The Datacenter as a Computer

An Introduction to the Design of Warehouse-Scale Machines

Luiz André Barroso, Jimmy Clidaras and Urs Hölzle

Google Inc.

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE # 6

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.

Notes for the 2nd Edition

After nearly four years of substantial academic and industrial developments in Warehouse-scale computing we are delighted to present our first major update to this lecture. Thanks largely to the help of our new co-author, Google Distinguished Engineer Jimmy Clidaras, the material on facility mechanical and power distribution design has been updated and greatly extended (see Chapters 4 and 5). Chapter 2 has been updated to reflect our better understanding of WSC software systems, in particularly with respect to the challenge of building responsive systems at increasingly large scale and the need for techniques that tolerate subsystem latency variability (the tail-tolerance problem). Chapter 3 now presents an overview of WSC interconnects and storage systems that was promised but lacking in the original edition. We hope this revised edition continues to meet the needs of educators and professionals in this area.

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. 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.

Ricardo Bianchini, Fred Chong, Jeff Dean, and Mark Hill provided extremely useful feedback on early drafts despite being handed a relatively immature early version of the text.

The first edition benefited from the feedback and corrections submitted by Vijay Rao, Robert Hundt, Mike Marty, David Konerding, Jeremy Dion, Juan Vargas, Artur Klauser, Pedro Reviriego Vasallo, Amund Tveit, Xiau Yu, Bartosz Prybylski, Laurie Doyle, Marcus Fontoura, Steve Jenkin, Evan Jones, Chuck Newman, Taro Tokuhiro, Jordi Torres and Christian Belady.

This edition of the book additionally benefits from the feedback and corrections submitted by Amin Vahdat, Dilip Agrawal and Thomas Olavson. 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.

Contents

1 Introduction 8
1.1 WAREHOUSE-SCALE COMPUTERS 9
1.2 EMPHASIS ON COST EFFICIENCY 10
1.3 NOT JUST A COLLECTION OF SERVERS 10
1.4 ONE DATACENTER VS. SEVERAL DATACENTERS 11
1.5 WHY WSCs MIGHT MATTER TO YOU 11
1.6 ARCHITECTURAL OVERVIEW OF WSCs 12
1.6.1 Storage 13
1.6.2 Networking Fabric 13
1.6.3 Storage Hierarchy 14
1.6.4 Quantifying Latency, Bandwidth, and Capacity 15
1.6.5 Power Usage 16
1.6.6 Handling Failures 17
2 Workloads and Software Infrastructure 18
2.1 DATACENTER VS. DESKTOP 18
2.2 PERFORMANCE AND AVAILABILITY TOOLBOX 20
2.3 CLUSTER-LEVEL INFRASTRUCTURE SOFTWARE 23
2.3.1 Resource Management 23
2.3.2 Hardware Abstraction and Other Basic Services 24
2.3.3 Deployment and Maintenance 24
2.3.4 Programming Frameworks 24
2.4 Application-Level Software 24
2.4.1 Workload Examples 25
2.4.2 Online: Web Search 25
2.4.3 Offline: Scholar Article Similarity 28
2.5 A MONITORING INFRASTRUCTURE 29
2.5.1 Service-Level Dashboards 29
2.5.2 Performance Debugging Tools 29
1 Platform-Level Monitoring 30
2.6 Buy vs. Build 30
2.7 Tail-tolerance 31
2.8 FURTHER READING 32
3 Hardware Building Blocks 33
3.1 COST-EFFICIENT SERVER HARDWARE 33
3.1.1 How About Parallel Application Performance? 34
3.1.2 How Low-End Can You Go? 37
3.1.3 Balanced Designs 38
3.2 WSC Storage 39
3.2.1 Unstructured WSC Storage 39
3.2.2 Structured WSC Storage 40

3.2.3 Interplay of storage and networking technology 41
3.3 WSC Networking 41
3.4 New references in this chapter: ..... **Error! Bookmark not defined.**
4 Datacenter Basics 45
4.1 DATACENTER TIER CLASSIFICATIONS AND SPECIFICATIONS 45
4.2 DATACENTER POWER SYSTEMS 46
4.2.1 UPS Systems 47
4.2.2 Power Distribution Units 48
4.2.3 Alternative: DC Distribution 49
4.3 DATACENTER COOLING SYSTEMS 51
4.3.1 CRACs, Chillers, and Cooling Towers 53
a CRACs 53
a Chillers 54
a Cooling towers 55
4.3.2 Free Cooling 56
4.3.3 Air Flow Considerations 57
4.3.4 In-Rack, In-Row Cooling, and Cold Plates 58
4.3.5 Case Study: Google's In-row Cooling 60
4.3.6 Container-Based Datacenters 62
5 Energy and Power Efficiency 64
5.1 DATACENTER ENERGY EFFICIENCY 64
5.1.1 The PUE metric 64
5.1.2 Issues with the PUE metric 66
5.1.3 Sources of Efficiency Losses in Datacenters 67
5.1.4 Improving the Energy Efficiency of Datacenters 68
5.1.5 Beyond the facility 68
5.2 THE ENERGY EFFICIENCY OF COMPUTING 70
5.2.1 Measuring Energy Efficiency 70
5.2.2 Server Energy Efficiency 70
5.2.3 Usage profile of Warehouse-scale Computers 72
5.3 ENERGY-PROPORTIONAL COMPUTING 73
5.3.1 Causes of Poor Energy Proportionality 75
5.3.2 Improving Energy Proportionality 75
5.3.3 Energy Proportionality - The Rest of the System 77
5.4 RELATIVE EFFECTIVENESS OF LOW-POWER MODES 78
5.5 THE ROLE OF SOFTWARE IN ENERGY PROPORTIONALITY 79
5.6 DATACENTER POWER PROVISIONING 80
5.6.1 Deploying the right amount of equipment 80
5.6.2 Oversubscribing Facility Power 81
5.7 TRENDS IN SERVER ENERGY USAGE 82
5.7.1 USING ENERGY STORAGE FOR POWER MANAGEMENT 83
5.8 CONCLUSIONS 84
5.8.1 Further Reading 85
6 Modeling Costs 86
6.1 CAPITAL COSTS 86
6.2 OPERATIONAL COSTS 88

6.3 CASE STUDIES 88
6.3.1 Real-World Datacenter Costs 91
6.3.2 Modeling a Partially Filled Datacenter 91
6.3.3 The Cost of Public Clouds 93
7 Dealing with Failures and Repairs 94
7.1 IMPLICATIONS OF SOFTWARE-BASED FAULT TOLERANCE 94
7.2 CATEGORIZING FAULTS 96
7.2.1 Fault Severity 96
7.2.2 Causes of Service-Level Faults 98
7.3 MACHINE-LEVEL FAILURES 99
7.3.1 What Causes Machine Crashes? 102
7.3.2 Predicting Faults 103
7.4 REPAIRS 104
7.5 TOLERATING FAULTS, NOT HIDING THEM 105
8 Closing Remarks 107
8.1 HARDWARE 108
8.2 SOFTWARE 109
8.3 ECONOMICS 110
8.4 KEY CHALLENGES 111
8.4.1 Rapidly Changing Workloads 111
8.4.2 Building Balanced Systems from Imbalanced Components 111
8.4.3 Curbing Energy Usage 111
8.4.4 Amdahl's Cruel Law 112
8.5 CONCLUSIONS 112

1 Introduction

The ARPANET is over forty years old, and the World Wide Web is approaching its 25th 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. 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 smaller, often mobile devices, combined with 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 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.[1] 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 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.

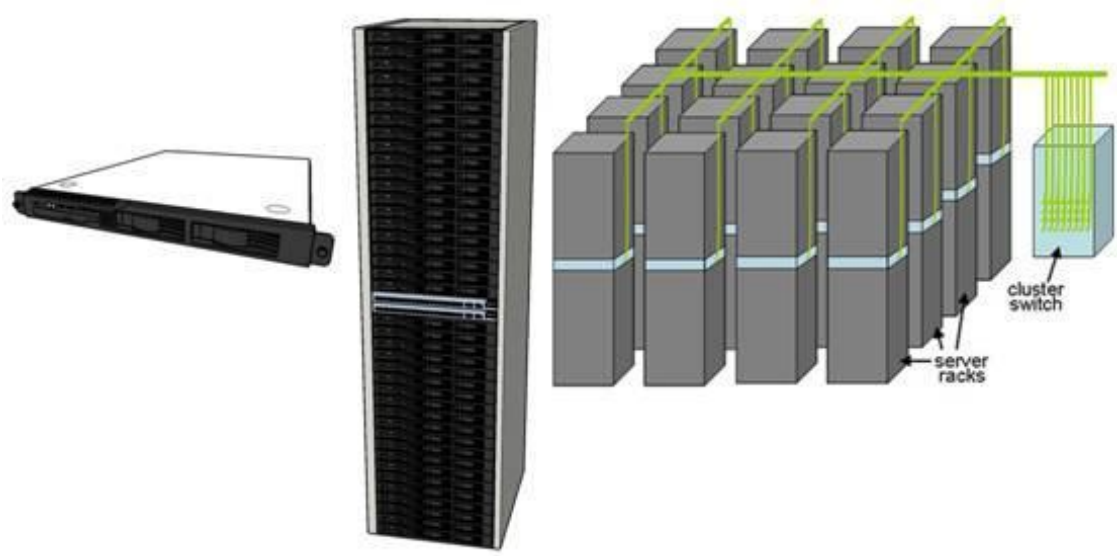

Figure 1-1: Typical elements in warehouse-scale systems: 1U server (left), 7´ rack with Ethernet switch (middle), and diagram of a small cluster with a cluster-level Ethernet switch/router (right).

[1] Arguably the relative statistics about sources of hardware faults might change substantially in these more integrated future systems, but silicon trends which point towards less reliable components and the likely continuing high impact of software-driven faults suggests that programmers of such systems will still need to deal with a fault-ridden platform.

1.6.1 Storage

Disk drives or Flash devices are connected directly to each individual server and managed by a global distributed file system (such as Google's GFS [32]) 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 allows some of the data management responsibilities to be outsourced to a NAS appliance vendor. Keeping storage separate from computing nodes also makes it easier to enforce quality of service guarantees as interference with other compute jobs is avoided. In contrast, using the collection of disks directly attached to server nodes requires a fault-tolerant file system at the cluster level. This is a more complex solution but one can lower hardware costs (the disks leverage the existing server enclosure) and improve 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 tends to provide high availability 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 early workloads. An additional advantage of having disks co-located with compute servers is that it enables distributed system software to exploit data locality, although given how networking performance has outpaced disk performance for the last decades such locality advantages are decreasingly useful.

Some WSCs, including Google's, deploy desktop-class disk drives (or their close cousins, Near Line drives) instead of enterprise-grade disks because of the substantial cost differential between the two. Since data is nearly always replicated in some distributed fashion (as in GFS), higher fault rates of non-enterprise disk models can often be tolerated. 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 [25] point out that several factors can affect disk reliability more substantially than manufacturing process and design.

Improvements in NAND Flash technology is making Solid State Drives (or SSDs) affordable for a growing glass of storage needs in WSCs. While the cost per byte stored in SSDs will remain much higher than in disks for the foreseeable future, many Web services have I/O rate requirements that cannot be easily achieved with disk based systems. Since SSDs can achieve IO rates multiple orders of magnitude higher than disks, they are increasingly displacing disk drives as the repository of choice for data bases in Web services.

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 commodity components, costing less than $30/Gbps per server to connect a single rack. As a result, bandwidth

within a rack of servers tends to be homogeneous. 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 port) than commodity rack switches. As a rule of thumb, a switch that has 10 times the bi-section bandwidth often 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. 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 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.

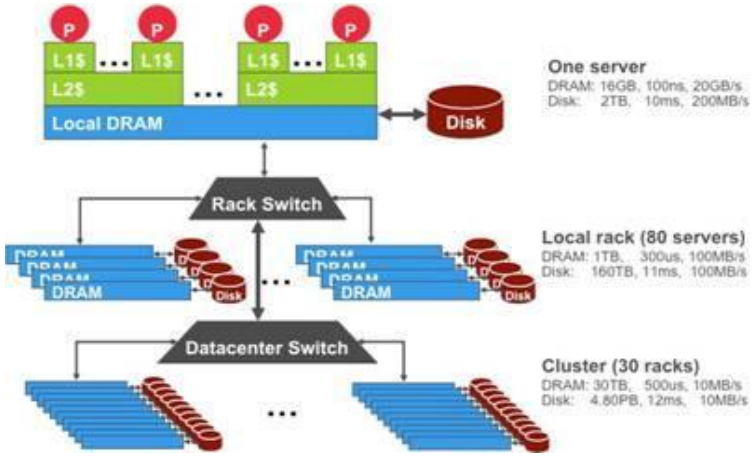

Figure 1-2: Storage hierarchy of a WSC.

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 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.

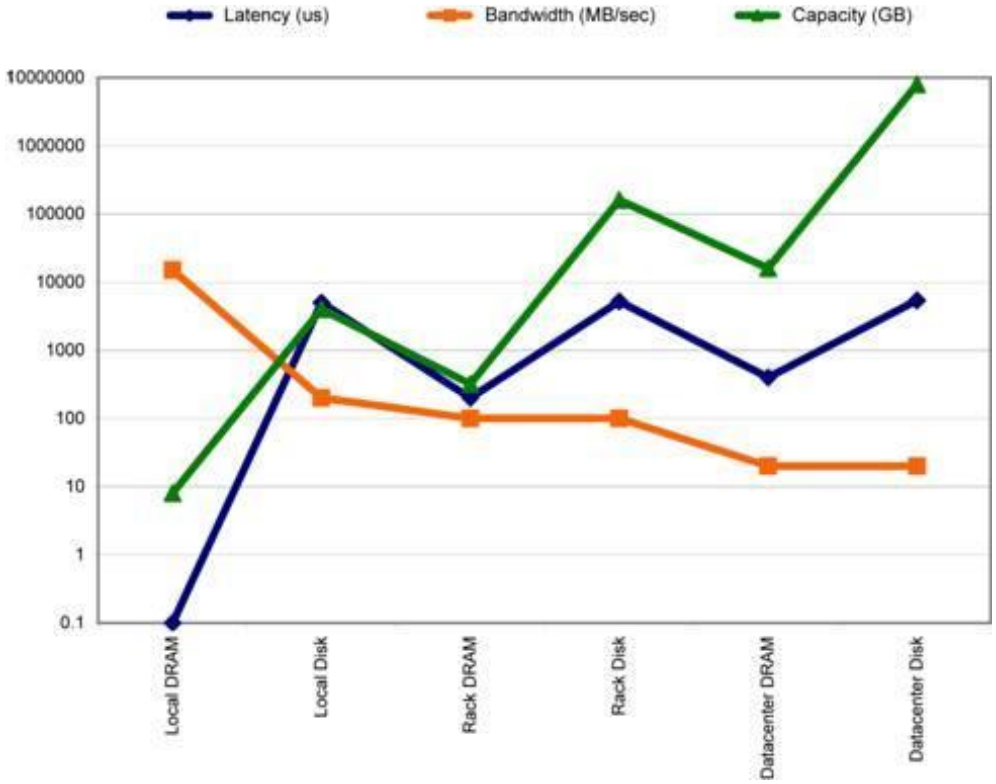

Figure 1-3: Latency, bandwidth, and capacity of a WSC.

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. NAND Flash technology that was originally developed for portable electronics has found target use cases in WSC systems more recently. Today Flash is a viable option for bridging the cost and performance gap between DRAM and disks, as displayed **Figure 1-4**. Flash's most appealing characteristic with respect to disks is its performance under random read operations, which is nearly three orders of magnitude better. In fact, Flash's performance is so high that it becomes a challenge to use it effectively in distributed storage systems since it would demand much higher bandwidth from the WSC fabric. We discuss Flash's potential and challenges further in Chapter 3.

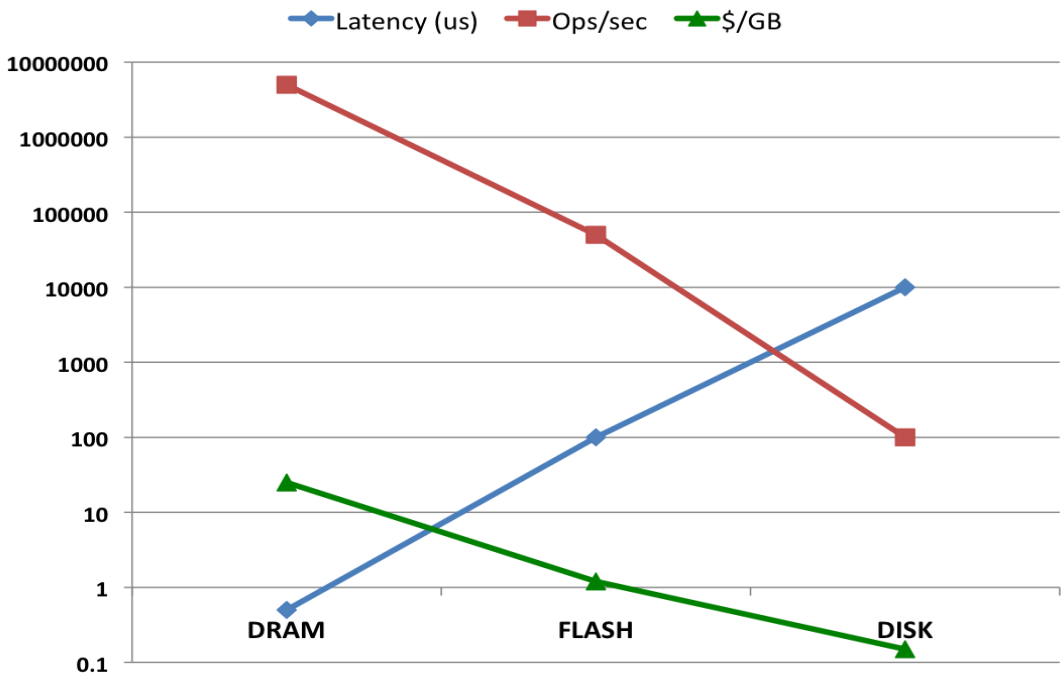

Figure 1-4: Performance and cost of NAND Flash with respect to DRAM and Disks

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 1-5 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.

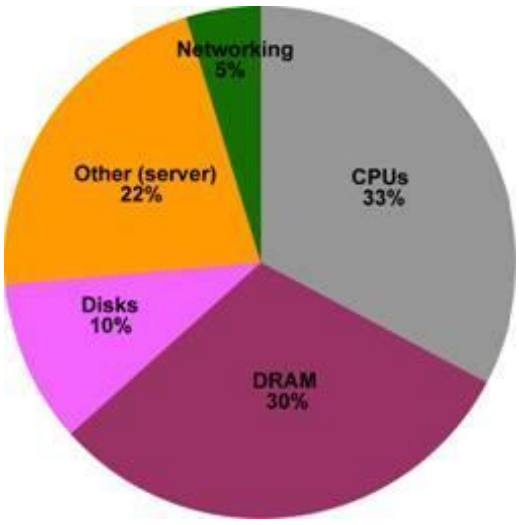

Figure 1-5: Approximate distribution of peak power usage by hardware subsystem in one of Google's datacenters (circa 2007).

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% [67][78]. 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.

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 [48], Hadoop [43], Sawzall [66], BigTable [13], Dynamo [20], Dremel [95], Spanner [96], 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 throughput and availability. It is particularly powerful when the replicated data are not often modified because replication makes updates more complex.
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. Use of very small shards (or microsharding) is particularly beneficial to load balancing and recovery.
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. 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. Microsharding (see above) makes dynamic load-balancing easier since smaller units of work can be changed to mitigate hotspots.
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

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, hurts performance, and reduces availability of distributed applications [92]. 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.
Centralized control	Yes		In theory, building distributed systems in which all actions are coordinated by a single master entity seems inadvasible as the resulting system availability would be bound by the availability of the master. Centralized control is nevertheless much simpler to implement and generally yields more responsive control actions. At Google we have tended towards centralized control models

			for much of our software infractructure (like MapReduce and GFS). Master availability is addressed by designing master failover protocols.
Canaries		Yes	A very rare but realistic catastrophic failure scenario in online services consists of a single request that is distributed to a very large number of servers exposing a program-crashing bug and resulting in system-wide outages. A technique often used at Google to avoid such situations is to first send the request to one (or a few) servers and only submit it to the rest of the system upon successful completion of that (canary) request.

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 yet 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 [32], 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 [49] 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 [67], 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 [31] 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?).

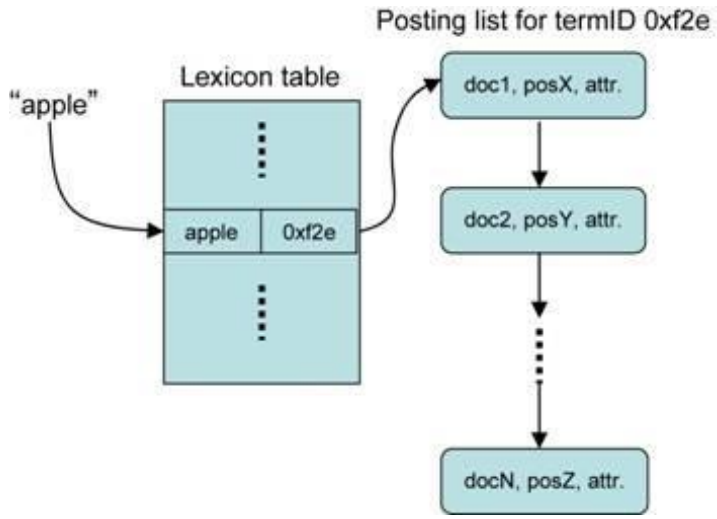

Figure 2-1: Logical view of a Web index.

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 [61]) as well as many other properties associated with the occurrence of the terms in the document (such 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.

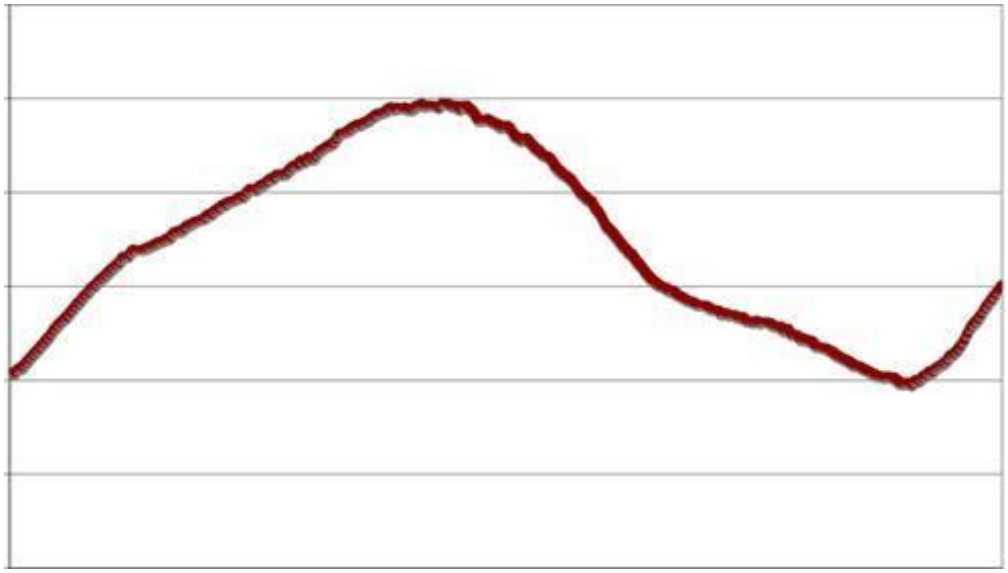

Figure 2-2: Example of daily traffic fluctuation for a search service in one datacenter; *x*-axis is a 24-h period and the *y*-axis is traffic measured in queries per second.

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 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 [74] 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 [73], Magpie [9], and X-trace [31], 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 [97] 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 very 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 eases complex system problems because it provides several options for addressing them. For example, an unwanted networking behavior might be 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 database 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 Tail-tolerance

Earlier in this chapter we described a number of techniques that are commonly used in large scale software systems to achieve high performance and availability. As system scale continues to increase to support more powerful online Web services we have found that such techniques are insufficient to deliver service-wide responsiveness with acceptable tail latency levels. Dean and Barroso [98] have argued that at large enough scale simply stamping out all possible sources of performance variability in individual system components is as impractical as making all components in a large fault-tolerant system fault-free.

For example, consider a system where each server typically responds in 10 ms but with a 99th percentile latency of 1 second. If a user request is handled on just one such server, 1 user request in 100 will be slow (take 1 second). Figure 2-3 shows how service level latency in this hypothetical scenario is impacted by very modest fractions of latency outliers. If a user request must collect responses from one hundred such servers in parallel, then 63% of user requests will take more than 1 second (marked as an 'x' in Figure 2-3). Even for services with only 1 in 10,000 requests experiencing over 1 second latencies at the single server level, a service with 2,000 such servers will see almost 1 in 5 user requests taking over 1 second (marked as an 'o') in the figure.

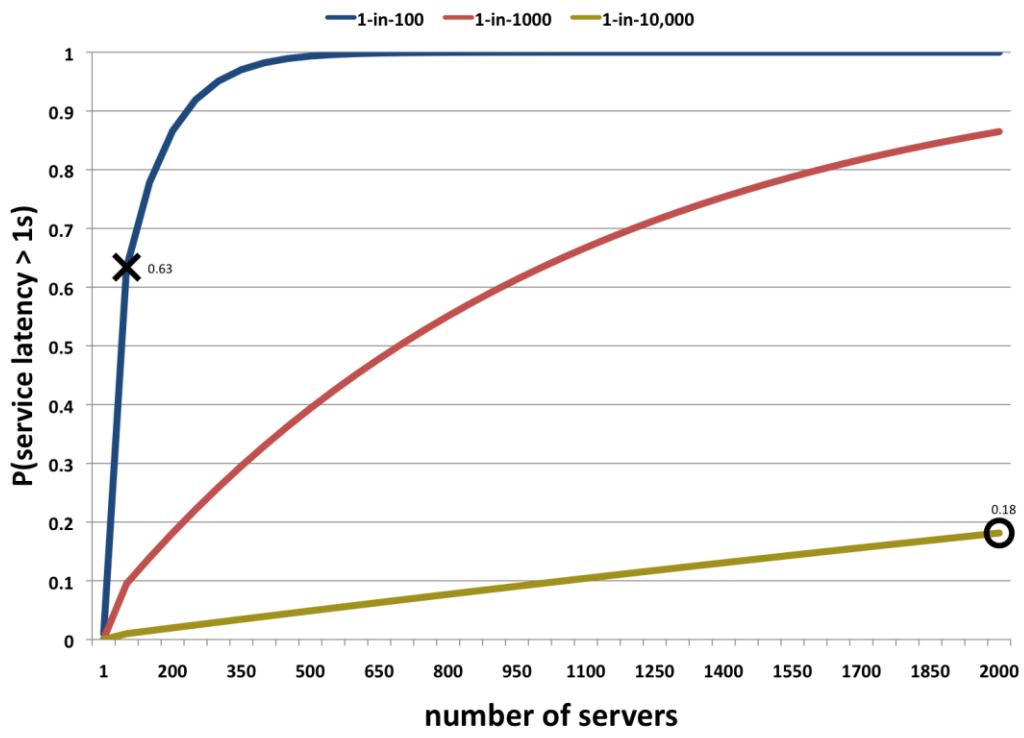

Figure 2-3: Probability of > 1 second service level response time as the system scale and frequency of server level high latency outliers varies

Dean and Barroso show examples of programming techniques that can tolerate these kinds of latency variabilities and still deliver low tail latency at the service level. The techniques they propose often take advantage of resource replication that has already been provisioned for fault-tolerance, thereby achieving small additional overheads for existing systems. They predict that tail tolerant techniques will become more invaluable in the next decade as we build ever more formidable online Web services.

2.8 FURTHER READING

The articles by Hamilton [44], Brewer [10], and Vogels [92] provide interesting further reading on how different organizations have reasoned about the general problem of deploying Internet services at a very large scale.

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 SERVER 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 [87] 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 [89]—with the top system in the price/performance category (HP ProLiant ML350 G5 [90]). 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 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.

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

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 that 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:

Execution time = 1 ms + f * [100 ns/# nodes + 100 µs * (1 − 1/# 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 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×.

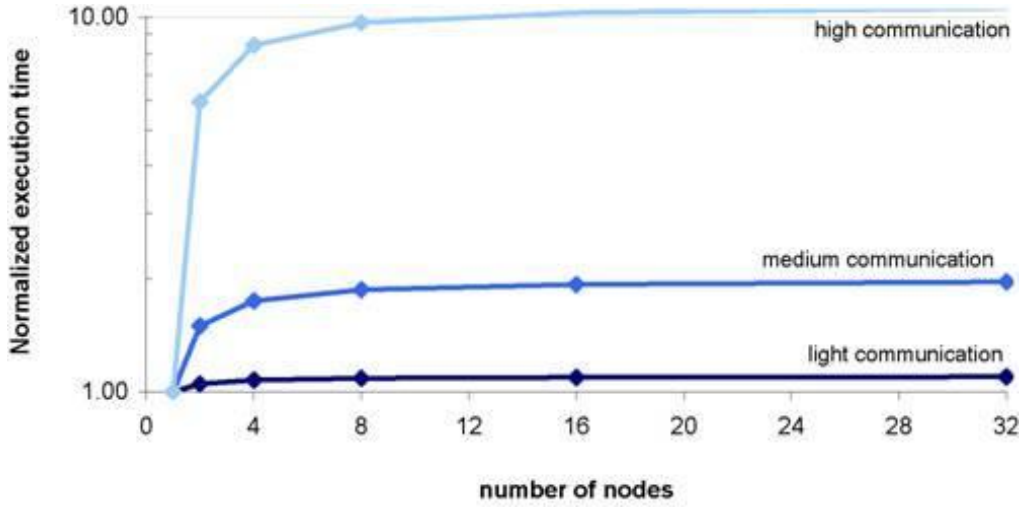

Figure 3-1: Execution time of parallel tasks as the number of SMP nodes increases for three levels of communication intensity. Execution time is normalized to the single node case and plotted in logarithmic scale

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 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.

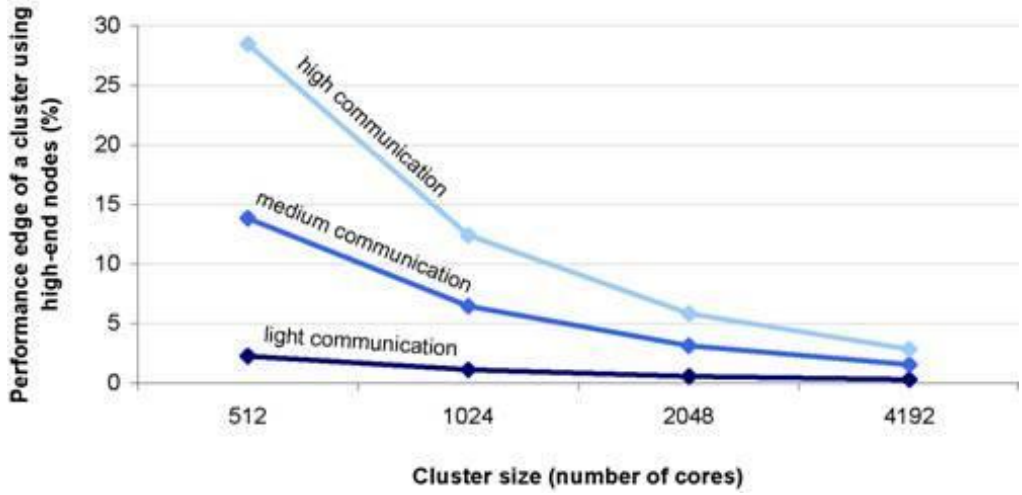

Figure 3-2: Performance advantage of a cluster built with high-end server nodes (128-core SMP) over a cluster with the same number of processor cores built with low-end server nodes (four-core SMP), for clusters of varying size.

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 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 use the argument laid out above to go further and use computing building blocks that are even smaller than today's server-class CPUs. The Piranha chip-multiprocessor [101] was one of the earliest systems to advocate the use of lower-end cores in enterprise-class server systems. In [5], we have argued that chip-multiprocessors using this approach are especially compelling for Google workloads. More recently even more radical approaches that leverage embedded-class CPUs have been proposed as possible alternatives for WSC systems. Lim et al [53], 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 [45] 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 a factor of k.

However, offsetting effects diminish these advantages so that increasingly smaller processing building blocks can eventually turn out to be unattractive for WSCs, a point argued by one of the authors [102], and summarized below.

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, limiting either speedup or scaleup [23][53]. 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 balanced system design 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. Very popular services that either have small data set sizes or significant data locality 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.

3.2 WSC Storage

The data manipulated by WSC workloads tends to fall into two categories: data that is private to individual running tasks and data that is part of the shared state of the distributed workload. Private data tends to reside in local DRAM or disk, is rarely replicated, and its management is simplified by virtue of its single user semantics. Shared data on the other hand has to be much more durable than typical running tasks and may need to be accessed by a large number of clients, therefore they require much more sophisticated distributed storage system. We discuss the main features of these WSC storage systems next.

3.2.1 Unstructured WSC Storage

Google's GFS [32] is an example of a storage system with a simple file-like abstraction (Google's Colossus system has since replaced GFS, but follows a similar architectural philosophy so we choose to describe the better known GFS here). GFS was designed in order to support the Web search indexing system (the system that turned crawled Web pages into index files for use in Web search), and therefore focuses high throughput for thousands of concurrent readers/writes and robust performance under high hardware failures rates. GFS users typically manipulate large quantities of data, and thus GFS is further optimized for large operations. The system architecture consists of a master, which handles metadata operations, and thousands of chunkserver (slave) processes running on every server with a disk drive, to manage the data chunks on those drives. In GFS, fault tolerance is provided by replication across machines instead within them, as is the case in RAID systems. Cross-machine replication allows the system to tolerate machine and network failures and enables fast recovery since replicas for a given disk or machine can be

spread across thousands of other machines, instead of just a handful as is the case in RAID boxes.

Although the initial version of GFS only supported simple replication, today's version (Colossus) has added support for more space-efficient Reed-Solomon codes, which tend to reduce the space overhead of replication by roughly a factor of two over simple replication for the same level of availability. An important factor in maintaining high availability is distributing file chunks across the whole cluster in such a way that a small number of correlated failures is extremely unlikely to lead to data loss. GFS takes advantage of knowledge about the known possible correlated fault scenarios and attempts to distribute replicas in a way that avoids their co-location in a single fault domain. Wide distribution of chunks across disks over a whole cluster is also key for speeding up recovery. Since replicas of chunks in a given disk are spread across possibly all machines in a storage cluster, reconstruction of lost data chunks is performed in parallel at high speed. Quick recovery is important since long recovery time windows leave under-replicated chunks vulnerable to data loss should additional faults hit the cluster. A comprehensive study of availability in distributed file systems at Google can be found at Ford et al [99]. A good discussion of the evolution of file system design at Google can also be found at McKusik and Quinlan [100].

3.2.2 Structured WSC Storage

The simple file abstraction of GFS and Colossus may be sufficient for systems that manipulate large blobs of data, but application developers also need the WSC equivalent of database-like functionality, where data sets can be structured and indexed for easy small updates or complex queries. Structured distributed storage systems such as Google's BigTable[13] and Amazon's Dynamo[20] were designed to fulfill those needs. When compared to traditional database systems, BigTable and Dynamo sacrifice some features such as richness of schema representation and strong consistency in favor of higher performance and availability at massive scales. BigTable, for example, presents a simple multi-dimensional sorted map consisting of row keys (strings) associated with multiple values organized in columns, forming a distributed sparse table space. Column values are associated with timestamps in order to support versioning and time-series.

The choice of eventual consistency in BigTable and Dynamo shifts the burden of resolving temporary inconsistencies to the applications using these systems. A number of application developers within Google have found it inconvenient to deal with weak consistency models and the limitations of the simple data schemes in BigTable. Second generation structured storage systems such as MegaStore [105] and subsequently Spanner [96] have been designed to address such concerns. Both MegaStore and Spanner provide richer schemas and SQL-like functionality while providing simpler, stronger consistency models. MegaStore sacrifices write throughput in order to provide synchronous replication. Spanner uses a new time base API to efficiently serialize globally-distributed transactions, and therefore also providing a simpler consistency model to applications that need seamless wide-area replication for fault tolerance.

At the other end of the structured storage spectrum from Spanner are systems that are aimed almost exclusively at high performance. Such systems tend to lack support for transactions or geographic replication, use simple key-value data models, and may have loose durability guarantees. Memcached [**Error! Reference source not found.**],

developed as a distributed DRAM-based object caching layer for LiveJournal.com is a popular example at the simplest end of the spectrum. The Stanford RAMCloud [107] system also uses a distributed DRAM-based data store but aims at much higher performance (over 1 million lookup operations per second per server) as well as durability in the presence of storage node failures. The FAWN [108] system also presents a key-value high-performance storage system but uses instead NAND Flash as the storage medium, and focuses also on energy efficiency.

3.2.3 Interplay of storage and networking technology

The success of WSC distributed storage systems can be partially attributed to the evolution of datacenter networking fabrics. Ananthanarayanan et al [109] observe that the gap between networking and disk performance has widened to the point that disk locality is no longer relevant in intra-datacenter computations. This observation enables dramatic simplifications in the design of distributed disk-based storage systems as well as utilization improvements since any disk byte in a WSC facility can in principle be utilized by any task regardless of their relative locality.

The emergence of Flash as a viable enterprise storage device technology for distributed storage systems presents a new challenge for datacenter networking fabrics. A single enterprise Flash device can achieve well over 100x the operations throughput of a disk drive. Such performance levels will stretch not only datacenter fabric bisection bandwidth but also require more CPU resources in storage nodes to process storage operations at such high rates. With today's fabrics Flash locality is therefore still rather relevant.

3.3 WSC Networking

Servers must be connected, and as the performance of servers increases over time, the demand for inter-server bandwidth naturally increases as well. But while we can double the aggregate compute capacity, or double the aggregate storage simply by doubling the number of compute or storage elements, networking has no straightforward horizontal scaling solution. Doubling leaf bandwidth is easy--with twice as many servers we'll have twice as many network ports and thus twice as much bandwidth. But if we assume that every server wants to talk to every other server, we need to double not just leaf bandwidth but *bisection bandwidth*, the bandwidth across the narrowest line that equally divides the cluster into two parts. (Using bisection bandwidth to characterize network capacity is common since randomly communicating processors must send about half the bits across the "middle" of the network.)

Unfortunately, doubling cross-section bandwidth is difficult because we can't just buy (or make) a bigger switch. Switch chips are pin- and power-limited in their size; for example, as of 2013, the largest switch chip sports a bisection bandwidth of about 1 Tbps (100 10Gbps ports) and no chips are available that can do 10 Tbps. We can build larger switches by cascading these switch chips, typically in the form of a fat tree or CLOS network as shown in **Figure 3-3** [110, **Error! Reference source not found.**].

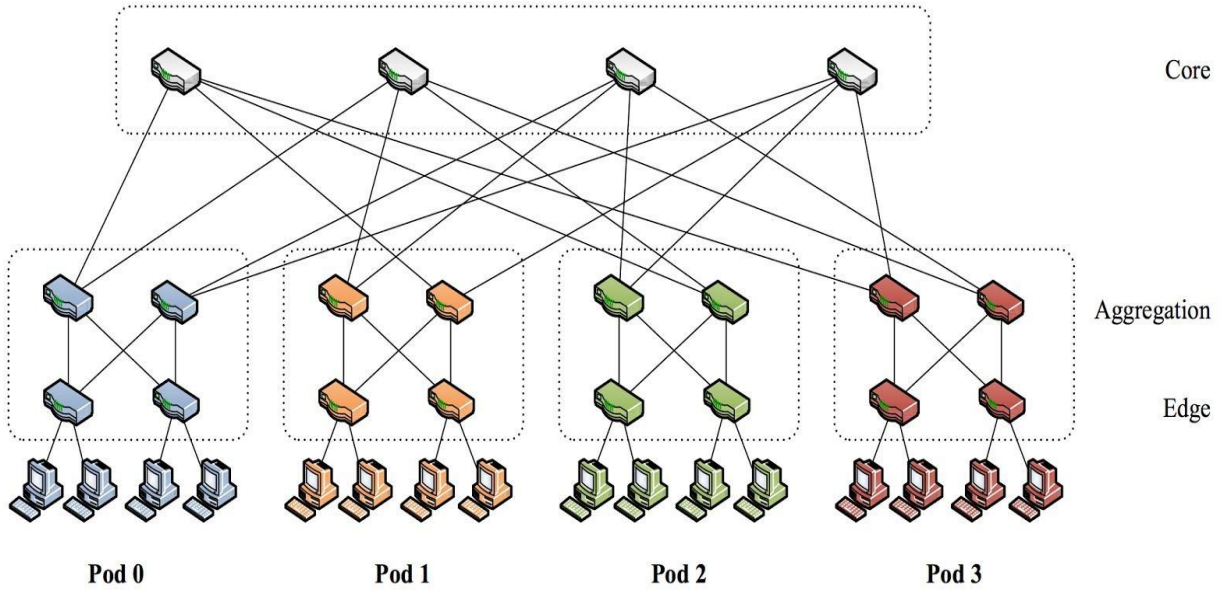

Figure 3-3: Sample three-stage fat tree topology. With appropriate scheduling, this tree can deliver the same throughput as a single-stage crossbar switch. (Image courtesy of Amin Vahdat)

Such a tree using k-port switches can support full throughput among $k^3/4$ servers using $5k^2/4$ switches, allowing networks with tens of thousands of ports. However, note that the cost of doing so increases significantly because each path from the server to another server now involves more ports. In the simplest network (a single stage consisting of a single central switch) each path consists of two ports: switch in, switch out. The above three-stage network quintuples that to ten ports, significantly increasing costs. So as bisection bandwidth grows, the cost per connected server grows as well. Port costs can be substantial, especially if a link spans more than a few meters, thus requiring an optical interface--today, the optical components of a 100m 10 Gbps link can easily cost several hundred dollars (including the cost of the two optical ports, fiber cable, fiber termination and installation), not including the networking components themselves (switches, NICs).

To avoid paying a thousand dollars or more in networking costs per machine, WSC implementors often reduce costs by oversubscribing the network at the top-of-rack switch. Generally speaking, a rack contains a small enough number of servers so they can be connected with a switch at a reasonable cost--it's not hard to find switches with 48 ports to interconnect 48 servers at full speed (say, 10 Gbps). In a full fat tree, each such switch would need the same number of ports facing "upwards" into the cluster fabric: the edge switches in the figure above devote half their ports to connecting servers, and half to the fabric. All those upwards-facing links in turn require more links in the aggregation and core layers, leading to an expensive network. In an oversubscribed network, we increase that 1:1 ratio between server and fabric ports. For example, with 2:1 oversubscription we only build a fat tree for half the bandwidth, reducing the size of the tree and thus its cost, but also reducing the available bandwidth per server by a factor of two. Each server can still

peak at 10 Gbps of traffic, but if all servers are simultaneously sending traffic they'll only be able to average 5 Gbps. In practice, oversubscription ratios of 4 to 10 are common. For example, a 48-port switch could connect 40 servers to 8 uplinks, for a 5:1 oversubscription (2 Gbps per server).

Another way to tackle network scalability is to offload some traffic to a special-purpose network. For example, if storage traffic is a big component of overall traffic, we could build a separate network to connect servers to storage units. If that traffic is more localized (not all servers need to be attached to all storage units) we can build smaller-scale networks, thus reducing costs. Historically, that's how all storage was networked: a SAN (storage area network) connected servers to disks, typically using FibreChannel networks rather than Ethernet. Today, Ethernet is becoming more common since it offers comparable speeds, and protocols such as FCoE (FibreChannel over Ethernet) and iSCSI (SCSI over IP) allow Ethernet networks to integrate well with traditional SANs.

Compared to WSCs, HPC supercomputer clusters often have a much lower ratio of computation to network bandwidth, because applications such as weather simulations distribute their data across RAM in all nodes, and nodes need to update neighboring nodes after performing relatively few floating-point computations. As a result, traditional HPC systems have used proprietary interconnects with leading-edge link bandwidths, much lower latencies (especially for common functions like barrier synchronizations or scatter/gather operations, which often are directly supported by the interconnect), and some form of a global address space (where the network is integrated with CPU caches and virtual addresses). Typically, such interconnects offer throughputs that are at least an order of magnitude higher than contemporary Ethernet or Infiniband solutions, but also much more expensive.

WSCs using VMs (or, more generally, task migration) pose further challenges to networks since connection endpoints (i.e., IP address/port combinations) can move from one physical machine to another. Typical networking hardware as well as network management software doesn't anticipate such moves and in fact often explicitly assume that they're not possible. For example, network designs often assume that all machines in a given rack have IP addresses in a common subnet, which simplifies administration and minimizes the number of required forwarding table entries routing tables. More importantly, frequent migration makes it impossible to manage the network manually--programming network elements needs to be automated, so the same cluster manager that decides the placement of computations also needs to update the network state. This need has led to much interest in OpenFlow [http://www.openflow.org/], which moves the network control plane out of the individual switches into a logically centralized controller.

For more details on cluster networking we refer to two excellent recent overview papers by Vahdat et al [**Error! Reference source not found.**] and Abts and Felderman [112].

4 Datacenter Basics

A datacenter is a building (or buildings) designed to house computing and storage infrastructure in a variety of networked formats. Its main function is to deliver the utilities needed by housed equipment and personnel: Power, cooling, shelter, and security. By classic definitions, there is little "work" produced by the datacenter. Other than some departing photons, the remainder of the energy is exclusively converted into heat. The delivery of input energy and subsequent removal of waste heat are at the heart of the datacenter's design and drive the vast majority of costs. These costs are proportional to the amount of power delivered and typically run in the $10–20/Watt range (see Section 6.1) but can vary considerably depending on size, location, and design.

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 [27] (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. Few datacenters today exceed 30 MW of critical capacity.

4.1 DATACENTER TIER CLASSIFICATIONS AND SPECIFICATIONS

The design of a datacenter is often classified as belonging to "Tier I–IV" [69]. The Uptime Institute, a professional services organization specializing in datacenters, and the Telecommunications Industry Association (TIA), an industry group accredited by ANSI and made up of approximately 400 member companies, both advocate a 4-tier classification loosely based on the power distribution, uninterruptible power supply (UPS), cooling delivery and redundancy of the datacenter [113, 114].

- Tier I datacenters have a single path for power distribution, UPS, and cooling distribution, without redundant components.
- Tier II adds redundant components to this design (N + 1), improving availability.
- Tier III datacenters have one active and one alternate distribution path for utilities. Each path has redundant components and are concurrently maintainable, that is, they provide redundancy even during maintenance.
- Tier IV datacenters have two simultaneously-active power and cooling distribution paths, redundant components in each path, and are supposed to tolerate any single equipment failure without impacting the load.

The Uptime Institute's specification is generally performance-based (with notable exceptions for the amount of backup diesel fuel, water storage, and ASHRAE temperature design points [115]). The specification describes topology rather than prescribing a specific list of components to meet the requirements, so there are many architectures that can achieve a given tier classification. In contrast, TIA-942 is very prescriptive and specifies a variety of implementation details such as building construction, ceiling height, voltage levels, types of racks, and patch cord labeling, for example.

Formally achieving tier classification certification is difficult and requires a full review from one of the granting bodies, and most datacenters are not formally rated. Most commercial datacenters fall somewhere between tiers III and IV, choosing a balance between construction cost and reliability. Generally, the lowest of the individual subsystem ratings (cooling, power, etc) determines the overall tier classification of the datacenter.

Real-world datacenter reliability is strongly influenced by the quality of the organization running the datacenter, not just by the design. The Uptime Institute reports that over 70% of datacenter outages are the result of human error, including management decisions on staffing, maintenance, and training [116]. Theoretical availability estimates used in the industry range from 99.7% for tier II datacenters to 99.98% and 99.995% for tiers III and IV, respectively [117].

4.2 DATACENTER POWER SYSTEMS

Figure 4-1 shows the components of one typical datacenter architecture. Power enters first at a utility substation (not shown) which transforms high voltage (typically 110kV and above) to medium voltage (typically less than 50kV). Medium voltage is used for site-level distribution to the primary distribution centers (also known as unit substations) which include the primary switchgear and medium-to-low voltage transformers (typically below 1000V).

From here, the power enters the building with the low-voltage lines going to the uninterruptible power supply (UPS) systems. The UPS switchgear will also take a second feed (at the same voltage) from a set of diesel generators that will cut in when utility power fails. An alternative is to use a flywheel/alternator assembly which is turned by an electric motor during normal operation, and couples to a diesel motor via a clutch during utility outages. In any case, the output lines from the UPS system are finally routed to the datacenter floor where they are connected to Power Distribution Units (PDUs). PDUs are the last layer in the transformation/distribution architecture and route individual circuits to the computer cabinets.

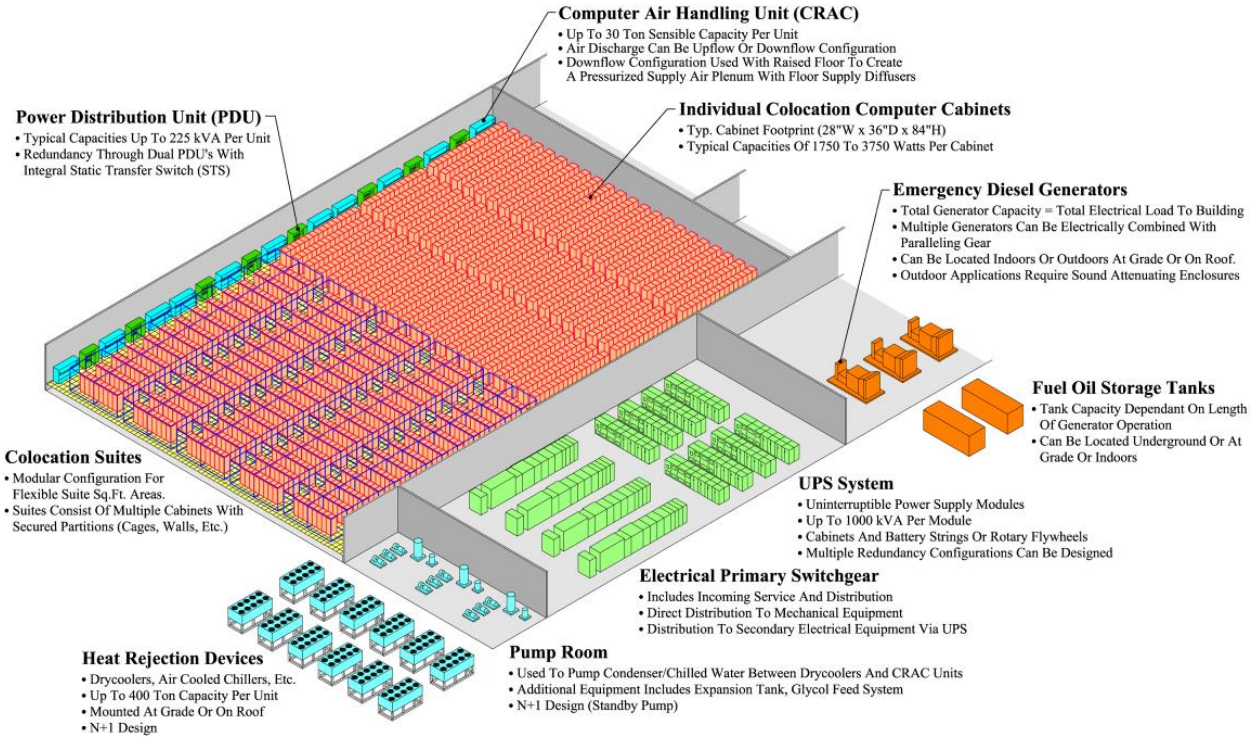

Figure 4-1: The main components of a typical datacenter (image courtesy of DLB Associates [23]).

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 seconds to start and assume the full rated load.
- Second, the UPS contains some form of energy storage (electrical or mechanical) to bridge the time between the utility failure and the availability of generator power.
- 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.

A traditional UPS employs an AC–DC–AC double conversion. Input AC is rectified to DC, which feeds a UPS-internal bus that is connected to strings of batteries. The output of the DC bus is then inverted back to AC to feed the datacenter PDUs. When utility power fails, input AC is lost but internal DC remains (from the batteries) so that AC output to the datacenter continues uninterrupted. Eventually, the generator starts and resupplies input AC power.

Because UPS systems take up a sizeable amount of space, they are usually housed in a separate room, not on the datacenter floor. Typical UPS sizes range from hundreds of kilowatts up to 2 MW or greater. Larger capacities are achieved by combining several smaller units.

Traditional double-conversion architectures are robust but inefficient, converting as much as 15% of the power flowing through them into heat. Newer designs such as line-interactive, delta-conversion, multi-mode, or flywheel systems operate at efficiencies in the range of 96-98% over a wide range of load cases. Additionally, "floating" battery architectures such as Google's on-board UPS [118] place a battery on the output side of the server's AC/DC power supply thus requiring only a small trickle charge and a simple switching circuit. These systems have demonstrated efficiencies exceeding 99%. A similar strategy is used by the OpenCompute UPS [119] which distributes a rack of batteries for every four server racks.

It's possible to use UPS systems not only in utility outages but also as a supplementary energy buffer for power and energy management. We discuss these proposals further in the next chapter.

4.2.2 Power Distribution Units

In our example datacenter, the UPS output is routed to PDUs on the datacenter floor. PDUs resemble breaker panels in residential houses but can also incorporate transformers for final voltage adjustments. They take a larger input feed and break it up into many smaller circuits that distribute power to the actual servers on the floor. Each circuit is protected by its own breaker so a short in a server or 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 a maximum of approximately 6 kW (20 or 30 A at 110–230 V). 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. The loss of one source does not interrupt power to the servers. In this scenario, the datacenter's UPS units are usually duplicated into an A and B side, so that even the failure of a UPS will not interrupt server power.

In North America, the input to the PDU is typically 480V 3-phase power. This requires the PDU to perform a final transformation step to deliver the desired 110V output for the servers, thus introducing another source of inefficiency. In the EU, input to the PDU is typically 400V 3-phase power. By taking power from any single phase to neutral combination, it is possible to deliver a desirable 230V without an extra transformer step. Using the same trick in North America requires computer equipment to accept 277V (as derived from the 480V input to the PDU), which unfortunately exceeds the upper range of standard power supplies.

Real-world datacenters contain many variants of the simplified design described here. These include the "paralleling" of generators or UPS units, an arrangement where multiple devices feed a shared bus so 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 (fully redundant pairs).

4.2.3 Alternative: DC Distribution

The use of high-voltage DC (HVDC) on the utility grid presents advantages in connecting incompatible power grids, providing resistance to cascading failures, long-distance transmission efficiency, etc. In datacenters, the case for DC distribution is centered around efficiency improvements, increased reliability from reduced component counts, and easier integration of distributed generators with native DC output.

In comparison with the double-conversion UPS mentioned above, DC systems eliminate the final inversion step of the UPS. If the voltage is selected to match the DC primary stage of the server power supply unit (PSU), three additional steps are eliminated: The PDU transformation, the PSU rectification, and the PSU power factor correction. Figure 4-2 compares the efficiency of these two systems.

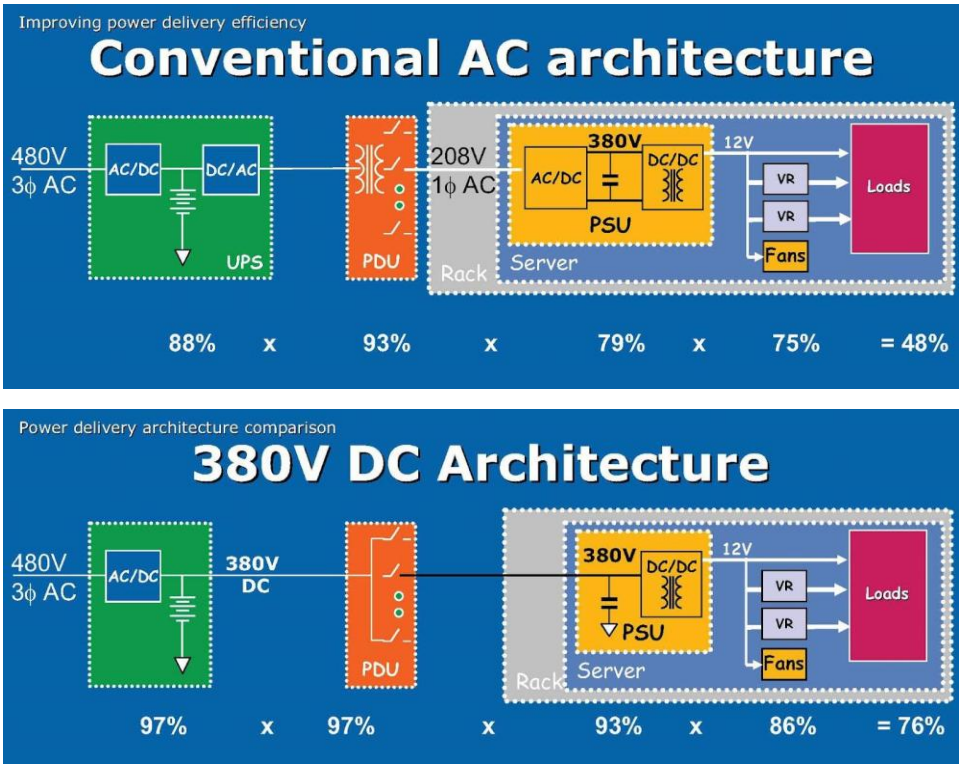

Figure 4-2: Traditional AC versus DC distribution efficiency for a data center application (Source: Pratt, Kumar, Bross, Aldridge, "Powering Compute Platforms in High Efficiency Data Centers," IBM Technology Symposium, 2006.)

The gains are significant.

However, current state-of-the-art UPS technologies are greatly improved with typical efficiencies in the 94-97% range for a variety of load factors. As mentioned above, by also choosing an output voltage of 400VAC from the UPS we can eliminate the PDU transformer. **Figure 4-3** shows the efficiency of such a system.

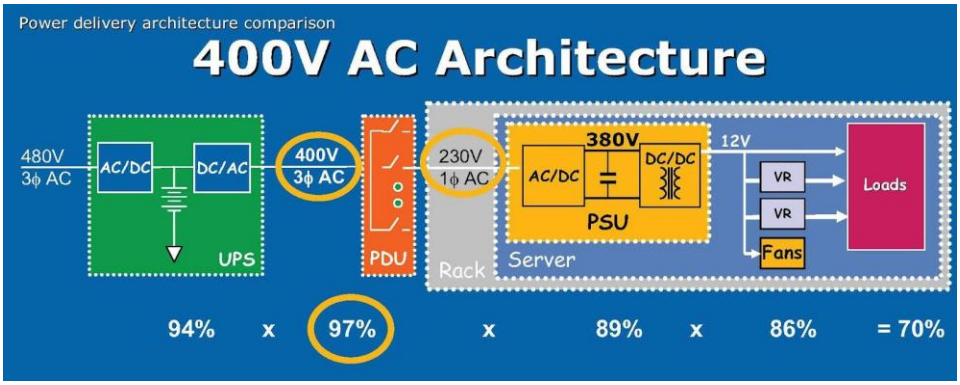

Figure 4-3: Efficiency of 400VAC distribution architecture (Source: Pratt, Kumar, Bross, Aldridge, "Powering Compute Platforms in High Efficiency Data Centers," IBM Technology Symposium, 2006.)

"Floating" battery architectures can offer further improvements by virtually eliminating UPS losses. Figure 4-4 shows the efficiency of a typical Google design using a "floating" UPS downstream of the server power supply. Despite using standard AC power distribution parts up to the server chassis, such a power architecture can exceed the efficiency of an all-DC architecture.

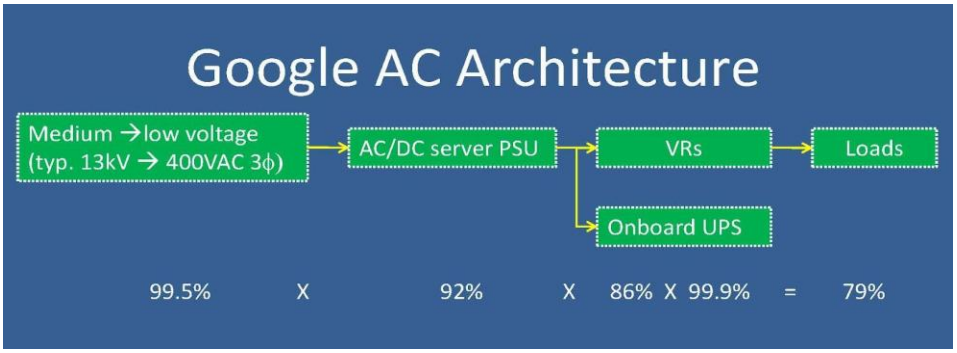

Figure 4-4: Efficiency diagram of Google's AC distribution for data centers

Commercial DC equipment is available, but costs are currently higher than comparable AC equipment. Similarly, the construction of large datacenters involves hundreds, and sometimes thousands of skilled workers. While only a subset of these will be electricians, the availability of DC technicians may lead to increased construction and service/operational costs.

However, DC power distribution is more attractive when integrating distributed power generators such as solar photovoltaic, fuel cells, and wind turbines. These power sources typically produce native DC and are easily integrated into an DC power distribution architecture.

4.3 DATACENTER COOLING SYSTEMS

Datacenter cooling systems remove the heat generated by all equipment. To remove heat, a cooling system must employ some hierarchy of loop systems, each bringing in a cold medium that warms up via some form of heat exchange and is somehow cooled back again. An open loop system replaces the outgoing warm medium with a cool supply from the outside, so that each cycle through the loop uses new material. A closed-loop system recirculates the same medium again and again, transferring heat to an adjacent upper loop in a heat exchanger, and eventually the environment. All systems must contain a loop to the outside environment for ultimate heat rejection.

The simplest topology is fresh air cooling (or air economization) — essentially, opening the windows. Such a system is shown in **Figure 4-5**. This is a single, open loop system that we will discuss in more detail in the section on free cooling.

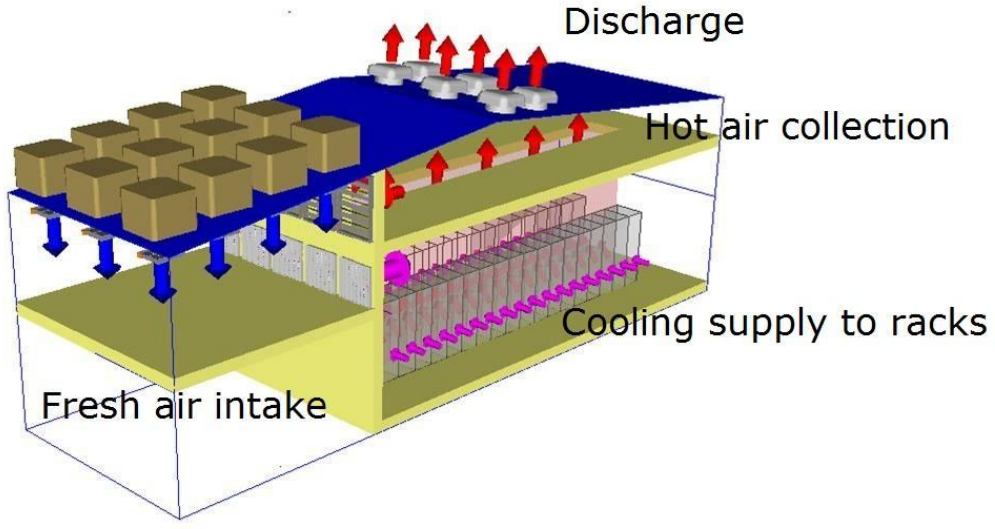

Figure 4-5: Airflow schematic of an air-economized datacenter (Source: Perspectives, James Hamilton's blog)

Closed loop systems come in many forms, the most common being the air circuit on the datacenter floor. Its function is to remove heat from the servers and transport it to a heat

exchanger. As shown in Figure 4.2, cold air flows to the servers, heats up, and eventually reaches a heat exchanger to cool it down again for the next cycle through the servers.

Typically, datacenters employ raised floors, concrete tiles installed onto a steel grid resting on stanchions 2–4 ft above the slab floor. The underfloor area often contains power cables to racks, but its primary purpose is to distribute cool air to the server racks. The airflow through the underfloor plenum, the racks, and back to the CRAC (*CRAC* is a 1960s term for *computer room air conditioning*) defines the primary air circuit.

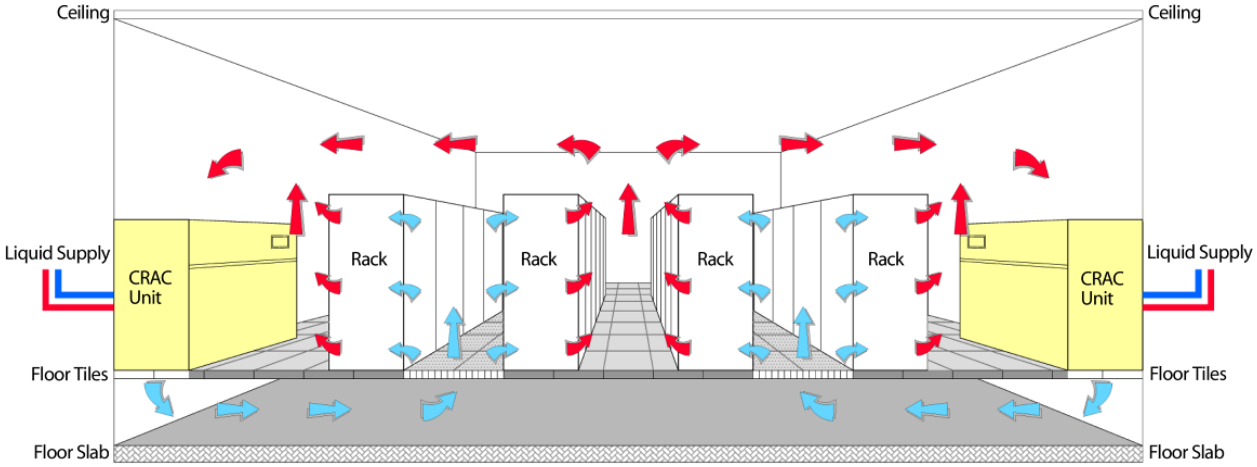

Figure 4-6: Datacenter raised floor with hot–cold aisle setup (image courtesy of DLB Associates [24]).

The simplest closed loop systems contain two loops. The first loop is the air circuit shown in **Figure 4-6**, and the second loop (i.e., the liquid supply to the CRACs) leads directly from the CRAC to external heat exchangers (typically placed on the building roof) that discharge the heat to the environment.

A three-loop system is shown in **Figure 4-7**. The CRACs (shown in blue) receive chilled water (called Process Chilled Water Supply or PCWS) from an intermediate circuit containing a chiller. The chiller exchanges the heat into a condenser water loop which is open to the atmosphere through cooling towers (We will discuss cooling towers in more detail below). The condenser water loop gets its name because it rejects the heat coming from the condenser side of the chiller.

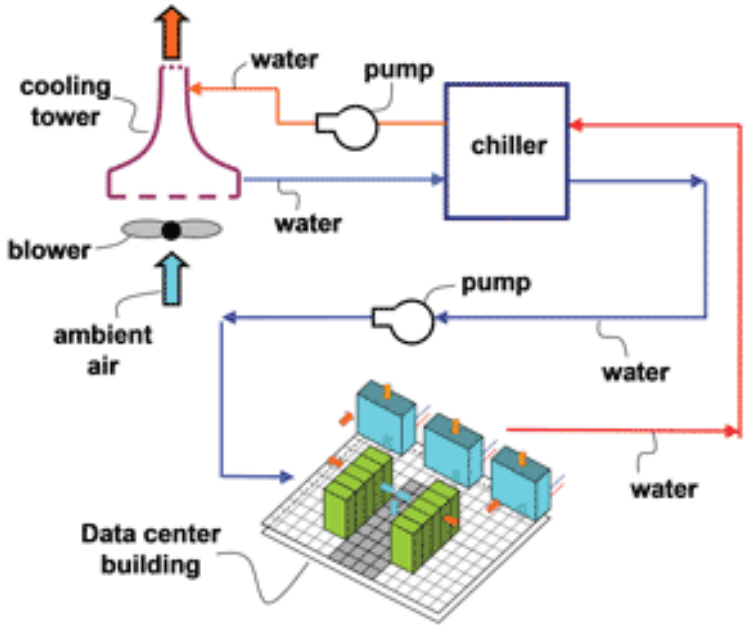

Figure 4-7: 3-loop datacenter cooling system (Source: Iyengar, Schmidt, Energy Consumption of Information Technology Data Centers)

Each topology presents tradeoffs in complexity, efficiency, and cost. For example, fresh air cooling can be very efficient but does not work in all climates, does not protect from airborne particulates, and can introduce complex control problems. Two loop systems are easy to implement, are relatively inexpensive to construct, and offer protection from external contamination, but typically have lower operational efficiency. A three loop system is the most expensive to construct and has moderately-complex controls, but offers contaminant protection and good efficiency when employing economizers.

Additionally, generators (and sometimes UPS units) back up most mechanical cooling equipment 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 supported by generators, thus significantly adding to the overall construction cost.

4.3.1 CRACs, Chillers, and Cooling Towers

CRACs, chillers, and cooling towers are some of the most important building blocks in datacenter cooling systems and we take a slightly closer look at each.

4.3.2 CRACs

All CRACs contain a heat exchanger, air mover, and controls. They mostly differ by the type of cooling they employ:

- Direct expansion (DX)
- Fluid solution

- Water cooled

A DX unit is a split air conditioner with cooling (evaporator) coils inside the CRAC, and heat-rejecting (condenser) coils outside the datacenter. The fluid solution CRAC shares this basic architecture but uses a water/glycol mixture flowing through its coils rather than a phase-change refrigerant. Finally, a water cooled CRAC connects to a chilled water loop.

CRAC units pressurize the raised floor plenum by blowing cold air into the underfloor space which then escapes through perforated tiles in front of the server racks. The air flows through the servers and is expelled into a "hot aisle". Racks are typically arranged in long rows that alternate between cold and hot aisles to reduce inefficiencies caused by mixing hot and cold air. In fact, many newer datacenters physically isolate the cold or hot aisles with walls [65].) As shown in **Figure 4-6**, the hot air produced by the servers recirculates back to the intakes of the CRACs where it is cooled and exhausted into the raised floor plenum again.

4.3.3 Chillers

A water-cooled chiller as shown in Figure 4-8 can be thought of as a water-cooled air conditioner.

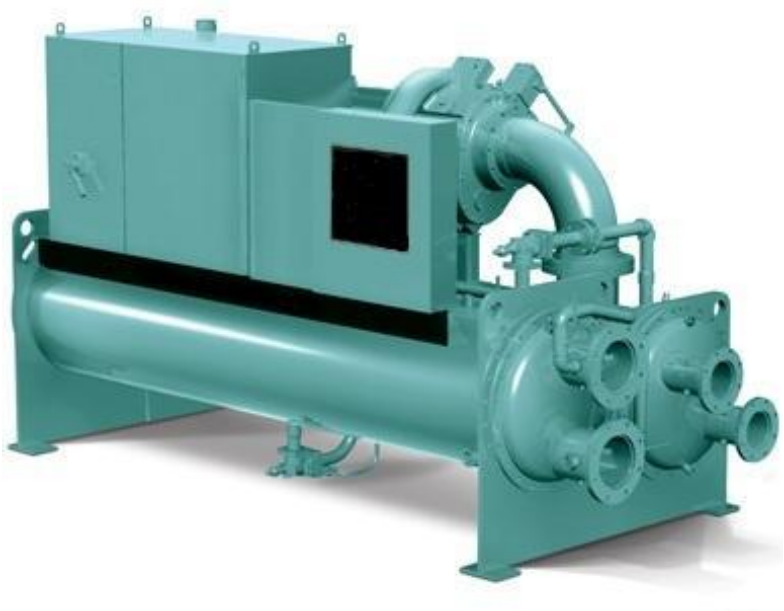

Figure 4-8: Water-cooled centrifugal chiller

It submerges the evaporator and condenser coils in water in two large, separate compartments which are joined via the top-mounted refrigeration system consisting of a compressor, expansion valve, and piping. As water flows over the submerged coils, it is cooled or heated depending on which side of the chiller it is on.

In the cold compartment, warm water from the datacenter floor enters and gets cooled down prior to returning to the PCWS loop. In the hot compartment, cool water from the condenser water loop enters and carries the heat away to the cooling towers for evaporation

cooling (rejection to the environment). Since the chiller uses a compressor, it consumes a significant amount of energy to perform its work.

4.3.4 Cooling towers

Cooling towers cool off a water stream by evaporating a portion of it into the atmosphere. The energy required to change the liquid into a gas is known as the latent heat of vaporization and the temperature of the water can be dropped significantly given favorable dry conditions. The water flowing through the tower comes directly from the chillers or from another heat exchanger connected to the PCWS loop. Figure 4-9 illustrates how it works.

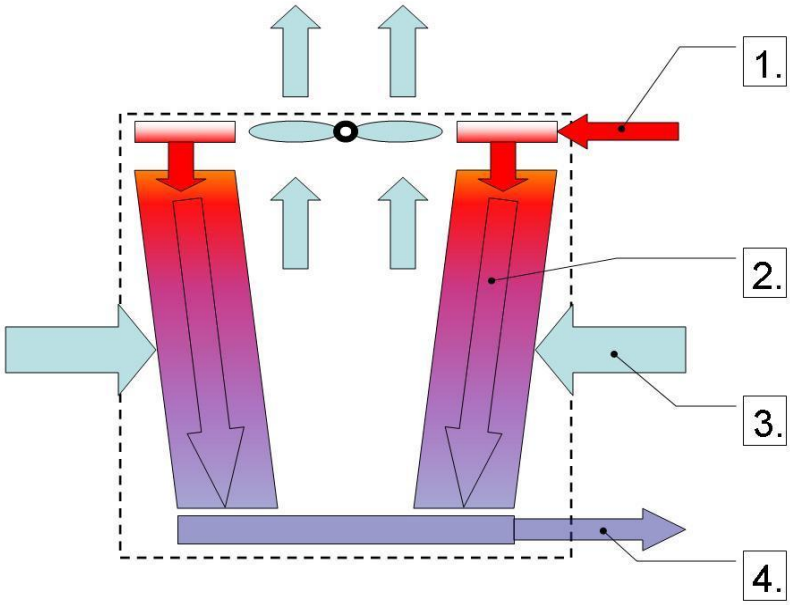

Figure 4-9: How a cooling tower works:

1 Hot water from the data center flows from the top of the cooling tower onto "fill" material inside the tower. The fill creates additional surface area to improve evaporation performance.

2 As the water flows down the tower, some of it evaporates, drawing energy out of the remaining water, thus cooling it down.

3 A fan on top draws air through the tower to aid evaporation. Dry air enters the sides and humid air exits the top.

4 The cool water is collected at the base of the tower and returned to the data center.

Cooling towers work best in temperate climates with low humidity; ironically, they do not work as well in very cold climates because they need additional mechanisms to prevent ice formation on the towers and in the pipes.

4.3.5 Free Cooling

"Free cooling" is not really free, but it's very efficient in comparison to using a chiller. Free cooling uses low external temperatures to help produce chilled water or uses outside air to cool servers.

As mentioned above, air-economized datacenters are open to the external environment and use low dry bulb temperatures for cooling. (The dry bulb temperature is what a conventional thermometer or weather report shows.) Large fans push outside air directly into the room or the raised floor plenum when outside temperatures are within limits (for an extreme experiment in this area, see [2]). Once the air flows through the servers, it is expelled outside the building. An air-economized system can be very efficient but can't easily control contamination, may require auxiliary cooling (when external conditions are not favorable), and may be difficult to control. Specifically, if there is a malfunction, temperatures will rise very quickly since there is little thermal storage in the cooling loop. In comparison, a water-based system can use a water storage tank to provide a thermal buffer.

Water-economized datacenters take advantage of the wet bulb temperature [120]. The wet bulb temperature is the lowest temperature that can be reached by the evaporation of water only. The dryer the air, the bigger the difference between dry bulb and wet bulb temperatures; the difference can exceed ten degrees Celsius, and thus a water-economized datacenter can run without chillers for many more hours per year.

Typical water-economized datacenters employ a parallel heat exchanger so that the chiller can be turned off when it's cool enough outside. Depending on the capacity of the cooling tower (which depends on outside air conditions), a control system balances water flow between the chiller and the cooling tower.

Yet another approach uses a radiator instead of a cooling tower, pumping the condenser fluid or process water through a fan-cooled radiator. Similar to the glycol/water-based CRAC, such systems use a glycol-based loop to avoid freezing. Such radiators work well in cold climates (say, a winter in Chicago) but less well at moderate or warm temperatures because the achievable cold temperature is limited by the external dry bulb temperature, and because convection heat transfer is less efficient than evaporation.

4.3.6 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 adjust the number of perforated tiles by replacing solid tiles with perforated ones or vice versa. For cooling to work well, the cold airflow coming through the tiles should match the horizontal airflow through the servers in the rack. For example, if a rack has 10 servers with an airflow of 100 cubic feet per minute (CFM) 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, the cool air will be used by some of the servers while others will ingest recirculated air from above the rack or other leakage paths.

Figure 4-10 shows the results of a Computational Fluid Dynamics (CFD) analysis for a rack that is oversubscribing the datacenter's airflow.

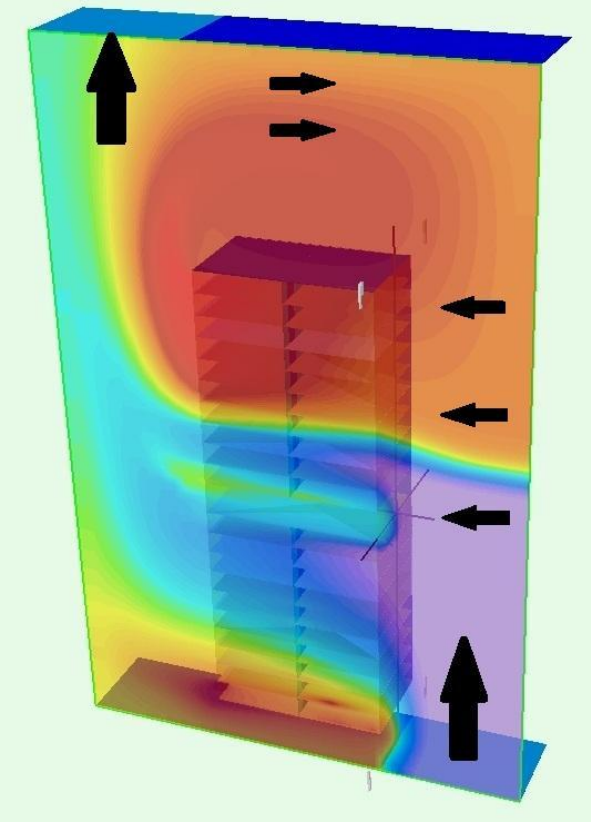

Figure 4-10: CFD model showing recirculation paths and temperature stratification for a rack with under-provisioned airflow**.**

In this example, recirculation across the top of the rack causes the upper servers to ingest warm air. The servers on the very bottom are also affected by a recirculation path under the rack. Blockages from cable management hardware cause a moderate warm zone about halfway up the rack.

The facility manager's typical response to such a situation is to lower the temperature of the CRAC output. That works but increases energy costs significantly, so it's better to fix the underlying problem instead and physically separate cold and warm air as much as possible, while optimizing 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 [65].

Air flow limits the power density of datacenters. For a fixed temperature differential across a server, a rack's airflow requirement increases with 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 needed in the underfloor plenum. At low densities this is easy to accomplish, but at some point the laws of physics start to catch up 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.

4.3.7 In-Rack, In-Row Cooling, and Cold Plates

In-rack cooling can 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 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-rack cooling might remove just part of the heat, or all heat, effectively replacing the CRACs. Obviously, chilled water needs to be brought to each rack, greatly increasing the cost of plumbing. Some operators may also worry about having water on the datacenter floor since leaky coils or accidents might spill water on equipment.

In-row cooling works like in-rack cooling except the cooling coils aren't in the rack, but adjacent to the rack. A capture plenum directs the hot air to the coils and prevents leakage into the cold aisle. Figure 4-11 shows an in-row cooling product and how it is placed between racks.

Figure 4-11: In row air conditioner

Finally, we can directly cool server components using cold plates, i.e., local, liquid-cooled heat sinks. It is usually impractical to cool all the components with cold plates, so instead we target the highest power dissipaters like CPUs, and use air to remove the remainder of the heat. Although cold plates can remove high local heat loads, they are relatively rare because of the cost and complexity of designing the plates and couplings needed to connect and disconnect the server loop from the larger rack loop, and the risk of having water in close proximity to the servers.

4.3.8 Case Study: Google's In-row Cooling

Google's "hot hut" is an in-row cooling system that removes all heat produced by racks, replacing conventional CRACs. Figure 4-12 and Figure 4-13 show a Google fan coil unit and row segment, respectively. The large arrows depict airflow direction.

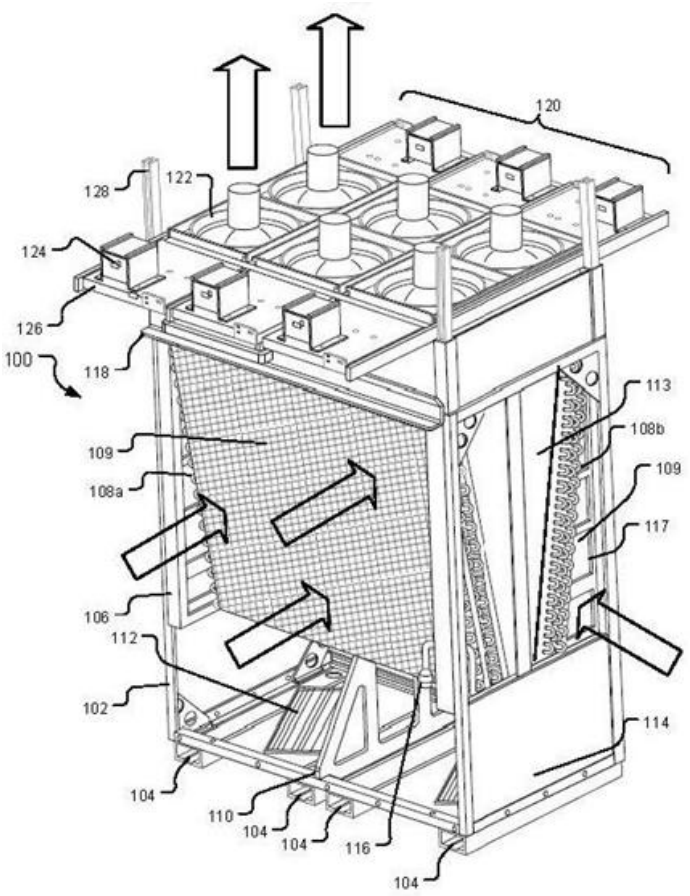

Figure 4-12: Google's "Hot Hut", an in-row, water-cooled fan coil unit.

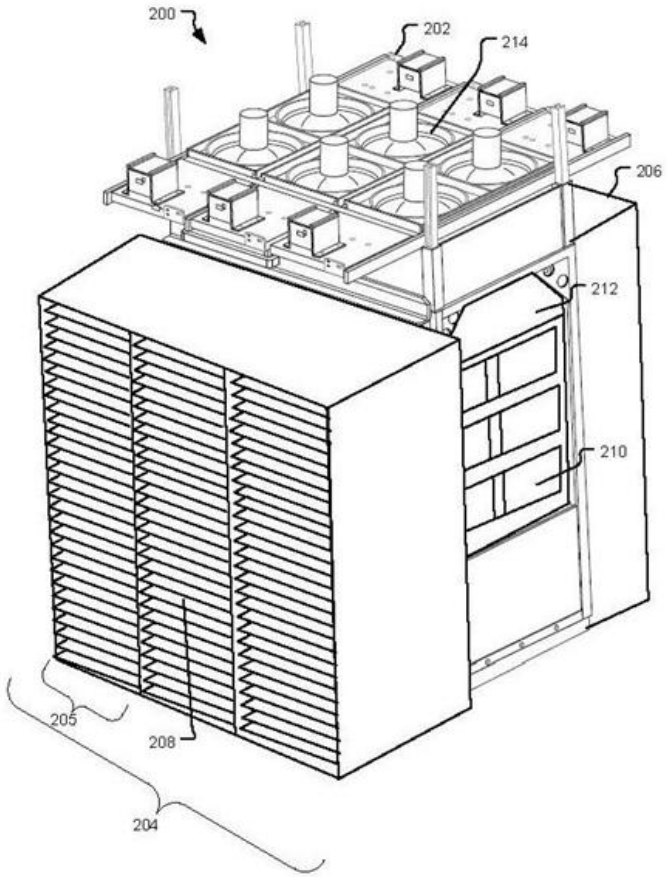

Figure 4-13: Racks are placed against the "Hot Hut" creating a warm air capture plenum.

Hot Huts form a row, with spacing determined by the desired cooling density. Racks are placed in front of them and exhaust their hot air into the Hot Huts. Air can also move laterally in a row, allowing a level of sharing for the cooling resources. Coupled with a water-economized cooling plant, this 3-loop topology is very efficient, using about 10% of the datacenter load to power the entire cooling system. Moreover, water economization enables chiller-less designs such as Google's datacenter in Belgium.

4.3.9 Container-Based Datacenters

Container-based datacenters go one step beyond in-row cooling by placing the server racks into a container (typically 20 or 40 ft long) and integrating heat exchange and power distribution into the container as well. Similar to in-row cooling, the container needs a supply of chilled water and uses coils to remove all heat. Close-coupled air handling 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. **Figure 4-14** shows an isometric cutaway of Google's container design.

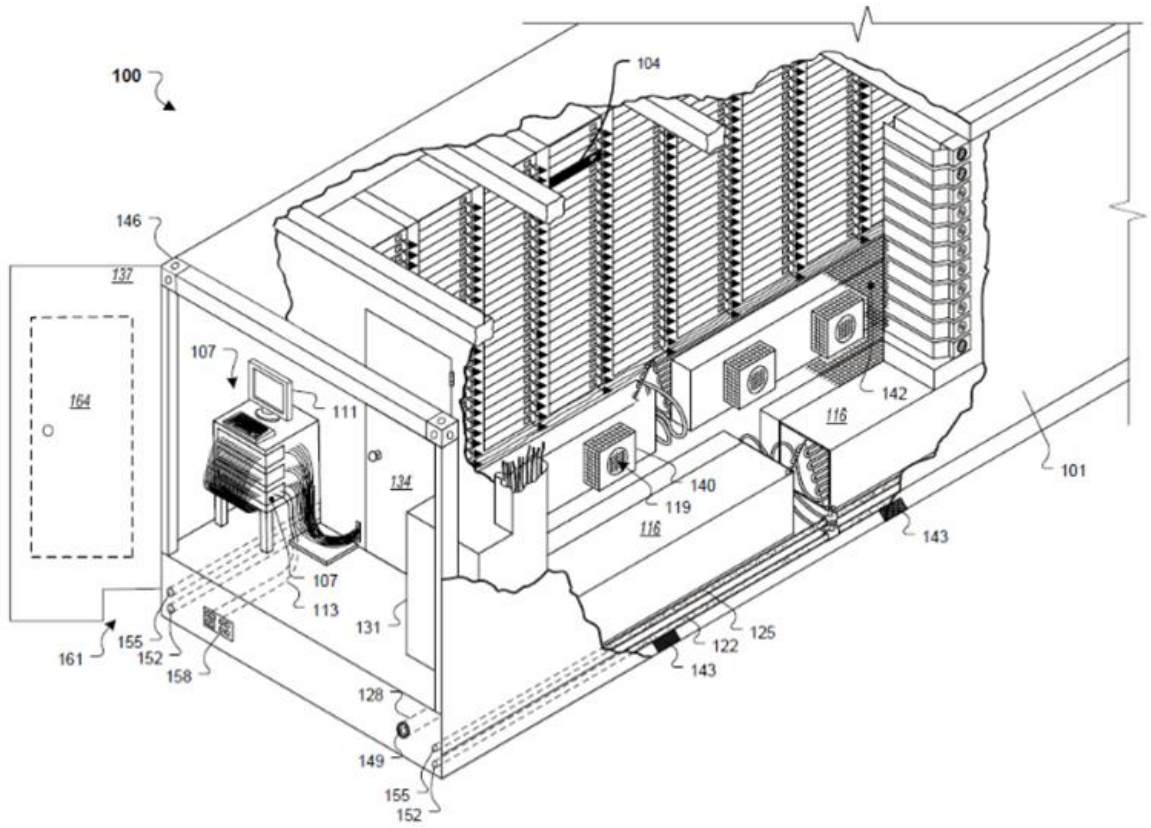

Figure 4-14: Google's container design includes all the infrastructure of the datacenter floor.

Like a regular datacenter room, they must be complemented by outside infrastructure such as chillers, generators, and UPS units to be fully functional.

To our knowledge, the first container-based datacenter was built by Google in 2005 [34], and the idea dates back to a Google patent application in 2003. This container-based facility

achieved high energy efficiency ratings, as we will discuss further in the following chapter. Figure 4-15 shows Google's container-based datacenter as viewed from the hangar bay. Today, modular designs are available from many vendors ([122,123,124,125,126,127]), and a number of operators including Microsoft [94] and eBay [121] are using containers in their facilities.

Figure 4-15: Google's container-based datacenter

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

The broadest definition of WSC energy efficiency would measure the energy used to run a particular workload (say, to sort a petabyte of data). Unfortunately, no two companies run the same workload and real-world application mixes change all the time so it is hard to benchmark real-world WSCs this way. Thus, even though such benchmarks have been contemplated as far back as 2008 [128] they haven't yet been found [129] and we doubt they ever will. However, it is useful to view energy efficiency as the product of three factors we can independently measure and optimize:

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

The first term (a) measures facility efficiency, the second server power conversion efficiency, and the third measures the server's architectural efficiency. We'll discuss these factors in the subsequent sections.

5.1.1 The PUE metric

Power usage effectiveness (PUE) reflects the quality of the datacenter building infrastructure itself [38], and captures the ratio of total building power to IT power (the power consumed by the actual computing and network equipment, etc.). (Sometimes IT power is also referred to as "critical power".)

PUE = (Facility power) / (IT Equipment power)

PUE has gained a lot of traction as a datacenter efficiency metric since widespread reporting started around 2009. We can easily measure PUE by adding electrical meters to the lines powering the various parts of a datacenter, thus determining how much power is used by chillers or a UPS.

Historically, the PUE for the average datacenter has been embarrassingly poor. According to a 2006 study [54], 85% of current datacenters were estimated to have a PUE of greater than 3.0. In other words, the building's mechanical and electrical systems consumed twice as much power as the actual computing load! Only 5% had a PUE of 2.0 or better.

A subsequent EPA survey of over 100 datacenters reported an average PUE value of 1.91 [130], and a 2012 Uptime Institute survey of over 1100 datacenters covering a range of geographies and datacenter sizes reported an average PUE value between 1.8 and 1.89 [131]. The distribution of results are shown in Figure 5-1. The study noted cold/hot aisle containment and increased cold aisle temperature as the most common improvements implemented. Large facilities reported the biggest improvements, and about half of small datacenters (<500 servers) still were not measuring PUE.

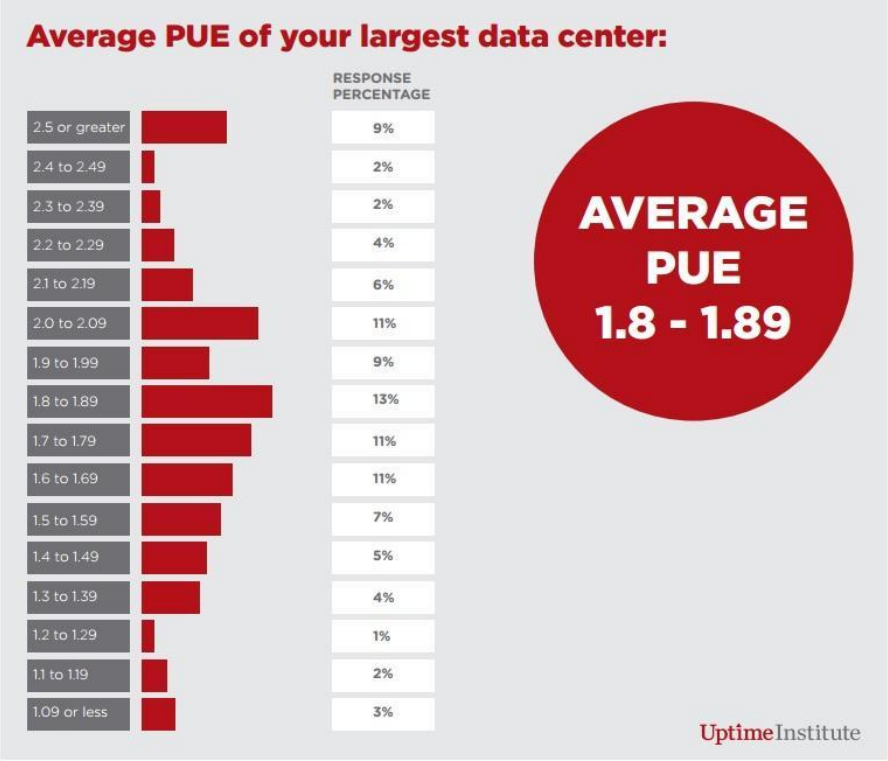

Figure 5-1: Uptime Institute survey of PUE for 1100+ datacenters

Very large operators (usually consumer Internet companies like Google, Microsoft, and eBay) have reported excellent PUE results over the past few years, typically below 1.2, though only Google has provided regular updates of their entire fleet based on a clearly defined metric [132]. At scale, it is easy to justify special attention to efficiency; for example, Google reported having saved over one billion dollars to date from energy efficiency measures [133].

5.1.2 Issues with the PUE metric

Although the Green Grid has published detailed guidelines on how to measure and report PUE [134], many published values aren't directly comparable, and sometimes PUE values are used in marketing documents to show best-case values that aren't real. The biggest factors that can skew PUE values are as follows:

- Not all PUE measurements include the same overheads. For example, some may include losses in the primary substation transformers, or losses in wires feeding racks from PDUs, whereas others may not. For example, Google reported a fleet-wide PUE of 1.12 using a comprehensive definition of overhead that includes all known sources, but could have reported a value of 1.06 with a more "optimistic" definition of overhead [135]. For PUE to be a useful metric, datacenter owners and operators should adhere to GreenGrid guidelines in measurements and reporting, and be transparent about the methods used in arriving at their results.
- Instantaneous PUEs differ from average PUEs. Over the course of a day or a year, a facility's PUE can vary considerably. For example, during a cold day it might be very low, but during the summer it might be considerably higher. Generally speaking, annual averages are most useful for comparisons.
- Some PUEs aren't real-world measurements. Often vendors publish "design" PUEs that are computed based on optimal operating conditions and nominal performance values, or publish a value measured during a short load test under optimal conditions. Typically, PUE values provided without details fall into this category.
- Some PUEs values have higher error bars because they're based on infrequent manual readings, or on coarsely placed meters that force some PUE terms to be estimated instead of measured. For example, if the facility has a single meter measuring the critical load downstream of the UPS, PDU and low-voltage distribution losses need to be estimated.

In practice, PUE values should be measured in real time. Not only does this provide a better picture of diurnal and seasonal variations, it also allows the operator to react to unusual readings during day-to-day operations. For example, someone may have left on a set of backup pumps after a periodic test, and with real-time metrics the operations team can quickly correct such problems after comparing expected vs actual PUE values.

The PUE metric has been criticized as not always indicating better energy performance, because PUEs typically worsen with decreasing load. For example, hypothetically lets say a particular datacenter runs at a PUE of 2.0 at 500kW load vs a PUE of 1.5 at 1MW load. If it's possible to run the given workload with 500kW of load (e.g., with newer servers), that

clearly is more energy efficient despite the inferior PUE. However, this criticism merely points out that PUE is just one of the three factors in the efficiency equation that started this chapter, and overall the widespread adoption of PUE measurements has arguably been the driver of the biggest improvements in datacenter efficiency in the past 50 years.

5.1.3 Sources of Efficiency Losses in Datacenters

For illustration, let us walk through the losses in a typical datacenter [42]. The first two transformation steps bring the incoming high-voltage power (>100kV) to medium-voltage distribution levels (typically 10-25kV) and, closer to the server floor to low voltage (typically 480V in US). Both steps should be very efficient, with losses typically below half a percentage point for each step. Stepping inside the building, a conventional double-conversion UPS causes most electrical losses, typically running at an efficiency of 88–94% under optimal load, significantly less if partially loaded (which is the common case). Rotary UPSes (flywheels) and high-efficiency UPSes can reach efficiencies of about 97%. The final transformation step in the PDUs accounts for an additional half-percent loss. Finally, 1–3% of power can be lost in the cables feeding low-voltage power (110 or 220V) to the racks (recall that a large facility can have a raised floor area that is >100m long or wide, so power cables can get quite long).

Most of the datacenter efficiency losses typically stem from cooling overhead, with chillers being the largest culprit.

Figure 5-2 shows the typical distribution of energy usage in a conventional datacenter with a PUE of 2.0.

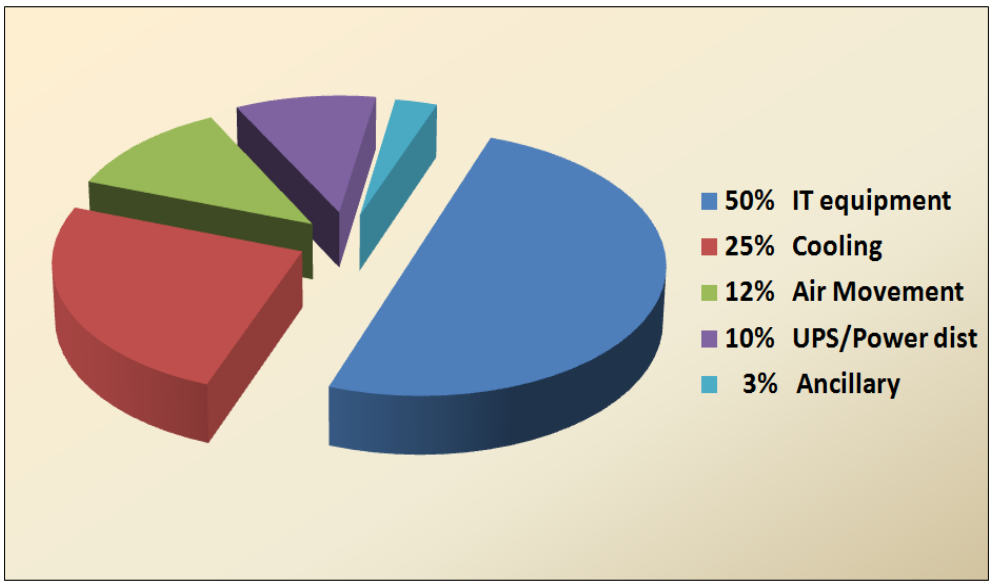

Figure 5-2: Power losses in a traditional (legacy) data center

Cooling losses are three times greater than power losses, presenting the most-promising target for efficiency improvements: if all cooling losses were eliminated, the PUE would drop to 1.26, whereas a zero-loss UPS system would only yield a PUE of 1.8. Typically, the worse a facility's PUE, the higher the percentage of the total loss coming from the cooling system

[157]. Intuitively, there are only so many ways to mess up a power distribution system, but many more ways to mess up cooling.

Much of this poor efficiency is caused by a historical lack of attention to efficiency, not by inherent limitations imposed by physics. Less than ten years ago, PUEs weren't formally used and a total overhead of 20% was considered unthinkably low, yet as of 2012 Google reported a fleet-wide annual average overhead of 12% [135] and many others are claiming similar values for their newest facilities. However, such excellent efficiency is still confined to a small set of datacenters, and most small datacenters probably haven't improved much.

5.1.4 Improving the Energy Efficiency of Datacenters

As discussed in the previous chapter, careful design for efficiency can substantially improve PUE [59][68][42]. To summarize, the key steps are:

- Careful air flow handling: segregate hot air exhausted by servers from cold air, and keep the path to the cooling coil short so that little energy is spent moving cold or hot air long distances.
- Elevated temperatures: keep the cold aisle at 25-30°C rather than 18–20°C. Higher temperatures make it much easier to cool datacenters efficiently. Virtually no server or network equipment actually needs intake temperatures of 20°C, and there is no evidence that higher temperatures cause more component failures [67,79,136].
- Free cooling: in most moderate climates, free cooling can eliminate the majority of chiller runtime or eliminate chillers altogether.
- Better power system architecture: UPS and power distribution losses can often be greatly reduced by selecting higher-efficiency gear, as discussed in the previous chapter.

In April of 2009, Google first published details of its datacenter architecture, including a video tour of a container-based datacenter built in 2005 [34]. This datacenter achieved a then (2008) state-of-the-art annual PUE of 1.24 yet differed from conventional datacenters in only a few major aspects: application of the principles listed above. In other words, while at first it may seem this datacenter is radically different from conventional ones, virtually all 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 in worst-case climates (and 1.2-1.3 in average climates), i.e., an efficiency far above the current industry average.

5.1.5 Beyond the facility

Recall the energy efficiency formula from the beginning of this chapter:

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

So far we've discussed the first term, facility overhead. The second term (b) accounts for overheads inside servers or other IT equipment using a metric analogous to PUE, server PUE (SPUE). SPUE 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. Substantial amounts of power may be lost in the server's power supply, voltage regulator modules (VRMs), and cooling fans. As shown in **Figure 5-3**, the losses inside the server can exceed those of the entire upstream datacenter power train.

SPUE measurements aren't standardized like PUE but are fairly straightforward to define. Almost all equipment contains two transformation steps: the first step transforms input voltage (typically 110-220V AC) to local DC current (typically 12V), and VRMs transform that down to the much lower voltages used by a CPU or by DRAM. (The first step requires an additional internal conversion within the power supply, typically to 380V DC, as shown in the yellow box below.) SPUE ratios of 1.6–1.8 were common just a few years ago; many power supplies were less than 80% efficient, and many motherboards use VRMs that were similarly inefficient, losing more than 25% of input power in electrical conversion losses. In contrast, a state-of-the-art SPUE should be less than 1.2 [17]; for example, current Google servers reach around 86% efficiency for an SPUE of 1.15.

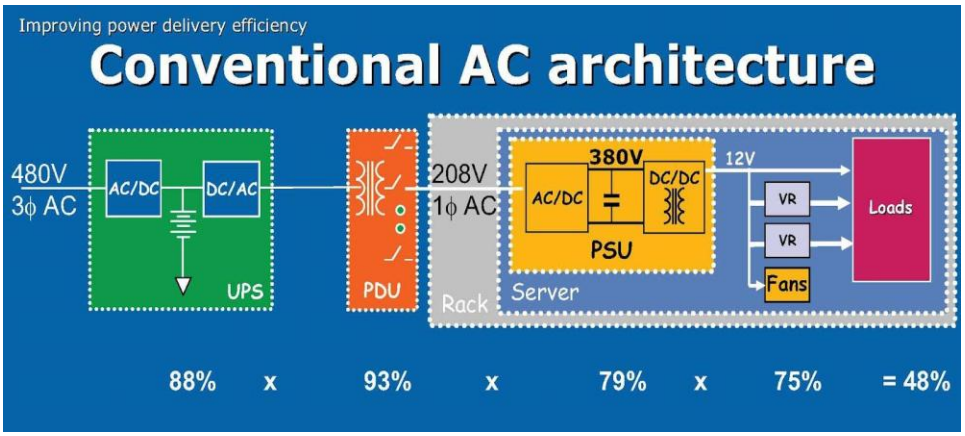

Figure 5-3: Datacenter power train losses (Source: Pratt, Kumar, Bross, Aldridge, "Powering Compute Platforms in High Efficiency Data Centers," IBM Technology Symposium, 2006.)

The product of PUE and SPUE constitutes an accurate assessment of the end-to-end electromechanical efficiency of a WSC. Such a 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 THE ENERGY EFFICIENCY OF COMPUTING

So far we have discussed efficiency in electromechanical terms, terms (a) and (b) of the efficiency equation, largely ignoring term (c) that accounts for how the electricity delivered to electronic components is actually translated into useful work. In a state of the art facility, the electromechanical components have a limited potential for improvement: Google's TPUE being approximately 1.3 means that even if we eliminate all electromechanical overheads the total energy efficiency can only improve by 24%. In contrast, the energy efficiency of computing has doubled approximately every 1.5 years in the last half century [138]. Even if such rates might decline due to CMOS scaling challenges [137], they are still expected to dwarf any electromechanical efficiency improvements. In the remainder of this chapter we focus on the energy and power efficiency of computing.

5.2.1 Measuring Energy Efficiency

Ultimately, one wants to measure the value obtained from the energy spent in computing, for example, to compare the relative efficiency of two WSCs or to guide the design choices for new systems. In high-performance computing (HPC), the Green 500 [37] benchmark ranks the energy efficiency of the world's top supercomputers using LINPACK. Similarly, server-level benchmarks such as Joulesort [82] and SPECpower [81] characterize other aspects of computing efficiency. Joulesort measures the total system energy to perform an out-of-core sort and attempts to derive a metric that enables the comparison of systems ranging from embedded devices to supercomputers. SPECpower 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. Two separate benchmarking efforts aim to characterize the efficiency of storage systems: the Emerald Program [141] by the Storage Networking Industry Association (SNIA) and the SPC-2/E [143] by the Storage Performance Council. Both benchmarks measure storage servers under different kinds of request activity and report ratios of transaction throughput per Watt.

5.2.2 Server Energy Efficiency

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 isolate efficiency differences in the hardware platform. In particular, SPEC power reporting rules help highlight a key energy usage feature of much of today's computing equipment: 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 5-4 shows the SPEC power benchmark results for the top performing entry as of June 2008, a system that represents the behavior of much of the installed base of servers today. 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!

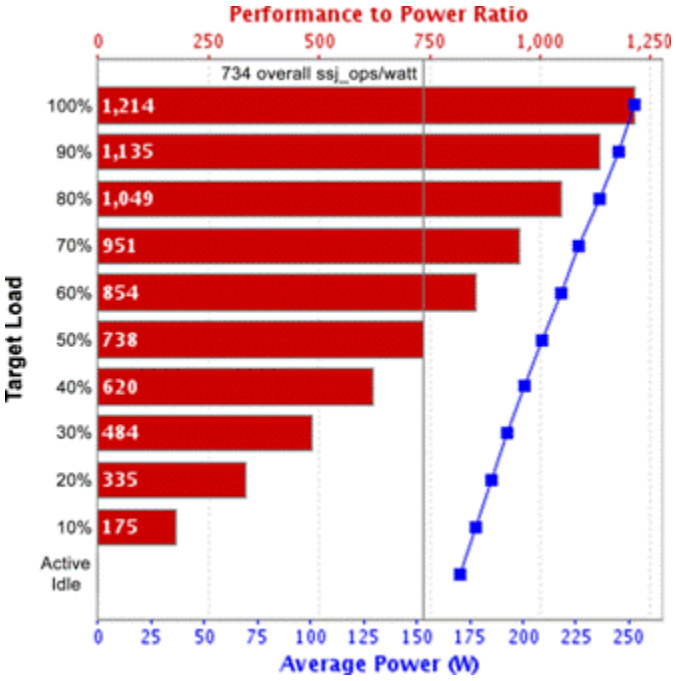

Figure 5-4: An example benchmark result for SPECpower_ssj2008; energy efficiency is indicated by bars, whereas power consumption is indicated by the line. Both are plotted for a range of utilization levels, with the average metric corresponding to the vertical dark line. The system has a single-chip 2.83 GHz quad-core Intel Xeon processor, 4 GB of DRAM, and one 7.2 k RPM 3.5” SATA disk drive.

5.2.3 Usage profile of Warehouse-scale Computers

Figure 5-5 shows the average CPU utilization of 5,000 Google servers during a 6-month period (measured circa 2007). 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.

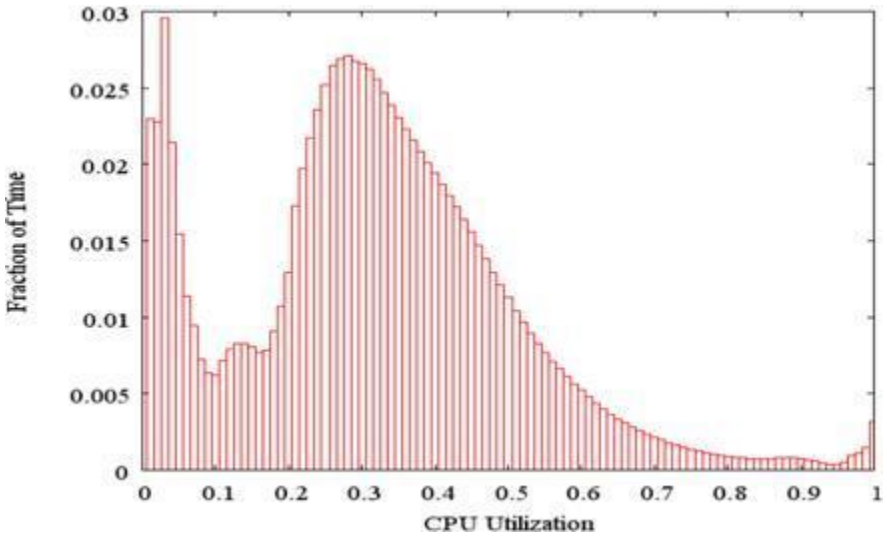

Figure 5-5: Activity profile of a sample of 5,000 Google servers over a period of 6 months.

Another feature of the energy usage profile of WSCs is not as clearly visible in Figure 5-5; 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 index. 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 [32], 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 [80] 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 standby) 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 a 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.

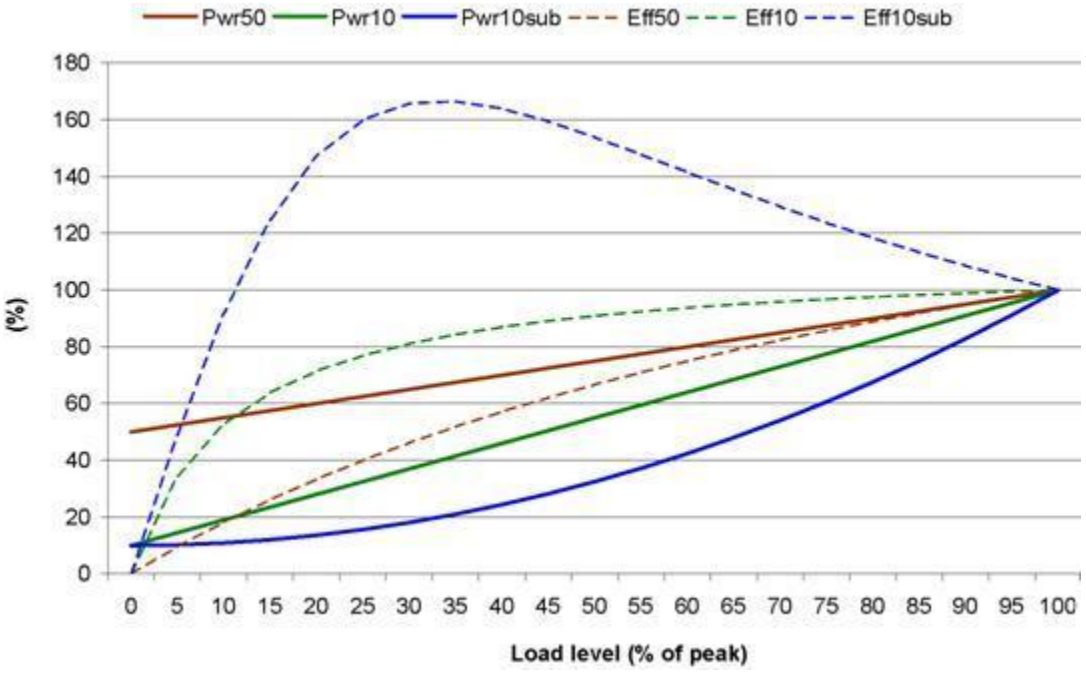

Figure 5-6: Power and corresponding power efficiency of three hypothetical systems: a typical server with idle power at 50% of peak (Pwr50 and Eff50), a more energy-proportional server with idle power at 10% of peak (Pwr10 and Eff10), and a sublinearly energy-proportional server with idle power at 10% of peak. The solid lines represent power % (normalized to peak power). The dashed lines represent efficiency as a percentage of power efficiency at peak.

The potential gains from energy proportionality in WSCs were evaluated by Fan et al. [28] in their power provisioning study. They used 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 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 5-7 shows the power usage of the main subsystems for a Google server (circa 2008) 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 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.

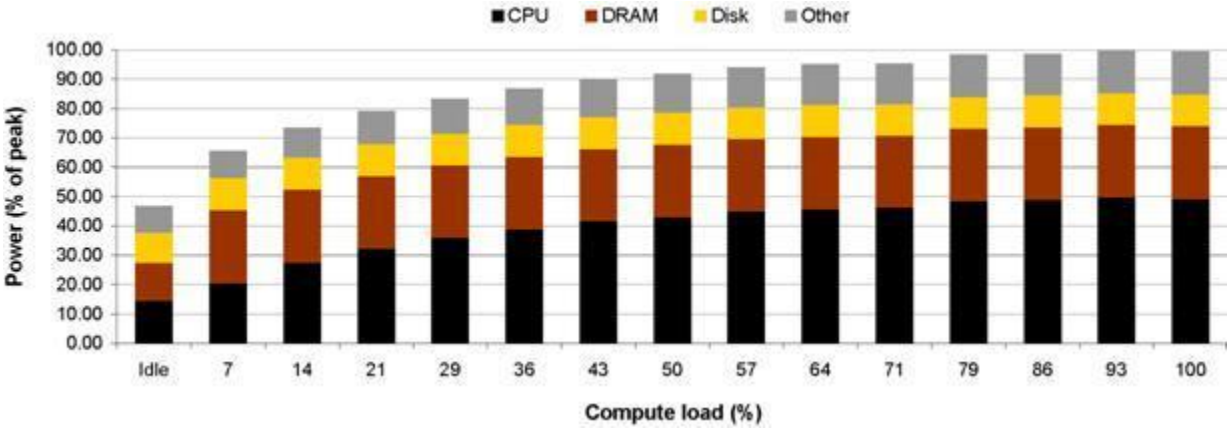

Figure 5-7: Subsystem power usage in an ×86 server as the compute load varies from idle to full usage.

5.3.2 Improving Energy Proportionality

Added focus on energy proportionality as a figure of merit in the past five years has resulted in notable improvements for server-class platforms. A meaningful metric of the energy

proportionality of a server for a WSC is the ratio between the energy efficiency at 30% and 100% utilizations. A perfectly proportional system will be as efficient at 30% as it is at 100%. As of the first edition of this book (early 2009), that ratio for the top ten SPECPower results was approximately 0.45 -- meaning that when used in WSCs, those servers exhibited less than half of their peak efficiency. As of November 2012, that figure has improved almost twofold, nearing .80. **Figure 5-8** shows increasing proportionality in Intel reference platforms between 2006 and 2012 [142]. While not yet perfectly proportional, the more recent systems are dramatically more energy proportional than their predecessors.

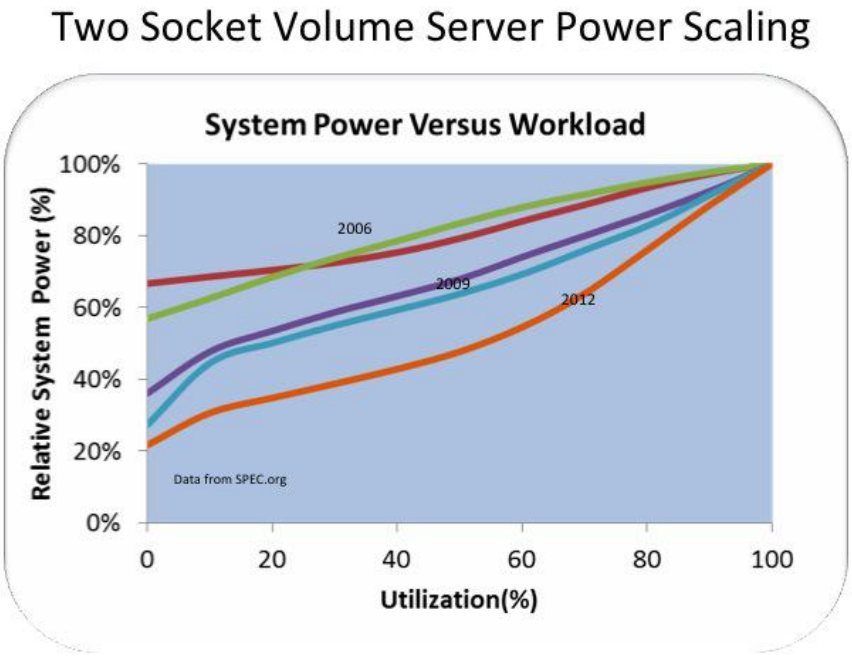

Figure 5-8 Normalized system power vs. utilization in Intel servers from 2006 and 2012. Courtesy of Winston Sounders.

Subramaniam and Feng [144] took a deeper look at energy proportionality at the server-level in order to understand where remaining bottlenecks resided. They defined parts of the CPU as the core (ALUs, FPUs, L1, and L2 caches) and uncore (memory controller, integrated I/O, and coherence engine). By taking advantage of newly-integrated on-chip energy meters, they profiled the full system through a combination of direct measurement and derivations for sub-systems such as DRAM power. As shown in Figure 5-9, they verified the proportionality improvements in package power, and the remaining sizable difference in the overall system.

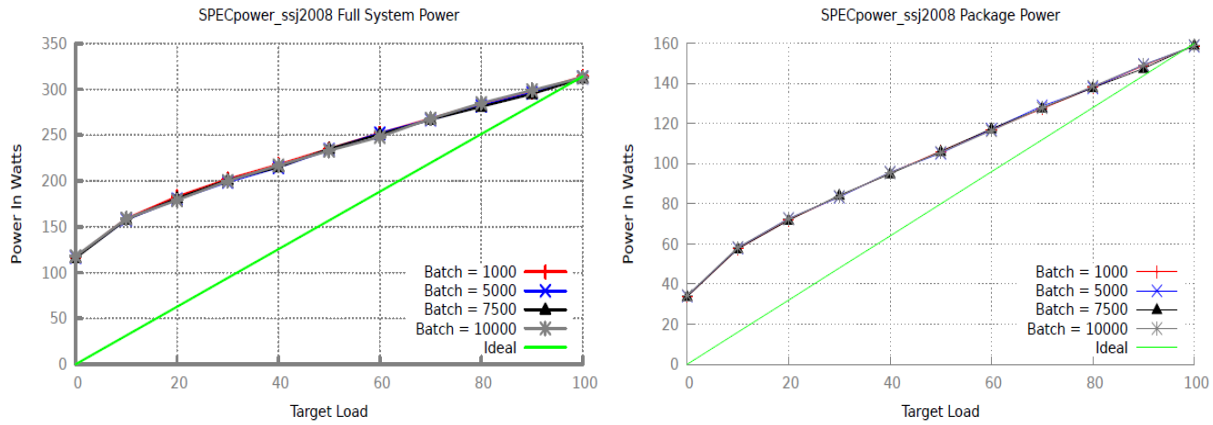

Figure 5-9: System versus package energy proportionality (Source: Subramaniam, Feng, Towards Energy-Proportional Computing for Enterprise-Class Server Workloads)

Figure 5-10 shows the almost non-existent proportionality in the uncore elements (left graph) as well as the very-limited dynamics of the DRAM system (right graph). Interestingly, DRAM power only exhibits proportionality in the range of 20-60% of target load.

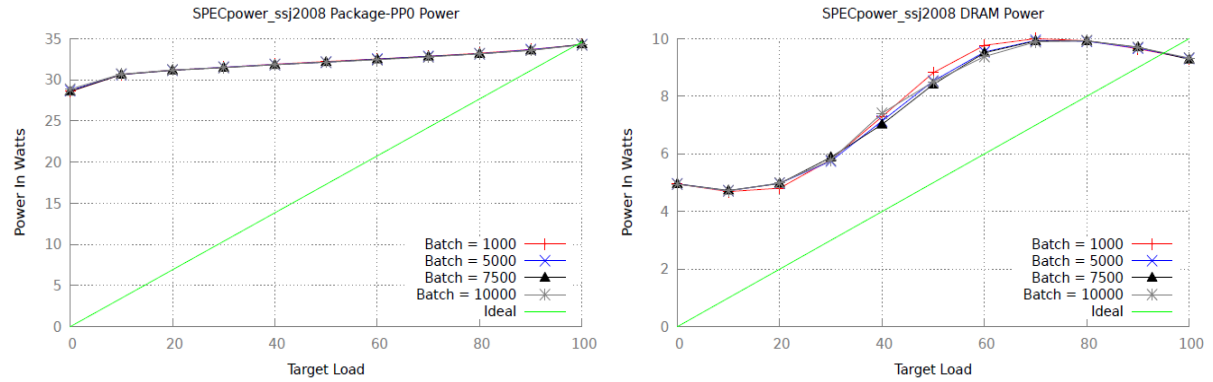

Figure 5-10: Uncore and DRAM energy proportionality (Source: Subramaniam, Feng, Towards Energy-Proportional Computing for Enterprise-Class Server Workloads)

5.3.3 Energy Proportionality - The Rest of the System

While processor energy proportionality has improved, greater effort is still required for DRAM, storage, and networking. Disk drives, for example, spend a large fraction of their energy budget (as much as 70% of their total power for high RPM drives) simply keeping the platters spinning. Improving energy efficiency and proportionality may require lower 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. [83] 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 drive.

Traditionally, datacenter networking equipment has exhibited rather poor energy proportionality. At Google we have measured switches that show little variability in energy consumption between idle and full utilization modes. Historically, servers didn't need that much network bandwidth, and switches were very expensive, so their overall energy footprint was relatively small (in the single digit percents of total IT power). However, as switches become more commoditized and bandwidth needs increase, it's possible to envision networking equipment being responsible for 10-20% of the facility energy budget. At that point, their lack of proportionality will be a severe problem. To illustrate this point, lets assume a system that exhibits a linear power usage profile as a function of utilization (u):

$$P(u) = Pi + u(1-Pi)$$

In the equation above, *Pi* represents the system's idle power, and peak power is normalized to 1.0. In such a system, energy efficiency becomes u/P(u), which reduces to the familiar Amdahl Law formulation below.

$$E(u) = 1(1-Pi) + Pi/u$$

Unlike the original Amdahl formula, we are not interested here in very high values of *u*, since it can only reach 1.0. Instead we are interested in values of utilization between 0.1 and 0.5. In that case, high values for *Pi* (low energy proportionality) will result in low efficiency. If every subcomponent of a WSC is highly energy proportional but one (say, networking or storage), that subcomponent will bound the whole system efficiency similarly to how the amount of serial work limits parallel speedup in Amdahl's formula.

Efficiency and proportionality of datacenter networks might improve in a few ways. Abts et al [139] describe how modern plesiochronous links can be modulated to adapt to usage as well as describing how topology changes and dynamic routing can create more proportional fabrics. The IEEE Energy Efficient Ethernet standardization effort [140], (802.3az) is also trying to pursue interoperable mechanisms that allow link level adaptation.

Finally, we 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.

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.

The use of very low power inactive modes with high frequency transitions has been proposed by Meisner et al [145] and Gandhi et al [146] as a way to achieve energy proportionality. The systems proposed, PowerNap and IdleCap, assume subcomponents have no useful low power modes other than full idleness and modulate active to idle transitions in all subcomponents in order to reduce power at lower utilizations while limiting the impact on performance. The promise of such approaches hinges on system-wide availability of very low power idle modes with very short active-to-idle and idle-to-active transition times, a feature that seems within reach for processors but more difficult to accomplish for other system components. In fact, Meisner et al [147] analyses the behavior of online data intensive workloads (such as Web search) and conclude that existing low power modes are insufficient to yield both energy proportionality and low latency.

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 too. 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 like PowerNap

(Meisner et al. [57]) 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.

Raghavendra et al [148] studied a 5-level coordinated power management scheme, considering per-server average power consumption, power capping at the server, enclosure, and group levels, as well as employing a virtual machine controller (VMC) to reduce the average power consumed across a collection of machines by consolidating workloads and turning unused machines off.

Such intensive power management poses nontrivial control problems. For one, applications may become unstable if some servers unpredictably slow down due to power capping. On the implementation side, power capping decisions may have to be implemented within milliseconds to avoid tripping a breaker. In contrast, overtaxing the cooling system may "only" result in a temporary thermal excursion which may not interrupt the performance of the WSC. For these reasons, power capping is not widely used today.

5.6 DATACENTER POWER PROVISIONING

Energy efficiency optimizations reduce electricity costs. In addition, they reduce construction costs: if, for example, free cooling eliminates chillers, then we don't have to purchase and install those chillers, nor do we have to pay for generators or UPSes to back them up. Such construction cost savings can double the overall savings from efficiency improvements.

Actually using the provisioned power of a facility is equally important. For example, if a facility operates at 50% of its peak power capacity, the effective provisioning cost per Watt used is doubled. This incentive to fully use the power budget of a datacenter is offset by the risk of exceeding its maximum capacity, which could result in outages.

5.6.1 Deploying the right amount of equipment

How many servers can we install in a 1MW facility? This simple question is harder to answer than it seems. First, server specifications usually provide very conservative values for maximum power consumption. Some system vendors such as Dell and HP, offer online power calculators [21][47] to provide better estimates, but it may be necessary to measure the actual power consumption of the dominant applications manually.

Second, actual power consumption varies significantly with load (thanks to energy proportionality), and it may be hard to predict the peak power consumption of a group of

servers: while any particular server might temporarily run at 100% utilization, the maximum utilization of a group of servers probably isn't 100%. But to do better we'd need to understand the correlation between the simultaneous power usage of large groups of servers. 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 Oversubscribing Facility Power

As soon as we use anything but the most conservative estimate of equipment power consumption to deploy clusters, we incur a certain risk that we exceed the available amount of power, i.e., we oversubscribe facility power. A successful implementation of power oversubscription increases the overall utilization of the datacenter's power budget while minimizing the risk of overload situations. We will expand on this issue because it has received much less attention in technical publications than the first two steps listed above, and it is a very real problem in practice [55].

Fan et al. [28] 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 5-11 shows the cumulative distribution of power usage over time for groups of 80 servers (rack), 800 servers (PDU), and 5,000 servers (cluster).

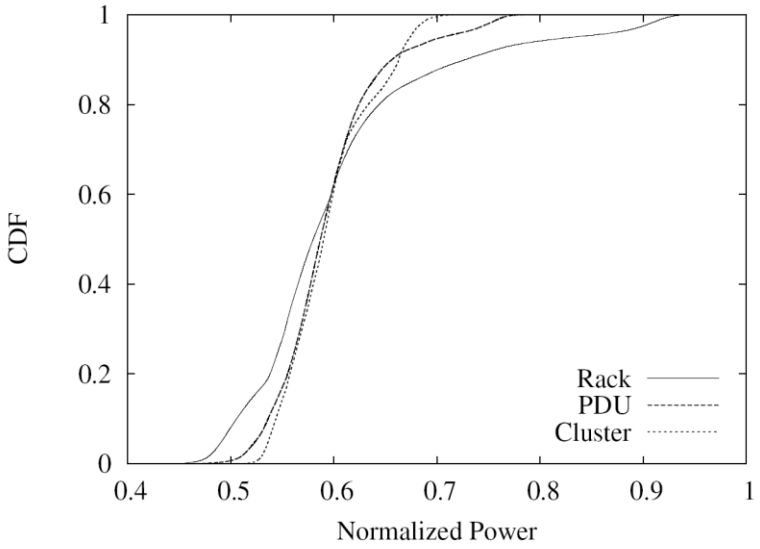

Figure 5-11: Cumulative distribution of the time that groups of machines spend at or below a given power level (power level is normalized to the maximum peak aggregate power for the corresponding grouping) (Fan et al. [28]).

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.

The study further 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, it's easy to end up with an underutilized facility even when paying attention to correct power ratings. For example, a facility typically needs to accommodate future growth, but keeping space open for such growth reduces utilization and thus increases unit costs. Various forms of fragmentation can also prevent full utilization. For example, perhaps we ran out of space in a rack because low-density equipment used it up, or perhaps we can't insert another server because we're out of network ports, or out of plugs or amps on the power strip. For example, a 2.5-kW circuit supports only four 520-W servers, limiting utilization to 83% on that circuit. Since the lifetimes of various WSC components differ (servers might be replaced every 3 years, cooling every 10 years, networking every 4 years, etc) it's difficult to plan for 100% utilization, and most organizations don't.

5.7 TRENDS IN SERVER ENERGY USAGE

While in the past dynamic voltage and frequency scaling (DVFS) used to be the predominant mechanism for managing energy usage in servers, today we face a different and more complex scenario. Given lithography scaling challenges, the operating voltage range of server-class CPUs is very narrow, resulting in ever decreasing gains from DVFS. **Figure 5-12** 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%. Instead, modern CPUs increasingly rely on multiple power planes within a die as their primary power

management mechanism, allowing whole sections of the chip to be powered down and back up quickly as needed.

A second trend is that CPUs continue to outpace other server components in energy efficiency and proportionality improvements. The result is a power budget breakdown with larger energy fractions from non-CPU subsystems, particularly at lower activity levels when the gap in energy proportionality between CPUs and other components widen.

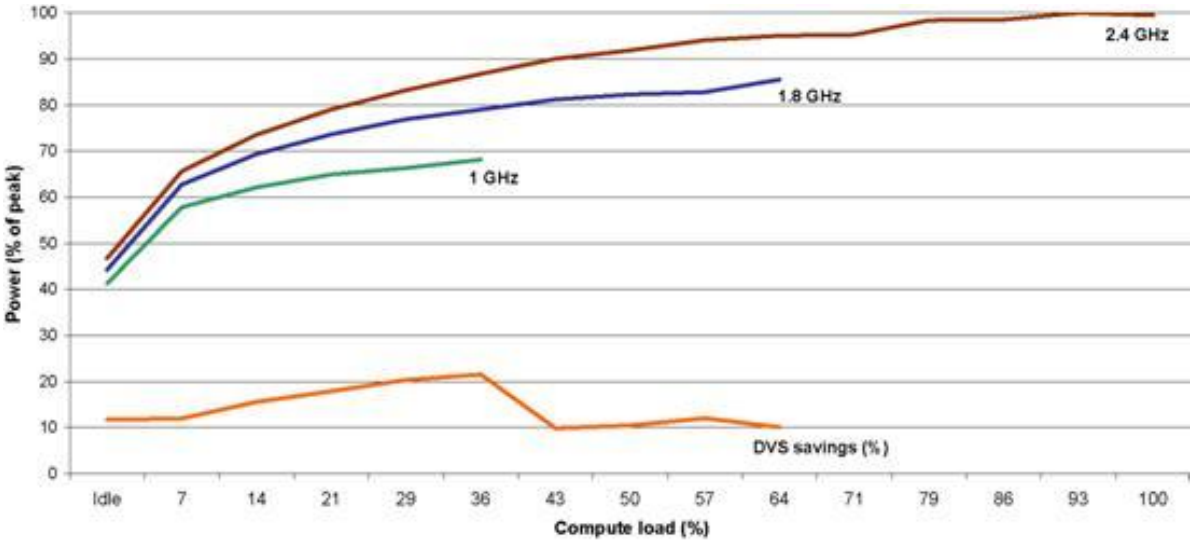

Figure 5-12: Power vs. compute load for an x86 server at three voltage-frequency levels and the corresponding energy savings.

5.7.1 USING ENERGY STORAGE FOR POWER MANAGEMENT

Several recent studies [149,150,151] have proposed to use energy stored in the facility's backup systems (e.g., UPS batteries) to optimize facility performance or reduce energy costs. Stored energy could be used to flatten the facility's load profile (using less utility power when it's most expensive), or to mitigate supply variability in a wind-powered facility, or to manage short demand peaks in oversubscribed facilities (using stored energy instead of capping the load).

In our opinion, the most promising use of energy storage in power management consists of managing short demand peaks. Power capping systems need some time to react intelligently to demand peak events, and may need to set peak provisioning levels well below the maximum breaker capacity in order to allow time for power capping to respond. A power capping system that can draw from energy storage sources for just a few seconds during an unexpected peak would allow the facility to safely operate closer to its maximum capacity, while requiring a relatively modest amount of additional energy storage capacity.

To our knowledge no such power management systems have yet been used in production systems. Deploying such a system is difficult and potentially costly. Besides the control complexity, the additional cost of batteries can be significant, since we can't just reuse the existing UPS capacity for power management, as doing so would make the facility more vulnerable in an outage. Furthermore, the types batteries typically used in UPS systems (lead-acid) don't age well under frequent cycling, so that more expensive technologies

might be required. While some have argued that expanded UPSes would be cost effective [151] we believe that the economic case has not yet been made in practice.

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 in energy efficiency from now on will need to come from computer scientists and engineers, and less so from mechanical or power conversion specialists (large opportunities remain for mechanical and power engineers in reducing facility costs in particular).

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 overloading the facility's power delivery system. Power capping promises 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 datacenter. 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. [71] 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 [29][30] focus specifically on the issue of datacenter power oversubscription and on dynamic voltage-frequency scaling as the mechanism to reduce peak power consumption. Managing temperature is the subject of the systems proposed by Heath et al. [46] and Moore et al. [58]. Finally, Pedram [152] provides an introduction to resource provisioning, and summarizes key techniques for dealing with management problems in the datacenter.

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, excluding depreciation: 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 [63][51]. 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 $9–13/W to build and smaller ones cost more. The cost numbers in the table below shouldn't be directly compared, since the scope of the projects may differ. For example, the amount quoted may or may not include land or the cost of a pre-existing building.

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 [70]
$9–13	Dupont Fabros 2011 10K report [153] contains financial information suggesting the following cost for its most recent

	facilities (built in 2010 and 2011 - see page 39 for critical load and page 76 for cost): $204M for 18.2 MW (NJ1 Phase I) => $11.23/W $116M for 13 MW (ACC6 Phase I) => $8.94/W $229M for 18.2MW (SC1 Phase 1) => $12.56/W
$8-10	Microsoft's investment of $130M for 13.2MW ($9.85/W) capacity expansion to its data center in Dublin, Ireland [154] Facebook is reported to have spent $210M for 28MW ($7.50/W) at its Prineville data center [155].

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

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.

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 $12/W datacenter over 12 years, the depreciation cost is $0.08/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 $0.05/W, for a total of $0.13/W per month. Typical interest rates vary over time, but many companies will pay interest in the 7–12% 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 $0.02/W per month, for a total of $0.19/W per month.

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 [75]. 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. For this example we chose a Dell PowerEdge R520 with 2 CPUs, 48GB of RAM and

four disks#. This server draws 340W at peak per Dell's configuration planning tool and costs approximately $7,700 as of 2012. The remaining base case parameters were chosen as follows:

- The cost of electricity is the 2012 average U.S. industrial rate of 6.7 cents/kWh.
- The interest rate a business must pay on loans is 8%, and we finance the servers with a 3-year interest-only loan.
- The cost of datacenter construction is $10/W amortized over 12 years.
- Datacenter Opex is $0.04/W per month.
- The datacenter has a power usage effectiveness (PUE) of 1.8.
- 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 shows a breakdown of the yearly TCO for case A among datacenter and server-related Opex and Capex components.[2]

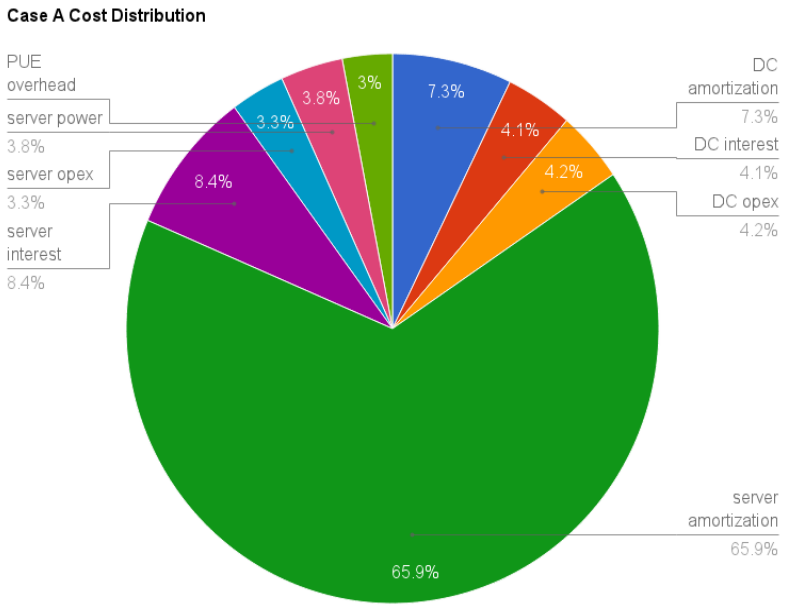

Figure 6-1: TCO cost breakdown for case study A.

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

In this example, which is typical of classical datacenters, the high server capital costs dominate overall TCO, with 78% 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 39% of the total, and energy costs to 26%, with server costs falling to 35%. 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.

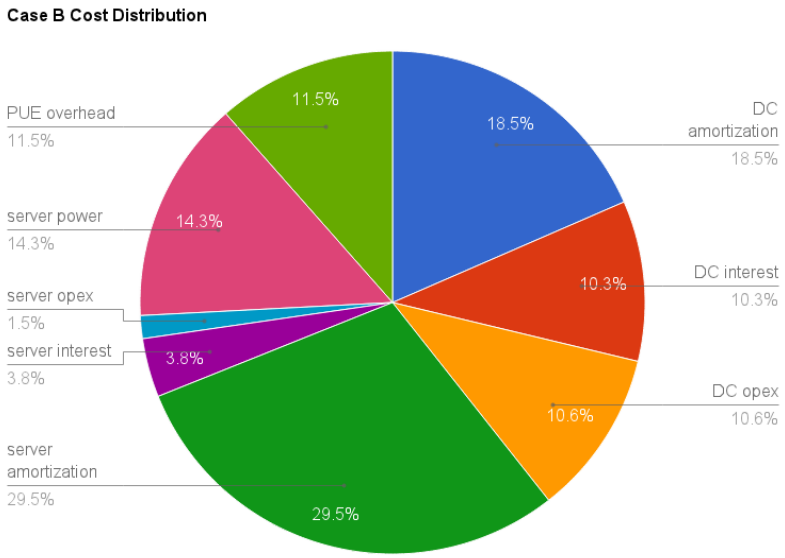

Figure 6-2: TCO cost breakdown for case study B (lower-cost, higher-power servers).

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 ($6,774 versus $11,683) because the server is so much cheaper. The relative importance of power-related costs may increase as shown in case B 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 [84], 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. 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.

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 6-3**), with only 25% 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 $6,774 vs. $9,443 in a half-full datacenter—all that for a server that costs just $2,000 to purchase.

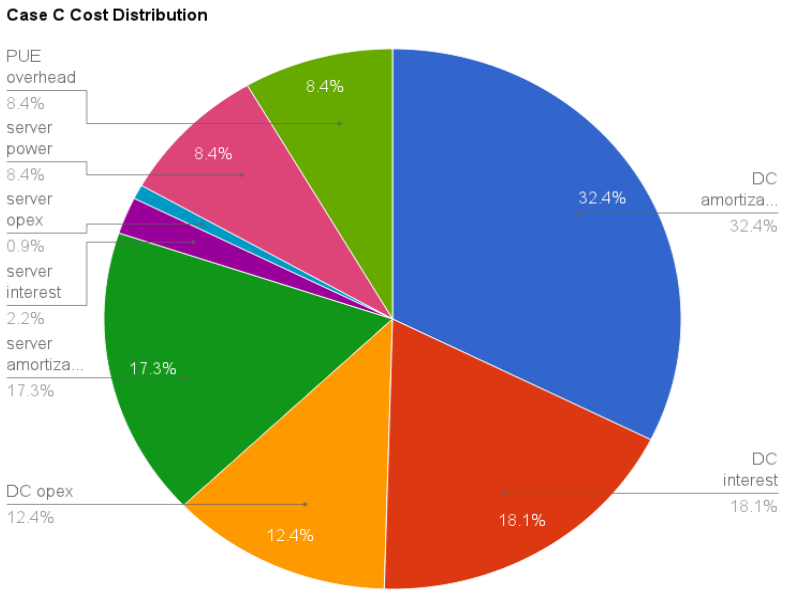

Figure 6-3: TCO case study C (partly filled facility).

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 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.

6.3.3 The Cost of Public Clouds

Instead of building your own datacenter and server, you can rent a virtual machine from a public cloud provider such as Google's Compute Engine or Amazon's EC2. The Dell server used in our example is roughly comparable to an AWS "High Memory Quadruple Extra Large" instance, which cost $1.80/hr as of January 2013 as an on-demand instance, and $6,200 plus $0.28/hr with a three-year contract.

Before we compare these with our cost model, consider the two very different pricing plans. Spot pricing is "pay as you go" pricing--you can start and stop a VM at any time, so if you need one for only a few days a year, on-demand pricing will be vastly cheaper than any other alternative. For example, you may need two servers to handle your peak load for 6 hours per day on weekdays, and one server during the rest of the year, and with a spot instance you'll only pay for 30 hours per week, vs 168 if you owned the server. On the other hand, spot instances are fairly expensive: at $1.80/hr, using one for three years will cost you $47,000, vs the roughly $20,000 of an owned server.#

If you need a server for an extended period, public cloud providers will lower the hourly price in exchange for a long-term commitment as well as an upfront fee. Using the above three-year contract as an example, a fully utilized instance will cost $6,200 upfront plus $7,500 in use charges, or $13,700 total, about 30% of what you would have paid for using an on-demand instance for three years. This cost is competitive with that of an owned machine, possibly even cheaper since you could further reduce your cost in case you didn't need the server anymore after year two.

How can a public cloud provider (who must make a profit on these prices) compete with your in-house costs? In one word: scale. As discussed in this chapter, many of the operational expenses are relatively independent of the size of the datacenter: if you want a security guard or a facilities technician on-site 24x7, it's the same cost whether your site is 1MW or 5MW. Furthermore, a cloud provider's capital expenses for servers and buildings likely are lower than yours, since they buy (and build) in volume. Google, for example, designs their own servers and datacenters to reduce cost.

Why, then, are on-demand instances so much more expensive? Because the cloud provider doesn't know whether you're going to need one or not, they need to keep around additional severs just in case someone wants them, so the utilization of the server pool used for on-demand instances will be substantially below 100% on average. If, for example, the typical use for an on-demand instance is to cover the 6 hours a day where traffic peaks, their utilization will be 25%, and thus the cost per hour is four times higher.

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 [44], 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 [32]. 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 pre-release 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. [93] 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. [60] 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 [36], 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 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.

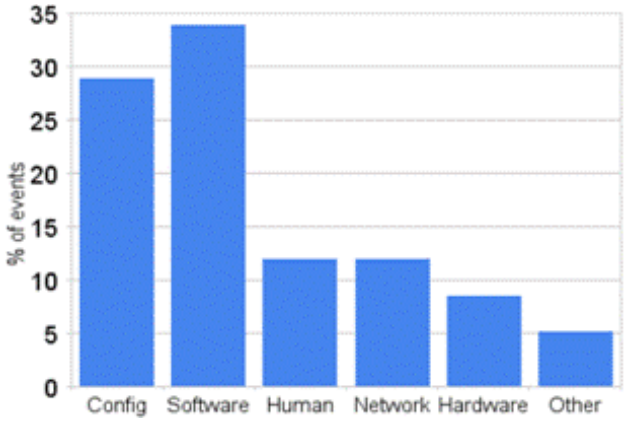

Figure 7-1: Distribution of service disruption events by most likely cause at one of Google's main services (preliminary data, 6 weeks only—from Google's Robert Stroud).

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. [50] 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 [77] 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.[3]

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.

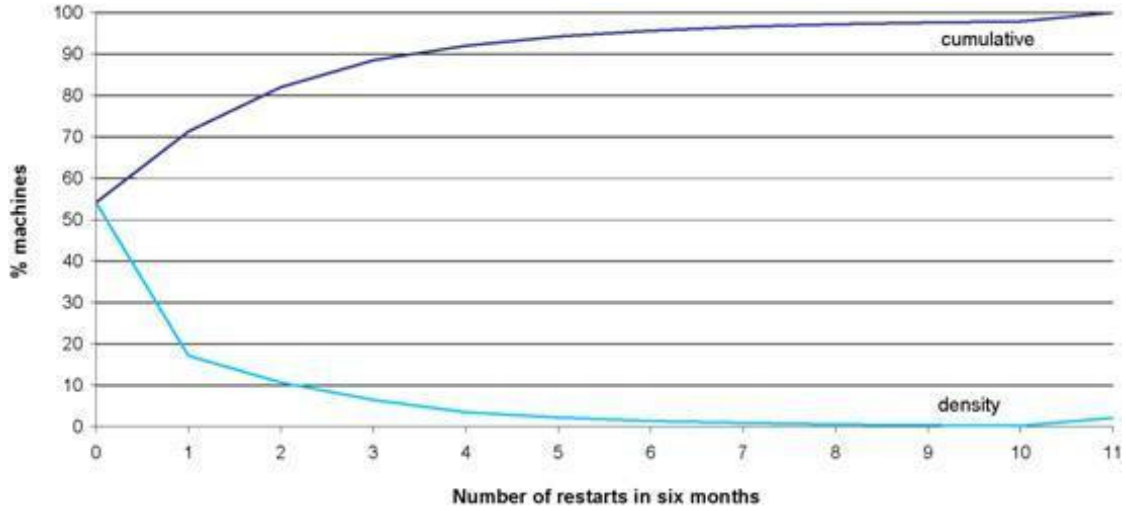

Figure 7-2: Distributions of machine restarts over 6 months at Google.

[3] As a reference, in an Intel Technical Brief [65], 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%.

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.

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 [64]. **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.

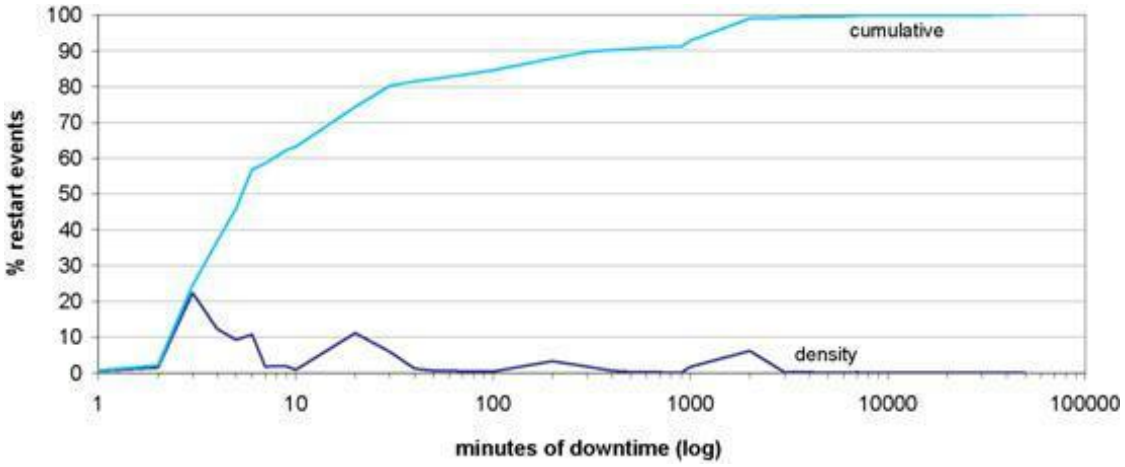

Figure 7-3: Distribution of machine downtime, observed at Google over 6 months. The average annualized restart rate across all machines is 4.2, corresponding to a mean time between restarts of just less than 3 months.

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, 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 [85] 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 study by Schroeder et al. [79] 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. A more recent study [156] found that a large fraction of DRAM errors can be attributed to hard (non-transient) errors and suggest that simple page retirement policies could mask a large fraction of DRAM errors in production systems while sacrificing only a negligible fraction of the total DRAM in the system.

Disk errors. Studies based on data from Network Appliances [3], Carnegie Mellon [78], and Google [67] 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.

The effect of ambient temperature on the reliability of disk drives has been well studied by Pinheiro et al [67] and El Sayed et al [136]. While common wisdom previously held that temperature had an exponentially negative effect on the failure rates of disk drives, both of these field studies found little or no evidence of that in practice. In fact, both of them suggest that most disk errors appear to be uncorrelated with temperature.

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. [67] 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.

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.

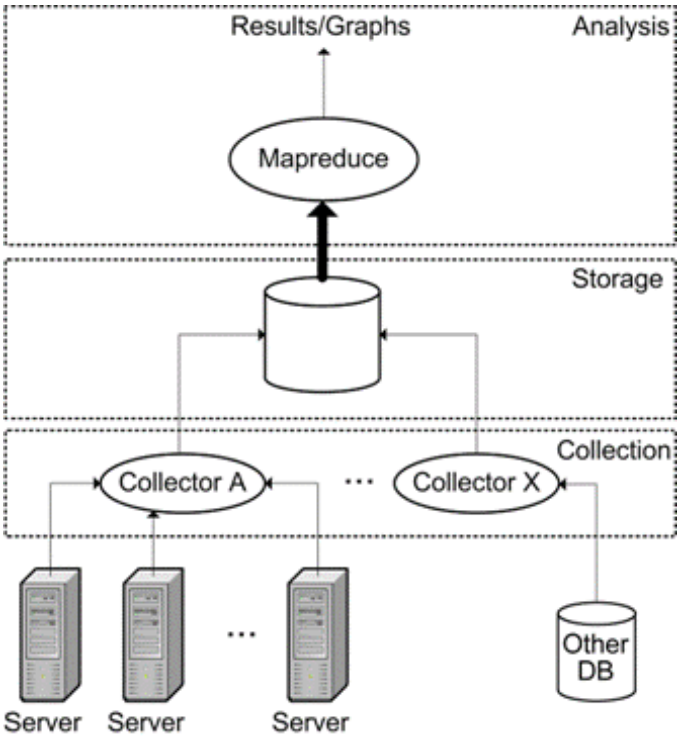

Figure 7-4: The Google System Health monitoring and analysis infrastructure.

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. [28].

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.

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 1.13 PUE across all its custom-built facilities as of late 2012 [33].

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.

9 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.

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.

5 L. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: the architecture of the Google cluster," IEEE Micro, April 2003.

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.

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.

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.

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.

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.

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. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Processing," CACM 36(6), June 1992.

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

25 J. Elerath and S. Shah, "Server class disk drives: how reliable are they?", IEEE Reliability and Maintainability, 2004 Annual Symposium—RAMS, January 2004.

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

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

28 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.

29 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.

30 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.

31 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.

32 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.

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

34 Google Inc., "Efficient Data Center Summit, April 2009". Available at http://www.google.com/about/datacenters/efficiency/external/2009-summit.html#tab0=4.

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

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

37 The Green 500. Available at http://www.green500.org.

38 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.

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

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

41 SNIA Green Storage Initiative. Available at http://www.snia.org/forums/green/.

42 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.

43 The Hadoop Project. Available at http://hadoop.apache.org

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

45 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.

46 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.

47 HP Power Calculator. Available at http://www.hp.com/configurator/powercalcs.asp.

48 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.

49 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.

50 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.

51 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.

52 W. Lang, J. Patel, and S. Shankar, "Wimpy Node Clusters: What about Non-Wimpy Workloads?", in Proceedings of the Sixth International Workshop on Data Management on New Hardware, June 2010.

53 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.

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

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

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

57 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.

58 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.

59 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.

60 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.

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

62 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.

63 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.

64 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.

65 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.

66 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.

67 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.

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

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

70 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.

71 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.

72 Reuters, "Dupont Fabros Technology, Inc. reports third quarter results," November 5, 2008.

73 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.

74 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.

75 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.

76 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.

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

78 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.

79 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.

80 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.

81 SPEC Power. Available at http://www.spec.org/power_ssj2008/.

82 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.

83 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.

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

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

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

87 Transaction Processing Performance Council. Available at http://www.tpc.org.

88 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.

89 TPC-C Executive summary for the Superdome-Itanium2, February 2007.

90 TPC-C Executive summary for the ProLiant ML350G5, September 2007.

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

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

93 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.

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

95 "Dremel: Interactive Analysis of Web-Scale Datasets", Sergey Melnik et al, Proc. of the 36th Int'l Conf on Very Large Data Bases (2010), pp. 330-339, 2010.

96 "Spanner: Google's Globally-Distributed Database", James C. Corbett et al, Proceedings of OSDI'12: Tenth Symposium on Operating System Design and Implementation, Hollywood, CA, October, 2012

97 "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure", Benjamin H. Sigelman et al, Google Research Report, http://research.google.com/pubs/pub36356.html, 2010.

98 "The Tail at Scale", Jeffrey Dean and Luiz André Barroso, Communications of the ACM, Vol 56, No. 2, February 2013, pp. 80-86.

99 "Availability in Globally Distributed Storage Systems". Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. In Proceedings of the 9th USENIX Sympsium on Operating Systems Design and Implementation (OSDI'10), October 2010, Vancouver, BC, Canada. http://static.usenix.org/event/osdi10/tech/full_papers/Ford.pdf

100 "GFS: Evolution on Fast-forward", Marshall Kirk McKusick, Sean Quinlan. ACM Queue, August 1, 2009. http://queue.acm.org/detail.cfm?id=1594206

101 Luiz André Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. 2000. "Piranha: a scalable architecture based on single-chip multiprocessing". *SIGARCH Comput. Archit. News* 28, 2 (May 2000), 282-293. DOI=10.1145/342001.339696 http://doi.acm.org/10.1145/342001.339696

102 Urs Hölzle, "Brawny cores still beat wimpy cores, most of the time.", IEEE MICRO, July/Aug. 2010

103 Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2011. “Disk-locality in datacenter computing considered irrelevant”. In Proceedings of the 13th USENIX conference on Hot topics in operating systems (HotOS'13). USENIX Association, Berkeley, CA, USA, 12-12.

104 John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Diego Ongaro, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman. 2011. “The case for RAMCloud”. Commun. ACM 54, 7 (July 2011), 121-130. DOI=10.1145/1965724.1965751 http://doi.acm.org/10.1145/1965724.1965751

105 “Megastore: Providing Scalable, Highly Available Storage for Interactive Services”, Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James S. Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, Vadim Yushprakh. Proceedings of the Conference on Innovative Data system Research (CIDR) (2011), pp. 223-234 http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/pubs/archive/36971.pdf

106 “Memcached”, B. Fitzpatrick et al, 2003. http://memcached.org/about,

107 “The Case for RAMClouds:Scalable High-Performance Storage Entirely in DRAM”. John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman. SIGOPS Operating Systems Review, Vol. 43, No. 4, December 2009, pp. 92-105

108 David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. 2011. “FAWN: a fast array of wimpy nodes”. Commun. ACM 54, 7 (July 2011), 101-109. DOI=10.1145/1965724.1965747 http://doi.acm.org/10.1145/1965724.1965747

109 Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, Ion Stoica, “Disk-Locality in Datacenter Computing Considered Irrelevant”. In Proceedings of the 13th Workshop on Hot Topics in Operating Systems (HotOS XIII), Napa, CA, May 2011.

110 “PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric”. Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri,Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Department of Computer Science and Engineering, University of California San Diego. SIGCOMM’09, August 17–21, 2009, Barcelona, Spain.

111 Vahdat, A., Al-Fares, M., Farrington, N., Mysore, R. N., Porter, G., Radhakrishnan, S. 2010. “Scale-out networking in the data center”. IEEE Micro 30(4):29-41; http://dx.doi.org/10.1109/MM.2010.72.

112 “A guided tour of data-center networking”. by: Dennis Abts, and Bob Felderman. In: Commun. ACM, Vol. 55, Nr. 6 (2012) , p. 44-51.

113 Uptime Institute Publications. http://uptimeinstitute.com/publications

114 Telecommunications Industry Association standards: http://global.ihs.com/tia_telecom_infrastructure.cfm?RID=Z56&MID=5280

115 Uptime Institute: Data Center Site Infrastructure Tier Standard: Topology. http://uptimeinstitute.com/publications

116 Uptime Institute: Data Center Site Infrastructure Tier Standard: Operational Sustainability. http://uptimeinstitute.com/publications

117 TIA-942 Telecommunications Infrastructure Standard for Data Centers. http://www.adc.com/us/en/Library/Literature/ 102264AE.pdf Page 5.

118 “Data center uninterruptible power distribution architecture”. William Whitted et al. http://www.google.com/patents/US7560831

119 Open Compute Project, Battery Cabinet Hardware v1.0. http://opencompute.org/wp/wp-content/uploads/2011/07/DataCenter-Battery-Cabinet-Specifications.pdf

120 Wet bulb temperature. http://en.wikipedia.org/wiki/Wet-bulb_temperature

121 eBay Containers Push Frontiers of Efficiency: http://www.datacenterknowledge.com/archives/2012/02/27/ebay-containers-push-frontiers-of-efficiency/

122 Dell Modular Data Center: http://content.dell.com/us/en/corp/by-need-it-productivity-deploy-systems-faster-modular-data-center.aspx?~ck=bt

123 IO: http://www.iodatacenters.com/modular-data-center/

124 Colt ftec Data Centre: http://www.colt.net/uk/en/products-services/data-centre-services/modular-data-centre-en.htm

125 SGI (Formerly Rackables): http://www.sgi.com/products/data_center/ice_cube/

126 APC/Schneider: http://www.apcmedia.com/salestools/WTOL-7NGRBS_R1_EN.pdf

127 HP Performance Optimized Datacenter (POD): http://h18004.www1.hp.com/products/servers/solutions/datacentersolutions/pod/index.html

128 The Green Grid. A Framework for Data Center Energy Efficiency: http://www.thegreengrid.org/~/media/WhitePapers/WhitePaper13FrameworkforDataCenterEnergyProductivity5908.pdf?lang=en

129 The Green Grid. The Latest on The Green Grid’s Productivity Research. http://www.thegreengrid.org/~/media/TechForumPresentations2011/TheLatestonTheGreenGridsProductivityResearch_2011.pdf?lang=en

130 Distribution of PUE Ratios: http://www.energystar.gov/ia/partners/prod_development/downloads/DataCenters_GreenGrid02042010.pdf

131 Uptime Institute, Important to recognize the dramatic improvement in data center efficiency: http://blog.uptimeinstitute.com/2012/09/important-to-recognize-the-dramatic-improvement-in-data-center-efficiency/

132 Google Data centers. http://www.google.com/about/datacenters/efficiency/internal/index.html#measuring-efficiency

133 Google green, The Big Picture: http://www.google.com/green/bigpicture/#/

134 The Green Grid, WP#22-Usage and Public Reporting Guidelines for The Green Grid's Infrastructure Metrics PUE/DCiE; http://www.thegreengrid.org/Global/Content/white-papers/Usage%20and%20Public%20Reporting%20Guidelines%20for%20PUE%20DCiE.aspx

135 Google Data centers. Efficiency: How we do it. http://www.google.com/about/datacenters/efficiency/internal/index.html#measuring-efficiency

136 “Temperature Management in Data Centers: Why Some (Might) Like It Hot”. Nosayba El-Sayed Ioan Stefanovici George Amvrosiadis Andy A. Hwang, Bianca Schroeder, SIGMETRICS'12. http://www.cs.toronto.edu/~bianca/papers/temperature_cam.pdf

137 “The Future of Computing Performance: Game Over or Next Level”, National Academies Press, Samuel H. Fuller and Lynette I. Millet, Editors; Committee on Sustaning Growth in Computing Performance, National Research Council, 2011. http://www.nap.edu/catalog.php?record_id=12980

138 Koomey, J.G.; Berard, S.; Sanchez, M.; Wong, H.; , "Implications of Historical Trends in the Electrical Efficiency of Computing," Annals of the History of Computing, IEEE , vol.33, no.3, pp.46-54, March 2011 -- doi: 10.1109/MAHC.2010.28 . http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5440129&isnumber=5986492

139 Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. 2010. “Energy proportional datacenter networks”. SIGARCH Comput. Archit. News 38, 3 (June 2010), 338-347. DOI=10.1145/1816038.1816004 http://doi.acm.org/10.1145/1816038.1816004

140 Christensen, K.; Reviriego, P.; Nordman, B.; Bennett, M.; Mostowfi, M.; Maestro, J.A.; , "IEEE 802.3az: the road to energy efficient Ethernet," Communications Magazine, IEEE , vol.48, no.11, pp.50-56, November 2010. doi: 10.1109/MCOM.2010.5621967. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5621967&isnumber=5621952

141 Storage Networking Industry Association (SNIA) Emerald Power Efficiency Measurement Specification V1.0, August 2011.

142 Winston Sounders, “Server Efficiency: Aligning Energy Use with Workloads”, Data Center Knowledge, June 12, 2012.

143 Storage Performance Council SPC Benchmark 2/Energy Extension V1.4, November 2012

144 Balaji Subramaniam, Wu-chun Feng. “Towards Energy-Proportional Computing for Enterprise-Class Server Workloads.” http://people.cs.vt.edu/~balaji/docs/papers/specpowerrapl_techreport_balaji.pdf

145 David Meisner, Brian T. Gold, and Thomas F. Wenisch. 2009. “PowerNap: eliminating server idle power”. SIGPLAN Not. 44, 3 (March 2009), 205-216. DOI=10.1145/1508284.1508269 http://doi.acm.org/10.1145/1508284.1508269

146 Gandhi, et al. “Power Capping Via Forced Idleness”. http://repository.cmu.edu/cgi/viewcontent.cgi?article=1868&context=compsci&sei-redir=1&referer=http%3A%2F%2Fscholar.google.com%2Fscholar%3Fhl%3Den%26q%3Dpower%2Bcapping%26btnG%3D%26as_sdt%3D1%252C5%26as_sdtp%3D#search=%22power%20capping%22

147 David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. 2011. “Power management of online data-intensive services”. SIGARCH Comput. Archit. News 39, 3 (June 2011), 319-330. DOI=10.1145/2024723.2000103 http://doi.acm.org/10.1145/2024723.2000103

148 Raghavendra et al. “No “Power” Struggles: Coordinated Multi-level Power Management for the Data Center.” http://www.cs.pitt.edu/~kirk/cs3150spring2010/2008_asplos_nopowerstruggles.pdf

149 Sriram Govindan, Di Wang, Anand Sivasubramaniam, Bhuvan Urgaonkar "Leveraging Stored Energy for Handling Power Emergencies in Aggressively Provisioned Datacenters" http://csl.cse.psu.edu/publications/asplos12.pdf

150 Di Wang, Chuangang Ren, Anand Sivasubramaniam, Bhuvan Urgaonkar, and Hosam Fathy. "Energy Storage in Datacenters: What, Where, and How much?" http://csl.cse.psu.edu/publications/sigmetrics12.pdf

151 Kontorinis et al. "Managing distributed ups energy for effective power capping in data centers" http://dl.acm.org/citation.cfm?id=2337216&bnc=1

152 Pedram, M "Energy-Efficient Datacenters" Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions. Oct. 2012

153 DuPont Fabros Technology Annual Report 2011. http://www.dft.com/sites/default/files/2011-10k.pdf

154 "Microsoft Expands Dublin Cloud Computing Hub" Data Center Knowledge. February 23rd, 2012. http://www.datacenterknowledge.com/archives/2012/02/23/microsoft-expands-dublin-cloud-computing-hub/

155 "Facebook Has Spent $210 Million on Oregon Data Center" Data Center Knowledge. January 30th, 2012. http://www.datacenterknowledge.com/archives/2012/01/30/facebook-has-spent-210-million-on-oregon-data-center/

156 Andy A. Hwang, Ioan Stefanovici, Bianca Schroeder. "Cosmic Rays Don't Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design", ASPLOS 2012, March 2012.

157 Preliminary assessment from Uptime Institute: IDC Data Center of the Future US Server Power Spend for 2005 as a baseline($6bn); applied a cooling factor of 1; applied a .6 multiplier to US data for WW amount; Belady,C., Malone, C.,"Data Center Power Projection to 2014", 2006 ITHERM, San Diego, CA (June 2006)

10 Author Biographies

Luiz André Barroso has worked across several engineering areas including software infrastructure, storage availability, energy efficiency, and hardware design. He was 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. He has lectured at Pontifica Universidade Catolica (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. Luiz is a Google Fellow, a Fellow of the ACM and the American Association for the Advancement of Science.

Jimmy Clidaras led Google's datacenter engineering program through multiple generations starting in 2004, with a special emphasis on energy- and cost-efficient design. He was the first director of Google's Platforms Infrastructure Engineering team, responsible for power, cooling, embedded software, and datacenter R&D engineering. Jimmy's original background is in aerospace engineering, having worked at Harris Corporation and E-Systems, where he developed space-flight hardware for communication and research satellites. He holds degrees in audio engineering ('84) and mechanical engineering ('94, Florida Atlantic University). He is currently a Distinguished Datacenter Engineer at Google and a Distinguished Alumnus of FAU. He remains engaged in datacenter research activities, continuing the search for disruptive technologies.

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.