Scheduling in-situ analysis tasks attached to HPC simulations

Ana Gainaru,
Guillaume Pallez, Scott Klasky

18th workshop on Scheduling for large-scale systems
Montréal, Québec, Canada, July 8-10, 2025

ORNL is managed by UT-Battelle LLC for the US Department of Energy

gainarua@ornl.gov

Why do we need scheduling for in-situ tasks?

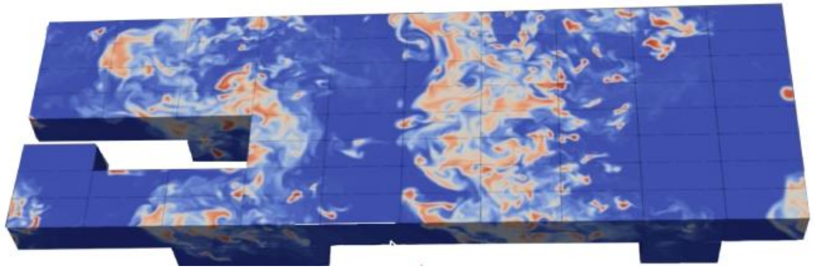
- Current HPC simulations generate up to PB data/step
  - Often **requiring post-processing** tasks in real time
    - QoI computation, compression, data transformation, pre-processing, check correctness, identify regions of interest
  - Could be done in-situ or on **dedicated cores**

- Pre-processing tasks executed every simulation step
  - **Time/resource constraints**
  - Some tasks are more important than others

This talk: Priority based scheduling with resource constraints

Examples of data processing tasks

- Post-processing data to **identify features**
  - E3sm (climate) data to identify the trajectory of tornadoes and refactor
  - QIUP (medical) data to identify cancerous cells
- **Post-processing data for training**
  - FASTRAN (fusion) data to identify regions in the training space where data is missing
- **Remote visualization**
  - S3D (combustion) data to visualize temperature in regions of interest
- **Surrogate model** execution
  - GE (aerospace) to predict the trajectory of the simulatio
- **Correctness checks**
  - GE (aerospace) data to audit properties of the data
- **Post-mortem** visualization and analysis
  - For non-critical tasks that will help scientists after the simulation is done

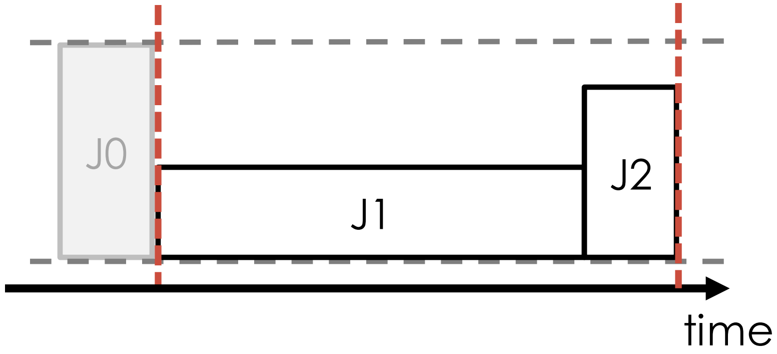


OAK RIDGE
National Laboratory

Current solutions

- **Our problem:** execute as many high priority tasks as possible
  - Input: set of tasks that need to be executed each simulation step
  - There is not enough space/time to execute all of them
  - Some tasks are critical, some are optional

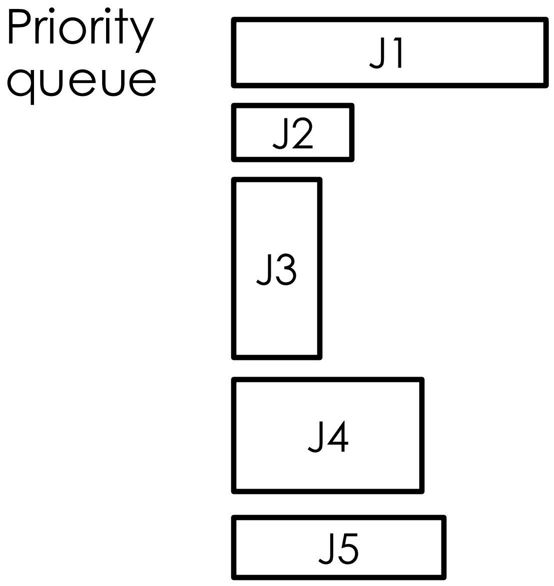
- **Schedulers in HPC:** Easy-BF
  - Jobs are ordered based on some priority criteria
    - FCFS, LJF, SJF
  - Backfilling based the queue order
    - And what job can start earliest
  - Conservative-BF as an alternative
    - Backfill with jobs in the order of their submission



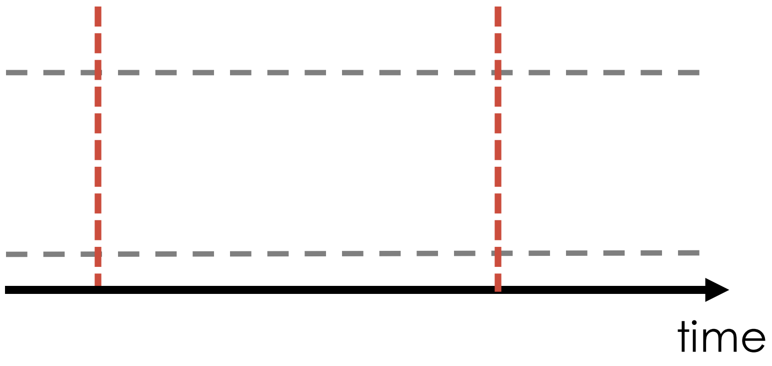
J0 finished, J1 and J2 are scheduled

- J1 starts running
- J2 is guaranteed a start after J1
- All other jobs are mutable
- Available area is between red lines

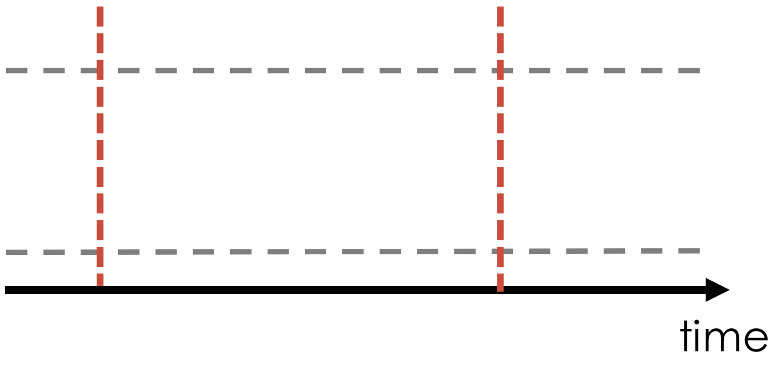
Example of limitation



- Limited time and resources to perform as many jobs as possible
  - Example one simulation loop (red lines)
  - Allocate external nodes
- Assuming we can set job priorities
  - J4 higher priority than J5



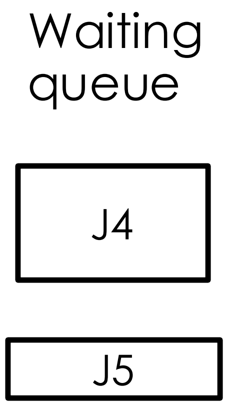
Easy-BF



Conservative-BF

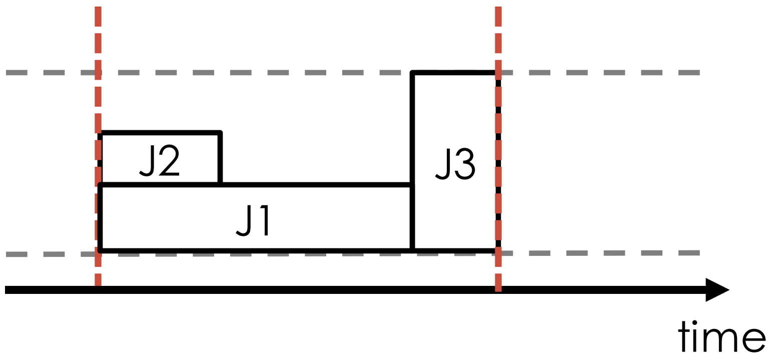
Example

- Both schedulers
  - J1 and J2 are guaranteed to start
  - J3 is guaranteed not to start later than where is scheduled
  - Everything else is mutable
- If J4 has a high priority than J5
  - Conservative-BF is preferable
- If J4 has a lower priority than J4
  - Easy-BF is preferable

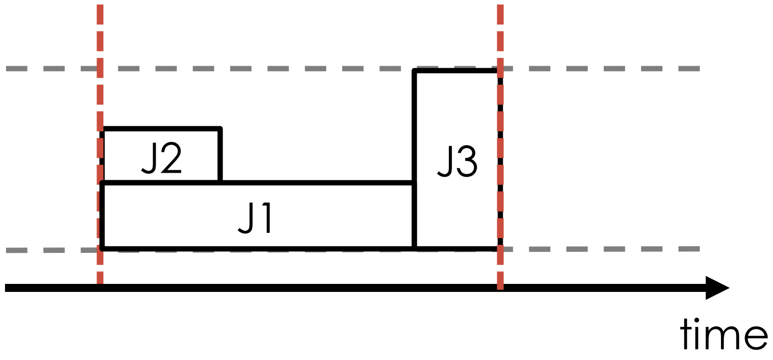

Waiting queue

J4

J5



Easy-BF



Conservative-BF

Our proposal for scheduling algorithm

- Philosophy
  - **Simplicity**
    - System administrators understand the rationale behind scheduling decisions
  - **Robustness**
    - Accommodate diverse workloads
  - Rely on qualitative constraints rather than rigid specifications

- Incorporate job importance
  - At the granularity of the job (set by users)
  - When all jobs share the same priority our algorithm reverts to Easy-BF

Our proposal for scheduling algorithm

- Main idea
  - Use several priority queues
  - Within a queue, jobs are scheduled with an **EASY-BF** strategy
  - Between queues, jobs are scheduled **conservatively**
    - Jobs from a queue with a higher index cannot delay jobs with a lower index
  - Minimize response times for high-priority jobs

How do we get scientists to set task priorities?

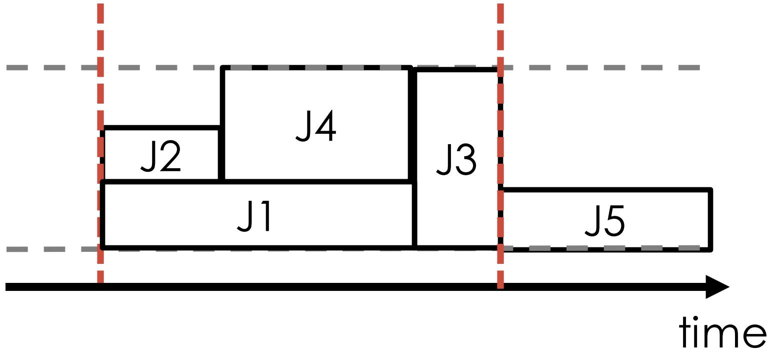
Our proposal for scheduling algorithm

- Main idea
  - Use several priority queues
  - Within a queue, jobs are scheduled with an EASY-BF strategy
  - Between queues, jobs are scheduled conservatively
    - Jobs from a queue with a higher index cannot delay jobs with a lower index
  - Minimize response times for high-priority jobs

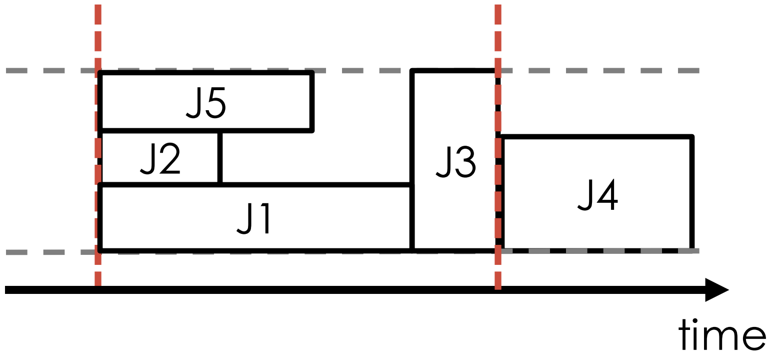
- **How to design priorities?**
  - Value-based (priority classes: high, low, medium)
    - E.g. *pre-processing for training, compression are high priority, QoI are low*
  - Frequency-based (run job X at least every T steps)
    - E.g. *compression is needed every step, QoI for visualization every 10 steps*

Priority-BF with our example

High priority: J1, J2, J3, **J4**
Low priority: **J5**



High priority: J1, J2, J3, **J5**
Low priority: **J4**



- **Strategies**
  - Jobs that did not finish by the end of the time window
    - Kill all jobs (fresh start), keep all jobs that started, keep only high priority jobs
  - Memory-less scheduling
    - Each loop uses the same queue (J5/J4 will starve) or updated queue

Evaluation

- Using ScheduleFlow simulator (for now)
  - Simple to use and to add new algorithms
  - For now we don’t need system characteristics
  - ***BatSim or WRENCH in the future***

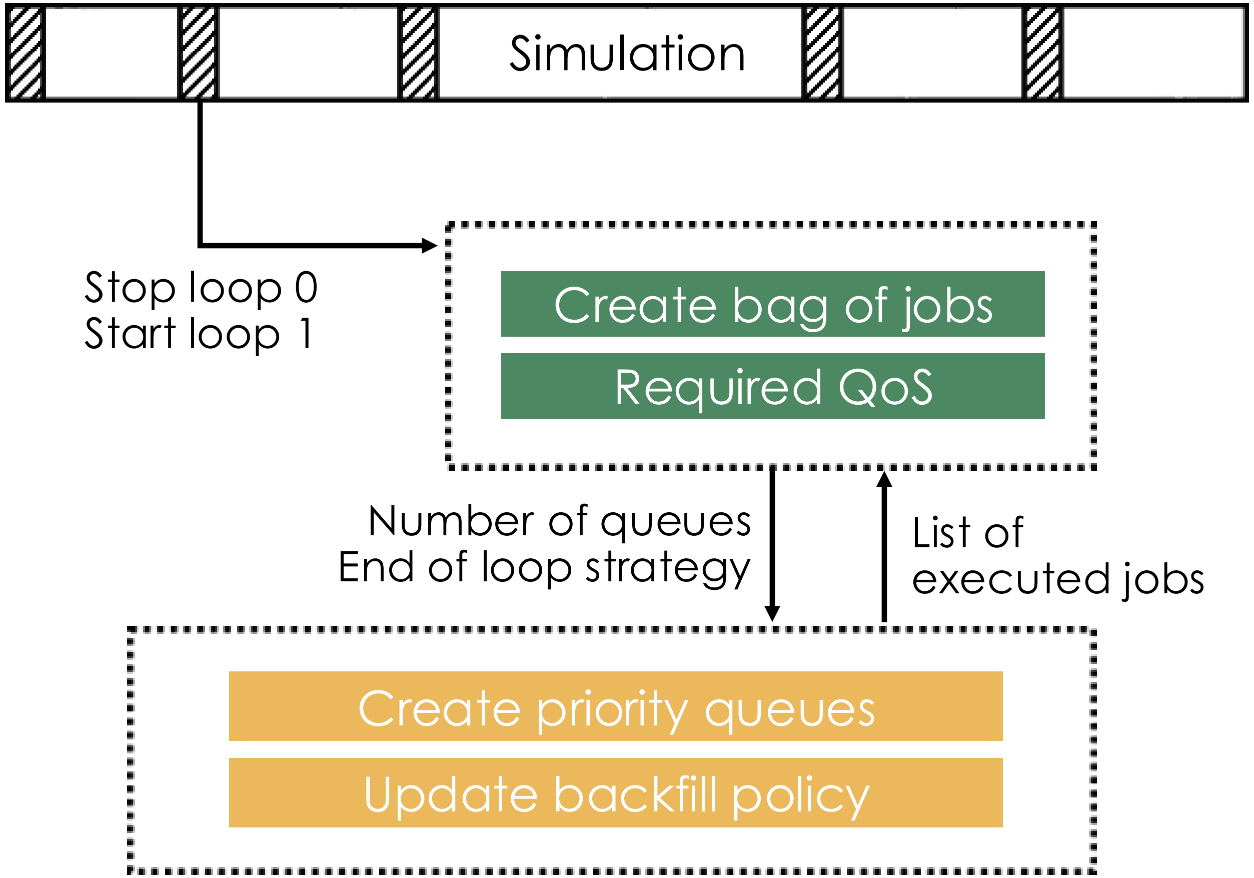
ScheduleFlow
SIMULATOR

- Experiments
  - Priority-BF compared to Easy-BF and Conservative-BF
    - Ordered using the same priorities
    - Simulated on ScheduleFlow with multiple queues
  - Neuroscience applications
    - Highly stochastic
    - Random priorities using values or QoS frequency

Metrics

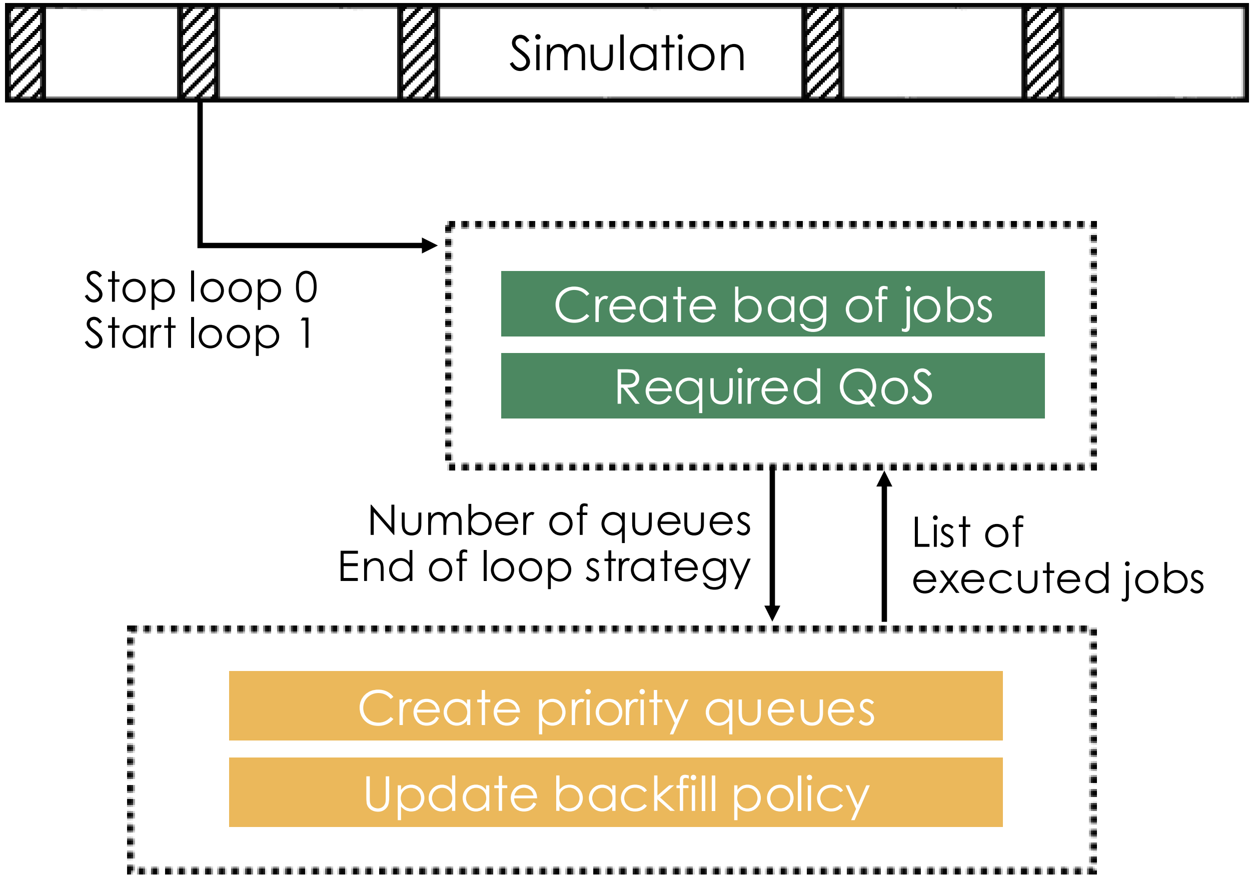
1. Average job runs in one loop
2. Number of misses
3. Response time for each job priority

Algorithms and implementation



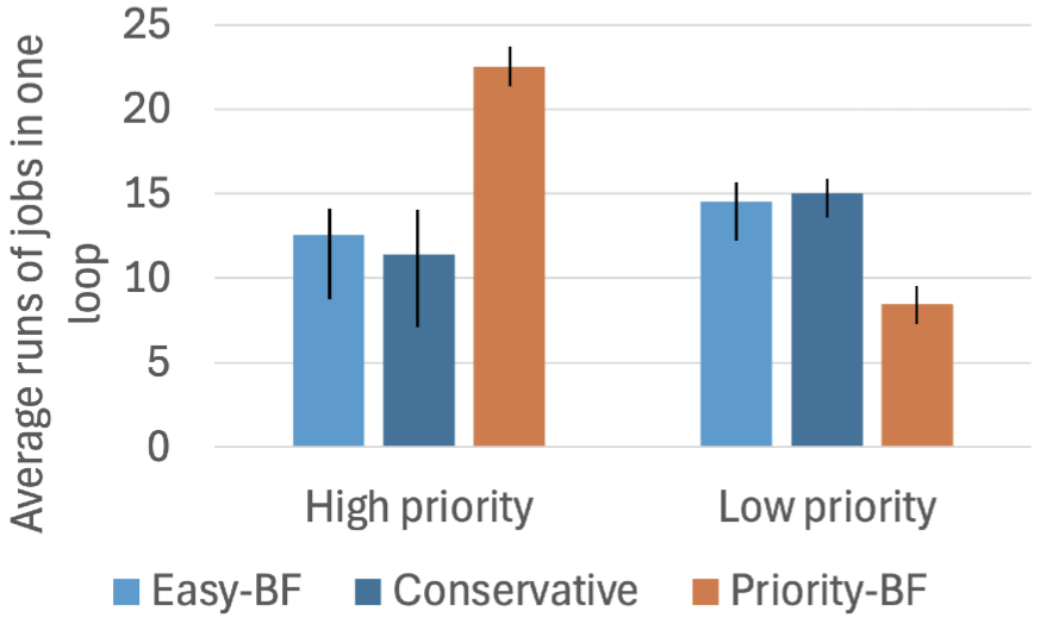
- Changes at the user level
  - Decide on number of queues
  - Set policies for the end of loop strategy
  - Update same queue task order
  - Update priorities
- Changes in the scheduler
  - Support multiple waiting queues
  - Support mid-execution start
  - New backfill strategy based on multiple queues

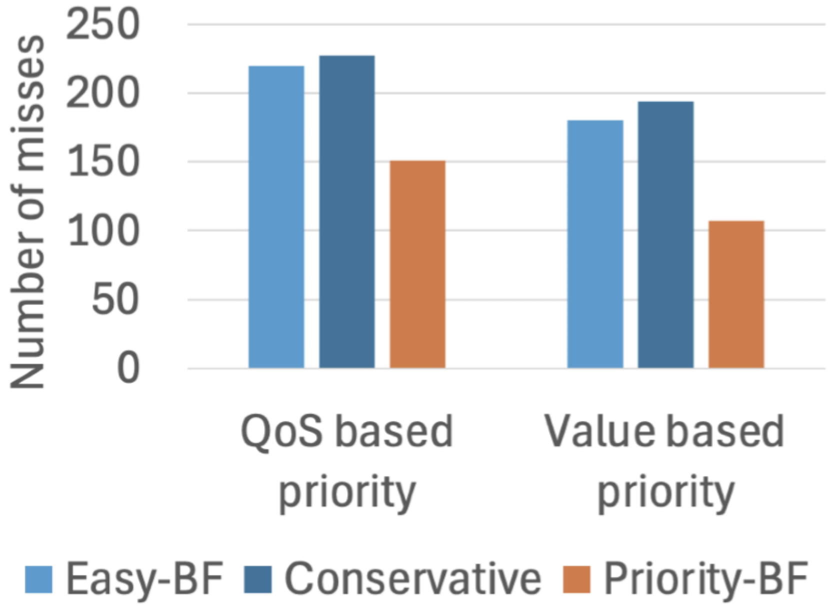
Algorithms and implementation



- Priority to queue mapping
  - **Value-based**
    - Implement as many queues as priority classes
    - Jobs do not transition from one class to another
  - **Frequency-based**
    - Two priority queues
    - Jobs that need executing in the current step are high
    - Everything else is low
    - Jobs move from one queue to another based on past schedule

Results





Value based priorities

Average number of times a job was executed across all simulation loops (max 30)

Frequency based priorities

Number of loops where a job was supposed to be executed and it wasn't

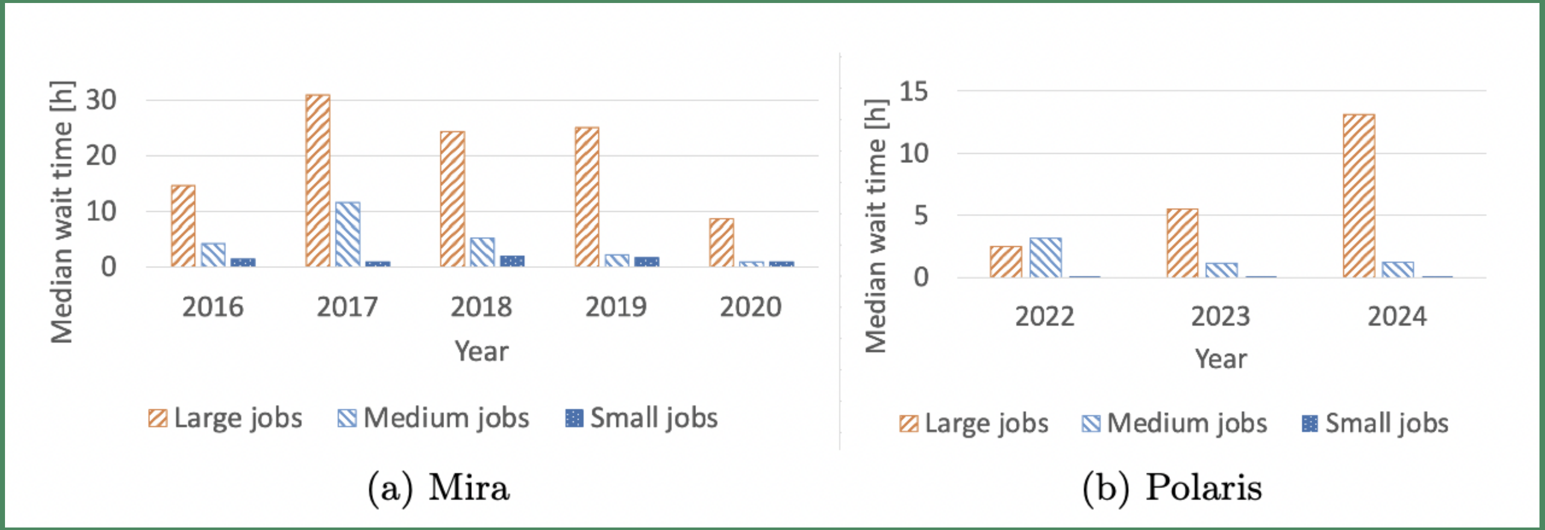
- 20 jobs (nodes, reqest, walltime, priority)
- 30 loops where loop i takes random time Xi

- 60 experiments with different random seeds
- Value and frequency based priorities

Oak Ridge National Laboratory

Moving beyond analysis tasks

- Can we use Priority-BF for existing jobs?
- Using ANL system logs
  - **Goal:** decrease the average wait time for long jobs
  - 3 levels of priorities



(a) Mira

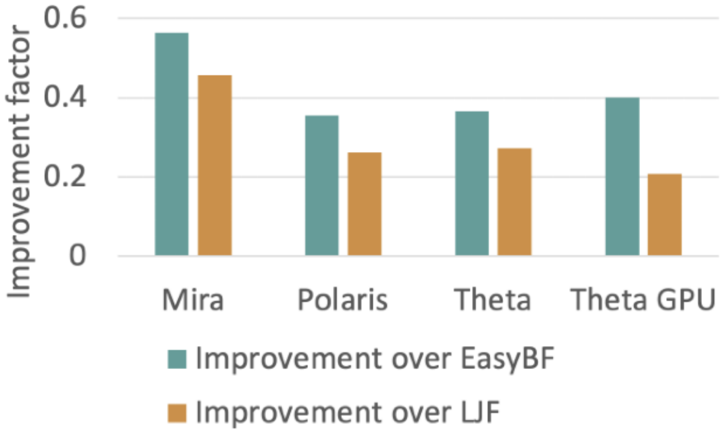
(b) Polaris

Jobs submitted to Mira and Polaris show increasing median wait times of hours, especially for large jobs

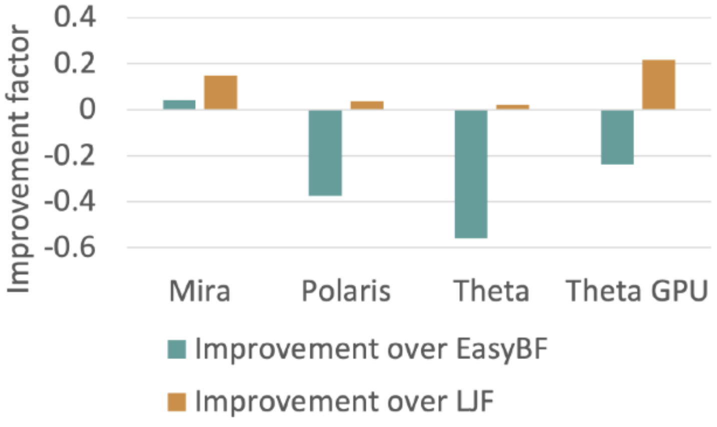
Logs of jobs in real systems

- Utilization is within 2% of Easy-BF and LJF
  - Response time improves for high priority jobs (20-55%)
  - Response time decreases by 3x for low priority jobs

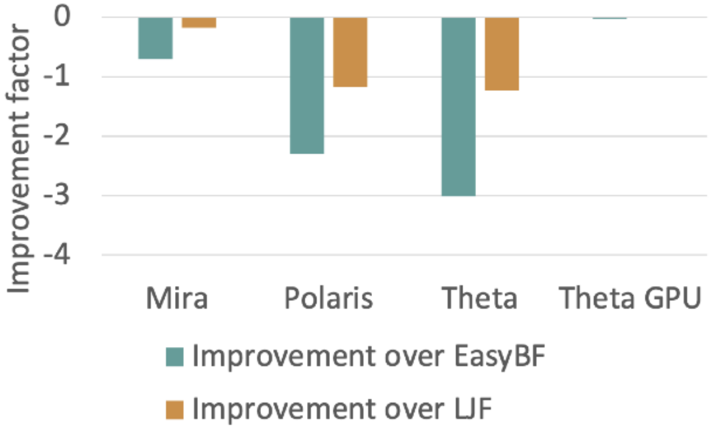
Response time for **high** priority jobs



Response time for **medium** priority jobs

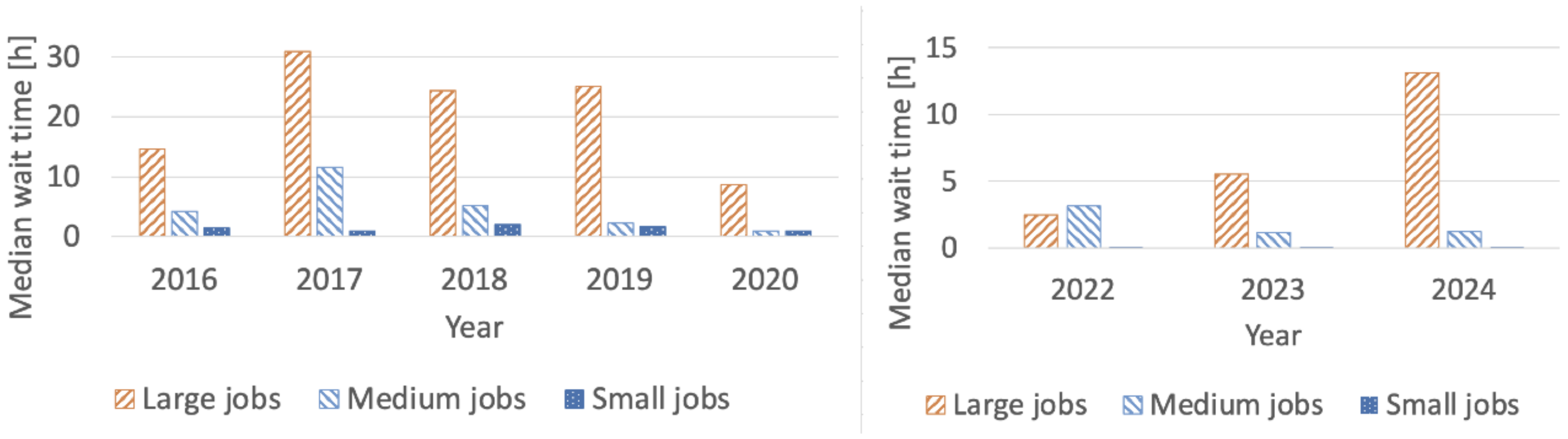


Response time for **low** priority jobs



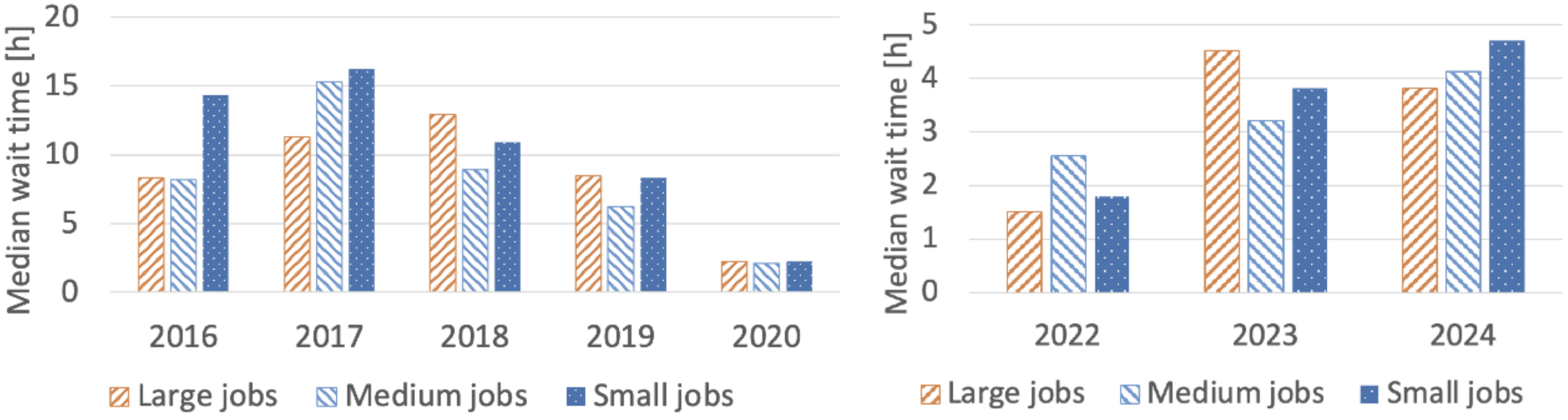
Overall results

- Uniform wait times
  - Average of hours even for small jobs
  - Decreased for large jobs
- Not necessary the best comparison
  - Simulation vs real life
  - More experiments are needed to better understand the impact



(a) Mira (b) Polaris

Based on submission time and start time recorded in the logs



(a) Mira (b) Polaris

Using Priority-BF

Conclusions

- Separating scheduling strategies between different classes of jobs is necessary
  - When dealing with limited time and resources
  - When jobs have different priorities
- Future works include
  - More simulations (e.g. BatSim) and experiments to understand the trade-offs
  - Apply the scheduling for several fields
  - Include decisions on where to compute tasks
    - In-situ on the producer, consumer or in-transit

- Scripts used and documentation: https://github.com/ORNL-Inria/PriorityBF