9781598295573 ebook

DOI: 10.2200/S00193ED1V01Y200905CAC006

A Publication in the Morgan & Claypool Publishers series

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE

Lecture #6

Series Editor: Mark D. Hill, University of Wisconsin, Madison

Series ISSN

ISSN 1935-3235 print

ISSN 1935-3243 electronic

The Datacenter as a Computer

An Introduction to the Design of Warehouse-Scale Machines

Luiz André Barroso and Urs Hölzle
Google Inc.

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE # 6

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

As computation continues to move into the cloud, the computing platform of interest no longer resembles a pizza box or a refrigerator, but a warehouse full of computers. These new large datacenters are quite different from traditional hosting facilities of earlier times and cannot be viewed simply as a collection of co-located servers. Large portions of the hardware and software resources in these facilities must work in concert to efficiently 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 warehouse-scale computer (WSC). We describe the architecture of WSCs, the main factors influencing their design, operation, and cost structure, and the characteristics of their software base. We hope it will be useful to architects and programmers of today's WSCs, as well as those of future many-core platforms which may one day implement the equivalent of today's WSCs on a single board.

KEYWORDS

computer organization and design, Internet services, energy efficiency, fault-tolerant computing, cluster computing, data centers, distributed systems, cloud computing.

Acknowledgments

While we draw from our direct involvement in Google's infrastructure design and operation over the past several years, most of what we have learned and now report here is the result of the hard work, the insights, and the creativity of our colleagues at Google. The work of our Platforms Engineering, Hardware Operations, Facilities, Site Reliability and Software Infrastructure teams is most directly related to the topics we cover here, and therefore, we are particularly grateful to them for allowing us to benefit from their experience. Ricardo Bianchini, Fred Chong and Mark Hill provided extremely useful feedback despite being handed a relatively immature early version of the text. Our Google colleagues Jeff Dean and Jimmy Clidaras also provided extensive and particularly useful feedback on earlier drafts. Thanks to the work of Kristin Weissman at Google and Michael Morgan at Morgan & Claypool, we were able to make this lecture available electronically without charge, which was a condition for our accepting this task. We were fortunate that Gerry Kane volunteered his technical writing talent to significantly improve the quality of the text. We would also like to thank Catherine Warner for her proofreading and improvements to the text at various stages. We are very grateful to Mark Hill and Michael Morgan for inviting us to this project, for their relentless encouragement and much needed prodding, and their seemingly endless patience.

This edition of the book benefits from the feedback and corrections submitted by Christian Belady, Jeremy Dion, Robert Hundt, Artur Klauser, David Konerding, Mike Marty, Vijay Rao, Jordi Torres, Amund Tveit, Juan Vargas, Pedro Reviriego Vasallo and Kristin Weissman. We are sincerely thankful for their help.

Note to the Reader

We very much appreciate any thoughts, suggestions, or corrections you might have on our manuscript. We plan to revise the book relatively often and will make sure to acknowledge explicitly any input that can help us improve its usefulness and accuracy. Thanks in advance for taking the time to contribute. Please submit your feedback at http://tinyurl.com/wsc-comments (http://spreadsheets.google.com/viewform?formkey=cmFrdHUxcTJ2SndCV2E4RzdTOTA0QXc6MA).

Contents

1. Introduction 1

1.1 Warehouse-Scale Computers 2
1.2 Emphasis on Cost Efficiency 3
1.3 Not Just a Collection of Servers 4
1.4 One Datacenter vs. Several Datacenters 4
1.5 Why WSCs Might Matter to You 5
1.6 Architectural Overview of WSCs 5
1.6.1 Storage 6
1.6.2 Networking Fabric 7
1.6.3 Storage Hierarchy 8
1.6.4 Quantifying Latency, Bandwidth, and Capacity 8
1.6.5 Power Usage 10
1.6.6 Handling Failures 11

2. Workloads and Software Infrastructure 13

2.1 Datacenter vs. Desktop 13
2.2 Performance and Availability Toolbox 15
2.3 Cluster-Level Infrastructure Software 19
2.3.1 Resource Management 20
2.3.2 Hardware Abstraction and Other Basic Services 20
2.3.3 Deployment and Maintenance 20
2.3.4 Programming Frameworks 21
2.4 Application-Level Software 21
2.4.1 Workload Examples 22
2.4.2 Online: Web Search 22
2.4.3 Offline: Scholar Article Similarity 24
2.5 A Monitoring Infrastructure 26
2.5.1 Service-Level Dashboards 26

2.5.2 Performance Debugging Tools 27
2.5.3 Platform-Level Monitoring 28
2.6 Buy vs. Build 28
2.7 Further Reading 29

3. Hardware Building Blocks 31
3.1 Cost-Efficient Hardware 31
3.1.1 How About Parallel Application Performance? 32
3.1.2 How Low-End Can You Go? 35
3.1.3 Balanced Designs 37

4. Datacenter Basics 39
4.1 Datacenter Tier Classifications 39
4.2 Datacenter Power Systems 40
4.2.1 UPS Systems 41
4.2.2 Power Distribution Units 41
4.3 Datacenter Cooling Systems 42
4.3.1 CRAC Units 42
4.3.2 Free Cooling 43
4.3.3 Air Flow Considerations 44
4.3.4 In-Rack Cooling 44
4.3.5 Container-Based Datacenters 45

5. Energy and Power Efficiency 47
5.1 Datacenter Energy Efficiency 47
5.1.1 Sources of Efficiency Losses in Datacenters 49
5.1.2 Improving the Energy Efficiency of Datacenters 50
5.2 Measuring the Efficiency of Computing 52
5.2.1 Some Useful Benchmarks 52
5.2.2 Load vs. Efficiency 54
5.3 Energy-Proportional Computing 56
5.3.1 Dynamic Power Range of Energy-Proportional Machines 57
5.3.2 Causes of Poor Energy Proportionality 58
5.3.3 How to Improve Energy Proportionality 59
5.4 Relative Effectiveness of Low-Power Modes 60
5.5 The Role of Software in Energy Proportionality 61
5.6 Datacenter Power Provisioning 62

5.6.1 Deployment and Power Management Strategies 62
5.6.2 Advantages of Oversubscribing Facility Power 63
5.7 Trends in Server Energy Usage 65
5.8 Conclusions 66
5.8.1 Further Reading 67

6. Modeling Costs 69
6.1 Capital Costs 69
6.2 Operational Costs 71
6.3 Case Studies 72
6.3.1 Real-World Datacenter Costs 74
6.3.2 Modeling a Partially Filled Datacenter 75

7. Dealing with Failures and Repairs 77
7.1 Implications of Software-Based Fault Tolerance 77
7.2 Categorizing Faults 79
7.2.1 Fault Severity 80
7.2.2 Causes of Service-Level Faults 81
7.3 Machine-Level Failures 83
7.3.1 What Causes Machine Crashes? 86
7.3.2 Predicting Faults 87
7.4 Repairs 88
7.5 Tolerating Faults, Not Hiding Them 89

8. Closing Remarks 91
8.1 Hardware 92
8.2 Software 93
8.3 Economics 94
8.4 Key Challenges 96
8.4.1 Rapidly Changing Workloads 96
8.4.2 Building Balanced Systems from Imbalanced Components 96
8.4.3 Curbing Energy Usage 96
8.4.4 Amdahl's Cruel Law 96
8.5 Conclusions 97

References 99

Author Biographies 107

CHAPTER 1

Introduction

The ARPANET is about to turn forty, and the World Wide Web is approaching its 20th anniversary. Yet the Internet technologies that were largely sparked by these two remarkable milestones continue to transform industries and our culture today and show no signs of slowing down. More recently the emergence of such popular Internet services as Web-based email, search and social networks plus the increased worldwide availability of high-speed connectivity have accelerated a trend toward server-side or "cloud" computing.

Increasingly, computing and storage are moving from PC-like clients to large Internet services. While early Internet services were mostly informational, today many Web applications offer services that previously resided in the client, including email, photo and video storage and office applications. The shift toward server-side computing is driven primarily not only by the need for user experience improvements, such as ease of management (no configuration or backups needed) and ubiquity of access (a browser is all you need), but also by the advantages it offers to vendors. Software as a service allows faster application development because it is simpler for software vendors to make changes and improvements. Instead of updating many millions of clients (with a myriad of peculiar hardware and software configurations), vendors need only coordinate improvements and fixes inside their datacenters and can restrict their hardware deployment to a few well-tested configurations. Moreover, datacenter economics allow many application services to run at a low cost per user. For example, servers may be shared among thousands of active users (and many more inactive ones), resulting in better utilization. Similarly, the computation itself may become cheaper in a shared service (e.g., an email attachment received by multiple users can be stored once rather than many times). Finally, servers and storage in a datacenter can be easier to manage than the desktop or laptop equivalent because they are under control of a single, knowledgeable entity.

Some workloads require so much computing capability that they are a more natural fit for a massive computing infrastructure than for client-side computing. Search services (Web, images, etc.) are a prime example of this class of workloads, but applications such as language translation can also run more effectively on large shared computing installations because of their reliance on massive-scale language models.

The trend toward server-side computing and the exploding popularity of Internet services has created a new class of computing systems that we have named *warehouse-scale computers*, or *WSC*s. The name is meant to call attention to the most distinguishing feature of these machines: the massive scale of their software infrastructure, data repositories, and hardware platform. This perspective is a departure from a view of the computing problem that implicitly assumes a model where one program runs in a single machine. In warehouse-scale computing, the program is an Internet service, which may consist of tens or more individual programs that interact to implement complex end-user services such as email, search, or maps. These programs might be implemented and maintained by different teams of engineers, perhaps even across organizational, geographic, and company boundaries (e.g., as is the case with mashups).

The computing platform required to run such large-scale services bears little resemblance to a pizza-box server or even the refrigerator-sized high-end multiprocessors that reigned in the last decade. The hardware for such a platform consists of thousands of individual computing nodes with their corresponding networking and storage subsystems, power distribution and conditioning equipment, and extensive cooling systems. The enclosure for these systems is in fact a building structure and often indistinguishable from a large warehouse.

1.1 WAREHOUSE-SCALE COMPUTERS

Had scale been the only distinguishing feature of these systems, we might simply refer to them as *datacenters*. Datacenters are buildings where multiple servers and communication gear are co-located because of their common environmental requirements and physical security needs, and for ease of maintenance. In that sense, a WSC could be considered a type of datacenter. Traditional datacenters, however, typically host a large number of relatively small- or medium-sized applications, each running on a dedicated hardware infrastructure that is de-coupled and protected from other systems in the same facility. Those datacenters host hardware and software for multiple organizational units or even different companies. Different computing systems within such a datacenter often have little in common in terms of hardware, software, or maintenance infrastructure, and tend not to communicate with each other at all.

WSCs currently power the services offered by companies such as Google, Amazon, Yahoo, and Microsoft's online services division. They differ significantly from traditional datacenters: they belong to a single organization, use a relatively homogeneous hardware and system software platform, and share a common systems management layer. Often much of the application, middleware, and system software is built in-house compared to the predominance of third-party software running in conventional datacenters. Most importantly, WSCs run a smaller number of very large applications (or Internet services), and the common resource management infrastructure allows significant

deployment flexibility. The requirements of homogeneity, single-organization control, and enhanced focus on cost efficiency motivate designers to take new approaches in constructing and operating these systems.

Internet services must achieve high availability, typically aiming for at least 99.99% uptime (about an hour of downtime per year). Achieving fault-free operation on a large collection of hardware and system software is hard and is made more difficult by the large number of servers involved. Although it might be theoretically possible to prevent hardware failures in a collection of 10,000 servers, it would surely be extremely expensive. Consequently, WSC workloads must be designed to gracefully tolerate large numbers of component faults with little or no impact on service level performance and availability.

1.2 EMPHASIS ON COST EFFICIENCY

Building and operating a large computing platform is expensive, and the quality of a service may depend on the aggregate processing and storage capacity available, further driving costs up and requiring a focus on cost efficiency. For example, in information retrieval systems such as Web search, the growth of computing needs is driven by three main factors.

- Increased service popularity that translates into higher request loads.
- The size of the problem keeps growing—the Web is growing by millions of pages per day, which increases the cost of building and serving a Web index.
- Even if the throughput and data repository could be held constant, the competitive nature of this market continuously drives innovations to improve the quality of results retrieved and the frequency with which the index is updated. Although some quality improvements can be achieved by smarter algorithms alone, most substantial improvements demand additional computing resources for every request. For example, in a search system that also considers synonyms of the search terms in a query, retrieving results is substantially more expensive—either the search needs to retrieve documents that match a more complex query that includes the synonyms or the synonyms of a term need to be replicated in the index data structure for each term.

The relentless demand for more computing capabilities makes cost efficiency a primary metric of interest in the design of WSCs. Cost efficiency must be defined broadly to account for all the significant components of cost, including hosting-facility capital and operational expenses (which include power provisioning and energy costs), hardware, software, management personnel, and repairs.

1.3 NOT JUST A COLLECTION OF SERVERS

Our central point is that the datacenters powering many of today's successful Internet services are no longer simply a miscellaneous collection of machines co-located in a facility and wired up together. The software running on these systems, such as Gmail or Web search services, execute at a scale far beyond a single machine or a single rack: they run on no smaller a unit than clusters of hundreds to thousands of individual servers. Therefore, the machine, the computer, *is* this large cluster or aggregation of servers itself and needs to be considered as a single computing unit.

The technical challenges of designing WSCs are no less worthy of the expertise of computer systems architects than any other class of machines. First, they are a new class of large-scale machines driven by a new and rapidly evolving set of workloads. Their size alone makes them difficult to experiment with or simulate efficiently; therefore, system designers must develop new techniques to guide design decisions. Fault behavior and power and energy considerations have a more significant impact in the design of WSCs, perhaps more so than in other smaller scale computing platforms. Finally, WSCs have an additional layer of complexity beyond systems consisting of individual servers or small groups of server; WSCs introduce a significant new challenge to programmer productivity, a challenge perhaps greater than programming multicore systems. This additional complexity arises indirectly from the larger scale of the application domain and manifests itself as a deeper and less homogeneous storage hierarchy (discussed later in this chapter), higher fault rates (Chapter 7), and possibly higher performance variability (Chapter 2).

The objectives of this book are to introduce readers to this new design space, describe some of the requirements and characteristics of WSCs, highlight some of the important challenges unique to this space, and share some of our experience designing, programming, and operating them within Google. We have been in the fortunate position of being both designers of WSCs, as well as customers and programmers of the platform, which has provided us an unusual opportunity to evaluate design decisions throughout the lifetime of a product. We hope that we will succeed in relaying our enthusiasm for this area as an exciting new target worthy of the attention of the general research and technical communities.

1.4 ONE DATACENTER VS. SEVERAL DATACENTERS

In this book, we define the computer to be architected as a datacenter despite the fact that Internet services may involve multiple datacenters located far apart. Multiple datacenters are sometimes used as complete replicas of the same service, with replication being used mostly for reducing user latency and improving serving throughput (a typical example is a Web search service). In those cases, a given user query tends to be fully processed within one datacenter, and our machine definition seems appropriate.

However, in cases where a user query may involve computation across multiple datacenters, our single-datacenter focus is a less obvious fit. Typical examples are services that deal with nonvolatile user data updates, and therefore, require multiple copies for disaster tolerance reasons. For such computations, a set of datacenters might be the more appropriate system. But we have chosen to think of the multi-datacenter scenario as more analogous to a network of computers. This is in part to limit the scope of this lecture, but is mainly because the huge gap in connectivity quality between intra- and inter-datacenter communications causes programmers to view such systems as separate computational resources. As the software development environment for this class of applications evolves, or if the connectivity gap narrows significantly in the future, we may need to adjust our choice of machine boundaries.

1.5 WHY WSCs MIGHT MATTER TO YOU

As described so far, WSCs might be considered a niche area because their sheer size and cost render them unaffordable by all but a few large Internet companies. Unsurprisingly, we do not believe this to be true. We believe the problems that today's large Internet services face will soon be meaningful to a much larger constituency because many organizations will soon be able to afford similarly sized computers at a much lower cost. Even today, the attractive economics of low-end server class computing platforms puts clusters of hundreds of nodes within the reach of a relatively broad range of corporations and research institutions. When combined with the trends toward large numbers of processor cores on a single die, a single rack of servers may soon have as many or more hardware threads than many of today's datacenters. For example, a rack with 40 servers, each with four 8-core dual-threaded CPUs, would contain more than two thousand hardware threads. Such systems will arguably be affordable to a very large number of organizations within just a few years, while exhibiting some of the scale, architectural organization, and fault behavior of today's WSCs. Therefore, we believe that our experience building these unique systems will be useful in understanding the design issues and programming challenges for those potentially ubiquitous next-generation machines.

1.6 ARCHITECTURAL OVERVIEW OF WSCs

The hardware implementation of a WSC will differ significantly from one installation to the next. Even within a single organization such as Google, systems deployed in different years use different basic elements, reflecting the hardware improvements provided by the industry. However, the architectural organization of these systems has been relatively stable over the last few years. Therefore, it is useful to describe this general architecture at a high level as it sets the background for subsequent discussions.

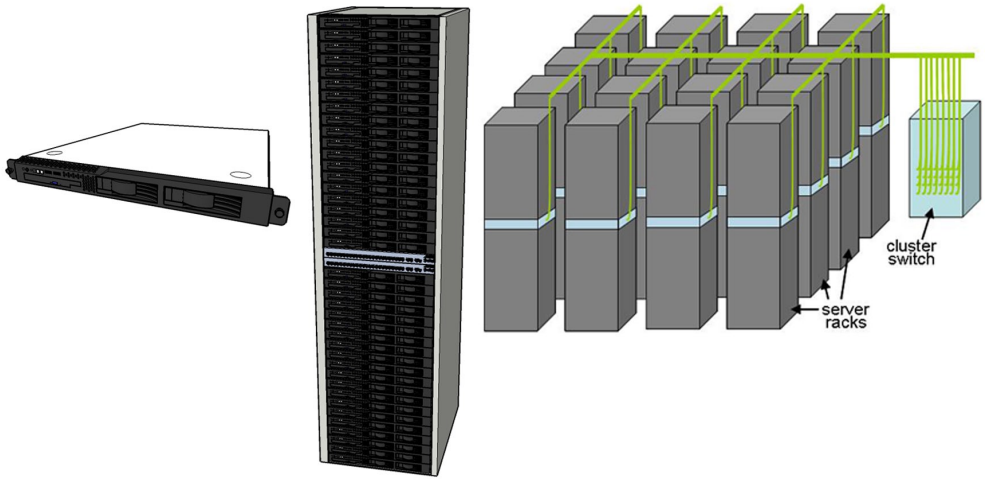

FIGURE 1.1: Typical elements in warehouse-scale systems: 1U server (left), 7′ rack with Ethernet switch (middle), and diagram of a small cluster with a cluster-level Ethernet switch/router (right).

Figure 1.1 depicts some of the more popular building blocks for WSCs. A set of low-end servers, typically in a 1U or blade enclosure format, are mounted within a rack and interconnected using a local Ethernet switch. These rack-level switches, which can use 1- or 10-Gbps links, have a number of uplink connections to one or more cluster-level (or datacenter-level) Ethernet switches. This second-level switching domain can potentially span more than ten thousand individual servers.

1.6.1 Storage

Disk drives are connected directly to each individual server and managed by a global distributed file system (such as Google's GFS [31]) or they can be part of Network Attached Storage (NAS) devices that are directly connected to the cluster-level switching fabric. A NAS tends to be a simpler solution to deploy initially because it pushes the responsibility for data management and integrity to a NAS appliance vendor. In contrast, using the collection of disks directly attached to server nodes requires a fault-tolerant file system at the cluster level. This is difficult to implement but can lower hardware costs (the disks leverage the existing server enclosure) and networking fabric utilization (each server network port is effectively dynamically shared between the computing tasks and the file system). The replication model between these two approaches is also fundamentally different. A NAS provides extra reliability through replication or error correction capabilities within each appliance, whereas systems like GFS implement replication across different machines and consequently

will use more networking bandwidth to complete write operations. However, GFS-like systems are able to keep data available even after the loss of an entire server enclosure or rack and may allow higher aggregate read bandwidth because the same data can be sourced from multiple replicas. Trading off higher write overheads for lower cost, higher availability, and increased read bandwidth was the right solution for many of Google's workloads. An additional advantage of having disks co-located with compute servers is that it enables distributed system software to exploit data locality. For the remainder of this book, we will therefore implicitly assume a model with distributed disks directly connected to all servers.

Some WSCs, including Google's, deploy desktop-class disk drives instead of enterprise-grade disks because of the substantial cost differential between the two. Because that data are nearly always replicated in some distributed fashion (as in GFS), this mitigates the possibly higher fault rates of desktop disks. Moreover, because field reliability of disk drives tends to deviate significantly from the manufacturer's specifications, the reliability edge of enterprise drives is not clearly established. For example, Elerath and Shah [24] point out that several factors can affect disk reliability more substantially than manufacturing process and design.

1.6.2 Networking Fabric

Choosing a networking fabric for WSCs involves a trade-off between speed, scale, and cost. As of this writing, 1-Gbps Ethernet switches with up to 48 ports are essentially a commodity component, costing less than $30/Gbps per server to connect a single rack. As a result, bandwidth within a rack of servers tends to have a homogeneous profile. However, network switches with high port counts, which are needed to tie together WSC clusters, have a much different price structure and are more than ten times more expensive (per 1-Gbps port) than commodity switches. In other words, a switch that has 10 times the bi-section bandwidth costs about 100 times as much. As a result of this cost discontinuity, the networking fabric of WSCs is often organized as the two-level hierarchy depicted in Figure 1.1. Commodity switches in each rack provide a fraction of their bi-section bandwidth for interrack communication through a handful of uplinks to the more costly cluster-level switches. For example, a rack with 40 servers, each with a 1-Gbps port, might have between four and eight 1-Gbps uplinks to the cluster-level switch, corresponding to an oversubscription factor between 5 and 10 for communication across racks. In such a network, programmers must be aware of the relatively scarce cluster-level bandwidth resources and try to exploit rack-level networking locality, complicating software development and possibly impacting resource utilization.

Alternatively, one can remove some of the cluster-level networking bottlenecks by spending more money on the interconnect fabric. For example, Infiniband interconnects typically scale to a few thousand ports but can cost $500–$2,000 per port. Similarly, some networking vendors are starting to provide larger-scale Ethernet fabrics, but again at a cost of at least hundreds of dollars

per server. Alternatively, lower-cost fabrics can be formed from commodity Ethernet switches by building "fat tree" Clos networks [1]. How much to spend on networking vs. spending the equivalent amount on buying more servers or storage is an application-specific question that has no single correct answer. However, for now, we will assume that intra rack connectivity is often cheaper than inter rack connectivity.

1.6.3 Storage Hierarchy

Figure 1.2 shows a programmer's view of storage hierarchy of a typical WSC. A server consists of a number of processor sockets, each with a multicore CPU and its internal cache hierarchy, local shared and coherent DRAM, and a number of directly attached disk drives. The DRAM and disk resources within the rack are accessible through the first-level rack switches (assuming some sort of remote procedure call API to them), and all resources in all racks are accessible via the cluster-level switch.

1.6.4 Quantifying Latency, Bandwidth, and Capacity

Figure 1.3 attempts to quantify the latency, bandwidth, and capacity characteristics of a WSC. For illustration we assume a system with 2,000 servers, each with 8 GB of DRAM and four 1-TB disk drives. Each group of 40 servers is connected through a 1-Gbps link to a rack-level switch that has an additional eight 1-Gbps ports used for connecting the rack to the cluster-level switch (an

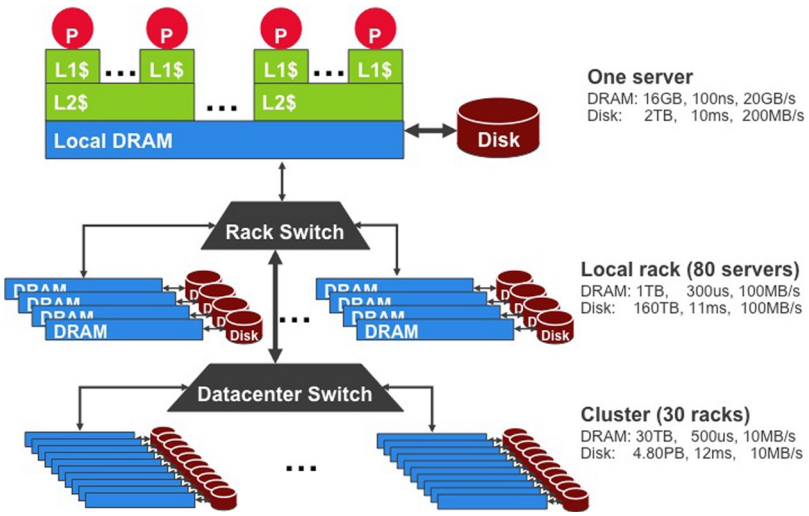

FIGURE 1.2: Storage hierarchy of a WSC.

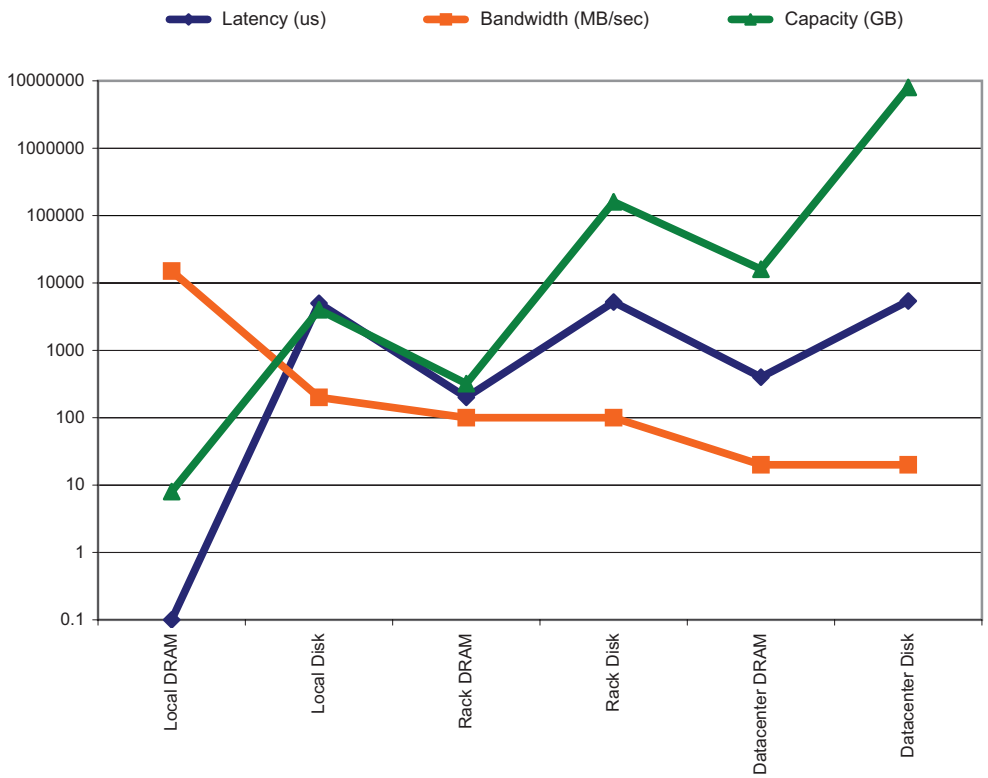

FIGURE 1.3: Latency, bandwidth, and capacity of a WSC.

oversubscription factor of 5). Network latency numbers assume a socket-based TCP-IP transport, and networking bandwidth values assume that each server behind an oversubscribed set of uplinks is using its fair share of the available cluster-level bandwidth. We assume the rack- and cluster-level switches themselves are not internally oversubscribed. For disks, we show typical commodity disk drive (SATA) latencies and transfer rates.

The graph shows the relative latency, bandwidth, and capacity of each resource pool. For example, the bandwidth available from local disks is 200 MB/s, whereas the bandwidth from off-rack disks is just 25 MB/s via the shared rack uplinks. On the other hand, total disk storage in the cluster is almost ten million times larger than local DRAM.

A large application that requires many more servers than can fit on a single rack must deal effectively with these large discrepancies in latency, bandwidth, and capacity. These discrepancies are much larger than those seen on a single machine, making it more difficult to program a WSC.

A key challenge for architects of WSCs is to smooth out these discrepancies in a cost-efficient manner. Conversely, a key challenge for software architects is to build cluster infrastructure and services that hide most of this complexity from application developers.

1.6.5 Power Usage

Energy and power usage are also important concerns in the design of WSCs because, as discussed in more detail in Chapter 5, energy-related costs have become an important component of the total cost of ownership of this class of systems. Figure 1.4 provides some insight into how energy is used in modern IT equipment by breaking down the peak power usage of one generation of WSCs deployed at Google in 2007 categorized by main component group.

Although this breakdown can vary significantly depending on how systems are configured for a given workload domain, the graph indicates that CPUs can no longer be the sole focus of energy efficiency improvements because no one subsystem dominates the overall energy usage profile. Chapter 5 also discusses how overheads in power delivery and cooling can significantly increase the actual energy usage in WSCs.

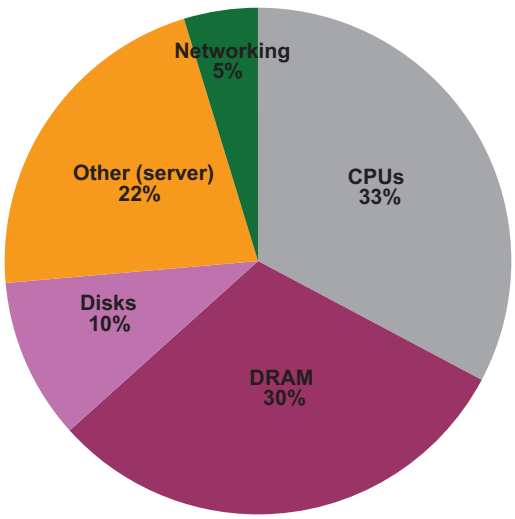

FIGURE 1.4: Approximate distribution of peak power usage by hardware subsystem in one of Google's datacenters (circa 2007).

1.6.6 Handling Failures

The sheer scale of WSCs requires that Internet services software tolerate relatively high component fault rates. Disk drives, for example, can exhibit annualized failure rates higher than 4% [65, 76]. Different deployments have reported between 1.2 and 16 average server-level restarts per year. With such high component failure rates, an application running across thousands of machines may need to react to failure conditions on an hourly basis. We expand on this topic further on Chapter 2, which describes the application domain, and Chapter 7, which deals with fault statistics.

CHAPTER 2

Workloads and Software Infrastructure

The applications that run on warehouse-scale computers (WSCs) dominate many system design trade-off decisions. This chapter outlines some of the distinguishing characteristics of software that runs in large Internet services and the system software and tools needed for a complete computing platform. Here is some terminology that defines the different software layers in a typical WSC deployment:

- *Platform-level software*—the common firmware, kernel, operating system distribution, and libraries expected to be present in all individual servers to abstract the hardware of a single machine and provide basic server-level services.
- *Cluster-level infrastructure*—the collection of distributed systems software that manages resources and provides services at the cluster level; ultimately, we consider these services as an operating system for a datacenter. Examples are distributed file systems, schedulers, remote procedure call (RPC) layers, as well as programming models that simplify the usage of resources at the scale of datacenters, such as MapReduce [19], Dryad [47], Hadoop [42], Sawzall [64], BigTable [13], Dynamo [20], and Chubby [7].
- *Application-level software*—software that implements a specific service. It is often useful to further divide application-level software into online services and offline computations because those tend to have different requirements. Examples of online services are Google search, Gmail, and Google Maps. Offline computations are typically used in large-scale data analysis or as part of the pipeline that generates the data used in online services; for example, building an index of the Web or processing satellite images to create map tiles for the online service.

2.1 DATACENTER VS. DESKTOP

Software development in Internet services differs from the traditional desktop/server model in many ways:

- *Ample parallelism*—Typical Internet services exhibit a large amount of parallelism stemming from both data- and request-level parallelism. Usually, the problem is not to find

parallelism but to manage and efficiently harness the explicit parallelism that is inherent in the application. Data parallelism arises from the large data sets of relatively independent records that need processing, such as collections of billions of Web pages or billions of log lines. These very large data sets often require significant computation for each parallel (sub) task, which in turn helps hide or tolerate communication and synchronization overheads. Similarly, request-level parallelism stems from the hundreds or thousands of requests per second that popular Internet services receive. These requests rarely involve read–write sharing of data or synchronization across requests. For example, search requests are essentially independent and deal with a mostly read-only database; therefore, the computation can be easily partitioned both within a request and across different requests. Similarly, whereas Web email transactions do modify user data, requests from different users are essentially independent from each other, creating natural units of data partitioning and concurrency.

- *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. Key pieces of Google's services have release cycles on the order of a couple of weeks compared to months or years for desktop software products. Google's front-end Web server binaries, for example, are released on a weekly cycle, with nearly a thousand independent code changes checked in by hundreds of developers—the core of Google's search services has been reimplemented nearly from scratch every 2 to 3 years. This environment creates significant incentives for rapid product innovation but makes it hard for a system designer to extract useful benchmarks even from established applications. Moreover, because Internet services are still a relatively new field, new products and services frequently emerge, and their success with users directly affects the resulting workload mix in the datacenter. For example, video services such as YouTube have flourished in relatively short periods and may present a very different set of requirements from the existing large customers of computing cycles in the datacenter, potentially affecting the optimal design point of WSCs in unexpected ways. A beneficial side effect of this aggressive software deployment environment is that hardware architects are not necessarily burdened with having to provide good performance for immutable pieces of code. Instead, architects can consider the possibility of significant software rewrites to take advantage of new hardware capabilities or devices.
- *Platform homogeneity*—The datacenter is generally a more homogeneous environment than the desktop as a target platform for software development. Large Internet services operations typically deploy a small number of hardware and system software configurations at any given time. Significant heterogeneity arises primarily from the incentives to deploy more cost-efficient components that become available over time. Homogeneity within a platform generation simplifies cluster-level scheduling and load balancing and reduces the

maintenance burden for platforms software (kernels, drivers, etc.). Similarly, homogeneity can allow more efficient supply chains and more efficient repair processes because automatic and manual repairs benefit from having more experience with fewer types of systems. In contrast, software for desktop systems can make few assumptions about the hardware or software platform they are deployed on, and their complexity and performance characteristics may suffer from the need to support thousands or even millions of hardware and system software configurations.

- *Fault-free operation*—Because Internet service applications run on clusters of thousands of machines—each of them not dramatically more reliable than PC-class hardware—the multiplicative effect of individual failure rates means that some type of fault is expected every few hours or less (more details are provided in Chapter 6). As a result, 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 need to 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.

Although the plentiful thread-level parallelism and a more homogeneous computing platform help reduce software development complexity in Internet services compared to desktop systems, the scale, the need to operate under hardware failures, and the speed of workload churn have the opposite effect.

2.2 PERFORMANCE AND AVAILABILITY TOOLBOX

Some basic programming concepts tend to occur often in both infrastructure and application levels because of their wide applicability in achieving high performance or high availability in large-scale deployments. The following table describes some of the most prevalent concepts.

	PERFORMANCE	AVAILABILITY	DESCRIPTION
Replication	Yes	Yes	Data replication is a powerful technique because it can improve both performance and availability. It is particularly powerful when the replicated data are not often modified because replication makes updates more complex.

(*continued*)

	PERFORMANCE	AVAILABILITY	DESCRIPTION
Sharding (partitioning)	Yes	Yes	Splitting a data set into smaller fragments (shards) and distributing them across a large number of machines. Operations on the data set are dispatched to some or all of the machines hosting shards, and results are coalesced by the client. The sharding policy can vary depending on space constraints and performance considerations. Sharding also helps availability because recovery of small data fragments can be done faster than larger ones.
Load-balancing	Yes		In large-scale services, service-level performance often depends on the slowest responder out of hundreds or thousands of servers. Reducing response-time variance is therefore critical. In a sharded service, load balancing can be achieved by biasing the sharding policy to equalize the amount of work per server. That policy may need to be informed by the expected mix of requests or by the computing capabilities of different servers. Note that even homogeneous machines can offer different performance characteristics to a load-balancing client if multiple applications are sharing a subset of the load-balanced servers.

(*continued*)

	PERFORMANCE	AVAILABILITY	DESCRIPTION
			In a replicated service, the load balancing agent can dynamically adjust the load by selecting which servers to dispatch a new request to. It may still be difficult to approach perfect load balancing because the amount of work required by different types of requests is not always constant or predictable.
Health checking and watchdog timers		Yes	In a large-scale system, failures often are manifested as slow or unresponsive behavior from a given server. In this environment, no operation can rely on a given server to respond to make forward progress. Moreover, it is critical to quickly determine that a server is too slow or unreachable and steer new requests away from it. Remote procedure calls must set well-informed time-out values to abort long-running requests, and infrastructure-level software may need to continually check connection-level responsiveness of communicating servers and take appropriate action when needed.
Integrity checks		Yes	In some cases, besides unresponsiveness, faults are manifested as data corruption. Although those may be rarer, they do occur and often in ways that underlying hardware or software checks do not catch (e.g., there are

(*continued*)

	PERFORMANCE	AVAILABILITY	DESCRIPTION
Integrity checks		Yes	known issues with the error coverage of some networking CRC checks). Extra software checks can mitigate these problems by changing the underlying encoding or adding more powerful redundant integrity checks.
Application-specific compression	Yes		Often a large portion of the equipment costs in modern datacenters is in the various storage layers. For services with very high throughput requirements, it is critical to fit as much of the working set as possible in DRAM; this makes compression techniques very important because the extra CPU overhead of decompressing is still orders of magnitude lower than the penalties involved in going to disks. Although generic compression algorithms can do quite well on the average, application-level compression schemes that are aware of the data encoding and distribution of values can achieve significantly superior compression factors or better decompression speeds.
Eventual consistency	Yes	Yes	Often, keeping multiple replicas up to date using the traditional guarantees offered by a database management system significantly increases complexity,

(*continued*)

PERFORMANCE	AVAILABILITY	DESCRIPTION
		hurts performance, and reduces availability of distributed applications [90]. Fortunately, large classes of applications have more relaxed requirements and can tolerate inconsistent views for limited periods, provided that the system eventually returns to a stable consistent state.

Response time of large parallel applications can also be improved by the use of redundant computation techniques. There are several situations that may cause a given subtask of a large parallel job to be much slower than its siblings, either due to performance interference with other workloads or software/hardware faults. Redundant computation is not as widely deployed as other techniques because of the obvious overheads involved. However, there are some situations in which the completion of a large job is being held up by the execution of a very small percentage of its subtasks. One such example is the issue of stragglers, as described in the paper on MapReduce [19]. In this case, a single slower worker can determine the response time of a huge parallel task. MapReduce's strategy is to identify such situations toward the end of a job and speculatively start redundant workers only for those slower jobs. This strategy increases resource usage by a few percentage points while reducing a parallel computation's completion time by more than 30%.

2.3 CLUSTER-LEVEL INFRASTRUCTURE SOFTWARE

Much like an operating system layer is needed to manage resources and provide basic services in a single computer, a system composed of thousands of computers, networking, and storage also requires a layer of software that provides an analogous functionality at this larger scale. We refer to this layer as the *cluster-level infrastructure*. The paragraphs that follow describe four broad groups of infrastructure software that make up this layer.

2.3.1 Resource Management

This is perhaps the most indispensable component of the cluster-level infrastructure layer. It controls the mapping of user tasks to hardware resources, enforces priorities and quotas, and provides basic task management services. In its simplest form, it can simply be an interface to manually (and statically) allocate groups of machines to a given user or job. A more useful version should present a higher level of abstraction, automate allocation of resources, and allow resource sharing at a finer level of granularity. Users of such systems should be able to specify their job requirements at a relatively high level (e.g., how much CPU performance, memory capacity, networking bandwidth, etc.) and have the scheduler translate those into an appropriate allocation of resources. It is increasingly important that cluster schedulers also consider power limitations and energy usage optimization when making scheduling decisions not only to deal with emergencies (such as cooling equipment failures) but also to maximize the usage of the provisioned datacenter power budget. Chapter 5 provides more detail on this topic.

2.3.2 Hardware Abstraction and Other Basic Services

Nearly every large-scale distributed application needs a small set of basic functionalities. Examples are reliable distributed storage, message passing, and cluster-level synchronization. Implementing this type of functionality correctly with high performance and high availability is complex in large clusters. It is wise to avoid reimplementing such tricky code for each application and instead create modules or services that can be reused. GFS [31], Dynamo [20], and Chubby [7] are examples of reliable storage and lock services for large clusters developed at Google and Amazon.

2.3.3 Deployment and Maintenance

Many of the tasks that are amenable to manual processes in a small deployment require a significant amount of infrastructure for efficient operations in large-scale systems. Examples are software image distribution and configuration management, monitoring service performance and quality, and triaging alarms for operators in emergency situations. The Autopilot system from Microsoft [48] offers an example design for some of this functionality for Windows Live datacenters. Monitoring the overall health of the hardware fleet also requires careful monitoring, automated diagnostics, and automation of the repairs workflow. Google's System Health Infrastructure, described in Pinheiro et al [65], is an example of the software infrastructure needed for efficient health management. Finally, performance debugging and optimization in systems of this scale need specialized solutions as well. The X-Trace [30] system developed at U.C. Berkeley is an example of monitoring infrastructure aimed at performance debugging of large distributed systems.

2.3.4 Programming Frameworks

The entire infrastructure described in the preceding paragraphs simplifies the deployment and efficient usage of hardware resources, but it does not fundamentally hide the inherent complexity of a large hardware cluster as a target for the average programmer. From a programmer's standpoint, hardware clusters have a deep and complex memory/storage hierarchy, heterogeneous components, failure-prone components, and scarcity of some resources (such as DRAM and datacenter-level networking bandwidth). The size of problems being solved obviously always complicates matters further. Some types of operations or subsets of problems are common enough in large-scale services that it pays off to build targeted programming frameworks that can drastically simplify the development of new products. MapReduce [19], BigTable [13], and Dynamo [20] are good examples of pieces of infrastructure software that greatly improve programmer productivity by automatically handling data partitioning, distribution, and fault tolerance within their respective domains.

2.4 APPLICATION-LEVEL SOFTWARE

Web search was one of the first large-scale Internet services to gain widespread popularity as the amount of Web content exploded in the mid-nineties, and organizing this massive amount of information went beyond what could be accomplished with the then existing human-managed directory services. However, as networking connectivity to homes and businesses continues to improve, it becomes more attractive to offer new services over the Internet, sometimes replacing computing capabilities that traditionally lived in the client. Web-based maps and email services are early examples of these trends. This increase in the breadth of services offered has resulted in a much larger diversity of application-level requirements. For example, a search workload may not require an infrastructure capable of high-performance atomic updates and is inherently forgiving of hardware failures (because absolute precision every time is less critical in Web search). This is not true for an application that tracks user clicks on sponsored links (ads). Clicks on ads are basically small financial transactions, which need many of the guarantees expected from a transactional database management system.

Once the diverse requirements of multiple services are considered, it becomes clear that the datacenter must be a general-purpose computing system. Although specialized hardware solutions might be a good fit for individual parts of services, the breadth of requirements makes it less likely that specialized hardware can make a large overall impact in the operation. Another factor against hardware specialization is the speed of workload churn; product requirements evolve rapidly, and smart programmers will learn from experience and rewrite the baseline algorithms and data

structures much more rapidly than hardware itself can evolve. Therefore, there is substantial risk that by the time a specialized hardware solution is implemented, it is no longer a good fit even for the problem area for which it was designed.

2.4.1 Workload Examples

Our objective here is not to describe Internet service workloads in detail, especially because the dynamic nature of this market will make those obsolete by publishing time. However, it is useful to describe at a high level two workloads that exemplify two broad classes of applications: online services and batch (offline) processing systems. Here we outline the basic architecture of a Web-search application as an example of an online system and a citation-based similarity computation that uses MapReduce as an example of a batch workload.

2.4.2 Online: Web Search

This is the quintessential "needle in a haystack" problem. Although it is hard to accurately determine the size of the Web at any point in time, it is safe to say that it consists of hundreds of billions of individual documents and that it continues to grow. If we assume the Web to contain 100 billion documents, with an average document size of 4 kB (after compression), the haystack is about 400 TB. The database for Web search is an index built from that repository by inverting that set of documents to create a repository in the logical format shown in Figure 2.1. A lexicon structure associates an ID to every term in the repository. The *termID* identifies a list of documents in which the term occurs, and some contextual information about it, such as position and various other attributes (e.g., is the term in the document title?).

The size of the resulting inverted index depends on the specific implementation, but it tends to be on the same order of magnitude as the original repository. The typical search query consists of a sequence of terms, and the system's task is to find the documents that contain all of the terms (an AND query) and decide which of those documents are most likely to satisfy the user. Queries can optionally contain special operators to indicate alternation (OR operators) or to restrict the search to occurrences of the terms in a particular sequence (phrase operators). For brevity we focus on the more common AND query.

Consider a query such as [*new york restaurants*]. The search algorithm must traverse the posting lists for each term (*new, york, restaurant*) until it finds all documents contained in all three posting lists. At that point it ranks the documents found using a variety of parameters, such as the overall importance of the document (in Google's case, that would be the PageRank score [59]) as well as many other properties associated with the occurrence of the terms in the document (such

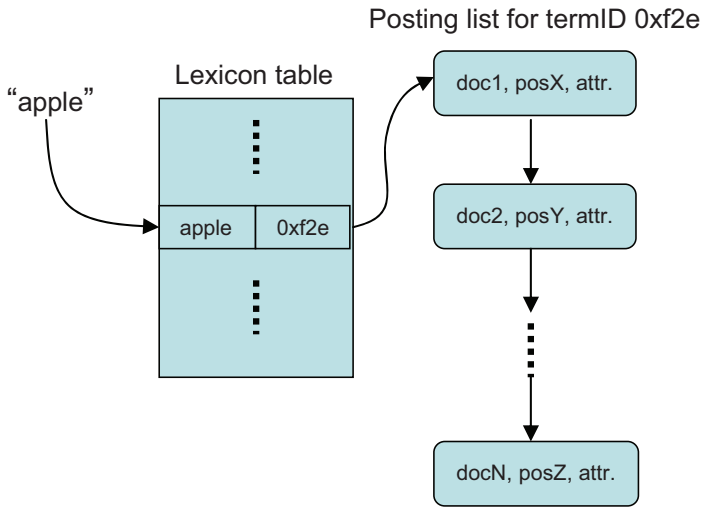

FIGURE 2.1: Logical view of a Web index.

as number of occurrences, positions, etc.) and returns the most highly ranked documents to the user.

Given the massive size of the index, this search algorithm may need to run across a few thousand machines. That is accomplished by splitting the index into load-balanced subfiles and distributing them across all of the machines. Index partitioning can be done by document or by term. The user query is received by a front-end Web server and distributed to all of the machines in the index cluster. As necessary for throughput or fault tolerance, multiple copies of index subfiles can be placed in different machines, in which case only a subset of the machines is involved in a given query. Index-serving machines compute local results, prerank them, and send their best results to the front-end system (or some intermediate server), which selects the best results from across the whole cluster. At this point, only the list of doc_IDs corresponding to the resulting Web page hits is known. A second phase is needed to compute the actual title, URLs, and a query-specific document snippet that gives the user some context around the search terms. This phase is implemented by sending the list of doc_IDs to a set of machines containing copies of the documents themselves. Once again, given the size of the repository, it needs to be partitioned and placed in a large number of servers.

The total user-perceived latency for the operations described above needs to be a fraction of a second; therefore, this architecture places heavy emphasis on latency reduction. However,

high-throughput is also a key performance metric because a popular service may need to support many thousands of queries per second. The index is updated frequently, but in the time granularity of handling a single query, it can be considered a read-only structure. Also, because there is no need for index lookups in different machines to communicate with each other except for the final merge step, the computation is very efficiently parallelized. Finally, further parallelism is available by exploiting the fact that there are no logical interactions across different Web search queries.

If the index is partitioned (sharded) by doc_ID, this workload has relatively small networking requirements in terms of average bandwidth because the amount of data exchanged between machines is typically not much larger than the size of the queries themselves (about a hundred bytes or so) but does exhibit some bursty behavior. Basically, the servers at the front-end act as traffic amplifiers as they distribute a single query to a very large number of servers. This creates a burst of traffic not only in the request path but possibly also on the response path as well. Therefore, even if overall network utilization is low, careful management of network flows is needed to minimize packet loss.

Finally, because Web search is an online service, it suffers from normal traffic variations because users are more active on the Web at different times of the day. Figure 2.2 illustrates this effect, showing that traffic at peak usage hours can be more than twice the traffic during off-peak periods. Such variability presents a challenge to system operators because the service must be sized for traffic intensities significantly higher than the average behavior.

2.4.3 Offline: Scholar Article Similarity

User requests provide many examples of large-scale computations required for the operation of Internet services. These computations are typically the types of data-parallel workloads needed to prepare or package the data that is subsequently used in online services. For example, computing PageRank or creating inverted index files from a Web repository is in this category. But here we use a different example: finding similar articles in a repository of academic papers and journals. This is a useful feature for Internet services that provide access to scientific publications, such as Google Scholar (http://scholar.google.com). Article similarity relationships complement keyword-based search systems as another way to find relevant information; after finding an article of interest, a user can ask the service to display other articles that are strongly related to the original article.

There are several ways to compute similarity scores, and it is often appropriate to use multiple methods and combine the results. In academic articles, various forms of citation analysis have been

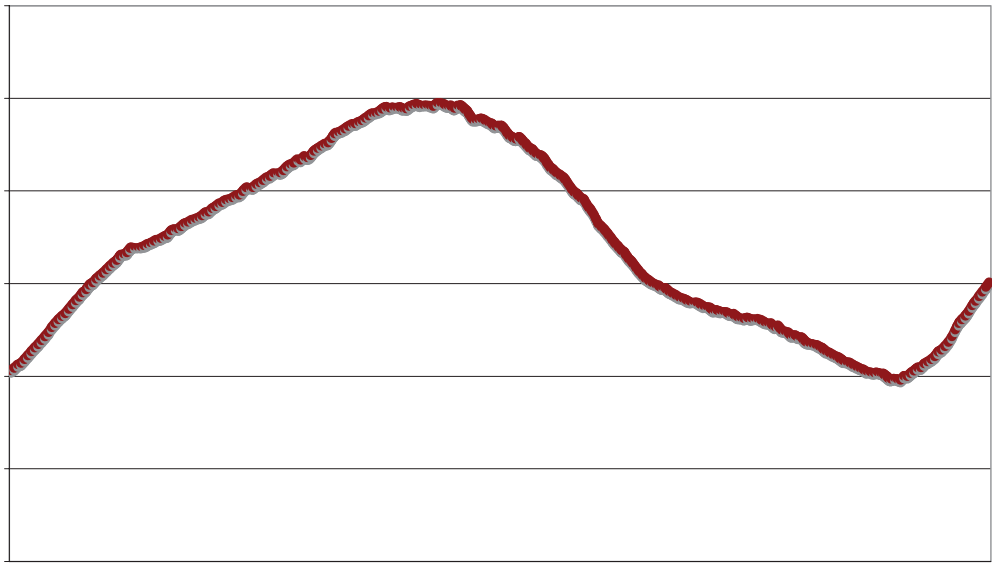

FIGURE 2.2: Example of daily traffic fluctuation for a search service in one datacenter; *x*-axis is a 24-h period and the *y*-axis is traffic measured in queries per second.

known to provide good-quality similarity scores. Here we consider one such type of analysis, called co-citation. The underlying idea is to count every article that cites articles A and B as a vote for the similarity between A and B. After that is done for all articles and appropriately normalized, we obtain a numerical score for the (co-citation) similarity between all pairs of articles and create a data structure that for each article returns an ordered list (by co-citation score) of similar articles. This data structure is periodically updated, and each update then becomes part of the serving state for the online service.

The computation starts with a citation graph that creates a mapping from each article identifier to a set of articles cited by it. The input data are divided into hundreds of files of approximately the same size (e.g., this can be done by taking a fingerprint of the article identifier, dividing it by the number of input files, and using the remainder as the file ID) to enable efficient parallel execution. We use a sequence of MapReduce runs to take a citation graph and produce co-citation similarity score vector for all articles. In the first Map phase, we take each citation list (A1, A2, A3, . . . , A*n*) and generate all pairs of documents in the citation list, feeding them to the Reduce phase, which counts all occurrences of each pair. This first step results in a structure that associates all pairs of co-cited documents with a co-citation count. Note that this becomes much less than a quadratic

explosion because most documents have a co-citation count of zero and are therefore omitted. A second MapReduce pass groups all entries for a given document, normalizes their scores, and generates a list of documents with decreasing similarity scores to the original one.

This two-pass data-parallel program executes on hundreds of servers with relatively lightweight computation in each stage followed by significant all-to-all communication between the Map and Reduce workers in each phase. Unlike Web search, however, the networking traffic is streaming in nature, which makes it friendlier to existing congestion control algorithms. Also contrary to Web search, the latency of individual tasks is much less important than the overall parallel efficiency of the workload.

2.5 A MONITORING INFRASTRUCTURE

An important part of the cluster-level infrastructure software layer is concerned with various forms of system introspection. Because the size and complexity of both the workloads and the hardware infrastructure make the monitoring framework a fundamental component of any such deployments, we describe it here in more detail.

2.5.1 Service-Level Dashboards

System operators must keep track of how well an Internet service is meeting its target service level. The monitoring information must be very fresh to let an operator (or an automated system) take corrective actions quickly and avoid significant disruption—within seconds, not minutes. Fortunately, the most critical information needed is restricted to just a few signals that can be collected from the front-end servers, such as latency and throughput statistics for user requests. In its simplest form, such a monitoring system could be simply a script that polls all front-end servers every few seconds for the appropriate signals and displays them to operators in a dashboard.

Large-scale services often need more sophisticated and scalable monitoring support, as the number of front-ends can be quite large, and more signals are needed to characterize the health of the service. For example, it may be important to collect not only the signals themselves but also their derivatives over time. The system may also need to monitor other business-specific parameters in addition to latency and throughput. The monitoring system may need to support a simple language that lets operators create derived parameters based on baseline signals being monitored. Finally, the system may need to generate automatic alerts to on-call operators depending on monitored values and thresholds. Getting a system of alerts (or alarms) well tuned can be tricky because alarms that trigger too often because of false positives will cause operators to ignore real ones, whereas alarms that trigger only in extreme cases might get the operator's attention too late to allow smooth resolution of the underlying issues.

2.5.2 Performance Debugging Tools

Although service-level dashboards let operators quickly identify service-level problems, they typically lack the detailed information that would be required to know "why" a service is slow or otherwise not meeting requirements. Both operators and the service designers need tools to let them understand the complex interactions between many programs, possibly running on hundreds of servers, so they can determine the root cause of performance anomalies and identify bottlenecks. Unlike a service-level dashboard, a performance debugging tool may not need to produce information in real-time for online operation. Think of it as the datacenter analog of a CPU profiler that determines which function calls are responsible for most of the time spent in a program.

Distributed system tracing tools have been proposed to address this need. These tools attempt to determine all the work done in a distributed system on behalf of a given initiator (such as a user request) and detail the causal or temporal relationships among the various components involved.

These tools tend to fall into two broad categories: black-box monitoring systems and application/middleware instrumentation systems. WAP5 [72] and the Sherlock system [8] are examples of black-box monitoring tools. Their approach consists of observing networking traffic among system components and inferring causal relationships through statistical inference methods. Because they treat all system components (except the networking interfaces) as black boxes, these approaches have the advantage of working with no knowledge of or assistance from applications or software infrastructure components. However, this approach inherently sacrifices information accuracy because all relationships must be statistically inferred. Collecting and analyzing more messaging data can improve accuracy but at the expense of higher monitoring overheads.

Instrumentation-based tracing schemes, such as Pip [71], Magpie [9], and X-trace [30], take advantage of the ability to explicitly modify applications or middleware libraries for passing tracing information across machines and across module boundaries within machines. The annotated modules typically also log tracing information to local disks for subsequent collection by an external performance analysis program. These systems can be very accurate as there is no need for inference, but they require all components of the distributed system to be instrumented to collect comprehensive data. The Dapper system, developed at Google, is an example of an annotation-based tracing tool that remains effectively transparent to application-level software by instrumenting a few key modules that are commonly linked with all applications, such as messaging, control flow, and threading libraries.

Finally, it is extremely useful to build the ability into binaries (or run-time systems) to obtain CPU, memory, and lock contention profiles of in-production programs. This can eliminate the need to redeploy new binaries to investigate performance problems.

2.5.3 Platform-Level Monitoring

Distributed system tracing tools and service-level dashboards are both measuring health and performance of applications. These tools can infer that a hardware component might be misbehaving, but that is still an indirect assessment. Moreover, because both cluster-level infrastructure and application-level software are designed to tolerate hardware component failures, monitoring at these levels can miss a substantial number of underlying hardware problems, allowing them to build up until software fault-tolerance can no longer mitigate them. At that point, service disruption could be severe. Tools that continuously and directly monitor the health of the computing platform are needed to understand and analyze hardware and system software failures. In Chapter 6, we discuss some of those tools and their use in Google's infrastructure in more detail.

2.6 BUY VS. BUILD

Traditional IT infrastructure makes heavy use of third-party software components such as databases and system management software, and concentrates on creating software that is specific to the particular business where it adds direct value to the product offering, for example, as business logic on top of application servers and database engines. Large-scale Internet services providers such as Google usually take a different approach in which both application-specific logic and much of the cluster-level infrastructure software is written in-house. Platform-level software does make use of third-party components, but these tend to be open-source code that can be modified in-house as needed. As a result, more of the entire software stack is under the control of the service developer.

This approach adds significant software development and maintenance work but can provide important benefits in flexibility and cost efficiency. Flexibility is important when critical functionality or performance bugs must be addressed, allowing a quick turn-around time for bug fixes at all levels. It is also extremely advantageous when facing complex system problems because it provides several options for addressing them. For example, an unwanted networking behavior might be very difficult to address at the application level but relatively simple to solve at the RPC library level, or the other way around.

Historically, a primary reason favoring build vs. buy was that the needed warehouse-scale software infrastructure simply was not available commercially. In addition, it may be hard for third-party software providers to adequately test and tune their software unless they themselves maintain large clusters. Lastly, in-house software may be simpler and faster because it can be designed to address only the needs of a small subset of services, and can therefore be made much more efficient in that domain. For example, BigTable omits some of the core features of a traditional SQL data-

base to gain much higher throughput and scalability for its intended use cases, and GFS falls short of offering a fully Posix compliant file system for similar reasons.

2.7 FURTHER READING

The articles by Hamilton [43], Brewer [10], and Vogels [90] provide interesting further reading on how different organizations have reasoned about the general problem of deploying Internet services at a very large scale.

CHAPTER 3

Hardware Building Blocks

As mentioned earlier, the architecture of warehouse-scale computers (WSCs) is largely defined by the choice of its building blocks. This process is analogous to choosing logic elements for implementing a microprocessor, or selecting the right set of chipsets and components in architecting a server platform. In this case, the main building blocks are server hardware, networking fabric, and storage hierarchy components. In this chapter we focus on the server hardware choices, with the objective of building intuition for how such choices are made. We hope to extend this section with additional material for storage and networking in an upcoming revision of this publication.

3.1 COST-EFFICIENT HARDWARE

Clusters of low-end servers are the preferred building blocks for WSCs today [5]. This happens for a number of reasons, the primary one being the underlying cost-efficiency of low-end servers when compared with the high-end shared memory systems that had earlier been the preferred building blocks for the high-performance and technical computing space. Low-end server platforms share many key components with the very high-volume personal computing market, and therefore benefit more substantially from economies of scale.

It is typically hard to do meaningful cost-efficiency comparisons because prices fluctuate and performance is subject to benchmark characteristics and the level of effort put into the benchmarking effort. Data from the TPC-C benchmarking [85] entries are probably the closest one can get to information that includes both hardware costs and application-level performance in a single set of metrics. Therefore, for this exercise we have compared the best performing TPC-C system in late 2007—the HP Integrity Superdome-Itanium2 [87]—with the top system in the price/performance category (HP ProLiant ML350 G5 [88]). Both systems were benchmarked within a month of each other, and the TPC-C executive summaries include a breakdown of the costs of both platforms so that we can do a rational analysis of cost-efficiency.

Table 3.1 shows the basic machine configuration for these two servers. Using the official benchmark metric, the ProLiant is about four times more cost-efficient than the Superdome. That is a large enough gap to be meaningful in platform selection decisions. The TPC-C benchmarking scaling rules arguably penalize the ProLiant in the official metrics by requiring a fairly large storage

TABLE 3.1: Server hardware configuration, benchmark results, and derived (unofficial) cost-efficiency metrics for a large SMP server and a low-end, PC-class server.

	HP INTEGRITY SUPERDOME-ITANIUM2	HP PROLIANT ML350 G5
Processor	64 sockets, 128 cores (dual-threaded), 1.6 GHz Itanium2, 12 MB last-level cache	1 socket, quad-core, 2.66 GHz X5355 CPU, 8 MB last-level cache
Memory	2,048 GB	24 GB
Disk storage	320,974 GB, 7,056 drives	3,961 GB, 105 drives
TPC-C price/performance	$2.93/tpmC	$0.73/tpmC
price/performance (server HW only)	$1.28/transactions per minute	$0.10/transactions per minute
Price/performance (server HW only) (no discounts)	$2.39/transactions per minute	$0.12/transactions per minute

subsystem in the official benchmarking configuration: the storage subsystem for the ProLiant accounts for about three fourths of the total server hardware costs, as opposed to approximately 40% in the Superdome setup. If we exclude storage costs, the resulting price/performance advantage of the ProLiant platform increases by a factor of 3× to more than 12×. Benchmarking rules also allow typical hardware discounts to be applied to the total cost used in the price/performance metric, which once more benefits the Superdome because it is a much more expensive system (~ $12M vs $75K for the ProLiant) and therefore takes advantage of deeper discounts. Assuming one would have the same budget to purchase systems of one kind or the other, it might be reasonable to assume a same level of discounting for either case. If we eliminate discounts in both cases, the ProLiant server hardware becomes roughly 20 times more cost-efficient than the Superdome.

3.1.1 How About Parallel Application Performance?

The analysis above is certainly unfair to the high-end server because it does not take into account its drastically superior intercommunication performance. Nodes in a large SMP may communicate

at latencies on the order of 100 ns, whereas the LAN-based networks usually deployed in clusters of servers will experience latencies at or above 100 μs. It is certainly true that workloads which have intensive communication patterns will perform significantly better in a 128 processor-core SMP than in an Ethernet-connected cluster of 32 four-core low-end servers. In WSC environments, however, it is likely that workloads will be too large for even the largest high-end SMP servers. Therefore, the interesting question to ask is how much better a cluster of thousands of processor cores can perform when it is built with one class of machines or the other. The following very simple model can help us understand such a comparison at a high level.

Assume that a given parallel task execution time can be roughly modeled as a fixed local computation time plus the latency penalty of accesses to global data structures. If the computation fits into a single large shared memory system, those global data accesses will be performed at roughly DRAM speeds (~100 ns). If the computation only fits in multiple of such nodes, some global accesses will be much slower, on the order of typical LAN speeds (~100 μs). Let us assume further that accesses to the global store are uniformly distributed among all nodes so that the fraction of global accesses that map to the local node is inversely proportional to the number of nodes in the system—a node here is a shared-memory domain, such as one Integrity system or one ProLiant server. If the fixed local computation time is of the order of 1 ms—a reasonable value for high-throughput Internet services—the equation that determines the program execution time is as follows:

$$\text{Execution time} = 1\text{ ms} + f * [100\text{ ns}/\#\text{ nodes} + 100\ \mu\text{s} * (1 - 1/\#\text{ nodes})]$$

where the variable f is the number of global accesses per (1 ms) work unit. In Figure 3.1, we plot the execution time of this parallel workload as the number of nodes involved in the computation increases. Three curves are shown for different values of f, representing workloads with light-communication ($f = 1$), medium-communication ($f = 10$), and high-communication ($f = 100$) patterns. Note that in our model, the larger the number of nodes, the higher the fraction of remote global accesses.

The curves in Figure 3.1 have two interesting aspects worth highlighting. First, under light communication, there is relatively small performance degradation from using clusters of multiple nodes. For medium- and high-communication patterns, the penalties can be quite severe, but they are most dramatic when moving from a single node to two, with rapidly decreasing additional penalties for increasing the cluster size. Using this model, the performance advantage of a single 128-processor SMP over a cluster of thirty-two 4-processor SMPs could be more than a factor of 10×.

By definition, WSC systems will consist of thousands of processor cores. Therefore, we would like to use this model to compare the performance of a cluster built with Superdome-like servers with one built with ProLiant-like ones. Here we assume that the per-core performance is the same

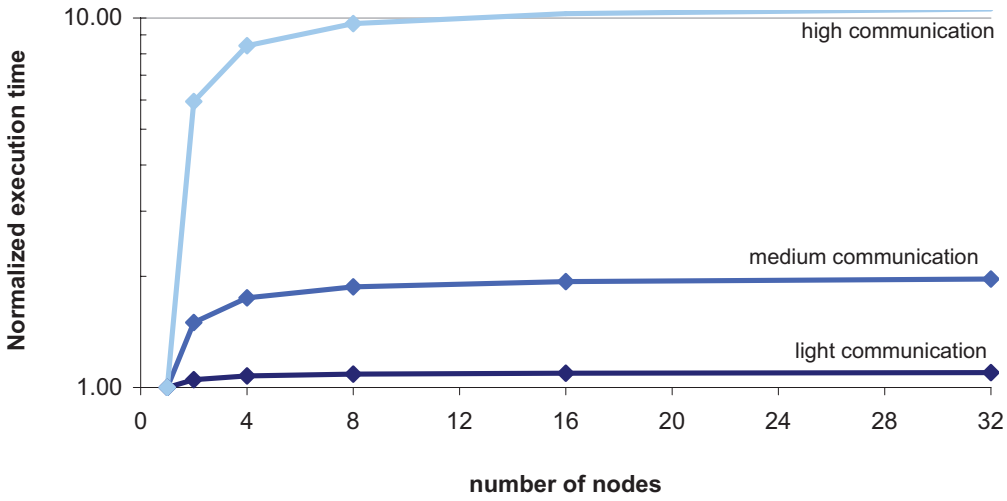

FIGURE 3.1: Execution time of parallel tasks as the number of SMP nodes increases for three levels of communication intensity. Execution time is normalized to the single node case and plotted in logarithmic scale.

for both systems and that servers are interconnected using an Ethernet-class fabric. We trust that although our model is exceedingly simple (e.g., it does not account for contention effects), it suffices to capture the effects we are interested in.

In Figure 3.2, we apply our model to clusters of size varying between 512 and 4,192 cores and show the performance advantage of an implementation using Superdome-type servers (128 cores in a single shared memory domain) vs. one using ProLiant-type servers (four-core SMP). In the figure, the cluster size of 512 cores compares the performance of a cluster of four Superdome-type systems with the performance of a cluster built with 128 ProLiant-type systems. It is interesting to note how quickly the performance edge of the cluster based on high-end server deteriorates as the cluster size increases. If the application requires more than two thousand cores, a cluster of 512 low-end servers performs within approximately 5% of one built with 16 high-end servers, even under a heavy communication pattern. With this low a performance gap, the price premium of the high-end server (4–20 times higher) renders it an unattractive option.

The point of this analysis is qualitative in nature. It is intended primarily to illustrate how we need to reason differently about baseline platform choice when architecting systems for applications that are too large for any single high-end server. The broad point is that the performance effects that matter most are those that benefit the system at the warehouse scale. Performance enhancements that have the greatest impact for computation that is local to a single node (such as fast SMP-style

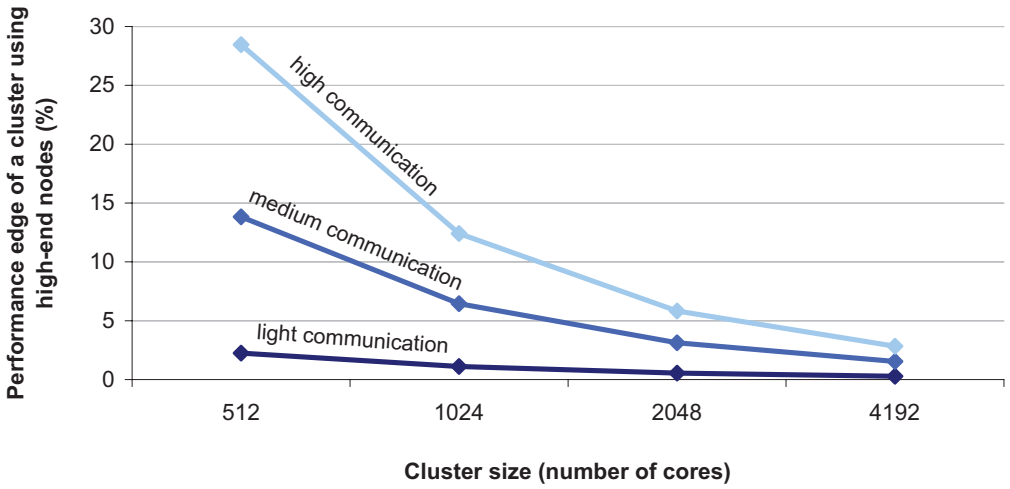

FIGURE 3.2: Performance advantage of a cluster built with high-end server nodes (128-core SMP) over a cluster with the same number of processor cores built with low-end server nodes (four-core SMP), for clusters of varying size.

communication in our example) are still very important. But if they carry a heavy additional cost, their cost-efficiency may not be as competitive for WSCs as they are for small-scale computers.

3.1.2 How Low-End Can You Go?

Clearly one could extend the argument laid out above to go further and use computing building blocks that are even smaller, such as what can be built using embedded components. Lim et al [51], for example, make the case for exactly such alternatives as being advantageous to low-end server platforms once all power-related costs are considered (including amortization of datacenter build-out costs and the cost of energy). More recently, Hamilton [44] makes a similar argument although using PC-class components instead of embedded ones. The advantages of using smaller, slower CPUs are a very similar to the arguments for using mid-range commodity servers instead of high-end SMPs:

- Multicore CPUs in mid-range servers typically carry a price/performance premium over lower-end processors so that the same amount of throughput can be bought two to five times more cheaply with multiple smaller CPUs.
- Many applications are memory-bound so that faster CPUs do not scale well for large applications, further enhancing the price advantage of simpler CPUs.

- Slower CPUs are more power efficient; typically, CPU power decreases by $O(k^2)$ when CPU frequency decreases by k.

However, offsetting effects diminish these advantages so that increasingly smaller building blocks become increasingly unattractive for WSCs.

Perhaps most importantly, although many Internet services benefit from seemingly unbounded request- and data-level parallelism, such systems are not immune from Amdahl's law. As the number of offered parallel threads increases, it can become increasingly difficult to reduce serialization and communication overheads. In a limit case, the amount of inherently serial work performed on behalf of a user request by an extremely slow single-threaded hardware will dominate overall execution time. Also, the larger the number of threads that are handling a parallelized request, the larger the variability in response times from all these parallel tasks because load balancing gets harder and unpredictable performance interference becomes more noticeable. Because often all parallel tasks must finish before a request is completed, the overall response time becomes the maximum response time of any subtask, and a larger number of subtasks will push further into the long tail of subtask response times, thus impacting overall service latency.

As a result, although hardware costs may diminish, software development costs may increase because more applications must be explicitly parallelized or further optimized. For example, suppose that a web service currently runs with a latency of 1-s per user request, half of it caused by CPU time. If we switch to a cluster with lower-end servers whose single-thread performance is three times slower, the service's response time will double to 2-s and application developers may have to spend a substantial amount of effort to optimize the code to get back to the 1-s latency level.

Networking requirements also increase with larger numbers of smaller systems, increasing networking delays and the cost of networking (since there are now more ports in an already expensive switching fabric). It is possible to mitigate this effect by locally interconnecting a small number of slower servers to share a network link, but the cost of this interconnect may offset some of the price advantage gained by switching to cheaper CPUs.

Smaller servers may also lead to lower utilization. Consider the task of allocating a set of applications across a pool of servers as a bin-packing problem—each of the servers is a bin, and we try to fit as many applications as possible into each bin. Clearly, that task is harder when the bins are small because many applications may not completely fill a server and yet use too much of its CPU or RAM to allow a second application to coexist on the same server.

Finally, even embarrassingly parallel algorithms are sometimes intrinsically less efficient when computation and data are partitioned into smaller pieces. That happens, for example, when the stop criterion for a parallel computation is based on global information. To avoid expensive global communication and global lock contention, local tasks may use heuristics that are based on their local

progress only, and such heuristics are naturally more conservative. As a result, local subtasks may execute for longer than they might have if there were better hints about global progress. Naturally, when these computations are partitioned into smaller pieces, this overhead tends to increase.

As a rule of thumb, a lower-end server building block must have a healthy cost-efficiency advantage over a higher-end alternative to be competitive. At the moment, the sweet spot for many large-scale services seems to be at the low-end range of server-class machines (which overlaps somewhat with that of higher-end personal computers).

3.1.3 Balanced Designs

Computer architects are trained to solve the problem of finding the right combination of performance and capacity from the various building blocks that make up a WSC. In this chapter we have shown an example of how the right building blocks are only apparent once one focuses on the entire WSC system. The issue of balance must also be addressed at this level. It is important to characterize the kinds of workloads that will execute on the system with respect to their consumption of various resources, while keeping in mind three important considerations:

- Smart programmers may be able to restructure their algorithms to better match a more inexpensive design alternative. There is opportunity here to find solutions by software-hardware co-design, while being careful not to arrive at machines that are too complex to program.
- The most cost-efficient and balanced configuration for the hardware may be a match with the combined resource requirements of multiple workloads and not necessarily a perfect fit for any one workload. For example, an application that is seek-limited may not fully use the capacity of a very large disk drive but could share that space with an application that needs space mostly for archival purposes.
- Fungible resources tend to be more efficiently used. Provided there is a reasonable amount of connectivity within a WSC, effort should be put on creating software systems that can flexibly utilize resources in remote servers. This affects balance decisions in many ways. For instance, effective use of remote disk drives may require that the networking bandwidth to a server be equal or higher to the combined peak bandwidth of all the disk drives locally connected to the server.

The right design point depends on more than the high-level structure of the workload itself because data size and service popularity also play an important role. For example, a service with huge data sets but relatively small request traffic may be able to serve most of its content directly from disk drives, where storage is cheap (in $/GB) but throughput is low. Extremely popular services that

either have small data set sizes or significant data locality to be exploited can benefit from in-memory serving instead. Finally, workload churn in this space is also a challenge to WSC architects. It is possible that the software base may evolve so fast that a server design choice becomes suboptimal during its lifetime (typically 3–4 years). This issue is even more important for the WSC as a whole because the lifetime of a datacenter facility generally spans several server lifetimes, or more than a decade or so. In those cases it is useful to try to envision the kinds of machinery or facility upgrades that may be necessary over the lifetime of the WSC system and take that into account during the design phase of the facility.

CHAPTER 4

Datacenter Basics

Datacenters are essentially very large devices that consume electrical power and produce heat. The datacenter's cooling system removes that heat—consuming additional energy in the process, whose heat must be removed as well. It is not surprising, then, that the bulk of the construction costs of a datacenter are proportional to the amount of power delivered and the amount of heat to be removed; in other words, most of the money is spent either on power conditioning and distribution or on cooling systems. Typical construction costs for a large datacenter are in the $10–20/W range (see Section 6.1) but vary considerably depending on size, location, and design.

4.1 DATACENTER TIER CLASSIFICATIONS

The overall design of a datacenter is often classified as belonging to "Tier I–IV" [67].

- Tier I datacenters have a single path for power and cooling distribution, without redundant components.
- Tier II adds redundant components to this design (N + 1), improving availability.
- Tier III datacenters have multiple power and cooling distribution paths but only one active path. They also have redundant components and are concurrently maintainable, that is, they provide redundancy even during maintenance, usually with an N + 2 setup.
- Tier IV datacenters have two active power and cooling distribution paths, redundant components in each path, and are supposed to tolerate any single equipment failure without impacting the load.

These tier classifications are not 100% precise. Most commercial datacenters fall somewhere between tiers III and IV, choosing a balance between construction costs and reliability. Real-world datacenter reliability is also strongly influenced by the quality of the organization running the datacenter, not just by the datacenter's design. Typical availability estimates used in the industry range from 99.7% availability for tier II datacenters to 99.98% and 99.995% for tiers III and IV, respectively.

Datacenter sizes vary widely. Two thirds of US servers are housed in datacenters smaller than 5,000 sq ft (450 sqm) and with less than 1 MW of critical power [26] (p. 27). Most large datacenters are built to host servers from multiple companies (often called co-location datacenters, or "colos") and can support a critical load of 10–20 MW. Very few datacenters today exceed 30 MW of critical capacity.

4.2 DATACENTER POWER SYSTEMS

Figure 4.1 shows the components of a typical datacenter. Power enters the building from an outside transformer typically located in the utility's substation; this part of the power system is often called "medium voltage" (typically 10–20 kV) to distinguish it from the high-voltage long-distance power lines (60–400 kV) and the "low-voltage" internal power distribution (110–600 V). The medium-voltage lines terminate at the primary switchgear, which includes breakers to protect against electrical faults and transformers to scale the voltage down to 400–600 V. The low-voltage power then flows into the uninterruptible power supply (UPS) systems, which also take a second feed (at the same voltage) from a set of diesel generators that will jump into action when utility power fails.

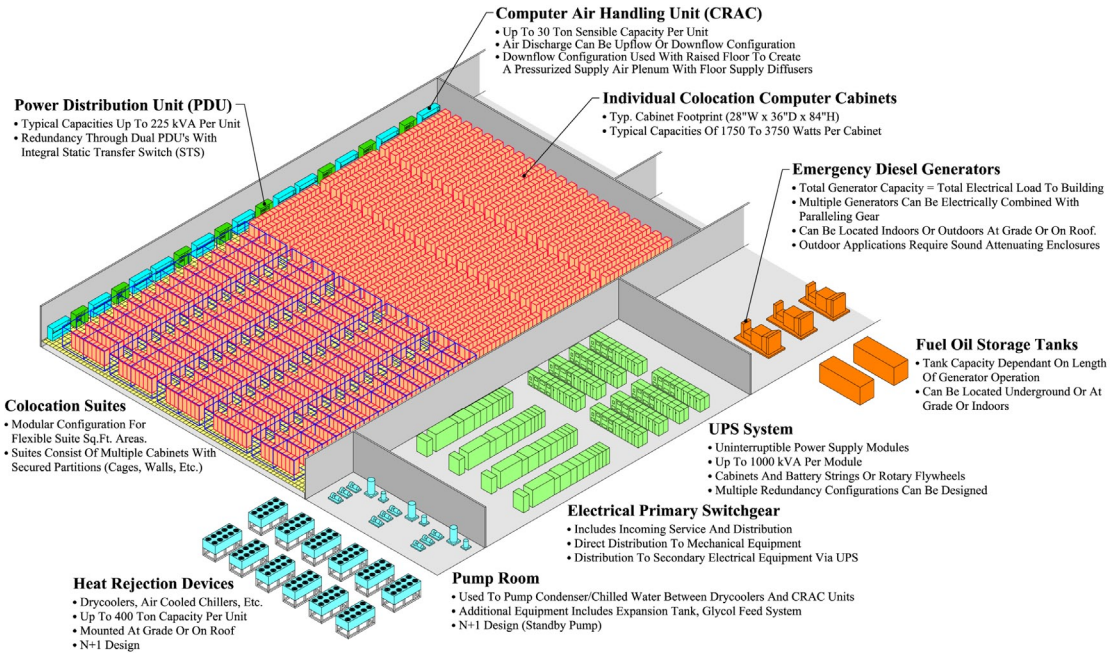

FIGURE 4.1: The main components of a typical datacenter (image courtesy of DLB Associates [23]).

4.2.1 UPS Systems

The UPS typically combines three functions in one system.

- First, it contains a transfer switch that chooses the active power input (either utility power or generator power). After a power failure, the transfer switch senses when the generator has started and is ready to provide power; typically, a generator takes 10–15 s to start and assume the full rated load.
- Second, the UPS contains batteries or flywheels to bridge the time between the utility failure and the availability of generator power. A typical UPS accomplishes this with an AC–DC–AC double conversion; that is, input AC power is converted to DC, which feeds a UPS-internal DC bus that is also connected to strings of batteries. The output of the DC bus is then converted back to AC to feed the datacenter equipment. Thus, when utility power fails, the UPS loses input (AC) power but retains internal DC power because the batteries still supply it, and thus retains AC output power after the second conversion step. Eventually, the generator starts and resupplies input AC power, relieving the UPS batteries of the datacenter load.
- Third, the UPS conditions the incoming power feed, removing voltage spikes or sags, or harmonic distortions in the AC feed. This conditioning is naturally accomplished via the double conversion steps.

Because the UPS batteries take up a sizeable amount of space, the UPS is typically housed in a separate UPS room, not on the datacenter floor. Typical UPS sizes range from hundreds of kilowatts up to 2 MW.

4.2.2 Power Distribution Units

The UPS output is then routed to power distribution units (PDUs) that sit on the datacenter floor. PDUs resemble breaker panels in residential houses: they take a hefty higher-voltage feed (typically 200–480 V) and break it up into the many 110- or 220-V circuits that feed the actual servers on the floor. Each circuit is protected by its own breaker so that a ground short in a server or a power supply will trip only the breaker for that circuit, not the entire PDU or even the UPS. A typical PDU handles 75–225 kW of load, whereas a typical circuit handles 20 or 30 A at 110–220 V, that is, a maximum of 6 kW. PDUs often provide additional redundancy by accepting two independent power sources (typically called “A side” and “B side”) and being able to switch between them with a very small delay so that the loss of one source does not interrupt power to the servers. In this scenario, the datacenter’s UPS units are duplicated into an A and B side, so that even the failure of a UPS will not interrupt server power.

Real-world datacenters contain many variants of the simplified design described here. Typical variants include the "paralleling" of generators or UPS units, an arrangement where multiple devices feed a shared bus so that the load of a failed device can be picked up by other devices, similar to handling disk failures in a RAID system. Common paralleling arrangements include N + 1 configurations (allowing one failure or maintenance), N + 2 configurations (allowing one failure even when one unit is offline for maintenance), and 2N (redundant pairs).

4.3 DATACENTER COOLING SYSTEMS

The cooling system is somewhat simpler than the power system. Typically, the datacenter's floor is raised—a steel grid resting on stanchions is installed 2–4 ft above the concrete floor (as shown in Figure 4.2). The under-floor area is often used to route power cables to racks, but its primary use is to distribute cool air to the server racks.

4.3.1 CRAC Units

CRAC units (*CRAC* is a 1960s term for *computer room air conditioning*) pressurize the raised floor plenum by blowing cold air into the plenum. This cold air escapes from the plenum through perforated tiles that are placed in front of server racks and then flows through the servers, which expel warm air in the back. Racks are arranged in long aisles that alternate between cold aisles and hot aisles to avoid mixing hot and cold air. (Mixing cold air with hot reduces cooling efficiency; some newer datacenters even partition off the hot aisles with walls to avoid leaking the hot air

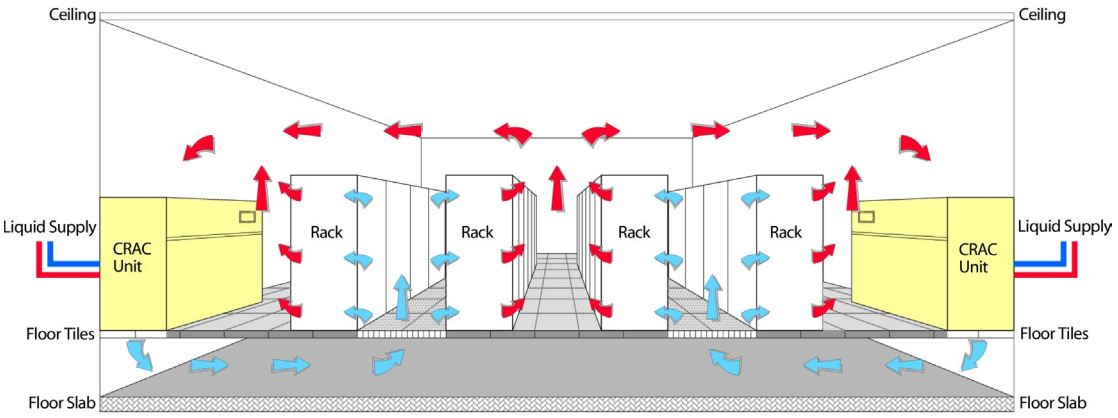

FIGURE 4.2: Datacenter raised floor with hot–cold aisle setup (image courtesy of DLB Associates [23]).

back into the cool room [63].) Eventually, the hot air produced by the servers recirculates back to the intakes of the CRAC units that cool it and then exhaust the cool air into the raised floor plenum again.

CRAC units consist of coils through which a liquid coolant is pumped; fans push air through these coils, thus cooling it. A set of redundant pumps circulate cold coolant to the CRACs and warm coolant back to a chiller or cooling tower, which expel the heat to the outside environment. Typically, the incoming coolant is at 12–14°C, and the air exits the CRACs at 16–20°C, leading to cold aisle (server intake) temperatures of 18–22°C. (The server intake temperature typically is higher than the CRAC exit temperature because it heats up slightly on its way to the server and because of recirculation that cannot be eliminated.) The warm coolant returns to a chiller that cools it down again to 12–14°C.

4.3.2 Free Cooling

Newer datacenters often insert a cooling tower to precool the condenser water loop fluid using "free cooling" before it reaches the chiller. Free cooling is not really free, but it is much more energy efficient than cooling with a chiller.

Water-based free cooling uses cooling towers to dissipate heat. The cooling towers use a separate cooling loop in which water absorbs the coolant's heat in a heat exchanger. In the cooling tower, the warm water then flows over a large-surface structure and transfers heat to the outside air via evaporation, thus cooling down. Note that the warm water can be cooled to a temperature lower than the ambient temperature if the air is relatively dry. As ambient air is drawn past a flow of water, evaporation lowers the temperature of the water close to the "wet bulb" air temperature, that is, lower than the ambient "dry bulb" air temperature. Cooling towers work best in temperate climates with low humidity; ironically, they do not work as well in very cold climates because additional mechanisms are needed to prevent ice formation on the towers.

Alternatively, a free cooling system can use a glycol-based radiator mounted outside the building to dissipate some of the heat. This works very well in cold climates (say, a winter in Chicago) but works less well at moderate or warm temperatures because heat exchange via air-to-air convection is less effective than via evaporation. Alternatively, some designs skip the heat exchange step via a CRAC and instead use large fans to push outside air directly into the room or the raised floor plenum when outside temperatures allow that (for an extreme experiment in this area, see [2]).

Most of the mechanical cooling equipment is also backed up by generators (and sometimes UPS units) because the datacenter cannot operate without cooling for more than a few minutes before overheating. In a typical datacenter, chillers and pumps can add 40% or more to the critical load that needs to be supported by generators.

4.3.3 Air Flow Considerations

Most datacenters use the raised floor setup discussed above. To change the amount of cooling delivered to a particular rack or row, we can simply adjust the number of perforated tiles in a row by replacing solid tiles with perforated ones or vice versa. For cooling to work well, the cold airflow coming through the tiles needs to match the horizontal airflow through the servers in the rack—for example, if a rack has 10 servers with an airflow of 100 cfm (cubic feet per minute) each, then the net flow out of the perforated tile should be 1,000 cfm (or higher if the air path to the servers is not tightly controlled). If it is lower, all the cool air will be sucked in by the some of the servers on the bottom of the rack, and the servers at the top will instead suck in warm air from above the rack; this undesirable effect is often called "recirculation" because warm air recirculates from the exhaust of one server into the intake of a neighboring server. Some datacenters deliver additional cold air from ducts above the server racks to solve that problem.

This need for matched air flow limits the power density of datacenters. For a fixed temperature differential across a server, a rack's airflow need increases with rack power consumption, and the airflow supplied via the raised floor tiles must increase linearly with power. That in turn increases the amount of static pressure we need to generate in the under floor plenum, which increases the fan power needed by CRACs to push cold air into the plenum. At low densities, this is easy to accomplish, but at some point the laws of physics start to catch up with us and make it economically impractical to further increase pressure and airflow. Typically, these limitations make it hard to exceed power densities of more than 150–200 W/sq ft without substantially increased cost.

As mentioned before, newer datacenters have started to physically separate the hot aisles from the room to eliminate recirculation and optimize the path back to the CRACs. In this setup the entire room is filled with cool air (because the warm exhaust is kept inside a separate plenum or duct system) and, thus, all servers in a rack will ingest air at the same temperature [63].

4.3.4 In-Rack Cooling

In-rack cooling products are a variant on the idea of filling the entire room with cool air and can also increase power density and cooling efficiency beyond the conventional raised-floor limit. Typically, an in-rack cooler adds an air-to-water heat exchanger at the back of a rack so that the hot air exiting the servers immediately flows over coils cooled by water, essentially short-circuiting the path between server exhaust and CRAC input. In some solutions, this additional cooling removes just part of the heat, thus lowering the load on the room's CRACs (i.e., lowering power density as seen by the CRACs), and in other solutions it completely removes all heat, effectively replacing the CRACs. The main downside of these approaches is that they all require chilled water to be brought to each rack, greatly increasing the cost of plumbing and the concerns over having water on the datacenter floor with couplings that might leak.

4.3.5 Container-Based Datacenters

Container-based datacenters go one step beyond in-rack cooling by placing the server racks into a standard shipping container and integrating heat exchange and power distribution into the container as well. Similar to full in-rack cooling, the container needs a supply of chilled water and uses coils to remove all heat from the air that flows over it. Air handling is similar to in-rack cooling and typically allows higher power densities than regular raised-floor datacenters. Thus, container-based datacenters provide all the functions of a typical datacenter room (racks, CRACs, PDU, cabling, lighting) in a small package. Like a regular datacenter room, they must be complemented by outside infrastructure such as chillers, generators, and UPS units to be fully functional.

It was recently unveiled that Google has built a container-based datacenter that has been in operation since 2005 [33], although the idea dates back to a Google patent application in 2003. The container-based facility described in [33] has achieved extremely high energy efficiency ratings compared with typical datacenters today, as we will discuss further in the following chapter. Microsoft has also announced that it will rely heavily on containers for new datacenters [92].

• • • •

CHAPTER 5

Energy and Power Efficiency

Energy efficiency has been a major technology driver in the mobile and embedded areas for some time but is a relatively new focus for general-purpose computing. Earlier work emphasized extending battery life but has since expanded to include reducing peak power because thermal constraints began to limit further CPU performance improvements. Energy management is now a key issue for servers and datacenter operations, focusing on the reduction of all energy-related costs including capital, operating expenses, and environmental impacts. Many energy-saving techniques developed for mobile devices are natural candidates for tackling this new problem space, but ultimately a warehouse-scale computer (WSC) is quite different from a mobile device. In this chapter, we describe some of the most relevant aspects of energy and power efficiency for WSCs, starting at the datacenter level and going all the way to component-level issues.

5.1 DATACENTER ENERGY EFFICIENCY

Energy efficiency of a WSC is broadly defined as the amount of computational work performed divided by the total energy used in the process. That is the notion that the Green Grid's Datacenter Performance Efficiency (DCPE) tries to capture [37,38]. No actual metric has been defined yet, but the general idea is to run a standardized datacenter-wide workload (say, a SPEC or TPC benchmark) and measure the total power consumed. Although this is a useful perspective, we are skeptical that DCPE will have much influence as a practical metric because it will be very hard to measure—few institutions will be able to ever measure it because the servers running in their datacenters are running actual applications and thus are not available for benchmarking.

Instead, we find it more useful to factor DCPE into three components that can be independently measured and optimized by the appropriate engineering disciplines, as shown in the equation below:

$$\text{Efficiency} = \frac{\text{Computation}}{\text{Total Energy}} = \underbrace{\left(\frac{1}{\text{PUE}}\right)}_{\text{(a)}} \times \underbrace{\left(\frac{1}{\text{SPUE}}\right)}_{\text{(b)}} \times \underbrace{\left(\frac{\text{Computation}}{\text{Total Energy to Electronic Components}}\right)}_{\text{(c)}}$$

EQUATION 5.1: Breaking an energy efficiency metric into three components: a facility term (a), a server energy conversion term (b), and the efficiency of the electronic components in performing the computation per se (c).

The first term in the efficiency calculation (a) is the power usage effectiveness (PUE), which reflects the quality of the datacenter building infrastructure itself [37], and captures the ratio of total building power to IT power, that is, the power consumed by the actual computing equipment (servers, network equipment, etc.). IT power is also referred to in the literature as critical power. Because PUE factors are oblivious to the nature of the computation being performed, they can be objectively and continuously measured by electrical monitoring equipment without any disruption to normal operations. Sadly, the PUE of the average datacenter is embarrassingly poor; according to a 2006 study [51], 85% of current datacenters were estimated to have a PUE of greater than 3.0, that is, the building's mechanical and electrical systems consume twice as much power as the actual computing load; only 5% have a PUE of 2.0. Other studies paint a somewhat better picture, showing an average PUE of approximately 2.0 for a set of 22 datacenters surveyed (Greenberg et al. [41] and Tschudi et al. [86]). Figure 5.1 shows PUE values from a recent update to Greenberg's survey (2007).

High PUE values are due to multiple sources of overhead (see Figure 5.2). In a typical raised-floor datacenter, chillers consume the largest fraction of power overhead, typically 30–50% of IT load. Computer room air conditioning (CRAC) units are next, consuming 10–30% of IT

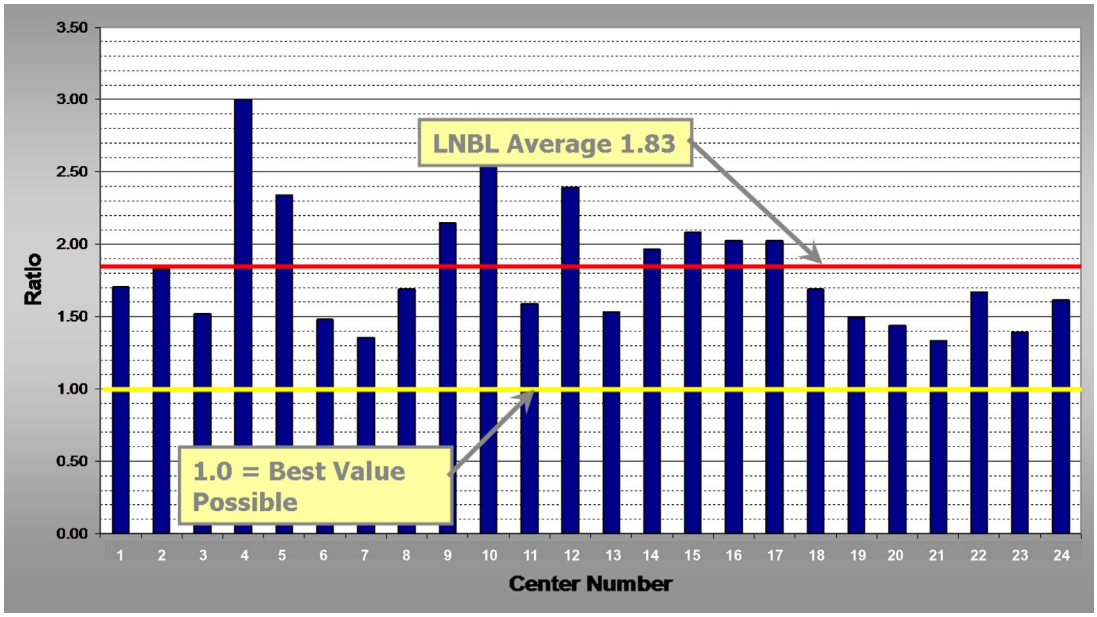

FIGURE 5.1: LBNL survey of the power usage efficiency of 24 datacenters, 2007 (Greenberg et al.) [41].

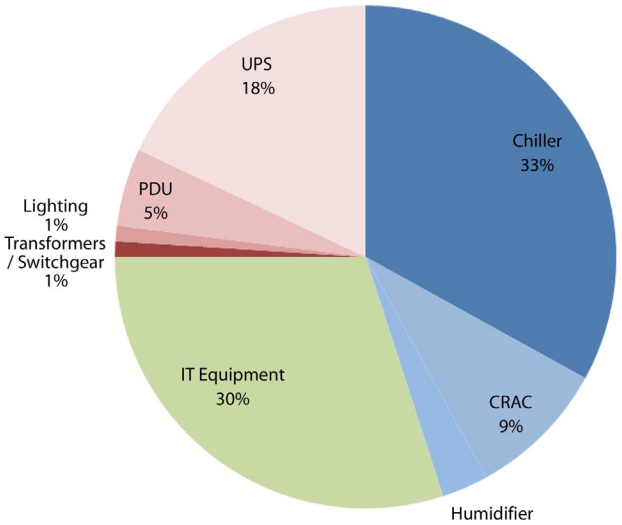

FIGURE 5.2: Breakdown of datacenter energy overheads (ASHRAE).

load (mostly in fans), followed by the UPS system, consuming 7–12% of critical power through AC–DC–AC conversion losses (relative losses are higher when the UPS is only lightly loaded). Other facility elements [humidifiers, power distribution units (PDUs), lighting] further contribute to higher PUE levels. Much of this poor efficiency is caused by a historical lack of attention to efficiency not by inherent limitations imposed by physics. It is commonly accepted that a well-designed and well-operated datacenter should have a PUE of less than 2, and the 2007 EPA report on datacenter power consumption states that in a "state-of-the-art" scenario a PUE of 1.4 is achievable by 2011 [26]. The most obvious improvements opportunities are the use of evaporative cooling towers, more efficient air movement, and the elimination of unnecessary power conversion losses.

5.1.1 Sources of Efficiency Losses in Datacenters

For illustration, let us walk through the sources of efficiency losses in a typical datacenter [41]. The transformers stepping down the incoming high-voltage power from 115 kV to the medium-voltage distribution lines (typically at 13.2 kV in the United States) are fairly efficient, and so are the transformers stepping it down further to 480 V. In both cases, transformation losses typically are below half a percentage. The uninterruptible power supply (UPS) is the source of most conversion losses, typically running at an efficiency of 88–94% in the best case (less, if they are lightly loaded). Rotary

UPS units (flywheels) and high-efficiency UPS units that avoid double conversion by bypassing the UPS in normal operation can reach efficiencies of about 97%. Finally, a fair amount of power can be lost bringing low-voltage power (110 or 220 V) to the racks if the respective cables are long. Recall that a large facility can have a raised floor area that is >100 m long or wide, and depending on distance and cable type, 1–3% of power can be lost in these cables.

Most of the efficiency losses, though, are encountered on the cooling side. Delivering cool air over long distances from CRACs to equipment racks consumes lots of fan power, as does moving warm air back to the CRAC intakes. Worse, during these long paths, cold and warm air may mix, greatly reducing the efficiency of the CRAC units by reducing the temperature delta available to CRACS [60]. Similarly, the common practice of keeping datacenters very cool requires chilled water temperatures near 10°C, increasing chiller load. Such low temperatures also lead to condensation on the coils of the CRAC units, further reducing their effectiveness and, ironically, requiring additional energy to be spent on rehumidification elsewhere.

5.1.2 Improving the Energy Efficiency of Datacenters

Careful design for efficiency can substantially improve PUE [57,66,39]. Although most datacenters run at PUEs of 2 or higher, much more efficient datacenters are possible. Some of the easiest steps to improve efficiency are to raise the cold aisle temperatures to 25–27°C instead of the traditional setting of 20°C. Virtually no server or network equipment actually needs intake temperatures of 20°C, and the fear of higher equipment failures with higher temperatures has little empirical evidence to back it up for these temperature ranges. Increasing the cold aisle temperature allows higher chilled water temperatures, improving the efficiency of chillers and reducing their runtime when combined with free cooling. Similarly, effective management of warm exhaust heat can greatly improve cooling efficiency; this is one of the primary reasons why container-based datacenters are more efficient. Finally, UPS and power distribution losses can often be greatly reduced by selecting higher-efficiency gear. In recent months, several companies announced datacenters with PUE values of less than 1.3 [57] and Google published data showing the average annual PUE of Google-designed datacenters at 1.2 [32].

In April of 2009, Google published details of its datacenter architecture, including a video tour of a container-based datacenter built in 2005 [33]. This datacenter achieved a state-of-the-art annual PUE of 1.24 in 2008 yet differs from conventional datacenters in only a few major aspects:

- Careful air flow handling: The hot air exhausted by servers is not allowed to mix with cold air, and the path to the cooling coil is very short so that little energy is spent moving cold or hot air long distances.

- Elevated cold aisle temperatures: The cold aisle of the containers is kept at about 27°C rather than 18–20°C. Higher temperatures make it much easier to cool datacenters efficiently.
- Use of free cooling: Several cooling towers dissipate heat by evaporating water, greatly reducing the need to run chillers. In most moderate climates, cooling towers can eliminate the majority of chiller runtime. Google's datacenter in Belgium even eliminates chillers altogether, running on "free" cooling 100% of the time.
- Per-server 12-V DC UPS: Each server contains a mini-UPS, essentially a battery that floats on the DC side of the server's power supply and is 99.99% efficient. These per-server UPSs eliminate the need for a facility-wide UPS, increasing the efficiency of the overall power infrastructure from around 90% to near 99% [34].

At first, it may seem that this datacenter is radically different from conventional ones, and in a sense that is true, but virtually all of its techniques can be applied to more conventional designs (e.g., no containers, no per-server UPS). Even in such a conventional-looking datacenter, these techniques should allow for a PUE between 1.35 and 1.45, i.e., an efficiency far above the current industry average.

Although PUE captures the facility overheads, it does not account for inefficiencies within the IT equipment itself. Servers and other computing equipment use less than 100% of their input power for actual computation. In particular, substantial amounts of power may be lost in the server's power supply, voltage regulator modules (VRMs), and cooling fans. The second term (b) in the efficiency calculation accounts for these overheads, using a metric analogous to PUE but instead applied to computing equipment: server PUE (SPUE). It consists of the ratio of total server input power to its useful power, where useful power includes only 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, useful power excludes all losses in power supplies, VRMs, and fans. No commonly used measurement protocol for SPUE exists today, although the Climate Savers Computing Initiative (climatesaverscomputing.org) is working on one. SPUE ratios of 1.6–1.8 are common in today's servers; many power supplies are less than 80% efficient, and many motherboards use VRMs that are similarly inefficient, losing more than 30% of input power in electrical conversion losses. In contrast, a state-of-the-art SPUE should be less than 1.2 [17].

The combination of PUE and SPUE therefore constitutes an accurate assessment of the joint efficiency of the total electromechanical overheads in facilities and computing equipment. Such true (or total) PUE metric (TPUE), defined as PUE * SPUE, stands at more than 3.2 for the average datacenter today; that is, for every productive watt, at least another 2.2 W are consumed! By contrast, a facility with a PUE of 1.2 and a 1.2 SPUE would use less than half as much energy. That is still not

ideal because only 70% of the energy delivered to the building is used for actual computation, but it is a large improvement over the status quo. Based on the current state of the art, an annual TPUE of 1.25 probably represents the upper limit of what is economically feasible in real-world settings.

5.2 MEASURING THE EFFICIENCY OF COMPUTING

So far we have discussed efficiency mostly in electrical terms, largely ignoring term (c) of Equation 5.1, which accounts for how much of the electricity delivered to electronic components in a system is actually translated into useful work. This last term is arguably the hardest one to measure objectively, given the general-purpose nature of computing systems. Ultimately, one wants to measure the amount of value obtained from the energy spent in computing. Such a measurement can be useful for comparing the relative efficiency of two WSCs or to guide the design choices for new systems. Unfortunately, no two companies run the same workload and real-world application mixes change all the time, so it is hard to benchmark using real-world data if the objective is to compare two WSCs. In the high-performance computing (HPC) area, there is a recent attempt to begin ranking the energy efficiency world's top supercomputers using existing HPC benchmarks (LINPACK) called the Green 500 [36]. We are not aware of a similar initiative for Internet services.

The lack of meaningful cluster-level benchmarks for WSCs does not prevent obtaining meaningful energy efficiency measurements. Existing installations can measure their own energy efficiency by load testing their services while adequately instrumenting the facility to measure or estimate energy usage. In the absence of standardized cluster-level benchmarks, server-level benchmarks can also be useful as long as meaningful extrapolations are possible. Two recently released benchmarks provide a good starting point for such energy efficiency accounting in servers: Joulesort [80] and SPECpower_ssj2008 benchmark [79]. Joulesort consists of measuring 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. SPECpower instead focuses on server-class systems and computes the performance-to-power ratio of a system running a typical business application on an enterprise Java platform. Similar energy efficiency benchmarks need to be developed for other important pieces of the computing infrastructure, such as networking switches and storage subsystems. The Storage Networking Industry Association is working on developing similar benchmarks for networked storage products [40].

5.2.1 Some Useful Benchmarks

Clearly, the same application binary can consume different amounts of power depending on the server's architecture and, similarly, an application can consume more or less of a server's capacity depending on software performance tuning. Benchmarks such as SPECpower_ssj2008 provide a standard application base that is representative of a broad class of server workloads, and it can help

us isolate efficiency differences in the hardware platform. In particular, SPEC power reporting rules help highlight a key energy usage feature of current servers: under low levels of utilization, computing systems tend to be significantly more inefficient than when they are exercised at maximum utilization. Unlike most performance benchmarks, SPEC power mandates that performance per watt be reported not only at peak utilization but across the whole utilization range (in intervals of 10%).

Figure 5.3 shows the SPEC power benchmark results for the top entry as of June 2008. The results show two metrics; performance (transactions per second)-to-power ratio and the average system power, plotted over 11 load levels. One feature in the figure is noteworthy and common to all other SPEC power benchmark results: the performance-to-power ratio drops dramatically as the target load decreases because the system power decreases much more slowly than does performance. Note, for example, that the energy efficiency at 30% load is less than half the efficiency at 100%. Moreover, when the system is idle, it is still consuming just under 175 W, which is over half of the peak power consumption of the server!

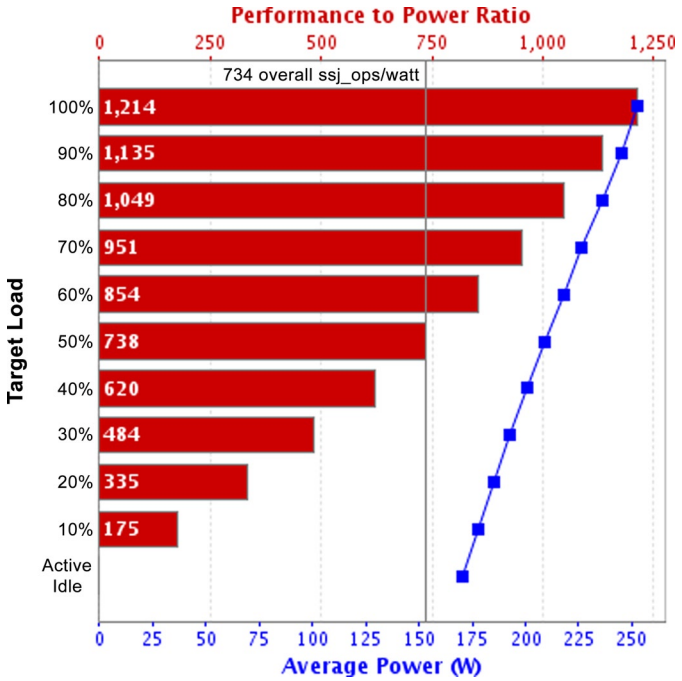

FIGURE 5.3: An example benchmark result for SPECpower_ssj2008; energy efficiency is indicated by bars, whereas power consumption is indicated by the line. Both are plotted for a range of utilization levels, with the average metric corresponding to the vertical dark line. The system has a single-chip 2.83 GHz quad-core Intel Xeon processor, 4 GB of DRAM, and one 7.2 k RPM 3.5″ SATA disk drive.

5.2.2 Load vs. Efficiency

Figure 5.4 shows the relative efficiency of the top 10 entries in the SPEC power benchmark when running at 30% load compared to the efficiency at 100%, as well as the ratio of idle vs. peak power—the rightmost value is the average of the 10 results. The behavior of the system in Figure 5.3 is common to modern server-class machines and is not unlike what we observe in Google servers. Such behavior would be acceptable if servers were, on average, running very close to peak load levels. Unfortunately, most datacenter operators report that this is not the case.

Figure 5.5 can be considered as an example load profile for WSCs. It shows the average CPU utilization of 5,000 Google servers during a 6-month period. Although the shape of the curve does vary across different clusters and workloads, a common trend is that, on average, servers spend relatively little aggregate time at high load levels. Instead, most of the time is spent within the 10–50% CPU utilization range. This activity profile turns out to be a perfect mismatch with the energy efficiency profile of modern servers in that they spend most of their time in the load region where they are most inefficient.

There is another feature of the energy usage profile of WSCs that is not as clearly visible in Figure 5.4; individual servers in these systems also spend little time completely idle. Consider, for example, a large Web search workload, such as the one described in Chapter 2, where queries are sent to a very large number of servers, each of which searches within its local slice of the entire in-

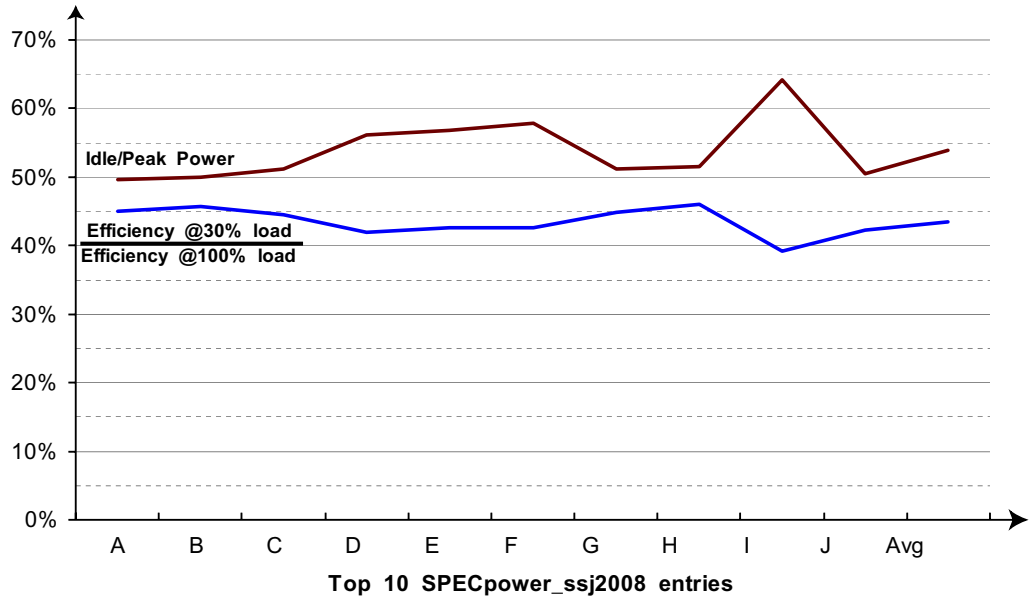

FIGURE 5.4: Idle/peak power and energy efficiency at 30% load (relative to the 100% load efficiency) of the top 10 SPECpower_ssj2008 entries. (data from mid 2008)

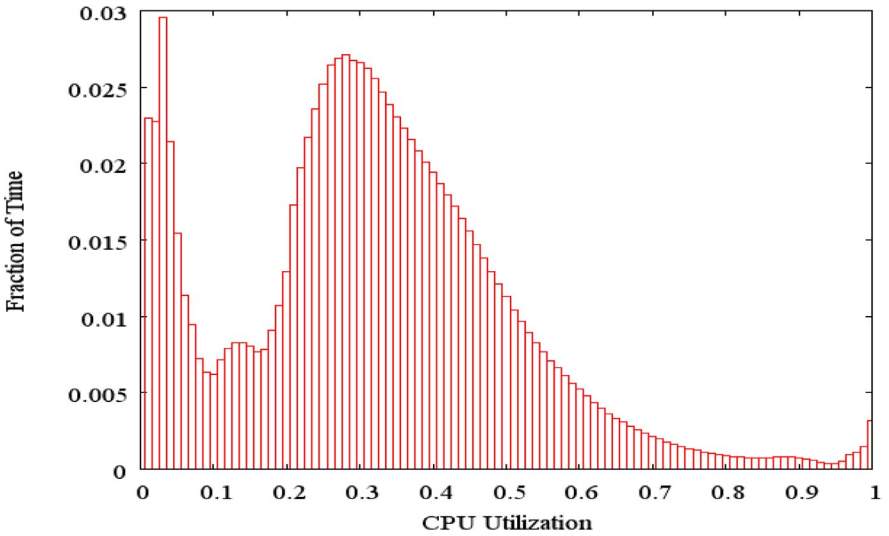

FIGURE 5.5: Activity profile of a sample of 5,000 Google servers over a period of 6 months.

dex. When search traffic is high, all servers are being heavily used, but during periods of low traffic, a server might still see hundreds of queries per second, meaning that any idleness periods are likely to be no longer than a few milliseconds.

The absence of significant idle intervals despite the existence of periods of low activity is largely a result of applying sound design principles to high-performance, robust distributed systems software. Large-scale Internet services rely on efficient load distribution to a large number of servers, creating a situation where when load is lighter we tend to have a lower load in multiple servers instead of concentrating the load in fewer servers and idling the remaining ones. Idleness can be manufactured by the application (or an underlying cluster management system) by migrating workloads and their corresponding state to fewer machines during periods of low activity. This can be relatively easy to accomplish when using simple replication models, when servers are mostly stateless (i.e., serving data that resides on a shared NAS or SAN storage system). However, it comes at a cost in terms of software complexity and energy for more complex data distribution models or those with significant state and aggressive exploitation of data locality.

Another reason why it may be difficult to manufacture useful idle periods in large-scale distributed systems is the need for resilient distributed storage. GFS [31], the Google File System, achieves higher resilience by distributing data chunk replicas for a given file across an entire cluster instead of concentrating them within only a small number of machines. This benefits file system performance by achieving fine granularity load balancing, as well as resiliency, because when a storage server

crashes (or a disk fails), the replicas in that system can be reconstructed by thousands of machines, making recovery extremely efficient. The consequence of such otherwise sound designs is that low traffic levels translate into lower activity for all machines instead of full idleness of a significant subset of them. Several practical considerations may also work against full idleness, as networked servers frequently perform many small background tasks on periodic intervals. The reports on the Tickless kernel project [78] provide other examples of how difficult it is to create and maintain idleness.

5.3 ENERGY-PROPORTIONAL COMPUTING

In an earlier article [6], we argued that the mismatch between server workload profile and server energy efficiency behavior must be addressed largely at the hardware level; software alone cannot efficiently exploit hardware systems that are efficient only when they are in inactive idle modes (sleep or stand-by) or when running at full speed. We believe that systems are inefficient when lightly used largely because of lack of awareness from engineers and researchers about the importance of that region to energy efficiency.

We suggest that energy proportionality should be added as a design goal for computing components. Ideally, energy-proportional systems will consume almost no power when idle (particularly in active idle states where they are still available to do work) and gradually consume more power as the activity level increases. A simple way to reason about this ideal curve is to assume linearity between activity and power usage, with no constant factors. Such linear relationship would make energy efficiency uniform across the activity range, instead of decaying with decreases in activity levels. Note, however, that linearity is not necessarily the optimal relationship for energy savings. Looking at Figure 5.5, it can be argued that because servers spend relatively little time at high activity levels, it might be fine even if efficiency decreases with increases in activity levels, particularly when approaching maximum utilization.

Figure 5.6 illustrates the possible energy efficiency of two hypothetical systems that are more energy-proportional than typical servers. The curves in red correspond to a typical server, such as the one in Figure 5.3. The green curves show the normalized power usage and energy efficiency of a more energy-proportional system, which idles at only 10% of peak power and with linear power vs. load behavior. Note how its efficiency curve is much superior to the one for the typical server; although its efficiency still decreases with the load level, it does so much less abruptly and remains at relatively high efficiency levels at 30% of peak load. The curves in blue show a system that also idles at 10% of peak but with a sublinear power vs. load relationship in the region of load levels between 0% and 50% of peak load. This system has an efficiency curve that peaks not at 100% load but around the 30–40% region. From an energy usage standpoint, such behavior would be a good match to the kind of activity spectrum for WSCs depicted in Figure 5.5.

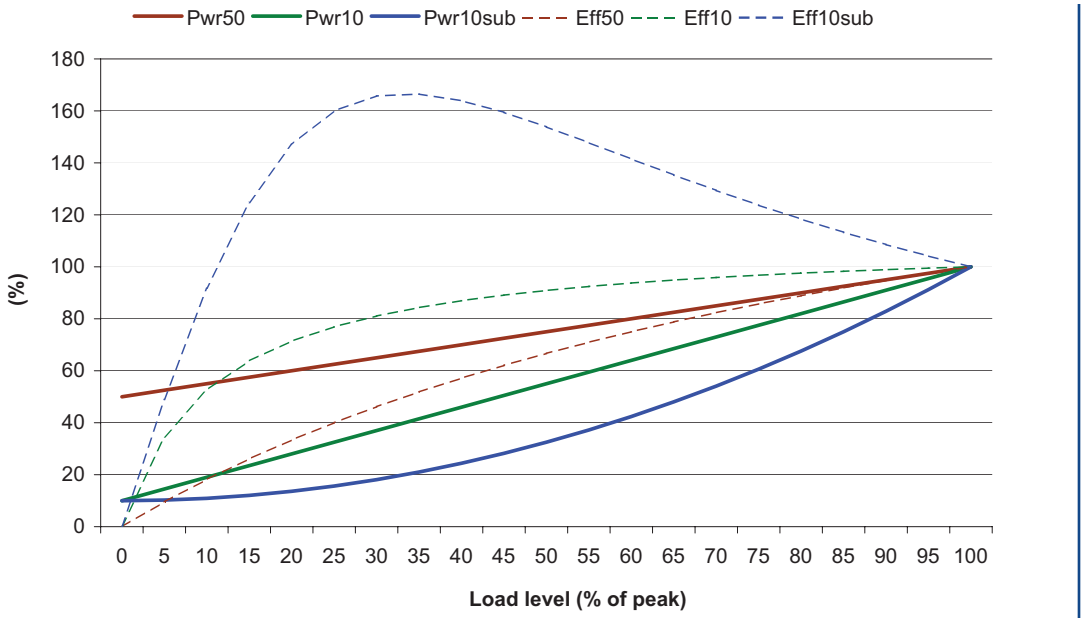

FIGURE 5.6: Power and corresponding power efficiency of three hypothetical systems: a typical server with idle power at 50% of peak (Pwr50 and Eff50), a more energy-proportional server with idle power at 10% of peak (Pwr10 and Eff10), and a sublinearly energy-proportional server with idle power at 10% of peak.

The potential gains from energy proportionality in WSCs were evaluated by Fan et al. [27] in their power provisioning study. They use traces of activity levels of thousands of machines over 6 months to simulate the energy savings gained from using more energy-proportional servers—servers with idle consumption at 10% of peak (similar to the green curves in Figure 5.6) instead of at 50% (such as the corresponding red curve). Their models suggest that energy usage would be halved through increased energy proportionality alone because the two servers compared had the same peak energy efficiency.

5.3.1 Dynamic Power Range of Energy-Proportional Machines

Energy-proportional machines would exhibit a wide dynamic power range—a property that is rare today in computing equipment but is not unprecedented in other domains. Humans, for example, have an average daily energy consumption approaching that of an old personal computer: about 120 W. However, humans at rest can consume as little as 70 W while being able to sustain peaks of well more than 1 kW for tens of minutes, with elite athletes reportedly approaching 2 kW [54]. Figure 5.7 lists

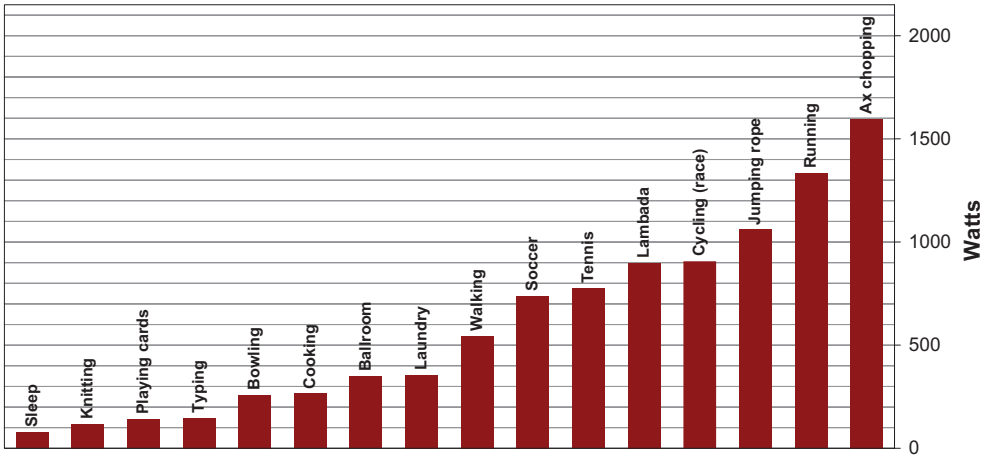

FIGURE 5.7: Human energy usage vs. activity levels (adult male) [54].

several occupational activities and their corresponding energy expenditures for adult males illustrating a dynamic power range of nearly 20× compared to the factor of 2× of today's typical computers. It is generally accepted that energy efficiency is one of the features favored by evolutionary processes; we wonder if energy proportionality might have been a successful means to achieve higher energy efficiency in organisms.

5.3.2 Causes of Poor Energy Proportionality

Although CPUs have a historically bad reputation regarding energy usage, they are not necessarily the main culprit for poor energy proportionality. For example, earlier Google servers spent as much as 60% of the total energy budget on CPU chips, whereas today they tend to use less than 50%. Over the last few years, CPU designers have paid more attention to energy efficiency than their counterparts for other subsystems. The switch to multicore architectures instead of continuing to push for higher clock frequencies and larger levels of speculative execution is one of the reasons for this more power-efficient trend.

Figure 5.8 shows the power usage of the main subsystems for a recent Google server as the compute load varies from idle to full activity levels. The CPU contribution to system power is nearly 50% when at peak but drops to less than 30% at low activity levels, making it the most energy-proportional of all main subsystems. In our experience, server-class CPUs have a dynamic power range that is generally greater than 3.0× (more than 3.5× in this case), whereas CPUs targeted at

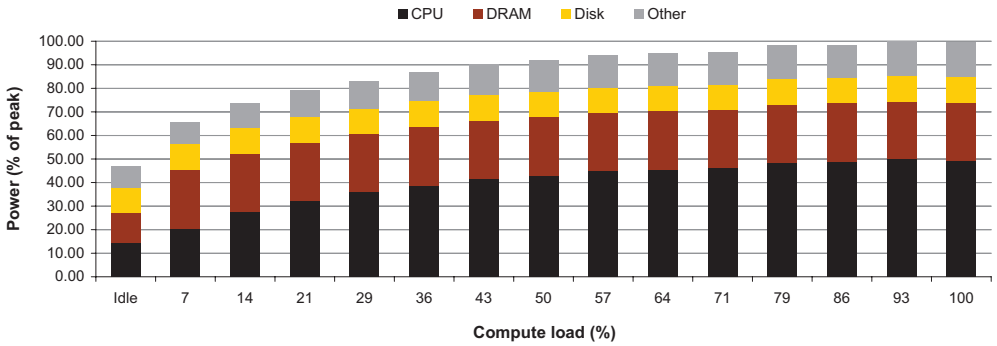

FIGURE 5.8: Subsystem power usage in an ×86 server as the compute load varies from idle to full usage.

the embedded or mobile markets can do even better. By comparison, the dynamic range of memory systems, disk drives, and networking equipment is much lower: approximately 2.0× for memory, 1.3× for disks, and less than 1.2× for networking switches. This suggests that energy proportionality at the system level cannot be achieved through CPU optimizations alone, but instead requires improvements across all components.

5.3.3 How to Improve Energy Proportionality

Added focus on energy proportionality as a figure of merit might create improvements across all system components simply through incremental engineering improvements. In some cases, however, greater innovation may be required. Disk drives, for example, spend a large fraction of their energy budget simply keeping the platters spinning, possibly as much as 70% of their total power for high RPM drives. Creating additional energy efficiency and energy proportionality may require smaller rotational speeds, smaller platters, or designs that use multiple independent head assemblies. Carrera et al. [11] considered the energy impact of multispeed drives and of combinations of server-class and laptop drives to achieve proportional energy behavior. More recently, Sankar et al. [81] have explored different architectures for disk drives, observing that because head movements are relatively energy-proportional, a disk with lower rotational speed and multiple heads might achieve similar performance and lower power when compared with a single-head, high RPM.

Finally, we should remind the readers that energy-proportional behavior is not only a target for electronic components but to the entire WSC system, including the power distribution and cooling infrastructure. Figure 5.9 shows efficiency curves derived from tests of several server power supplies, which also indicate lack of proportionality less than 30% of the peak-rated power. Because

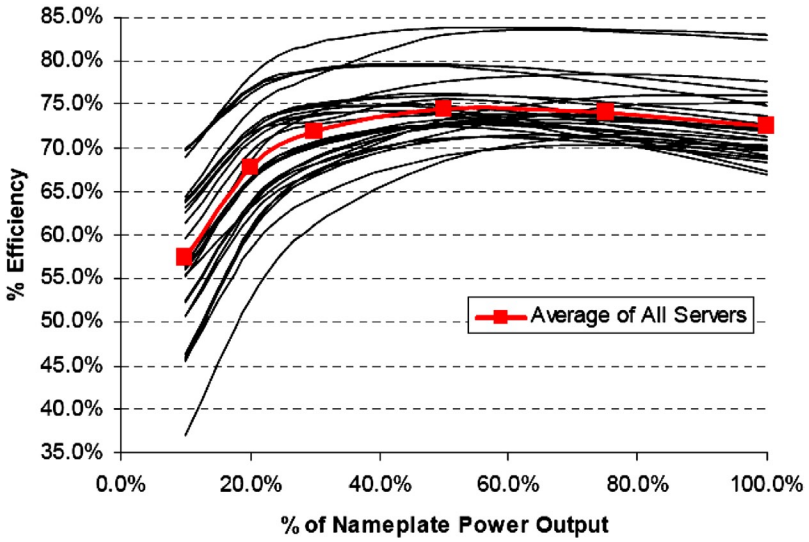

FIGURE 5.9: A survey of conversion efficiency of several server power supplies (by Ton and Fortenbury [84]).

power supplies tend to have significantly greater peak capacity than their corresponding computing equipment, it is not unusual for them to be operating much below their peak efficiency ratings).

5.4 RELATIVE EFFECTIVENESS OF LOW-POWER MODES

As discussed earlier, the existence of long idleness intervals would make it possible to achieve higher energy proportionality by using various kinds of sleep modes. We call these low-power modes *inactive* because the devices are not usable while in those modes, and typically a sizable latency and energy penalty is incurred when load is reapplied. Inactive low-power modes were originally developed for mobile and embedded devices, and they are very successful in that domain. However, most of those techniques are a poor fit for WSC systems, which would pay the inactive-to-active latency and energy penalty too frequently. The few techniques that can be successful in this domain are those with very low wake-up latencies, as is beginning to be the case with CPU low-power halt states (such as the ×86 C1E state).

Unfortunately, these tend to also be the low-power modes with the smallest degrees of energy savings. Large energy savings are available from inactive low-power modes such as spun-down disk drives. A spun-down disk might use almost no energy, but a transition to active mode incurs

a latency penalty 1,000 times higher than a regular access. Spinning up the disk platters adds an even larger energy penalty. Such a huge activation penalty restricts spin-down modes to situations in which the device will be idle for several minutes, which rarely occurs in servers.

Active low-power modes are those that save energy at a performance cost while not requiring inactivity. CPU voltage-frequency scaling is an example of an active low-power mode because it remains able to execute instructions albeit at a slower rate. The (presently unavailable) ability to read and write to disk drives at lower rotational speeds is another example of this class of low-power modes. In contrast with inactive modes, active modes are useful even when the latency and energy penalties to transition to a high-performance mode are significant. Because active modes are operational, systems can remain in low-energy states for as long as they remain below certain load thresholds. Given that periods of low activity are more common and longer than periods of full idleness, the overheads of transitioning between active energy savings modes amortize more effectively.

5.5 THE ROLE OF SOFTWARE IN ENERGY PROPORTIONALITY

We have argued that hardware components must undergo significant improvements in energy proportionality to enable more energy-efficient WSC systems. However, more intelligent power management and scheduling software infrastructure does play an important role in this area. For some component types, achieving perfect energy-proportional behavior may not be a realizable goal. Designers will have to implement software strategies for intelligent use of power management features in existing hardware, using low-overhead inactive or active low-power modes, as well as implementing power-friendly scheduling of tasks to enhance energy proportionality of hardware systems. For example, if the activation penalties in inactive low-power modes can be made small enough, techniques of the class described in the PowerNap article (Meisner et al. [55]) could be used to achieve energy-proportional behavior with components that only support inactive low-power modes.

This software layer must overcome two key challenges: encapsulation and performance robustness. Energy-aware mechanisms must be encapsulated in lower-level modules to minimize exposing additional infrastructure complexity to application developers; WSC application developers already deal with unprecedented scale and platform-level complexity. In large-scale systems, completion of an end-user task also tends to depend on large numbers of systems performing at adequate levels. If individual servers begin to exhibit excessive response time variability as a result of mechanisms for power management, the potential for service-level impact is fairly high and can result in the service requiring additional machine resources, resulting in little net improvements.

5.6 DATACENTER POWER PROVISIONING

Energy efficiency optimizations are naturally associated with smaller electricity costs. However, another energy-related cost factor is sometimes more significant than the electricity costs themselves: the cost of building a datacenter facility capable of providing a given level of power to a group of servers (also called the power provisioning costs). Using the Uptime Institute power provisioning cost range of $10–22 per deployed IT watt, and a typical 10-year depreciation cycle, the yearly costs of provisioning 1 watt of IT power falls in the $1.00–2.20 range.

By comparison, the corresponding electricity cost of 1 watt of IT power can be about $1.20, assuming the average cost of commercial electricity in the United States at just under $0.07 per kWh and a PUE factor of 2.0. One implication of this cost structure is that the electricity savings from investments in energy efficiency at the machine level can result in as much as twice those savings in power provisioning costs. Another consequence is the importance of maximizing the power provisioning budget built out in a facility. For example, if a facility operates at an average of 50% of its peak power capacity, provisioning cost per watt used is doubled.

Maximizing usage of the available power budget is also important for existing facilities because it can let the computing infrastructure grow or enable upgrades without requiring the acquisition of new datacenter capacity, which can take years if it involves new construction. The incentive to fully use the power budget of a datacenter is offset by the business risk of exceeding its maximum capacity, which could result in outages or costly violations of service agreements.

5.6.1 Deployment and Power Management Strategies

Determining the right deployment and power management strategies requires understanding the simultaneous power usage characteristics of groups of hundreds or thousands of machines over time. This is complicated by three important factors:

1. The rated maximum power (or nameplate value) of computing equipment is usually overly conservative and therefore of limited usefulness.
2. Actual power consumed by servers varies significantly with the amount of activity, making it hard to predict.
3. Different applications exercise large-scale systems differently.

Because of these factors, maximizing datacenter power usage involves three main steps:

1. Measure or otherwise accurately estimate the actual power usage range of all computing equipment. Major system vendors, such as Dell and HP, offer online power calculators [21] [46] that can be useful if measuring is not feasible.

2. Based on application resource usage profiling, determine opportunities for consolidating workloads in the smallest machine footprint that will still support peak usage. Cluster schedulers and virtual machine technology such as VMware [89] and Xen [4] can facilitate this process.
3. Understand the statistical properties of the simultaneous power usage of large groups of servers to reveal opportunities for deploying more computers by oversubscribing the facility power. Groups of servers are not always simultaneously running at their peak power levels. The larger the group of servers and the higher the application diversity, the less likely it is to find periods of simultaneous very high activity.

5.6.2 Advantages of Oversubscribing Facility Power

Oversubscribing the facility power consists of hosting a number of systems (and storage, networking, etc.) where the addition of their cumulative peak power will exceed the facility's maximum IT power budget. A successful implementation of power oversubscription will actually increase the overall utilization of the datacenter's energy budget and enable hosting of a larger number of servers while making overload situations extremely unlikely. We will expand on this issue because it is has received much less attention in technical publications than the first two steps listed above, and it is a very real problem in practice [53].

Fan et al. [27] have studied the potential opportunity of oversubscribing the facility power by analyzing the power usage behavior of clusters with up to 5,000 servers running various workloads at Google during a period of 6 months. One of their key results is summarized in Figure 5.10, which shows the cumulative distribution of power usage over time for groups of 80 servers (rack), 800 servers (PDU), and 5,000 servers (cluster).

Power is normalized to the peak aggregate power of the corresponding group. The figure shows, for example, that although rack units spend about 80% of their time using less than 65% of their peak power, they do reach 93% of their peak power at some point during the 6 months' observation window. For power provisioning, this indicates a very low oversubscription opportunity at the rack level because only 7% of the power available to the rack was stranded. However, with larger machine groups, the situation changes. In particular, the whole cluster never ran above 72% of its aggregate peak power. Thus, if we had allocated a power capacity to the cluster that corresponded to the sum of the peak power consumption of all machines, 28% of that power would have been stranded. This means that within that power capacity, we could have hosted nearly 40% more machines.

This study also evaluates the potential of more energy-proportional machines to reduce peak power consumption at the facility level. It suggests that lowering idle power from 50% to 10% of peak (i.e., going from the red to the green curve in Figure 5.6) can further reduce cluster peak power usage by more than 30%. This would be equivalent to an additional 40%+ increase in facility hosting capacity.

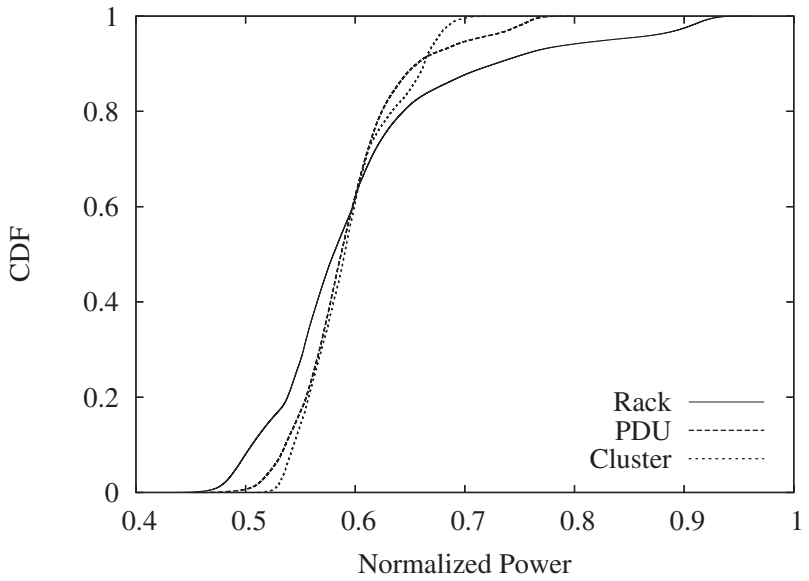

FIGURE 5.10: Cumulative distribution of the time that groups of machines spend at or below a given power level (power level is normalized to the maximum peak aggregate power for the corresponding grouping) (Fan et al. [27]).

The study also found that mixing different workloads within a cluster increased the opportunities for power oversubscription because this reduces the likelihood of synchronized power peaks across machines. Once oversubscription is used, the system needs a safety mechanism that can deal with the possibility that workload changes may cause the power draw to exceed the datacenter capacity. This can be accomplished by always allocating some fraction of the computing resources to a workload that runs in a lower priority class or that otherwise does not have strict deadlines to meet (many batch workloads may fall into that category). Such workloads can be quickly paused or aborted to reduce facility load. Provisioning should not be so aggressive as to require this mechanism to be triggered too often, which might be the case if oversubscription is applied at the rack level, for example.

In a real deployment, facility power can still be underused even after careful equipment power measurements, effective consolidation, and the deployment of an oversubscription strategy, due to other practical considerations. For example, a facility is rarely fully populated on initial commissioning but tends to be sized to accommodate business demand growth. Therefore, the gap between deployed and used power tends to be larger in new facilities. Facility power can also suffer from a form of fragmentation. Power usage can be left stranded simply because the addition of one more unit (a server, rack, or PDU) might exceed that level's limit. For example, a 2.5-kW circuit may

support only four 520-W servers, which would guarantee a 17% underutilization of that circuit. If a datacenter is designed such that the PDU-level peak capacity exactly matches the sum of the peak capacities of all of its circuits, such underutilization percolates up the power delivery chain, becoming truly wasted at the datacenter level.

5.7 TRENDS IN SERVER ENERGY USAGE

WSC designs, as with any machine design, attempt to achieve high energy efficiency but also to strike a balance between the capacity and performance of various subsystems that matches the expected resource requirements of the target class of workloads. Although this optimal point may vary over time for any given workload, the aggregate behavior of a wide set of workloads will tend to vary more slowly. Recognition of this behavior provides a useful input to system design. For example, an index search program may perform best for a given ratio of CPU speed to storage capacity: too much storage makes search too slow, and too little storage may underuse CPUs. If the desired ratio remains constant but the energy efficiency of various components evolve at a different pace, the energy consumption budget can shift significantly over time.

We have observed this kind of phenomenon over the past few years as CPU energy efficiency improvements have outpaced those of DRAM and disk storage. As a result, CPUs that used to account for more than 60% of the system energy budget are now often less than 45–50%. Consider the system used in Figure 5.8 as a balanced server design today as an example: two dual-core x86 CPUs at 2.4 GHz, 8 GB of DRAM, and two disk drives. Moore's law still enables creation of CPUs with nearly twice the computing speed every 18 months or so (through added cores at minor frequency changes) and at nearly the same power budget. But over the last few years, this pace has not been matched by DRAM and disk technology, and if this trend continues, the energy usage in a well-balanced server design might be dominated by the storage subsystem.

One consequence of such a trend is the decreasing utility of CPU voltage-frequency scaling for power management. Figure 5.11 shows the potential power savings of CPU dynamic voltage scaling (DVS) for that same server by plotting the power usage across a varying compute load for three frequency-voltage steps. Savings of approximately 10% are possible once the compute load is less than two thirds of peak by dropping to a frequency of 1.8 GHz (above that load level the application violates latency SLAs). An additional 10% savings is available when utilization drops further to one third by going to a frequency of 1 GHz. However, as the load continues to decline, the gains of DVS once again return to a maximum of 10% or so as a result of lack of energy proportionality at the system level. Although power savings in the 10–20% range is not negligible, they are not particularly impressive especially considering that most systems today have no other power management control knobs other than CPU DVS. Moreover, if the CPU continues to decrease in importance in overall system power, the impact of DVS is bound to decrease further.

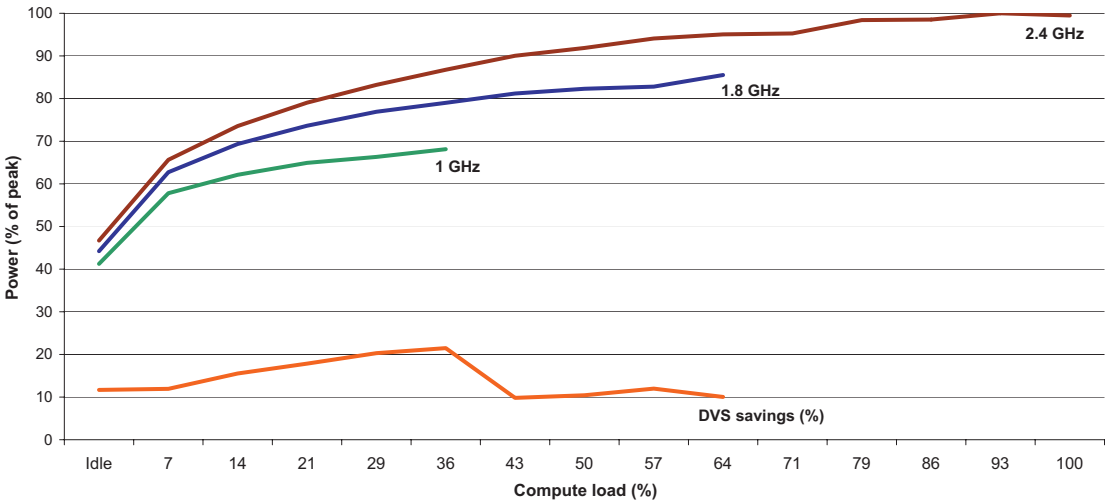

FIGURE 5.11: Power vs. compute load for an x86 server at three voltage-frequency levels and the corresponding energy savings.

5.8 CONCLUSIONS

Energy efficiency is a key cost driver for WSCs, and we expect energy usage to become an increasingly important factor of WSC design. The current state of the industry is poor: the average real-world datacenter and the average server are far too inefficient, mostly because efficiency has historically been neglected and has taken a backseat relative to reliability, performance, and capital expenditures. As a result, the average WSC wastes two thirds or more of its energy.

The upside of this history of neglect is that sizable improvements are almost trivial to obtain—an overall factor of two efficiency improvements is possible without much risk by simply applying best practices to datacenter and server designs. Unfortunately, the path beyond this low-hanging fruit is more difficult, posing substantial challenges to overcome inherently complex problems and often unfavorable technology trends. Once the average datacenter achieves state-of-the-art PUE levels, and servers are deployed with high-efficiency power supplies that are already available today, the opportunity for further efficiency improvements in those areas is less than 40%. From a research and development standpoint, the greater opportunities for gains from now on will need to come from computer scientists and engineers, and less so from mechanical or power conversion specialists.

First, power and energy must be better managed to minimize operational cost. Power determines overall facility cost because much of the construction cost is directly related to the maximum power draw that must be supported. Overall energy usage determines the electricity bill as well as much of the environmental impact. Today's servers can have high maximum power draws that are rarely reached in practice but that must be accommodated or controlled (throttled) to avoid over-

loading the facility's power delivery system. Power capping is a promising technique to manage the aggregate power of a pool of servers, but it is difficult to reconcile with availability, that is, the need to use peak processing power in an emergency caused by a sudden spike in traffic or by a failure in a different cluster. In addition, peak server power is increasing despite the continuing shrinking of silicon gate sizes, driven by a combination of increasing operating frequencies, larger cache and memory sizes, and faster off-chip communication (DRAM and I/O buses as well as networking speeds).

Second, today's hardware does not gracefully adapt its power usage to changing load conditions, and as a result, a server's efficiency degrades seriously under light load. Energy proportionality promises a way out of this dilemma but may be challenging to implement across all subsystems—for example, disks do not naturally lend themselves to lower-power active states. Systems for work consolidation that free up and power down entire servers present an avenue to create energy-proportional behavior in clusters built with non-energy-proportional components but are harder to implement and manage, requiring transparent process migration and degrading the WSC's ability to react to sudden upticks in load. Furthermore, high-performance and high-availability distributed systems software tends to spread data and computation in a way that reduces the availability of sufficiently large idle periods at any one system. Energy management-aware software layers must then manufacture idleness in a way that minimizes the impact on performance and availability.

Finally, energy optimization is a complex end-to-end problem, requiring an intricate coordination across hardware, operating systems, virtual machines, middleware, applications, and operations organizations. Even small mistakes can ruin energy savings, for example, when a suboptimal device driver generates too many interrupts or when network chatter from neighboring machines keeps a machine from quiescing. There are too many components involved for perfect coordination to happen naturally, and we currently lack the right abstractions to manage this complexity. In contrast to hardware improvements such as energy-proportional components that can be developed in relative isolation, solving this end-to-end problem at scale will be much more difficult.

5.8.1 Further Reading

Management of energy, peak power, and temperature of WSCs are becoming the targets of an increasing number of research studies. Chase et al. [14], G. Chen et al. [15], and Y. Chen et al. [16] consider schemes for automatically provisioning resources in datacenters taking energy savings and application performance into account. Raghavendra et al. [69] describe a comprehensive framework for power management in datacenters that coordinates hardware-level power capping with virtual machine dispatching mechanisms through the use of a control theory approach. Femal and Freeh [28,29] focus specifically on the issue of datacenter power oversubscription and on dynamic voltage-frequency scaling as the mechanism to reduce peak power consumption. Finally, managing temperature is the subject of the systems proposed by Heath et al. [45] and Moore et al. [56].

• • • •

CHAPTER 6

Modeling Costs

To better understand the potential impact of energy-related optimizations, let us examine the total cost of ownership (TCO) of a datacenter. At the top level, costs split up into capital expenses (Capex) and operational expenses (Opex). *Capex* refers to investments that must be made upfront and that are then depreciated over a certain time frame; examples are the construction cost of a datacenter or the purchase price of a server. *Opex* refers to the recurring monthly costs of actually running the equipment; electricity costs, repairs and maintenance, salaries of on-site personnel, and so on. Thus, we have:

TCO = datacenter depreciation + datacenter Opex + server depreciation + server Opex

We focus on top-line estimates in this chapter, simplifying the models where appropriate. More detailed cost models can be found in the literature [61,50]. For academic purposes, our simplified model is accurate enough to model all major costs; the primary source of inaccuracy compared to real-world datacenters will be the model input values such as the cost of construction.

6.1 CAPITAL COSTS

Datacenter construction costs vary widely depending on design, size, location, and desired speed of construction. Not surprisingly, adding reliability and redundancy makes datacenters more expensive, and very small or very large datacenters tend to be more expensive (the former because fixed costs cannot be amortized over many watts and the latter because large centers require additional infrastructure such as electrical substations). Table 6.1 shows a range of typical datacenter construction costs, expressed in dollars per watt of usable critical power, drawn from a variety of sources. As a rule of thumb, most large datacenters probably cost around $12–15/W to build and smaller ones cost more.

Characterizing costs in terms of dollars per watt makes sense for larger datacenters (where size-independent fixed costs are a relatively small fraction of overall cost) because all of the datacenter's primary components—power, cooling, and space—roughly scale linearly with watts. Typically, approximately 80% of total construction cost goes toward power and cooling, and the remaining 20% toward the general building and site construction.

TABLE 6.1: Range of datacenter construction costs expressed in U.S. dollars per watt of critical power.

COST/W	SOURCE
$12–25	Uptime Institute estimates for small- to medium-sized datacenters; the lower value is for "Tier 1" designs that are rarely used in practice [68]
$10+	Microsoft's purchase of two 10-MW datacenters in California for $200 million; this cost excludes the value of land and buildings [74]
$10–14	Dupont Fabros S-1 filing [25] (p. 6) contains financial information suggesting the following costs: $102.4 million for the purchase of ACC4, an existing 9.6 MW facility ($10.67/W) $180–300 million for the construction of four 18.2 MW facilities ($10/W—$13.40) The earnings release for Q3 2008 [70] lists the construction cost of a completed 18.2-MW Chicago facility as $190 million, or $10.44/W.

Critical power is defined as the peak power level that can be provisioned to IT equipment.

Cost varies with the degree of desired redundancy and availability, and thus we always express cost in terms of dollar per critical watt, that is, per watt that can actually be used by IT equipment. For example, a datacenter with 20 MW of generators may have been built in a 2N configuration and provide only 6 MW of critical power (plus 4 MW to power chillers). Thus, if built for $120 million, it has a cost of $20/W, not $6/W. Industry reports often do not correctly use the term *critical power*; so our example datacenter might be described as a 20-MW datacenter or even as a 30-MW datacenter if it is supplied by an electrical substation that can provide 30 MW. Frequently, cost is quoted in dollars per square ft, but that metric is less useful because it cannot adequately compare projects and is used even more inconsistently than costs expressed in dollars per watt. In particular, there is no standard definition of what space to include or exclude in the computation, and the metric does not correlate well with the primary cost driver of datacenter construction, namely, critical power. Thus, most industry experts avoid using dollars per square feet to express cost.

The monthly depreciation cost (or amortization cost) that results from the initial construction expense depends on the duration over which the investment is amortized (which is related to its expected lifetime) and the assumed interest rate. Typically, datacenters are depreciated over periods of 10–15 years. Under U.S. accounting rules, it is common to use straight-line depreciation where

the value of the asset declines by a fixed amount each month. For example, if we depreciate a $15/W datacenter over 12 years, the depreciation cost is $0.10/W per month. If we had to take out a loan to finance construction at an interest rate of 8%, the associated monthly interest payments add an additional cost of $0.06/W, for a total of $0.16/W per month. Typical interest rates vary over time, but many companies will pay interest in the 10–13% range.

Server costs are computed similarly, except that servers have a shorter lifetime and thus are typically depreciated over 3–4 years. To normalize server and datacenter costs, it is useful to characterize server costs per watt as well, using the server's peak real-life power consumption as the denominator. For example, a $4,000 server with an actual peak power consumption of 500 W costs $8/W. Depreciated over 4 years, the server costs $0.17/W per month. Financing that server at 8% annual interest adds another $0.03/W per month, for a total of $0.20/W per month, about the same cost per watt as the datacenter.

6.2 OPERATIONAL COSTS

Datacenter Opex is harder to characterize because it depends heavily on operational standards (e.g., how many security guards are on duty at the same time or how often generators are tested and serviced) as well as on the datacenter's size (larger datacenters are cheaper because fixed costs are amortized better). Costs can also vary depending on geographic location (climate, taxes, salary levels, etc.) and on the datacenters design and age. For simplicity, we will break operational cost into a monthly charge per watt that represents items like security guards and maintenance, and electricity. Typical operational costs for multi-MW datacenters in the United States range from $0.02 to $0.08/W per month, excluding the actual electricity costs.

Similarly, servers have an operational cost. Because we are focusing just on the cost of running the infrastructure itself, we will focus on just hardware maintenance and repairs, as well as on electricity costs. Server maintenance costs vary greatly depending on server type and maintenance standards (e.g., four-hour response time vs. two business days).

Also, in traditional IT environments, the bulk of the operational cost lies in the applications, that is, software licenses and the cost of system administrators, database administrators, network engineers, and so on. We are excluding these costs here because we are focusing on the cost of running the physical infrastructure but also because application costs vary greatly depending on the situation. In small corporate environments, it is not unusual to see one system administrator per a few tens of servers, resulting in a substantial per-machine annual cost [73]. Many published studies attempt to quantify administration costs, but most of them are financed by vendors trying to prove the cost-effectiveness of their products, so that reliable unbiased information is scarce. However, it is commonly assumed that large-scale applications require less administration, scaling to perhaps 1,000 servers per administrator.

6.3 CASE STUDIES

Given the large number of variables involved, it is best to illustrate the range of cost factors by looking at a small number of case studies that represent different kinds of deployments. First, we consider a typical new multi-megawatt datacenter in the United States (something closer to the Uptime Institute's Tier 3 classification), fully populated with servers at the high end of what can still be considered a volume rack-mountable server product in 2008. For this example we chose a Dell 2950 III EnergySmart with 16 GB of RAM and four disks, which draws 300 watts at peak per Dell's datacenter capacity planning tool and costs approximately $6,000 (list price) as of 2008. The remaining base case parameters were chosen as follows:

- The cost of electricity is the 2006 average U.S. industrial rate of 6.2 cents/kWh.
- The interest rate a business must pay on loans is 12%, and we finance the servers with a 3-year interest-only loan.
- The cost of datacenter construction is $15/W amortized over 12 years.
- Datacenter Opex is $0.04/W per month.
- The datacenter has a power usage effectiveness (PUE) of 2.0.
- Server lifetime is 3 years, and server repair and maintenance is 5% of Capex per year.
- The server's average power draw is 75% of peak power.

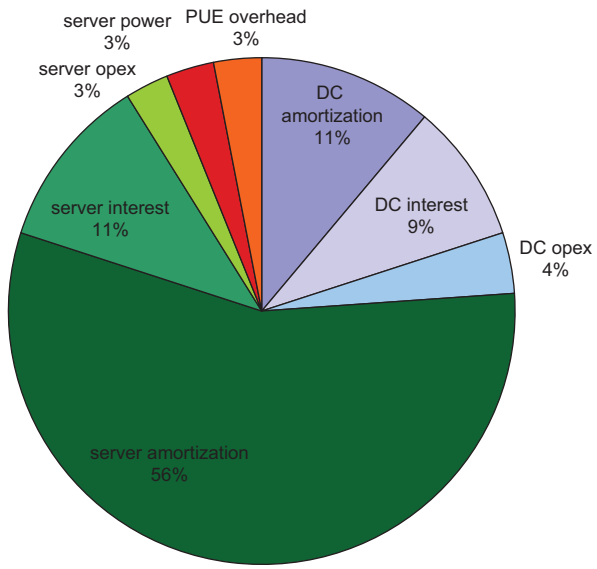

FIGURE 6.1: TCO cost breakdown for case study A.

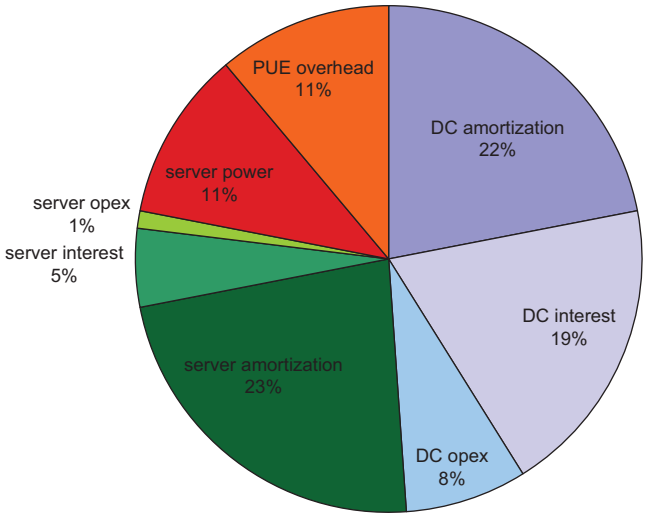

FIGURE 6.2: TCO cost breakdown for case study B (lower-cost, higher-power servers).

Figure 6.1 shows a breakdown of the yearly TCO for case A among datacenter and server-related Opex and Capex components.[1]

In this example, which is typical of classical datacenters, the high server capital costs dominate overall TCO, with 69% of the monthly cost related to server purchase and maintenance. However, commodity-based lower-cost (and perhaps lower-reliability) servers, or higher power prices, can change the picture quite dramatically. For case B (see Figure 6.2), we assume a cheaper, faster, higher-powered server consuming 500 W at peak and costing only $2,000 in a location where electricity cost is $0.10/kWh. In this case, datacenter-related costs rise to 49% of the total, and energy costs to 22%, with server costs falling to 29%. In other words, the hosting cost of such a server, that is, the cost of all infrastructure and power to house it, is more than twice the cost of purchasing and maintaining the server in this scenario.

Note that even with the assumed higher power price and higher server power, the absolute 3-year TCO in case B is lower than in case A ($8,702 vs $10,757) because the server is so much cheaper. The relative importance of power-related costs is likely to increase as shown in case B

[1] An online version of the spreadsheet underlying the graphs in this section is at http://spreadsheets.google.com/pub?key=phRJ4tNx2bFOHgYskgpoXAA&output=xls.

because the power consumption (and performance) of CPUs has increased by eight times between 1995 and 2007 or approximately 19% annually over the past 12 years [82], whereas the sale price of low-end servers have stayed relatively stable. As a result, the dollars per watt cost of server hardware is trending down, whereas electricity and construction costs are trending up. The result of these trends is clear: increasingly, the total cost of a server will be primarily a function of the power it consumes, and the server's purchase price will matter less. In other words, over the long term, the datacenter facility costs (which are proportional to power consumption) will become a larger and larger fraction of total cost.

We are not predicting that the future will indeed play out like this—but it will unless things change.

6.3.1 Real-World Datacenter Costs

In fact, real-world datacenter costs are even higher than modeled so far. All of the models presented so far assume that the datacenter is 100% full and that the servers are fairly busy (75% of peak power corresponds to a CPU utilization of approximately 50%; see Chapter 5). In reality, this often is not the case. For example, because datacenter space takes a while to build, we may want to keep a certain amount of empty space to accommodate future deployments. In addition, server layouts assume overly high (worst case) power consumption. For example, a server may consume "up to" 500 W with all options installed (maximum memory, disk, PCI cards, etc.), but the actual configuration being deployed may only use 300 W. If the server layout assumes the "name plate" rating of 500 W, we will only reach a utilization factor of 60% and thus the actual datacenter costs per server increase by 1.66×. Thus, in reality, actual monthly costs per server often are considerably higher than shown above because the datacenter-related costs increase inversely proportional to datacenter power utilization.

As discussed in Chapter 5, reaching high datacenter power utilization is not as simple as it may seem. Even if the vendor provides a power calculator to compute the actual maximum power draw for a particular configuration, that value will assume 100% CPU utilization. If we install servers based on that value and they run at only 30% CPU utilization on average (consuming 200 W instead of 300 W), we just left 30% of the datacenter capacity stranded. On the other hand, if we install based on the average value of 200 W and at month's end the servers actually run at near full capacity for a while, our datacenter will overheat or trip a breaker. Similarly, we may choose to add additional RAM or disks to servers at a later time, which would require physical decompaction of server racks if we left no slack in our power consumption calculations. Thus, in practice, datacenter operators leave a fair amount of slack space to guard against these problems. Reserves of 20–50% are

common, which means that real-world datacenters rarely run at anywhere near their rated capacity. In other words, a datacenter with 10 MW of critical power will often consume a monthly average of just 4–6 MW of actual critical power (plus PUE overhead).

6.3.2 Modeling a Partially Filled Datacenter

To model a partially filled datacenter, we simply scale the datacenter Capex and Opex costs (excluding power) by the inverse of the occupancy factor (Figure 6.3). For example, a datacenter that is only two thirds full has a 50% higher Opex. Taking case B above but with a 50% occupancy factor, datacenter costs completely dominate the cost (see Figure 6.2), with only 19% of total cost related to the server. Given the need for slack power just discussed, this case is not as far-fetched as it may sound. Thus, improving actual datacenter usage (e.g., using power capping) can substantially reduce real-world datacenter costs. In absolute dollars, the server TCO in a perfectly full datacenter is $8,702 vs. $12,968 in a half-full datacenter—all that for a server that costs just $2,000 to purchase.

Partially used servers also affect operational costs in a positive way because the server is using less power. Of course, these savings are questionable because the applications running on those servers are likely to produce less value. Our TCO model cannot capture this effect because it is based on the cost of the physical infrastructure only and excludes the application that is running on this

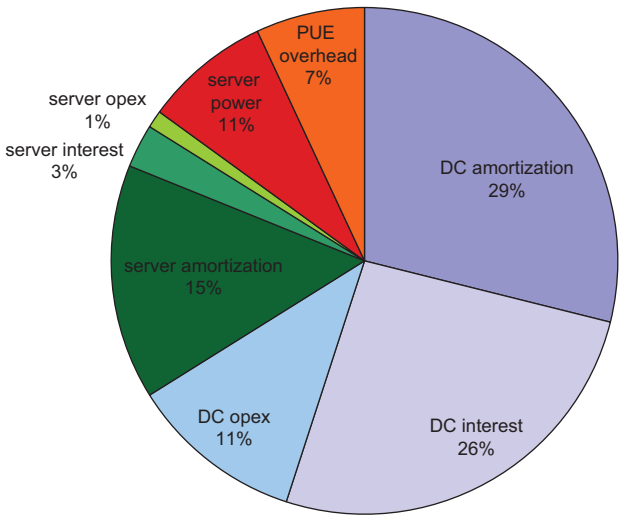

FIGURE 6.3: TCO case study C (partly filled facility).

hardware. To measure this end-to-end performance, we can measure a proxy for application value (e.g., the number of bank transactions completed or the number of Web searches) and divide the TCO by that. For example, if we had a datacenter costing $1 million per month and completing 100 million transactions per month, the cost per transaction would be 1 cent. If, on the other hand, traffic is lower at one month and we complete only 50 million transactions, the cost per transaction doubles to 2 cents. In this chapter, we have focused exclusively on hardware costs, but it is important to keep in mind that, ultimately, software performance and server utilization matter just as much.

• • • •

CHAPTER 7

Dealing with Failures and Repairs

The promise of Web-based, service-oriented computing will be fully realized only if users can trust that the services in which they increasingly rely will be always available to them. This availability expectation translates into a high-reliability requirement for building-sized computers. Determining the appropriate level of reliability is fundamentally a trade-off between the cost of failures (including repairs) and the cost of preventing them. For traditional servers the cost of failures is thought to be very high, and thus designers go to great lengths to provide more reliable hardware by adding redundant power supplies, fans, error correction coding (ECC), RAID disks, and so on. Many legacy enterprise applications were not designed to survive frequent hardware faults, and it is hard to make them fault-tolerant after the fact. Under these circumstances, making the hardware very reliable becomes a justifiable alternative.

In warehouse-scale computers (WSCs), however, hardware cannot easily be made "reliable enough" as a result of the scale. Suppose a cluster has ultra-reliable server nodes with a stellar mean time between failures (MTBF) of 30 years (10,000 days)—well beyond what is typically possible to achieve at a realistic cost. Even with these ideally reliable servers, a cluster of 10,000 servers will see an average of one server failure per day. Thus, any application that needs the entire cluster to be up to work will see an MTBF no better than 1 day. In reality, typical servers see an MTBF substantially less than 30 years, and thus the real-life cluster MTBF would be in the range of a few hours between failures. Moreover, large and complex Internet services are often composed of several software modules or layers that are not bug-free and can themselves fail at even higher rates than hardware components. Consequently, WSC applications must work around failed servers in software, either with code in the application itself or via functionality provided via middleware such as a provisioning system for virtual machines that restarts a failed VM on a spare node. Some of the implications of writing software for this environment are discussed by Hamilton [43], based on experience on designing and operating some of the largest services at MSN and Windows Live.

7.1 IMPLICATIONS OF SOFTWARE-BASED FAULT TOLERANCE

The inevitability of failures in WSCs makes fault-tolerant software inherently more complex than software that can assume fault-free operation. As much as possible, one should try to implement

a fault-tolerant software infrastructure layer that can hide much of this failure complexity from application-level software.

There are some positive consequences of adopting such a model, though. Once hardware faults can be tolerated without undue disruption to a service, computer architects have some leeway to choose the level of hardware reliability that maximizes overall system cost efficiency. This leeway enables consideration, for instance, of using inexpensive PC-class hardware for a server platform instead of mainframe-class computers, as discussed in Chapter 3. In addition, this model can lead to simplifications in common operational procedures. For example, to upgrade system software in a cluster, you can load a newer version in the background (i.e., during normal operation), kill the older version, and immediately start the newer one. Hardware upgrades can also follow a similar procedure. Basically, the same fault-tolerant software infrastructure mechanisms built to handle server failures could have all the required mechanisms to support a broad class of operational procedures. By choosing opportune time windows and rate-limiting the pace of kill–restart actions, operators can still manage the desired amount of planned service-level disruptions.

The basic property being exploited here is that, unlike in traditional server setups, it is no longer necessary to keep a server running at all costs. This simple requirement shift affects almost every aspect of the deployment, from machine/datacenter design to operations, often enabling optimization opportunities that would not be on the table otherwise. For instance, let us examine how this affects the recovery model. A system that needs to be highly reliable in the presence of unavoidable transient hardware faults, such as uncorrectable errors caused by cosmic particle strikes, may require hardware support for checkpoint recovery so that upon detection the execution can be restarted from an earlier correct state. A system that is allowed to go down upon occurrence of such faults may choose not to incur the extra overhead in cost or energy of checkpointing.

Another useful example involves the design trade-offs for a reliable storage system. One alternative is to build highly reliable storage nodes through the use of multiple disk drives in a mirrored or RAIDed configuration so that a number of disk errors can be corrected on the fly. Drive redundancy increases reliability but by itself does not guarantee that the storage server will be always up. Many other single points of failure also need to be attacked (power supplies, operating system software, etc.), and dealing with all of them incurs extra cost while never assuring fault-free operation. Alternatively, data can be mirrored or RAIDed across disk drives that reside in multiple machines—the approach chosen by Google's GFS [31]. This option tolerates not only drive failures but also entire storage server crashes because other replicas of each piece of data are accessible through other servers. It also has different performance characteristics from the centralized storage server scenario. Data updates may incur higher networking overheads because they require communicating with multiple systems to update all replicas, but aggregate read bandwidth can be greatly increased because clients can source data from multiple end points (in the case of full replication).

In a system that can tolerate a number of failures at the software level, the minimum requirement made to the hardware layer is that its faults are always detected and reported to software in a timely enough manner as to allow the software infrastructure to contain it and take appropriate recovery actions. It is not necessarily required that hardware transparently corrects all faults. This does not mean that hardware for such systems should be designed without error correction capabilities. Whenever error correction functionality can be offered within a reasonable cost or complexity, it often pays to support it. It means that if hardware error correction would be exceedingly expensive, the system would have the option of using a less expensive version that provided detection capabilities only. Modern DRAM systems are a good example of a case in which powerful error correction can be provided at a very low additional cost.

Relaxing the requirement that hardware errors be detected, however, would be much more difficult because it means that every software component would be burdened with the need to check its own correct execution. At one early point in its history, Google had to deal with servers that had DRAM lacking even parity checking. Producing a Web search index consists essentially of a very large shuffle/merge sort operation, using several machines over a long period. In 2000, one of the then monthly updates to Google's Web index failed prerelease checks when a subset of tested queries was found to return seemingly random documents. After some investigation a pattern was found in the new index files that corresponded to a bit being stuck at zero at a consistent place in the data structures; a bad side effect of streaming a lot of data through a faulty DRAM chip. Consistency checks were added to the index data structures to minimize the likelihood of this problem recurring, and no further problems of this nature were reported. Note, however, that this workaround did not guarantee 100% error detection in the indexing pass because not all memory positions were being checked—instructions, for example, were not. It worked because index data structures were so much larger than all other data involved in the computation, that having those self-checking data structures made it very likely that machines with defective DRAM would be identified and excluded from the cluster. The following machine generation at Google did include memory parity detection, and once the price of memory with ECC dropped to competitive levels, all subsequent generations have used ECC DRAM.

7.2 CATEGORIZING FAULTS

An efficient fault-tolerant software layer must be based on a set of expectations regarding fault sources, their statistical characteristics, and the corresponding recovery behavior. Software developed in the absence of such expectations risks being prone to outages if the underlying faults are underestimated or requiring excessive overprovisioning if faults are assumed to be much more frequent than in real life.

Providing an accurate quantitative assessment of faults in WSC systems would be challenging given the diversity of equipment and software infrastructure across different deployments. Instead, we will attempt to summarize the high-level trends from publicly available sources and from our own experience.

7.2.1 Fault Severity

Hardware or software faults can affect Internet services in varying degrees, resulting in different service-level failure modes. The most severe modes may demand very high reliability levels, whereas the least damaging ones might have more relaxed requirements that can be achieved with less expensive solutions. We broadly classify service-level failures into the following categories, listed in decreasing degree of severity:

- Corrupted: committed data that are impossible to regenerate, are lost, or corrupted
- Unreachable: service is down or otherwise unreachable by the users
- Degraded: service is available but in some degraded mode
- Masked: faults occur but are completely hidden from users by the fault-tolerant software/hardware mechanisms.

Acceptable levels of robustness will differ across those categories. We expect most faults to be masked by a well-designed fault-tolerant infrastructure so that they are effectively invisible outside of the service provider. It is possible that masked faults will impact the service's maximum sustainable throughput capacity, but a careful degree of overprovisioning can ensure that the service remains healthy.

If faults cannot be completely masked, their least severe manifestation is one in which there is some degradation in the quality of service. Here, different services can introduce degraded availability in different ways. One example of such classes of failures proposed by Brewer [10] is when a Web Search system uses data partitioning techniques to improve throughput but loses some of the systems that serve parts of the database. Search query results will be imperfect but probably still acceptable in many cases. Graceful degradation as a result of faults can also manifest itself by decreased freshness. For example, a user may access his or her email account, but new email delivery is delayed by a few minutes, or some fragments of the mailbox could be temporarily missing. Although these kinds of faults also need to be minimized, they are less severe than complete unavailability. Internet services need to be deliberately designed to take advantage of such opportunities for gracefully degraded service. In other words, this support is often very application-specific and not something easily hidden within layers of cluster infrastructure software.

Service availability/reachability is very important, especially because Internet service revenue is often related in some way to traffic volume [12]. However, perfect availability is not a realistic goal for Internet-connected services because the Internet itself has limited availability characteristics. Chandra et al. [91] report that Internet end points may be unable to reach each other between 1% and 2% of the time due to a variety of connectivity problems, including routing issues. That translates to an availability of less than two "nines." In other words, even if your Internet service is perfectly reliable, users will, on average, perceive it as being no greater than 99.0% available. As a result, an Internet-connected service that avoids long-lasting outages for any large group of users and has an average unavailability of less than 1% will be difficult to distinguish from a perfectly reliable system. Google measurements of Internet availability indicate that it is likely on average no better than 99.9% when Google servers are one of the end points, but the spectrum is fairly wide. Some areas of the world experience significantly lower availability.

Measuring service availability in absolute time is less useful for Internet services that typically see large daily, weekly, and seasonal traffic variations. A more appropriate availability metric is the fraction of requests that is satisfied by the service, divided by the total number of requests made by users; a metric called *yield* by Brewer [10].

Finally, one particularly damaging class of failures is the loss or corruption of committed updates to critical data, particularly user data, critical operational logs, or relevant data that are hard or impossible to regenerate. Arguably, it is much more critical for services not to lose data than to be perfectly available to all users. It can also be argued that such critical data may correspond to a relatively small fraction of all the data involved in a given service operation. For example, copies of the Web and their corresponding index files are voluminous and important data for a search engine but can ultimately be regenerated by recrawling the lost partition and recomputing the index files.

In summary, near perfect reliability is not universally required in Internet services. Although it is desirable to achieve it for faults such as critical data corruption, most other failure modes can tolerate lower reliability characteristics. Because the Internet itself has imperfect availability, a user may be unable to perceive the differences in quality of service between a perfectly available service and one with, say, 4 nines of availability.

7.2.2 Causes of Service-Level Faults

In WSCs, it is useful to understand faults in terms of their likelihood of affecting the health of the whole system, such as causing outages or other serious service-level disruption. Oppenheimer et al. [58] studied three Internet services, each consisting of more than 500 servers, and tried to identify the most common sources of service-level failures. They conclude that operator-caused or misconfiguration

errors are the largest contributors to service-level failures, with hardware-related faults (server or networking) contributing to 10–25% of the total failure events.

Oppenheimer's data are somewhat consistent with the seminal work by Gray [35], which looks not at Internet services but instead examines field data from the highly fault-tolerant Tandem servers between 1985 and 1990. He also finds that hardware faults are responsible for a small fraction of total outages (less than 10%). Software faults (~60%) and maintenance/operations faults (~20%) dominate the outage statistics.

It is somewhat surprising at first to see hardware faults contributing to such few outage events in these two widely different systems. Rather than making a statement about the underlying reliability of the hardware components in these systems, such numbers indicate how successful the fault-tolerant techniques have been in preventing component failures from affecting high-level system behavior. In Tandem's case, such techniques were largely implemented in hardware, whereas in the systems Oppenheimer studied, we can attribute it to the quality of the fault-tolerant software infrastructure. Whether software- or hardware-based, fault-tolerant techniques do particularly well when faults are largely statistically independent, which is often (even if not always) the case in hardware faults. Arguably, one important reason why software-, operator-, and maintenance-induced faults have a high impact on outages is because those are more likely to affect multiple systems at once, therefore creating a correlated failure scenario that is much more difficult to overcome.

Our experience at Google is generally in line with Oppenheimer's classification, even if the category definitions are not fully consistent. Figure 7.1 represents a rough classification of all events

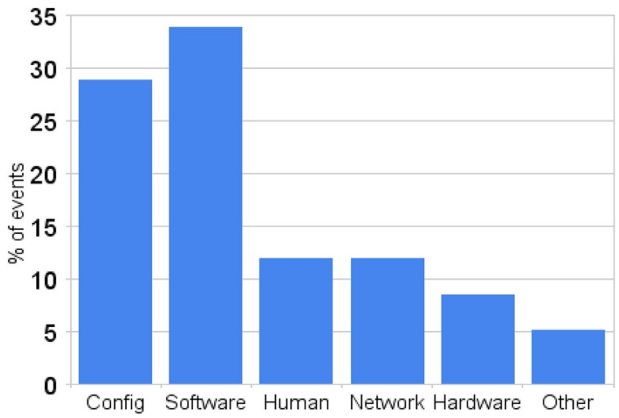

FIGURE 7.1: Distribution of service disruption events by most likely cause at one of Google's main services (preliminary data, 6 weeks only—from Google's Robert Stroud).

that corresponded to noticeable disruptions at the service level in one of Google's large-scale online services. These are not necessarily outages (in fact, most of them are not even user-visible events) but correspond to situations where some kind of service degradation is noticed by the monitoring infrastructure and has to be scrutinized by the operations team. As expected, the service is less likely to be disrupted by machines or networking faults than by software errors, faulty configuration data, and human mistakes.

7.3 MACHINE-LEVEL FAILURES

An important factor in designing fault-tolerant distributed systems is understanding availability at the server level. Here we consider machine-level failures to be all the situations that lead to a server being down, whatever the cause might be (e.g., including operating system bugs).

As with cluster-service failures, there are relatively little published field data on server availability. A 1999 study by Kalyanakrishnam et al. [49] finds that Windows NT machines involved in a mail routing service for a commercial organization were on average 99% available. The authors observed 1,100 reboot events across 66 servers and saw an average uptime of 11.82 days (median of 5.54 days) and an average downtime of just less than 2 hours (median of 11.43 minutes). About one half of the reboots were classified as abnormal, that is, because of a system problem instead of a normal shutdown. Only 10% of the reboots could be blamed on faulty hardware or firmware. The data suggest that application faults, connectivity problems, or other system software failures are the largest known crash culprits. If we are only interested in the reboot events classified as abnormal, we arrive at an MTTF of approximately 22 days, or an annualized machine failure rate of more than 1,600%.

Schroeder and Gibson [75] studied failure statistics from high-performance computing systems at Los Alamos National Laboratory. Although these are not the class of computers that we are interested in here, they are made up of nodes that resemble individual servers in WSCs so their data are relevant in understanding machine-level failures in our context. Their data span nearly 24,000 processors, with more than 60% of them deployed in clusters of small-scale SMPs (2–4 processors per node). Although the node failure rates vary by more than a factor of 10× across different systems, the failure rate normalized by number of processors is much more stable—approximately 0.3 faults per year per CPU—suggesting a linear relationship between number of sockets and unreliability. If we assume servers with four CPUs, we could expect machine-level failures to be at a rate of approximately 1.2 faults per year or an MTTF of approximately 10 months. This rate of server failures is more than 14 times lower than the one observed in Kalyanakrishnam's study![2]

[2]As a reference, in an Intel Technical Brief [61], the 10-month failure rate of servers in Intel datacenters is reported as being 3.83%, corresponding to an annualized failure rate of approximately 4.6%.

Google's machine-level failure and downtime statistics are summarized in Figure 7.1 and Figure 7.2 (courtesy of Google's Andrew Morgan). The data are based on a 6-month observation of all machine restart events and their corresponding downtime, where downtime corresponds to the entire time interval where a machine is not available for service, regardless of cause. These statistics are over all of Google's machines. For example, they include machines that are in the repairs pipeline, planned downtime for upgrades, as well as all kinds of machine crashes.

Figure 7.2 shows the distribution of machine restart events. More than half of the servers are up throughout the observation interval, and more than 95% of machines restart less often than once a month. The tail, however, is relatively long (the figure truncates the data at 11 or more restarts). For example, approximately 1% of all machines restart more often than once a week.

Several effects, however, are smudged away by such large-scale averaging. For example, we typically see higher than normal failure rates during the first few months of new server product introduction. The causes include manufacturing bootstrapping effects, firmware and kernel bugs, and occasional hardware problems that only become noticeable after a large number of systems are in use. If we exclude from the sample all machines that are still suffering from such effects, the annualized restart rate drops by approximately a factor of 2, corresponding to more than 5 months between restarts, on average. We also note that machines with very frequent restarts are less likely to be on active service for long. If we exclude machines that restart more often than once per week—approximately 1% of the population—the average annualized restart rate drops to 2.53.

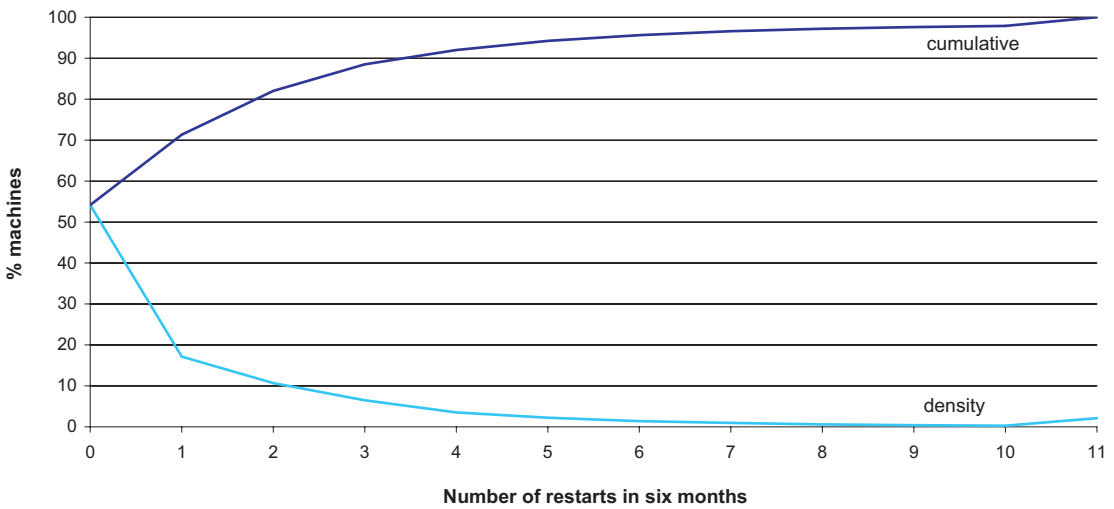

FIGURE 7.2: Distributions of machine restarts over 6 months at Google.

Restart statistics are key parameters in the design of fault-tolerant software systems, but the availability picture is only complete once we combine it with downtime data—a point that has been well articulated earlier by the Berkeley ROC project [62]. Figure 7.3 shows the distribution of downtime from the same population of Google servers. The *x*-axis displays downtime, against both density and cumulative machine distributions. Note that the data include both planned reboots and those caused by miscellaneous hardware and software failures. Downtime includes all the time because a machine stopped operating until it has rebooted and basic node services were restarted. In other words, the downtime interval does not end when the machine finishes reboot but when key basic daemons are up.

Approximately 55% of all restart events last less than 6 minutes, 25% of them last between 6 and 30 minutes, with most of the remaining restarts finishing in about a day. Approximately 1% of all restart events last more than a day, which likely corresponds to systems going into repairs. The average downtime is just more than 3 hours, a value that is heavily affected by the group of restart events that last between 30 and 2,000 minutes (~33 hours). There are several causes for these slow restarts, including file system integrity checks, machine hangs that require semiautomatic restart processes, and machine software reinstallation and testing. The resulting average machine availability is 99.84%, or nearly "three nines."

When provisioning fault-tolerant software systems, it is also important to focus on real (unexpected) machine crashes, as opposed to the above analysis that considers all restarts. In our experience,

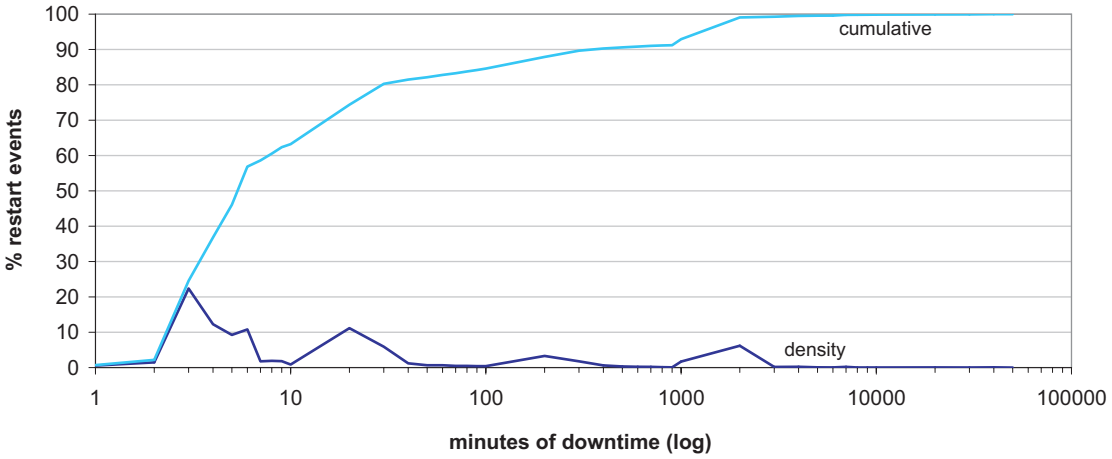

FIGURE 7.3: Distribution of machine downtime, observed at Google over 6 months. The average annualized restart rate across all machines is 4.2, corresponding to a mean time between restarts of just less than 3 months.

the crash rate of mature servers (i.e., those that survived infant mortality) ranges between 1.2 and 2 crashes per year. In practice, this means that a service that uses 2,000 servers should plan to operate normally while tolerating a machine crash approximately every 2.5 hours, or approximately 10 machines per day. Given that the expected machine downtime for 99% of all restart cases is less than 2 days, one would need as few as 20 spare machines to safely keep the service fully provisioned. A larger margin might be desirable if there is a large amount of state that must be loaded for a machine to be ready for service.

7.3.1 What Causes Machine Crashes?

Reliably identifying culprits for machine crashes is generally difficult because there are many situations in which a transient hardware error can be hard to distinguish from an operating system or firmware bug. However, there is significant indirect and anecdotal evidence suggesting that software-induced crashes are much more common than those triggered by hardware faults. Some of this evidence comes from component-level diagnostics. Because memory and disk subsystem faults were the two most common diagnostics for servers sent to hardware repairs within Google in 2007, we will focus on those.

DRAM soft-errors. Although there are little available field data on this topic, it is generally believed that DRAM soft error rates are extremely low once modern ECCs are used. In a 1997 IBM white paper, Dell [22] sees error rates from chipkill ECC being as low as six errors for 10,000 one-GB systems over 3 years (0.0002 errors per GB per year—an extremely low rate). A survey article by Tezzaron Semiconductor in 2004 [83] concludes that single-error rates per Mbit in modern memory devices range between 1,000 and 5,000 FITs (faults per billion operating hours), but that the use of ECC can drop soft-error rates to a level comparable to that of hard errors.

A recent study by Schroeder et al. [77] evaluated DRAM errors for the population of servers at Google and found FIT rates substantially higher than previously reported (between 25,000 and 75,000) across multiple DIMM technologies. That translates into correctable memory errors affecting about a third of Google machines per year and an average of one correctable error per server every 2.5 hours. Because of ECC technology, however, only approximately 1.3% of all machines ever experience uncorrectable memory errors per year.

Disk errors. Studies based on data from Network Appliances [3], Carnegie Mellon [76], and Google [65] have recently shed light onto the failure characteristics of modern disk drives. Hard failure rates for disk drives (measured as the annualized rate of replaced components) have typically ranged between 2% and 4% in large field studies, a much larger number than the usual manufacturer specification of 1% or less. Bairavasundaram et al. [3] looked specifically at the rate of latent sector

errors—a measure of data corruption frequency—in a population of more than 1.5 million drives. From their data we can extrapolate that a 500-GB drive might see on average 1.3 corrupted sectors per year. This extrapolation is somewhat misleading because corruption events are not uniformly distributed over the population but instead concentrated on a subset of "bad" devices. For example, their study saw less than 3.5% of all drives develop any errors over a period of 32 months.

The numbers above suggest that the average fraction of machines crashing annually due to disk or memory subsystem faults should be less than 10% of all machines. Instead we observe crashes to be more frequent and more widely distributed across the machine population. We also see noticeable variations on crash rates within homogeneous machine populations that are more likely explained by firmware and kernel differences.

Another indirect evidence of the prevalence of software-induced crashes is the relatively high mean time to hardware repair observed in Google's fleet (more than 6 years) when compared to the mean time to machine crash (6 months or less).

It is important to mention that a key feature of well-designed fault-tolerant software is its ability to survive individual faults whether they are caused by hardware or software errors.

7.3.2 Predicting Faults

The ability to predict future machine or component failures is highly valued because that could avoid the potential disruptions of unplanned outages. Clearly, models that can predict most instances of a given class of faults with very low false-positive rates can be very useful, especially when those predictions involve short time-horizons—predicting that a memory module will fail within the next 10 years with 100% accuracy is not particularly useful from an operational standpoint.

When prediction accuracies are less than perfect, which unfortunately tends to be true in most cases, the model's success will depend on the trade-off between accuracy (both in false-positive rates and time horizon) and the penalties involved in allowing faults to happen and recovering from them. Note that a false component failure prediction incurs all of the overhead of the regular hardware repair process (parts, technician time, machine downtime, etc.). Because software in WSCs is designed to gracefully handle all the most common failure scenarios, the penalties of letting faults happen are relatively low; therefore, prediction models must have much greater accuracy to be economically competitive. By contrast, traditional computer systems in which a machine crash can be very disruptive to the operation may benefit from less accurate prediction models.

Pinheiro et al. [65] describe one of Google's attempts to create predictive models for disk drive failures based on disk health parameters available through the Self-Monitoring Analysis and Reporting Technology standard. They conclude that such models are unlikely to predict most failures and will be relatively inaccurate for the failures the models do predict. Our general experience is that only a small subset of failure classes can be predicted with high enough accuracy to produce useful operational models for WSCs.

7.4 REPAIRS

An efficient repair process is critical to the overall cost efficiency of WSCs. A machine in repairs is effectively out of operation, so the longer a machine is in repairs the lower the overall availability of the fleet. Also, repair actions are costly in terms of both replacement parts and the skilled labor involved. Lastly, repair quality—how likely it is that a repair action will actually fix a problem while accurately determining which (if any) component is at fault—affects both component expenses and average machine reliability.

There are two characteristics of WSCs that directly affect repairs efficiency. First, because of the large number of relatively low-end servers involved and the presence of a software fault-tolerance layer, it is not as critical to quickly respond to individual repair cases because they are unlikely to affect overall service health. Instead, a datacenter can implement a schedule that makes the most efficient use of a technician's time by making a daily sweep of all machines that need repairs attention. The philosophy is to increase the rate of repairs while keeping the repairs latency within acceptable levels.

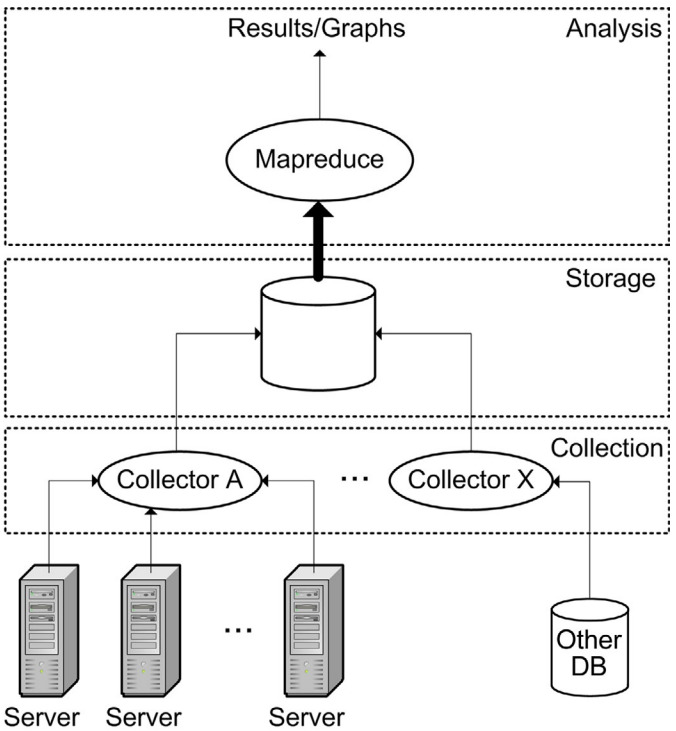

FIGURE 7.4: The Google System Health monitoring and analysis infrastructure.

In addition, when there are many thousands of machines in operation, massive volumes of data about machine health can be collected and analyzed to create automated systems for health determination and diagnosis. The Google *System Health* infrastructure illustrated in Figure 7.4 is one example of a monitoring system that takes advantage of this massive data source. It constantly monitors every server for configuration, activity, environmental, and error data. System Health stores this information as a time series in a scalable repository where it can be used for various kinds of analysis, including an automated machine failure diagnostics tool that uses machine learning methods to suggest the most appropriate repairs action.

In addition to performing individual machine diagnostics, the System Health Infrastructure has been useful in other ways. For example, it monitors the stability of new system software versions and has helped pinpoint a specific batch of defective components across a large fleet of machines. It has also been valuable in longer-term analytical studies, such as the disk failure study by Pinheiro et al. mentioned in the previous section and the datacenter-scale power provisioning study by Fan et al. [27].

7.5 TOLERATING FAULTS, NOT HIDING THEM

The capacity of well-designed fault-tolerant software to mask large numbers of failures with relatively little impact to service-level metrics could have unexpectedly dangerous side effects. Consider a three-tier application representing a Web service with the back-end tier replicated three times. Such replicated setups have the dual purpose of increasing peak throughput as well as tolerating server faults when operating below peak capacity. Assume that the incoming request rate is at 50% of total capacity. At this level, this setup could survive one back-end failure with little disruption in service levels. However, a second back-end failure would have a dramatic service-level impact that could theoretically result in a complete outage.

This simplistic example illustrates a feature of systems with large amounts of internal redundancy. They can tolerate failures so well that an outside observer might be unaware of how much internal slack remains, or in other words, how close to the edge one might be. In those cases, the transition from healthy behavior to meltdown can be very abrupt, which is not a desirable property. This example emphasizes the importance of comprehensive monitoring, both at the application (or service) level as well as the machine infrastructure level, so that faults can be well tolerated and yet visible to operators. This enables prompt corrective action when the amount of internal redundancy approaches the limits of what the fault-tolerant software layer can handle.

Still, broken machines eventually must be repaired. When we can batch repairs, we can lower the costs per repair below those in traditional scenarios where a repair must happen immediately, which requires more costly staging of replacement parts as well as the additional costs of bringing a service technician on site. For reference, service contracts for IT equipment that provide on-site

repair within 24 hours typically come at an annual cost of 5–15% of the equipment's value; 4-hour response times usually double that cost. In comparison, repairs in large server farms should be cheaper. To illustrate the point, assume that a WSC has enough scale to keep a full-time repairs technician busy. Assuming 1 hour per repair and an annual failure rate of 5%, a system with 40,000 servers would suffice; in reality, that number will be considerably smaller because the same technician can also handle installs and upgrades. Let us further assume that the hourly cost of a technician is $100 and that the average repair requires replacement parts costing 10% of the system cost; both of these assumptions are quite generously high. Still, for a cluster of servers costing $2,000 each, we arrive at an annual cost per server of 5% * ($100 + 10% * $2,000) = $15, or 0.75% per year. In other words, keeping large clusters healthy can be quite affordable.

• • • •

CHAPTER 8

Closing Remarks

Rising levels of broadband Internet access are making it possible for a growing number of applications to move from the desktop to a Web services delivery model. In this model, commonly referred to as cloud computing, datacenters with massive amounts of well-connected processing and storage resources can be efficiently amortized across a large user population and multiple ubiquitous workloads. These datacenters are quite different from traditional co-location 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 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 tens to hundreds 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 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 the past decades. Instead it looks more like a building or warehouse—computer architecture meets traditional (building) architecture. We have therefore named this emerging class of machines *warehouse-scale computers* (*WSC*s).

Hardware and software architects need to develop a better understanding of the characteristics of this class of computing systems so that they can continue to design and program today's WSCs. But architects should keep in mind that these WSCs are the precursors of tomorrow's everyday datacenters. The trend toward aggressive many-core parallelism should make even modest-sized

computing systems approach the behavior of today's WSCs in a few years, when thousands of hardware threads might be available in single enclosure.

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 section, we summarize the main characteristics of WSC systems described in previous sections and list some important challenges and trends.

8.1 HARDWARE

The building blocks of choice for WSCs are commodity server-class machines, consumer- or enterprise-grade disk drives, 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 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—in clusters with tens of thousands of systems, even clusters with highly reliable servers 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 cost (per gigabyte) of DRAM or FLASH storage make them prohibitively expensive for large data sets or infrequently accessed data; therefore, disks 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. The use of many small-scale servers demands networking fabrics with very high port counts and high bi-section 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

Because of its scale, complexity of the architecture (as seen by the programmer), and the need to tolerate frequent failures, WSCs are more complex programming targets than traditional computing systems that operate on much smaller numbers of components.

Internet services must achieve high availability, typically aiming for a target of 99.99% or better (about an hour of downtime per year). Achieving fault-free operation on a large collection of hardware and system software is hard, and made harder by the large number of servers involved. Although it might be theoretically possible to prevent hardware failures in a collection of 10,000 servers, it would surely be extremely expensive. Consequently, warehouse-scale workloads must be designed to gracefully tolerate large numbers of component faults with little or no impact on service-level performance and availability.

This workload differs substantially from that running in traditional high-performance computing (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:

- *Ample parallelism*—Typical Internet services exhibit a large amount of 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.
- 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 a few weeks, compared to months or years for desktop software products.
- *Platform homogeneity*—The datacenter is generally 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. Significant heterogeneity arises primarily from the incentives to deploy more cost-efficient components that become available over time.
- Fault-free operation—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.

8.3 ECONOMICS

The relentless demand for greater cost efficiency in computing vs. higher computing performance has made cost become the primary metric in the design of WSC systems. And cost efficiency must

be defined broadly to account for all the significant components of cost including hosting 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 systems 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. Unfortunately, most of today's components (with the exception of the CPU) are far from being energy proportional.

In addition, 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 1 W of power for every watt of server power. Unfortunately, many existing 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 1.19 PUE across all its custom-built facilities as of late 2008 [32].

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 well 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 KEY CHALLENGES

We are still learning how to best design and use this new class of machines, but our current understanding allows us to identify some of the architectural challenges in this space that appear most pressing.

8.4.1 Rapidly Changing Workloads

Internet services are in their infancy as an application area, and 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, consider how the YouTube video sharing site exploded in popularity in a period of a few months and how distinct the needs of such an application are from earlier Web services such as email or search. Parts of WSCs include building structures that are expected to last more than a decade to leverage the construction investment. The difficult 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 Building Balanced Systems from Imbalanced Components

Processors continue to get faster and more energy efficient despite the end of the MHz race as more aggressive many-core products are introduced. Memory systems and magnetic storage are not evolving at the same pace, whether in performance or energy efficiency. Such trends are making computer performance and energy usage increasingly dominated by non-CPU components and that also applies to WSCs. Ultimately, sufficient research focus must shift to these other subsystems or further increases in processor technology will not produce noticeable system level improvements. In the meantime, architects must try to build efficient large-scale systems that show some balance in performance and cost despite the shortcomings of the available components.

8.4.3 Curbing Energy Usage

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. Otherwise, the requirements of future computing systems will demand an ever-growing share of the planet's scarce energy budget, creating a scenario where energy usage increasingly curbs growth in computing capabilities.

8.4.4 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. 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.

8.5 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 cheap 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 understand this relatively new area, and we trust that in the years to come, their combined efforts will solve many of the fascinating problems arising from warehouse-scale systems.

• • • •

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity datacenter network architecture," in Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, Seattle, WA, August 17–22, 2008.

[2] D. Atwood and J. G. Miner, "Reducing datacenter costs with an air economizer," IT@Intel Brief, August 2008.

[3] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," in Proceedings of the 2007 ACM SIGMETRICS international Conference on Measurement and Modeling of Computer Systems, San Diego, CA, June 12–16, 2007. SIGMETRICS '07. doi:10.1145/1254882.1254917

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, October 19–22, 2003. SOSP '03. doi:10.1145/945445.945462

[5] L. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: the architecture of the Google cluster," IEEE Micro, April 2003. doi:10.1109/MM.2003.1196112

[6] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," IEEE Computer, vol. 40, no. 12 (Dec. 2007).

[7] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in Proceedings of OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November 2006.

[8] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in Proceedings of SIGCOMM, 2007.

[9] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, "Magpie: online modeling and performance-aware systems," in Proceedings of USENIX HotOS IX 2003.

[10] E. A. Brewer, "Lessons from giant-scale services," IEEE Internet Computing, vol. 5, no. 4 (July/Aug. 2001), pp. 46–55. doi:10.1109/4236.939450

[11] E. V. Carrera, E. Pinheiro, and R. Bianchini, "Conserving disk energy in network servers," in Proceedings of the 17th Annual International Conference on Supercomputing, San Francisco, CA, June 23–26, 2003. ICS '03. doi:10.1145/782814.782829

[12] B. Chandra, M. Dahlin, L. Gao, and A Nayate, "End-to-end WAN service availability," in Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems—Volume 3, San Francisco, CA, March 26–28, 2001.

[13] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in Proceedings of OSDI 2006, Seattle, WA, 2004.

[14] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," in Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), Banff, Alberta, Canada, June 2001. doi:10.1145/502034.502045

[15] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy aware server provisioning and load dispatching for connection-intensive Internet services," Microsoft Research Technical Report MSR-TR-2007-130, 2007.

[16] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in Proceedings of the ACM SIGMETRICS '05, Banff, Alberta, Canada, June 2005. doi:10.1145/1064212.1064253

[17] Climate savers computing efficiency specs. Available at http://www.climatesaverscomputing.org/about/tech-specs.

[18] E. F. Coyle, "Improved muscular efficiency displayed as tour de France champion matures," Journal of Applied Physiology, Mar. 2005. doi:10.1152/japplphysiol.00216.2005

[19] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1 (2008), pp. 107–113.

[20] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in the Proceedings of the 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, October 2007.

[21] Dell Energy Calculator. Available at http://www.dell.com/calc.

[22] T. J. Dell, "A white paper on the benefits of chipkill-correct ECC for PC server main memory," IBM Microelectronics Division, Rev 11/19/1997.

[23] D. Dyer, "Current trends/challenges in datacenter thermal management—a facilities perspective," presentation at ITHERM, San Diego, CA, June 1, 2006.

[24] J. Elerath and S. Shah, "Server class disk drives: how reliable are they?", IEEE Reliability and Maintainability, 2004 Annual Symposium—RAMS, January 2004. doi:10.1109/RAMS.2004.1285439

[25] Dupont Fabros Technology Inc. SEC Filing (S-11) 333-145294, August 9, 2007.

[26] U.S. Environmental Protection Agency, "Report to Congress on server and datacenter energy efficiency," Public Law 109-431, August 2, 2007.

[27] X. Fan, W. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in Proceedings of the 34th Annual International Symposium on Computer Architecture, San Diego, CA, June 09–13, 2007. ISCA '07. doi:10.1145/1250662.1250665

[28] M. E. Femal and V. W. Freeh, "Safe overprovisioning: using power limits to increase aggregate throughput," in 4th International Workshop on Power Aware Computer Systems (PACS 2004), December 2004, pp. 150–164.

[29] M. E. Femal and V. W. Freeh, "Boosting datacenter performance through non-uniform power allocation," in Proceedings of the Second International Conference on Automatic Computing, June 13–16, 2005, pp. 250-261.

[30] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "X-trace: a pervasive network tracing framework," in Proceedings of 4th USENIX Symposium on Networked Systems Design & Implementation, 2007, pp. 271–284.

[31] S. Ghemawat, H. Gobioff, and S-T. Leung, "The Google file system", in Proceedings of the 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October 2003. doi:10.1145/945445.945450

[32] Google Inc., "Efficient computing—step 2: efficient datacenters". Available at http://www.google.com/corporate/green/datacenters/step2.html.

[33] Google Inc., "Efficient Data Center Summit, April 2009". Available at http://www.google.com/corporate/green/datacenters/summit.html.

[34] Google Inc., "Efficient Data Center, Part 1". Available at http://www.youtube.com/watch?v=Ho1GEyftpmQ, starting at 0:5930.

[35] J. Gray, "A census of tandem system availability between 1985 and 1990," Tandem Technical Report 90.1, January 1990.

[36] The Green 500. Available at http://www.green500.org.

[37] The Green Grid datacenter power efficiency metrics: PUE and DCiE. Available at http://www.thegreengrid.org/sitecore/content/Global/Content/white-papers/The-Green-Grid-Data-Center-Power-Efficiency-Metrics-PUE-and-DCiE.aspx.

[38] Green Grid, "Quantitative analysis of power distribution configurations for datacenters". Available at http://www.thegreengrid.org/gg_content/.

[39] Green Grid, "Seven strategies to improve datacenter cooling efficiency". Available at http://www.thegreengrid.org/gg_content/.

[40] SNIA Green Storage Initiative. Available at http://www.snia.org/forums/green/.

[41] S. Greenberg, E. Mills, and B. Tschudi, "Best practices for datacenters: lessons learned from benchmarking 22 datacenters," 2006 ACEEE Summer Study on Energy Efficiency in Buildings. Available at http://eetd.lbl.gov/EA/mills/emills/PUBS/PDF/ACEEE-datacenters.pdf.

[42] The Hadoop Project. Available at http://hadoop.apache.org

[43] J. Hamilton, "On designing and deploying internet-scale services," in Proceedings of USENIX LISA, 2007.

[44] J. Hamilton, "Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for internet-scale services," in Proceedings of the 4th Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, January 4–7, 2009.

[45] T. Heath, A. P. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini, "Mercury and freon: temperature emulation and management for server systems," in Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October 2006.

[46] HP Power Calculator. Available at http://www.hp.com/configurator/powercalcs.asp.

[47] M. Isard, M. Budiu, Y. Yu, A. Birrell, and Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in Proceedings of the 2nd ACM Sigops/Eurosys European Conference on Computer Systems 2007, Lisbon, Portugal, March 21–23, 2007.

[48] M. Isard, "Autopilot: automatic datacenter management". SIGOPS Operating Systems Review, vol. 41, no. 2 (Apr. 2007), pp. 60–67. DOI: http://doi.acm.org/10.1145/1243418.1243426.

[49] M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer, "Failure data analysis of a LAN of Windows NT based computers," Reliable Distributed Systems, IEEE Symposium on, vol. 0, no. 0, pp. 178, 18th IEEE Symposium on Reliable Distributed Systems, 1999. doi:10.1109/RELDIS.1999.805094

[50] J. Koomey, K. Brill, P. Turner, J. Stanley and B. Taylor, "A Simple Model for Determining True Total Cost of Ownership for Data Centers", Uptime Institute White Paper, Version 2, October, 2007. doi:10.1145/1394608.1382148

[51] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt, "Understanding and designing new server architectures for emerging warehouse-computing environments," Computer Architecture, International Symposium on, vol. 0, no. 0, pp. 315–326, 2008 International Symposium on Computer Architecture, 2008.

[52] C. Malone and C. Belady, "Metrics to characterize datacenter & IT equipment energy use," in Proceedings of the Digital Power Forum, Richardson, TX, September 2006.

[53] Mike Manos. "The Capacity Problem". Available at http://loosebolts.wordpress.com/2009/06/02/chiller-side-chats-the-capacity-problem/.

[54] W. D. McArdle, F. I. Katch, and V. L. Katch, Sports and Exercise Nutrition, second ed., LWW Publishers, 2004.

[55] D. Meisner, B. Gold, and T. Wenisch, "PowerNap: eliminating server idle power," in Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Washington, DC, March 2009.

[56] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, "Making scheduling 'cool': temperature-aware workload placement in datacenters," in Proceedings of the Annual Conference on USENIX Annual Technical Conference, Anaheim, CA, April 10–15, 2005.

[57] D. Nelson, M. Ryan, S. DeVito, K. V. Ramesh, P. Vlasaty, B. Rucker, B. Da y Nelson, et al., "The role of modularity in datacenter design". Sun BluePrints Online, http://www.sun.com/storagetek/docs/EED.pdf.

[58] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do Internet services fail, and what can be done about it?," in Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems–Volume 4, Seattle, WA, March 26–28, 2003.

[59] L. Page and S. Brin, "The anatomy of a large-scale hypertextual search engine". Available at http://infolab.stanford.edu/~backrub/google.html.

[60] C. Patel et al., "Thermal considerations in cooling large scale high compute density datacenters". Available at http://www.flomerics.com/flotherm/technical_papers/t299.pdf.

[61] C. Patel et al., "Cost model for planning, development and operation of a datacenter". Available at http://www.hpl.hp.com/techreports/2005/HPL-2005-107R1.pdf.

[62] D. A. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft. "Recovery-oriented computing (ROC): motivation, definition, techniques, and case studies". UC Berkeley Computer Science Technical Report UCB//CSD-02-1175, March 15, 2002.

[63] M. K. Patterson and D. Fenwick, "The state of datacenter cooling," Intel Corporation White Paper. Available at http://download.intel.com/technology/eep/data-center-efficiency/state-of-date-center-cooling.pdf.

[64] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the data: parallel analysis with Sawzall", Scientific Programming Journal, vol. 13, no. 4 (2005), pp. 227–298.

[65] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in Proceedings of 5th USENIX Conference on File and Storage Technologies (FAST 2007), San Jose, CA, February 2007.

[66] PG&E, "High performance datacenters". Available at http://hightech.lbl.gov/documents/DATA_CENTERS/06_DataCenters-PGE.pdf.

[67] W. Pitt Turner IV, J. H. Seader, and K. G. Brill, "Tier classifications define site infrastructure performance," Uptime Institute White Paper.

[68] W. Pitt Turner IV and J. H. Seader, "Dollars per kW plus dollars per square foot are a better datacenter cost model than dollars per square foot alone," Uptime Institute White Paper, 2006.

[69] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No 'power' struggles: coordinated multi-level power management for the datacenter', in Proceedings of the ACM

International Conference on Architectural Support for Programming Languages and Operating Systems, Seattle, WA, March 2008.

[70] Reuters, "Dupont Fabros Technology, Inc. reports third quarter results," November 5, 2008.

[71] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat, "Pip: detecting the unexpected in distributed systems," in Proceedings of USENIX NSDI, 2006.

[72] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat, "WAP5: black box performance debugging for wide-area systems," in Proceedings of the 15th International World Wide Web Conference, 2006.

[73] Robert Frances Group, "Total cost of ownership for Linux in the enterprise," July 2002. Available at http://www-03.ibm.com/linux/RFG-LinuxTCO-vFINAL-Jul2002.pdf. doi:10.1088/1742-6596/78/1/012022

[74] SAVVIS press release, "SAVVIS sells assets related to two datacenters for $200 million," June 29, 2007. Available at http://www.savvis.net/corp/News/Press+Releases/Archive/SAVVIS+Sells+Assets+Related+to+Two+Data+Centers+for+200+Million.htm.

[75] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," Journal of Physics: Conference Series 78 (2007).

[76] B. Schroeder and G. A. Gibson, "Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?," in Proceedings of the 5th USENIX Conference on File and Storage Technologies, February 2007.

[77] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: a large-scale field study," to appear in the Proceedings of SIGMETRICS, 2009.

[78] S. Siddha, V. Pallipadi, and A. Van De Ven, "Getting maximum mileage out of tickless," in Proceedings of the Linux Symposium, Ottawa, Ontario, Canada, June 2007.

[79] SPEC Power. Available at http://www.spec.org/power_ssj2008/.

[80] S. Rivoire, M. A. Shah, P. Ranganathan, C. Kozyrakis, and Justin Meza, "Models and metrics to enable energy-efficiency optimizations," Computer, vol. 40, no. 12 (Dec. 2007), pp. 39–48.

[81] S. Sankar, S. Gurumurthi, and M. R. Stan, "Intra-disk parallelism: an idea whose time has come," in Proceedings of the ACM International Symposium on Computer Architecture, June 2008, pp. 303–314.

[82] Techarp.com, "Intel desktop CPU guide". Available at http://www.techarp.com/showarticle.aspx?artno=337&pgno=6.

[83] Terrazon Semiconductor, "Soft errors in electronic memory—a white paper". Available at http://www.tezzaron.com/about/papers/soft_errors_1_1_secure.pdf.

[84] M. Ton and B. Fortenbury, "High performance buildings: datacenters—server power supplies," Lawrence Berkeley National Laboratories and EPRI, December 2005.

[85] Transaction Processing Performance Council. Available at http://www.tpc.org.

[86] W. F. Tschudi, T. T. Xu, D. A. Sartor, and J. Stein, "High performance datacenters: a research roadmap," Lawrence Berkeley National Laboratory, Berkeley, CA, 2003.

[87] TPC-C Executive summary for the Superdome-Itanium2, February 2007.

[88] TPC-C Executive summary for the ProLiant ML350G5, September 2007.

[89] VMware infrastructure architecture overview white paper. Available at http://www.vmware.com/pdf/vi_architecture_wp.pdf.

[90] W. Vogels, "Eventually consistent," ACM Queue, October 2008. Available at http://queue.acm.org/detail.cfm?id=1466448.

[91] B. Chandra, M. Dahlin, L. Gao, A.-A. Khoja, A .Nayate, A. Razzaq, and A. Sewani, "Resource management for scalable disconnected access to Web services," Proceedings of the 10th International Conference on World Wide Web, Hong Kong, Hong Kong, May 1–5, 2001, pp. 245–256.

[92] "Microsoft's Top 10 Business Practices for Environmentally Sustainable Data Center." Available at http://www.microsoft.com/environment/our_commitment/articles/datacenter_bp.aspx.

Author Biographies

Luiz André Barroso is a distinguished engineer at Google, where he has worked across several engineering areas including software infrastructure, fault analysis, energy efficiency, and hardware design. Luiz was also the first manager of Google's Platforms Engineering team, the group responsible for designing the company's computing platform. Prior to Google, he was a member of the research staff at Digital Equipment Corporation (later acquired by Compaq), where his group did some of the pioneering work on processor and memory system design for commercial workloads. That research led to the design of Piranha, a single-chip multiprocessor that helped inspire some of the multicore CPUs that are now in the mainstream. Before joining Digital, Luiz was one of the designers of the USC RPM, a field-programmable gate array-based multiprocessor emulator for rapid hardware prototyping. He has lectured at Pontifícia Universidade Católica (PUC)-Rio de Janeiro (Brazil) and Stanford University, holds a Ph.D. in computer engineering from the University of Southern California and B.S/M.S. degrees in electrical engineering from the PUC, Rio de Janeiro.

Urs Hölzle served as Google's first vice president of engineering and led the development of Google's technical infrastructure. His current responsibilities include the design and operation of the servers, networks, and datacenters that power Google. He is also renowned for both his red socks and his free-range leonberger, Yoshka (Google's top dog). Urs joined Google from the University of California, Santa Barbara, where he was an associate professor of computer science. He received his masteral degree in computer science from ETH Zurich in 1988 and was awarded a Fulbright scholarship that same year. In 1994, he earned a Ph.D. from Stanford University, where his research focused on programming languages and their efficient implementation.

As one of the pioneers of dynamic compilation, also known as "just-in-time compilation," Urs invented fundamental techniques used in most of today's leading Java compilers. Before joining Google, Urs was a co–founder of Animorphic Systems, which developed compilers for Smalltalk and Java. After Sun Microsystems acquired Animorphic Systems in 1997, he helped build Javasoft's high-performance Hotspot Java compiler.