ScaDS.AI
DRESDEN LEIPZIG
CENTER FOR SCALABLE DATA ANALYTICS AND ARTIFICIAL INTELLIGENCE

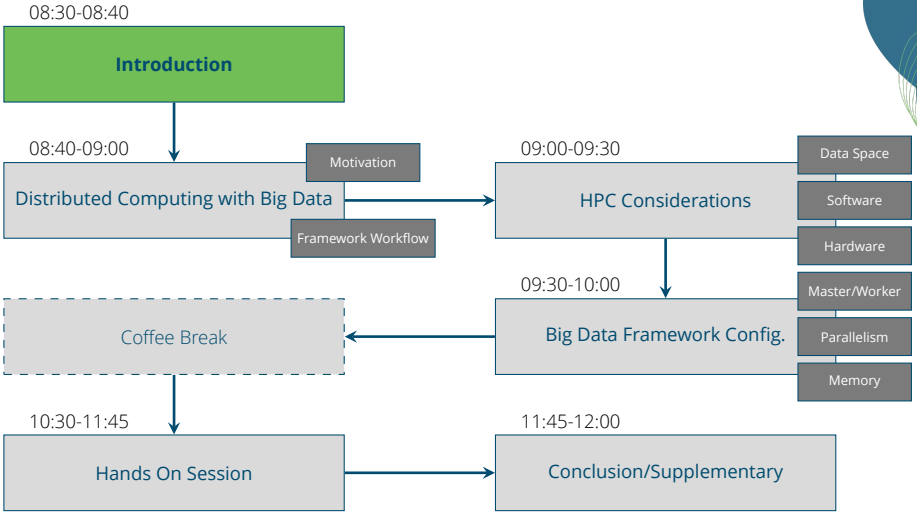
Big Data Processing on HPC

Siavash Giashvand, Apurv D. Kulkarni, Christoph Lehmann, Elias Werner

May 2025

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

SACHSEN
Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages.
Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.


08:30-08:40
Introduction
08:40-09:00
Distributed Computing with Big Data
Motivation
Framework Workflow
09:00-09:30
HPC Considerations
Data Space
Software
Hardware
09:30-10:00
Big Data Framework Config.
Master/Worker
Parallelism
Memory
Coffee Break
10:30-11:45
Hands On Session
11:45-12:00
Conclusion/Supplementary

Intention of the Training

- Making things understandable from a user's point of view
- Show interplay between Big Data frameworks and HPC environment
- Interactive sessions with toy examples

Note

- We do **not** want to show everything that is possible. We do **not** want to show the best optimized big data processing pipeline.
- We want to provide an overview of using a big data processing engine on an HPC machine.

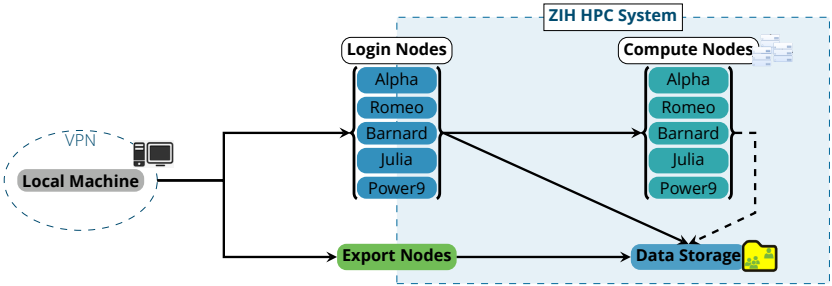
Recap: What is an HPC machine?

- Cluster of many single nodes (single "computers") for pooling their resources (computation and memory)
- Job scheduler Slurm manages resource/hardware allocation
- Software environment can be set up easily via a module system
- Collaborative working spirit, shared machine
- Access to HPC is given via a project

Technical workflow:

Hint

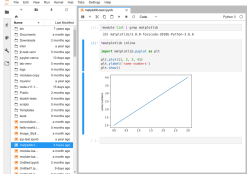
https://compendium.../application/overview/



Recap: Access the ZIH HPC System

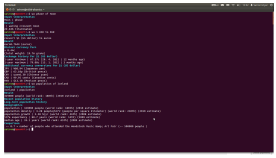
JupyterHub

- browser based approach
- easiest way for beginners



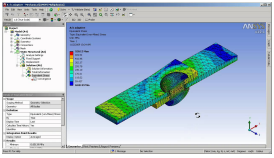
SSH connection (CLI)

- "classical" approach
- command line interface (CLI) knowledge is necessary



Desktop visualization

- esp. in the case of using GUI-based software
- e.g. Ansys, Vampir,...



Note

- When working from **outside of the university network**, the ZIH HPC system can be accessed **only via Virtual Private Network (VPN)**
- Please use **provided** login information for this training

Hint

More info on VPN:

HPC system

- **Facility**: Center for Information Services and High Performance Computing (ZIH)
- **Details**:
  - More than 60,000 cores,
  - 720 GPUs,
  - Flexible storage hierarchy with about 16 PB total capacity,
  - Linux, shared login nodes, filesystems, batchsystem Slurm,
  - Perfect platform for highly scalable, data-intensive and compute-intensive applications.

HPC and Big Data Processing – Why it Fits

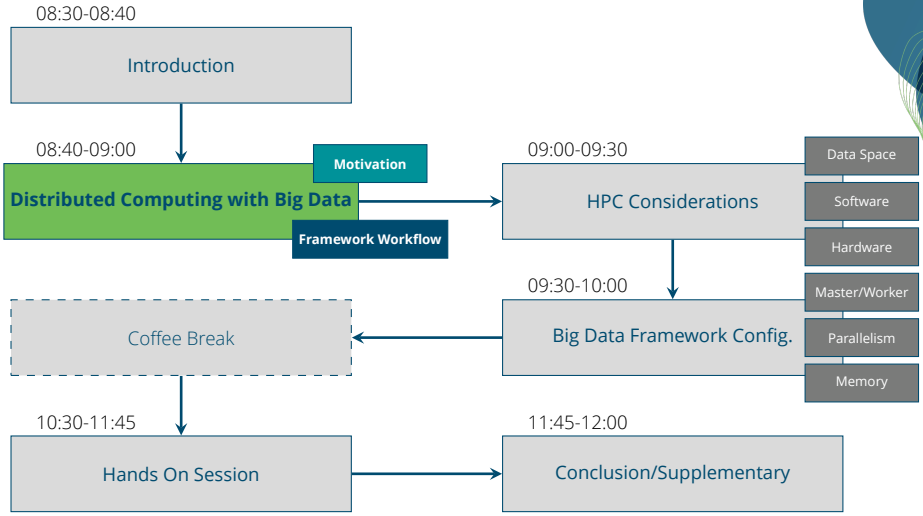
Big Data Processing Engines:

- Software designed to make scaling of big data processing easy across node boundaries
- Handle a huge amount of available and distributed compute resources
- Distribute work across multiple workers

HPC clusters:

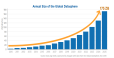
- Make it easy to get powerful compute hardware
- Use many individual machines (nodes) to provide compute power
- Are specialized for very fast inter-node communication and I/O

Big data processing engines unleash the potentially sheer infinite scalability of HPC clusters.

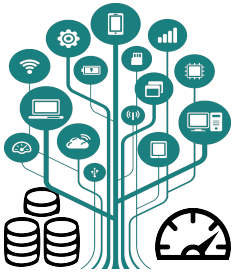

08:30-08:40
Introduction
08:40-09:00
Distributed Computing with Big Data
Motivation
Framework Workflow
09:00-09:30
HPC Considerations
Data Space
Software
Hardware
09:30-10:00
Big Data Framework Config.
Master/Worker
Parallelism
Memory
Coffee Break
10:30-11:45
Hands On Session
11:45-12:00
Conclusion/Supplementary

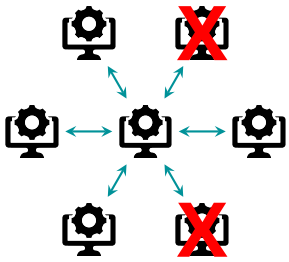
Why Distributed Computing?

- V's of Big Data
  - Volume
  - Velocity
  - Variety
- Distributed computing on HPC
  - Large number of compute nodes
    - 1000+ compute nodes with memory upto 512GB/node
  - Powerful multicore CPUs
    - AMD EPYC, Intel(R) Xeon(R) Platinum
  - Fast network interconnect
    - Infiniband EDR links - 100 Gbit/s
  - Location independent access
  - Ease of access using tools like Jupyterhub/Jupyter notebook
- Robust and resilient against failure

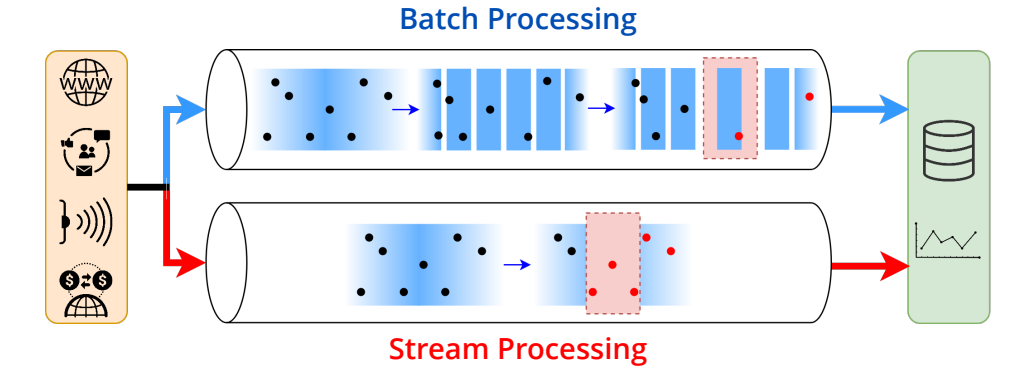


source

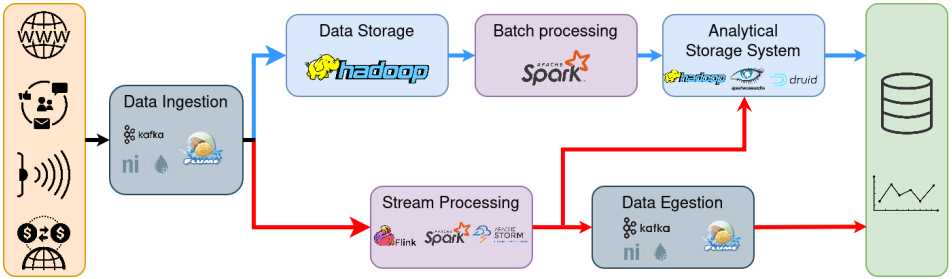




Processing Patterns



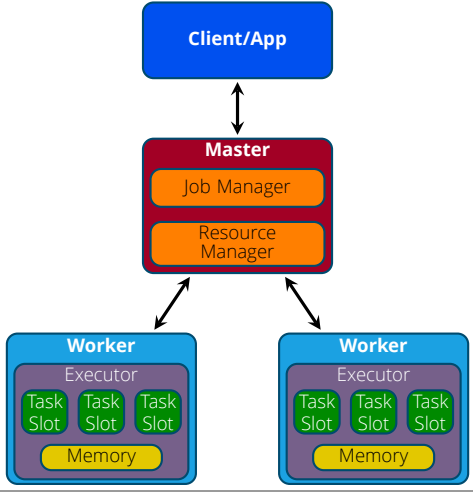
Processing Patterns



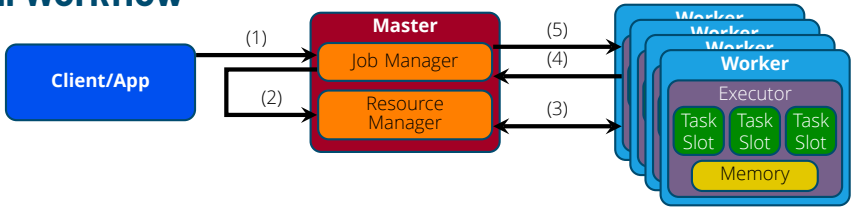
Processing Framework Architecture

Components of a typical Big Data Framework:

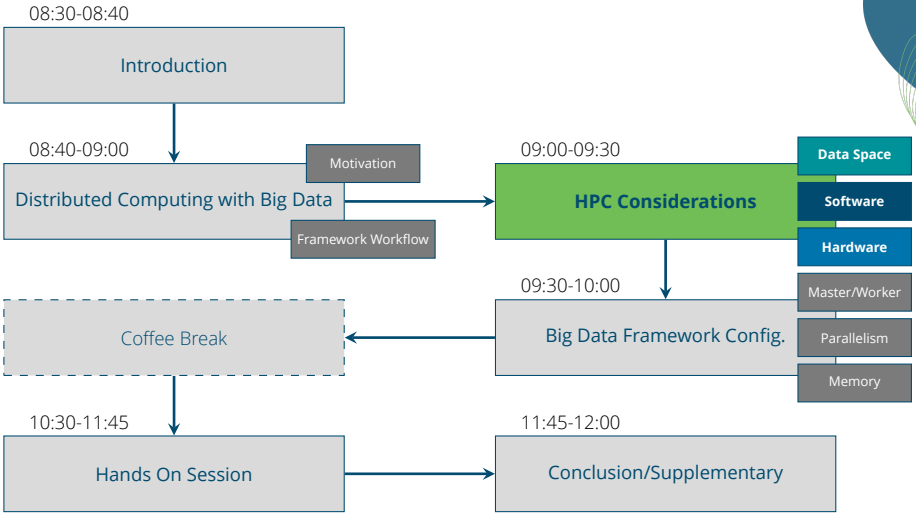
- **Client**: Contains the code and submits it to the cluster. Entrypoint of any big data process.
- **Master**:
  - **Job Manager**: Controls the application execution.
  - **Resource Manager**: Requests and manages taskmanager slots.
- **Worker**:
  - **Executors/Taskmanagers**: These are worker processes consisting of a certain number of task slots.
  - **Task Slot**: Each task slot contains a single executable task. A task is a part of the application.



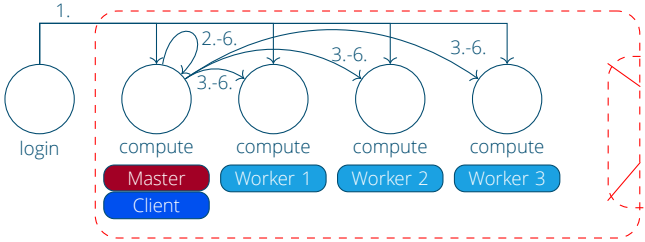
General Workflow

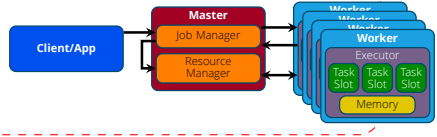


1. A job is submitted by the driver to the job manager.
2. The job manager requests the resources from the resource manager.
3. The resource manager allocates the resources of executors.
4. Once the resources are available, the offered Taskslots can be accessed by the job manager.
5. The job manager deploys the tasks to the executors present on the worker nodes.


08:30-08:40
Introduction
08:40-09:00
Distributed Computing with Big Data
Motivation
Framework Workflow
09:00-09:30
HPC Considerations
Data Space
Software
Hardware
09:30-10:00
Big Data Framework Config.
Master/Worker
Parallelism
Memory
Coffee Break
10:30-11:45
Hands On Session
11:45-12:00
Conclusion/Supplementary

Workflow for working with Big Data Frameworks on HPC





1. Allocate resources from Slurm
2. Load respective software modules
3. Configure framework on the allocated resources
4. Start cluster (master, worker daemons, executors, etc.)
5. Submit processing pipeline
6. Tear down the cluster after the pipeline finishes

Data Management on HPC System

- Various data storage systems differ in capacity, bandwidth, IOPS rate, etc.
- Organization of these storage systems in **files systems**
- Many file systems, users, data require mechanisms for data management (duration, size, permission)
- The concept of workspaces is used on the ZIH system for data management

Hint

For this presentation: a file system refers to a "place" to store data and a **workspace** refers to the "access" to that place.

Filesystem Storage Overview

- Each filesystem serves a respective capacity, performance, duration, lifetime.
- User creates a workspace with defined expiration date.
- A workspace can be extended multiple times, depending on the filesystem.
- Following is the overview of the filesystem available at ZIH:

Filesystem	Scope	Type	Mount Point	Size	Duration (extensions)
Temporary	Local	Ext4	`/tmp` or `/dev/shm`	95 GB	Job Duration
horse	Global temporary	Lustre	`/data/horse`	20 PB	100 days (x9)
walrus	Global temporary	Lustre	`/data/walrus`	20 PB	365 days (x2)
weasel	Global temporary	WEKAio	`/data/weasel`	1 PB	-
User Space	Global permanent	Lustre	`/home/<user-id>`	50 GB	unlimited
Project Space	Global permanent	NFS	`/project/<project-id>`	on demand	unlimited
Archive	Inter./ Long term	Tapes	`/data/archiv`	on demand	3 years

Workspaces

- On the ZIH system, data has a limited lifetime depending on the filesystem;
- Users create a workspace on filesystems with defined expiration date;
- Data is deleted automatically after expiration.

Some commands to manage workspaces on the ZIH system:

Command	Description
`ws_find --list`	Find available workspace filesystems
`ws_allocate -F <filesystem> <name> <duration>`	Allocate workspace
`ws_list`	List your workspaces and get information
`ws_extend -F <filesystem> <name> <duration>`	Extend workspace

Hint

https://compendium.../data_lifecycle/workspaces/

Data Transfer: To/From HPC System

- Using `scp`, `rsync` and `sftp` at:
  - `dataport1.hpc.tu-dresden.de` or,
  - `dataport2.hpc.tu-dresden.de`

Example

```
# Copy a file from your workstation to ZIH systems
marie@local$ scp <file> dataport1.hpc.tu-dresden.de:<target-location>
# Add –r to copy the whole directory
marie@local$ scp -r <directory> dataport1.hpc.tu-dresden.de:<target-location>
# Copy a file from ZIH systems to your workstation
marie@local$ scp dataport1.hpc.tu-dresden.de:<file> <target-location>
```

Hint

https://compendium.../data_transfer/dataport_nodes/

Data Transfer: Within HPC System

- Linux commands like `cp` and `mv` can be used transfer small data

Example

```
# Copy the training data into horse workspace directory
marie@login$ cp <my-file> /data/horse/ws/marie-myworkspace/
```

- Datamovers are used to move data faster within the ZIH HPC system.
  - For using such datamover machines, you have to use commands prefixed with **dt**: `dtcp`, `dtwget`, `dtmv`, `dtrm`, `dtrsync`, `dttar`, `dtls`
  - `dtqueue --me` to check the status of the data transfer
  - These commands create a Slurm job with dedicated resources to conduct the data handling/transfer.

Example

```
# dtcp for copying data from one filesystem/location to another filesystem/location
marie@login$ dtcp -r <fs\_origin>/results <fs\_destination>/

# dtmv for moving data from one filesystem/location to another filesystem/location
marie@login$ dtmv <fs\_origin>/ws/results <fs\_destination>/ws/ws-archive/

# dttar for archive data "results.gz" from one filesystem/location to another filesystem/location
marie@login$ dttar -czf<fs\_destination>/results.tgz <fs\_origin>/ws/results
```

Compute Hardware at ZIH

ZIH HPC System consists of heterogeneous compute resources:

- CPU-based (of different manufactures)
- GPU-based (of different manufactures)
- Large shared memory nodes

Name	System	Nodes	Cores/Node	Threads/Core	Memory/Node [in MB]	GPUs/Node
Barnard	CPU	630	104	2	515,000	0
Romeo	CPU	192	128	2	505,000	0
Capella	GPU	144	64	1	768,000	-
Alpha	GPU	37	48	2	990,000	8
Power	GPU	30	44	4	254,000	6
Julia	SMP system	1	896	1	48,390,000	-

Hint

https://compendium.../jobs_and_resources/hardware_overview/

Running Programs on an HPC Machine

- user-defined calculations, programs etc. are not run directly and interactively on an HPC system (as commonly on a personal workstation or laptop)
- running some program on an HPC machine means running it within a temporary resource allocation
- the definition of a program and a resource allocation is a **job**

On the ZIH system: job scheduling and workload management with **slurm**

- manages jobs and provides an interface for the users to submit their jobs
- evaluates resource requirements and priorities, distributes jobs to suitable compute nodes –> "job queue"

Note

Most crucial issue esp. for HPC beginners: You need to specify in advance compute, memory, and time resources according to your program's needs. We will refer to this later.

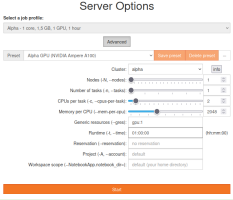
More details about job schedulers can be found at: `https://hpc-wiki.info/hpc/Scheduling_Basics`

Job Scheduler Basics: Running Jobs

JupyterHub

Login to JupyterHub

Set parameters and load the required modules



Running the application either using **Console** or **Jupyter Notebook**

Terminal Notebook

Interactive

Login to ZIH HPC System login-node using console

Using `srun` to alocate resources and attach shell using `--pty bash -l`

```
marie@login$ srun <params>
```

Load the required modules

```
marie@compute$ module load <module>
```

Run the application

```
marie@compute$ python my_script.py
```

Batch

Login to ZIH HPC System login-node using console

Create a sbatch script

```
#!/bin/bash
#SBATCH --partition=alpha   # Use alpha partition
#SBATCH --nodes=1           # Use one node
#SBATCH --cpus-per-task=2   # Use 2 threads per task
#SBATCH --mem=8000          # Use 8gb of RAM
#SBATCH --gres=gpu:1        # Use 1 GPU per node
#SBATCH --time=00:05:00     # five minutes should be enough

# Loading required modules
module load modenv/hiera CUDA/11.1.1 OpenMPI/4.0.5 PyTorch/1.9.0

# Run the application
python my_script.py >> output
```

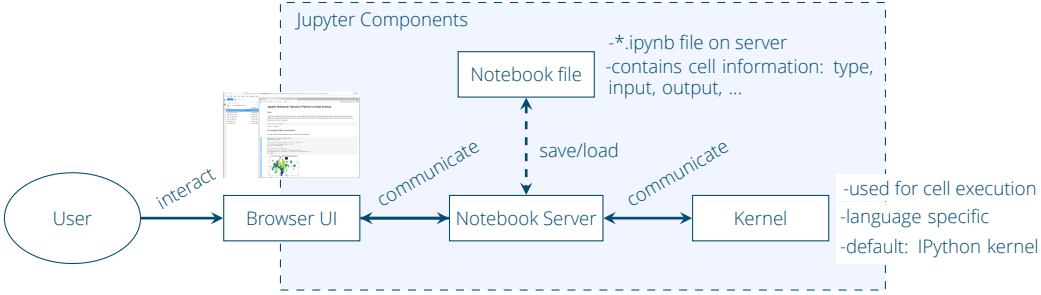
Run the application by submitting the sbatch script usng `sbatch` command

```
marie@login$ sbatch my_sbatch_script.sbatch
```

Note

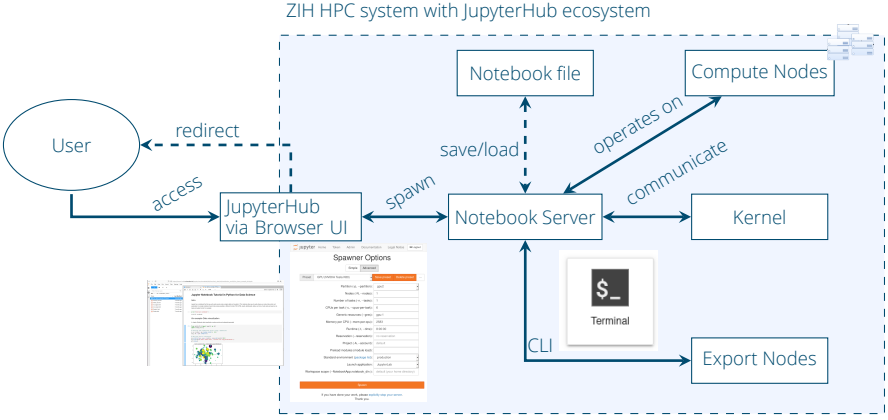
JupyterHub job is inherently a batch job. Since it is a little different approach than batch job submission using CLI, it is shown here as a separate method.

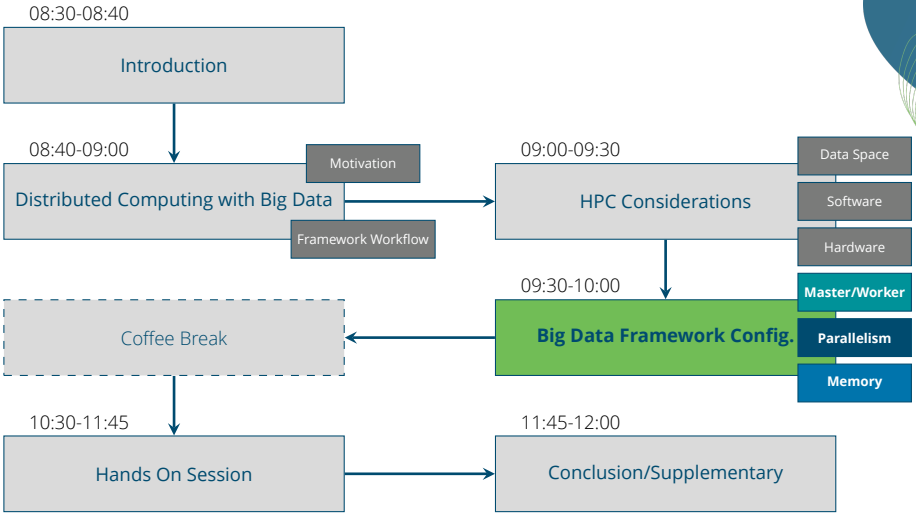
Jupyter Overview – Behind the Scenes



structure according to: jupyter.readthedocs.io/.../content-architecture

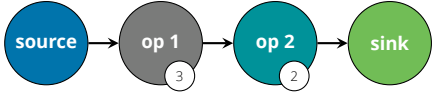
Access the ZIH System via JupyterHub



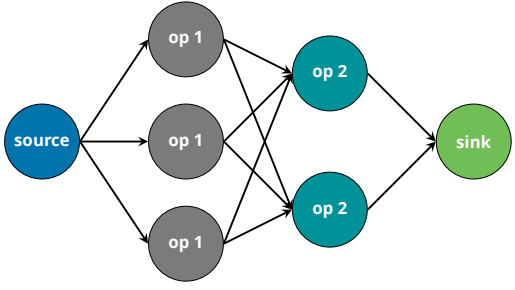

08:30-08:40
Introduction
08:40-09:00
Distributed Computing with Big Data
Motivation
Framework Workflow
09:00-09:30
HPC Considerations
Data Space
Software
Hardware
09:30-10:00
Big Data Framework Config.
Master/Worker
Parallelism
Memory
Coffee Break
10:30-11:45
Hands On Session
11:45-12:00
Conclusion/Supplementary

Parallelism: Job Graphs

- Any big data processing job can be represented using a job graph (also known as a data flow graph), which shows how data flows.
- **Logical job graph** depicts a high-level overview of a job. A node in a logical job graph represents the processes or operations, while edges show where the data flows between these operators.
- **Physical job graph**, which is derived from the logical job graph, shows how the processes are actually implemented on a system. It considers parallelism, where each node represents a task.

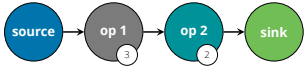


Logical job graph



Physical job graph

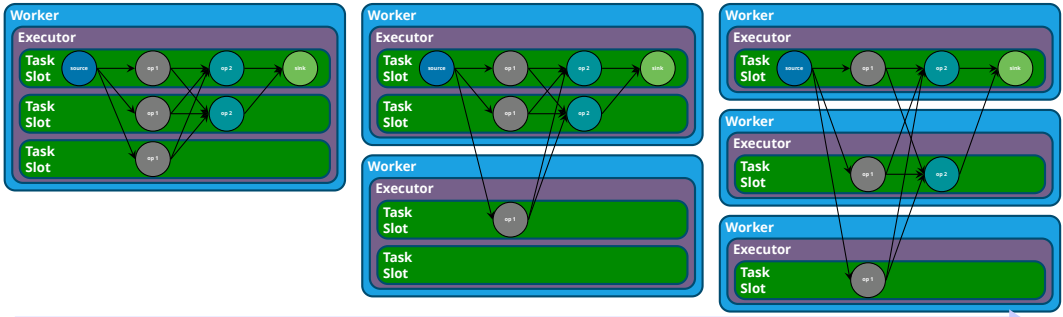
Parallelism: Job Graphs (cont..)



1 Worker, 1 Executor/Worker, 3 Task-Slots/Executor

2 Workers, 1 Executor/Worker, 2 Task-Slots/Executor

3 Workers, 1 Executor/Worker, 1 Task-Slots/Executor



Parallelism

Apache Flink (v1.15.0)

- Configuring task slots per taskmanager:
  - $\#Slots/Tskmgr = \frac{\#CPU/Node}{\#Tskmgr/Node} = \#CPU/Node$
  - **flink-conf.yaml**

```
taskmanager.numberOfTaskSlots: <INTEGER>
```

- Configuring default parallelism:
  - $Parallelism_{max} = \#Slots/Tskmgr \times \#Nodes = \#CPU/Node \times \#Nodes$
  - **flink-conf.yaml**: Default=1

```
parallelism.default: <INTEGER>
```

Apache Spark (v3.3.0)

- Configuring executor cores:
  - $\#Cores/Executor = \#CPUs/\#Node$
  - **spark-env.sh**

```
export SPARK_WORKERS_CORES=<INTEGER>
export SPARK_EXECUTOR_CORES=<INTEGER>
```

- Configuring default parallelism:
  - $Parallelism_{max} = \#Cores/Executor \times \#Nodes$
  - **spark-default.conf**: Default = total number of cores allocated to all executors.

```
spark.default.parallelism
```

Tskmgr=Taskmanager

Aspects of Parallel Programming

- **Applicability:** Not all methods/algorithms are parallelizable, i.e. to be divided into independent sub tasks.
  - not possible: sequential algorithms (e.g. stochastic gradient descent for machine learning training)
- **Load Balancing and Data Partitioning:** Split data into chunks of comparable size that can be processed in parallel.
- **Communication/Aggregation Overhead:** Running something in parallel needs communication and aggregation of results. Choose a reasonable level of parallelism, but not the max level.

Note that parallel programming is a vast field, and in this workshop, we are only scratching the surface.

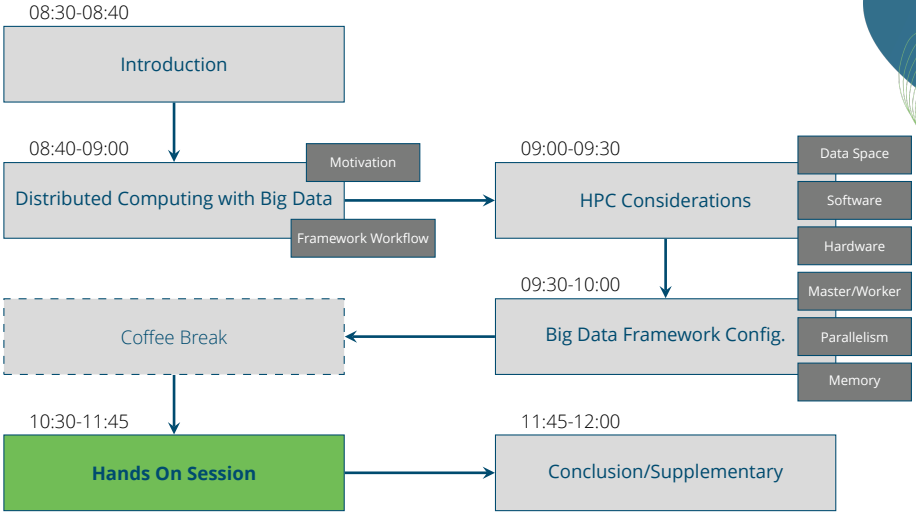
Crucial Aspects When Using Spark

1. **Lazy Evaluation**
   - Transformations like `filter()`, `select()`, are lazy and no execution until an action (e.g., `count()`, `show()`) is triggered.
   - See docs for transformations vs. actions:
     `https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations`
2. **Shuffling & Partitioning**
   - Spark works with distributed data (RDD = resilient distributed dataset)
   - Shuffling is the process of (re-)distributing data chunks among the cluster (caused e.g. by `groupBy`, `join`, `repartition`).
   - Repartition only when needed; use `coalesce()` to reduce partitions without full shuffle.
   - Spark controls the number of partitions internally, but this also can be set by user

Hint

Spark has a lot of configurable parameters that need to be considered within development of a concrete application.

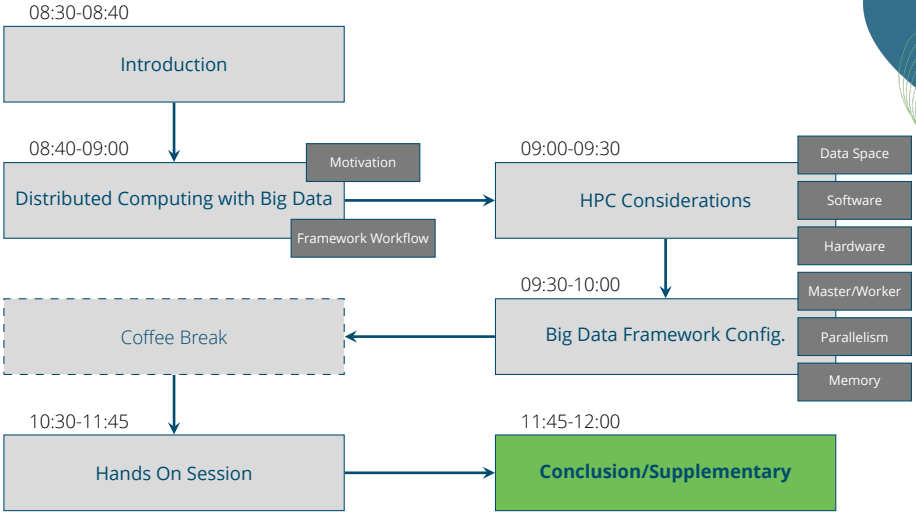
Big Data Processing on HPC - Demo


08:30-08:40
Introduction
08:40-09:00
Distributed Computing with Big Data
Motivation
Framework Workflow
09:00-09:30
HPC Considerations
Data Space
Software
Hardware
09:30-10:00
Big Data Framework Config.
Master/Worker
Parallelism
Memory
Coffee Break
10:30-11:45
Hands On Session
11:45-12:00
Conclusion/Supplementary

Hands-on session: Overview

Assume, we want to analyse the New York City taxi dataset. Here are some information:

- New York City taxi dataset
  - contains taxi trips within NYC with many details:
    - Start/end times
    - Locations
    - Cost
    - Number of persons
    - ...
  - Original source: `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`
  - Exercise is stored at `/data/horse/ws/s4122485-ai4seismology/big_data` .
- Analysis aims:
  - Use Jupyter as frontend with a small Apache Spark cluster as backend.
  - Follow the tasks in the Jupyter notebook.
  - (for details, see next slide)


08:30-08:40
Introduction
08:40-09:00
Distributed Computing with Big Data
Motivation
Framework Workflow
09:00-09:30
HPC Considerations
Data Space
Software
Hardware
09:30-10:00
Big Data Framework Config.
Master/Worker
Parallelism
Memory
Coffee Break
10:30-11:45
Hands On Session
11:45-12:00
Conclusion/Supplementary

Conclusions

- Big Data Processing works on HPC, but is more complex (no fixed resources, but SLURM scheduling)
- HPC and Big Data Processing are also complex individual tasks, that need time to learn about
- Leveraging HPC infrastructure by Big Data frameworks can lead to efficient distribution of data intensive tasks
- For effective deployment of Big Data frameworks in HPC, HPC principles need to be considered
- On ZIH HPC system:
  - Apache Flink and Apache Spark are available via the module system
  - A useful basic configuration of Big Data frameworks is provided via `BigDataFrameworkConfigure` module and `framework-configure.sh` script
  - JupyterHub can leverage Big Data clusters and compute visual outputs (e.g. maps)

Conclusions

- In case of questions start at the official docs for the ZIH system: https://compendium.hpc.tu-dresden.de/
- Software ecosystems are changing dynamically, esp. in Big Data and ML: with HPC systems it is hard to react immediately
- Contact us with your own ideas, experiences and wishes!

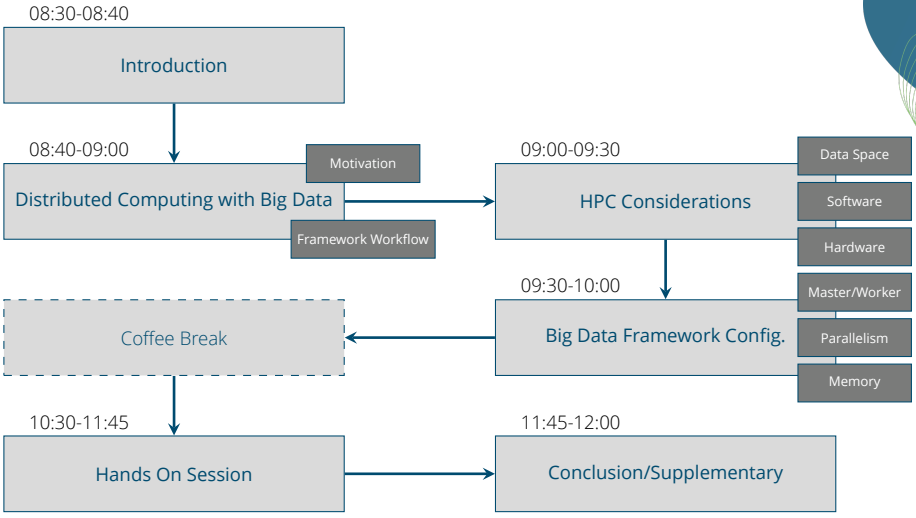
Note

- Logins (scadsxxx) will work for further 10 days
- If your are interested afterwards please apply for your own HPC project. More information can be found at: https://compendium.../application/overview/

ScaDS.AI DRESDEN LEIPZIG

Thank you

Contact us at:
transferscads.ai@tu-dresden.de


08:30-08:40
Introduction
08:40-09:00
Distributed Computing with Big Data
Motivation
Framework Workflow
09:00-09:30
HPC Considerations
Data Space
Software
Hardware
09:30-10:00
Big Data Framework Config.
Master/Worker
Parallelism
Memory
Coffee Break
10:30-11:45
Hands On Session
11:45-12:00
Conclusion/Supplementary

Flink Taskmanager Memory Model

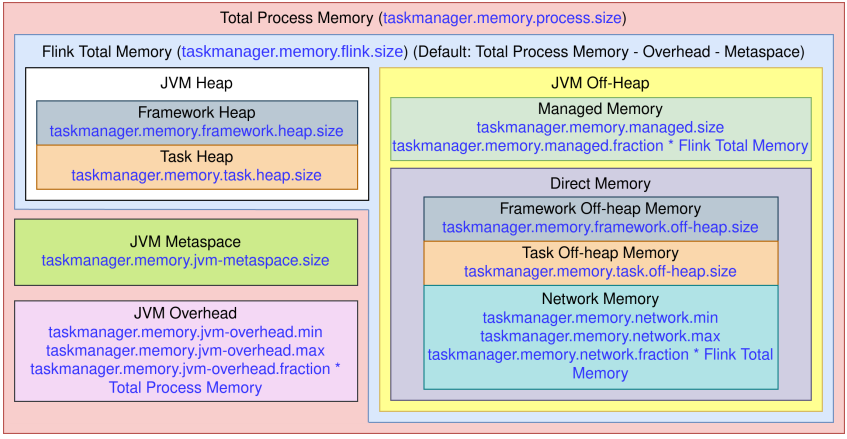


Figure 1: Executor memory model

Flink Taskmanager Memory Model (cont..)

- Framework Heap/Off-Heap Memory: Memory dedicated to flink framework and its internal data structures
- Task Heap/Off-Heap Memory: Used to run tasks (operators) and user code.
- Managed Memory: Used for RocksDB state backend, sorting, hashtable, caching of tintermediate results.
- Network Memory: Used for data exchange between tasks and acts as network buffer.
- JVM Metaspace: Used for storing metadata for classes.
- JVM Overhead: Used for JVM overheads like code cache, GC collection space, etc.

Spark Executor Memory Model

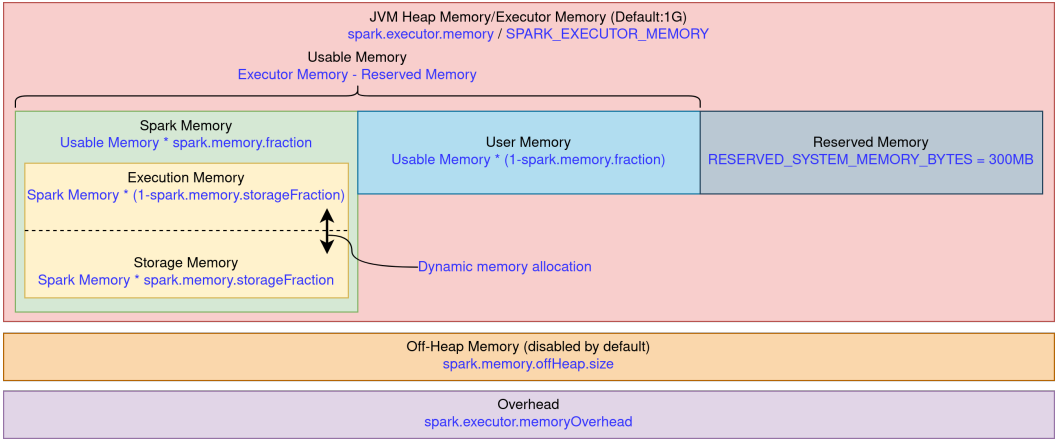


Figure 2: Executor memory model

Spark Executor Memory Model (cont..)

- Executor Memory: Maximum JVM memory where objects are also bounded by garbage collector
- Execution Memory: Used for storing temporary data created during execution of the task like shuffle, join, sort, aggregation etc.
- Storage Memory: Used for storing cached and broadcasting data.
- User Memory: Used for storing UDFs and user defined data structures.
- Reserved Memory: Hard-coded memory used for storing spark internal objects.
- Off-Heap memory: Additional memory that accounts for operations like VM overheads, interned strings, other native overheads, etc.

Performance Analysis – Job Monitoring with PIKA

- Simple approach for overview and to check used resources for a job: PIKA (hardware performance monitoring stack)
- monitoring during and after runtime: access PIKA web interface at https://selfservice.zih.tu-dresden.de/l/index.php/hpcportal/jobmonitoring/z../jobs_and_resources
- Choose between “Live” (for running jobs) and “Jobs” (finished jobs with filtering by date/time if needed)

Hint

More info about PIKA at:https://compendium.../software/pika/

Workflow: Using git

Git is a version-control system, commonly used for managing source code and coordinating the work of multiple contributors on a software project.

Please consider the following basics while using git on the ZIH system:

- git is available on all cluster nodes
- can be used on HPC in the same way as in the shell of your local machine, e. g. `git clone`, `git add`, `git commit`, `git push` etc.
- every project collaborator is working on his own copy and changes are pushed to the remote repository
- no binary files are contained in the repo (this keeps the storage requirement low)
- cloned repo copies are located in user space

Note

Using the same repo copy for different users will cause permission and access issues - DON'T DO THAT.