Systems for ML: Training

COS 316: Principles of Computer System Design
Lecture 18

Wyatt Lloyd

Training a Big LLM Challenges

- **Massive, MASSIVE compute job, how can we finish it in a reasonable timeframe (~months)?!**
- Failures are inevitable, how do we deal with them?
- How do we store and access all the training data?

How does the Model Learn?

Training Data

9111	1234	9081
9111	1234	8291
9081	8291	1234
9111	8291	1234
1234	1234	9081

How Big is this Computation?

- DeepSeek-V3:
  - Model has 671 billion parameters
  - Trained on 12.8 trillion tokens

- Training a 4k parameter model on my laptop (microgpt)
  - 1000 tokens takes 40 seconds

- Training DeepSeek-V3 on my laptop would take longer than:
  - $1.28*10^{13}$ tokens * 40s/1000 tokens = $5.12*10^{11}$s = 16224 years

Let’s Make This Faster

- Training on dedicated GPU clusters instead of my wimpy laptop

- Llama3 was trained on:
  - 24,576 H100 GPUs
    - 8 GPUs/server
    - 2 servers/rack
    - 192 racks/pod
    - 8 pods

How Do We Used 24K GPUs?!

- Data Parallelism
- Expert Parallelism
- Pipeline Parallelism
- Context Parallelism
- Tensor Parallelism

Data Parallelism

- Loss / gradients are typically calculated over minibatches
- Use a minibatch size > number of GPUs
- Calculate parameter updates in parallel
  - GPU 1 processes documents 1-10
  - GPU 2 processes documents 11-20
  - GPU 3 processes documents 21-30
  - …
- Average the parameter updates from all GPU models and continue
  - (The math works out for this)

Memory Problems

- Each parameter requires 8 bytes
  - 10 billion parameters requires 80 GB of memory
  - H100s have 80 GB of memory

- We can’t fit a 671 billion parameter model in there!
  - Expert parallelism → ~37 billion parameters per expert
  - Still doesn’t fit

Shard the model across GPUs

- **Pipeline Parallelism**: Distribute the layers of model across GPUs

- GPU 1 has layers 1-12
- GPU 2 has layers 13-24
- …
- GPU 5 has layers 49-60

- Communicate computations between layers

- Now our 37 billion parameters / 5 ~= 7 billion/GPU ~= 70GB memory and fits in an H100!

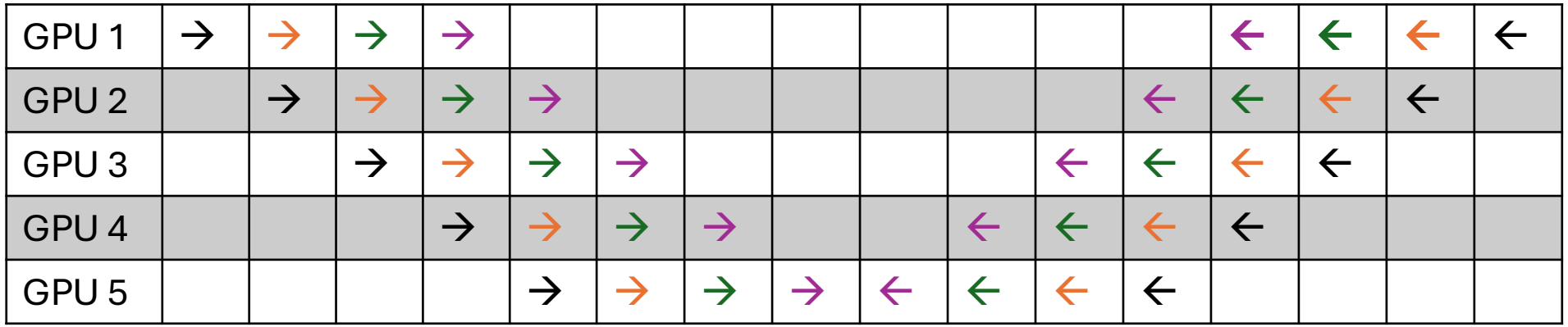
Pipeline Parallelism

GPU 1	→								←
GPU 2		→						←	
GPU 3			→				←		
GPU 4				→		←			
GPU 5					→	←			

We've solved the memory problem, but what is not great about the current solution?

Pipeline Parallelism

Let’s fill the pipeline



Before we were using 1/5 of available GPUs

Now we are using 1/2!

Parallelism in Practice

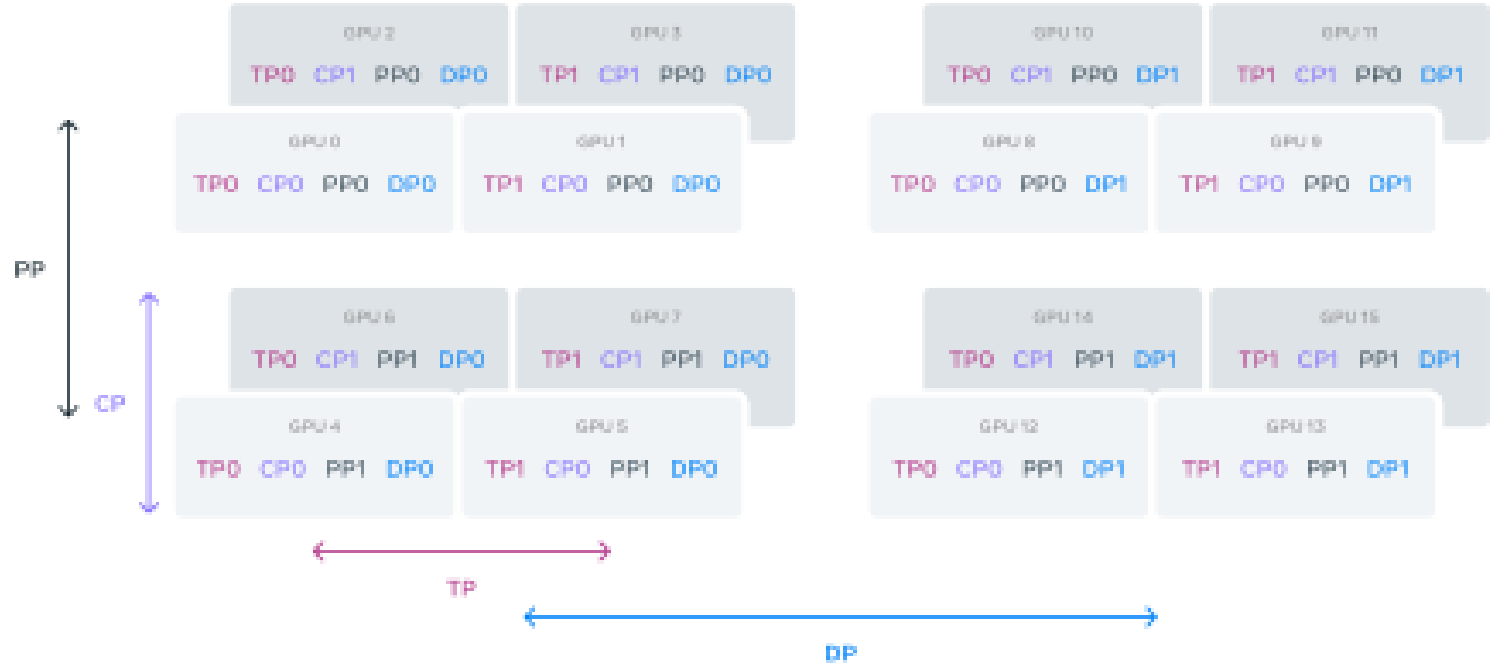


Figure 5 Illustration of 4D parallelism. GPUs are divided into parallelism groups in the order of [TP, CP, PP, DP], where DP stands for FSDP. In this example, 16 GPUs are configured with a group size of |TP|=2, |CP|=2, |PP|=2, and |DP|=2. A GPU's position in 4D parallelism is represented as a vector, $[D_1, D_2, D_3, D_4]$, where D_i is the index on the i-th parallelism dimension. In this example, GPU0[TP0, CP0, PP0, DP0] and GPU1[TP1, CP0, PP0, DP0] are in the same TP group, GPU0 and GPU2 are in the same CP group, GPU0 and GPU4 are in the same PP group, and GPU0 and GPU8 are in the same DP group.

Llama Team “The Llama 3 Herd of Models” Arxiv 2024

Training a Big LLM Challenges

- Massive, MASSIVE compute job, how can we finish it in a reasonable timeframe (~months)?
- **Failures are inevitable, how do we deal with them?!**
- How do we store and access all the training data?

Failures are inevitable

- 24K GPUs x several months …

Failures are inevitable

Component	Category	Interruption Count	% of Interruptions
Faulty GPU	GPU	148	30.1%
GPU HBM3 Memory	GPU	72	17.2%
Software Bug	Dependency	54	12.9%
Network Switch/Cable	Network	35	8.4%
Host Maintenance	Unplanned Maintenance	32	7.6%
GPU SRAM Memory	GPU	19	4.5%
GPU System Processor	GPU	17	4.1%
NIC	Host	7	1.7%
NCCL Watchdog Timeouts	Unknown	7	1.7%
Silent Data Corruption	GPU	6	1.4%
GPU Thermal Interface + Sensor	GPU	6	1.4%
SSD	Host	3	0.7%
Power Supply	Host	3	0.7%
Server Chassis	Host	2	0.5%
IO Expansion Board	Host	2	0.5%
Dependency	Dependency	2	0.5%
CPU	Host	2	0.5%
System Memory	Host	2	0.5%

Table 5 Root-cause categorization of unexpected interruptions during a 54-day period of Llama 3 405B pre-training. About 78% of unexpected interruptions were attributed to confirmed or suspected hardware issues.

Llama Team “The Llama 3 Herd of Models” Arxiv 2024

Fault Tolerance

- We could use replicated state machines to mask these failures…
  - *Why is this not a great idea?*

Fault Tolerance w/ Checkpoints

- Checkpoint progress periodically
- Write checkpoint to storage system
- If there is a failure, restart from the previous checkpoint

- Tradeoff: Checkpoint frequency
  - Checkpoint more often → less work to redo on failure
  - Checkpoint more often → more extra work done on each failure
  - Tune to minimize expected completion time
  - (and optimize checkpointing and restarting)

Who Does the Restart?

- Ideally the training system:
  - 419 unexpected interruptions during 54-day period of llama3 training
    - 3 required manual intervention
- How?
  - Failure detection
    - Ping, read stats on devices, monitor performance from other devices, …
  - Automated remediation
    - Restart device or exclude from cluster until fixed/replaced

Training a Big LLM Challenges

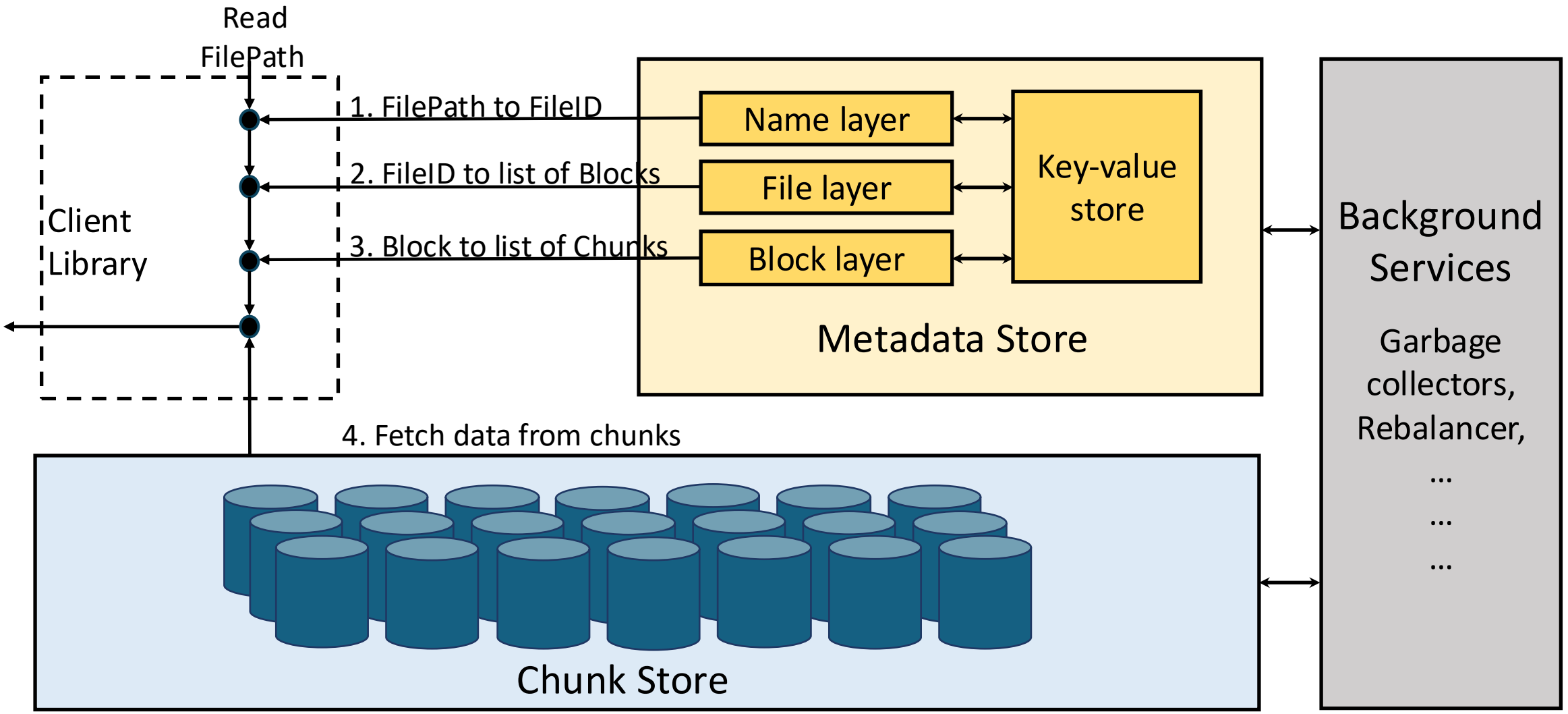
- Massive, MASSIVE compute job, how can we finish it in a reasonable timeframe (~months)?
- Failures are inevitable, how do we deal with them?
- **How do we store and access all the training data?!**

How to Store and Access all this data?

- Training data AND intermediate data like checkpoints

- Llama3 used a distributed storage system (Tectonic) with:
  - 240 Peta bytes of storage across 7500 servers
  - Used SSDs not HDDs

Tectonic Design



Pan et al. “Facebook’s Tectonic Filesystem: Efficiency from Exascale.” Fast 2021 (Talk Slides)

Fault Tolerance for Tectonic

- Metadata store
  - Very frequently accessed
  - Some data changing frequently
  - Want high availability
    - If a server is down, takes down some big chunk of the storage
  - Solution: Implemented with Replicated State Machines

- Chunk store
  - Stores the actual 240 PB of data
  - Want high-ish availability
    - If a chunk server is down, only affects reading data from it
    - (can just write data elsewhere)
  - But cannot lose data

Fault Tolerance for Tectonic's Chunk Store

- Can replicate for fault tolerance
  - Store 3 copies of the data on 3 different disks
  - But requires 3x the number of disks

- Alternative: erasure coding
  - Simple example is XOR:
    - Instead of storing A, A, B, B
    - Store A, B, A XOR B
  - In practice use Reed-Solomon codes for tunable fault tolerance and tunable tradeoff between construction cost, reconstruction cost/storage overhead

Training a Big LLM

- Massive, MASSIVE compute job, how can we finish it in a reasonable timeframe (~months)?
  - Use GPUs, lots
  - Data parallelism, pipeline parallelism, more parallelisms
- Failures are inevitable, how do we deal with them?
  - Checkpoint computation periodically
  - Failure detection and automated remediation
- How do we store and access all the training data?
  - Distributed storage system