CHAPTER 3

WSC Hardware Building Blocks

As mentioned earlier, the architecture of WSCs is largely defined by the hardware building blocks chosen. This process is analogous to choosing logic elements for implementing a microprocessor, or selecting the right set of chipsets and components for a server platform. In this case, the main building blocks are server hardware, networking fabric, and storage hierarchy components. This chapter focuses on these building blocks, with the objective of increasing the intuition needed for making such choices.

3.1 SERVER HARDWARE

Clusters of mid-range servers are the preferred building blocks for WSCs today [BDH03]. This is true for a number of reasons, the primary one being the underlying cost-efficiency of mid-range 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. The continuing CPU core count increase has also reached a point that most VM/task instances can comfortably fit into a two-socket server. Such server platforms share many key components with the 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 benchmarking. In the first edition, we showed a comparison of TPC-C benchmark [TPC] results from a system based on a high-end server (HP Integrity Superdome-Itanium2 [TPC07a]) and one based on a low-end server (HP ProLiant ML350 G5 [TPC07b]). The difference in cost-efficiency was over a factor of four in favor of the low-end server. When looking for more recent data for this edition, we realized that there are no competitive benchmarking entries that represent the high-end server design space. As we observed in 2009, the economics of the server space made that class of machines occupy a small niche in the marketplace today. The more interesting discussions now are between mid-range server nodes and extremely low end (so-called "wimpy") servers, which we cover later in this chapter.

3.1.1 SERVER AND RACK OVERVIEW

Servers hosted in individual racks are the basic building blocks of WSCs. They are interconnected by hierarchies of networks, and supported by the shared power and cooling infrastructure.

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

As discussed in Chapter 1, WSCs use a relatively homogeneous hardware and system software platform. This simplicity implies that each server generation needs to provide optimized performance and cost for a wide range of WSC workloads and the flexibility to support their resource requirements. Servers are usually built in a tray or blade enclosure format, housing the motherboard, chipset, and additional plug-in components. The motherboard provides sockets and plug-in slots to install CPUs, memory modules (DIMMs), local storage (such as Flash SSDs or HDDs), and network interface cards (NICs) to satisfy the range of resource requirements. Driven by workload performance, total cost of ownership (TCO), and flexibility, several key design considerations determine the server's form factor and functionalities.

- *CPU*: CPU power, often quantified by the thermal design power, or TDP; number of CPU sockets and NUMA topology; CPU selection (for example, core count, core and uncore frequency, cache sizes, and number of inter-socket coherency links).
- *Memory*: Number of memory channels, number of DIMMs per channel, and DIMM types supported (such as RDIMM, LRDIMM, and so on).
- *Plug-in IO cards*: Number of PCIe cards needed for SSD, NIC, and accelerators; form factors; PCIe bandwidth and power, and so on.
- *Tray-level power and cooling, and device management and security options*: Voltage regulators, cooling options (liquid versus air-cooled), board management controller (BMC), root-of-trust security, and so on.
- *Mechanical design*: Beyond the individual components, how they are assembled is also an important consideration: server form-factors (width, height, depth) as well as front or rear access for serviceability.

Figure 3.1 shows a high-level block diagram of the key components of a server tray. Figure 3.2 shows photographs of server trays using Intel and IBM processors.

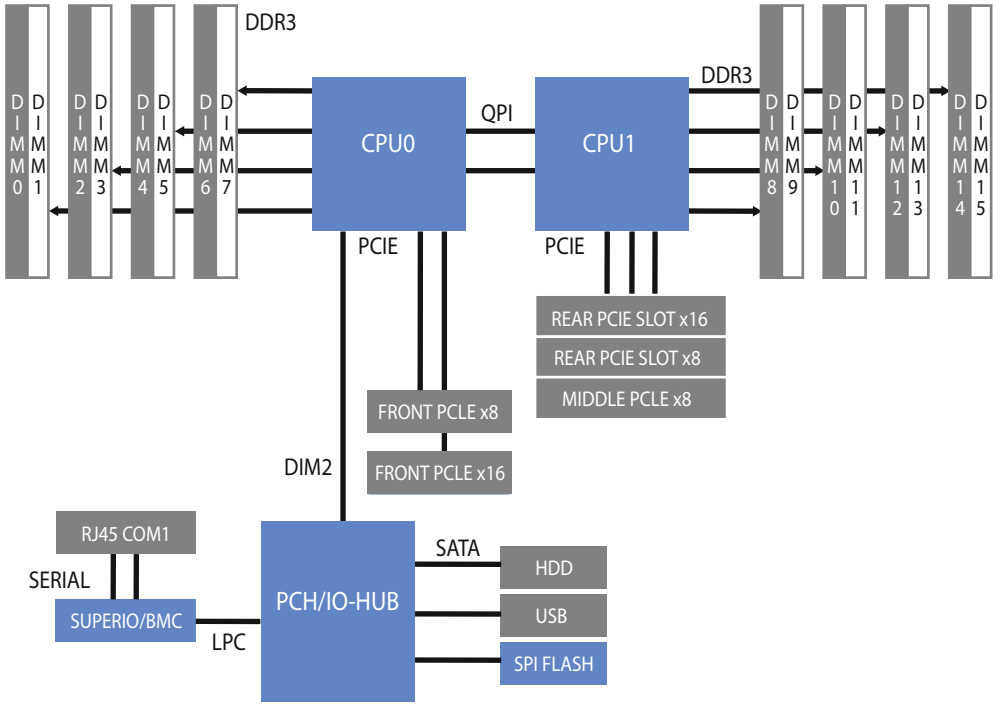


Figure 3.1: Block diagram of a server.

Going into more system details, the first x86-server supports two Intel Haswell CPU sockets, and one Wellsburg Platform Controller Hub (PCH). Each CPU can support up to 145W TDP (for example, Intel's 22 nm-based Haswell processor with 18-core per socket and 45MB L3 shared cache). The server has 16 DIMM slots, supporting up to two DIMMs per memory channel (2DPC) with ECC DRAM. With Integrated Voltage Regulators, the platforms allow per core DVFS. The system supports 80 PCIe Gen3 lanes (40 lanes per CPU), and various PCIe plug-ins with different IO width, power, and form factors, allowing it to host PCIe cards for SSD, 40 GbE NIC, accelerators. It also includes several SATA ports and supports both direct-attached storage and PCIe-attached storage appliance ("disk trays").

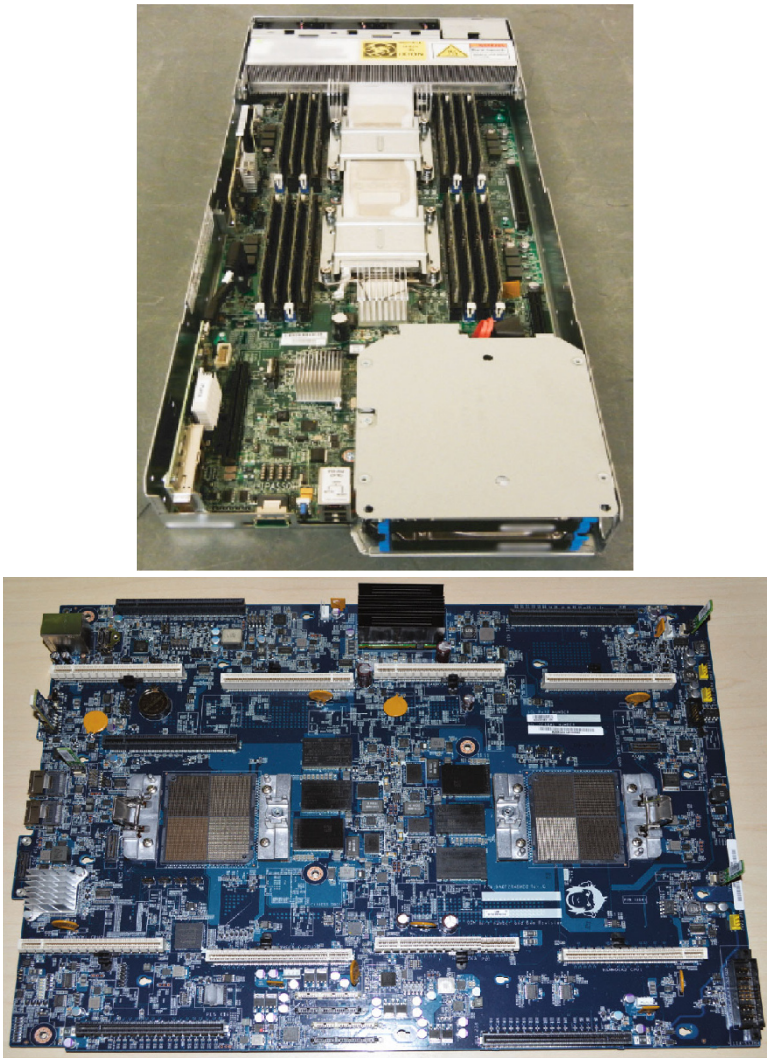


Figure 3.2: Example server trays: (top) Intel Haswell-based server tray and (bottom) IBM Power8-based server tray.

The second server is similar, except it supports IBM Power CPUs with higher thread counts and TDP, more PCIe lanes (96 lanes per two-socket), and up to 32 DDR3 DIMMs (twice that of the two-socket Intel-based system). The system maximizes the memory and IO bandwidth supported by the platform, in order to support a wide range of workloads and accelerators. We also balance the choice of CPU, in terms of core count, frequency, and cache sizes, with the available memory system bandwidth.

The rack is the physical structure that holds tens of servers together. Racks not only provide the physical support structures, but also handle shared power infrastructure, including power delivery, battery backup, and power conversion (such as AC to 48V DC). The width and depth of racks vary across WSCs: some are classic 19-in wide, 48-in deep racks, while others can be wider or shallower. The width and depth of a rack can also constrain server form factor designs. It is often convenient to connect the network cables at the top of the rack, such a rack-level switch is appropriately called a Top of Rack (TOR) switch. These switches are often built using merchant switch silicon, and further interconnected into scalable network topologies (such as Clos) described in more detail below. For example, Google's Jupiter network uses TOR switches with 64x 40 Gbps ports. These ports are split between downlinks that connect servers to the TOR, and uplinks that connect the TOR to the rest of the WSC network fabric. The ratio between number of downlinks and uplinks is called the *oversubscription ratio*, as it determines how much the intra-rack fabric is over-provisioned with respect to the data center fabric.

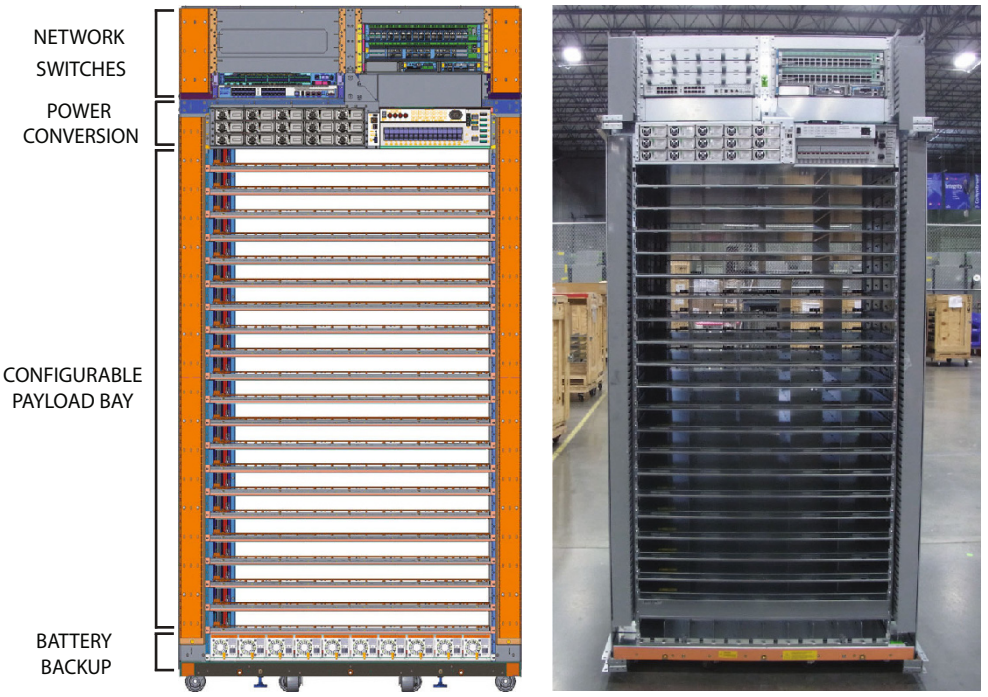


Figure 3.3: Machine racks like this support servers, storage, and networking equipment in Google's data centers.

Figure 3.3 shows an example rack in a Google data center. Aside from providing configurable physical structure for server trays with different widths and heights (for example, it can host four Haswell servers per row), it also provides the configurable power conversion and battery backup support, with redundancy, to match the IT power load. The TOR switch and rack management unit are located at the top of the rack and are further connected to a data center fabric to connect many racks.

The Open Compute Project (http://opencompute.org) also contains detailed specifications of many hardware components for WSCs.

3.1.2 THE IMPACT OF LARGE SMP COMMUNICATION EFFICIENCY

Simple processor-centric cost-efficiency analyses do not account for the fact that large Shared-Memory Multiprocessors (SMPs) benefit from drastically superior intercommunication performance than clusters of low-end servers connected by commodity fabrics. Nodes in a large SMP may communicate at latencies on the order of 100 ns, whereas LAN-based networks, usually deployed in clusters of servers, will experience latencies at or above 100 μs. For parallel applications that fit within a single large SMP (for example, SAP HANA), the efficient communication can translate into dramatic performance gains. However, WSC workloads are unlikely to fit within an SMP, therefore it is important to understand the relative performance of clusters of large SMPs with respect to clusters of low-end servers each with smaller number of CPU sockets or cores (albeit the same server-class CPU cores as used in large SMPs). The following simple model can help make these comparisons.

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 fits in only a multiple of such nodes, some global accesses will be much slower, on the order of typical LAN speeds (~100 μs). 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. 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:

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

where the variable f is the *number of global accesses per work unit* (1 ms). In Figure 3.4, 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.

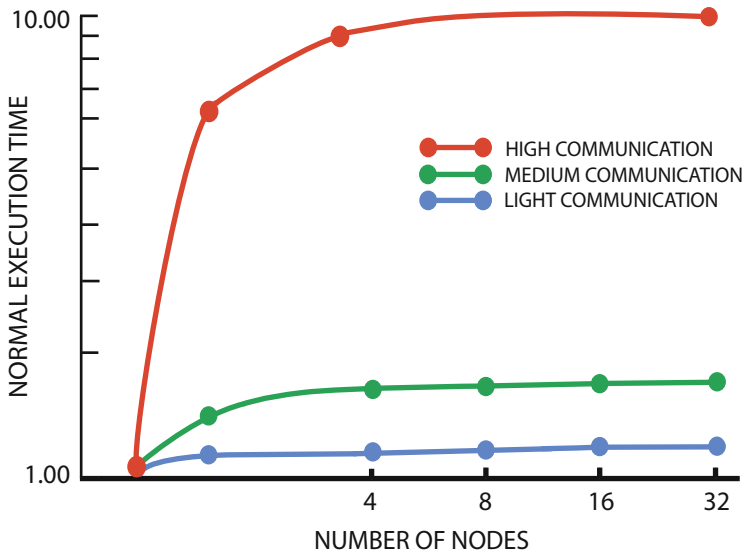


Figure 3.4: Execution time of parallel tasks as the number of SMP nodes increases for three levels of communication intensity. Execution time is normalized to the single node case and plotted in logarithmic scale.

The curves in Figure 3.4 have two interesting aspects worth highlighting. First, under light communication, there is relatively small performance degradation from using clusters of multiple nodes. For medium- and high-communication patterns, the penalties can be quite severe, but they are most dramatic when moving from a single node to two, with rapidly decreasing additional penalties for increasing the cluster size. Using this model, the performance advantage of a single 128-processor SMP over a cluster of thirty-two 4-processor SMPs could be more than a factor of 10×.

By definition, WSC systems will consist of thousands of processor cores. Therefore, we would like to use this model to compare the performance of a cluster built with large SMP servers with one built with low-end 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. Although our model is exceedingly simple (for example, it does not account for contention effects), it clearly captures the effects we are interested in.

In Figure 3.5, we apply our model to clusters varying between 512 and 4,192 cores and show the performance advantage of an implementation using large SMP servers (128 cores in a single shared memory domain) versus one using low-end servers (four-core SMPs) The figure compares the performance of clusters with high-end SMP systems to low-end systems, each having between 512 and 4,192 cores, over three communication patterns.. Note how quickly the performance edge

of the cluster based on high-end servers deteriorates as the cluster size increases. If the application requires more than 2,000 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 a performance gap this low, the price premium of the high-end server (4–20 times higher) renders it an unattractive option.

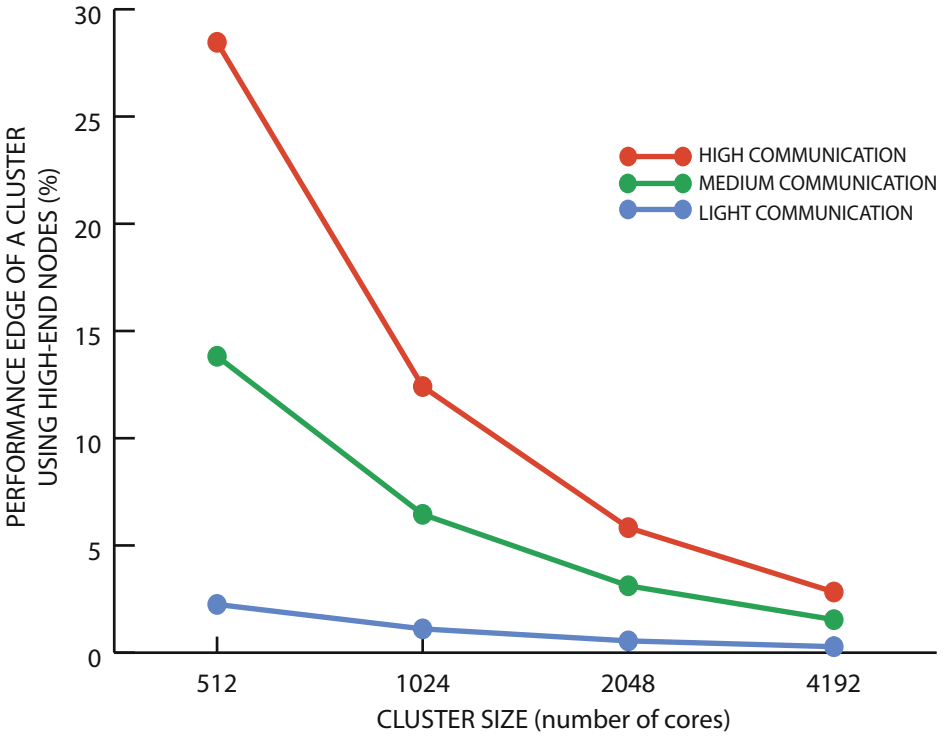


Figure 3.5: Deteriorating performance advantage of a cluster built with large SMP server nodes (128-core SMP) over a cluster with the same number of processor cores built with low-end server nodes (four-core SMP), for clusters of varying size.

The point of this analysis is qualitative in nature: it primarily illustrates how we need to reason differently about our 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 on computation, that are 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.

Since the first edition of this book was released, the number of cores per processor has steadily increased, allowing larger cluster sizes with smaller numbers of individual server nodes, and has essentially moved us toward the left side of Figure 3.5. However, given the scale of warehouse-scale systems, the discussion above still holds qualitatively. The increasing number of cores per single system does have implications on the broader system balance, and this is discussed further in the last section of this chapter. This analysis framework can also be useful in reasoning about intra-rack and data center-level networking bandwidth provisioning.

3.1.3 BRAWNY VS. WIMPY SERVERS

Clearly one could use the argument laid out above to go further and use CPU cores that are even smaller or wimpier than today's server-class CPU cores. The Piranha chip multiprocessor [Bar+00] was one of the earliest systems to advocate the use of lower-end cores in enterprise-class server systems. In [BDH03], we argued that chip multiprocessors using this approach are especially compelling for Google workloads. More recently, even more radical approaches that leverage embedded-class CPUs (wimpy processors) have been proposed as possible alternatives for WSC systems. Lim et al. [Lim+08], 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 data center build-out costs and the cost of energy). Hamilton [Ham09] makes a similar argument, although using PC-class components instead of embedded ones. The advantages of using smaller, slower CPUs are 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 cheaper with multiple smaller CPUs.
- Many applications are memory- or I/O-bound so that faster CPUs do not scale well for large applications, further enhancing the price advantage of simpler CPUs.
- Slower CPUs tend to be more power efficient; typically, CPU power decreases by $O(k^2)$ when CPU frequency decreases by a factor of k.

The FAWN (Fast Array of Wimpy Nodes) project [And+11] at Carnegie Mellon has explored the utility of wimpy cores as the basis for building an energy efficient key-value storage system, with an emphasis on flash memory. The nature of the workload in FAWN storage servers makes them a good fit for less powerful cores since the computation tends to be more I/O- and memory latency-bound than CPU-bound.

Several commercial products have also explored designs with a small number of mobile-class cores. Such systems often provide an integrated interconnect to attach disks, flash storage, and Eth-

ernet ports that can be shared among the servers. For example, HP's Moonshot Servers [HPM13] defined a blade-style chassis that can accommodate 45 server cartridges, including mobile x86 or ARM-based CPUs in a 4.3U form factor. More recently, Microsoft's Project Olympus [MPO] discussed using ARM-based CPUs for search, storage, and machine learning workloads.

In the previous edition of this book, we summarized some of the tradeoffs with very low-performing cores that can make them unattractive for WSCs (a point discussed at length by one of the authors [Höl10]. Specifically, 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 [DWG92, Lim+08]. In the limit, the amount of inherently serial work performed on behalf of a user request by extremely slow single-threaded hardware will dominate overall execution time.

Also, the more the number of threads that handle a parallelized request, the larger the *variability* in response times from all these parallel tasks, exacerbating the tail latency problem discussed in Chapter 2. One source of large performance variability that occurs on multi-core architectures is from opportunistic overclocking, with vendor-specific names such as Turbo Boost or Turbo CORE. The premise behind this feature is to run the CPU at higher frequencies when there are sufficient electrical and thermal margins to do so. The largest beneficiaries of this feature are single-threaded sequential workloads, which can receive up to 76% higher CPU frequency than the nominal processor frequency.[3] This level of performance variability has several effects: it degrades Amdahl's law by inflating single-threaded performance and further exacerbates variability in response times for distributed scale-out applications by adding a complex performance dimension (number of active CPU cores, electrical, and thermal margins on the CPU). As multi-core processors continue to scale up core counts, addressing this source of heterogeneity within the system becomes a more pressing concern.

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.

[3] https://ark.intel.com/products/codename/37572/Skylake#@server

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, with as many applications as possible packed inside 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 based on local progress only, and such heuristics are naturally more conservative. As a result, local subtasks may execute longer than they might if there were better hints about global progress. Naturally, when these computations are partitioned into smaller pieces, this overhead tends to increase.

A study by Lim et al. [Lim+13] illustrates some of the possible perils of using wimpy cores in a WSC workload. The authors consider the energy efficiency of Atom-based (wimpy) and Xeon-based (brawny) servers while running a memcached server workload. While the Atom CPU uses significantly less power than the Xeon CPU, a cluster provisioned with Xeon servers outperforms one provisioned with Atom servers by a factor of 4 at the same power budget.

Also, from the perspective of cloud applications, most workloads emphasize single-VM performance. Additionally, a bigger system is more amenable to being deployed and sold as smaller VM shapes, but the converse is not true. For these reasons, cloud prefers brawny systems as well.

However, in the past few years there have been several developments that make this discussion more nuanced, with more CPU options *between* conventional wimpy cores and brawny cores. For example, several ARMv8-based servers have emerged with improved performance (for example, Cavium ThunderX2 [CTX2] and Qualcomm Centriq 2400 [QC240]), while Intel Xeon D processors use brawny cores in low-power systems on a chip (SoCs). These options allow server builders to choose from a range of wimpy and brawny cores that best fit their requirements. As a recent example, Facebook's Yosemite microserver uses the one-socket Xeon D CPU for its scale-out workloads and leverages the high-IPC cores to ensure low-latency for web serving. To reduce the network costs, the design shares one NIC among four SoC server cards.

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. We expect more options to populate the spectrum between wimpy and brawny cores, and WSC server design to continue to evolve with these design options.

3.2 COMPUTING ACCELERATORS

Historically, the deployment of specialized computing accelerators (non-general-purpose CPUs) in WSCs has been very limited. As discussed in the second edition of this book, although they promised greater computing efficiencies, such benefits came at the cost of drastically restricting the number of workloads that could benefit from them. However, this has changed recently. Traditional improvements from Moore's law scaling of general-purpose systems has been slowing down. But perhaps more importantly, deep learning models began to appear and be widely adopted, enabling specialized hardware to power a broad spectrum of machine learning solutions. WSC designs responded to these trends. For example, Google not only began to more widely deploy GPUs, but also initiated a program to build further specialized computing accelerators for deep learning algorithms [Jou+17]. Similarly, Microsoft initiated at program to deploy FPGA-based accelerators in their fleet [Put+14].

Neural network (NN) workloads (described in Chapter 2) execute extremely high numbers of floating point operations. Figure 3.6, from OpenAI, shows the growth of compute requirements for neural networks [OAI18]. Since 2013, AI training compute requirements have doubled every 3.5 months. The growth of general-purpose compute has significantly slowed down, with a doubling rate now exceeding 4 or more years (vs. 18–24 months expected from Moore's Law). To satisfy the growing compute needs for deep learning, WSCs deploy GPUs and other specialized accelerator hardware.

Project Catapult (Microsoft) is the most widely deployed example of using reconfigurable accelerator hardware to support DNNs. They chose FPGAs over GPUs to reduce power as well as the risk that latency-sensitive applications wouldn't map well to GPUs. Google not only began to widely deploy GPUs but also started a program to build specialized computing accelerators as the Tensor Processing Units (TPU) [Jou+17]. The TPU project at Google began with FPGAs, but we abandoned them when we saw that FPGAs at that time were not competitive in performance compared to GPUs, and TPUs could use much less power than GPUs while being as fast or faster, giving them potentially significant benefits over both FPGAs and GPUs.

Time to convergence is a critical metric for ML training. Faster time to convergence improves ML model development due to faster training iterations that enable efficient model architecture and hyperparameter exploration. As described inChapter 2, multiple learners or replicas are typically used to process input examples. Increasing the number of learners, however, can have a detrimental impact on model accuracy, depending upon the model and mode of training (synchronous vs. asynchronous). For deep learning inference, many applications are user-facing and have strict response latency deadlines.

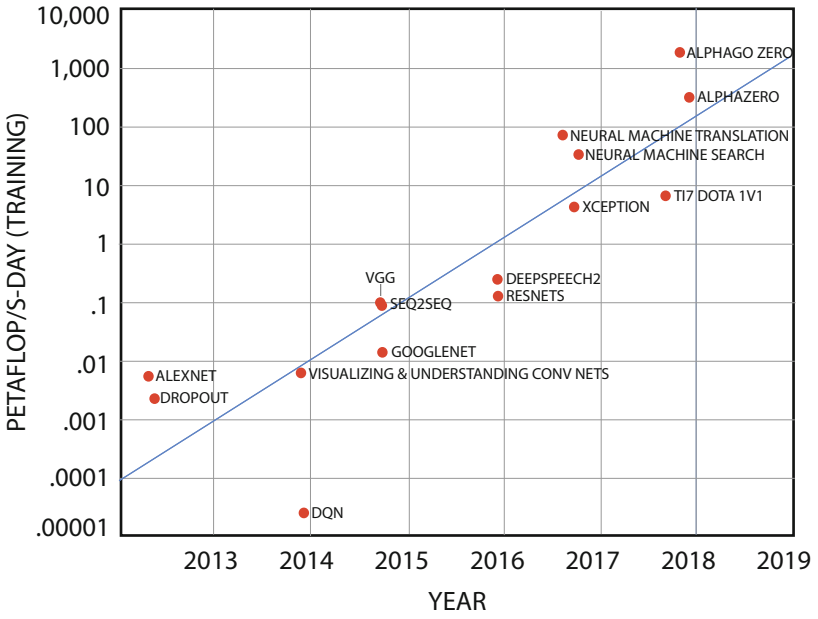


Figure 3.6: Training compute requirements for models over time [OAI18].

3.2.1 GPUs

GPUs are configured with a CPU host connected to a PCIe-attached accelerator tray with multiple GPUs. GPUs within the tray are connected using high-bandwidth interconnects such as NVlink. Multiple GPU trays are connected to the data center network with NICs. The GPUs and NICs communicate directly through PCIe without data transfer through the host. Training on GPUs can be performed synchronously or asynchronously, with synchronous training providing higher model accuracy. Synchronous training has two phases in the critical path: a compute phase and a communication phase that reconciles the parameters across learners. The performance of such a synchronous system is limited by the slowest learner and slowest messages through the network. Since the communication phase is in the critical path, a high performance network that can enable fast reconciliation of parameters across learners with well-controlled tail latencies is important for high-performance deep learning training. Figure 3.7 shows a network-connected pod of GPUs used for training.

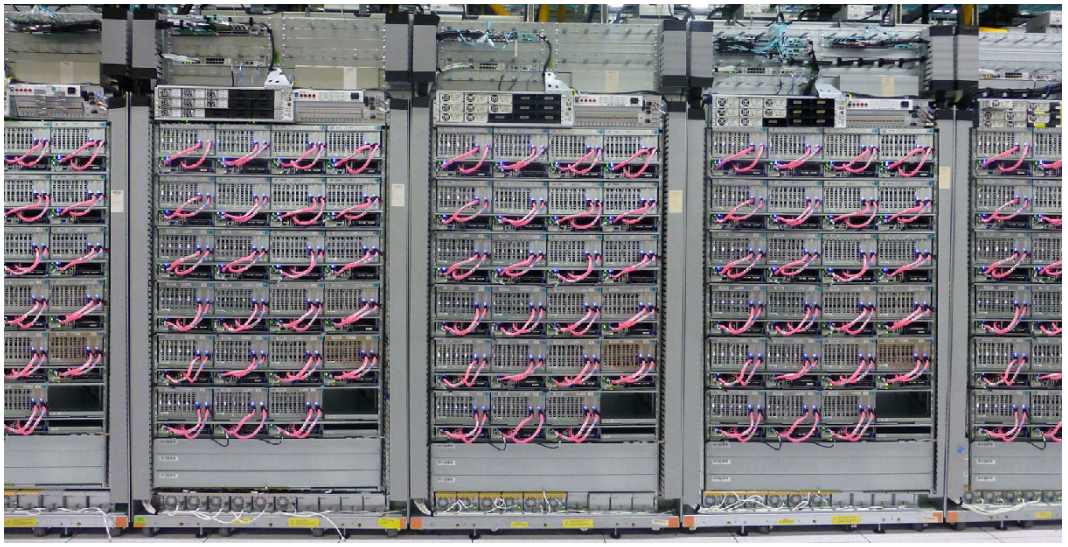


Figure 3.7: Interconnected GPUs for training.

3.2.2 TPUS

While well suited to ML workloads, GPUs still are relatively general purpose devices, and in recent years designers have further specialized them to ML-specific ASICs that drop any vestiges of graphics or high-precision functional units. TPUs are used for training and inference. TPUv1 is an inference-focused accelerator connected to the host CPU through PCIe links; a detailed architecture and performance review can be found in [Jou+17].

TPUv2, in contrast, is a very different ASIC focused on training workloads (Figure 3.8). Each TPU board is connected to one dual socket server. Inputs for training are fed to the system using the data center network from storage racks. Figure 3.8 also shows the block diagram of each TPUv2 chip. Each TPUv2 consists of two Tensor cores. Each Tensor core has a systolic array for matrix computations (MXU) and a connection to high bandwidth memory (HBM) to store parameters and intermediate values during computation.

Multiple TPUv2 accelerator boards are connected through a custom high bandwidth torus network (Figure 3.9) to provide 11 petaflops of ML compute. The accelerator boards in the TPUv2 *pod* work in lockstep to train a deep learning model using synchronous training [Dea]. The high bandwidth network enables fast parameter reconciliation with well-controlled tail latencies, allowing near ideal scalability for training across a pod [Dea].

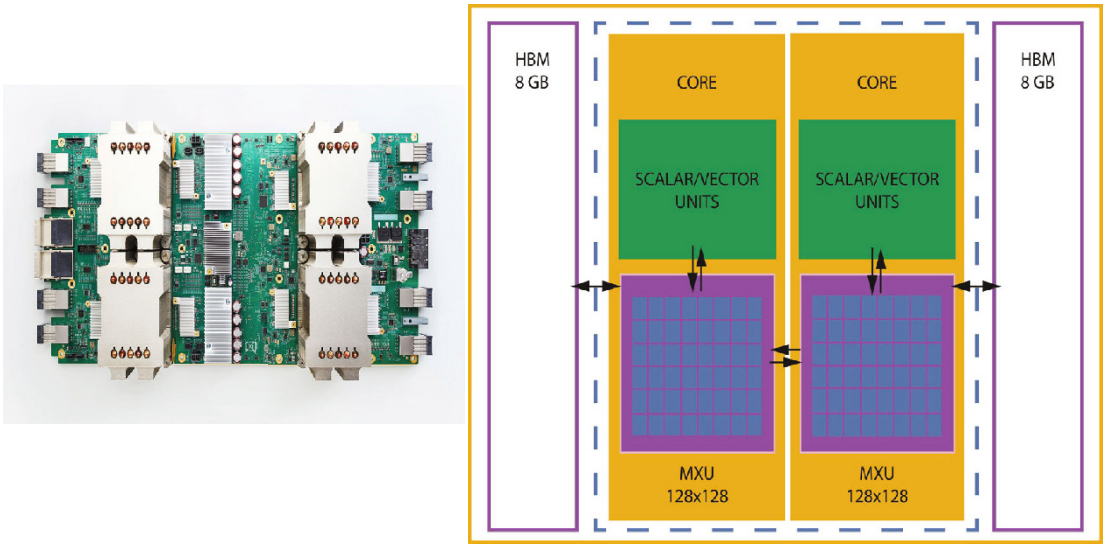


Figure 3.8: TPUv2.

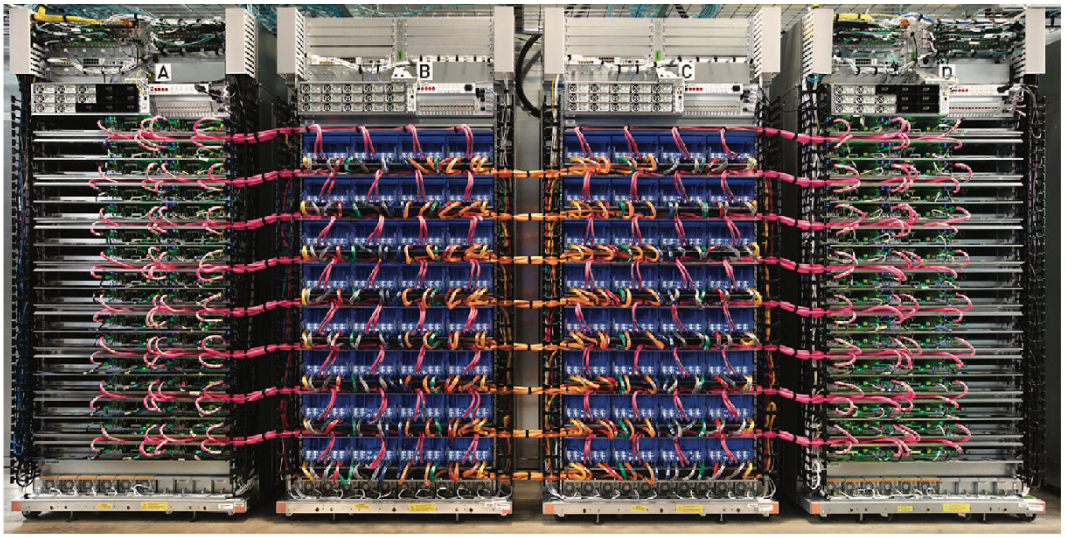


Figure 3.9: Four-rack TPUv2 pod.

TPUv3 is the first liquid-cooled accelerator in Google's data center. Liquid cooling enables TPUv3 to provide eight times the ML compute of TPUv2, with the TPUv3 pod providing more than 100 petaflops of ML compute. Such supercomputing-class computational power supports dramatic new capabilities. For example, AutoML [GCAML], coupled with the computing power

of TPUs, enables rapid neural architecture search and faster advances in ML research. Figure 3.10 shows a board with four TPUv3 chips. Figure 3.11 shows a pod of third-generation TPUs.

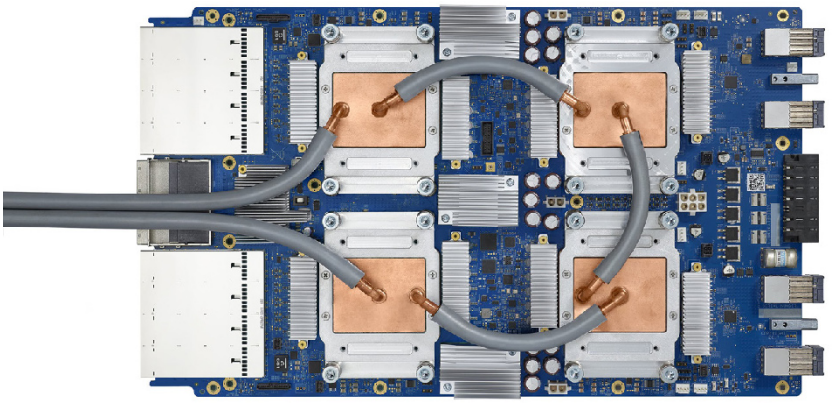


Figure 3.10: TPUv3.

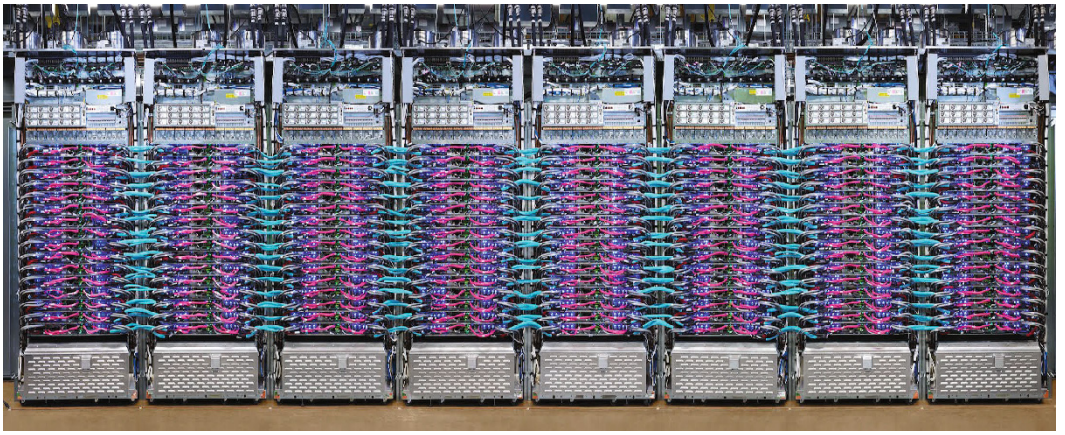


Figure 3.11: Eight-rack TPUv3 pod.

3.3 NETWORKING

3.3.1 CLUSTER 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 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 needs 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 bisection bandwidth is difficult because we can't just buy (or make) an arbitrarily large switch. Switch chips are pin- and power-limited in size; for example, a typical merchant silicon switch chip can support a bisection bandwidth of about 1 Tbps (16x 40 Gbps 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,[4] as shown in Figure 3.12 [Mys+09, Vah+10].

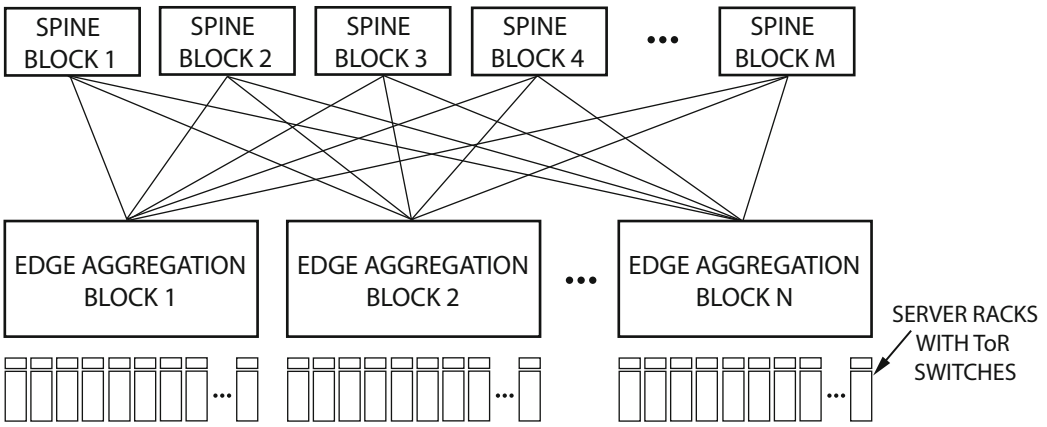


Figure 3.12: Sample three-stage fat tree topology. With appropriate scheduling this tree can deliver the same throughput as a single-stage crossbar switch.

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, the cost of doing so increases significantly because each path 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 and switch out. The above three-stage network quintuples that to 10 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 100 m 10 Gbps link can easily cost several hundred

[4] Clos networks are named after Charles Clos, who first formalized their properties in 1952.

dollars (including the cost of the two optical ports, fiber cable, fiber termination, and installation), not including the networking components themselves (switches and NICs).

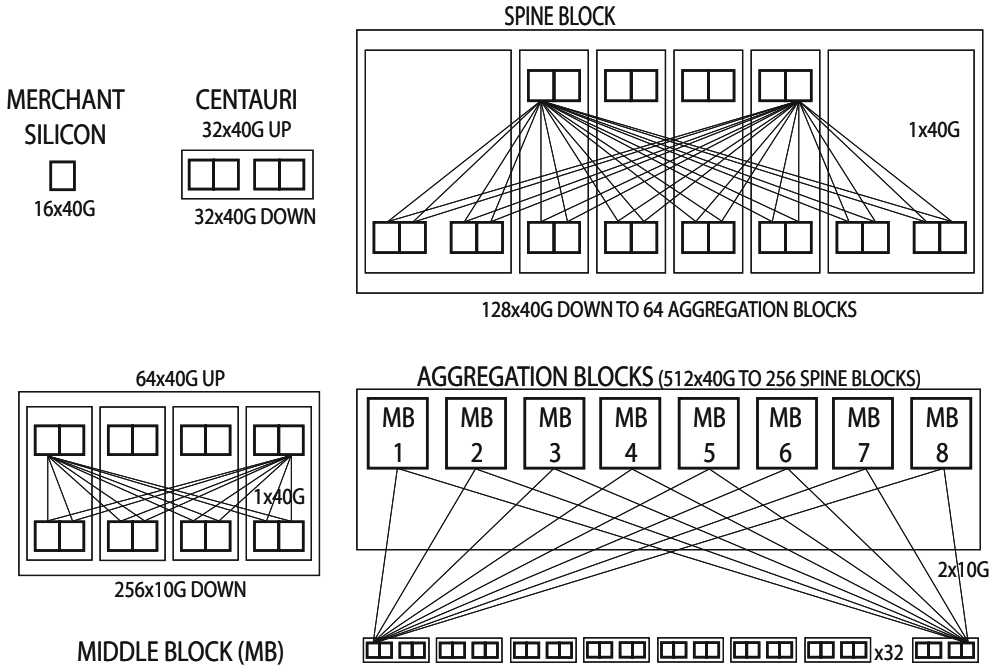
To reduce costs per machine, WSC designers often oversubscribe the network at the top-of-rack switch. Generally speaking, a rack contains a small enough number of servers so they can be connected with a switch at a reasonable cost. It's not hard to find switches with 48 ports to interconnect 48 servers at full speed (say, 40 Gbps). In a full fat tree, each such switch would need the same number of ports facing "upward" 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 upward-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 build a fat tree for only 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 40 Gbps of traffic, but if all servers are simultaneously sending traffic, they'll only be able to average 20 Gbps. In practice, oversubscription ratios of 4–10 are common. For example, a 48-port switch could connect 40 servers to 8 uplinks, for a 5:1 oversubscription (8 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 could 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 (FC [AI11]) networks rather than Ethernet. Today, Ethernet is becoming more common since it offers comparable speeds, and protocols such as FibreChannel over Ethernet) (FCoE [AI462]), SCSI over IP (iSCSI [iSC03]), and more recently NVMe over Fabric (NVMeoF [NVM]) allow Ethernet networks to integrate well with traditional SANs.

Figure 3.13 shows the structure of Google's Jupiter Clos network [Sin+15]. This multi-stage network fabric uses low-radix switches built from merchant silicon, each supporting 16x 40 Gbps ports. Each 40 G port could be configured in 4x10 G or 40 G mode. A server is connected to its ToR switch using 40 Gbps Ethernet NICs. Jupiter's primary building block is Centauri, a 4RU chassis housing two line cards, each with two switch chips. In an example ToR configuration, each switch chip is configured with 48x10 G to servers and 16x10 G to the fabric, yielding an oversubscription ratio of 3:1. Servers can also be configured with 40 G mode to have 40 G burst bandwidth.

The ToR switches are connected to layers of aggregation blocks to increase the scale of the network fabric. Each Middle Block (MB) has four Centauri chassis. The logical topology of an MB is a two-stage blocking network, with 256x10 G links available for ToR connectivity and 64x40 G available for connectivity to the rest of the fabric through the spine blocks.

Jupiter uses the same Centauri chassis as the building block for the aggregation blocks. Each ToR chip connects to eight middle blocks with dual redundant 10G links. This aids fast reconver-



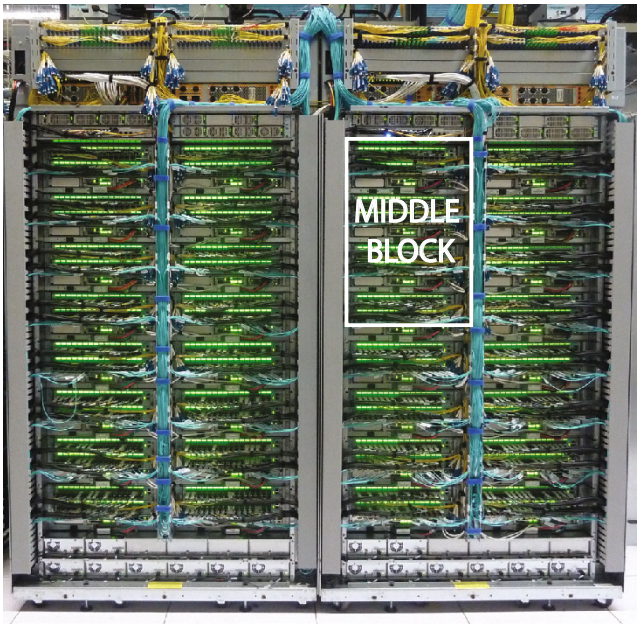


Figure 3.13: Google's Jupiter network. Starting from NIC, to ToR, to multiple switches, to campus networks. Top: (1) switch chip and Centauri chassis, (2) middle block, and (3) spine block and topology. Bottom: A Jupiter rack with middle block highlighted.

gence for the common case of single link failure or maintenance. Each aggregation block exposes 512x40 G (full pop) or 256x40 G (depop) links toward the spine blocks. Six Centauri chassis are grouped in a spine block exposing 128x40 G ports toward the aggregation blocks. Jupiter limits the size to 64 aggregation blocks for dual redundant links between each spine block and aggregation block pair at the largest scale, once again for local reconvergence on single link failure. At this maximum size, the bisection bandwidth is 1.3 petabits per second.

Jupiter employs a separate aggregation block for external connectivity, which provides the entire pool of external bandwidth to each aggregation block. As a rule of thumb, 10% of aggregate intra-cluster bandwidth is allocated for external connectivity using one to three aggregation blocks. These aggregation blocks are physically and topologically identical to those used for ToR connectivity. However, the ports normally employed for ToR connectivity are reallocated to connect to external fabrics.

With Jupiter, the intra-cluster fabric connects directly to the inter-cluster networking layer with Cluster Border Routers (CBRs). Multiple clusters are deployed within the same building and multiple buildings on the same campus. The job scheduling and resource allocation infrastructure leverages campus-level and building-level locality. The design further replaces vendor-based inter cluster switching with Freedome, a two-stage fabric that uses the Border Gateway Protocol (BGP) at both the inter-cluster and intra-campus connectivity layers to provide massive inter-cluster bandwidth within buildings and the campus at a lower cost than existing solutions.

Compared to WSCs, High-Performance Computing (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 are much more expensive.

WSCs using VMs (or, more generally, task migration) pose further challenges to networks since connection endpoints (that is, IP address/port combinations) can move from one physical machine to another. Typical networking hardware as well as network management software don'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.

The need for a programmable network has led to much interest in OpenFlow (http://www.openflow.org/), P4 (www.p4.org), and software-defined networking (SDN), which move the network control plane out of individual switches into a logically centralized controller [Höl12, ONF12, Jai+13, Sin+15]. Controlling a network from a logically centralized server offers many advantages; in particular, common networking algorithms such as computing reachability, shortest paths, or max-flow traffic placement become much simpler to solve compared to their implementation in current networks where each individual router must solve the same problem while dealing with limited visibility (direct neighbors only), inconsistent network state (routers that are out of sync with the current network state), and many independent and concurrent actors (routers). Network management operations also become simple because a global view can be used to move a network domain, often consisting of thousands of individual switches, from one consistent state to another while simultaneously accounting for errors that may require rollback of the higher-level management operation underway. Recent developments in P4 further enable a protocol- and switch-independent high-level language that allows for programming of packet-forwarding data planes, further increasing flexibility.

In addition, servers are easier to program, offering richer programming environments and much more powerful hardware. As of 2018, a typical router control processor consists of a Xeon-based 2-4 core SoC with two memory channels and 16 GB DRAM. The centralization of the control plane into a few servers also makes it easier to update their software. SDN is a natural match for data center networking, since the applications running in a WSC are already managed by a central entity, the cluster manager. Thus it is natural for the cluster manager to also configure any network elements that applications depend on. SDN is equally attractive to manage WAN networks, where logically centralized control simplifies many routing and traffic engineering problems [Höl12, Jai+13].

For more details on cluster networking, see these excellent recent overview papers: Singh et al. [Sin+15], Kumar et al. [Kum+15], Vahdat et al. [Vah+10], Abts and Felderman [AF12], and Abts and Kim [AK11].

3.3.2 HOST NETWORKING

On the host networking side, WSC presents unique challenges and requirements: high throughputs and low latency with efficient use of host CPU, low tail latencies, traffic shaping (pacing and rate limiting), OS-bypass, stringent security and line-rate encryption, debuggability, QoS and congestion control, etc. Public cloud computing further requires features such as support for virtualization and VM migration. Two general approaches can be combined to meet these requirements: onload

where the host networking software leverages host CPU to provide low latency and rich features, and offload that uses compute in NIC cards for functions such as packet processing and crypto. There has been active research and development in this area, for example Azure's use of FPGA as bump-in-the-wire [Fir+18], Amazon's customized NIC for bare-metal support, Google's host-side traffic shaping [Sae+17], and Andromeda approach toward cloud network virtualization [Dal+18].

3.4 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. In contrast, shared data must be much more durable and is accessed by a large number of clients, thus requiring a much more sophisticated distributed storage system. We discuss the main features of these WSC storage systems next.

3.4.1 DISK TRAYS AND DISKLESS SERVERS

Figure 3.14 shows an example of a disk tray used at Google that hosts tens of hard drives (22 drives in this case) and provides storage over Ethernet for servers in the WSC. The disk tray provides power, management, mechanical, and network support for these hard drives, and runs a customized software stack that manages its local storage and responds to client requests over RPC.

In traditional servers, local hard drives provide direct-attached storage and serve as the boot/logging/scratch space. Given that most of the traditional needs from the storage device are now handled by the network attached disk trays, servers typically use one local (and a much smaller) hard drive as the boot/logging device. Often, even this disk is removed (perhaps in favor of a small flash device) to avoid the local drive from becoming a performance bottleneck, especially with an increasing number of CPU cores/threads, leading to diskless servers.

A recent white paper [Bre+16] provides more details on the requirements of hard drives for WSCs, focusing on the tradeoffs in the design of disks, tail latency, and security. While current hard drives are designed for enterprise servers, and not specifically for WSC use-case, this paper argues that such "cloud disks" should aim at a global optimal in view of five key metrics: (1) higher I/Os per second (IOPS), typically limited by seeks; (2) higher capacity; (3) lower tail latency when used in WSCs; (4) meeting security requirements; and (5) lower total cost of ownership (TCO). The shift in use case and requirements also creates new opportunities for hardware vendors to explore new physical design and firmware optimizations.

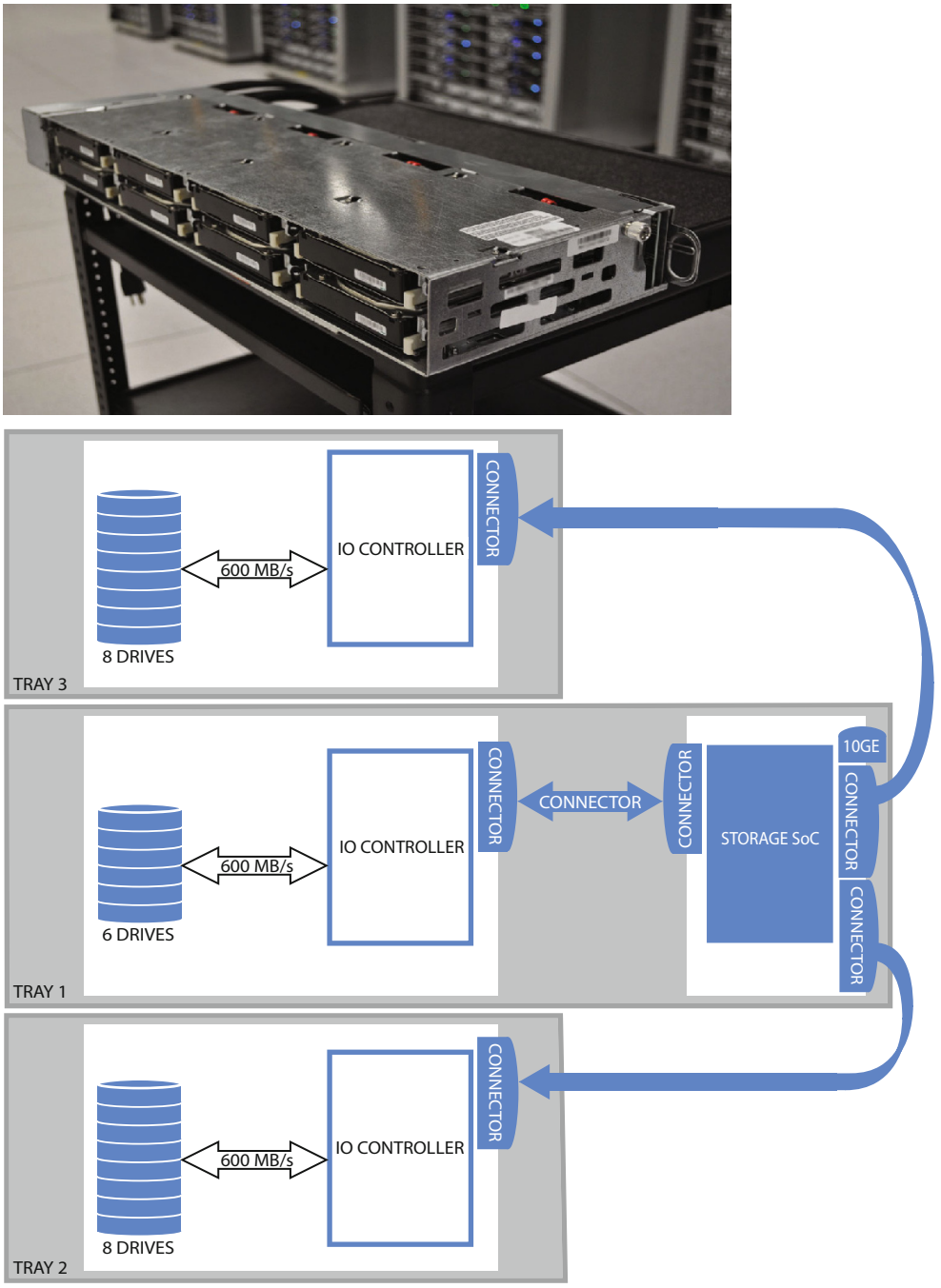


Figure 3.14: (Top) Photograph of a disk tray. (Bottom) Block diagram.

3.4.2 UNSTRUCTURED WSC STORAGE

Google's GFS [GGL03] 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 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 on high throughput for thousands of concurrent readers/writers 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 primary server (master), which handles metadata operations, and thousands of chunkserver (secondary) 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 of 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.

Although the initial version of GFS supported only simple replication, Colossus and its externally available cousin GCS have 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. Colossus optimizes for 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, in case additional faults hit the cluster. A comprehensive study of availability in distributed file systems at Google can be found in Ford et al. [For+10]. A good discussion of the evolution of file system design at Google can also be found in McKusik and Quinlan [McKQ09].

3.4.3 STRUCTURED WSC STORAGE

The simple file abstraction of Colossus and GCS may suffice 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 [Cha+06] and Amazon's DynamoDB [DeC+07], were designed to fulfill those needs. Compared to traditional database systems, Bigtable and DynamoDB sacrifice some features, such as the 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 DynamoDB 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 [Bak+11] and subsequently Spanner [Cor+12] were 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, providing a simpler consistency model to applications that need seamless wide-area replication for fault tolerance. Both Megastore and Spanner sacrifice some efficiency in order to provide a simpler programming interface.

Clickstream and Ads data management, for example, is an important use-case of structured storage systems. Such systems require high availability, high scalability of NoSQL systems, and the consistency and usability of SQL databases. Google's F1 system [Shu+13] uses Spanner as datastore, and manages all AdWords data with database features such as distributed SQL queries, transactionally consistent secondary indexes, asynchronous schema changes, optimistic transactions, and automatic change history recording and publishing. The Photon [Ana+13] scalable streaming system supports joining multiple continuously flowing streams of data in real-time with high scalability and low latency, with exactly-once semantics (to avoid double-charging or missed clicks) eventually.

At the other end of the structured storage spectrum from Spanner are systems that aim 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 [Fit+03], developed as a distributed DRAM-based object caching layer, is a popular example at the simplest end of the spectrum. The Stanford RAMCloud [Ous+09] system also uses a distributed DRAM-based data store but aims at much higher performance (over one million lookup operations per second per server) as well as durability in the presence of storage node failures. The FAWN-KV [And+11] system also presents a key-value high-performance storage system but instead uses NAND flash as the storage medium, and has an additional emphasis on energy efficiency, a subject we cover more extensively in Chapter 5.

3.4.4 INTERPLAY OF STORAGE AND NETWORKING TECHNOLOGY

The success of WSC distributed storage systems can be partially attributed to the evolution of data center networking fabrics. Ananthanarayanan et al. [Ana+11] observe that the gap between networking and disk performance has widened to the point that disk locality is no longer relevant in intra-data center 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.

Flash devices pose a new challenge for data center networking fabrics. A single enterprise flash device can achieve well over 100x the operations throughput of a disk drive, and one server machine with multiple flash SSDs could easily saturate a single 40 Gb/s network port even within a rack. Such performance levels will stretch not only data center fabric bisection bandwidth but also require more CPU resources in storage nodes to process storage operations at such high rates. Looking ahead, rapid improvements in WSC network bandwidth and latency will likely match flash SSD performance and reduce the importance of flash locality. However, emerging non-volatile memory (NVM) has the potential to provide even higher bandwidth and sub-microsecond access latency. Such high-performance characteristics will further bridge the gap between today's DRAM and flash SSDs, but at the same time present an even bigger challenge for WSC networking.

3.5 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 discussed many examples of how the right building blocks are apparent only when one considers 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 dollars per 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 three to four years). This issue is even more important for the WSC as a whole because the lifetime of a data center 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.

3.5.1 SYSTEM BALANCE: STORAGE HIERARCHY

Figure 3.15 shows a programmer's view of storage hierarchy of a hypothetical WSC. As discussed earlier, the server consists of a number of processor sockets, each with a multicore CPU and its internal cache hierarchy, local shared and coherent DRAM, a number of directly attached disk drives, and/or flash-based solid state drives. The DRAM and disk/flash resources within the rack are accessible through the first-level rack switches (assuming some sort of remote procedure call API to them exists), and all resources in all racks are accessible via the cluster-level switch. The relative balance of various resources depends on the needs of target applications. The following configuration assumes an order of magnitude less flash capacity than traditional spinning media since that is roughly the relative cost per byte difference between these two technologies.

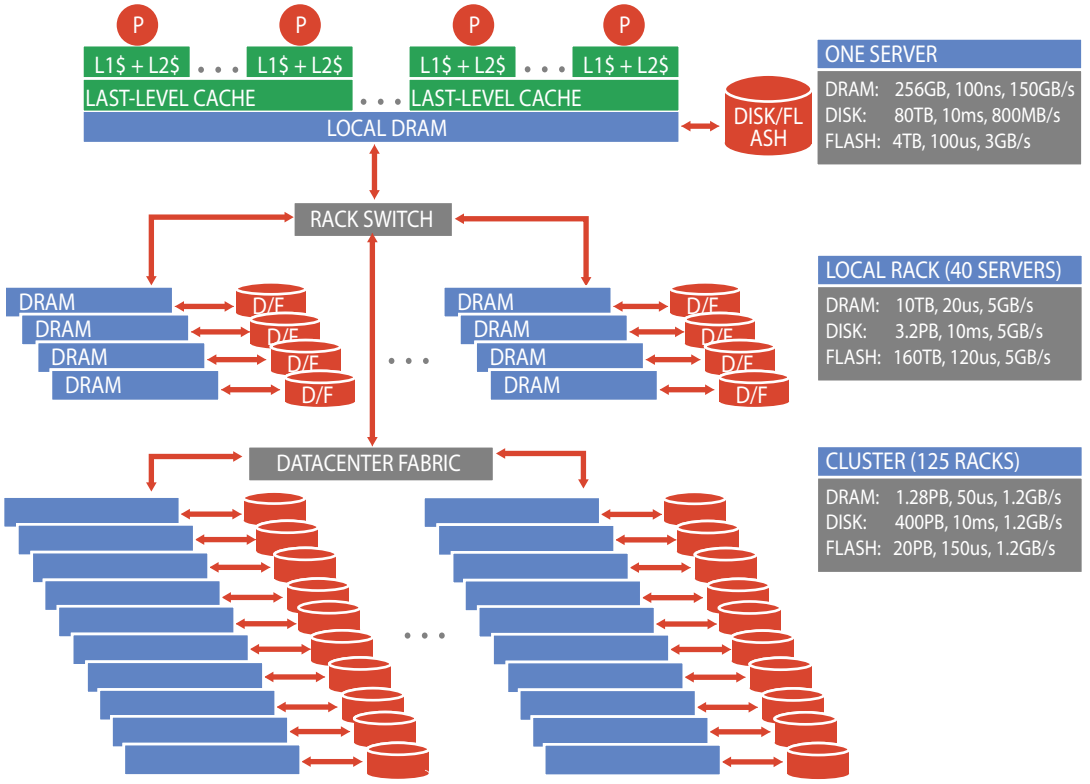


Figure 3.15: Storage hierarchy of a WSC.

3.5.2 QUANTIFYING LATENCY, BANDWIDTH, AND CAPACITY

Figure 3.16 attempts to quantify the latency, bandwidth, and capacity characteristics of a WSC. For illustration we assume a system with 5,000 servers, each with 256 GB of DRAM, one 4 TB SSD, and eight 10 TB disk drives. Each group of 40 servers is connected through a 40-Gbps link to a rack-level switch that has an additional 10-Gbps uplink bandwidth per machine for connecting the rack to the cluster-level switch (an oversubscription factor of four). Network latency numbers assume a 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. For disks, we show typical commodity disk drive (SATA) latencies and transfer rates.

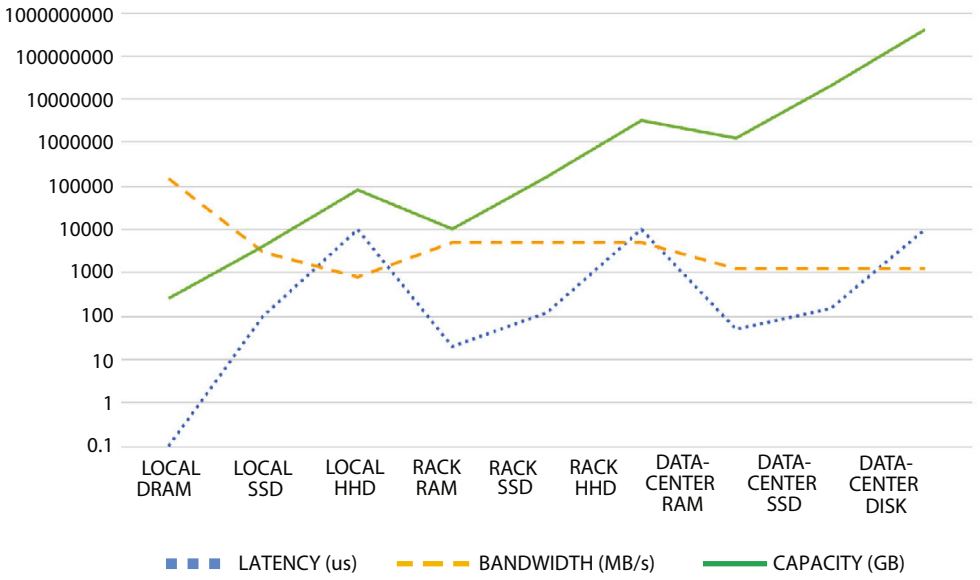


Figure 3.16: Latency, bandwidth, and capacity of WSC storage hierarchy levels.

The graph shows the relative latency, bandwidth, and capacity of each resource pool. For example, the bandwidth available from local SSDs is about 3 GB/s, whereas the bandwidth from off-rack SSDs is just 1.25 GB/s via the shared rack uplinks. On the other hand, total disk storage in the cluster is more than one 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. For example, NAND flash technology, originally developed for portable electronics, has found target use cases in WSC systems. Flash-based SSDs are a viable option for bridging the cost and performance gap between DRAM and disks, as displayed in Figure 3.16. 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 demands much higher bandwidth from the WSC fabric, as well as microsecond performance support from the hardware/software stack. Note that in the worst case, writes to flash can be several orders of magnitude slower than reads, and garbage collection can further increase write amplification and tail latency. Characteristics of flash around read/write

asymmetry, read/write interference, and garbage collection behaviors introduce new challenges and opportunities in adopting low-latency storage tiers in a balanced WSC design.

Emerging technologies such as non-volatile memories (NVM) (for example, Intel 3D Xpoint based memory [3DX] and fast SSD products such as Samsung Z-NAND [Sam17]) add another tier between today's DRAM and flash/storage hierarchy. NVM has the potential to provide cheaper and more scalable alternatives to DRAM, which is fast approaching its scaling bottleneck, but also presents challenges for WSC architects who now have to consider data placement, prefetching, and migration over multiple memory/storage tiers. NVM and flash also present new performance and efficiency challenges and opportunities, as traditional system design and software optimizations lack support for their microsecond (μs)-scale latencies. A new set of hardware and software technologies are needed to provide a simple programming model to achieve high performance [Bar+17].