TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

EquilibrIO: Taming the I/O Tides in High-Performance Computing

(to be presented at IEEE CLUSTER 2025)

Taylan Özden, Ahmad Tarraf, and Felix Wolf

Technical University of Darmstadt

Motivation: Data-intensive applications

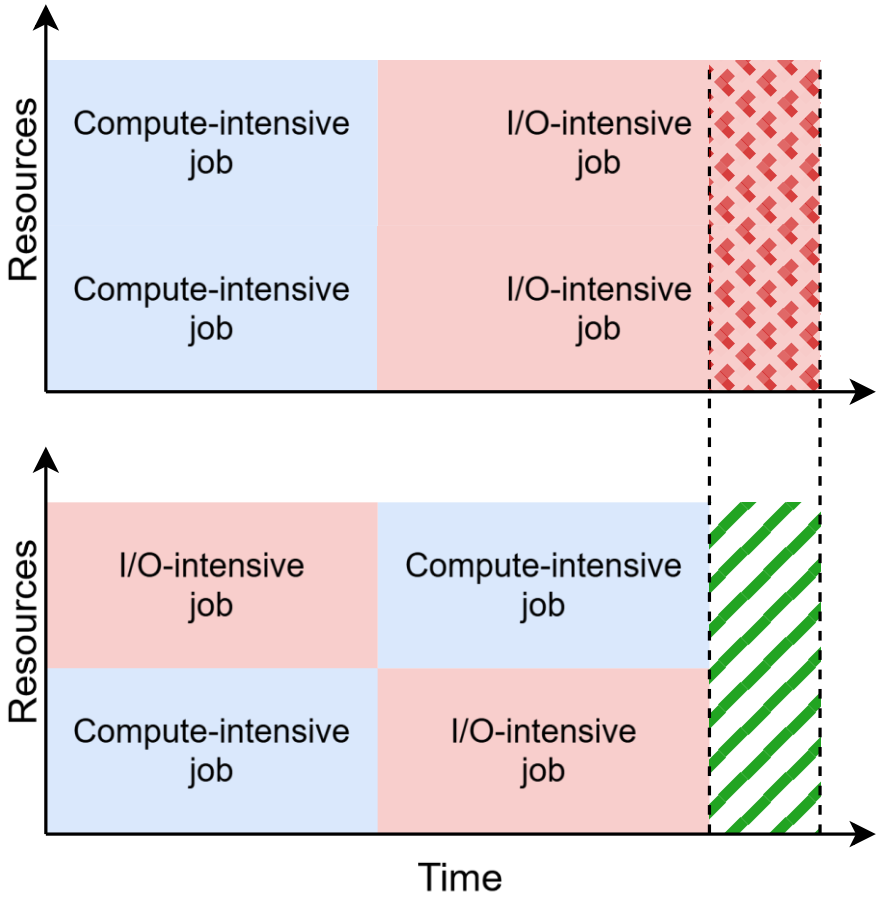
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Amdahl's law: "The overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used."

- Consequently, data-intensive application suffer more from lower I/O bandwidth than compute-intensive ones

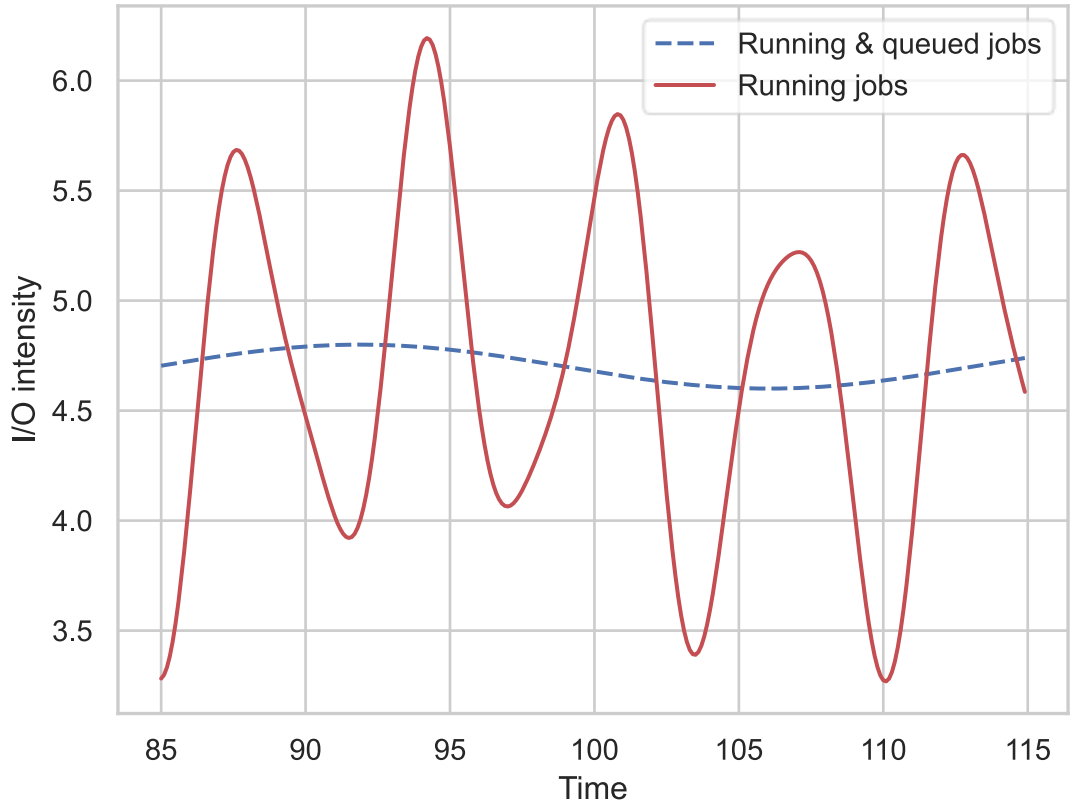
Imbalance between computation and I/O



- Compute-intensive jobs can better tolerate data-intensive ones scheduled alongside
- We propose *EquilibrIO*, a novel scheduling algorithm to mitigate I/O contention by
  - keeping the I/O intensity of running jobs close to the average I/O intensity of the entire workload, while
  - still maintaining schedule fairness

Intuition behind our approach

- When I/O-intensive jobs do not overlap, the probability of contention reduces
- *Hypothesis*: The overall I/O intensity of running jobs will oscillate around the average I/O intensity of the entire workload (incl. queue)
- *Example*: stock market
  - Unfortunately, hard to impossible to predict the outcome in the future
- However, jobs in the queue are an indicator for the near future

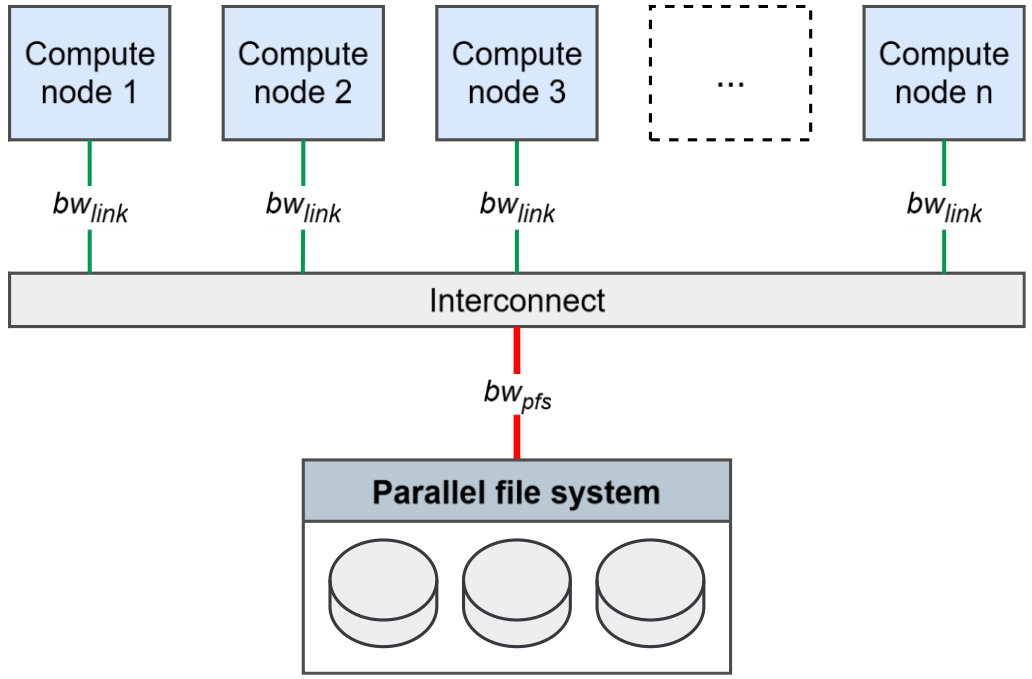


Problem definition & assumptions

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- **No scheduling of I/O bandwidth**
- I/O intensity of a job (roughly) known (e.g., via Darshan logs)
- Applications have
  - exclusive access to compute nodes
  - shared access to the parallel file system (PFS)
- During I/O-intensive phases, congestion occurs, and applications compete for I/O resources



I/O intensity

- For all running jobs (RJ) and queued jobs (QJ), we introduce three different I/O intensity measurements for
  - each job: $io\_intensity_j := \frac{io\_walltime_j}{total\_walltime_j} \cdot average\_bw_j$
  - the system: $io\_intensity(\mathbb{S}) := \frac{\sum_{j \in RJ}(io\_intensity_j)}{|RJ|}$
  - the workload: $io\_intensity(\mathbb{W}) := \frac{\sum_{j \in RJ \cup QJ}(io\_intensity_j)}{|RJ \cup QJ|}$

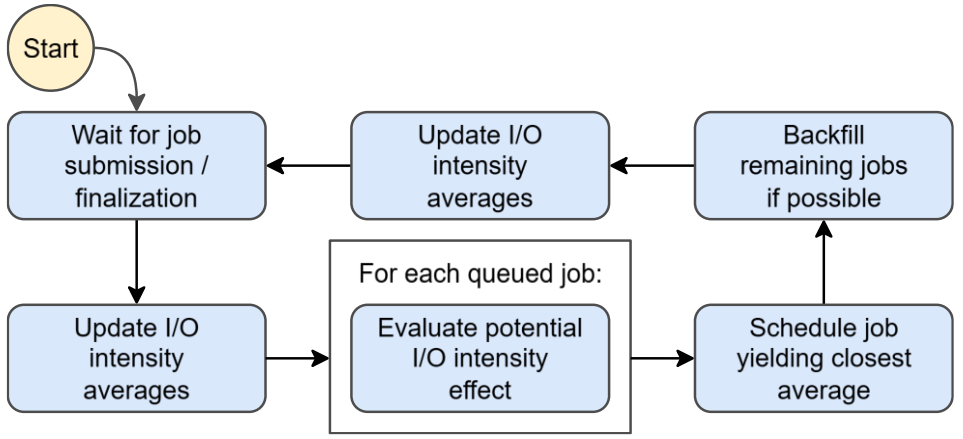
Scheduling algorithm

- Balances the I/O intensity of the executing workload by minimizing

$$|io\_intensity(\mathbb{W}) - io\_intensity(\mathbb{S})|$$

- Invoked at each job submission or completion
- Scheduling events modify the system or workload I/O intensity
- Provides a fallback if no I/O information is available
- Employs backfilling if available resources are not sufficient for the optimal candidate

Event	Affected metric
Job submission	$io\_intensity(\mathbb{W})$
Job admission	$io\_intensity(\mathbb{S})$
Job completion	$io\_intensity(\mathbb{S})$, $io\_intensity(\mathbb{W})$



Preventing job starvation

- Making decisions purely based on I/O intensity may cause starvation
- We introduce a weighted priority metric based on the I/O intensity of a job and its arrival time

- Let $\alpha \in [0,1]$ be the *reordering intensity* the site administrator can choose with
  - $\alpha = 0$ representing first-come first-serve (FCFS)
  - $\alpha = 1$ maximum optimization for I/O intensity

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Fairness priority value

- We derive our proposed fairness priority metric for all queued jobs in the set QJ

- For each candidate $c \in QJ$, we define λ_c, representing its normalized arrival time in the system

- The scheduler calculates the fairness priority value $\lambda_c \in [0,1]$ for all jobs in the queue

$$\lambda'_{min} := \min_{c \in QJ}(arrival\_time_c)$$

$$\lambda'_{max} := \max_{c \in QJ}(arrival\_time_c)$$

$$\lambda_c := \frac{arrival\_time_c - \lambda'_{min}}{\lambda'_{max} - \lambda'_{min}}$$

Weighted priority
(combining fairness with I/O intensity)

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- For each candidate c, we calculate and normalize its intensity delta $\delta_c \in [0,1]$
- The scheduler calculates the weighted priority based on the reordering intensity α
- In the final step, the scheduler chooses the best candidate, represented by the minimum weighted priority value, and schedules the job

$$io\_intensity(\mathbb{S})_c \coloneqq \frac{\sum_{j \in RJ}(io\_intensity_j) + io\_intensity_c}{|RJ| + 1}$$

$$\delta'_c \coloneqq |io\_intensity(\mathbb{W}) - io\_intensity_c|$$

$$\delta'_{min} \coloneqq \min_{c \in QJ}(\delta'_c),$$

$$\delta'_{max} \coloneqq \max_{c \in QJ}(\delta'_c)$$

$$\delta_c \coloneqq \frac{\delta'_c - \delta'_{min}}{\delta'_{max} - \delta'_{min}}$$

$$wp\colon \mathbb{R}^3 \to \mathbb{R}, wp(\alpha, \lambda, \delta) \mapsto (1-\alpha) \cdot \lambda + \alpha \cdot \delta$$

Algorithm

- At each invocation, we recalculate the I/O intensity averages and make scheduling decisions based on the weighted priority

Require: queued jobs (list): QJ,
number of free nodes ($\mathbb{N}_0^+$): $nodes_{free}$,
invocation trigger (enum): $trigger$,
triggering job (job): job
reservations (hashmap[job $\rightarrow \mathbb{R}^+$]): $reservations$

1: **if** $trigger = \text{JOB\_ARRIVAL}$ **then**
2: $\quad add\_job\_to\_workload\_io\_intensity(job)$
3: **else if** $trigger \in \{\text{JOB\_COMPLETED}, \text{JOB\_KILLED}\}$ **then**
4: $\quad remove\_job\_from\_system\_io\_intensity(job)$
5: $\quad remove\_job\_from\_workload\_io\_intensity(job)$
6: $\quad$ **delete** $reservations_{job}$
7: $nodes_{min} \leftarrow \min(nodes_j : j \in QJ)$ **if** $QJ \neq \{\}$ **else** ∞
8: **while** $QJ \neq \{\}$ **and** $nodes_{free} \geq nodes_{min}$ **do**
9: $\quad C \leftarrow get\_best\_candidates(QJ)$
10: $\quad c \leftarrow C_1$
11: $\quad$ **if** $nodes_c \leq nodes_{free}$ **then**
12: $\quad\quad admit\_job(c)$
13: $\quad\quad nodes_{free} \leftarrow nodes_{free} - nodes_c$
14: $\quad\quad add\_job\_to\_system\_io\_intensity(c)$
15: $\quad\quad reservations_c \leftarrow current\_time + walltime_c$
16: $\quad\quad reservation_{max} \leftarrow max_{value}(reservations)$
17: $\quad\quad QJ \leftarrow QJ \setminus \{c\}$
18: $\quad\quad nodes_{min} \leftarrow \min(nodes_j : j \in QJ)$ **if** $QJ \neq \{\}$ **else** ∞
19: $\quad$ **else**
20: $\quad\quad C \leftarrow C \setminus \{c\}$
21: $\quad\quad$ **break**
22: **if** $C \neq \text{NULL}$ **then**
23: $\quad$ **for** $c \in C$ **do** ▷ *Backfill pass*

Experimental setup

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Evaluating increasing α

- Analyzing the effects of the reordering intensity on the executed workload
- Continuously increase its value and evaluate $\alpha \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$
- Expecting EquilibrIO to move $io\_intensity(\mathbb{S})$ closer to $io\_intensity(\mathbb{W})$
- Evaluating the achieved performance improvement in I/O-intensive jobs

Evaluating decreasing knowledge

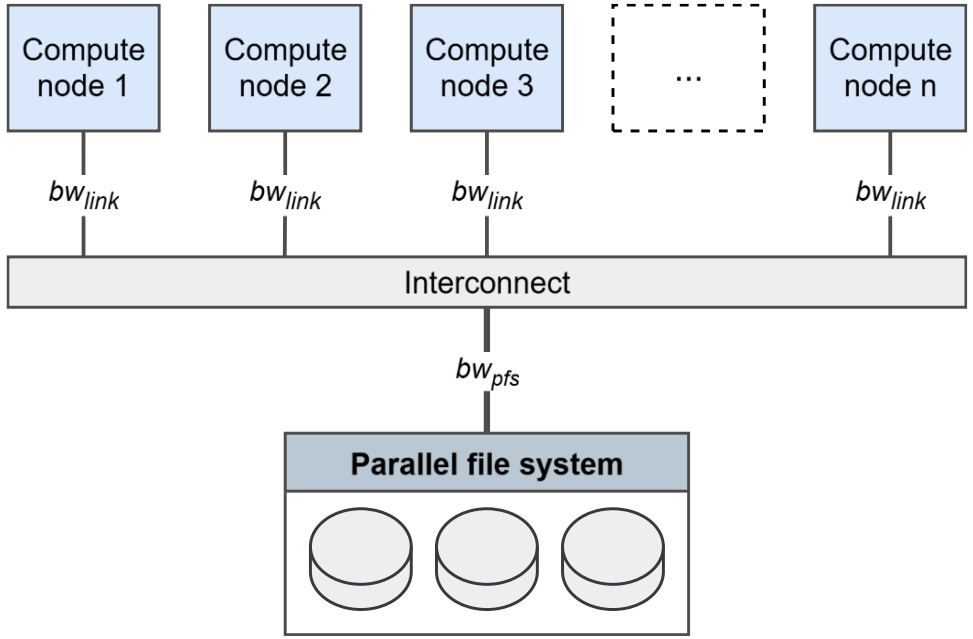
- EquilibrIO provides a fallback mechanism to FCFS if no I/O information is available
- Choose a balanced reordering intensity based on the previous experiment
- Continuously remove I/O information
- Evaluate with $\{100\%, 85\%, 70\%, 55\%, 40\%, 25\%, 10\%, 0\%\}$ a-priori knowledge

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Experimental evaluation

- We use ElastiSim, a batch-system simulator to evaluate our experiments
- We investigated platforms with open access to job and I/O profiles and simulated a system inspired by ANL's Theta (scaled down by the factor 4):
  - 4392 → 1098 compute nodes
  - 172 → 43 GB/s PFS bandwidth (write)
  - 100 Gb/s network connection



https://elastisim.github.io

Taylan Özden, Tim Beringer, Arya Mazaheri, Hamid Mohammadi Fard, Felix Wolf: ElastiSim: A Batch-System Simulator for Malleable Workloads. In Proc. of the 51st International Conference on Parallel Processing (ICPP), Bordeaux, France, pages 1–11, ACM, August 2022 [DOI].

Experimental workload

Information retrieval

- Workload based on Darshan logs collected on Theta from 2017–2023
- Darshan (w/o extended tracing) only provides coarse-grained I/O information, such as
  - Total number of bytes
  - Accumulated time in I/O operations
- Estimating I/O time and bandwidth based on available data

Workload generation

- Divided in low-to-medium and high I/O-intensity jobs
  - High-intensity jobs spend at least 10% of their time doing I/O, reaching 10 GB/s on average
- We combine 5000 low-to-medium intensity jobs with four periods of high-intensity jobs,
  - Each peak comprises 40 jobs
- 5160 simulated jobs in total

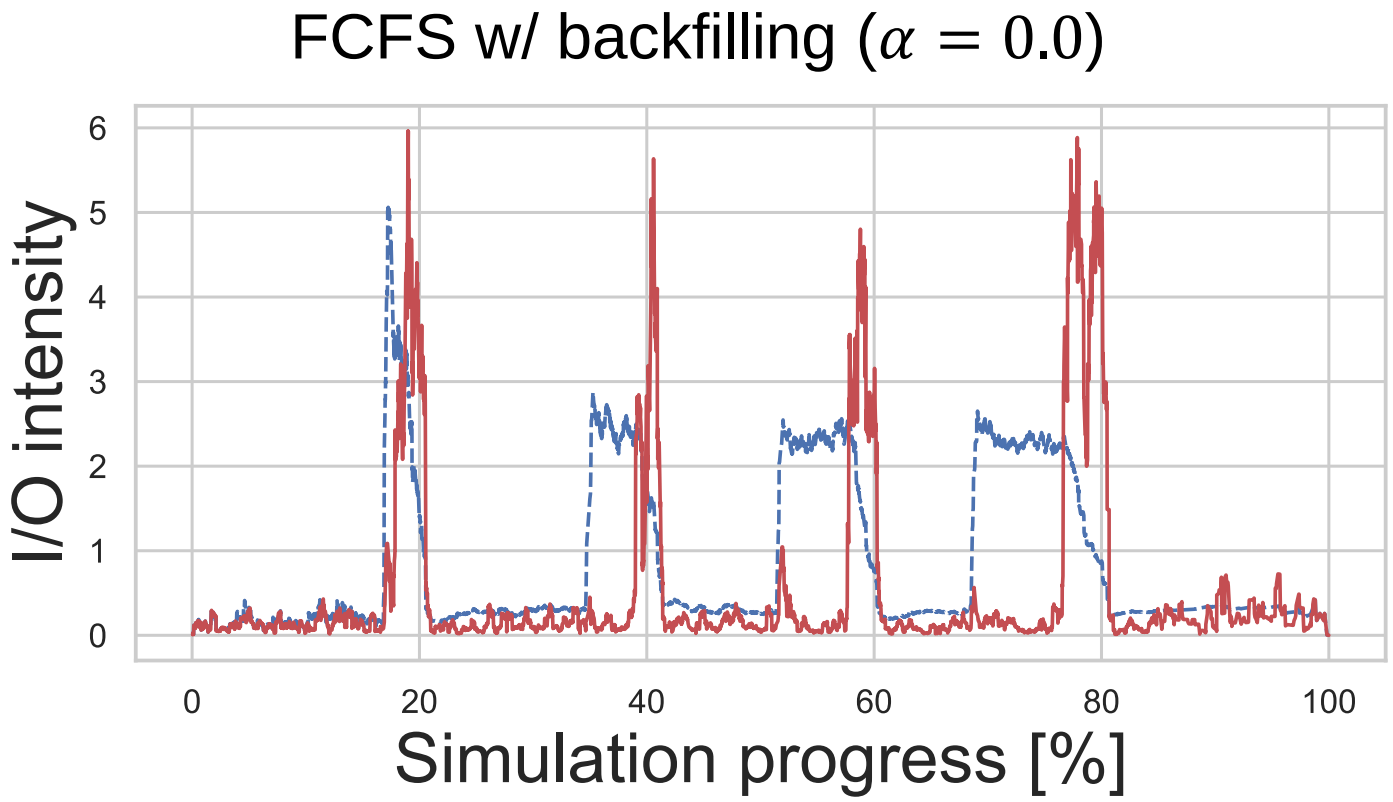
Evaluation metrics

- **Job slowdown**: how much longer a job takes to finalize compared to its isolated execution
- **I/O slowdown**: how much longer I/O tasks take to complete compared to when executed in isolation
- **Utilization**: the fraction of time spent on non-I/O tasks (calculated per job)
- **Displacement**: absolute distance of how far a job is displaced compared to its arrival time order
- **Mean distance**: Average distance between $io\_intensity(\mathbb{S})$ and $io\_intensity(\mathbb{W})$

Results Experiment 1

TECHNISCHE UNIVERSITÄT DARMSTADT

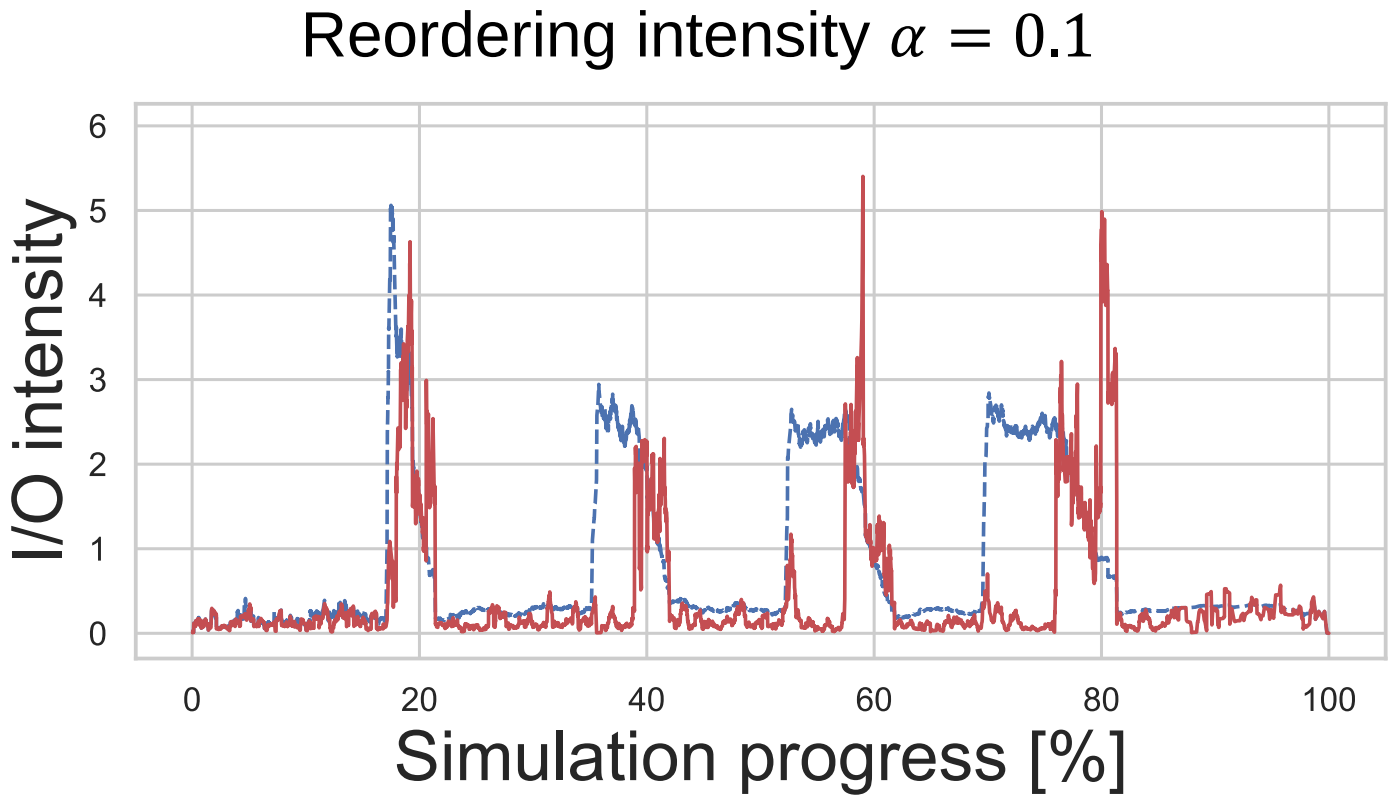
Parallel Programming



- Red solid line represents $io\_intensity(\mathbb{S})$
- Blue dashed line represents $io\_intensity(\mathbb{W})$

- Mean distance: **0.78**

Results
Experiment 1



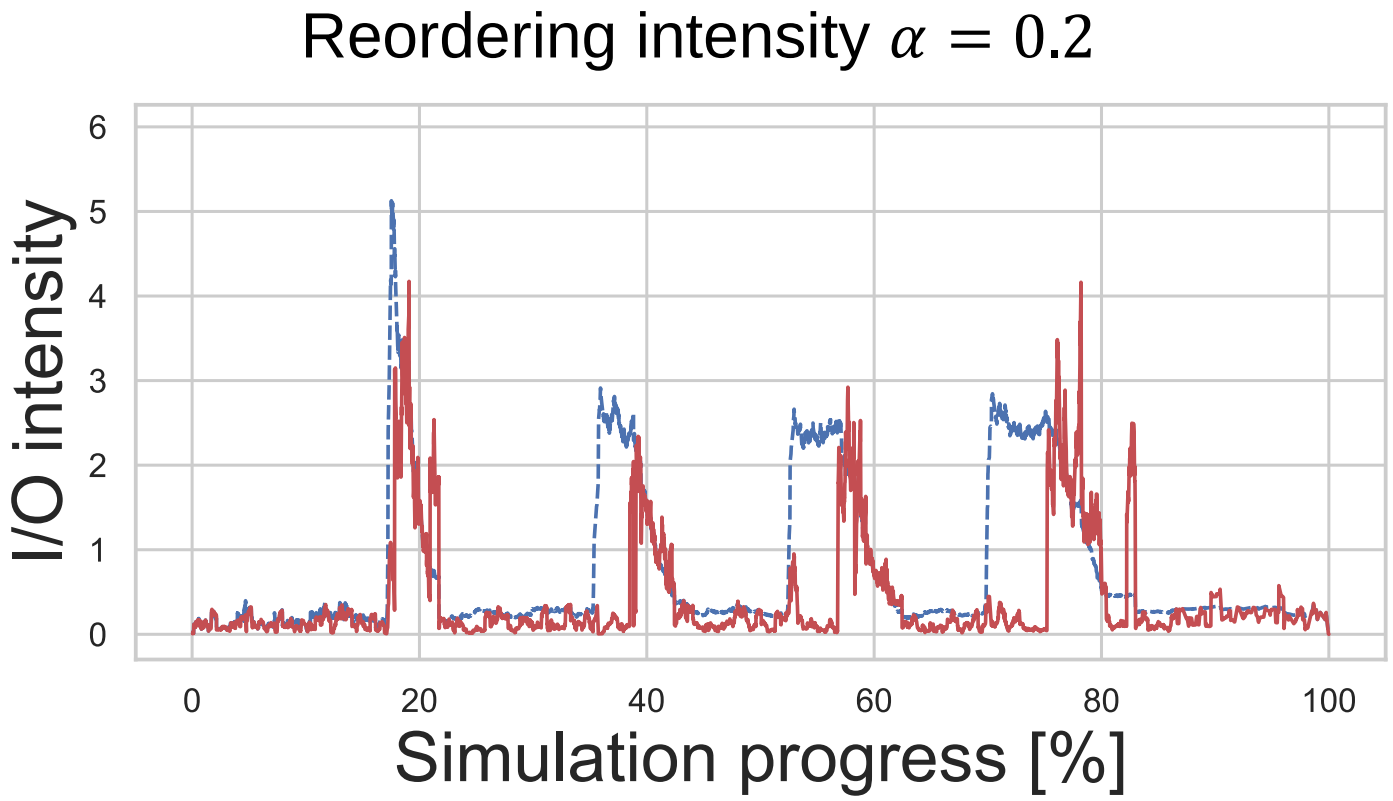
- Red solid line represents $io\_intensity(\mathbb{S})$
- Blue dashed line represents $io\_intensity(\mathbb{W})$

- Mean distance: **0.59**

Results Experiment 1

TECHNISCHE UNIVERSITÄT DARMSTADT

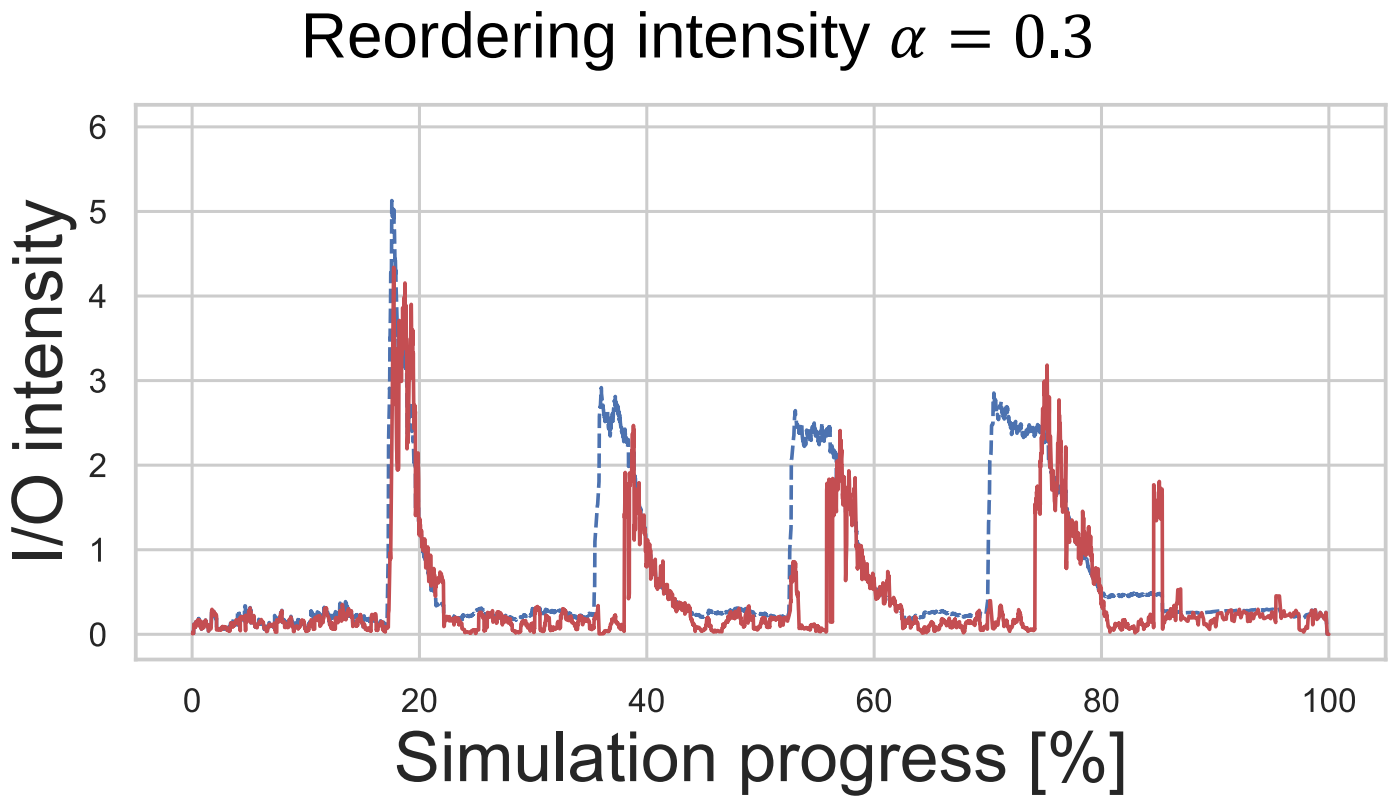
Parallel Programming



- Red solid line represents $io\_intensity(\mathbb{S})$
- Blue dashed line represents $io\_intensity(\mathbb{W})$

- Mean distance: **0.48**

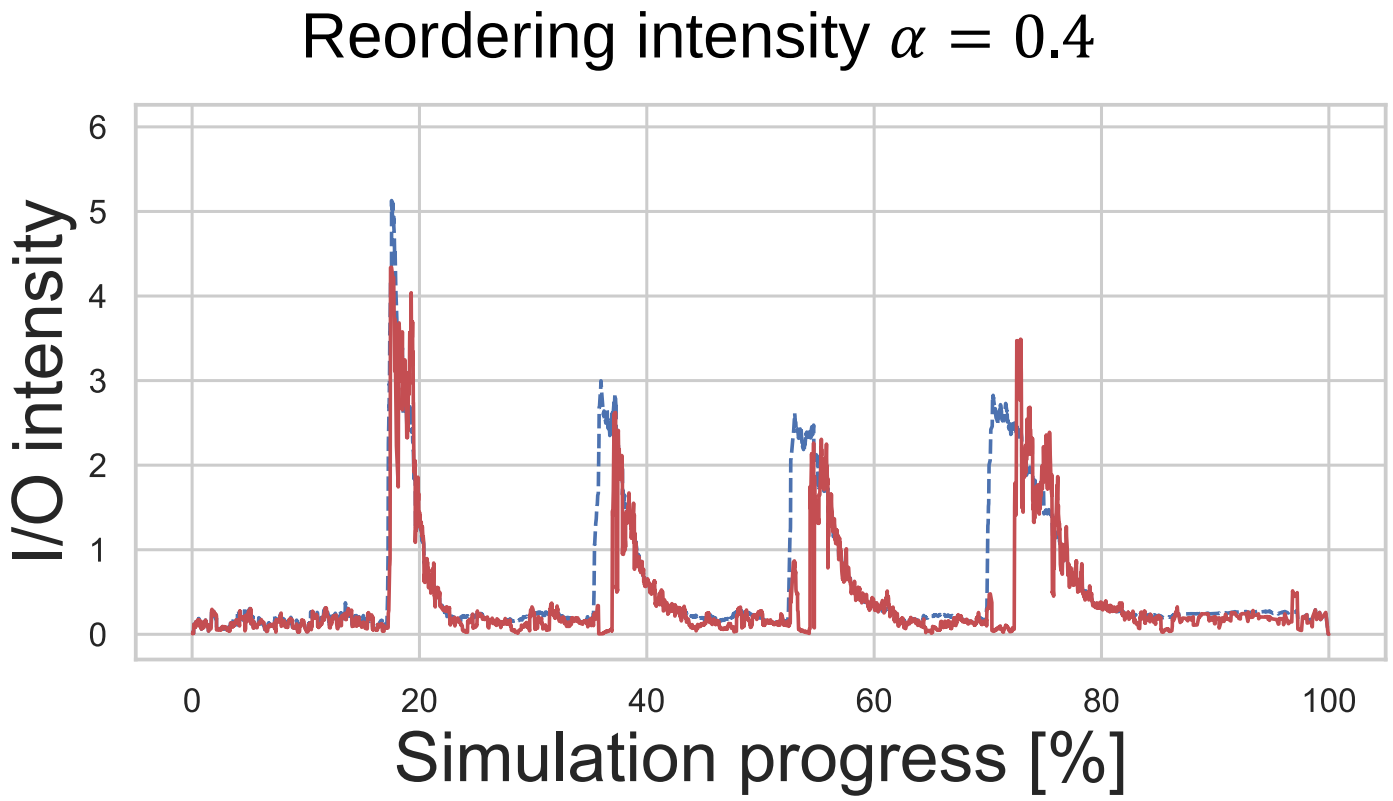
Results
Experiment 1



- Red solid line represents $io\_intensity(\mathbb{S})$
- Blue dashed line represents $io\_intensity(\mathbb{W})$

- Mean distance: **0.38**

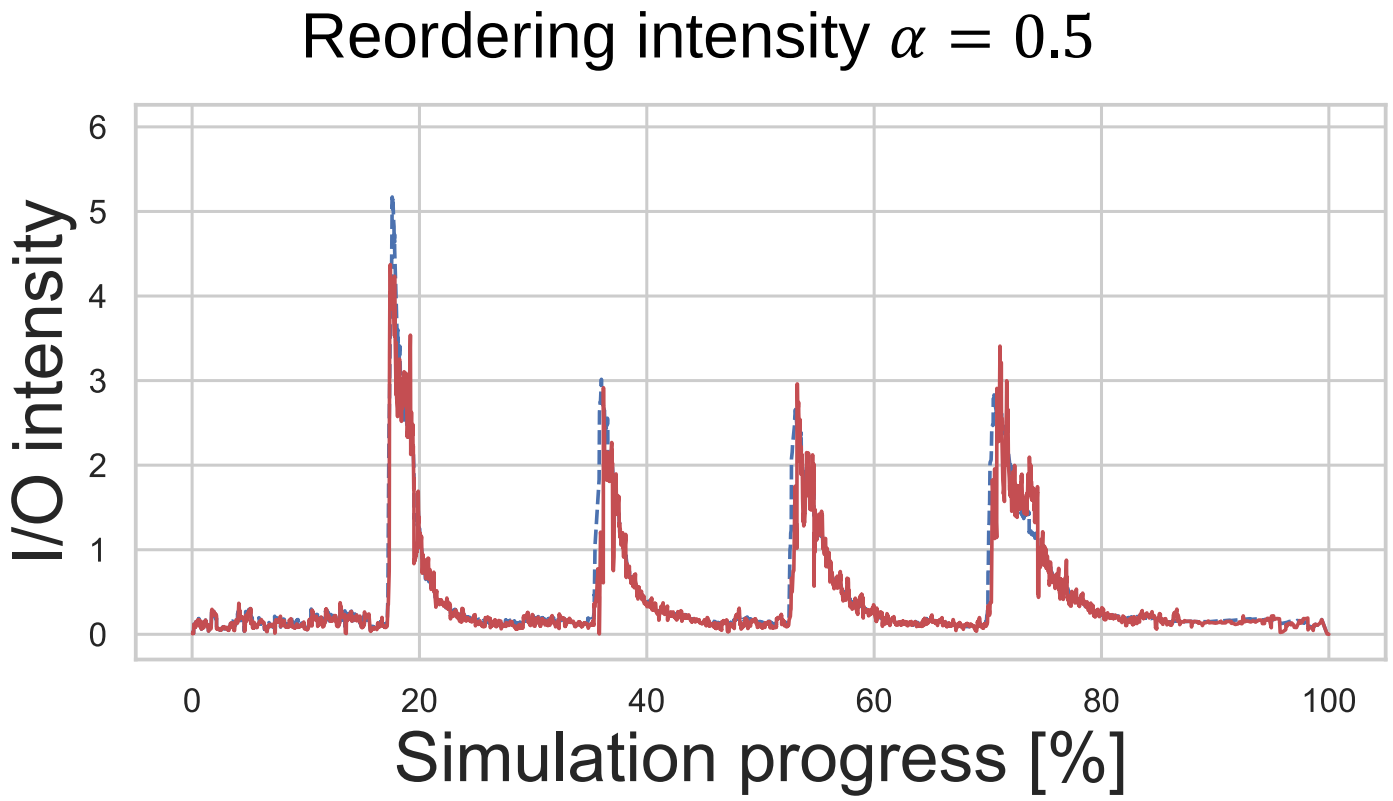
Results Experiment 1



- Red solid line represents $io\_intensity(\mathbb{S})$
- Blue dashed line represents $io\_intensity(\mathbb{W})$

- Mean distance: **0.21**

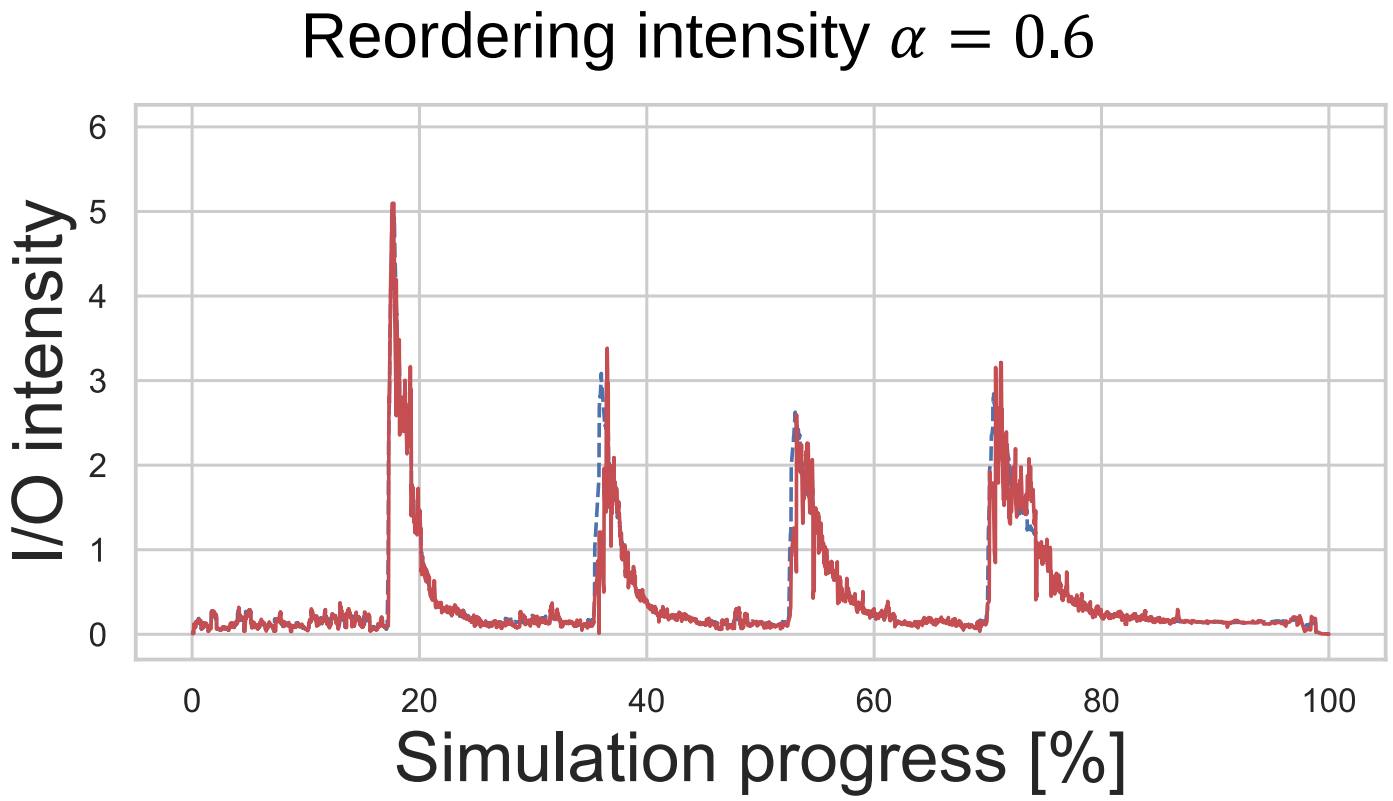
Results Experiment 1



- Red solid line represents $io\_intensity(\mathbb{S})$
- Blue dashed line represents $io\_intensity(\mathbb{W})$

- Mean distance: **0.07**

Results Experiment 1



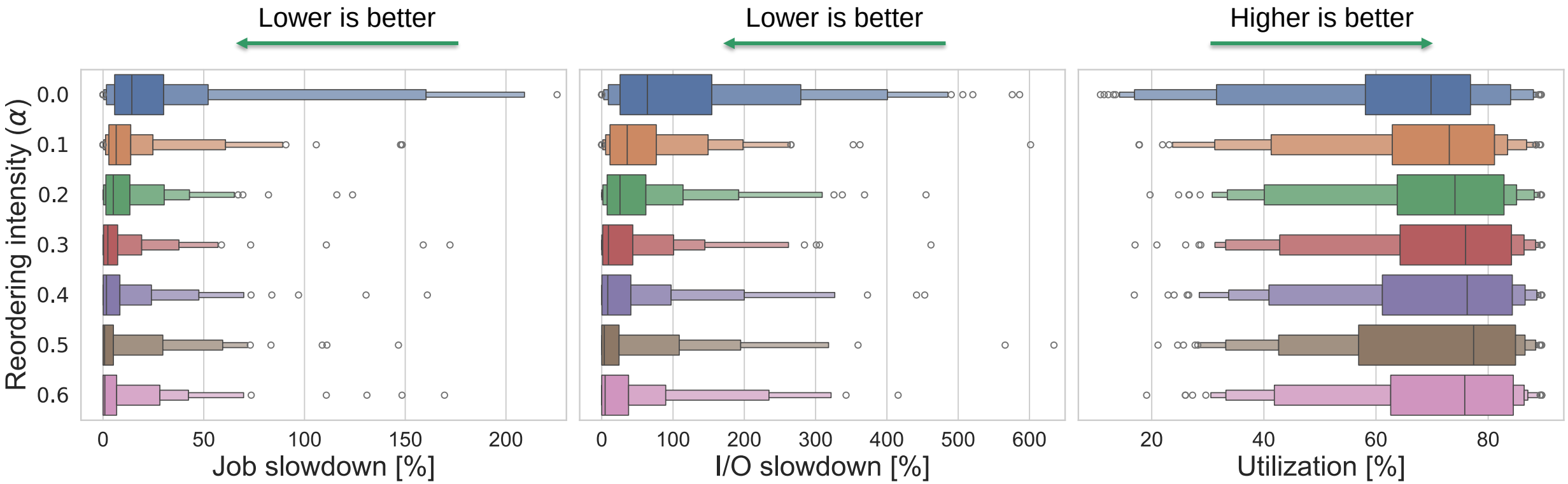
- Red solid line represents $io\_intensity(\mathbb{S})$
- Blue dashed line represents $io\_intensity(\mathbb{W})$

- Mean distance: **0.06**

Results
Experiment 1 (job metrics)

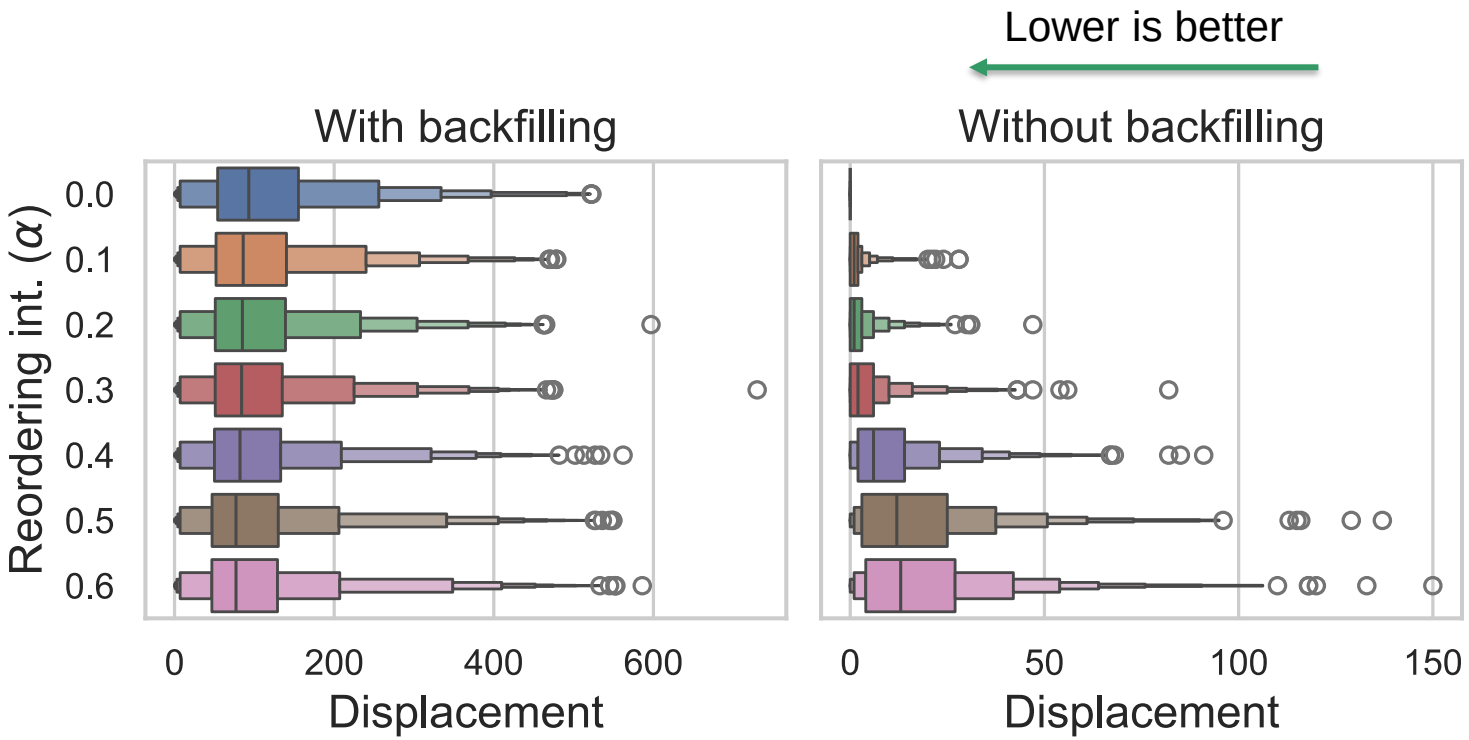
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming



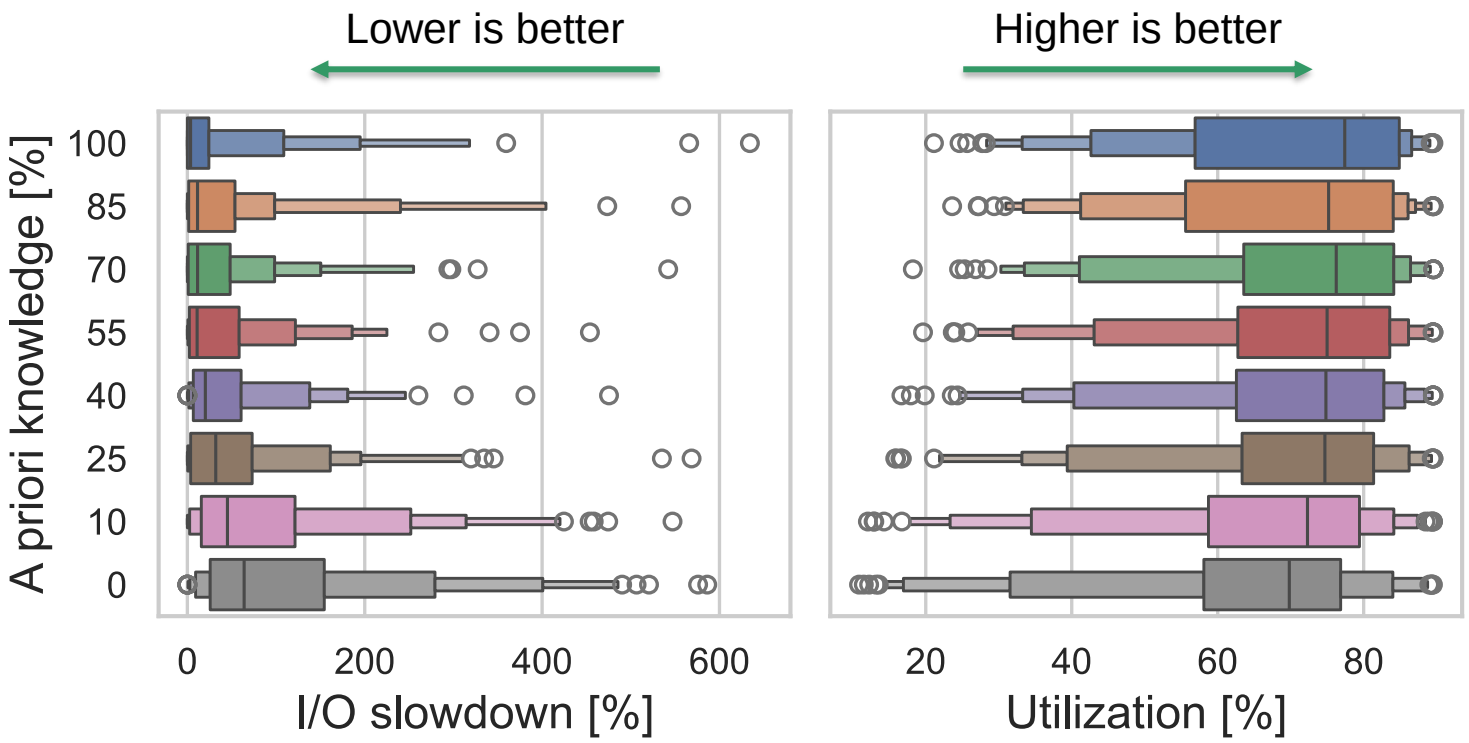
Removed outliers in job slowdown above 230% (4 values, maximum of 316% for $\alpha = 0.0$)
and in I/O slowdown above 650% (9 values, maximum of 1659% for $\alpha = 0.3$)

Results
Experiment 1 (displacement)



- Displacements caused by backfilling dominate those caused by deliberate reordering
- Significant fairness observable for $\alpha \leq 0.3$ even without considering backfilling

Results
Experiment 2 (job metrics)



- 85% and 70% a-priori-knowledge still yields remarkable optimization
- 25% and even 10% a-priori-knowledge can lead to an apparent performance improvement

Related work

- Mitigating I/O contention is in active research
- However, many approaches either
  - (1) employ I/O scheduling
    - Scheduling under congestion (Gainaru et al.), CALCioM (Dorier et al.), IO-Sets (Boito et al.)
  - (2) require dedicated hardware
    - Burst-buffer enabled scheduling (Herbein et al.)
  - (3) interfere with job execution (Zhou et al.)
  - (4) require detailed a-priori I/O information
    - SchedP (Wu et al.)

- EquilibrIO is an I/O-aware *job scheduling* algorithm
  - No dedicated hardware requirements
  - No interference after job admission
  - Require none to minimal I/O information
- However, those approaches are not mutually exclusive, for example
  - EquilibrIO + I/O scheduling can operate alongside and potentially further improve I/O performance

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Conclusion & outlook

- EquilibrIO can reduce the median I/O slowdown from 64.0% to 3.6%, while still maintaining fairness at $\alpha = 0.5$
- Even limited a-priori-knowledge yields remarkable performance improvement
  - 25% already exploits half of the optimization potential
- Future work
  - Dynamic adaptation of the reordering intensity
  - Evaluating IOPS performance improvement

- Accepted paper: join us at the **IEEE CLUSTER 2025** conference
- Contact: Taylan Özden (taylan.oezden@tu-darmstadt.de)

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Thank you!

EuroHPC Joint Undertaking

With funding from the:

Federal Ministry of Research, Technology and Space

NHR4CES NHR for Computational Engineering Science