Scaling Deep Learning Pipeline on Superpod H800 Infrastructure

Information Technology Service Center
The Hong Kong University of Science and Technology

Speaker: Kin Fai TSE
Manager (Research Computing), Ph. D.

19 Nov 2024

Code snippets in this slides will be available at: https://github.com/hkust-hpc-team/hkust-hpc
under /workshops/20241119-scaling-dl-pipeline

Last Updated: 6/24/2025

Scaling DL Pipeline…?

Why do we even care

Faster Research Outputs; Better Quality of Life

Research & Course works are *Time Sensitive*

- Before becoming obsolete / irrelevant
- Before deadline

Superpod platform is designed to *Scales Horizontally*

- Most tools are not designed to scale
  - Linux shell utils does not scale beyond single core
  - Python libraries commonly 1 core

It is <u>quite trivial</u> to do it right

- Simple order-of-magnitude (gu)estimate
  Use Kilo ≈ 1000 (103), +15% to final answer – instead of Kilo = 1024 (210)
- All command, recipe, workflows included

DL Pipeline
Common Activities

Data Acquisition
- Public Datasets
- Web Crawling
- Acquire from Data Owners
- Professionally Prepared
- Generated

Data Ingress
- Data transfer into Superpod storage

Data Preprocessing
- Data filtering
- Format conversion
- Batch processing (domain-specific)

DL Model Training

Model Evaluation

Data Loading
- Getting data from storage to training nodes

Scale-out Training
- Model training on multiple GPUs
- Model training across multiple nodes

Checkpointing
- Writing parameters to storage

Over past few years
What changed? What did not?

DL Research

- **Model** – Towards 10B+ Params.: 7B, 13B, 8x7B, 70B, …
- **Dataset** – Towards 10TBs, 100TBs, PBs
- **Data** – Higher Data Dimensions: 3D Video, Scene Model; 4D: Time-series Volumetric Data; Graph

Superpod (per-node)

- **Accelerator** – Dense matrix 1TFLOPS[3] of fp/bf16, 2TFLOPS fp8: ≈12-24 RTX4090 in one node
- **Memory & Compute** – 2TB, 224 Threads
- **T1-fs[1] File Read** – 20K IOPS[4] or 20 GB/s[5]
  - /scratch
- **T1-fs File Write** – 10 GB/s[5] Seq. Write

General Technology

- **Storages** – Capacity Scale Faster than bandwidth
- **Primary Data Pool** – T2-fs[2]: NFS / S3
  - /home
  - /project
- **Unix / Linux tools** – 30 Years of `ls; cp; mv; rm`
- **Harddisk Speed** – Largely Unchanged
- **PC** – Hard to Contain a Dataset

Internet Infrastructure

- **Campus Network** – 1Gbps
- **Campus WiFi** – 384Mbps
- **Overseas Connection** – approx. 1Gbps, if lower, just pray

[1] Tier 1 storage (T1-fs): /scratch
[2] Tier 2 storage (T2-fs): /project, /home
[3] FLOPS: Floating-point Operations per sec.
[4] IOPS: IO Operation per sec.
[5] Subject to availability of cluster-wide IO bandwidth

Every ten-fold increase in scale brings new engineering challenges

Frederick P. Brooks

[1] Brooks, Frederick P. (1986). doi:10.1109/MC.1987.1663532
[2] https://en.wikipedia.org/wiki/No_Silver_Bullet

Sample Workflow

Multi-modal LLM Training

Data Acquisition

4 TB of public text corpus to download

Sources:

- huggingface
- public git LFS
- some list of URLs

Data Ingress

20 TB of in-house audio dataset, ea.

- *[name].wav*
  - 10-30s raw audio
  - 400KB-4MB range
  - average ~1M
- *[name].txt*
  - Transcript for [name].wav

Data Preprocessing

- For text corpus:
  - Tokenize
- For audio/text pair:
  - Filter short audio
  - Tokenize using an ML Model
- Prepare tokenized audio and text into readily loadable format

DL Model Training

Model Evaluation

Data Loading

- Sample from all available datasets at some ratio

Scale-out Training

- Dense 7B parameters
- Precision: 16 bit
- Target: Training through 1T tokens
- Context length at 4K

Checkpointing

- Checkpoint for resume training
- At least every 15 mins.

We will refer to this workflow for some sample estimations.

Data Acquisition

1. Potential pitfalls and advice for downloading from Internet
2. Recipes for download, from
   a) Huggingface
   b) git LFS
   c) URLs

Data Acquisition

Potential Pitfalls & Suggestions

Potential Issues

- **Large volume** – Both in size and # of files
- **Geographically distributed** – Latency matters
- **Shared, limited overseas BW** – Costlier than local
- **Physical Limitations** – Same method, increased distance ⇒ smaller bandwidth

Solutions

- **Re-use** – Find local copies, avoid download if possible
- **Reduce** – Download what you use / Download On-the-Fly
- **Proximity** – Find nearer / local mirrors
- **Load-balance** – Download from multiple mirrors for redundancy and load balance
- **Batch** – If supported

Data Acquisition

General Advices

Technical

- Download to T2-fs
  - /project
- Always checksum if available
- Estimate a completion time (ETA) based on download speed[1]
- Think of how to resume if download is broken

Non-technical

- Try to complete several files so you can start development earlier
- Be ***Legal, Ethical and Fair*** when using online data, including downloading

*Don't be **too aggressive**, it will back-fire*

[1] See data transfer section for a Gbps / MBps ⇔ TB-per-day conversion chart

An insider story…

What is *too* *aggressive*?

On 21 Jun.
NSCD[1], used to speed up most network access stuff
found dead on both login nodes

```
Jun 21 03:40:25 slogin-01 systemd[1]: nscd.service: Main process exited, code=dumped, status=6/ABRT
Jun 21 03:40:25 slogin-01 systemd[1]: nscd.service: Failed with result 'core-dump'.
Jun 21 03:40:25 slogin-01 systemd[1]: nscd.service: Consumed 1min 50.579s CPU time.
Jun 21 03:40:25 slogin-01 systemd[1]: nscd.service: Scheduled restart job, restart counter is at 1.
Jun 21 03:40:25 slogin-01 systemd[1]: nscd.service: Consumed 1min 50.579s CPU time.
Jun 21 03:40:53 slogin-01 systemd[1]: nscd.service: Main process exited, code=dumped, status=6/ABRT
Jun 21 03:40:53 slogin-01 systemd[1]: nscd.service: Failed with result 'core-dump'.
Jun 21 03:40:53 slogin-01 systemd[1]: nscd.service: Scheduled restart job, restart counter is at 2.
Jun 21 03:41:20 slogin-01 systemd[1]: nscd.service: Main process exited, code=dumped, status=6/ABRT
Jun 21 03:41:20 slogin-01 systemd[1]: nscd.service: Failed with result 'core-dump'.
```

Cause?

An overflow bug in NSCD

with trigger condition

"Hundreds unique names resolved per second"

"For an hour non-stop"

Who would even think of for *normal use case?*

You can *avoid it by not being too aggressive*

[1] Name Service Cache Daemon, caches DNS resolutions (look-ups) among other stuff.

Don’t push the boundary of the Internet.
It will break.

Data Acquisition Recipe
HuggingFace

If you were using HuggingFace elsewhere (e.g. lab)

- Transfer your HuggingFace cache from Lab into Superpod (this directory below)

```
~/.cache/huggingface$ ls
datasets hub modules token
```

Otherwise – Use CPU queue to download

- You only need that line loading the model, not GPU

```
from transformers import AutoModelForCausalLM, AutoTokenizer
model_id = "mistralai/Mixtral-8x7B-v0.1"
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id)
```

- Or use huggingface-cli

[1] Please refer to data transfer section for best transfer option.

Data Acquisition Recipe
Git LFS

Extension of Git version control for binary files

- Step [1-3]: Get what you need, ASAP
- Step 4: Minimize storage use

```
# 1. clone without downloading any file [1]
git clone --no-checkout https://huggingface.co/datasets/m-a-p/Matrix large-text-data
cd large-text-data
# Set some faster download configs for this repo
git config lfs.concurrenttransfers 20 # is probably the max
git config lfs.batch true
git config lfs.transfer.maxretries 10

# 2. look at what files you are interested most
# at most ~20-30 files you would prioritize working on
~/large-text-data$ git lfs ls-files -n
A_cc_en_sample.jsonl
book_all.0000.jsonl
…
wiki_news.0000.jsonl
wiki_qa.0000.jsonl
```

When downloading

- use include, exclude and wildcard (*) to select paths

```
# 3. only download the files you need (first)
# Use CPU queue, 2-4 cores is enough
git lfs pull --include="wiki*,paper_math*" --exclude="wiki_qa*"
```

Best practices

- Do not modify raw downloaded files
- House-keep the download cache daily during download
  - Otherwise, you need 2x storage size

```
# 5. delete the files cached by git LFS
# This shows you what would be deleted, the hash matches `git lfs ls-files`, does not delete anything
git lfs prune --dry-run --verbose --force
# !This line DELETES the cache
#git lfs prune --verbose --force
```

Dataset for actual use[1] or practice[2]

[1] Zhang, Ge., et al. (2024) MAP-Neo: Highly Capable and Transparent Bilingual Large Language Model Series. DOI:10.48550/arXiv.2405.1932

[1] Courtesy to Ruibin YUAN for sharing Git LFS repo: https://huggingface.co/datasets/m-a-p/Matrix
[2] For Git LFS with small files, please try https://github.com/larsxschneider/lfstest-manyfiles

Data Acquisition Recipe
URL List

List of URLs

- Millions of lines
- Each one line in a text file

```
https://huggingface.co/Qwen/Qwen2.5-...model-00001-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00002-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00003-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00004-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00005-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00006-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00007-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00008-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00009-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00010-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00011-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00012-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00013-of-00014.safetensors?download=true
https://huggingface.co/Qwen/Qwen2.5-...model-00014-of-00014.safetensors?download=true
...
...
...
```

Demo:
Download a portion in a URL list file in parallel

- `cat` - umber non-empty lines to generate filename
- `head | tail` - Last 7 lines end at line 13 (incl.) = line 7-13
- `awk | tr` - Some magic to fix spaces
- `xargs` - 2 parallel downloads, each fork handles 1 url (increase if you have small files)
- `wget` - write file to <number>.bin, download log to <number>.log just in case there is an error

```
# Use CPU queue, 2-4 cores is enough
# Delete "echo" to download
cat -b ../urls.txt \
  | head -n 13 \
  | tail -n 7 \
  | awk '{print $1,$2}' \
  | tr '\n' '\0' \
  | xargs -0 -P 2 -n 1 bash -c 'while IFS=" " read -r name url <<< "$1"
&& shift; do echo wget -o $name.log -O $name.bin "$url"; done' --
```

Data Acquisition

Other Popular Options

You may explore these options if it fits your case as well

- huggingface-cli: CLI for huggingface to download datasets & models
  - https://huggingface.co/docs/huggingface_hub/en/guides/cli
  - https://pypi.org/project/huggingface-hub/
- S3cmd: for access to s3API-compliant storage
  - https://github.com/s3tools/s3cmd
- Rclone: for access to various cloud vendors
  - https://github.com/rclone/rclone
- aria2: An alternative to wget that may work better for list of URL
  - https://aria2.github.io/

Data Transfer

1. Standard workflow for data transfer, including
   a) Ingress/Egress from/to outside of Superpod
   b) Transfer between Tier 1 (T1) Cache Storage and Tier 2 (T2) NFS Storage
2. Simple recipes for parallel file operations

Data Transfer

Data Ingress/Egress Options

via Campus Network

Upload via SSH, available 24x7 as long as cluster login is up

Pros

- Initiate transfer whenever you need
- No notice required

Cons

- Max. 1Gbps = 10 TB per day
- Your PC/Workstation must remain on for the whole transfer
- Transfer during office hour may affect your lab's internet quality

On-site Data Transfer at ITSC Service Center

A service for Bring-Your-Own-Disk transfer at 10Gbps max.
24x7 transfer after initial setup

Pros

- Recommended for >20TB; especially if your data is already on multiple hard-disk or SSD
- ITSC provides workstation for upload
- Does not degrade your lab's Internet quality

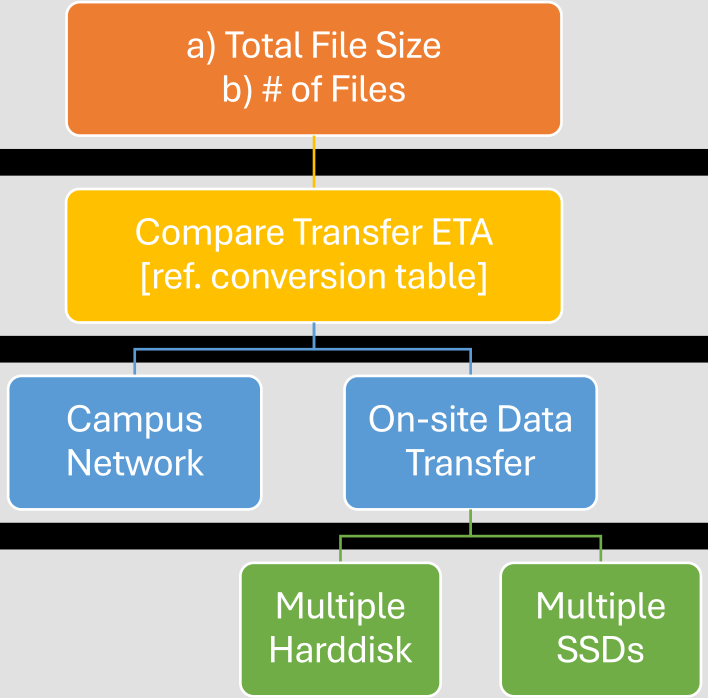
Cons

- Reservation in advance
- Need to copy data to temporary storage
  ITSC offers 44TB hard-disk storage on request
- More manual work

Data Transfer

Reference Decision Flowchart

1. Count
2. Evaluate
3. Choose a Method
4. Prepare for Transfer
5. Parallelize



Data Transfer

Bandwidth Conversion Table

Typical Scenario	Home 100M Broadband	WiFi	**Campus Network / Home 1G Broadband**	**Harddisk[3]**	**Consumer 2.5" SSD[3]**	**Data Upload at ITSC**	NVMe M.2 SSD	NFS Storage - /home - /project	DDN Storage - /scratch (per-node)
Total File Size									
Bit/s[1]	**100Mbps**	**384Mbps**	**1Gbps**			**10Gbps**		**100Gbps**	**800Gbps**
Byte/s[2]	12.5M	48M	125M	**200MBps**	**650MBps**	1.25G	**5GBps**	12.5G	**20GBps**
Byte/hour	45 G	170 G	450 G	720 G	2.3 T	4.5 T	18 T	45 T	360 T
Byte/day	1 T	3.8 T	10 T	16 T	50 T	100 T	400 T	1 P	8 P
Number of Files									
IOPS	-	-	-	200	100 kilo	-	100 kilo		20 kilo
IO/hour				720 kilo	360 mil.		3.6 bil.		720 mil.
IO/day				16 mil.	8 bil.		80 bil.		16 bil.

[1] Widely used in networking to describe transfer speed, = 1/8 of Byte-per-sec
[2] Widely used in storage to describe read write capability
[3] per-disk bandwidth

Data Transfer

Evaluating Transfer Options

Transfer ETA = max(A, B)

- A = Total File Size / min(bandwidths)
- B = # of Files / total IOPS

20 TB of in-house audio dataset, ea. on HDD

- *[name].wav*
  - 10-30s raw audio
  - 400KB-4MB range
  - average ~1M
- *[name].txt*
  - Transcript for [name].wav

Example:

I don't have HDD/SSD packed already

I. Count
   1. Size = 20 TB
   2. # Files = 20 TB / 1MB = 20 mils

II. Estimate
   1. **Transfer via Campus Network**
      A. 20TB / 10TB-per-day = **48 hours**, or **4 nights** if not copying during office hour
      B. N/A
   2. Transfer via one 32TB HDD
      A. 20TB / 2.3TB-per-hour *x 2* = 18 hours
      1 Transfer into HDD, 1 transfer out of HDD
      B. 20 mils / 16 mils-per-day *x 2* = **2.5 days**
   3. **Transfer via multiple 8TB SSD**
      A. 20TB / 4.5TB-per-hour *x 2* = **9 hours**
      B. 20 mils / 360 mils-per-hours *x 2* = 7 mins

Data Transfer Recipe
Parallel File Operations

`ls mv cp scp rm`

- Not for TBs of files
- Not for billions+ files

Use parallel substitute on the right

- with caveats! NOT identical
- read each command's --help before use

`xargs` is good stuff

```
#ls -a # find
# ==>
fdfind --one-filesystem --unrestricted --unrestricted . $PWD

#mv ==> copy, always VERIFY, then remove

#[s]cp -dr /path/to/src/ [user@remote:]/path/to/dest/
# ==>
# 1. use fpsync parallel copy
fpsync -t ~/.fpsync/ -vv -n 8 -O '-x|.zfs' -f 16384 -n 20G
/path/to/src/ [user@remote:]/path/to/dest/
# 2. use rsync to check for missing or incomplete
rsync -ahHXic /path/to/src/ [user@remote:]/path/to/dest/

#rm -rf /path/to/remove/
# ==>
# Double check your path, it deletes REALLY fast
fdfind --one-filesystem -uu0 . /path/to/remove/ | xargs -r0 -P
$(nproc) -n 256 rm -rf
```

Data Preprocessing

1. Potential pitfalls and advice for batch preprocessing
2. Sample workflow to parallelize in python3

Data Preprocessing

Potential Pitfalls & Suggestions

Potential Issues

- **Large volume**
  Both in size and # of files
- **Heavy workload**
  Processing 1 file may require some parallelize
- **Broken data**
  One broken data crash the job
- **Resource intensive**
  If processed by a DL model
- **Extra effort**
  Do you really need to parallelize that much?

Suggestions

- **Handle Unexpected**
  Your data always contain broken items in some imaginative ways – try-catch each data processed
- **Verify Output**
  Debug through the pipeline with first 100 data, or find the output unusable later
- **Estimate ETA**
  Give a test run, if ETA a few days, just leave it
- **Calculate Required Speedup**
  How much speed up you need to finish it in reasonable time
- **Start Simple**
  Submit 1 job per large file

Our Sample Workflow

1^{st} Iteration

```python
import glob

for filename in glob.glob("*.wav"):
    txt_filename = filename.replace(".wav", ".txt")
    with open(filename, "rb") as f:
        # Tokenize audio with ML model
        pass
    with open(txt_filename, "r") as f:
        # Tokenize text
        pass
```

Data Preprocessing

- For text corpus:
  - Tokenize
- For audio/text pair:
  - Filter short audio
  - Tokenize using an ML Model

One process won't work

- Need multiple instance of ML model and batching to tokenize 20TB audio
- Same text tokenizer for corpus and audio transcript, text tokenizer may support batching
- ~ 20 millions audio files, glob not good idea

Changes

- Use fdfind to build a file list
- Split into 2 scripts that accept args to specify [start:end] from file list to process
  - text processing
  - audio processing
- Submit as SLURM array jobs

```
# Example usage:
./audio_preproc.py filelist.txt 0 10000
./audio_preproc.py filelist.txt 10000 20000
# ...
# Example usage:
./text_preproc.py filelist.txt 0 1000000
./text_preproc.py filelist.txt 1000000 2000000
# ...
```

Our Sample Workflow
2nd Iteration

```python
# file_list = "file_list.txt"; start = 0; end = 1;
# This is a big file, 500 GB of text
with open(file_list, "r") as flist:
    for filename in flist[start:end]:
        filename = filename.strip()
        with open(filename, "r") as current_file:
            file_length = # ...
            # TODO: process it
            pass
```

Data Preprocessing

- For text corpus:
  - Tokenize
- For audio/text pair:
  - Filter short audio
  - Tokenize using an ML Model

The file is 500GB of JSONL

- Each line is a JSON
- Consider parallelization e.g. multiprocessing

Should I read the file here, or in the subprocess?

Answer depends

Maybe some code was readily available

I. Just use it (modify introduce bugs)

Written from scratch

I. Pass only (filename, subrange) into subprocess
II. Subprocess write tokens to file directly, don't return GBs of tokens back to main process

Our Sample Workflow

Final – Minimal Solution

```python
import multiprocessing as mp
import numpy as np
import json
import sys
from pathlib import Path

chunk_size = 1000000

def main():
    filename = sys.argv[1]
    with open(filename, "r") as f:
        total_lines = sum(1 for _ in f)  # count lines
    num_cpus = mp.cpu_count()  # get number of cpus
    with mp.Pool(num_cpus) as pool:
        # queue a job for each 3-tuple below
        # return: list[int] (# valid lines for each chunk)
        valid_lines = pool.starmap(
            runner,
            [  # 3-tuples: (filename, start, end)
                (filename, i, min(i + chunk_size, total_lines))
                for i in range(0, total_lines, chunk_size)
            ],
        )
    print(f"Valid lines: {sum(valid_lines)}/{total_lines}")
```

```python
def runner(filename: str, start: int, end: int, dtype=np.int16) -> int:
    global tokenizer
    sequences = []
    with open(filename, 'r') as f:
        for i, line in enumerate(f):
            if i < start:
                continue
            if i >= end:
                break
            try:
                text = json.loads(line)["text"]
                encoded = tokenizer.encode(text)
                sequences.append(encoded)
            except Exception as e:
                print(f"Error: {e}")
                print(f"Failed to encode text at {i}: {text}")
    sequences = np.array(sequences, dtype=dtype)
    save_path = f"{filename.replace}_{start:9d}.npz"
    np.savez_compressed(save_path, sequences=sequences)
    return len(sequences)
```

Data Loading

1. Dataloading is overhead, but how to be sure?
2. Best practice for data loading

Data Loading
Simple Smoke Test

How to check your data loader's speed?

1. Estimate IO bandwidth required per minibatch / iteration
   1. Should have a size estimate
   2. Won't get true value
2. Just run it
   1. Bypass training code using "continue"
   2. Should run much faster
   3. e.g. 100x compared to with training code

```python
from datetime import datetime

# ...
# Your training loop
for iter_num, train_data in enumerate(train_dataloader):
    print(datetime.now(), iter_num)
    continue

    # your original training code
    (x, y) = train_data
    y = model(x)
    # ...
```

Data Loading
Proper Data Sizing

Goal
- I. Maintain sustainable IO bandwidth
  1. < 500MBps - 1GBps per node
  2. Load from T1-fs

Bandwidth = Size / Time
- I. Time – for GPU to compute 1 iteration
  1. [harder] Depends on model and # params
- II. Size – data consumed in 1 iteration
  1. Size per item
  2. Microbatch size

If loading too much data
- I. No general fix
- II. May need to change model input / architecture

If loading inefficiently
- I. Try increase workers for data loader

Feel free to reach us or drop me an email

Timing GPU to compute 1 iteration

```
from datetime import datetime


def train():
    for iter_num, train_data in enumerate(train_dataloader):
        # at this point, train_data is loaded
        # no data loading issue in-between
        print(datetime.now(), iter_num)

        # your original training code
        (x, y) = train_data
        y_ = model(x)
        # ...
        loss = criterion(y, y_)
        loss.backward()
        optimizer.step()
        # ...

        # last line before end of loop
        loss.item().to("cpu")
        print(datetime.now(), iter_num)
```

Monitoring and Debugging Scale-out Training

1. GPU utilization
   a) Simple recipe to monitor GPU
   b) Common diagnosis and direction
2. Debugging: What if your code stop working for no obvious reason?

Scale-out Monitoring

Session Focus: Monitor GPU is being used effectively

Monitoring / graphing training stat

I. Parameter distributions

II. Loss, learning rate etc.

Here are some options

I. Weights & Biases

II. TensorboardX for pyTorch

[1] If any of these threshold are violated for > few % of samples, and it is not during checkpointing, take a close look at your job.

Scale-out Monitoring

Main Indicators & Instrumentation

3 Main Indicators

1. GPU Utilization %
   1. Is there enough work on GPU?

2. GPU Power Consumption
   1. Calculation takes a lot of power
   2. Memory / Network transfer do not
   3. Typical DL training
      Calculation ≫ Transfers

3. PCI-e Receive (RX) / Send (TX) [to/from GPU]
   1. Loading batched inputs: sustained RX
   2. Checkpointing: A spike TX, slightly longer for large models
   3. Logging: Small sustained TX
      e.g. logging loss every 50 steps

Simple Monitoring along SLURM job

```
test $SLURM_LOCALID -gt 0 ||
  nvidia-smi dmon --delay 15 --select pumt --options DT --filename
$PWD/out/slurm/slurm-$SLURM_JOB_ID.$(hostname).gutil &
# Your python command to launch
stdbuf -e0 -o0 python3 pretrain/redpajama.py
# Kill the nvidia-smi monitoring process
kill %1
wait
```

	Utilization	Power	PCI-e RX/TX
Column Label	sm	pwr	rxpci txpci
Interpretation	Higher is better	Higher is better	Below Threshold
Normal Range	95-100%	500-700W	100-10000MB/s
Threshold[1]	>= 90%	>= 350W	< 20000MB/s

Scale-out Monitoring

Typical Output – nvidia-smi dmon

Please report if gtemp >=85C

20240101	19:49:31	0	552	55	60	100	31	0	0	0	0	74333	993	0	8432	1276
20240101	19:49:46	0	564	55	60	100	61	0	0	0	0	74333	993	0	6776	5024
20240101	19:50:02	0	583	55	62	100	59	0	0	0	0	74333	993	0	6156	1848
20240101	19:50:17	0	571	55	62	100	66	0	0	0	0	74333	993	0	5183	2441
20240101	19:50:33	0	617	50	59	100	66	0	0	0	0	74333	993	0	1304	146
20240101	19:50:48	0	566	50	58	100	51	0	0	0	0	74333	993	0	5881	599
20240101	19:51:04	0	115	40	50	100	0	0	0	0	0	74333	993	0	0	0
20240101	19:51:19	0	569	54	60	100	37	0	0	0	0	74333	993	0	5842	1798
20240101	19:51:35	0	583	54	61	100	31	0	0	0	0	74333	993	0	6053	1818
20240101	19:51:50	0	582	54	62	100	63	0	0	0	0	74333	993	0	6628	1496
20240101	19:52:06	0	565	54	60	100	61	0	0	0	0	74333	993	0	6098	1565
20240101	19:52:21	0	564	55	63	100	59	0	0	0	0	74333	993	0	5879	1561
#Date	Time	gpu	pwr	gtemp	mtemp	sm	mem	enc	dec	jpg	ofa	fb	bar1	ccpm	rxpci	txpci
#YYYYMMDD	HH:MM:SS	Idx	W	C	C	%	%	%	%	%	%	MB	MB	MB	MB/s	MB/s
20240101	19:52:37	0	558	55	62	100	66	0	0	0	0	74333	993	0	3839	139
20240101	19:52:52	0	613	49	58	100	66	0	0	0	0	74333	993	0	1347	386
20240101	19:53:08	0	577	50	59	100	50	0	0	0	0	74333	993	0	5747	607
20240101	19:53:23	0	571	53	60	100	58	0	0	0	0	74333	993	0	5882	612
20240101	19:53:39	0	562	53	61	100	31	0	0	0	0	74333	993	0	8127	1351
20240101	19:53:54	0	571	54	62	100	33	0	0	0	0	74333	993	0	6555	1519
20240101	19:54:10	0	574	56	61	100	62	0	0	0	0	74333	993	0	6484	1549
20240101	19:54:25	0	561	54	62	100	60	0	0	0	0	74333	993	0	5769	1270

Saving checkpoint

Can stay ~550W even for small model on multi-node[1]

If lower than ~70GB increase batch size by 1 until out of GPU mem

Typical RX~10GB/s and TX~1GB/s

[1] Llama LLM down-sized from 7B to ~0.5B fp-16 4096 context windows, DDP on 16 nodes, minibatch=16/GPU, no grad-accu, i.e. global batch size=16*8*16=2048

Scale-out Debugging

Simple trick for multimode issues

For >1 node training

I. When it works on 1 node but not >1

Common scenario

1. Stuck at init

```
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
```

2. Splitted

```
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
```

Can be code / framework configuration / node issue

Turn these logging on

I. Ask around if anyone faced similar issue before

II. May share us a log if you really have no idea
At least we can assert that some node is OK to rule out node issue

```
export PYTHONFAULTHANDLER=1
export CUDA_LAUNCH_BLOCKING=1
export NCCL_DEBUG=INFO
# export NCCL_DEBUG=TRACE
export NCCL_DEBUG_SUBSYS=INIT
```

Checkpointing Techniques

1. Alternatives to taking less checkpoints
2. Taking checkpoint on SLURM timeout

Checkpointing

Opt. Flowchart

Checkpoint(ckpt) sizing and time estimate

Determine a ckpt. Strategy

I. Select the right storage tier

II. Use shared ckpt. If needed

[IMPORTANT] Check you can load a checkpoint

I. Especially for sharded: does it load with different number of GPUs / processes

Have some code that housekeep checkpoints

Checkpointing

Sizing and timing ckpt.

Guestimate by # of parameters

- # params * 5 * precision_in_byte

Our model training:

- 7B parameters
- 16 bit = 2 Bytes
- Size: 70G per 15-min

Scale-out Training

- Dense 7B parameters
- Precision: 16 bit
- Target: Training through 1T tokens
- Context length at 4K

Checkpointing

- Checkpoint for resume training
- At least every 15 mins.

Single-thread bandwidth

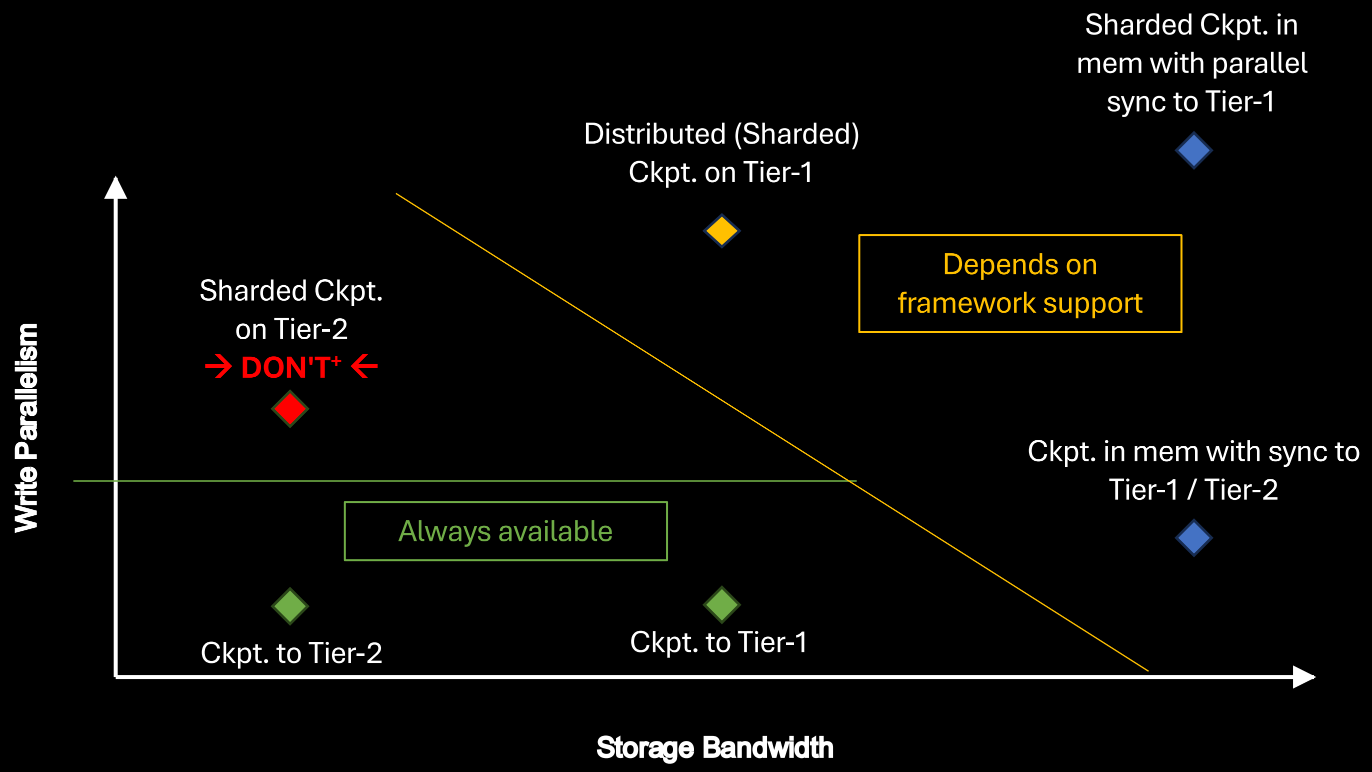
- In-memory [/dev/shm]: 4GB/s
- T1-fs [/scratch]: 2GB/s
- T2-fs [/project]: 500MB/s
  - Note whole-cluster upperbound ~10GB/s

Our example

- 2min+ on T2-fs
- 35s on T1-fs (reasonable)

Checkpointing

Optimization Dimensions



+ We offer Tier-1 storage for the purpose, please *be merciful* to Tier-2

Checkpointing

Last ckpt. on Job Timeout

Some framework may have built-in support

If not, use this

```
with PoxisSignalHandler() as signal_handler:
    # Training loop
    for iter_num, train_data in enumerate(train_dataloader):
        if signal_handler.is_terminated:
            break
# TODO: save last model
# save_statedict(....)
```

```
import signal

class PoxisSignalHandler:
    def __init__(self):
        self.is_terminated = False
        self.last_signal = None

    def __enter__(self):
        signal.signal(signal.SIGHUP, self.handle_exit_signal)
        signal.signal(signal.SIGINT, self.handle_exit_signal)
        signal.signal(signal.SIGTERM, self.handle_exit_signal)
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        signal.signal(signal.SIGHUP, signal.SIG_DFL)
        signal.signal(signal.SIGINT, signal.SIG_DFL)
        signal.signal(signal.SIGTERM, signal.SIG_DFL)

    def handle_exit_signal(self, signum, frame):
        self.is_terminated = True
        self.last_signal = signum
```

Checkpointing

New Ckpt. Methods

Asynchronous checkpointing [1]

- Looks like experimental
- Remember to check for ckpt. loading or corruption if using

Distributed in-memory checkpointing with redundancy

- Various works in this field, e.g. [2]

[1] Please check this for reference: https://pytorch.org/blog/reducing-checkpointing-times/
[2] Wang, Z., et al. SOSP (2023). GEMINI: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints. DOI:10.1145/3600006.3613145

Q & A

Thank you for your participation.

Code snippets in this slides will be available at: https://github.com/hkust-hpc-team/hkust-hpc
under /workshops/20241119-scaling-dl-pipeline