Question #1

PP523-524 6.15 (b, c)

WSC programmers often use data replication to overcome failures in the software. Hadoop HDFS , for example, employs three-way replication (one local copy, one remote copy in the rack, and one remote copy in a separate rack), but it's worth examining when such replication is needed.

Approx. number events in 1st year	Cause	Consequence
1 or 2	Power utility failures	Lose power to whole WSC; doesn't bring down WSC if UPS and generators work (generators work about 99% of time).
4	Cluster upgrades	Planned outage to upgrade infrastructure, many times for evolving networking needs such as recabling, to switch firmware upgrades, and so on. There are about nine planned cluster outages for every unplanned outage.
1000s	Hard-drive failures	2%–10% annual disk failure rate (Pinheiro et al., 2007)
	Slow disks	Still operate, but run 10× to 20× more slowly
	Bad memories	One uncorrectable DRAM error per year (Schroeder et al., 2009)
	Misconfigured machines	Configuration led to ~30% of service disruptions (Barroso and HÖlzle, 2009)
	Flaky machines	1% of servers reboot more than once a week (Barroso and HÖlzle, 2009)
5000	Individual server crashes	Machine reboot; typically takes about 5 min (caused by problems in software or hardware).

Figure 6.1 List of outages and anomalies with the approximate frequencies of occurrences in the first year of a new cluster of 2400 servers. We label what Google calls a cluster an *array*; see Figure 6.5. Based on Barroso, L.A., 2010. Warehouse Scale Computing [keynote address]. In: Proceedings of ACM SIGMOD, June 8–10, 2010, Indianapolis, IN.

Figure 6.1

b. Assuming the failure data in Figure 6.1 and a 1000-node Hadoop cluster, what kind of availability does it have with one-, two-, and three-way replications? What can you reason about the benefits of replication, at scale?

Considering One-way replication, we know that if a single node fails, then data is lost. So considering that there is 5000 Individual Server Crashes across the 2400 servers, then our 1000-node Hadoop cluster will experience 2083 crashes per year (assuming the same ratio), furthermore, the hard-drive failures alone would affect 2 % - 10 % of the nodes, which comes out to roughly 20 – 100 nodes per year in the 1000-node cluster, making this setup really just unacceptable for availability.

With Two-way replication, if a node fails then the second copy still exists, but if both fail simultaneously then data is now lost. Again, referencing the 5000 crashes per 2400 servers means we can tell that there are roughly 2.1 crashes per server per year, so if two of our randomly chosen nodes store the data, failure probability is reduced but not eliminated, meaning that this is still more resilient overall but still risky in larger-scale operations.

For Three-way replication, it is much more available because even if two copies fail, a third will exist. And the probability of all three failing at the same time is exponentially lower, this helps protect against rack-level failures and is often used in production environments for its data durability. There is a trade-off however, with higher storage costs and I/O traffic.

In summary for the benefits of the different types of replications :

One-way replication : Not reliable due to node crashes and potential disk failures.

Two-way replication : Better than One-way but still risky.

Three-way replication : Highly reliable, as it protects against cluster failures, but still increases storage and network overhead.

c. The relative overhead of replication varies with the amount of data written per local compute hour. Calculate the amount of extra I/O traffic and network traffic (within and across rack) for a 1000-node Hadoop job that sorts 1 PB of data, where the intermediate results for data shuffling are written to the HDFS.

Breaking down the data we have :

$$Total\ Data\ To\ Sort = 1\ PB = 1000\ TB$$

$$HDFS\ Replication\ Factor = 3$$

$$Total\ HDFS\ Storage\ Needed = 3 * 1000\ TB = 3000\ TB$$

So, considering the I/O Traffic, there will need to be :

$$Total\ IO\ Writes = 3 * Input\ Data = 3\ PB$$

With Network Traffic we know that :

Within-rack traffic, only one copy is stored in the local rack.

Across-rack traffic, one copy must be sent to a different rack.

So, from this we know that for each 1 PB of input data the network transfers are :

$$Within\ Rack\ Traffic = 1\ PB$$

$$Across\ Rack\ Traffic = 1\ PB$$

Thus, total network traffic :

$$Within\ Rack\ Traffic + Across\ Rack\ Traffic = Total\ Network\ Traffic$$

$$1\ PB + 1\ PB = 2\ PB = Total\ Network\ Traffic$$

Question #2

PP525 6.17 (a, b)

One trend in high-end servers is toward the inclusion of nonvolatile flash memory in the memory hierarchy, either through solid-state disks (SSDs) or PCI Express-attached cards. Typical SSDs have a bandwidth of 250 MB/s and latency of 75 µs, whereas PCIe cards have a bandwidth of 600 MB/s and latency of 35 µs.

WSC Memory Hierarchy

Figure 6.6 shows the latency, bandwidth, and capacity of memory hierarchy inside a WSC, and Figure 6.7 shows the same data visually. These figures are based on the following assumptions (Barroso et al., 2013):

	Local	Rack	Array
DRAM latency (µs)	0.1	300	500
Flash latency (µs)	100	400	600
Disk latency (µs)	10,000	11,000	12,000
DRAM bandwidth (MB/s)	20,000	100	10
Flash bandwidth (MB/s)	1000	100	10
Disk bandwidth (MB/s)	200	100	10
DRAM capacity (GB)	16	1024	31,200
Flash capacity (GB)	128	20,000	600,000
Disk capacity (GB)	2000	160,000	4,800,000

Figure 6.6 Latency, bandwidth, and capacity of the memory hierarchy of a WSC (Barroso et al., 2013). Figure 6.7 plots this same information.

Figure 6.6

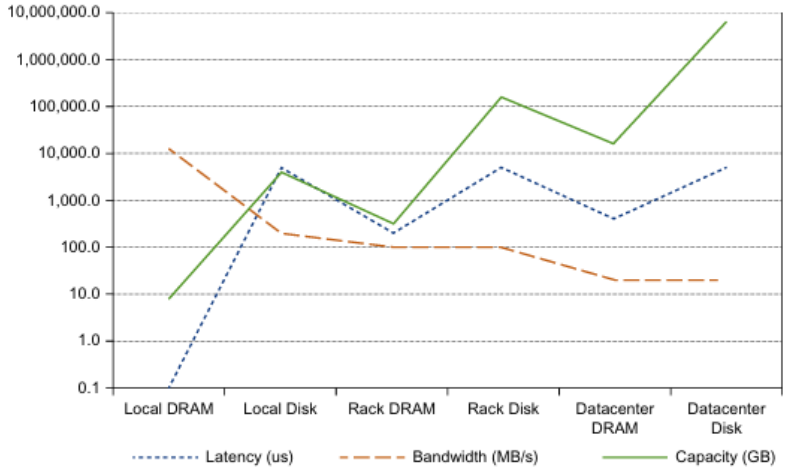

Figure 6.7 Graph of latency, bandwidth, and capacity of the memory hierarchy of a WSC for data in Figure 6.6 (Barroso et al., 2013).

Figure 6.7

a. Take Figure 6.7 and include these points in the local server hierarchy. Assuming that identical performance scaling factors like DRAM are accessed at different hierarchy levels, how do these flash memory devices compare when accessed across the rack? Across the array?

First, let's create a table with all our data points including the SSD and PCIe:

Memory Type	Latency (μs)	Bandwidth (MB/s)
Local DRAM	0.1	20,000
PCIe Flash	35	600
SSD (SATA)	75	250
Local Flash	100	1,000
Local Disk	10,000	200
Rack DRAM	300	100
Rack Flash	400	100
Rack Disk	11,000	100
Array DRAM	500	10
Array Flash	600	10
Array Disk	12,000	10

From this we can now begin to break down some comparisons for the flash devices.

- **Locally** : PCIe Flash (35 μs , 600 MB/s) is noticeably faster than SSD (75 μs , 250 MB/s).
- **Across the rack** : The latency for flash increases from about 100 μs (local flash) to 400 μs (rack flash).

This means that whether using PCIe Flash or SSD, accessing them remotely on the rack introduces a significant delay.

- **Across the array** : The latency increases further to around 600 μs (array flash). This severe latency penalty shows that both SSD and PCIe Flash perform much worse when accessed across an array.
- **Bandwidth Impact** : Locally , the flash devices offer around 1,000 MB/s. However, this drops to roughly 100 MB/s across the rack and down to 10 MB/s across the array.

Meaning that both types of flash see dramatic reductions in throughput when not used locally.

Concluding from this, even though PCIe Flash is faster than SSD locally, both suffer major performance mentalities when accessed across the rack or array. Therefore, we can say that flash devices are best used as local storage to maintain their optimal performance.

b. Discuss some software-based optimizations that can utilize the new level of the memory hierarchy.

Some simpler software-based optimizations include:

1. **Local Caching :** Keep frequently accessed data in the fastest, local storage (like DRAM or local flash) to avoid slower remote access.
2. **Smart Data Placement :** Store hot or critical data on faster storage devices (PCIe Flash/SSD) while less-used data goes to slower tiers.

3. **Prefetching :** Load data into cache ahead of time and overlap transfers with computation so that remote accesses don't slow things down.

Question #3

PP607 7.2 (a,b,c)

*Consider the neural network model MLP0 from Figure 7.5. That model has 20 M weights in five fully connected layers (neural network researchers count the input layer as if it were a layer in the stack, but it has no weights associated with it). For simplicity, let's assume that those layers are each the same size, so each layer holds 4 M weights. Then assume that each layer has identical geometry, so each group of 4 M weights represents a 2 K*2 K matrix. Because the TPU typically uses 8-bit numerical values, 20 M weights take up 20 MB.*

Name	DNN layers	Weights	Operations/Weight
MLP0	5	20M	200
MLP1	4	5M	168
LSTM0	58	52M	64
LSTM1	56	34M	96
CNN0	16	8M	2888
CNN1	89	100M	1750

Figure 7.5 Six DNN applications that represent 95% of DNN workloads for inference at Google in 2016, which we use in Section 7.9. The columns are the DNN name, the number of layers in the DNN, the number of weights, and operations per weight (operational intensity). Figure 7.41 on page 595 goes into more detail on these DNNs.

a. For batch sizes of 128, 256, 512, 1024, and 2048, how big are the input activations for each layer of the model (which, except for the input layer, are also the output activations of the previous layer)? Now considering the whole mode (i.e., there's just the input to the first layer and the output from the last layer), for each batch size, what is the transfer time for input and output over PCIe Gen3 x16, which has a transfer speed of about 100 Gibit/s?

From the figure and the provided info in the question, we can see the following:

- MLP0 has 5 fully connected layers.
- Thus, each layer has 4M weights, forming a 2K x 2K matrix.
- The batch sizes we have are 128, 256, 512, 1024, 2048.
- Our transfer speed is 100 Gibit/s which equals 12.5 GB/s.

Each fully connected layer in MLP0 is represented by a 2K × 2K (i.e. 2048 × 2048) weight matrix. This means the activation for each layer is a vector of 2048 elements. Since each activation is 1 byte (8 bits), the size per layer is:

$$Activation\ Size\ per\ Layer = Batch\ Size * 2048\ Bytes$$

Now calculating the activation size per layer per batch size below :

Batch Size	Activation Size Per Layer (Bytes)
128	128 * 2048 = 256 KB
256	256 * 2048 = 512 KB
512	512 * 2028 = 1 MB

1024	1024 * 2048 = 2 MB
2048	2048 * 2048 = 4 MB

Although MLP0 has 5 layers, only the input to the first layer and the output from the last layer cross the PCIe interface. Thus, the total data transferred per inference is:

$$Total\ Activation\ Size = 2\ x\ Activation\ Size\ per\ Layer$$

Batch Size	Total Activation Size (Bytes)
128	0.5 MB
256	1 MB
512	2 MB
1024	4 MB
2048	8 MB

PCIe Gen3 x16 runs at about 100 Gibit/s, which is roughly 12.5 GB/s (or 12,500 MB/s).

$$Trasfer\ Time = (\ Total\ Data\ in\ MB\ /\ 12{,}500) * 1000$$

Batch Size	Transfer Time (ms)
128	(0.5 / 12,500) * 1000 = 0.04 ms
256	(1 / 12,500) * 1000 = 0.08 ms
512	(2 / 12,500) * 1000 = 0.16 ms
1024	(4 / 12,500) * 1000 = 0.32 ms
2048	(8 / 12,500) * 1000 = 0.64 ms

Even though the MLP0 model has 5 layers, only the first input and the final output are transferred over PCIe. For each batch size, the transfer time is computed based on the combined size of these two activations, resulting in the times above.

b. Given the memory system speed of 30 GiB/s, give a lower bound for the time the TPU takes to read the weights of MLP0 from memory. How much time does it take for the TPU to read a 256 x 256 "tile" of weights from memory?

We know that the memory system's speed is 30 GiB/s and the total weights in MLP0 is 20 MB. We can then :

Convert Memory speed to MB/s : $30 * 1024 = 30{,}720\ MB/s$

Compute read time : $\frac{20\ MB}{30{,}720\ MB/s} = 0.00065\ s = 0.65\ s$

Considering for each 256 x 256 weight tile which is 65,536 weights, with each weight being 1 byte, the tile size is 65.5 KB. So, computing read time can be done below :

$$\frac{65.5\ KB}{30{,}720\ MB/s} = 0.0021\ ms = Time\ to\ Read\ a\ 256\ x\ 256\ Tile$$

c. Show how to calculate the TPU's 92 T operations/second, given that we know that the systolic array matrix multiplier has 256 256 elements, each of which performs an 8-bit multiply-accumulate operation (MAC) each cycle. In high-performance-computing marketing terms, a MAC counts as two operations.

Computing the Operations per cycle here we know that based on the matrix :

$$256 * 256 = 65{,}536\ MAC\ units$$

And each unit performs to ops per cycle, so :

$$65{,}536 * 2 = 131{,}072\ ops\ per\ cycle$$

Now compute the total operations per second :

$$131{,}072 * 10^9 = 92.1\ TeraOps\ per\ second.$$

Question #4

Assume a NetApp storage server has the following components and MTTF:

a) *10 disks, each rated at 1,000,000-hour MTTF*

b) *SCSI controller, 500,000-hour MTTF*

c) *power supply, 200,000-hour MTTF*

d) *fan, 200,000-hour MTTF*

e) *SCSI cable, 1,000,000-hour MTTF*

The age of the component is important in failures; all the above failures are independent. Compute the MTTF of the system as a whole.

We can start by calculating the MTTF for each component :

$$Failure\ Rate = \frac{1}{MTTF}$$

$$Failure\ for\ Disks = \frac{10}{1{,}000{,}000} = 0.00001$$

$$Failure\ for\ SCSI\ Controller = \frac{1}{500{,}000} = 0.000002$$

$$Failure\ for\ Power\ Supply = \frac{1}{200{,}000} = 0.000005$$

$$Failure\ for\ Fan = \frac{1}{200{,}000} = 0.000005$$

$$Failure\ for\ SCSI\ cable = \frac{1}{1{,}000{,}000} = \ 0.000001$$

Now we can sum up the independent failures to obtain the total failure rate :

$$0.0001 + 0.000002 + 0.00005 + 0.00005 + 0.000001 = 0.000023$$

Now knowing this we can compute the System MTTF :

$$MTTF_{system} = \frac{1}{0.000023} = 43{,}478.26\ hours.$$

Question #5

Given a more powerful A100 GPU 80GB PCIe, using a FP16 tensor core at 1,935GB/s memory bandwidth, we run one common workload -- a seven billion parameter LLM like Llama 2.

Calculate:

1) the operations per byte (ops:byte) ratio

From NVIDIA A100 Datasheet :

$$FP16\ Tensor\ Core\ Compute\ Bandwitdh = 312\ TFLOPS$$

$$GPU\ Memory\ Bandwith = 1{,}935\ GB/s$$

To solve properly lets first convert TFLOPS to FLOPS and GB/s to B/s :

$$Compute\ Bandwith = \ 312 * 10^{12}\ FLOPS$$

$$Memory\ Bandwith = 1{,}935 * 10^{9}\ B/s$$

Now computing the ratio :

$$\frac{312 * 10^{12}\ FLOPS}{1{,}935 * 10^{9}\ B.s} = 161.2\ ops/byte$$

So, the operations per byte ratio is ~ 161.2 ops / byte.

2) The arithmetic intensity. Arithmetic intensity is the number of compute operations an algorithm takes divided by the number of byte accesses it requires and is a hardware-agnostic measurement.

We know that the Arithmetic Intensity is given by :

$$Arithmetic\ Intensity = \frac{Total\ Compute\ Operations}{Total\ Memory\ Movements}$$

And from the LLM Inference Guide for Llama 2 7B was calculated at 62 ops / byte.

3) to make the most of the compute capacity that we're paying for, how many requests do we want to batch a time during inference to fill our KV cache to increase the throughput?

From the LLM Inference guide, the calculate for batch size follows the available kv cache size. So with our available GPU memory for KV Cache at :

$$80\ \ GB - 14\ GB\ (Model\ Weights) = 66\ GB$$

And the maximum number of tokens that can fit into that GPU Memory :

$$\frac{66\ GB}{0.00052\ GB/token} = 126{,}923\ tokens$$

Since Llama 2 7B has a sequence length of 4096 tokens, the optimal batch size is :

$$\frac{126,923}{4096} \cong 31$$

Thus, to maximize compute efficiency, we should batch approximately 31 requests at a time.