OAK RIDGE National Laboratory

Priority-BF: a Task Manager for Priority-Based Scheduling

Ana Gainaru, Guillaume Pallez, Scott Klasky

31st International European Conference on Parallel and Distributed Computing
Euro-Par 2025
Dresden, Germany, August 29, 2025

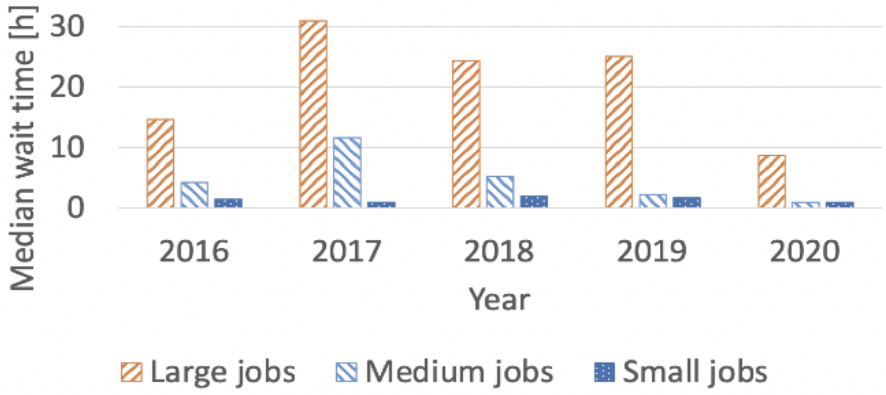
ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

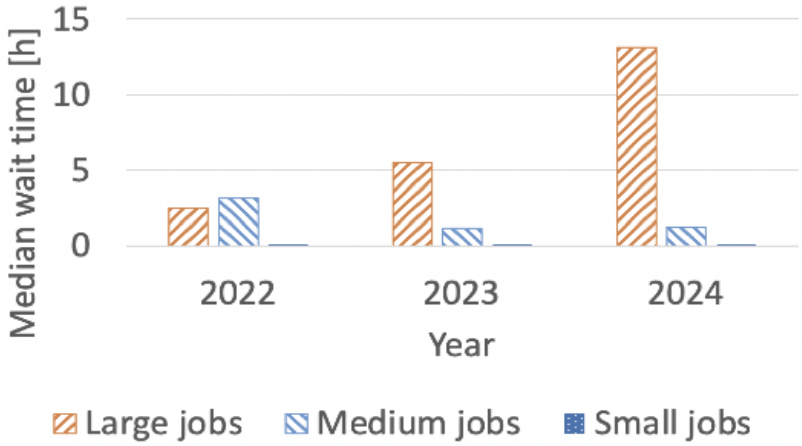
gainarua@ornl.gov

Priority-based scheduling

- **Why is priority-based scheduling needed?**
  - HPC wants to encourage large jobs
    - With additional priorities based on each configuration
  - Current HPC simulations generate up to PB data/step
    - Often requiring post-processing tasks in real time
    - Some tasks are more important than others
- **This talk**
  - Our solution
    - Limitations of current scheduling strategies
    - Our philosophy and implementation
  - Results
    - Priority for large jobs
    - In-situ tasks



(a) Mira



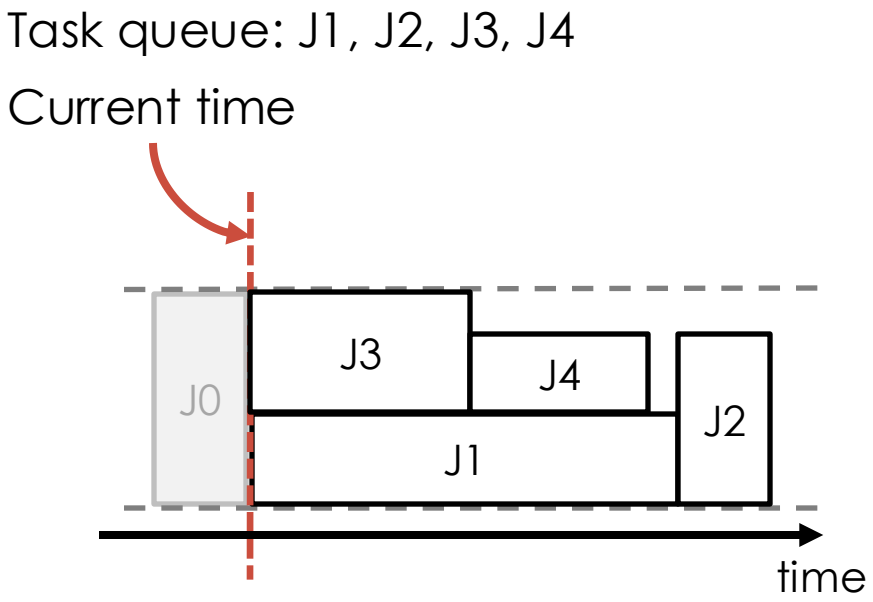
(b) Polaris

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

Priority based scheduling

- Current solutions in HPC
- Easy-BF, Conservative-BF
- Our philosophy

General scheduling problem



- Algorithm **input**
  - Set of tasks that need to be executed
  - State of the machine at current time
- Algorithm **output**
  - Preliminary start time for each of the tasks in the queue

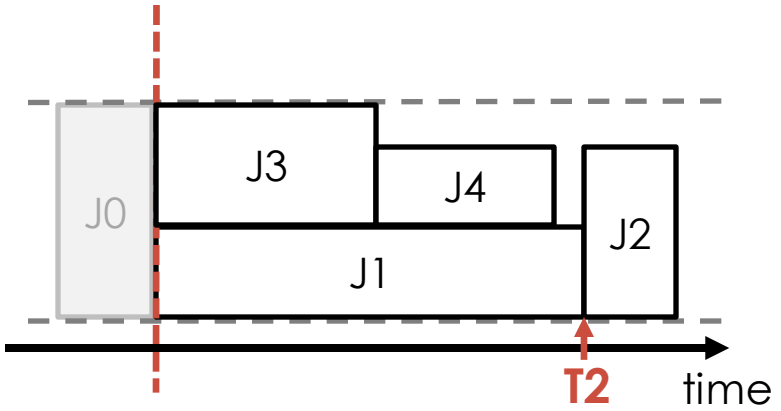
- Current solutions
  - **Batch scheduling**
    - Divide the list of tasks in batches and compute an optimal schedule within a batch
  - **Online scheduling**
    - Recompute the schedule on job end and when a job is added to the queue

Preferred in HPC

Current solutions

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

Task queue: J1, J2, J3, J4



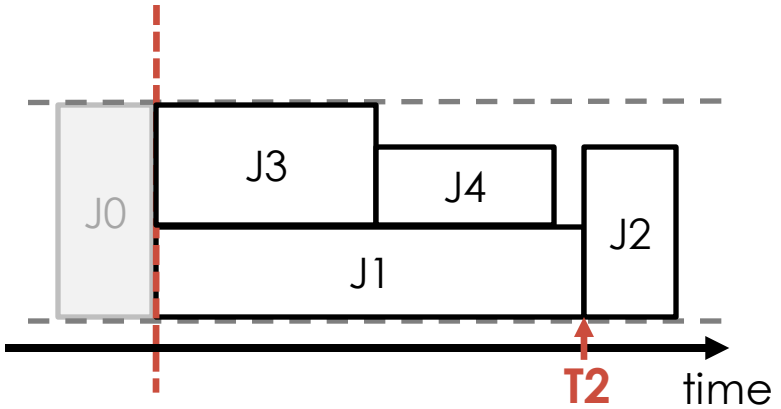
J0 finished, J1 and J2 are scheduled

- J1 and J3 start running
- J2 **is guaranteed** a start the latest at time T2
- J4 is mutable

Current solutions

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

Task queue: J1, J2, J3, J4

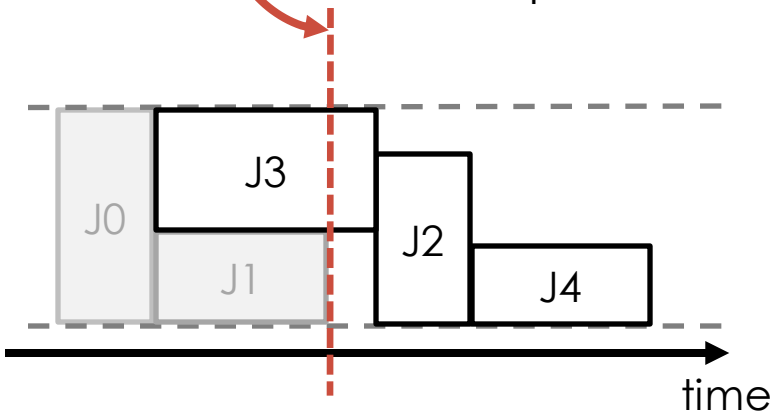


J0 finished, J1 and J2 are scheduled

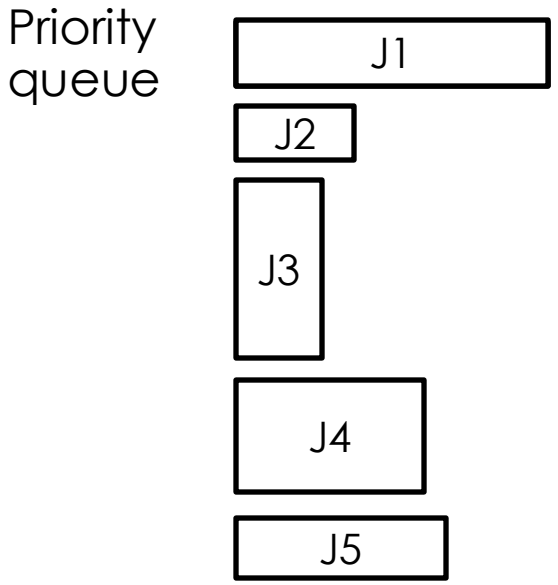
- J1 and J3 start running
- J2 **is guaranteed** a start the latest at time T2
- J4 is mutable

Recompute schedule

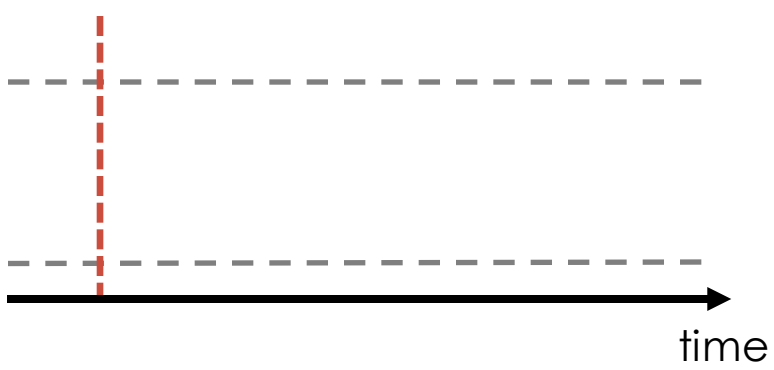
Task queue: J2, J4



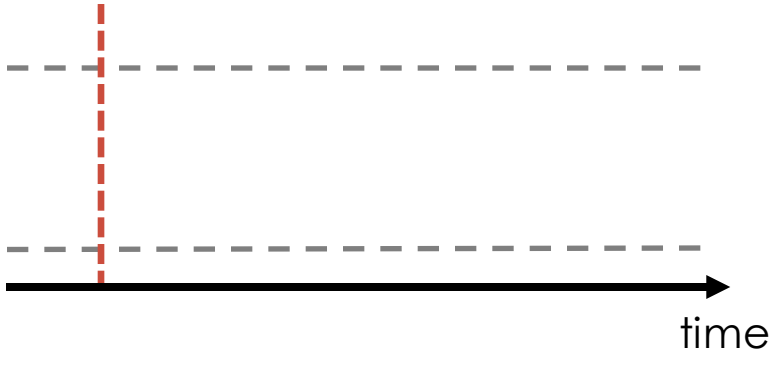
Limitation for priorities



- **Goal:** Minimize the wait time for high priority jobs
  - Given fixed amount of resources
  - The order of execution will influence the wait time
- Assuming we can set job priorities
  - Simplest: **based on job size/user** etc
  - Becomes more complex when priorities are based on the type of science being done

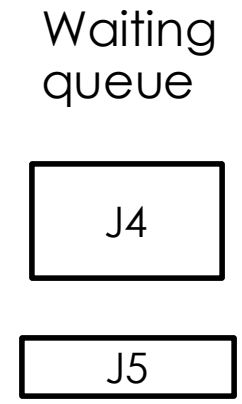


Easy-BF

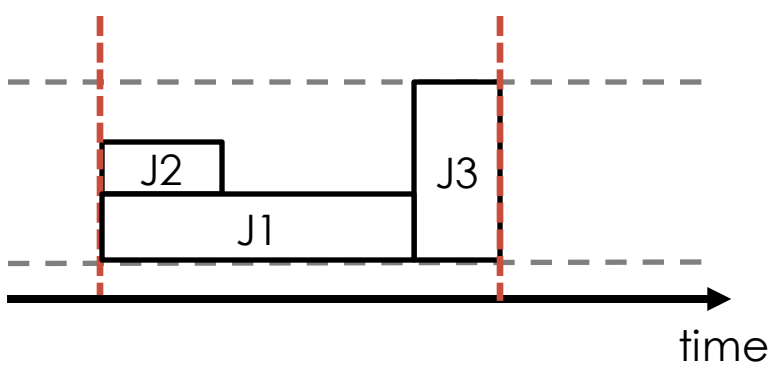


Conservative-BF

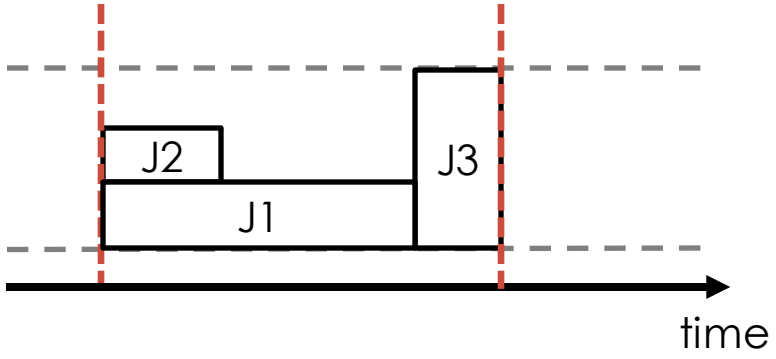
Example



- 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



Easy-BF



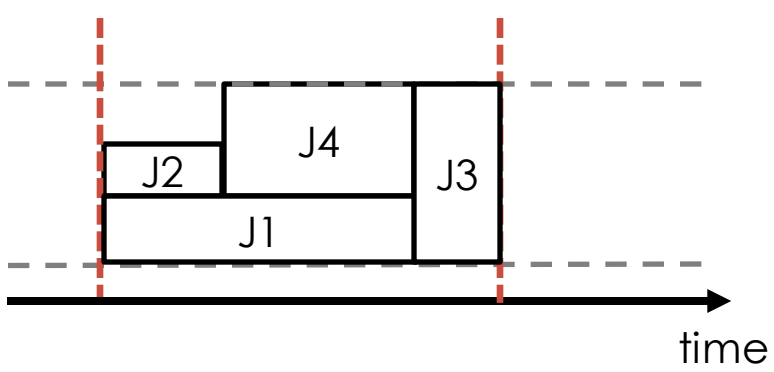
Conservative-BF

Example

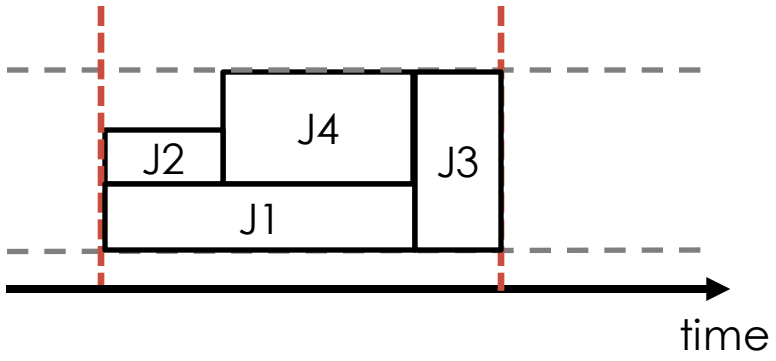
Waiting queue

J5

- 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



Easy-BF

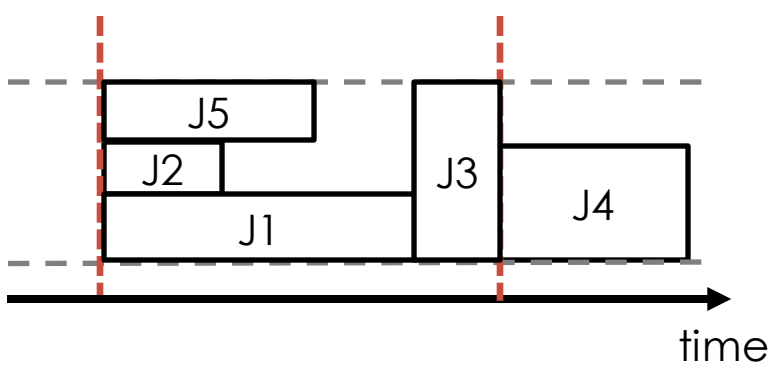


Conservative-BF

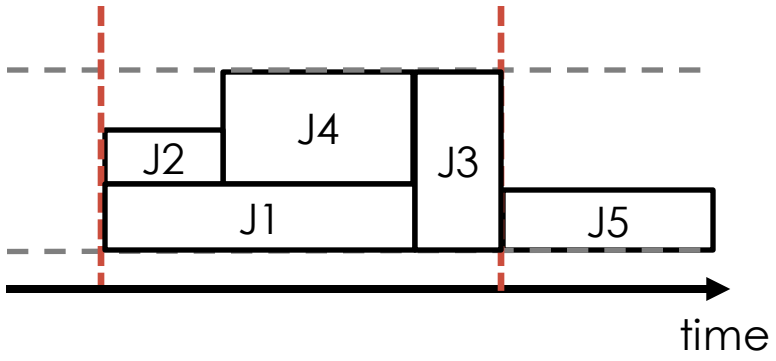
Example

Waiting queue

- 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



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

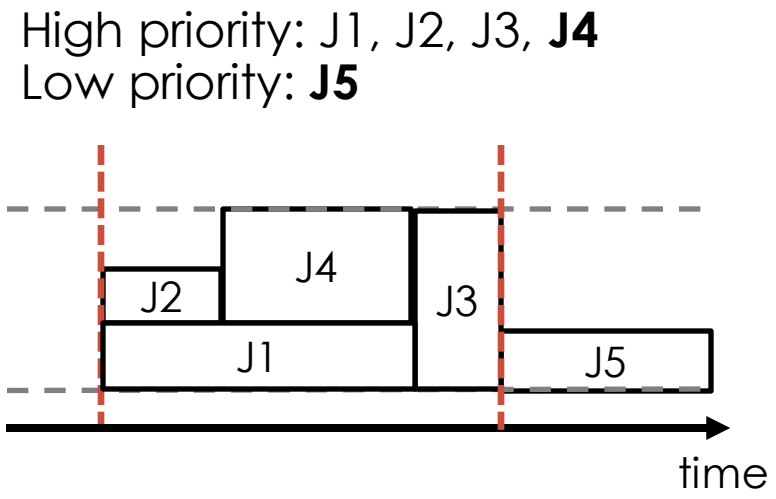
- Incorporate job importance
  - At the granularity of the job (set by users)
  - When all jobs share the same priority our algorithm reverts to Easy-BF

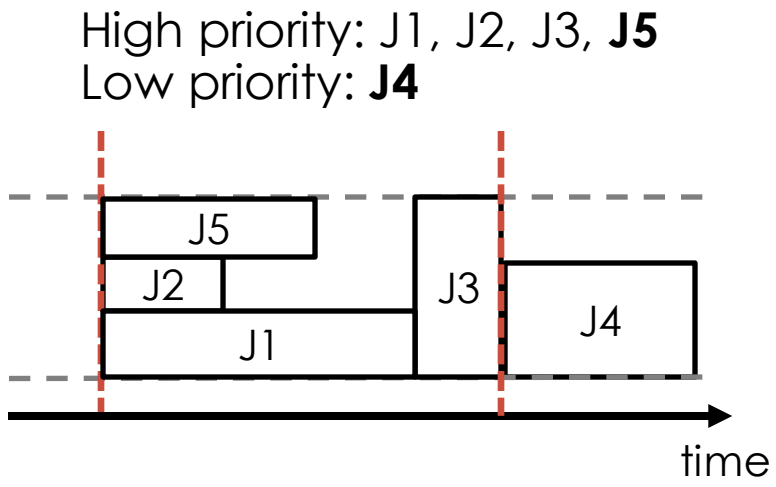
Our proposal for scheduling algorithm

- 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 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





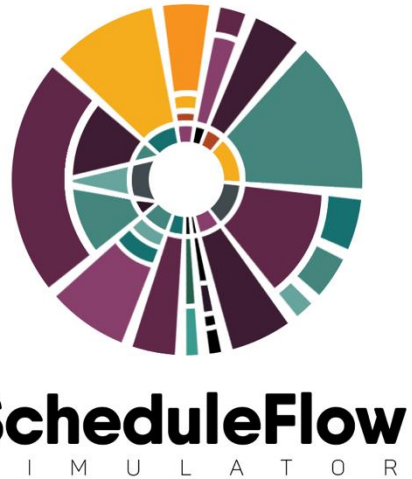
- **Strategies for in-situ scheduling**
  - Jobs that did not finish by the end of the time window
    - Kill all jobs (fresh start), keep all jobs that started, keep only high priority jobs
  - Memory-less scheduling
    - Each loop uses the same queue (J5/J4 will starve) or updated queue

Evaluation

- Evaluation details and implementation
- Results for scheduling large jobs in HPC
- Results for scheduling in-situ tasks

Evaluation

- Using the ScheduleFlow simulator
  - Simple to use and to add new algorithms
  - For now, we don't need system characteristics
    - More complex simulators (BatSim or WRENCH) in the future
  - **Priority-BF compared to Easy-BF and Conservative-BF**
    - Ordered using the same priorities



- Experiments
  - **HPC scheduling**
    - Using ANL system logs with 3 levels of priorities
    - Goal: decrease the average wait time for long jobs
  - **In-situ scheduling**
    - Neuroscience highly stochastic applications
    - Random priorities using values or QoS frequency

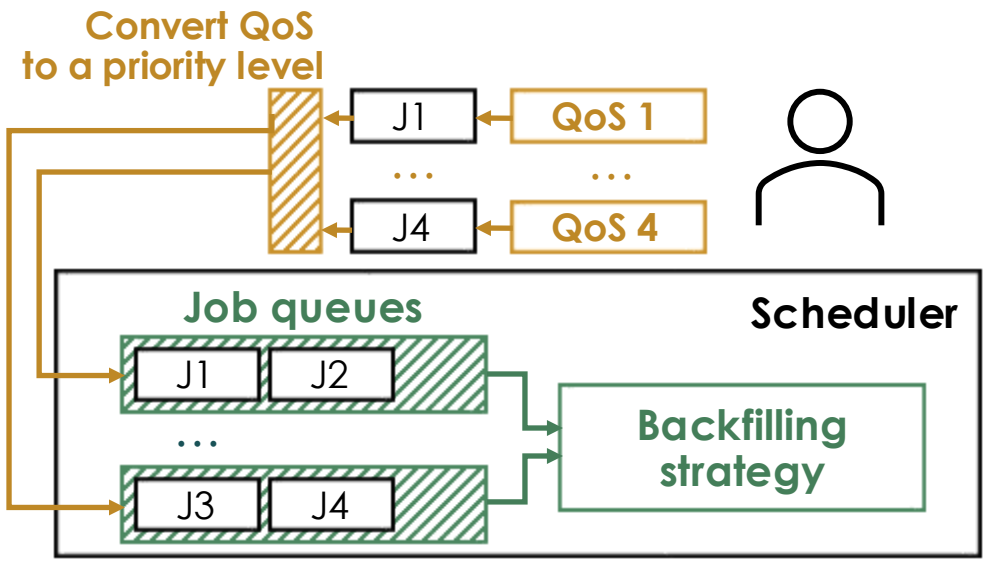
Metrics

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

Implementation changes to the simulator

General changes

- Support multiple waiting queues
- New backfill strategy based on multiple queues



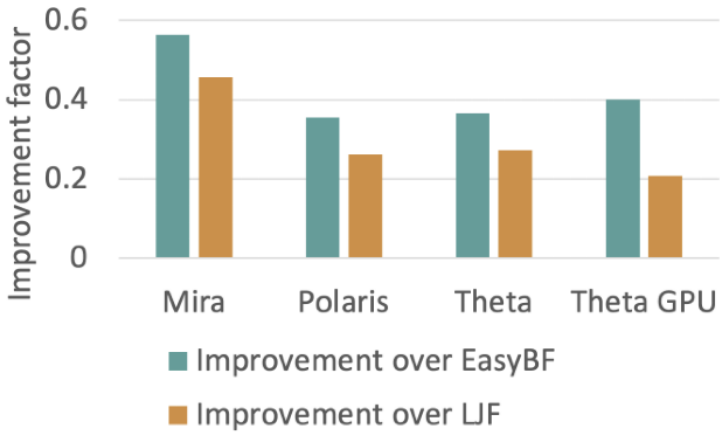
Required by the in-situ scheduling

- 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

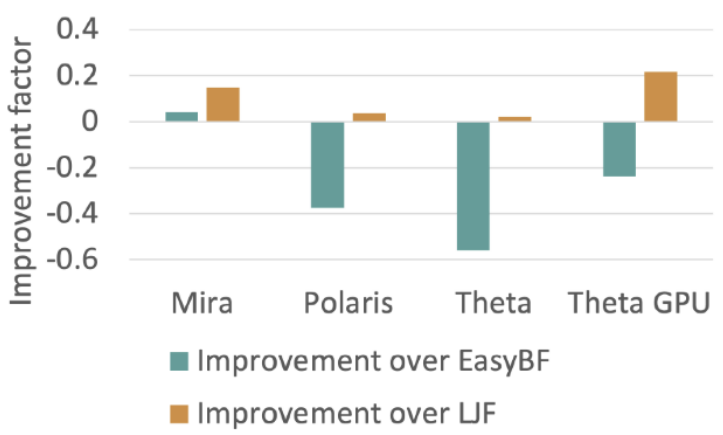
Logs of jobs in real systems

- Utilization is within 2% of Easy-BF and LJF
  - Response time improves for high priority jobs (20-55%)
    - From an average of **5h to 2.5h for Polaris** and **from 17h to 8h for Mira**
  - Response time decreases by 3x for low priority jobs
    - From an average of **minutes to 1.5h – 3h** for Polaris and Mira

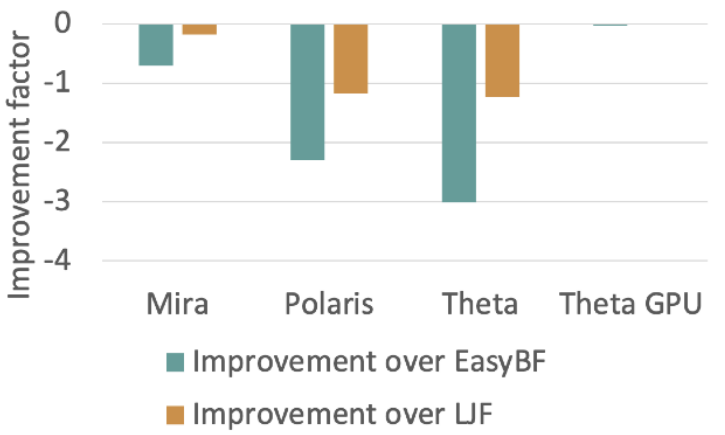
Response time for **high** priority jobs



Response time for **medium** priority jobs

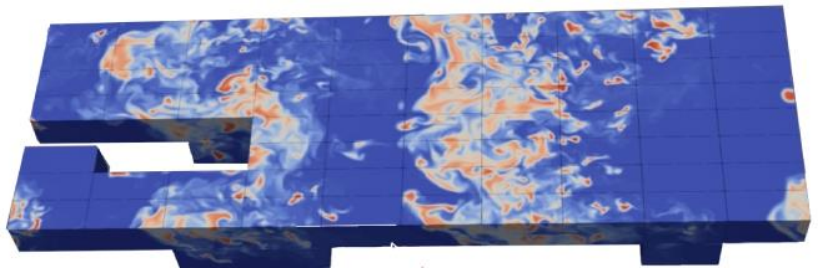


Response time for **low** priority jobs

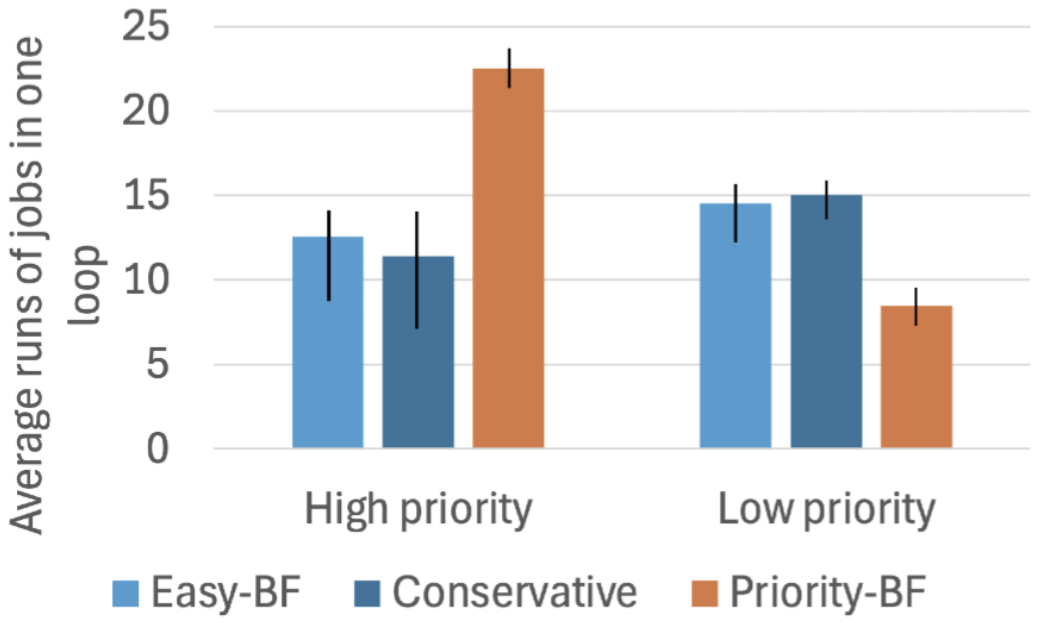


In-situ data processing tasks

- 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

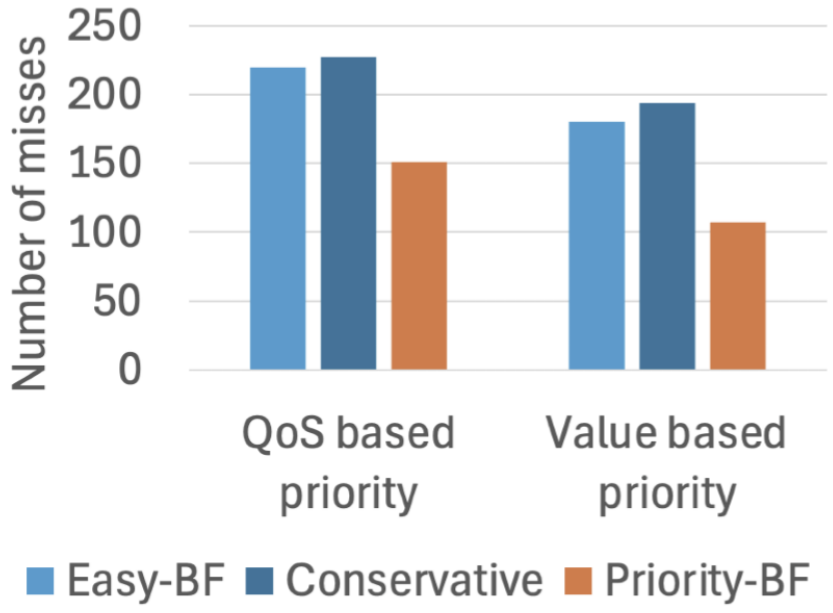


Results



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

gainarua@ornl.gov

Conclusions

- Priority-based scheduling
  - Requires separated scheduling strategies between different classes of jobs
  - Necessary when dealing with limited time and resources
  - Necessary when dealing with high job throughput
    - Early start guarantees high utilization without impacting the wait time for high priority jobs
- Future works include
  - More simulations and experiments to understand the trade-offs
  - Apply the scheduling in-situ tasks for several domain sciences
  - Include decisions on where to compute tasks
    - In-situ on the producer, consumer or in-transit

PriorityBF

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