Chapter 6 Solutions

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

6.1 a. The servers being 10% faster means that fewer servers are required to achieve the same overall performance. From the numbers in Figure 6.13, there are initially 45,978 servers. Assuming a normalized performance of 1 for the baseline servers, the total performance desired is thus 45,978 × 1 = 45,978. The new servers provide a normalized performance of 1.1, thus the total servers required, y, is y × 1.1 = 45,978. Thus y = 41,799. The price of each server is 20% more than the baseline, and is thus 1.2 × 1450 = $1,740. Therefore the server CAPEX is $72,730,260.

b. The servers use 15% more power, but there are fewer servers as a result of their higher performance. The baseline servers use 165 W, and thus the new servers use 1.15 × 165 = 190 W. Given the number of servers, and therefore the network load with that number of servers (377,795 W), we can determine the critical load needed to support the higher powered servers. Critical load = number of servers × watts per server + network load. Critical load = 41,799 × 190 W + 377,795 W. Critical load = 8,319,605 W. With the same utilization rate and the same critical load usage, the monthly power OPEX is $493,153.

c. The original monthly costs were $3,530,920. The new monthly costs with the faster servers is $3,736,648, assuming the 20% higher price. Testing different prices for the faster servers, we find that with a 9% higher price, the monthly costs are $3,536,834, or roughly equal to the original costs.

6.2 a. The low-power servers will lower OPEX but raise CAPEX; the net benefit or loss depends on how these values comprise TCO. Assume server operational and cooling power is a fraction x of OPEX and server cost is a fraction y of CAPEX. Then, the new OPEX, CAPEX, and TCO, by using low-power servers, are given by:

$$OPEX' = OPEX[(1 - 0.15)x + (1 - x)]$$
$$CAPEX' = CAPEX[(1 + 0.2)y + (1 - y)]$$
$$TCO' = OPEX' + CAPEX' = OPEX(1 - 0.15x) + CAPEX(1 + 0.2y)$$

For example, for the data in Figure 6.14, we have

$$OPEX' = \$560{,}000(1 - 0.15 \times 0.87) = \$486{,}920$$
$$CAPEX' = \$3{,}225{,}000(1 + 0.2 \times 0.62) = \$3{,}624{,}900$$
$$TCO' = OPEX' + CAPEX' = \$4{,}111{,}820$$

In this case, TCO > TCO′, so the lower power servers are not a good tradeoff from a cost perspective.

b. We want to solve for the fraction of server cost, S, when TCO = TCO′:

$$TCO = TCO' = OPEX(1 - 0.15x) + CAPEX(1 + Sy)$$
$$TCO - OPEX(1 - 0.15x) = CAPEX(1 + Sy)$$

$$S = \frac{TCO - OPEX(1 - 0.15x) - CAPEX}{CAPEXy}$$

We can solve for this using the values in Figure 6.14:

$$S = \frac{\$3{,}800{,}000 - \$560{,}000(1 - 0.15 \times 0.87) - \$3{,}225{,}000}{\$3{,}255{,}000(0.62)} \approx 0.044$$

So, the low-power servers need to be less than 4.4% more expensive than the baseline servers to match the TCO.

If the cost of electricity doubles, the value of x in the previous equations will increase as well. The amount by which it increases depends on the other costs involved. For the data in Figure 6.14, the cost of electricity doubling will make the monthly power use equal to \$475,000 × 2 = \$950,000. This increases total cost to be \$4,275,000 and gives a value of $x \approx 0.92$.

$$S = \frac{\$4{,}275{,}000 - \$1{,}035{,}000(1 - 0.15 \times 0.92) - \$3{,}225{,}000}{\$3{,}255{,}000(0.62)} \approx 0.079$$

At this value of x, $S \approx 7.9\%$, slightly more than before. Intuitively this makes sense, because increasing the cost of power makes the savings from switching to lower-power hardware greater, making the break-even point for purchasing low-power hardware less aggressive.

6.3 a. The baseline WSC used 45,978 servers. At medium performance, the performance is 75% of the original. Thus the required number of servers, x, is $0.75 \times x = 45{,}978$. $x = 61{,}304$ servers.

b. The server cost is remains the same baseline \$1,450, but the per-server power is only 60% of the baseline, resulting in 0.6 × 165 W = 99 W per server. These numbers yield the following CAPEX numbers: Network: \$16,444,934.00; Server: \$88,890,800.00. Like in 6.1, we calculate the total critical load as Critical load = number of servers × watts per server + network load. Critical load = 61,304 × 99 W + 531,483 W. Critical load = 6,600,579 W. This critical load, combined with the servers, yields the following monthly OPEX numbers: Power: \$391,256; Total: \$4,064,193.

c. At 20% cheaper, the server cost is 0.8 × \$1,450 = \$1,160. Assuming a slowdown of X%, the number of servers required is 45,978/(1 – X). The network components required can be calculated through the spreadsheet once the number of servers is known. The critical load required is Critical load = number of servers × watts per server + network load. Given a reduction of power Y% compared to the baseline server, the watts per server = (1 – Y) × 165 W. Using these numbers and plugging them into the spreadsheet, we can find a few points where the TCO is approximately equal.

6.4 For a constant workload that does not change, there would not be any performance advantage of using the normal servers at medium performance versus the slower but cheaper servers. The primary differences would be in the number of

servers required for each option, and given the number of servers, the overall TCO of each option. In general, the server cost is one of the largest components, and therefore reducing the number of servers is likely to provide greater benefits than reducing the power used by the servers.

6.5 Running all of the servers at medium power (option 1) would mean the WSC could handle higher peak loads throughout the day (such as the evening in populated areas of the world when internet utilization increases), compared to the cheaper but slower servers (option 2).

This resiliency to spikes in load would come at the cost of increased CAPEX and OPEX cost, however. A detailed analysis of peak and average server load should be considered when making such a purchasing decision.

6.6 There are multiple abstractions made by the model. Some of the most significant abstractions are that the server power and datacenter power can be represented by a single, average number. In reality, they are likely to show significant variation throughout the day due to different usage levels. Similarly, all servers may not have the same cost, or identical configurations. They may be purchased over a time period, leading to multiple different kinds of servers or different purchase prices. Furthermore, they may be purchased with different usages in mind, such as a storage-oriented server with more hard drives or a memory-capacity oriented server with more RAM. There are other details that are left out, such as any licensing fees or administration costs for the servers. When dealing with the large number of servers in a WSC, it should be safe to make assumptions about average cost across the different systems, assuming there are no significant outliers. There are additional assumptions in the cost of power being constant (in some regions its pricing will vary throughout the day). If there are significant variations in power pricing and power usage throughout the day, then it can lead to significantly different TCO versus a single average number.

Case Study 2: Resource Allocation in WSCs and TCO

6.7 a. Consider the case study for a WSC presented in Figure 6.13: We must provision 8 MW for 45,978 servers. Assuming no oversubscription, the nameplate power for each server is:

$$P_{nameplate} = \frac{8 \text{ MW}}{45{,}978 \text{ servers}} \approx 174 \frac{\text{W}}{\text{server}}$$

In addition, the cost per server is given in Figure 6.13 as $1450.

Now, if we assume a new nameplate server power of 200 W, this represents an increase in nameplate server power of 200 – 174 = 26 W/server, or approximately 15%. Since the average power utilization of servers is 80% in this WSC, however, this translates to a 15% × 80% = 12% increase in the cost of server power. To calculate the percentage increase in power and cooling, we consider the power usage effectiveness of 1.45. This means that an increase of server power of 12% translates to a power and cooling increase of 12% × 1.45 = 17.4%.

If we also assume, as in Figure 6.13, that the original cost per server is \$1,450, a new server would cost \$3,000 – \$1,450 = \$1,550 more, for an increase of about 109%.

We can estimate the effects on TCO by considering how power and cooling infrastructure and server cost factor into monthly amortized CAPEX and OPEX, as given in Figure 6.14. Increasing the cost of power and cooling infrastructure by 17.4% gives a new monthly cost of \$475,000 × 1.174 = \$557,650 (an increase of \$82,650) and increasing server cost by 109% gives a new monthly cost of \$2,000,000 × 2.09 = \$4,180,000 (an increase of \$2,180,000).

The new monthly OPEX increases by \$82,650 + \$2,180,000 = \$2,262,650.

b. We can use a similar process as in Part a, to estimate the effects on TCO for this cheaper but more power-hungry server:

Nameplate server power increases by 300 – 174 = 126 W/server, or approximately 72%. This translates to an increase in power and cooling of 72% × 1.45 ≈ 104%.

In addition, a new server would cost \$2,000 – \$1,450 = \$550 more, for an increase of about 38%.

We can once again estimate the effects on TCO by calculating the new monthly cost of power and cooling infrastructure as \$475,000 × 1.72 = \$817,000 (an increase of \$342,000) and the new monthly server cost as \$2,000,000 × 1.38 = \$2,760,000 (an increase of \$760,000).

The new monthly OPEX increases by \$342,000 + \$760,000 = \$1,102,000. This option does not increase TCO by as much as that in Part a.

c. If average power usage of the servers is only 70% of the nameplate power (recall in the baseline WSC it was 80%), then the average server power would decrease by 12.5%. Given a power usage effectiveness of 1.45, this translates to a decrease in server power and cooling of 12.5% × 1.45 = 18.125%. This would result in a new monthly power and cooling infrastructure cost of \$475,000 × (1 – 0.18125) = \$388,906.25 (a decrease of \$86,093.75).

6.8 a. Assume, from Figure 6.13, that our WSC initially contains 45,978 servers. If each of these servers has a nameplate power of 300 W, the critical load power for servers is given by 45,978 × 300 = 13,793,400 W.

If, in actuality, the servers had an average power consumption of 225 W, the average power load of the servers in the WSC would be 45,978 × 225 = 10,345,050 W.

This means that (13,793,400 – 10,345,050)/13,793,400 = 25% of the provisioned power capacity remains unused. The monthly cost of power for such servers, assuming \$0.07 per KWh, from Figure 6.13, and 720 hours per month, is $10{,}345.05 \text{ KW} \times 0.07 \frac{\$}{\text{KWh}} \times 720 \text{ h} = \$521{,}390.52$.

b. If we instead assume the nameplate power of the server to be 500 W, the critical load power for servers would be given by $45{,}978 \times 500 = 22{,}989{,}000$ W.

If, in actuality, the servers had an average power consumption of 300 W, the average power load of the servers in the WSC would be $45{,}978 \times 300 = 13{,}793{,}400$ W.

This means that $(22{,}989{,}000 - 13{,}793{,}400)/22{,}989{,}000 = 40\%$ of the provisioned power capacity remains unused. The monthly cost of power for such servers, assuming \$0.07 per KWh, from Figure 6.13, and 720 hours per month, is $13{,}793.4 \text{ KW} \times 0.07 \frac{\$}{\text{KWh}} \times 720 \text{ h} = \$695{,}187.36$.

6.9 Assuming infrastructure can be scaled perfectly with the number of machines, the *per-server* TCO of a data-center whose capacity matches its utilization will not change.

In general, however, it depends on how a variety factors such as power delivery and cooling scale with the number of machines. For example, the power consumption of data-center cooling infrastructure may not scale perfectly with the number of machines and, all other things equal, could require a larger per-server TCO for adequately cooling higher data-center capacities. Conversely, server manufacturers often provide discounts for servers bought in bulk; all other things equal, the per-server TCO of a data-center with a higher capacity could be less than that of a lower capacity.

6.10 During off-peak hours the servers could potentially be used for other tasks, such as off-line batch processing jobs. Alternatively, the operators could try to sell the excess capacity by offering computing services that are priced cheaper during the off-peak hours versus peak hours. Other options to save cost include putting the servers into low-power modes, or consolidating multiple workloads onto less servers and switching off the now idle servers. If there are certain classes of servers that can more efficiently serve the smaller load (such as Atom-based servers versus Xeon-based servers), then they could be used instead. However, each of these options effectively reduce compute capacity available during the more idle periods. If the workload happens to have a spike in activity (e.g., a large news event leads to significant traffic during an off-peak period), then these options run the risk of not having enough compute capacity available for those spikes. Switching to low-power modes will allow the servers to more quickly respond to higher spikes in load rather than consolidation and shutting off of servers, but will provide less cost and power savings.

6.11 There are many different possible proposals, including: server consolidation, using low-power modes, using low-power processors, using embedded-class processors, using low-power devices such as solid-state disks, developing energy-proportional servers, and others. The challenges to many of these proposals is developing models that are detailed enough to capture all of the impacts of the different operational modes, as well as having accurate workload traces to drive the power models. Based on the proposals, some advantages include lower power usage or better response times. Potential disadvantages include inflexibility, lower performance, or higher CAPEX costs.

Exercises

6.12 a. One example of when it may be more beneficial to improve the instruction- or thread-level parallelism than request-level parallelism is if the latency of a workload is more important than the throughput. While request-level parallelism can enable more concurrent requests to be processed (increasing system throughput), without improving instruction- or thread-level parallelism, each of those requests will be processed at a slower speed.

b. The impact of increasing request-level parallelism on software design depends on the workload being parallelized. The impact can range from minimal—for applications which do not keep any state and do not communicate with multiple instances of themselves, such as web servers—to far-reaching—for applications which require fine-grained communicate or shared state, such as database software. This is because by increasing request-level parallelism, communication and state must be shared across racks in a datacenter or even datacenters across the world, introducing a range of issues such as data consistency and redundancy.

c. In addition to the software design overheads discussed in Part b., one example of the potential drawbacks of increasing request-level parallelism, is that more machines will be required to increase system throughput, which, for a fixed TCO, may require the purchase of more inexpensive, commodity machines. Such machines may fail more frequently than more expensive machines optimized for high instruction- and thread-level parallelism, and mechanisms will have to be put in place to counteract these machine failures to prevent the loss of work and data.

6.13 a. At a high level, one way to think about the effect of round-robin scheduling for compute-heavy workloads is that it more evenly spreads the amount of computation across a given number of machines compared to consolidated scheduling. There are a variety of trade-offs present by making such a scheduling decision, and we present several here for discussion.

In terms of power and cooling, round-robin scheduling may decrease the power density of a given group of servers, preventing hot spots from forming in the data center and possibly requiring less aggressive cooling solutions to be employed. On the other hand, modern server power supplies are not very efficient at low utilizations (this means more power is lost in conversion for lower utilizations than for higher utilizations), so under-utilizing many machines can lead to losses in power efficiency.

In terms of performance, round-robin scheduling will likely benefit compute-heavy workloads because there will be no resource sharing between processes, in this example, which would otherwise lead to performance degradation. However, if processes accessed the same data, they might benefit from being located on the same machine, as one process could fetch data, making it readily available by the time the other process needed to use it.

For reliability, round robin scheduling will decrease the number of jobs lost due to machine failure (assuming machine failures occur on machines running jobs). Placing jobs which must communicate with one another across many machines, however, may introduce new points of failure, such as networking equipment.

b. In general, the effect of round-robin scheduling I/O-heavy workloads will depend on how data is laid out across the servers and the access patterns of the workload itself. For example, if data are replicated across different servers, the scheduling decision will have less of an effect on performance than if the data are partitioned across racks. In the first case, wherever a process is scheduled, it will be able to access its data locally; in the second case, processes must be scheduled on the same rack that their data is located on in order to not pay the cost of traversing the array of racks to access their data.

c. Assuming the bandwidth available at the networking hardware used to connect the servers being used is limited, and the topology of the network used to connect the servers is tree-like (e.g., a centralized switch connecting the racks and localized switches connecting servers within racks), scheduling the jobs in a round-robin fashion at the largest scope (array of racks) can help balance the bandwidth utilization for requests across each of the racks. If the applications are network-intensive because they communicate with one another, however, round-robin scheduling will actually create additional traffic through the switches used to connect various servers across the racks.

6.14 a. Total dataset size is 300 GB, network bandwidth 1 Gb/s, map rate is 10 s/GB, reduce rate is 20 s/GB. 30% of data will be read from remote nodes, and each output file is written to two other nodes. According to Figure 6, disk bandwidth is 200 MB/s. For simplicity, we will assume the dataset is broken up into an equal number of files as there are nodes. With 5 nodes, each node will have to process 300 GB/5 = 60 GB. Thus 60 GB × 0.3 = 18 GB must be read remotely, and 42 GB must be read from disk. Using the disk bandwidths from Figure 6.6, we calculate the time for remote data access as 18 GB/100 MB/s = 180 seconds, and the time for local data access as 42 GB/200 MB/s = 210 seconds; the data must be accessed for both the map and the reduce. Given the map rate, the map will take 60 GB × 10 s/GB = 600 seconds; given the reduce rate, the reduce will take 60 GB × 20 s/GB = 1200 seconds. The total expected execution time is therefore (180 + 210) × 2 + 600 + 1200 = 2508 seconds. The primary bottleneck is the reduce phase, while the total data transfer time is the secondary bottleneck. At 1,000 nodes, we have 300 GB/1000 = 300 MB per node. Thus 300 MB × 0.3 = 90 MB must be read remotely, and 210 MB must be read from local disk. These numbers give the following access times: network. = 90 MB/100 MB/s = 0.9 seconds, and disk = 210 MB/200 MB/s = 1.05 seconds. The map time is 300 MB × 10 s/GB = 3 seconds, and the reduce time is 300 MB × 20 s/GB = 6 seconds. The bottlenecks actually remain identical across the different node sizes as all communication remains local within a rack and the problem is divided up evenly.

b. With 40 nodes per rack, at 100 nodes there would be 3 racks. Assume an equal distribution of nodes (~33 nodes per rack). We have 300 GB/100 = 3 GB per node. Thus 900 MB must be read remotely, and there is a 2/3 chance it must go to another rack. Thus 600 MB must be read from another rack, which has disk bandwidth of 10 MB/s, yielding 60 seconds to read from a different rack. The 300 MB to read from within the rack will take 300 MB/100 MB/s = 3 seconds. Finally the time to do the map is 30 seconds, and the reduce is 60 seconds. The total runtime is therefore $(60 + 3) \times 2 + 30 + 60 = 216$. Data transfer from machines outside of the rack wind up being the bottleneck.

c. Let us calculate with 80% of the remote accesses going to the same rack. Using the numbers from b., we get 900 MB that must be read remotely, of which 180 MB is from another rack, and 720 MB is within a node's rack. That yields 180 MB/10 MB/s = 18 seconds to read remote data, and 720 MB/100 MB/s = 7.2 seconds to read local data. The total runtime is therefore $(18 + 7.2) \times 2 + 30 + 60 = 140.4$ seconds. Now the reduce phase is the bottleneck.

d. The MapReduce program works by initially producing a list of each word in the map phase, and summing each of the instances of those words in the reduce phase. One option to maximize locality is, prior to the reduce phase, to sum up any identical words and simply emit their count instead of a single word for each time they are encountered. If a larger amount of data must be transferred, such as several lines of text surrounding the word (such as for creating web search snippets), then during the map phase the system could calculate how many bytes must be transferred, and try to optimize the placement of the items to be reduced to minimize data transfer to be within racks.

6.15 a. Assume that replication happens within the 10-node cluster. One-way replication means that only one copy of the data exists, the local copy. Two-way replication means two copies exist, and three-way replication means three copies exist; also assume that each copy will be on a separate node. The numbers from Figure 6.1 show the outages for a cluster of 2400 servers. We need to scale the events that happen based on individual servers down to the 10 node cluster. Thus this gives approximately 4 events of hard-drive failures, slow disks, bad memories, misconfigured machines, and flaky machines per year. Similarly we get approximately 20 server crashes per year. Using the calculation in the Example in 6.1, we get Hours Outage = $(4 + 1 + 1 + 1) \times 1$ hour + $(1 + 20) \times 5$ minutes = 8.75 hours. This yields 99.9% availability, assuming one-way replication. In two-way replication, two nodes must be down simultaneously for availability to be lost. The probability that two nodes go down is $(1 - .999) \times (1 - .999) = 1e - 6$. Similarly the probability that three nodes go down is 1e – 9.

b. Repeating the previous problem but this time with 1,000 nodes, we have the following outages: Hours Outage = $(4 + 104 + 104 + 104) \times 1$ hour + $(104 + 2083) \times 5$ minutes = 316 + 182.25 = 498.25 hours. This yields 94.3% availability with one-way replication. In two-way replication, two nodes must be

down simultaneously, which happens with probability $(1 - .943) \times (1 - .943) = 0.003$; in other terms, two-way replication provides 99.7% availability. For three-way replication, three nodes need to be down simultaneously, which happens with probability $(1 - .943) \times (1 - .943) \times (1 - .943) = 0.0002$; in other terms, three-way replication provides 99.9998% availability.

c. For a MapReduce job such as sort, there is no reduction in data from the map phase to the reduce phase as all of the data is being sorted. Therefore there would be 1PB of intermediate data written between the phases. With replication, this would double or triple the I/O and network traffic. Depending on the availability model, this traffic could either be fully contained within a rack (replication only within a rack) or could be across racks (replication across racks). Given the lower disk bandwidth when going across racks, such replication must be done in a way to avoid being on the critical path.

d. For the 1000 node cluster, we have 94.3% availability with no replication. Given that the job takes 1 hour to complete, we need to calculate the additional time required due to restarts. Assuming the job is likely to be restarted at any time due to a failure in availability, on average the job will therefore restart half-way through the task. Thus if there is one failure, the expected runtime = 1 hour + 0.067 × (1 hour/2). However, for a restarted task there is still some probability that it will again have a failure and must restart. Thus the generalized calculation for expected runtime = $1\ hour + \sum_{n=1}^{n=inf} 0.067^n \times \left(\frac{1\ hour}{2}\right)$. This equation approximates to 1.036 hours.

e. The disk takes 10,000 microseconds to access, and the write of 1 KB of data at 200 MB/s will take 5 microseconds. Therefore each log takes 10,005 microseconds to complete. With two and three-way replication, writing to a node in the rack will take 11,010 microseconds, and in the array will take 12,100 microseconds. This leads to 10% and 21% overhead, respectively. If the log was in-memory and only took 10 µs to complete, this leads to 110099900% and 120999900% overhead, assuming the replicas are written to disk. If instead they are written to remote memory, then the time to write the replicas are 100 + 10 = 110 µs and 300 + 100 = 400 µs for rack and array writes, respectively. These times translate to 1000% and 3900% overhead.

f. Assuming the two network trips are within the same rack, and the latency is primarily in accessing DRAM, then there is an additional 200 µs due to consistency. The replications cannot proceed until the transaction is committed, therefore the actions must be serialized. This requirement leads to an additional 310 µs per commit for consistency and replication in-memory on other nodes within the rack.

6.16 a. The "small" server only achieves approximately 3 queries per second before violating the <0.5 second response SLA. Thus to satisfy 30,000 queries per second, 10,000 "small" servers are needed. The baseline server on the other hand achieves 7 queries per second within the SLA time, and therefore needs approximately 4,286 servers to satisfy the incoming queries. If each baseline

server cost $2000, the "small" servers must be $857.20 (42% cheaper) or less to achieve a cost advantage.

b. Figure 6.1 gives us an idea of the availability of a cluster of nodes. With 4,286 servers, we can calculate the availability of all nodes in the cluster. Looked at in another way, if we take 1-Availability, it is the probability that at least one node is down, giving us an estimate of how much overprovisioned the cluster must be to achieve the desired query rate at the stated availability. Extrapolating, we get the Hours Outage = (4 + 447 + 447 + 447) × 1 hour + (447 + 8929) × 5 minutes = 1345 + 781 = 2126 hours. This yields 75.7% availability, or 24.3% probability that a server is down. To calculate 99% cluster availability, or 1% probability that the cluster doesn't have enough capacity to service the requests, we calculate how many simultaneous failures we must support. Thus we need (.243)^n = 0.001, where n is the number of simultaneous failures. Thus a standard cluster needs approximately 5 extra servers to maintain the 30,000 queries per second at 99.9% availability. Comparatively, the cluster with "small" servers will have an availability based on: Hours Outage = (4 + 1041 + 1041 + 1041 × 1.3) × 1 hour + (1041 × 1.3 + 20833) × 5 minutes = 3439 + 1849 = 5288 hours. This yields 39.6% availability. Following the equations from above, we need (.604)^n = 0.001. Thus we need approximately 14 extra servers to maintain the 30,000 queries per second at 99.9% availability. If each baseline server cost $2000, the "small" servers must be $857.00 or less to achieve a cost advantage.

c. In this problem, the small servers provide 30% of the performance of the baseline servers. With 2400 servers, we need 8000 small servers, assuming linear scaling with node size. At 85% scaling, we need 9412 servers, and at 60% scaling, we need 13,333 servers. Although computation throughput will improve as cluster sizes are scaled up, it may become difficult to partition the problem small enough to take advantage of all of the nodes. Additionally, the larger the cluster, there will potentially be more intra-array communication, which is significantly slower than intra-rack communication.

6.17 a. See Figure S.1. In terms of latency, the devices perform more favorably than disk and less favorably than DRAM across both the array and rack. In terms of bandwidth, these devices perform more poorly than both DRAM and disk across the array and rack.

b. Software might be changed to leverage this new storage by storing large amounts of temporary data to Flash or PCIe devices when operating at the local level to benefit from the reduced latency and increased bandwidth compared to disk, and storing permanent data in disk to benefit from the better bandwidth the disk has to offer over the rack and array levels—where data will ultimately be fetched from.

c. (This question refers to a different problem.)

d. Flash memory performs well for random accesses to data, while disks perform well when streaming through sequential data. A cloud service provider

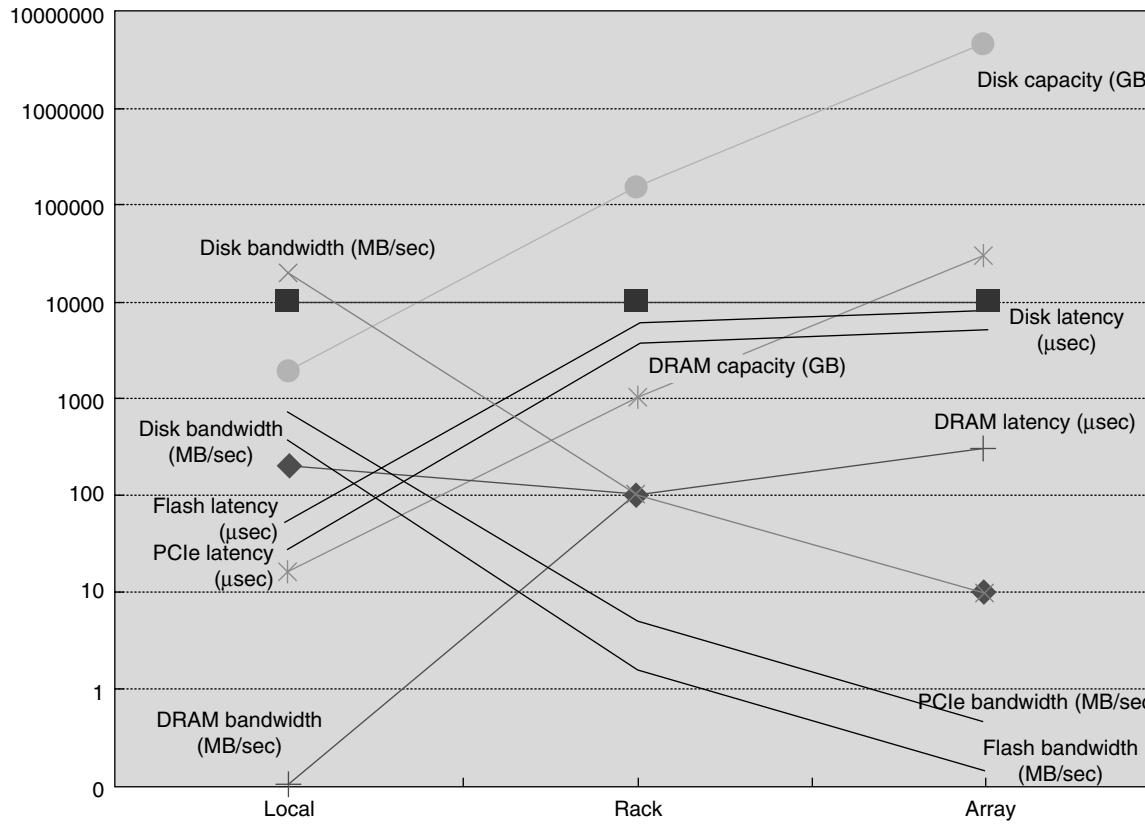


Figure S.1 Graph of latency, bandwidth, and capacity of the memory hierarchy of a WSC.

might look at the different types of workloads being run on his or her data center and offer as an option to his or her customers the option of using Flash memory to improve the performance of workloads which perform many random accesses to data.

6.18 a. For total storage capacity, let's assume the average size of a Netflix movie is 75 minutes (to take into account some 30-minute television shows and 2-hour movies). 75 minutes in seconds is $75\text{ m} \times 60\frac{\text{s}}{\text{m}} = 4500\text{ s}$. Taking into account each of the playback formats, per title, the storage requirements for each of the video formats is $\left(500\frac{\text{Kb}}{\text{s}} \times 4500\text{ s} \times \frac{1\text{MB}}{8 \times 1000\text{ Kb}} = 281.25\text{ MB}\right) + \left(1000\frac{\text{Kb}}{\text{s}} \times 4500\text{ s} \times \frac{1\text{MB}}{8 \times 1000\text{ Kb}} = 562.5\text{ MB}\right) + \left(1600\frac{\text{Kb}}{\text{s}} \times 4500\text{ s} \times \frac{1\text{MB}}{8 \times 1000\text{ Kb}} = 900\text{ MB}\right) + \left(2200\frac{\text{Kb}}{\text{s}} \times 4500\text{ s} \times \frac{1\text{MB}}{8 \times 1000\text{ Kb}} = 1237.5\text{ MB}\right) = 2981.25\text{ MB}$. Given that there were 12,000 titles, the total storage capacity would need to be at least 35,775,000 MB.

We know that there were 100,000 concurrent viewers on the site, but we do not know what playback format each of them was streaming their title in. We assume

a uniform distribution of playback formats and get an average playback bitrate of (500 + 1000 + 1600 + 2200)/4 = 1325 kbps. This makes for an average I/O and network bandwidth of 100,000 × 1325 = 132,500,000 kbps.

b. Per user, the access pattern is streaming, as it is unlikely a user will watch more than one movie concurrently. Temporal and spatial locality will be low, as titles will simply be streamed in from disk (unless the user chooses to replay a part of a title). The exact working set size will depend on the user's desired playback bitrate and the title, but in general, for the files that Netflix hosts, will be large.

Per movie, assuming more than one person is watching different parts of the movie, the access pattern is random, as multiple people will be reading different parts of the movie file at a time. Depending on how closely in time many users are to one another in the movie, the movie data may have good temporal and spatial locality. The working set size depends on the movie and title, but will be large.

Across all movies, assuming many people are watching many different movies, the access pattern is random. There will be little temporal or spatial locality across different movies. The working set size could be very large depending on the number of unique movies being watched.

c. In terms of both performance and TCO, DRAM is the greatest, then SSDs, then hard drives. Using DRAM entirely for storing movies will be extremely costly and may not deliver much improved performance because movie streaming is predominantly limited by network bandwidth which is much lower than DRAM bandwidth. Hard drives require a lower TCO than DRAM, but their bandwidth may be slower than the network bandwidth (especially if many random seeks are being performed as may be the case in the Netflix workload). SSDs would be more expensive, but could provide a better balance between storage and network bandwidth for a Netflix-like workload.

6.19 a. Let's assume that at any given time, the average user is browsing MB of content, and on any given day, the average user uploads MB of content.

b. Under the assumptions in Part a., the amount of DRAM needed to host the working set of data (the amount of data currently being browsed), assuming no overlap in the data being viewed by the users, is 100,000d MB = 100d GB. 96 GB of DRAM per server means that there must be (100d GB)/96 GB ≈ $\lceil 1.04d \rceil$ servers to store the working set of data. Assuming a uniform distribution of user data, every server's memory must be accessed to compose a user's requested page, requiring 1 local memory access and $\lceil 1.04d \rceil$ remote accesses.

c. The Xeon processor may be overprovisioned in terms of CPU throughput for the memcached workload (requiring higher power during operation), but may also be provisioned to allow for large memory throughput, which would benefit the memcached workload. The Atom processor is optimized for low power consumption for workloads with low CPU utilization like memcached, however, the Atom may not be fit for high memory throughput, which would constrain the performance of memcached.

d. One alternative is to connect DRAM or other fast memories such as Flash through other device channels on the motherboard such as PCIe. This can effectively increase the capacity of the system without requiring additional CPU sockets. In terms of performance, such a setup would most likely not be able to offer the same bandwidth as traditionally-connected DRAM. Power consumption depends on the interface being used; some interfaces may be able to achieve similar power consumptions as traditionally-attached DRAM. In terms of cost, such a device may be less expensive as it only involves adding additional memory (which would be purchased in any event) without adding an additional CPU and socket to address that memory. Reliability really depends on the device being used and how it is connected.

e. In case (1), co-locating the memcached and storage servers would provide fast access when data needs to be brought in or removed from memcached, however, it would increase the power density of the server and require more aggressive cooling solutions and would be difficult to scale to larger capacities. Case (2) would require higher latencies than case (1) when bring data in or removing data from memcached, but would be more amenable to scaling the capacity of the system and would also require cooling less power per volume. Case (3) would require very long access latencies to bring data in and remove data from memcached and may be easier to maintain than case (2) due to the homogeneity of the contents of the racks (hard drive and memory failures will mainly occur in one portion of the data center).

6.20 a. We have 1 PB across 48,000 hard drives, or approximately 21 GB of data per disk. Each server has 12 hard disks. Assuming each disk can maintain its maximum throughput, and transfers data entirely in parallel, the initial reading of data at each local node takes 21 GB/200 MB/s = 105 seconds. Similarly writing data back will also take 105 seconds, assuming the data is perfectly balanced among nodes and can be provided at the total aggregate bandwidth of all of the disks per node.

b. With an oversubscription ratio of 4, each NIC only gets 1/4 Gb/sec, or 250 Mb/sec. Assuming all data must be moved, the data is balanced, and all of the movement can be done in parallel, each server must handle 1 PB/4000 = 250 GB of data, meaning it must both send and receive 250 GB of data during the shuffle phase (2×250 GB)/(2×250 Mb/sec) = 8,000 seconds, or 2.22 hours.

c. If data movement over the network during the shuffle phase is the bottleneck, then it takes approximately 6 hours to transfer 2×250 GB of data per server. This translates to each 1 Gb NIC getting approximately 93 Mb/s of bandwidth, or approximately 10:1 oversubscription ratio. However, these numbers assume only the data is being shuffled, no replicas are being sent over the network, and no redundant jobs are being used. All of these other tasks add network traffic without directly improving execution time, which implies the actual oversubscription ratio may be lower.

d. At 10 Gb/s with no oversubscription, we again calculate the transfer time as sending and receiving 250 GB of data. Assuming the server still has two

NICs, we calculate the transfer time as (2 × 250 GB)/(2 × 10 Gb/sec) = 200 seconds, or 0.06 hours.

e. There are many possible points to mention here. The massively scale-out approach allows for the benefits of having many servers (increased total disk bandwidth, computational throughput, any failures affect a smaller percentage of the entire cluster) and using lower-cost components to achieve the total performance goal. However, providing enough network bandwidth to each of the servers becomes challenging, especially when focusing on low-cost commodity components. On the other hand, a small-scale system offers potential benefits in ease of administration, less network distance between all the nodes, and higher network bandwidth. On the other hand, it may cost significantly more to have a small-scale system that provides comparable performance to the scale-out approach.

f. Some example workloads that are not communication heavy include computation-centric workloads such as the SPEC CPU benchmark, workloads that don't transfer large amounts of data such as web search, and single-node analytic workloads such as MATLAB or R. For SPEC CPU, a High-CPU EC2 instance would be beneficial, and for the others a high-memory instance may prove the most effective.

6.21 a. Each access-layer switch has 48 ports, and there are 40 servers per rack. Thus there are 8 ports left over for uplinks, yielding a 40:8 or a 5:1 oversubscription ratio. Halving the oversubscription ratio leads to a monthly Network costs of $560,188, compared to the baseline of $294,943; total costs are $3,796,170 compared to $3,530,926. Doubling the oversubscription ratio leads to monthly Network costs of $162,321, and total costs of $3,398,303.

b. With 120 servers and 5 TB of data, each server handles approximately 42 GB of data. Based on Figure 6.2, there is approximately 6:1 read to intermediate data ratio, and 9.5:1 read to output data ratio. Assuming a balanced system, each system must read 42 GB of data, of which 21 GB is local, 21 GB is remote (16.8 GB in the rack, 4.2 GB in the array). For intermediate data, each system must handle 7 GB of data, of which 4.9 GB is local, 2.1 GB is remote (1.9 GB in the rack, 0.2 GB in the array). Finally for writing, each system must handle 4.4 GB of data, of which 3.1 GB is local, 1.3 GB is remote (1.2 GB in the rack, 0.1 GB in the array). We will make the simplifying assumption that all transfers happen in parallel, therefore the time is only defined by the slowest transfer. For reading, we obtain the following execution times: local = 21 GB/200 MB/s = 105 seconds, rack = 16.8 GB/100 MB/s = 168 seconds, array = 4.2 GB/10 MB/s = 420 seconds. For intermediate data: local = 4.9 GB/200 MB/s = 24.5 seconds, rack = 1.9 GB/100 MB/s = 19 seconds, array = 0.2 GB/10 MB/s = 20 seconds. For writing: local = 3.1 GB/200 MB/s = 15.5 seconds, rack = 1.2 GB/100 MB/s = 12 seconds, array = 0.1 GB/10 MB/s = 10 seconds. The total performance is thus 420 + 24.5 + 15.5 = 460 seconds. If the oversubscription ratio is cut by half, and bandwidth to the rack and array double, the read data time will be cut in half from 420 seconds to 210 seconds, resulting in total execution time of 250 seconds. If the oversubscription is doubled, the execution time is 840 + 40 + 24 = 904 seconds.

Network costs can be calculated by plugging the proper numbers of switches into the spreadsheet.

c. The increase in more cores per system will help reduce the need to massively scale out to many servers, reducing the overall pressure on the network. Meanwhile, optical communication can potentially substantially improve the network in both performance and energy efficiency, making it feasible to have larger clusters with high bandwidth. These trends will help allow future data centers to be more efficient in processing large amounts of data.

6.22 a. At the time of print, the "Standard Extra Large", "High-Memory Extra Large", "High-Memory Double Extra Large", and "High-Memory Quadruple Extra Large", and "Cluster Quadruple Extra Large" EC2 instances would match or exceed the current server configuration.

b. The most cost-efficient solution at the time of print is the "High-Memory Extra Large" EC2 instance at $0.50/hour. Assuming the number of hours per month is $30 \times 24 = 720$, the cost of hosting the web site on EC2 would be 720 hours × $0.50/hour = $360.

c. Data transfer IN to EC2 does not incur a cost, and we conservatively assume that all 100 GB/day of data is transferred OUT of EC2 at a rate of $0.120/GB. This comes to a monthly cost of 30 days/month × 100 GB × $0.120/GB = $48/month . The cost of an elastic IP address is $0.01/hour, or $7.20/month. The monthly cost of the site is now $360 + $48 + $7.20 = $415.20.

d. The answer to this question really depends on your department's web traffic. Modestly-sized web sites may be hosted in this manner for free.

e. Because new arrivals may happen intermittently, a dedicated EC2 instance may not be needed and the cheaper spot instances can be utilized for video encoding. In addition, the Elastic Block Store could allow for gracefully scaling the amount of data present in the system.

6.23 a. There are many possible options to reduce search query latency. One option is to cache results, attempting to keep the results for the most frequently searched for terms in the cache to minimize access times. This cache could further improve latency by being located either in main memory or similar fast memory technology (such as Flash). Query results can be improved by removing bottlenecks as much as possible. For example, disk is likely to be a bottleneck if the search must access anything from disk due to its high access latency. Therefore one option is to avoid accessing disk as much as possible by storing contents either in main memory or non-volatile solid state storage. Depending on how many queries each search server is receiving, the network could potentially be a bottleneck. Although the data transferred per search query is likely small (only a page of text and a few images is returned to the user), the number of concurrent connections may be quite high. Therefore a separate core to help network processing or a TCP offload engine may also provide benefits in reducing query latency.

b. Monitoring statistics would need to be at multiple levels. To understand where time is spent in low-level code in a single server, a tool such as *Oprofile* would

help provide a picture of what code is taking the most time. At a higher level, a tool such as *sar* or *perf* would help identify the time the system is spending in different states (user versus kernel) and on different devices (CPU, memory, disk, network utilization). All of this information is critical to identifying which resources are the bottlenecks. At an even higher level, a network monitoring tool at the rack switch level can help provide a picture of network traffic and how servers are sending and receiving packets. All of this information must be combined at a cluster level to understand the bottlenecks for each search. This high level tool would need to be implemented through a cluster-level profiler that can collect the data for each individual node and switch, and aggregate them into a single report that can be analyzed.

c. We want to achieve an SLA of 0.1 sec for 95% of the queries. Using the normal distribution to create a cumulative density function, we find that 95% of queries will require 7 disk accesses. If disk is the only bottleneck, we must have 7 accesses in 0.1 seconds, or 0.014 seconds per access. Thus the disk latency must be 14 ms.

d. With in-memory results caching, and a hit rate of 50%, we must only worry about misses to achieve the SLA. Therefore 45% of the misses must satisfy the SLA of <0.1 seconds. Given that 45 out of the remaining 50% of accesses must satisfy the SLA, we look at the normal distribution CDF to find the 45%/50% = 90^{th} percentile. Solving for that, we find the 90th percentile requires approximately 6 disk accesses, yielding a required disk latency of 0.016 seconds, or 16 ms.

e. Cached content can become stale or inconsistent upon change to the state of the content. In the case of web search, this inconsistency would occur when the web pages are re-crawled and re-indexed, leading to the cached results becoming stale with respect to the latest results. The frequency depends on the web service; if the web is re-crawled every evening, then the cache would become inconsistent every night. Such content can be detected by querying the caches upon writing new results; if the caches have the data in their cache, then it must be invalidated at that time. Alternatively, the servers storing the persistent results could have a reverse mapping to any caches that may contain the content. If the caches are expected to frequently have their contents changed, this scheme may not be efficient due to the frequent updates at the persistent storage nodes.

6.24 a. Let's assume that most CPUs on servers operate in the range of 10–50% utilization as in Figure 6.3, and that average server utilization is the average of these values, 30%. If server power is perfectly proportional to CPU utilization, the average PSU efficiency will be 75%.

b. Now, the same power will be supplied by two PSUs, meaning that for a given load, half of the power will be supplied by one PSU and the other half will be supplied by the other PSU. Thus, in our example, the 30% power load from the system will be divided between the two PSU, 15% on each. This causes the efficiency of the PSUs to each be 65%, now.

c. Let's assume the same average server utilization of 30% from Part a. In this case, each PSU will handle the load of $\frac{16}{6} = 2\frac{2}{3}$ servers for a PSU utilization of $30\% \times 2\frac{2}{3} = 80\%$. At this load, the efficiency of each PSU is at least 80%.

6.25 a. In Figure 6.25, we can compute stranded power by examining the normalized power of the CDF curve for a particular group of machines when the CDF is equal to 1. This is the normalized power when all of the machines are considered. As we can see from Figure 6.25(b), at the cluster level, almost 30% of the power is stranded; at the PDU level, slightly more than 80% of the power is stranded; and at the rack level, around 5% of the power is stranded. At larger groups of machines, larger oversubscription may be a more viable option because of the lack of synchronization in the peak utilization across a large number of machines.

b. Oversubscription could cause the differences in power stranding at different groups of machines. Larger groups of machines may provide more opportunities for oversubscription due to their potential lack of synchronization of peak utilizations. Put another way, for certain workloads, it is unlikely that all machines will simultaneously require their peak rated power consumption at the cluster level, but at the per-server (PDU) level, maximum utilization is a likely occurrence.

c. From Figure 6.14, referenced in the case study, the total monthly cost of a data-center provisioned for peak capacity (100% utilization) is $3,800,000. If, however, we provision the data-center for actual use (78% utilization), we would only require $N' = 0.78N$ machines, where N is the original number of servers in the data-center and the new utilization of each machine is 100% (note that we could instead choose to use any number of machines between $0.78N$ and N, but in this case we choose the minimum number of machines because server cost, CAPEX, dominates total cost).

With fewer machines, most costs in Figure 6.14 are reduced, but power use increases due to the increased utilization of the system. The average power utilization of the data-center increases by 100% – 80% = 20%. To estimate the effect on TCO, we conservatively scale only the cost of server hardware and power and cooling infrastructure by 0.78 and the cost of power use by 1.20 to get a new monthly cost of operation of ($2,000,000 × 0.78) + $290,000 + ($765,000 × 0.78) + $170,000 + ($475,000 × 1.20) + $85,000 = $3,271,700, for a savings of $528,300 per month.

d. Let's assume there are X servers per PDU, Y servers per rack, and Z servers per array. At the PDU level, we can afford to oversubscribe by around 30%, for a total of $\lfloor 1.3X \rfloor - X$ additional servers. Similarly, at the rack and cluster level, we can afford to oversubscribe by around $\lfloor 1.2Y \rfloor - Y$ and $\lfloor 1.05Z \rfloor - Z$ servers, respectively. The total number of additional servers we can employ is given by $(\lfloor 1.3X \rfloor - X) + (\lfloor 1.2Y \rfloor - Y) + (\lfloor 1.2Z \rfloor - Z)$.

e. In order to make the optimization in Part d. work, we would need some way of throttling the utilization of the system. This may require additional support from the software or hardware and could result in increased service times, or, in the worst case, job starvation whereby a user's job would not be serviced at all.

f. Preemptive policies may lead to inefficient utilization of system resources because all resources are throttled by default; such policies, however, may provide better worst-case guarantees by making sure power budgets are not violated. This technique may be most beneficial in settings where a controlled power and performance environment is required.

   Conversely, reactive policies may provide more efficient utilization of system resources by not throttling resource utilization unless violations occur. This, however, can lead to problems when contention for resources causes power budgets to be exceeded, resulting in unexpected (from an application's perspective) throttling. If the resource utilization of a workload is well understood, and this technique can safely be applied, it could beneficially provide applications with more resources than a preemptive policy.

g. Stranded power will increase as systems become more energy proportional. This is because at a given utilization, system power will be less for an energy proportional system than a non-energy proportional system, such as that in Figure 6.4. Lower system power means that the same number of system at a given utilization will require even less power than what is provisioned, increasing the amount of stranded power and allowing even more aggressive oversubscribing to be employed.

6.26 a. First let's calculate the efficiency prior to the UPS. We obtain 99.7% × 98% × 98% × 99% = 94.79% efficient. A facility-wide UPS is 92% efficient, yielding a total efficiency of 94.79% × 92% = 87.21% efficiency. A battery being 99.99% efficient yields 94.79% × 99.99% = 94.78% efficiency, or 7.57% more efficient.

b. Assume that the efficiency of the UPS is reflected in the PUE calculation. That is, the case study's base PUE of 1.45 is with a 92% efficient facility-wide UPS and 87.21% efficiency. The batteries are 7.57% more efficient. All power must go through the transformations, thus the PUE is directly correlated with this efficiency. Therefore the new PUE is 1.34. This PUE reduces the size of the total load to 10.72MW; the same number of servers is needed, so the critical load remains the same. Thus the monthly power costs are reduced from $474,208 to $438,234. The original total monthly costs were $3,530,926.

   Let us assume the facility-wide UPS is 10% of the total cost of the facility that is power and cooling infrastructure. Thus the total facility cost is approximately $66.1M without the UPS. We now need to calculate the new total cost of the facility (assuming a battery has the same depreciation as the facility) by adding the cost of the per-server battery to the base cost, and amortizing the new facility costs. By doing so, we find the batteries can cost ~14% of the per-server cost and break even with the facility-wide UPS. If we assume a depreciation time the same as the server (and add the battery directly into the per-server cost), we find the batteries can cost ~5% of the per-server cost and break even with the facility-wide UPS.

c. The per-server UPS likely has lower costs, given the cost of batteries, and reduces the total power needed by the entire facility. Additionally it removes a single point of failure by instead distributing the UPS. However, if the batteries cannot provide the other functionality that a UPS provides (such as power conditioning), there will still be a need for additional facility equipment. The management model likely becomes more complex, as the administrators must manage as many UPS's as there are servers, as opposed to a single facility-wide UPS (or two if redundancy is provided). In particular, they need to ensure that each of those UPS properly work, and message the server that the power has been interrupted in case special action must be taken (such as checkpointing state and shutting down servers).

6.27 a. Total operational power = (1 + Cooling inefficiency multiplier) × IT equipment power. For this problem, there is an 8 MW datacenter, with 80% power usage, electricity costs of $0.10 per kWH, and cooling-inefficiency of 0.8. Thus the baseline costs are Total operational power × electricity costs = (1 + 0.8) × (8 MW × 80%) × $0.10 per kWH = $1152 per hour. If we improve cooling efficiency by 20%, this yields a cooling-inefficiency of 0.8 × (1 – 20%) = 0.64, and a cost of $1049.60 per hour. If we improve energy efficiency by 20%, this yields a power usage of 80% × (1 – 20%) = 0.64, and a cost of $921.60 per hour. Thus improving the energy efficiency is a better choice.

b. To solve for this question, we use the equation (1 + 0.8) × (8 MW × 80% × (1 – x)) × $0.10 per kWH = $1049.60, where x is the percentage improvement in IT efficiency to break even 20% cooling efficiency improvement. Solving, we find x = 8.88%.

c. Overall, improving server energy efficiency will more directly lower costs rather than cooling efficiency. However, both are important to the total cost of the facility. Additionally, the datacenter operator may more directly have the ability to improve cooling efficiency, as opposed to server energy efficiency, as they control the design of the building (e.g., can use air cooling, run at increased temperatures, etc.), but do not directly control the design of the server components (while the operator can choose low power parts, they are at the mercy of the companies providing the low power parts).

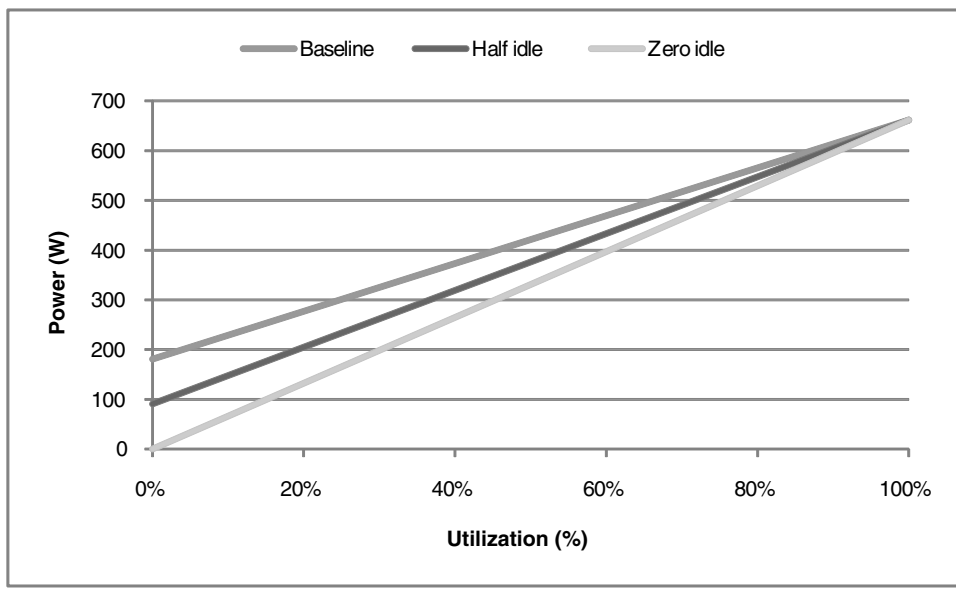
6.28 a. The COP is the ratio of the heat removed (Q) to the work needed to remove the heat (W). Air returns the CRAC unit at 20 deg. C, and we remove 10 KW of heat with a COP of 1.9. COP = Q/W, thus 1.9 = 10 KW/W. W = 5.26 KJ. In the second example, we are still removing 10 KW of heat (but have a higher return temperature by allowing the datacenter to run hotter). This situation yields a COP of 3.1, thus 3.1 = 10 KW/W. W = 3.23 KJ. Thus we save ~2 KJ by allowing the air to be hotter.

b. The second scenario is 3.23 KJ/5.26 KJ = 61.4% more efficient, providing 39.6% savings.

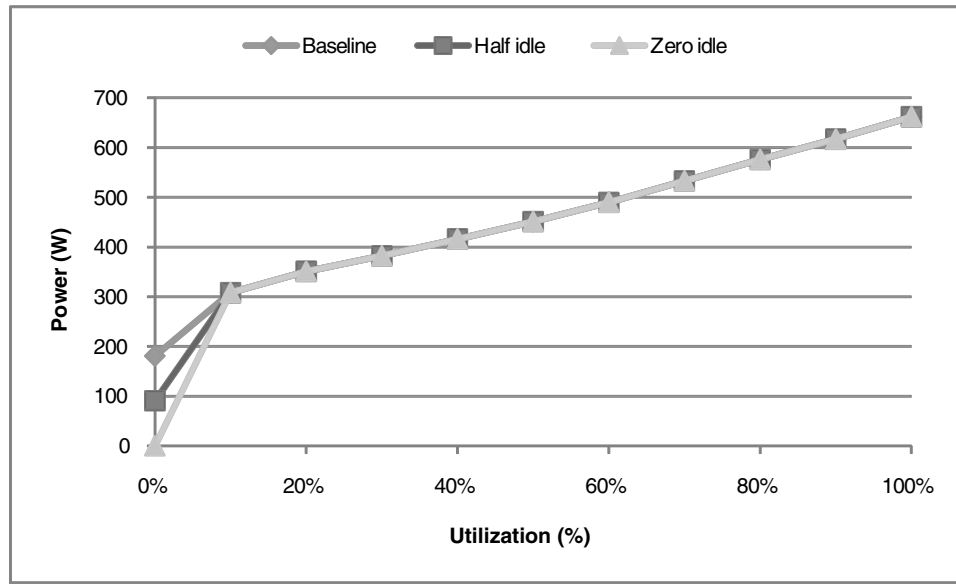
c. When multiple workloads are consolidated, there would be less opportunity for that server to take advantage of ACPI states to reduce power, as the server requires higher total performance. Depending on the most optimal point to

run all servers (e.g., if running all servers at 70% utilization is better than half of the servers at 90% and half at 50%), the combination of the two algorithms may result in suboptimal power savings. Furthermore, if changing ACPI states changes the server's reporting of utilization (e.g., running in a lower-power state results in 80% utilization rather than 60% utilization at a higher state) then opportunities to consolidate workloads and run at better efficiency points will be missed. Thus using the two algorithms together may result in information being obscured between the two algorithms, and therefore missed opportunities for better power saving. Other potential issues are with algorithms that may have conflicting goals. For example, one algorithm may try to complete jobs as quickly as possible to reduce energy consumption and put the server into a low-power sleep mode (this "race-to-sleep" algorithm is frequently used in cell phones). Another algorithm may try to use low-power ACPI modes to reduce energy consumption. These goals conflict with each other (one tries to use maximum throughput for as brief as possible, the other tries to use minimum acceptable throughput for as long as possible), and can lead to unknown results. One potential way to address this problem is to introduce a centralized control plane that manages power within servers and across the entire datacenter. This centralized control plane could use the inputs that are provided to the algorithms (such as server utilization, server state, server temperature, datacenter temperature) and decide the appropriate action to take, given the actions available (e.g., use ACPI modes, consolidate workloads, migrate workloads, etc.).

6.29 a.



b.

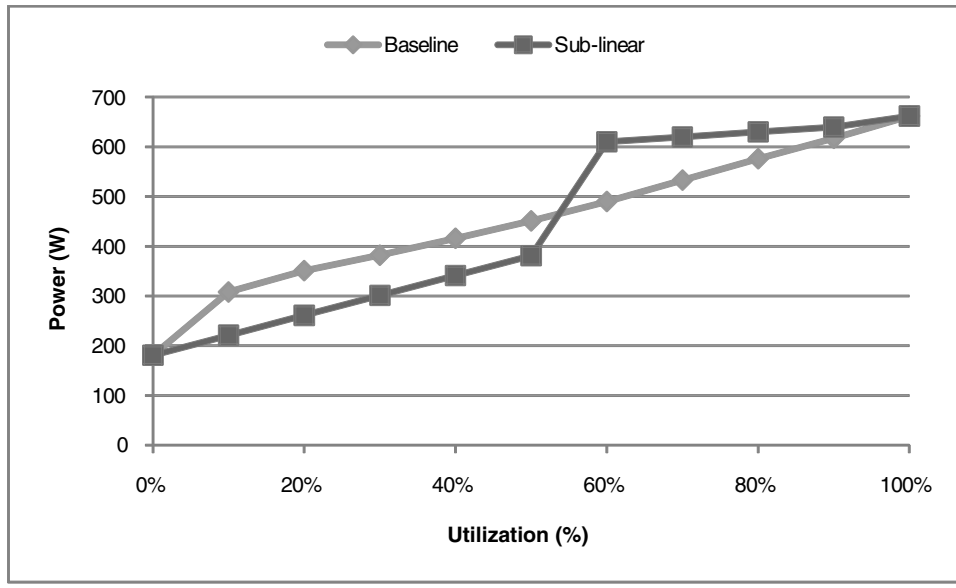


We can see that although reducing the idle power can help when the server is not utilized, there is still a strong non-linearity between 0% and 100% utilization, meaning that operating at other utilization points is likely less efficient than it can potentially be.

c. Using the numbers from column 7, the performance from column 2, and the power from the previous problems, and assuming we run the workload constantly for 1 hour, we get the following table.

	Power					Energy (WH)			Performance
	Servers	Performance	Baseline	Half idle	Zero idle	Baseline	Half idle	Zero idle	Baseline
0%	109	0	181	90.5	0	19729	9864.5	0	0
10%	80	290762	308	308	308	24640	24640	24640	23260960
20%	153	581126	351	351	351	53703	53703	53703	88912278
30%	246	869077	382	382	382	93972	93972	93972	213792942
40%	191	1159760	416	416	416	79456	79456	79456	221514160
50%	115	1448810	451	451	451	51865	51865	51865	166613150
60%	51	1740980	490	490	490	24990	24990	24990	88789980
70%	21	2031260	533	533	533	11193	11193	11193	42656460
80%	15	2319900	576	576	576	8640	8640	8640	34798500
90%	12	2611130	617	617	617	7404	7404	7404	31333560
100%	8	2889020	662	662	662	5296	5296	5296	23112160
Total:	1001					380888	371023.5	361159	934784150

d. One potential such non-linear curve is shown here:



This curve yields the following table:

	Power		Energy (WH)				Performance
	Servers	Performance	Baseline	Sub-linear	Baseline	Sub-linear	Baseline
0%	109	0	181	181	19729	19729	0
10%	80	290762	308	221	24640	17680	23260960
20%	153	581126	351	261	53703	39933	88912278
30%	246	869077	382	301	93972	74046	214000000
40%	191	1159760	416	341	79456	65131	222000000
50%	115	1448810	451	381	51865	43815	167000000
60%	51	1740980	490	610	24990	31110	88789980
70%	21	2031260	533	620	11193	13020	42656460
80%	15	2319900	576	630	8640	9450	34798500
90%	12	2611130	617	640	7404	7680	31333560
100%	8	2889020	662	662	5296	5296	23112160
Total:	1001				380888	326890	935000000

6.30 a. Using figure 6.26, we get the following table:

	Power		Energy (WH)		Performance		
	Servers	Performance	Case A	Case B	Case A	Case B	Baseline
0%	109	0	181	250	19729	27250	0
10%	80	290762	308	275	24640	22000	23260960
20%	153	581126	351	325	53703	49725	88912278
30%	246	869077	382	340	93972	83640	214000000
40%	191	1159760	416	395	79456	75445	222000000
50%	115	1448810	451	405	51865	46575	167000000
60%	51	1740980	490	415	24990	21165	88789980
70%	21	2031260	533	425	11193	8925	42656460
80%	15	2319900	576	440	8640	6600	34798500
90%	12	2611130	617	445	7404	5340	31333560
100%	8	2889020	662	450	5296	3600	23112160
Total:	1001				380888	350265	935000000

b. Using these assumptions yields the following table:

	Power				Energy (WH)				Performance
	Servers	Performance	Case A	Case B	Linear	Case A	Case B	Linear	Baseline
0%	504	0	181	250	0	91224	126000	0	0
10%	6	290762	308	275	66.2	1848	1650	397.2	1744572
20%	8	581126	351	325	132.4	2808	2600	1059.2	4649008
30%	11	869077	382	340	198.6	4202	3740	2184.6	9559847
40%	26	1159760	416	395	264.8	10816	10270	6884.8	30153760
50%	57	1448810	451	405	331	25707	23085	18867	82582170
60%	95	1740980	490	415	397.2	46550	39425	37734	165393100
70%	123	2031260	533	425	463.4	65559	52275	56998.2	249844980
80%	76	2319900	576	440	529.6	43776	33440	40249.6	176312400
90%	40	2611130	617	445	595.8	24680	17800	23832	104445200
100%	54	2889020	662	450	662	35748	24300	35748	156007080
Total:	1000					352918	334585	223954.6	980692117

We can see that this consolidated workload is able to reduce the total energy consumption for case A, and provide slightly higher performance. We can see that the linear energy-proportional model provides the best results, yielding the least amount of energy. Having 0 idle power and linear energy-proportionality provides significant savings.

c. The data is shown in the table in b. We can see that the consolidation also provides savings for Case B, but slightly less savings than for Case A. This is due to the lower overall power of the Case B server. Again the linear energy-proportional model provides the best results, which is notable given its higher peak power usage. In this comparison, the 0 idle power saves a significant amount of energy versus the Case B servers.

6.31 a. With the breakdowns provided in the problem and the dynamic ranges, we get the following effective per-component dynamic ranges for the two cases:

	Effective dynamic range	
CPU	1.5	0.99
Memory	0.46	0.6
Disks	0.143	0.13
Networking/other	0.192	0.324
Total	2.295	2.044

b. From the table we can see the dynamic ranges are 2.295x and 2.044x for server 1 and server 2, respectively.

c. From a., we can see that the choice of components will affect the overall dynamic range significantly. Because memory and networking/other makes up a much larger portion of the second system its dynamic range is ~10% less than the first server, indicating worse proportionality. To effectively address server energy proportionality we must improve the dynamic range of multiple components; given memory's large contribution to power, it is one of the top candidates for improving dynamic range (potentially through use of different power modes).

6.32 **Note:** This problem cannot be done with the information given.

6.33 We can see from Figure 6.12 that users will tolerate some small amount of latency (up to ~4X more than the baseline 50 ms response time) without a loss in revenue. They will tolerate some additional amount of latency (10X slower than the baseline 50 ms response time) with a small hit to revenue (1.2%). These results indicate that in some cases there may be flexibility to trade off response time for actions that may save money (for example, putting servers into low-power low-performance modes). However, overall it is extremely important to keep a high level of service, and although user satisfaction may only show small changes, lower service performance may be detrimental enough to deter users to

other providers. Avoiding downtime is even more important, as a datacenter going down may result in significant performance hits on other datacenters (assuming georeplicated sites) or even service unavailability. Thus avoiding downtime is likely to outweigh the costs for maintaining uptime.

6.34 a. The following table shows the SPUE given different fan speeds. SPUE is the ratio of total power drawn by the system to power used for useful work. Thus a baseline SPUE of 1.0 is when the server is drawing 350 W (its baseline amount).

Fan speed	Fan power	SPUE	TPUE, baseline PUE	TPUE, Improved PUE
0	0	1	1.7	1.581
10000	25.54869684	1.072996	1.824094	1.696407
12500	58.9420439	1.168406	1.98629	1.84725
18000	209	1.597143	2.715143	2.525083

b. We see an increase from 1.073 to 1.168 for increasing from 10,000 rpm to 12,500 rpm, or approximately a 8.9% increase. Going from 10,000 rpm to 18,000 rpm results in a 48.8% increase in SPUE. We can see that the TPUE of the improved case at 18,000 rpm is significantly higher than the TPUE of the baseline case with the fans at 10,000 or 12,5000 rpm. Even if the fans used 12,500 rpm for the higher temperature case, the TPUE would still be higher than the baseline with the fans at 10,000 rpm.

c. The question should be, "Can you identify another design where changes to TPUE are lower than the changes to PUE?" For example, in exercise 6.26, one level of inefficiency (losses at the UPS) is moved to the IT equipment by instead using batteries at the server level. Looking at PUE alone, this design appears to significantly approve PUE by increasing the amount of power used at the IT equipment (thereby decreasing the ratio of power used at the facility to power used at the IT equipment). In actuality, this design does not enable any more work to be accomplished, yet PUE does not reflect this fact. TPUE, on the other hand, will indicate the decreased efficiency at the IT level, and thus TPUE will change less than PUE. Similarly, designs that change power used at the facility level to power used at the IT level will result in greater changes to PUE than TPUE. One other example is a design that utilizes very high power fans at the server level, and thereby reduces the speed of the fans in the facility.

6.35 a. The two benchmarks are similar in that they try to measure the efficiency of systems. However, their focuses differ. SPEC power is focused on the performance of server side Java (driven by CPU and memory), and the power used by systems to achieve that performance. JouleSort on the other hand is focused on balanced total system performance (including CPU, memory,

and I/O) and the energy required to sort a fixed input size. Optimizing for SPEC power would focus on providing the peak performance at the lowest power, and optimizing only for the CPU and memory. On the other hand, optimizing for JouleSort would require making sure the system is energy efficient at all phases, and needs I/O performance that is balanced with the CPU and memory performance.

To improve WSC energy efficiency, these benchmarks need to factor in the larger scale of WSCs. They should factor in whole cluster level performance of up to several thousand machines, including networking between multiple systems potentially across racks or arrays. Energy should also be measured in a well defined manner to capture the entire energy used by the datacenter (in a similar manner to PUE). Additionally, the benchmarks should also take into account the often interactive nature of WSCs, and factor in client-driven requests which lead to server activity.

b. Joulesort shows that high performing systems often come with a non-linear increase in power. Although they provide the highest raw throughput, the total energy is higher for the same amount of work done versus lower power systems such as a laptop or embedded class computer. It also shows that efficiencies across the total system are important to reduce energy; lower power DRAM and solid state drives instead of hard disks, both reduce total system energy for the benchmark.

c. I would try to optimize the use of the memory hierarchy by the benchmark to improve the energy efficiency. More effective use of the processor's cache will help avoid lengthy stalls from accessing DRAM. Similarly, more effective use of DRAM will reduce stalls from going to disk, as well as disk access power. I would also consider having the application take more advantage of low power modes and aggressively putting the system into the optimal power state given expected delays (such as switching the CPU to low power modes when disk is being accessed).

6.36 a. First calculate the hours of outage due to the various events listed in Figure 6.1. We obtain: Hours outage = $(4 + 250 + 250 + 250) \times 1$ hour + $(250) \times 5$ minutes = 754 hours + 20.8 hours = 774.8 hours. With 8760 hours per year, we get an availability of $(8760 - 774.8)/8760 = 91.2\%$ availability. For 95% availability, we need the cluster to be up 8322 hours, or 438 hours of downtime. Focusing on the failures, this means reducing from 750 hours of hardware-related failures to 438 hours, or 58.4% as many failures. Thus we need to reduce from 250 failures for hard drives, memories, and flaky machines (each) to 146 failures.

b. If we add the server failures back in, we have a total of 754 hours + 438 hours of unavailability. If none of the failures are handled through redundancy, it will essentially be impossible to reach 95% availability (eliminating each of the hardware-related events completely would result in 94.95% availability). If 20% of the failures are handled, we have a total of 754 + $(250 + 5000 \times (1 - 20\%)) \times$ minutes = 754 + 70.8 hours of unavailability. The other hardware events need to be reduced to 121 events each to achieve

95% availability. With 50% of failures handled the other events need to be reduced to 129 events each.

c. At the scale of warehouse-scale computers, software redundancy is essential to provide the needed level of availability. As seen above, achieving even 95% availability with (impossibly) ultra-reliable hardware is essentially impossible due failures not related to hardware. Furthermore, achieving that level of reliability in hardware (no crashes ever due to failed components) results in extremely expensive systems, suitable for single use cases and not large scale use cases (for example, see HP's NonStop servers). Thus software must be used to provide the additional level of redundancy and handle failures.

However, having redundancy at the software level does not entirely remove the need for reliable systems. Having slightly less reliable servers results in more frequent restarts, causing greater pressure on the software to maintain a reliable entire system. If crashes are expected to be frequent, certain software techniques may be required, such as frequent checkpointing, many redundant computations, or replicated clusters. Each technique has its own overheads associated with it, costing the entire system performance and thereby increasing operating costs. Thus a balance must be struck between the price to achieve certain levels of reliability and the software infrastructure needed to accommodate the hardware.

6.37 a. There is a significant price difference in unregistered (non-ECC) and registered (ECC supporting) DDR3 memories. At the sweet spot of DDR3 (approximately 4 GB per DIMM), the cost difference is approximately 2X higher for registered memory.

Based on the data listed in section 6.8, a third of the servers experience DRAM errors per year, with an average of 22,000 correctable errors and 1 uncorrectable error. For ease of calculation, we will assume that the DRAM costs are for chipkill supporting memory, and errors only cause 5 minutes of downtime due to a reboot (in reality, some DRAM errors may indicate bad DIMMs that need to be replaced). With ECC, and no other failures, 1/3 of the servers are unavailable for a total of 5 minutes per year. In a cluster of 2400 servers, this results in a total of 66.67 hours of unavailability due to memory errors. The entire cluster has a total of 21,024,000 hours of runtime per year, yielding 99.9997% availability due to memory errors. With no error correction, 1/3 of the servers are unavailable for $22{,}000 \times 5$ minutes per year. In a cluster of 2400 servers, this results in a total of 1,466,666 hours of downtime per year. This yields 93.02% availability due to memory errors. In terms of uptime per dollar, assuming a baseline DDR3 DIMM costs $50, and each server has 6 DIMMs, we obtain an uptime per dollar of 14.60 hrs/$ for the ECC DRAM, and 27.16 hrs/$ for the non-ECC DRAM. Although the non-ECC DRAM has definite advantages in terms of uptime per dollar, as described in section 6.8 it is very desirable to have ECC RAM for detecting errors.

6.38 a. Given the assumed numbers ($2000 per server, 5% failure rate, 1 hour of service time, replacement parts cost 10% of the server, $100/hr technician time), we obtain a $300 cost for fixing each server. With a 5% failure rate, we expect a $15 annual maintenance cost per server.

b. In the case of WSC, server repairs can be batched together. Repairing in such a manor can help reduce the technician visit costs. With software redundancy to overcome server failures, nodes can be kept offline temporarily until the best opportunity is available for repairing them with little impact to the overall service (minor performance loss). On the other hand, for traditional enterprise datacenter, each of those applications may have a strong dependency on individual servers being available for the service to run. In those cases, it may be unacceptable to wait to do repairs, and emergency calls to repair technicians may incur extra costs.

Additionally, the WSC model (where thousands of computers are running the same image) allows the administrator to more effectively manage the homogenous cluster, often enabling one administrator to handle up to 1000 machines. In traditional enterprise datacenters, only a handful of servers may share the same image, and even in those cases, the applications must often be managed on a per-server basis. This model results in a high burden on the administrator, and limits the number of servers they can manage to well under 100 per admin.