Best Practices in Organizing I/O for ML Projects

Monthly Storage Talks

Ali Doosthosseini | 07.02.2024

KISSKI

KI-Servicezentrum für sensible und kritische Infrastrukturen

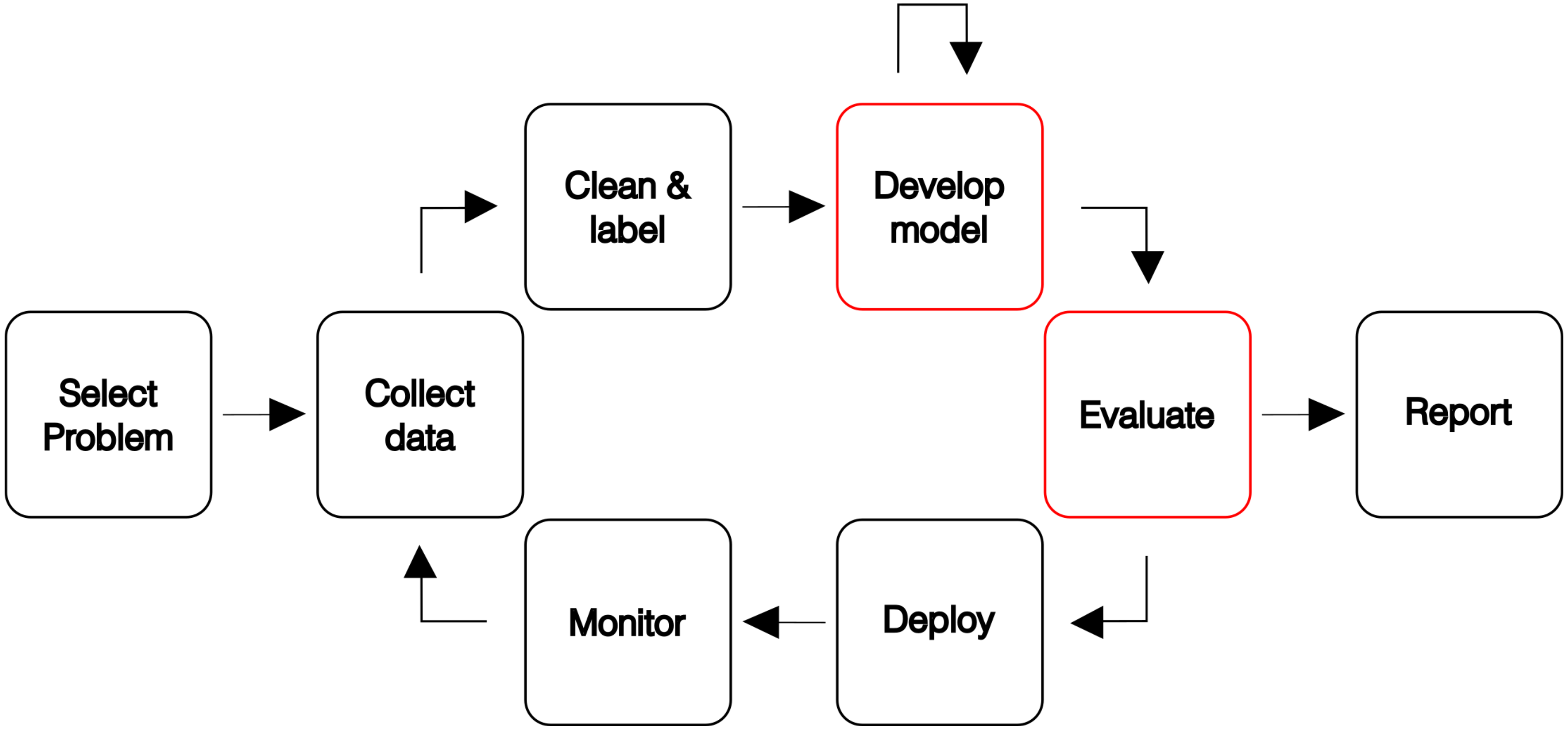
What will be discussed

- Roadmap for ML with GPU cores
- How and where to store data for ML
- Best practices to improve efficiency and reliability

Motivation

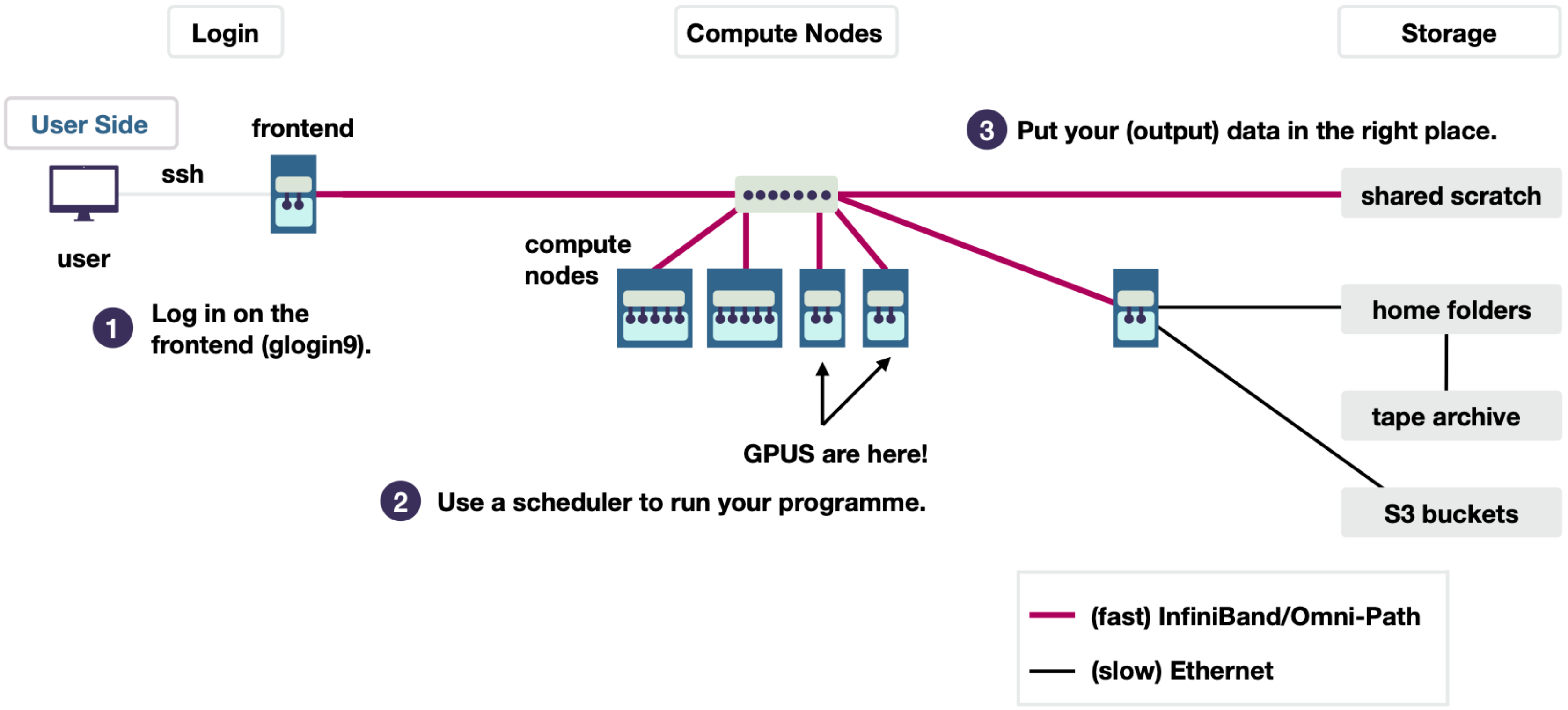
- ML / Deep Learning requires large data
  - Many storage formats and locations exist
  - How to find best use-case
- Data processing involves network, storage I/O, CPU and GPU
  - How to identify potential bottlenecks and improve it
- For parallel training, information needs to be shared among cores
  - Model parallelism and/or data parallelism
  - What software/tools are available to implement this?

Deep Learning with GPUs



Source: *fullstackdeeplearning.com*

Deep Learning on HPC



Source: *DL with GPU course*

Getting Started

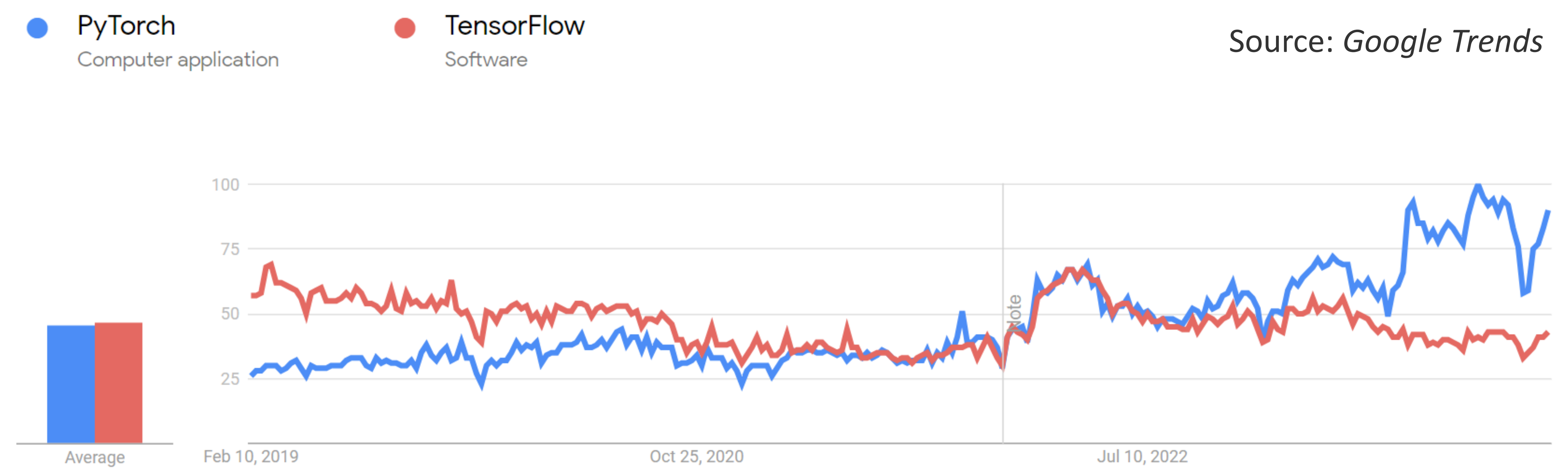
- Python is highly recommended as language
  - Large availability of tools and frameworks
  - GWDG offers JupyterHPC, incl. containers with GPU
- VSCode is recommended IDE
  - Extensions that cover many use cases
  - Compatible with remote development
- Maintain environments with conda or venv
- Containerize with docker, apptainer, singularity, etc.

Available Toolsets and Packages

- PyTorch (recommended)
  - Developed by Facebook in 2016, most popular in academia
- Tensorflow
  - Developed by Google in 2015, still relatively popular.
  - Especially useful in large-scale and production environments
- JAX
  - Currently in development by Google
  - Aimed towards more advanced users.

PyTorch vs TensorFlow

Source: *Google Trends*

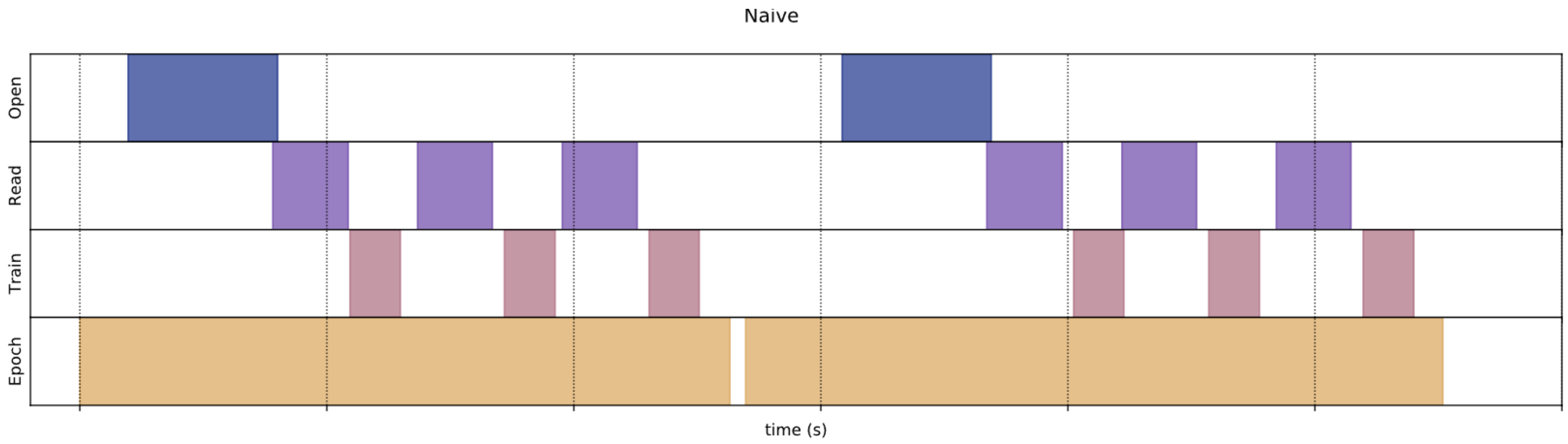


Programming with PyTorch

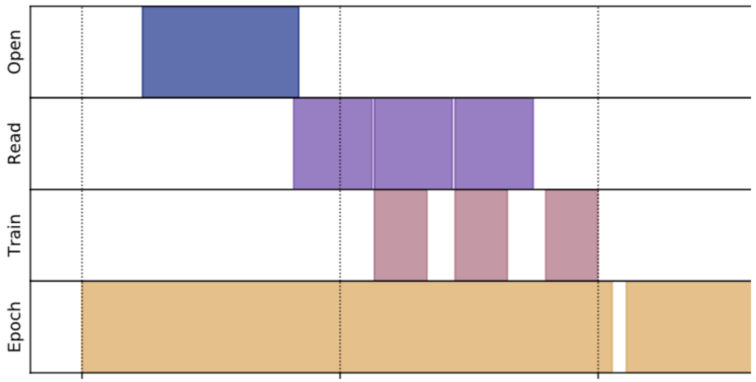
- Fully supports Nvidia GPUs (recommended)
  - Compatible releases for most systems and CUDA versions.
- Supports Apple Metal since 2022
- Limited support for AMD GPUs (not recommended)
- I/O can be optimized using DataLoader and other methods

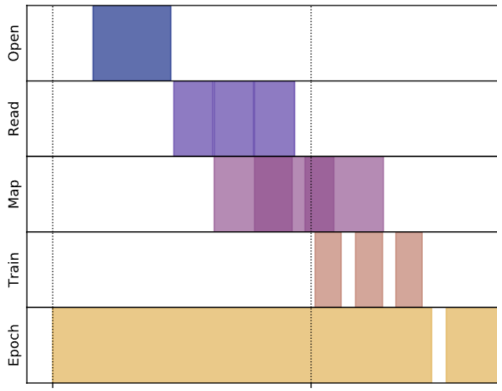
Loading Data into GPU

- Naïve approach
  - Extraction
  - Transformation
  - Loading



- Optimized
  - Prefetching
  - Parallelization
  - Caching

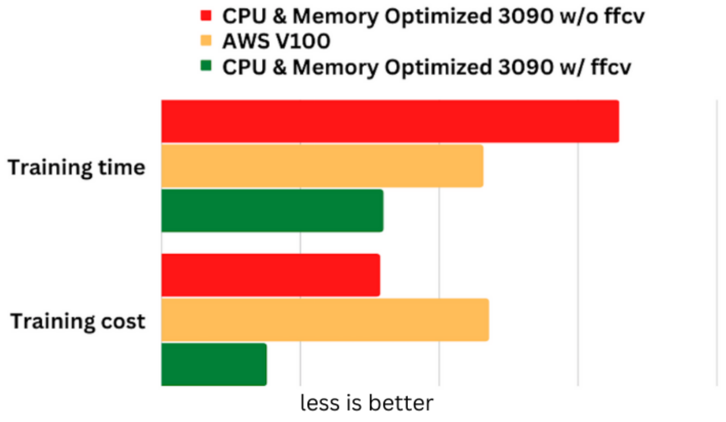




Source: *tensorflow*

Optimizing PyTorch

- FFCV
  - Alleviates I/O bottleneck
  - Easy to replace code with DataLoader
  - 2x to 3x faster on V100 and RTX 3090

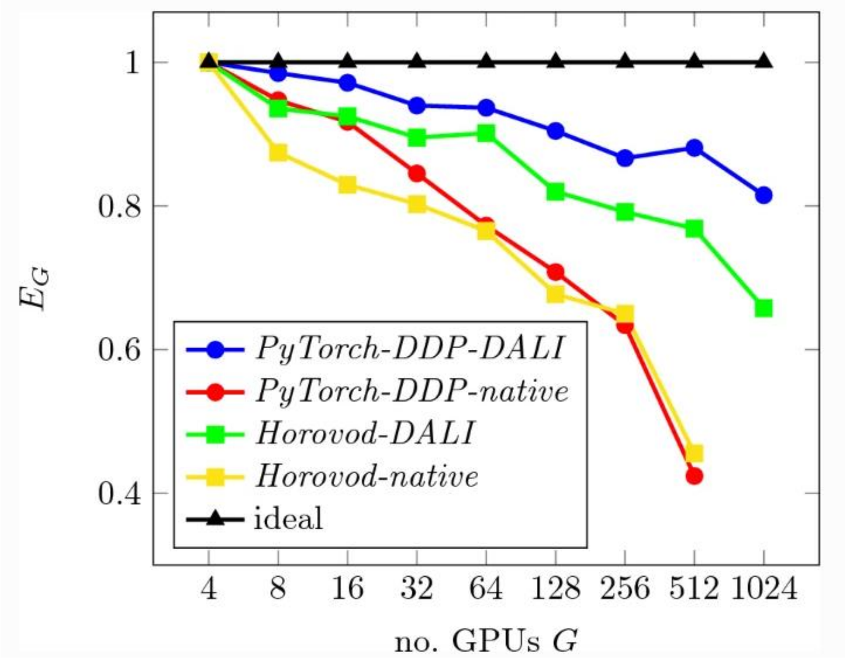


Source: *Genesis Cloud*

- NVIDIA Data Loading Library (DALI)
  - Optimized for performance and flexibility
  - Alleviates CPU bottleneck
  - Achieved 50% to 80% speedup on Jülich cluster

Distributed Deep Learning

- DataDistributedParallel
  - Native PyTorch class
  - Large community, good user support
- Horovod
  - Developed by Uber
  - Compatible with AWS, Azure, Apache Spark
- FairSale
  - Developed by Facebook, fully sharded
- DeepSpeed
  - Developed by Microsoft, supports model parallelism and data parallelism



Source: *Jülich Supercomputing Centre*

Where to store data?

- Codes and scripts
  - Small file size, backup and version control necessary
- Software and containers
  - Large storage supporting many files, fast I/O
- Training/testing data
  - Very large storage, must be reproducible, fast I/O
- Results
  - Must be reproducible, should be stored separately once completed
- Archive
  - Stored in long-term cold storage with backup

GWDG HPC Storage Environment

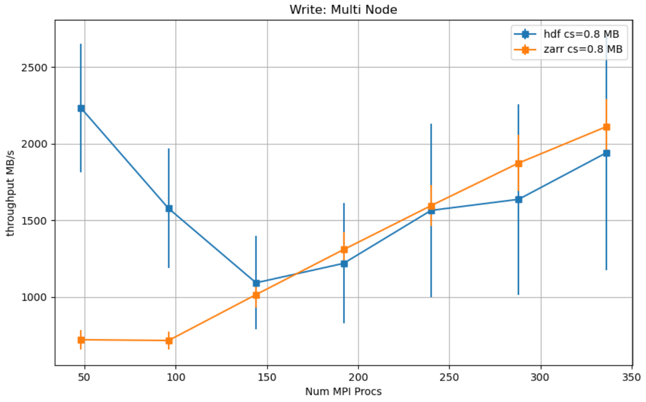
- Work storage: scratch and project folders
  - Very large, fast I/O, no backup
  - Suitable for temporary storage of data and results
- Home storage
  - Small, very fast I/O, regular backups
  - Suitable for storing configurations, sensitive codes and scripts
  - Not suitable for hosting large software
- Tape Archive
  - Extremely large, very slow I/O, RAID Redundancy
  - Suitable for storing backups and storing project data after completion

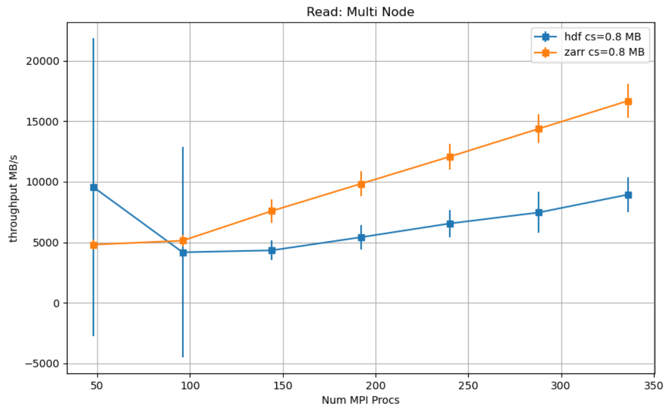
How to store data?

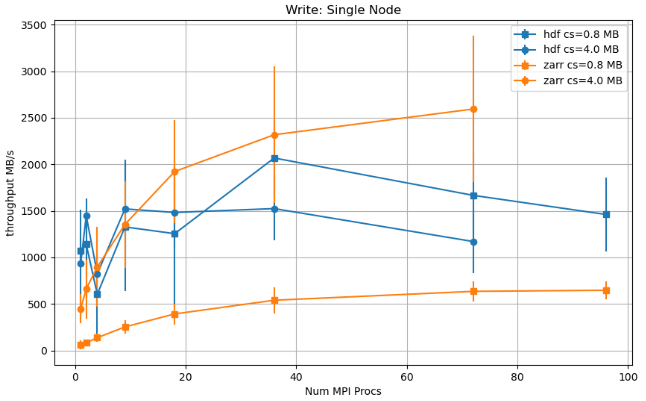
- Investigate suitable formats for specific use-case, storage
- Language-agnostic file formats: JSON, XML, CSV, Feather, HDF5, Parquet, Pickle
- Analyze performance and determine use cases, e.g.:
  - **JSON**: Not suitable for large and complex data; slow on read
  - **XML**: Slow performance
  - **CSV**: Cannot store complex data types, e.g. images, audio. Difficult to handle missing data
  - **Feather**: Very fast, lightweight. May not be suitable for data with multiple data frames.
  - **HDF5**: Designed to store large and complex data, supports compression, parallel I/O, data chunking, fast performance. Requires specialized software.
  - **Parquet**: Developed for big data processing. Very fast. Efficiency by partitioning and compressing data columns. Requires specialized software.
  - **Pickle**: Highly flexible, fast performance. Only usable in Python.

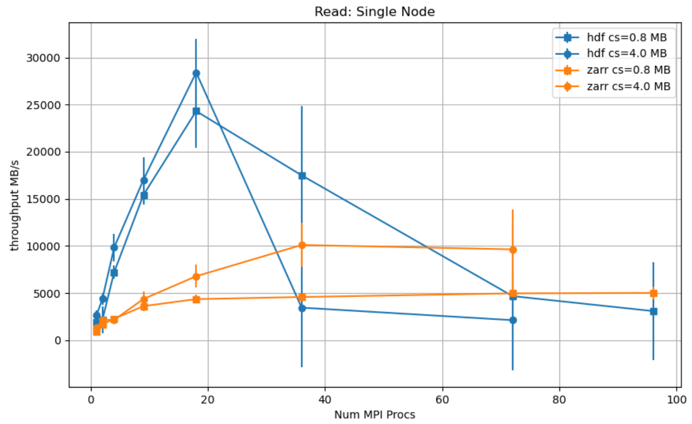
Performance Evaluation

- Research different toolsets and benchmark for specific use-case
  - tf.data API, Kubeflow AI framework, TensorStore, S3, Dask
- Establish best workflow, data format for use-case
- Use secure methods for sensitive data









Coding and Version Control

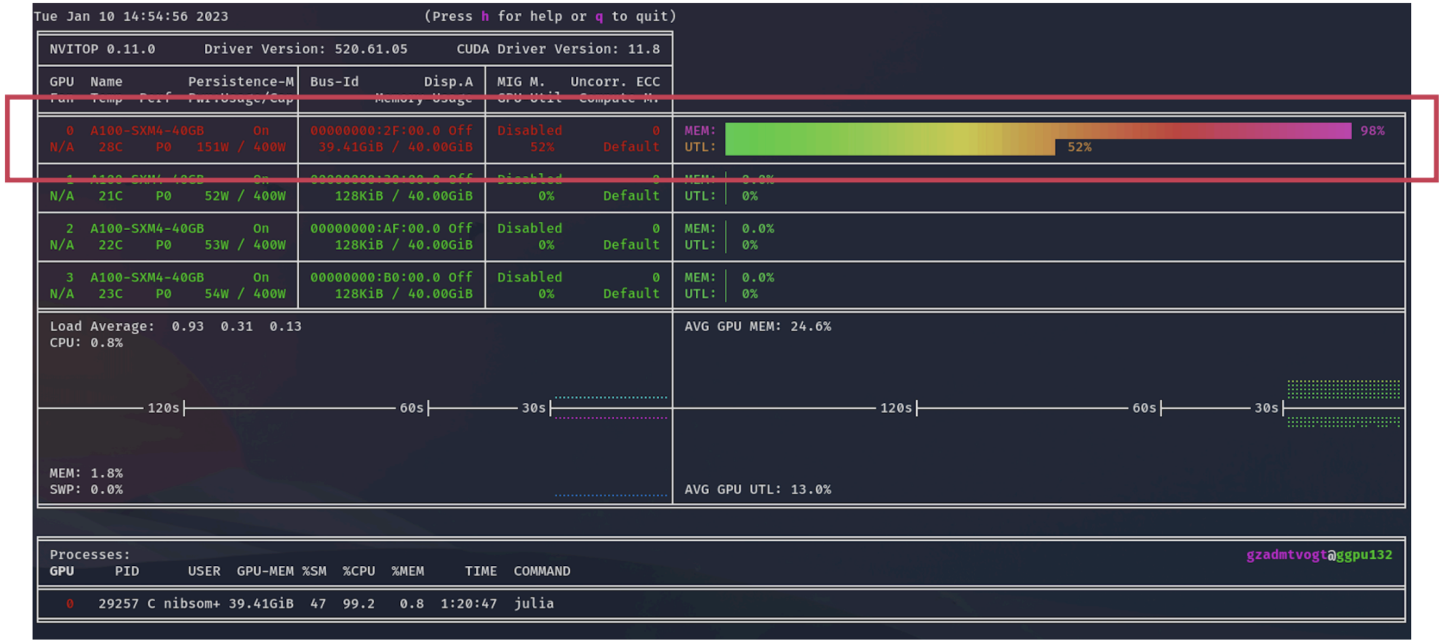
- Throughout development, many iterations are needed
  - Develop pipeline, test on smaller scale, explore various methods
- Test at each major step to prevent problems later in development
- Write comments and documentation as needed
  - Remove clutter, refactor code
- Older code can be very useful, all versions should remain accessible
- Git (Github/Gitlab) is the most well-known and popular VCS
- At later stages, use workflow system, e.g., Snakemake

Job Scheduling

- Login nodes are not designed for computation
- Jobs are submitted with SLURM
  - Interactive jobs for development and testing with `--pty`
- Parallel training on single node or multi-node configurations
  - Only use multi-node configuration when necessary
- RAM, GPUs, etc. must be configured according to model architecture
- Make sure GPU has enough VRAM, monitor usage

Monitoring

- Monitor GPU, CPU usage
  - Command: `nvitop`
  - CPU, MEM, GPU
- Identify bottlenecks
  - Storage I/O
  - Data preprocessing on CPU
  - GPU processing
  - Network speed



Example output from running `nvitop`.

Follow-up Material

- Explore hardware, technologies
- Use-case examples
  - Deep Learning with GPU course
    Follow GWDG on YouTube
- Libraries, tools, frameworks
  - From Github, papers, workshops

Summary and Discussion

- Variety of ML tools available for specific use-cases, worth exploring
- Workflows can differ but important to follow general best practices
- Data storage
  - Store data in format that is efficient for ML
  - Data storage must be chosen wisely based on trade-offs between speed, reliability and volume
- Identify and fix bottlenecks
  - Keep the balance between the modules for maximum efficiency