UCF | AREA 67

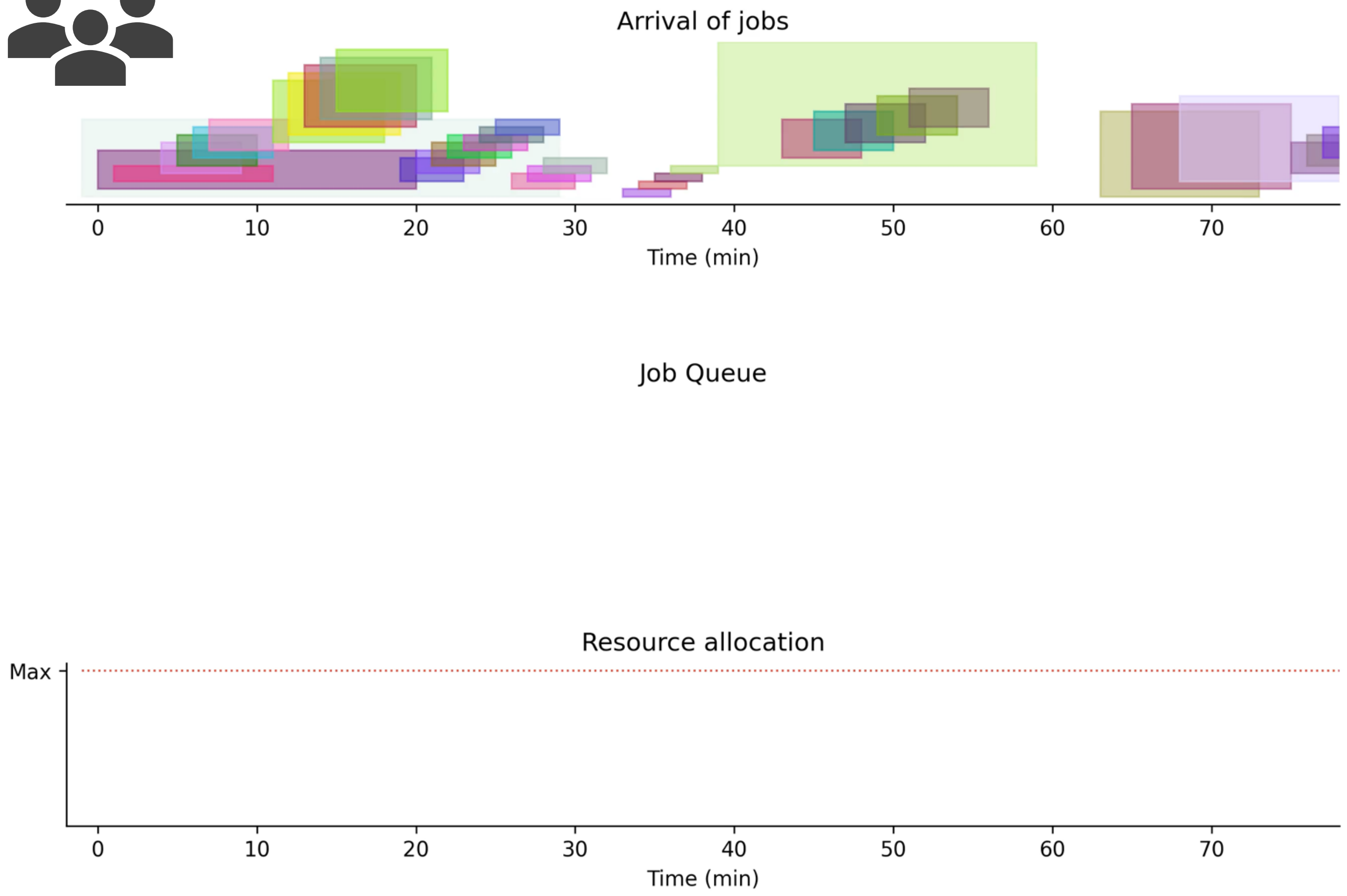
UNIVERSITY OF CENTRAL FLORIDA

Sandia National Laboratories

Workload-Adaptive Scheduling for Efficient Use of Parallel File Systems in HPC Clusters

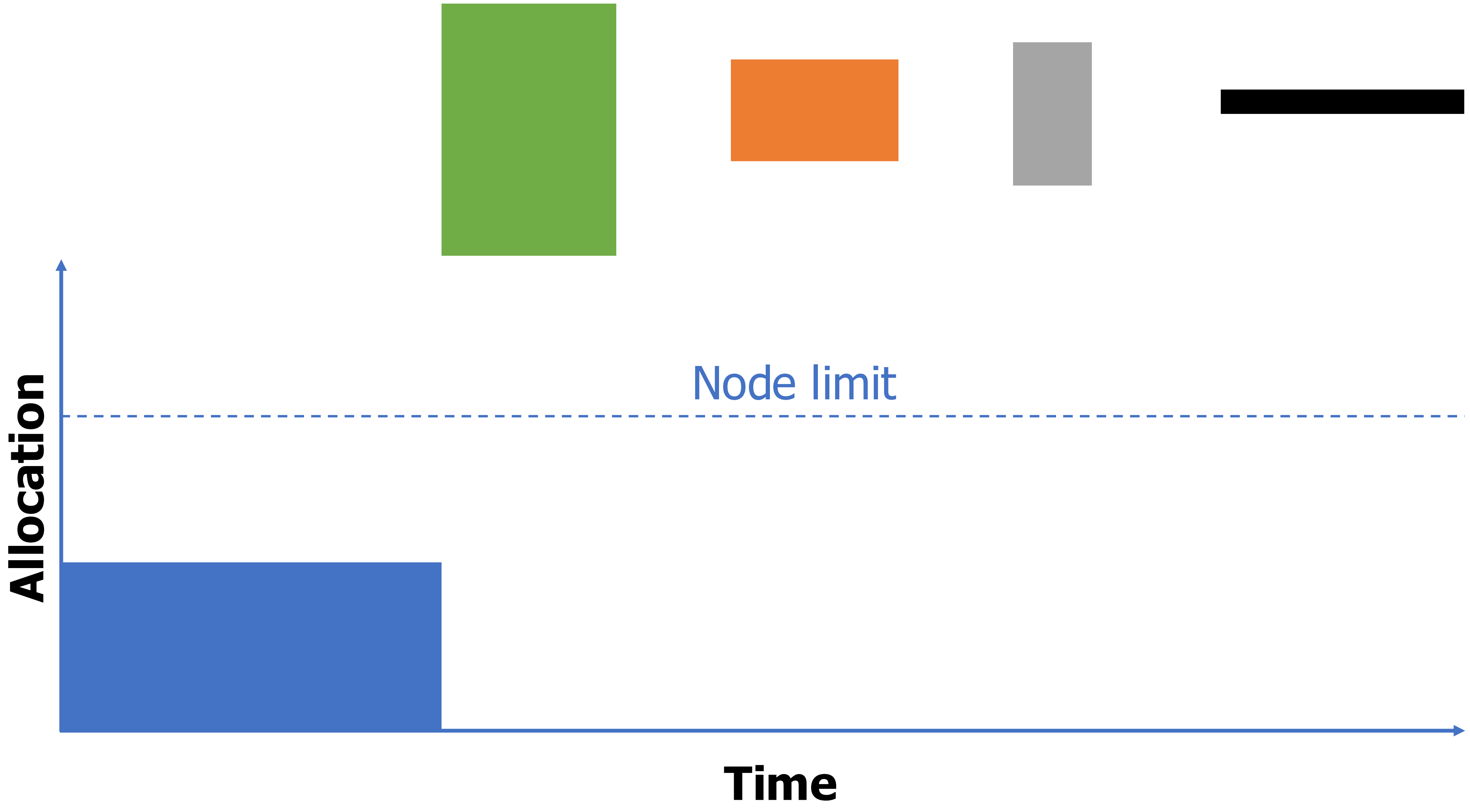
Alexander V. Goponenko (UCF), Benjamin A. Allan (SNL), James M. Brandt (SNL), and Damian Dechev (UCF)

Job scheduling on HPC cluster



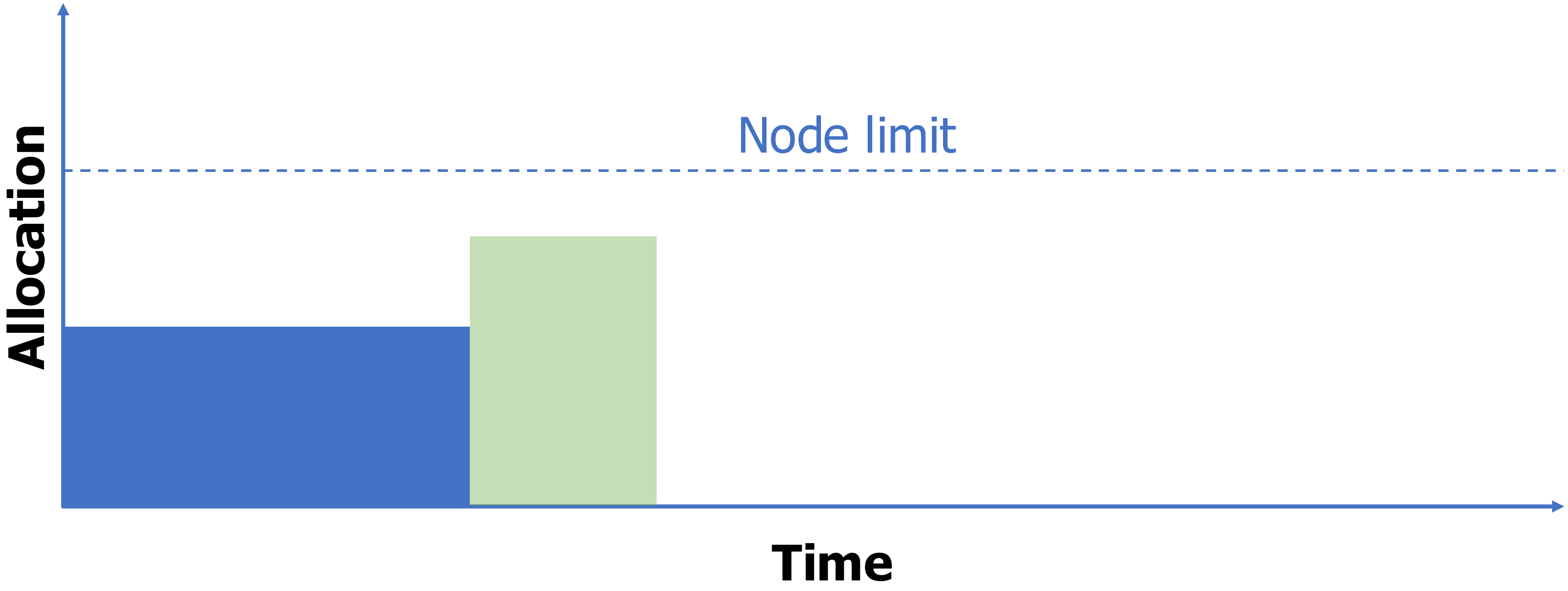
- Users submit jobs and specify jobs' resource requirements
  - Essential parameters:
    - Number of nodes
    - Time limit
- Scheduling algorithms determine the order of starting jobs during shortage of resources
- NP-hard problem

Scheduling with backfilling



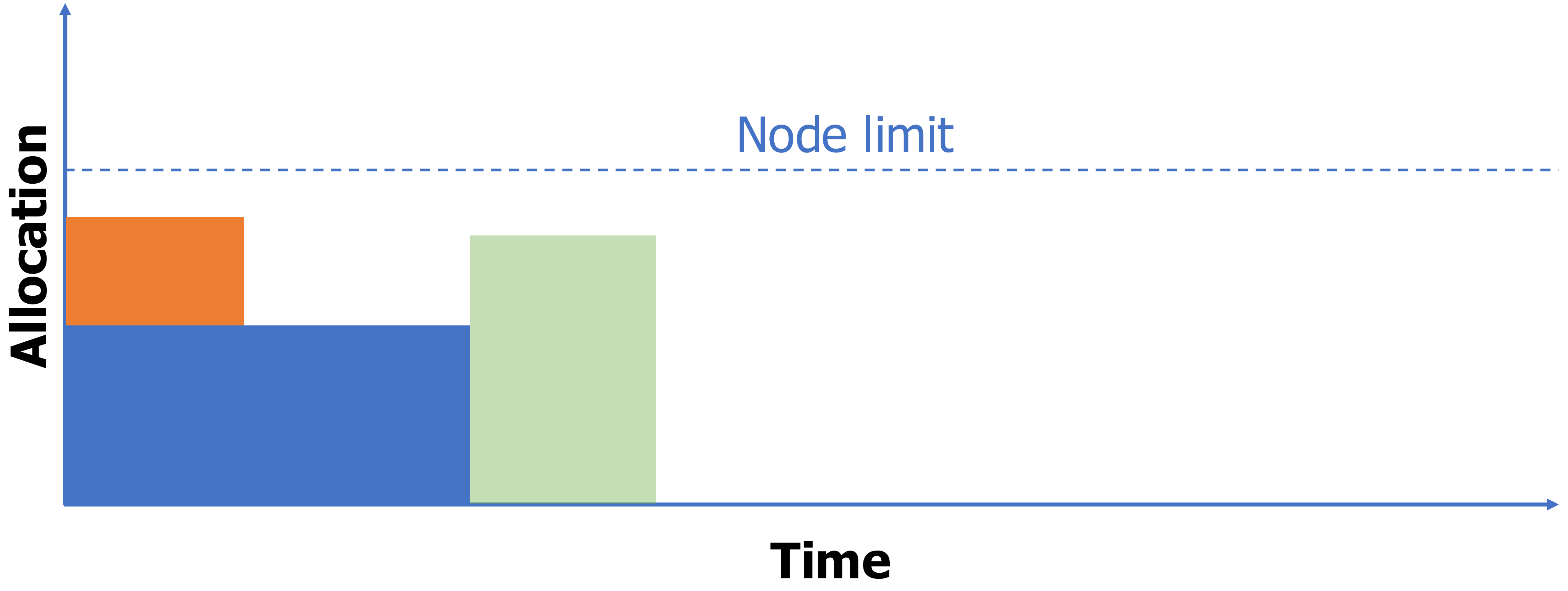
Scheduling with backfilling

- If a job cannot run,
  reserve resources
  to prevent any further delays



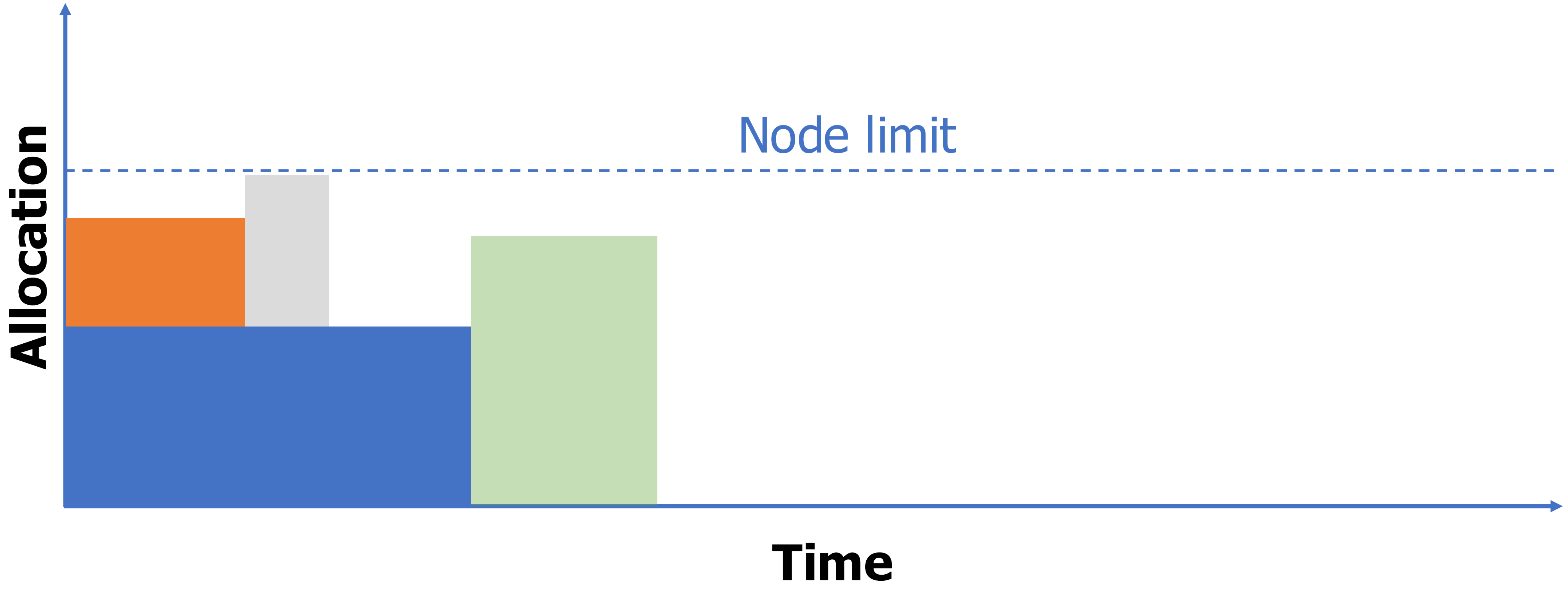
Scheduling with backfilling

- If a job cannot run,
  reserve resources
  to prevent any further delays



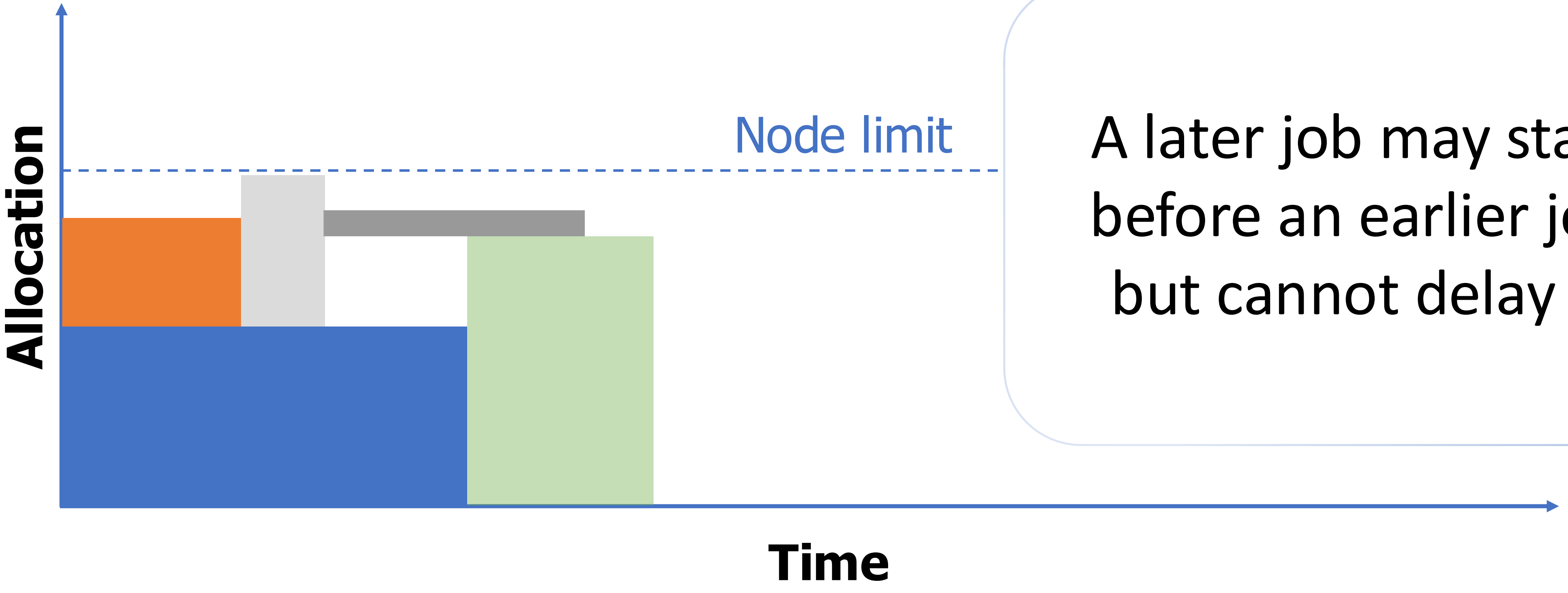
Scheduling with backfilling

- If a job cannot run,
  reserve resources
  to prevent any further delays



Scheduling with backfilling

- If a job cannot run,
  reserve resources
  to prevent any further delays

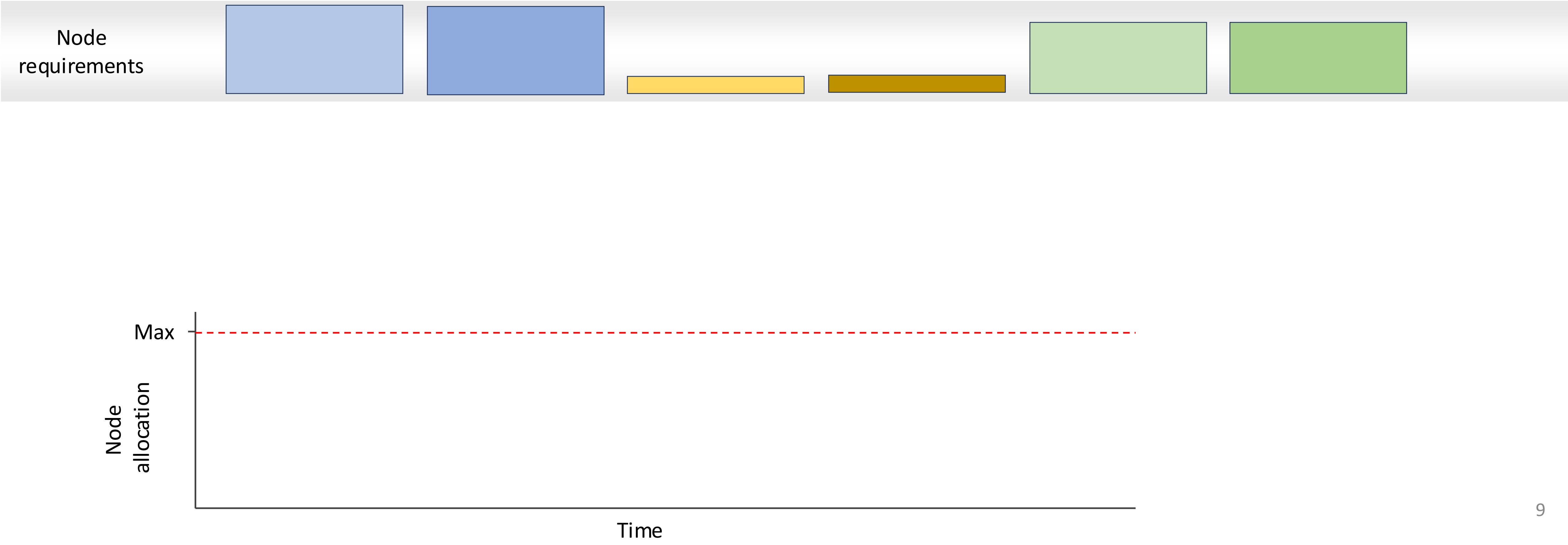


A later job may start before an earlier job but cannot delay it

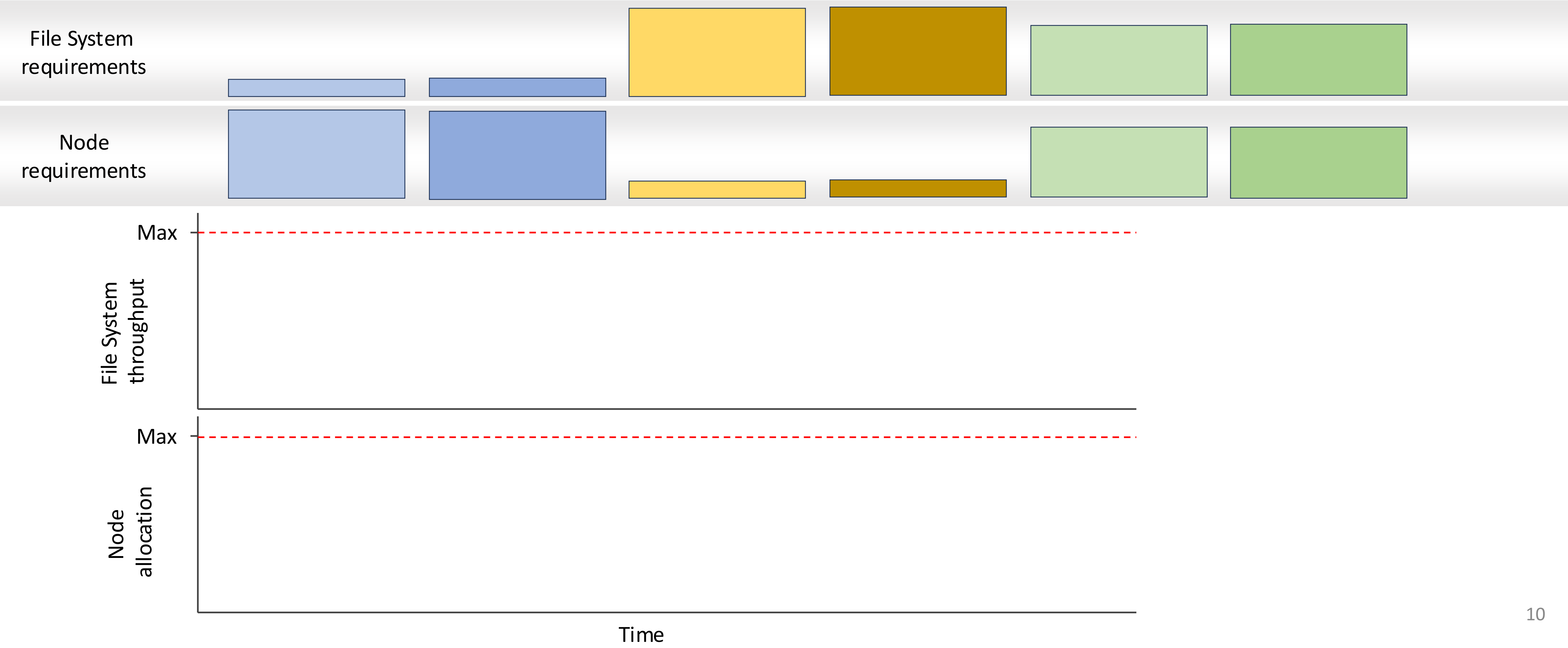
Modern HPC systems require multiple-resource scheduling

- Modern system are complicated
  - Many resources (burst buffers, GPU, etc.)
  - I/O bottlenecks (network, parallel file system)
- Modern schedulers should –
  - – schedule jobs
    - aiming at improving efficiency
    - accounting for user policy considerations
  - – handle various resource constraints
  - – reduce user's burden to provide resource requirements
  - – anticipate that jobs' runtime and resource usage may depend on how the jobs are scheduled

Single-resource scheduling



Multi-resource scheduling



“Common” I/O-aware scheduling

- Key features of “common” I/O-aware scheduling[1]
  - The scheduler estimates the file system throughput of the jobs
  - The scheduler doesn’t let the total estimated throughput to exceed the file system bandwidth
- Our implementation can be configured to perform “common” I/O-aware scheduling and workload-adaptive scheduling

[1] M. R. Wyatt, S. Herbein, T. Gamblin, and M. Taufer, “AI4IO: A suite of AI-based tools for IO-aware scheduling,” *Int. J. High Perform. Comput. Appl.* 2022, doi: 10.1177/10943420221079765.

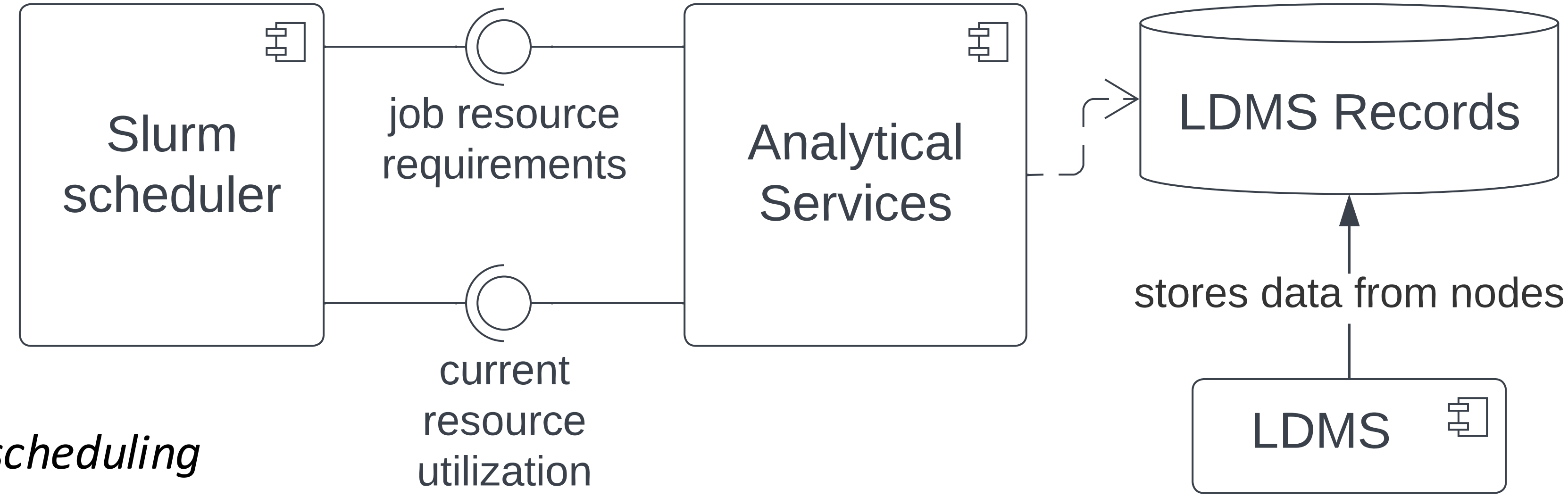
Outline

Overview of our scheduling system

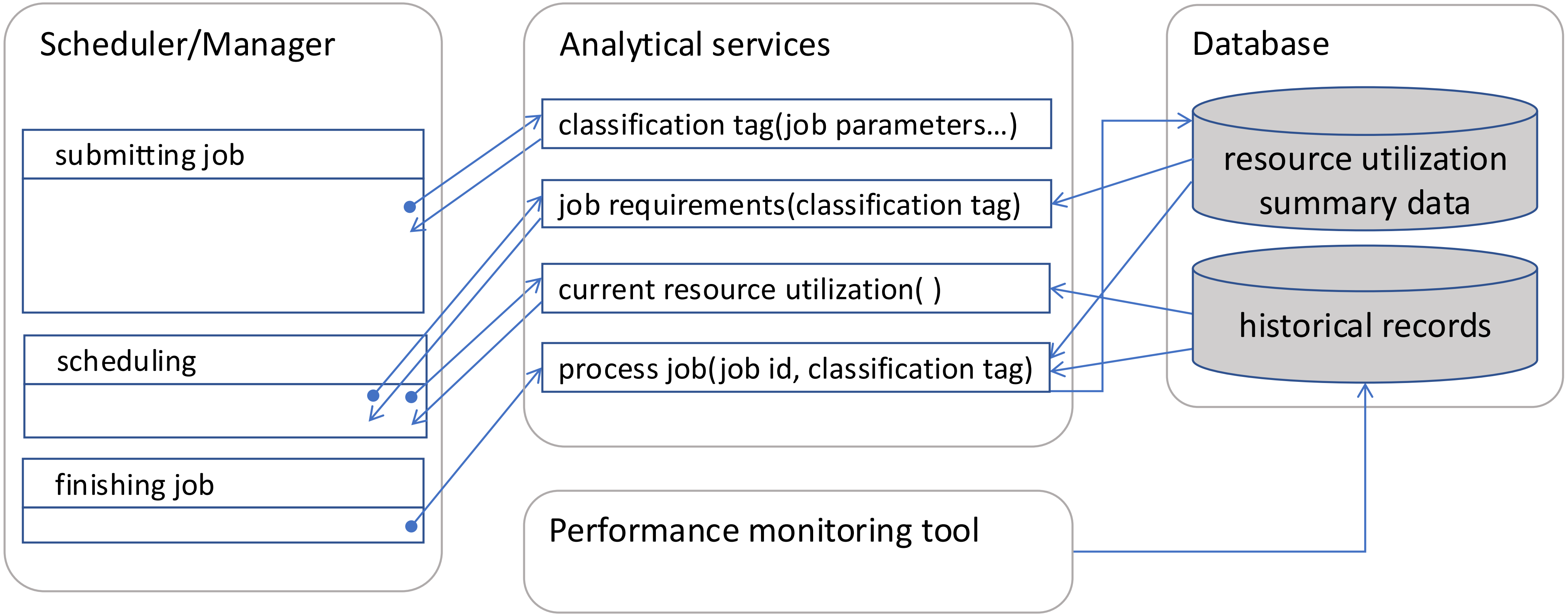
Shortcomings of "common" I/O-aware scheduling

Workload-adaptive scheduling

Overview of our system

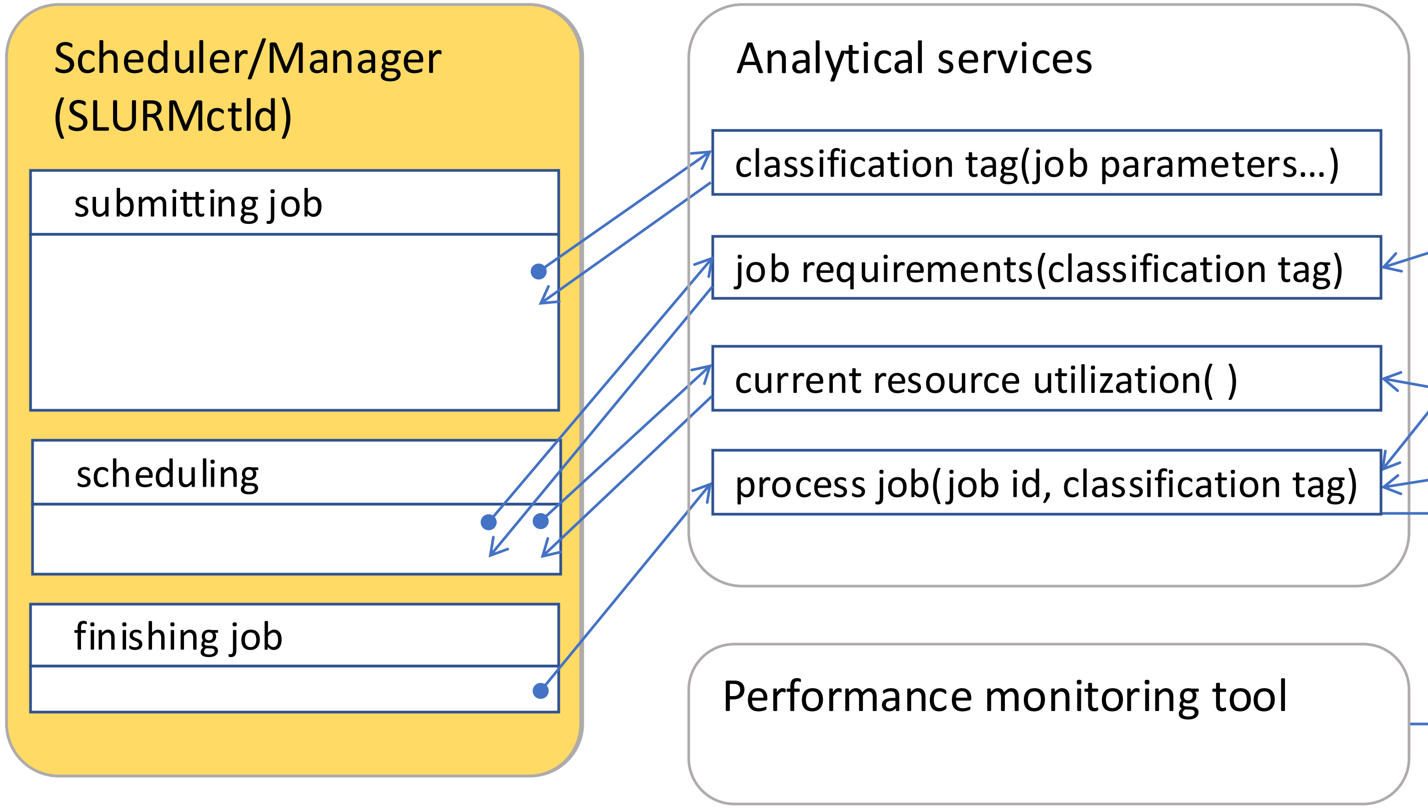


- Slurm – *scheduling*
  - https://slurm.schedmd.com/
  - https://github.com/algo74/slurm/tree/workload-adaptive-paper-2024
- Analytical services – *estimating resource requirements*
  - https://github.com/algo74/py-sim-serv/tree/workload-adaptive-paper-2024
- Lightweight Distributed Metric Service (**LDMS**) – *measuring resource usage*
  - https://github.com/ovis-hpc/ldms


Scheduler/Manager
submitting job
scheduling
finishing job
Analytical services
classification tag(job parameters...)
job requirements(classification tag)
current resource utilization( )
process job(job id, classification tag)
Database
resource utilization summary data
historical records
Performance monitoring tool

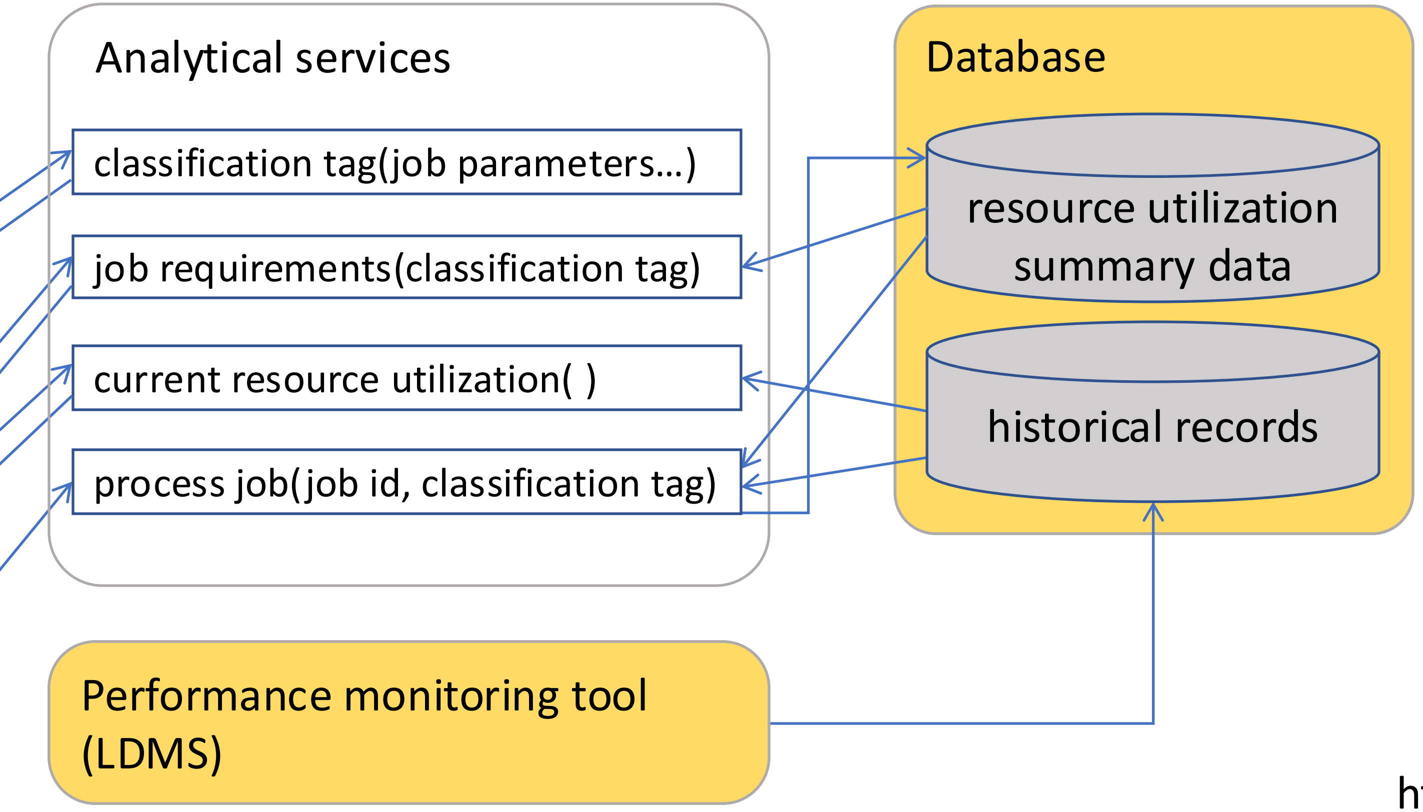
Scheduler

- Request resource requirements for jobs
- Request real-time utilization of the resources
- Take into account the obtained values during scheduling
- Set job ID on the nodes
- Send a signal when a job is completed



https://github.com/algo74/slurm/tree/workload-adaptive-paper-2024

Measuring resource usage

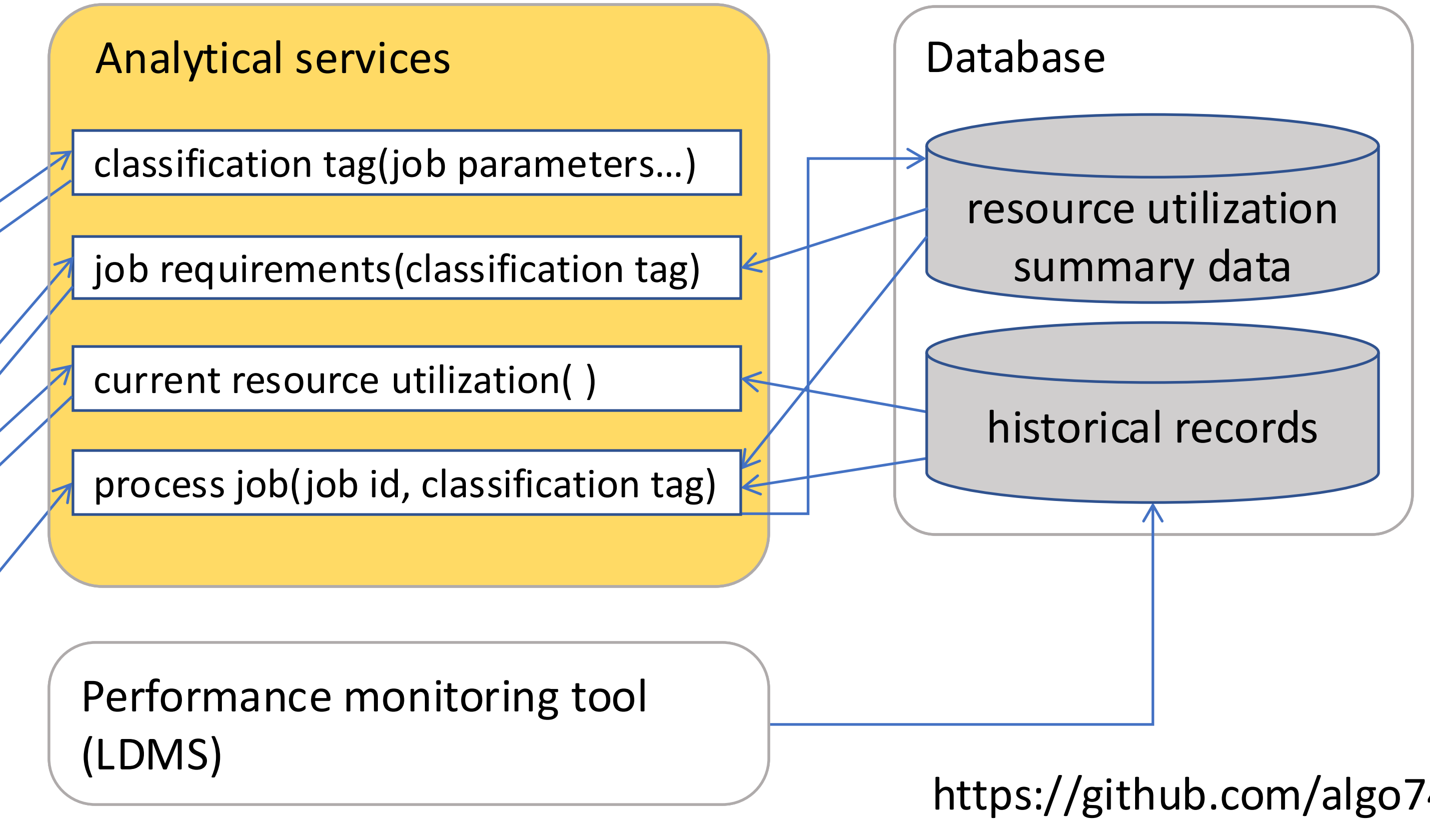


- Lightweight Distributed Metric Service (**LDMS**)
  - No modification
  - jobinfo plugin associates records with jobs

https://github.com/ovis-hpc/ldms

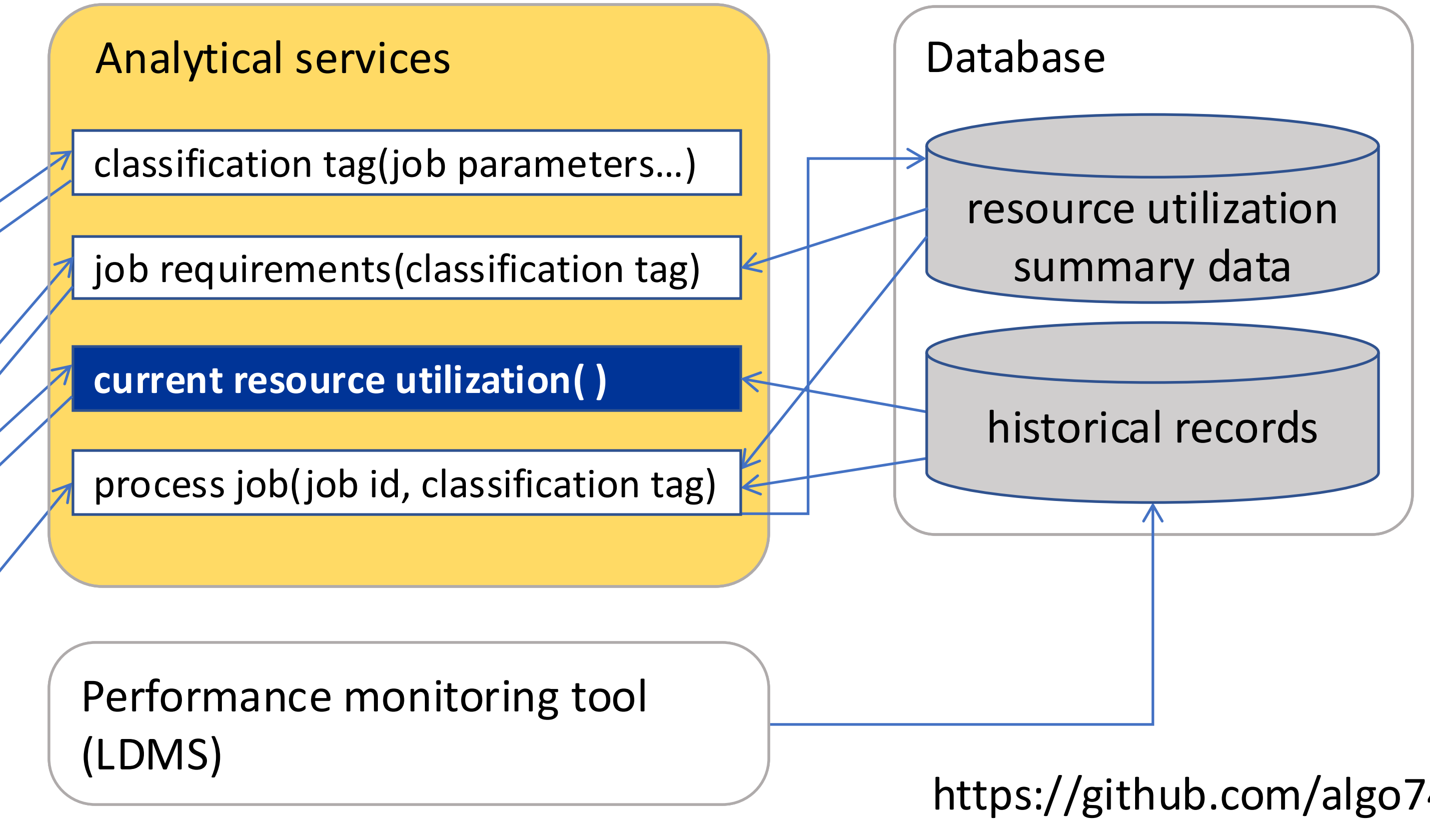
Analytical services

- Prediction of jobs’ resource requirements
  - *classification tag(…)*
  - *job requirements(…)*
  - *process job(…)*
- Real-time utilization of resources
  - *current resource utilization(…)*



https://github.com/algo74/py-sim-serv/tree/workload-adaptive-paper-2024

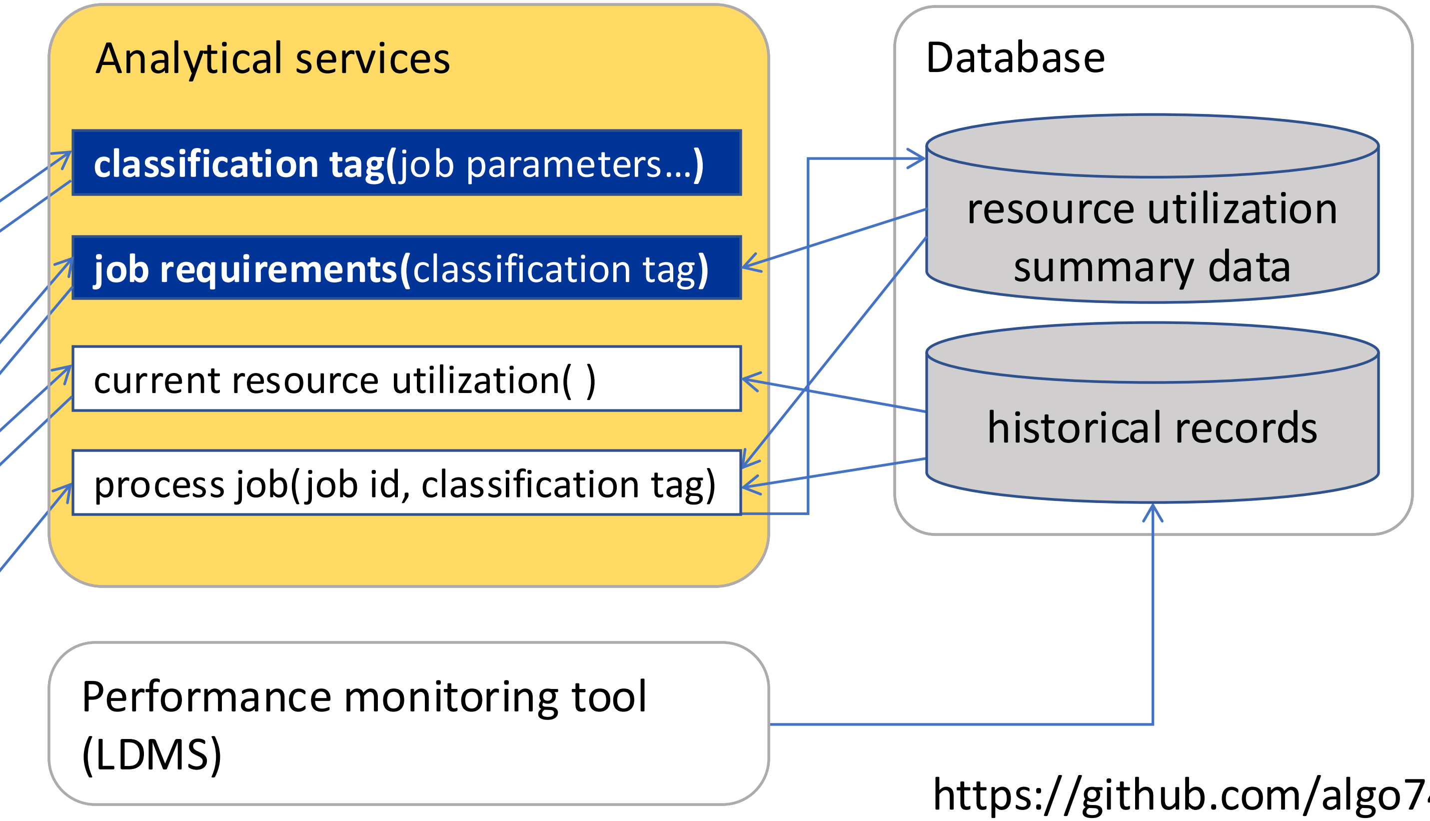
Real-time utilization of resources



- Periodically retrieve recent LDMS records
- Calculate whole-system utilization of the resources
- Fulfill requests with the latest value

https://github.com/algo74/py-sim-serv/tree/workload-adaptive-paper-2024

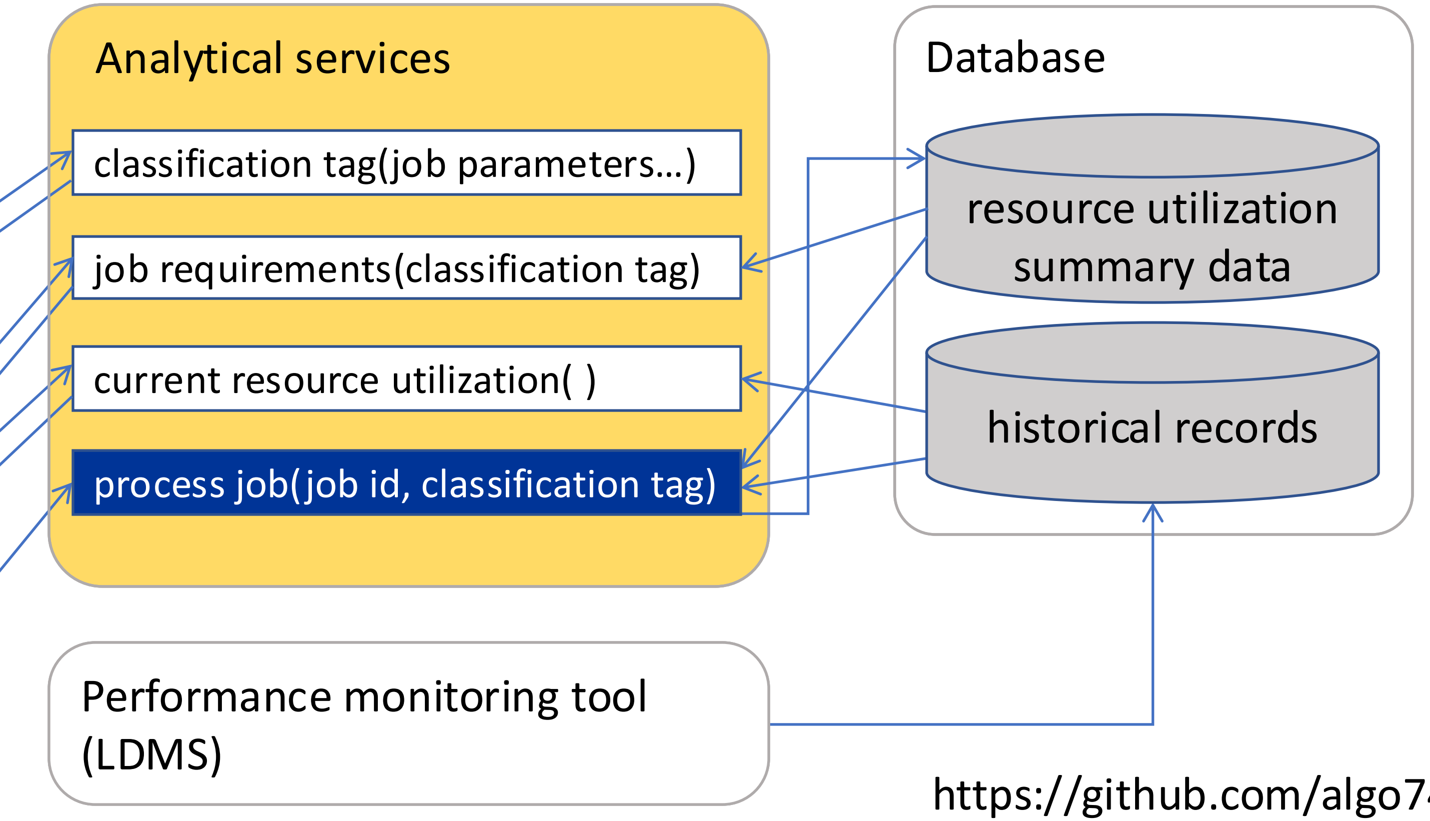
Predicting jobs' resource requirements



- **Step 1:** Classify the job into a group based on the job's parameters, *e.g.*
  - User ID
  - Job type
  - Script name
  - User-specified timelimit
  - Requested number of nodes
- **Step 2:** Retrieve the most recent estimate for the group
  - *We maintain estimates of the groups in a database*

https://github.com/algo74/py-sim-serv/tree/workload-adaptive-paper-2024

Processing finished jobs



- Retrieve LDMS records for the job
- Calculate the average usage and the variance for the job
- Recalculate and update the estimates for the group
  - Exponentially weighted moving average

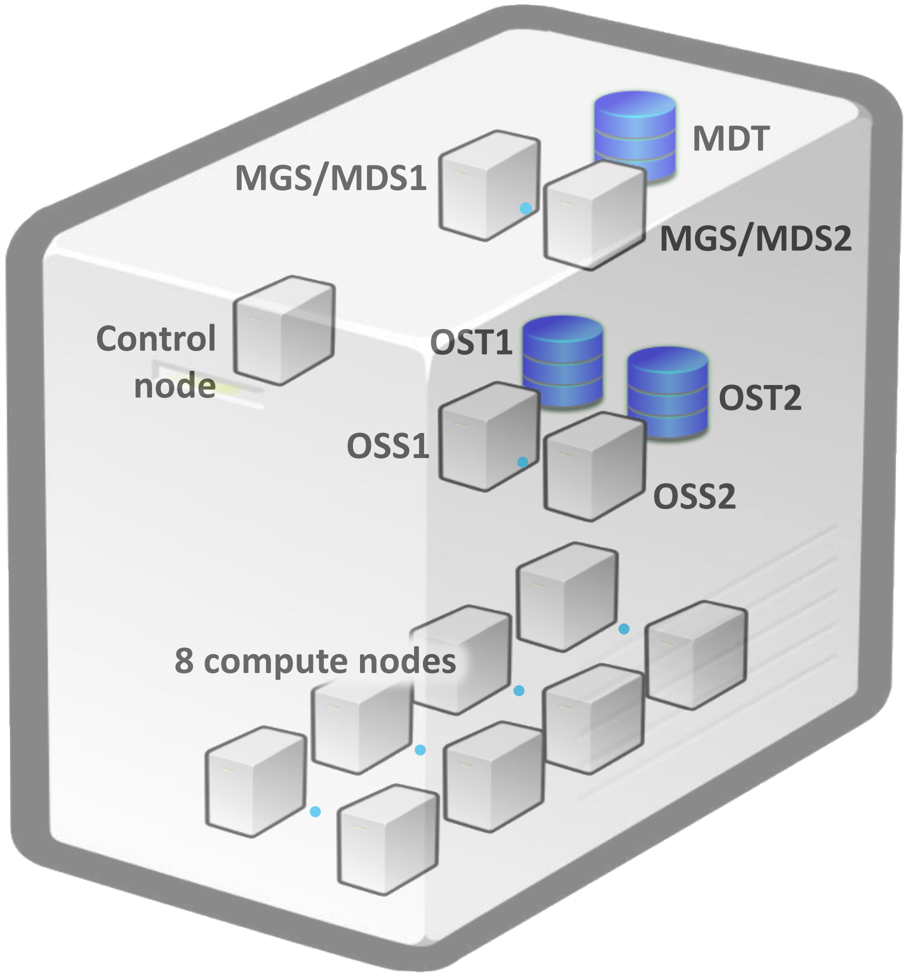
https://github.com/algo74/py-sim-serv/tree/workload-adaptive-paper-2024

Shortcomings of "common" I/O-aware scheduling

- Using "bandwidth" as the throughput limit not necessarily leads to the best performance
- Susceptibility to errors in estimating resource requirements
- May increase idling of in-demand resources

Cluster of virtual machines

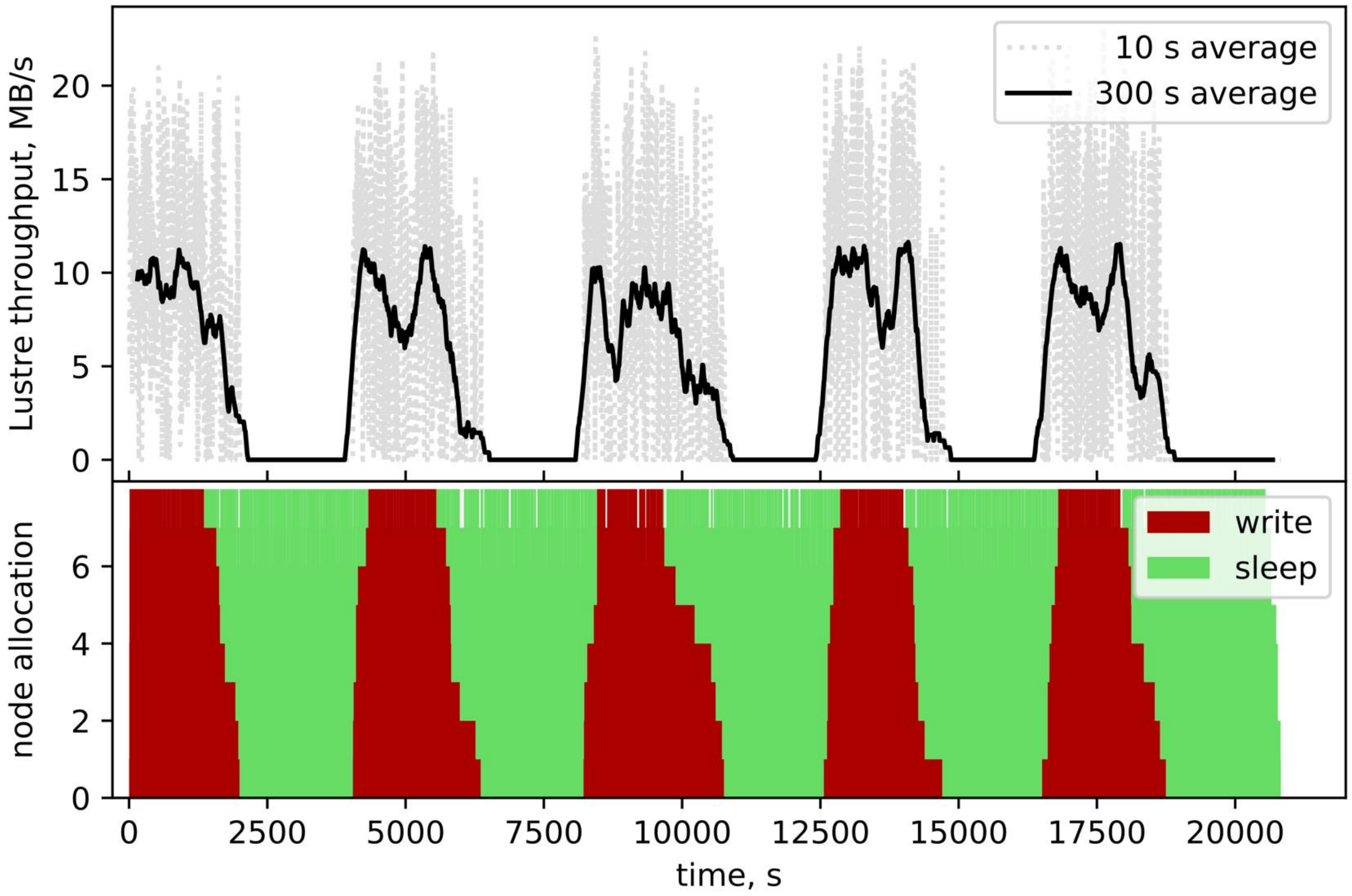
- Host
  - Intel Xeon E5-2697 v2 (12 cores, 2.70 GHz)
  - 64 GiB RAM
- Lustre
  - 2 MGS/MDS
  - 2 OSS
- Nodes
  - 1 control node
  - 8 compute nodes



UCF

Performance of I/O-aware scheduling:
Job queue 1

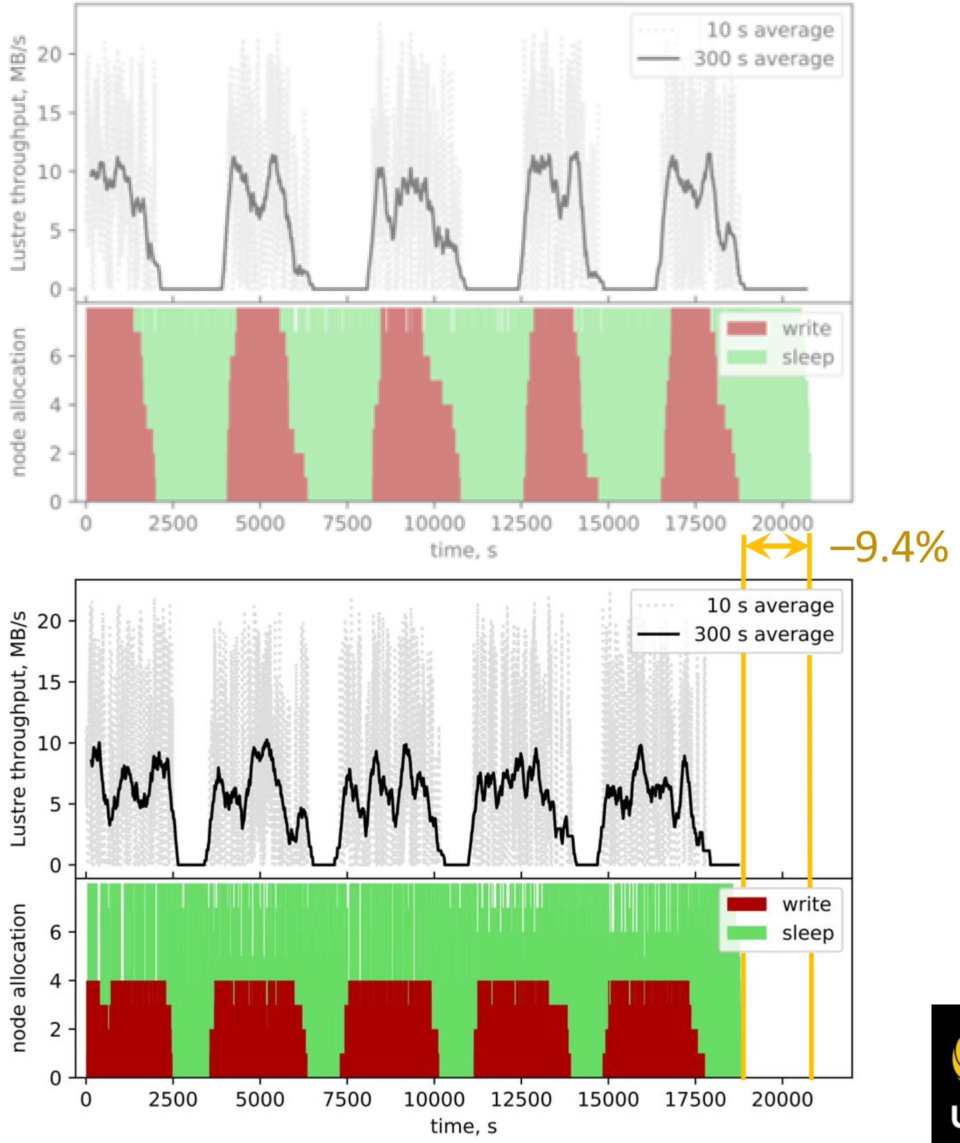
- Periodical pattern:
  5 waves of "write" and "sleep" jobs
- Default Slurm scheduler
  - Start the jobs in order they appear in the queue



Performance of I/O-aware scheduling:
Job queue 1

- Default Slurm scheduler

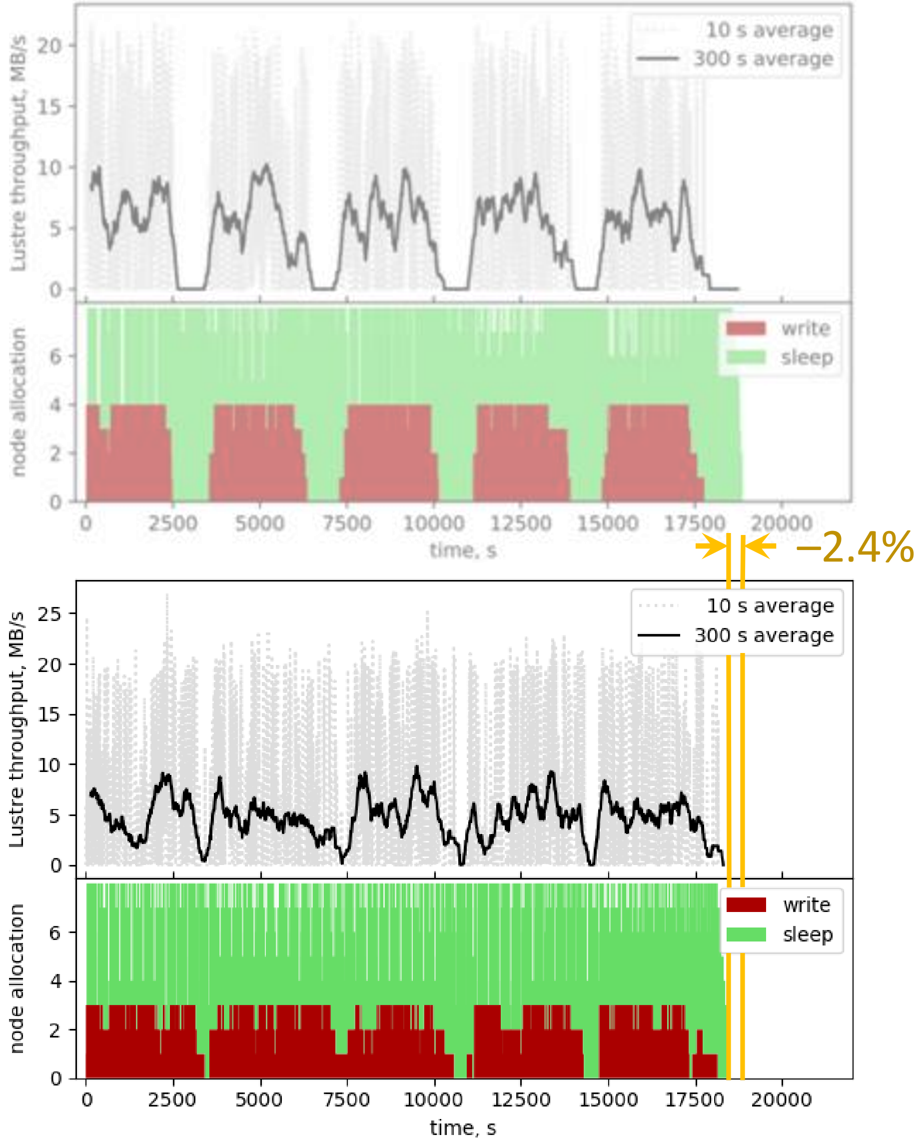
- I/O-aware scheduling
  - 4 "write" jobs simultaneously
  - 9.4% faster



Performance of I/O-aware scheduling:
Job queue 1

- 4 “write” jobs simultaneously

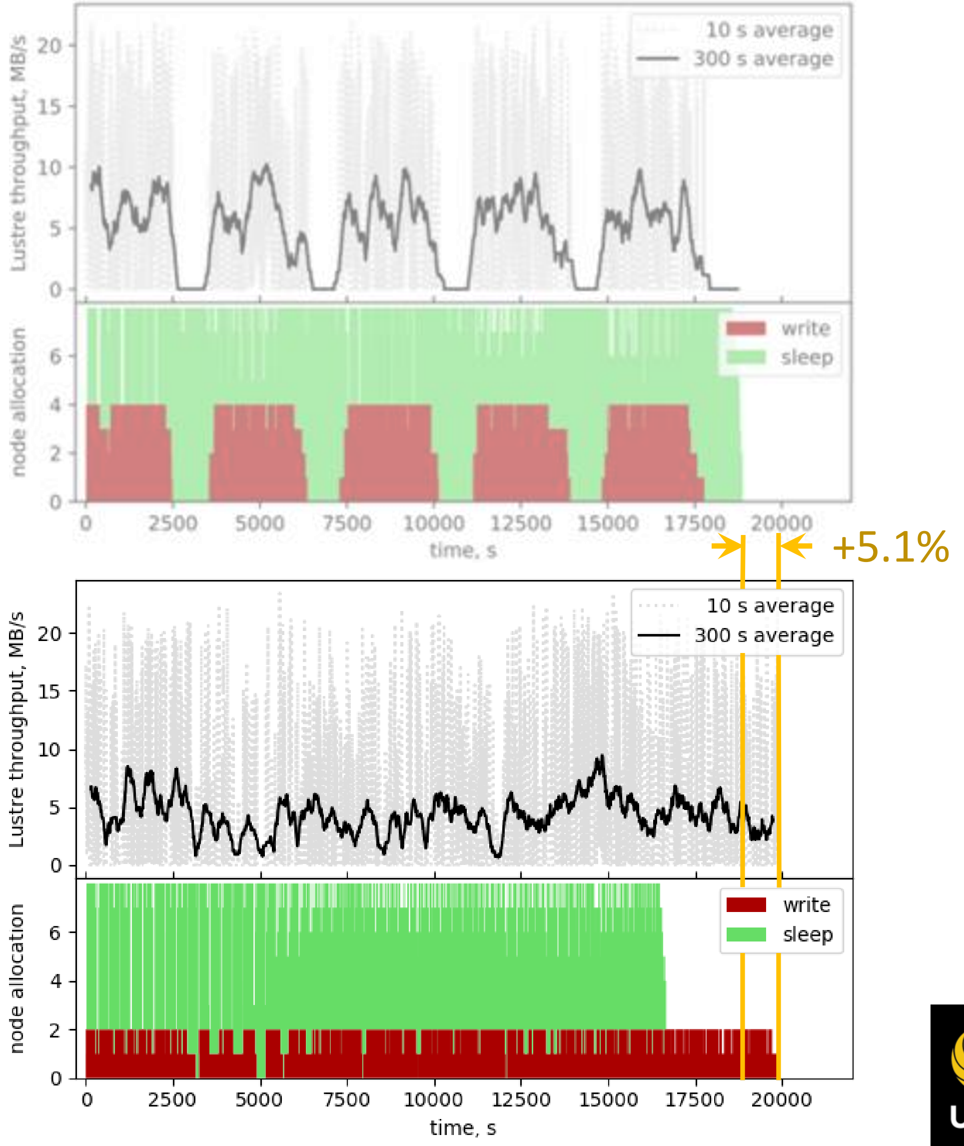
- 3 “write” jobs simultaneously (additional 2.4% speedup)



UCF

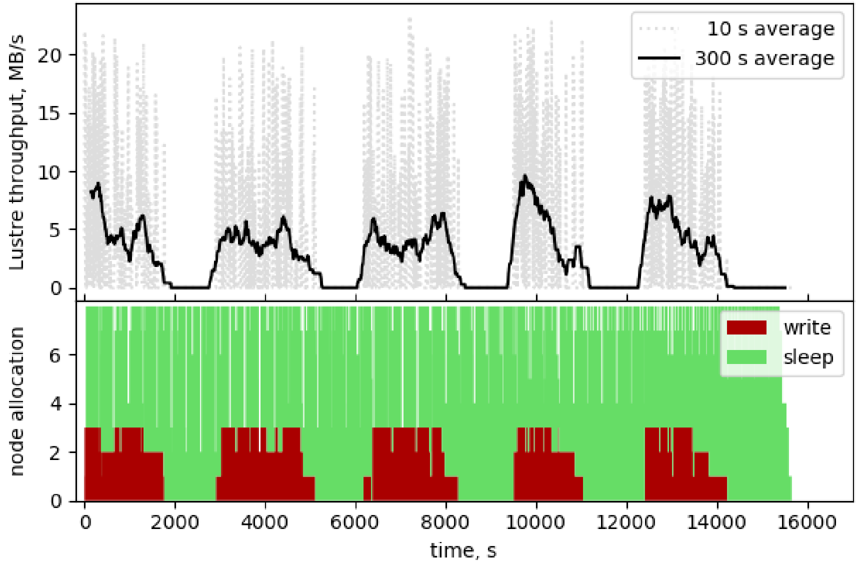
Performance of I/O-aware scheduling:
Job queue 1

- 4 “write” jobs simultaneously
- 2 “write” jobs simultaneously (decrease in performance)

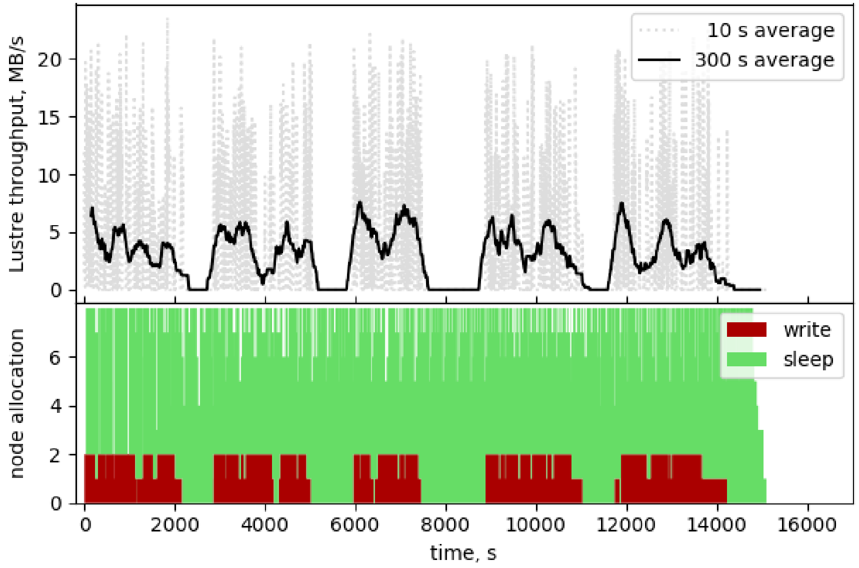


Performance of I/O-aware scheduling:
Job queue 2 (half the ”write” jobs)

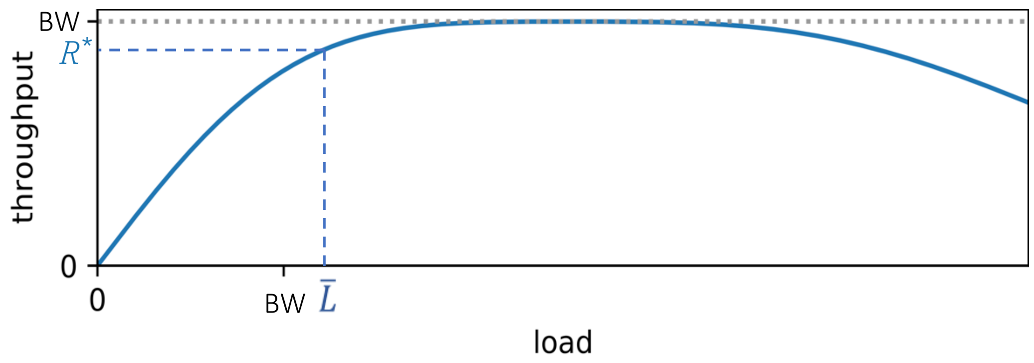
- 3 “write” jobs simultaneously

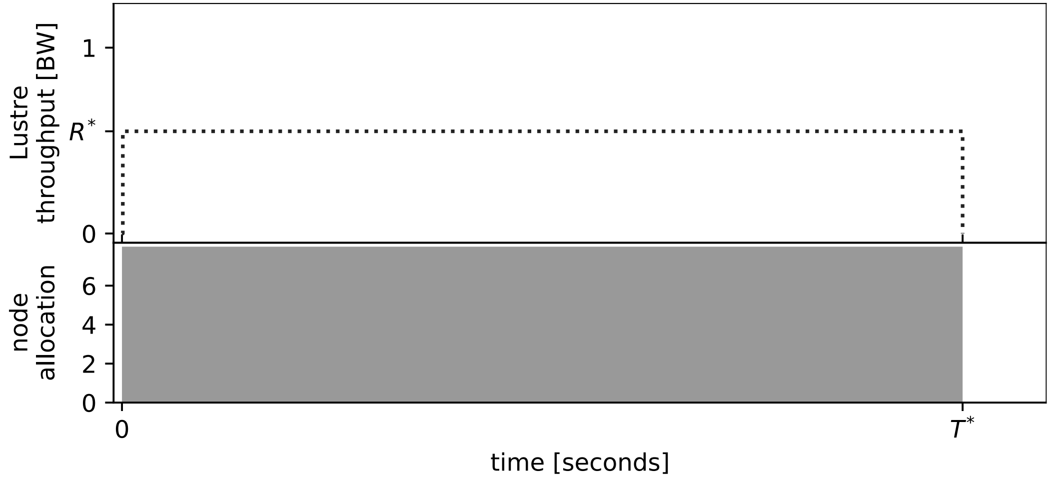


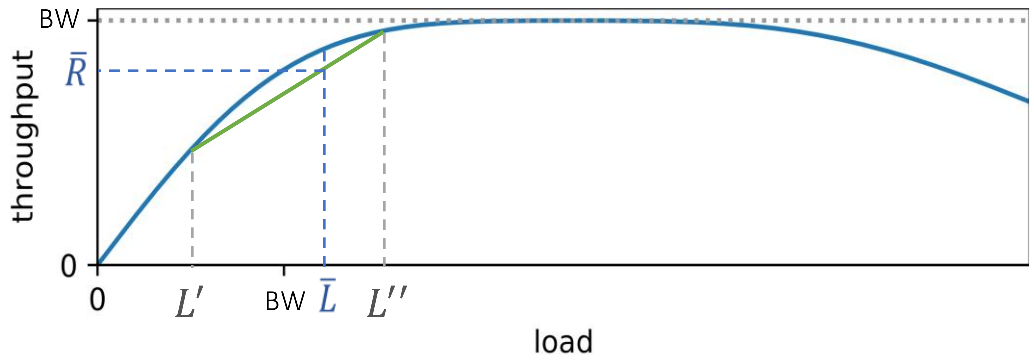
- 2 “write” jobs simultaneously (best performance)

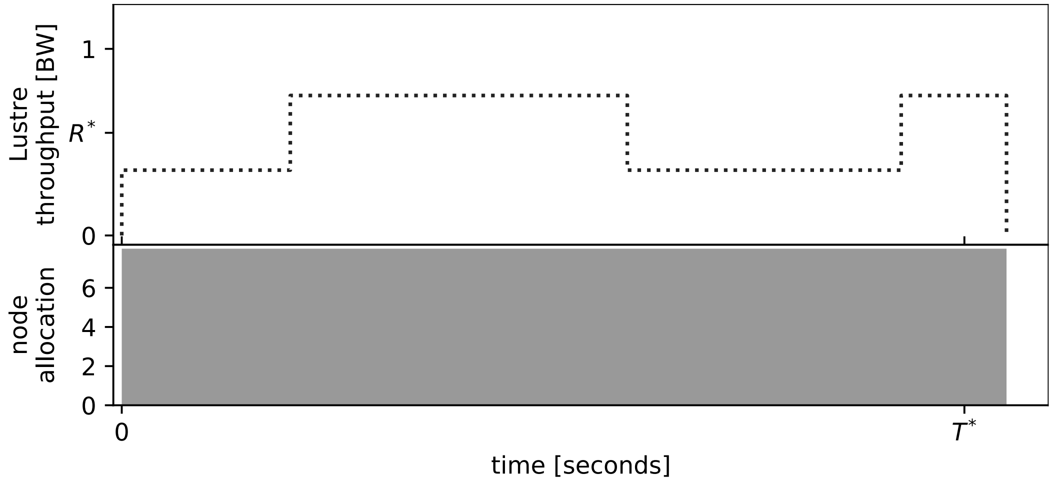


Stable load is optimal

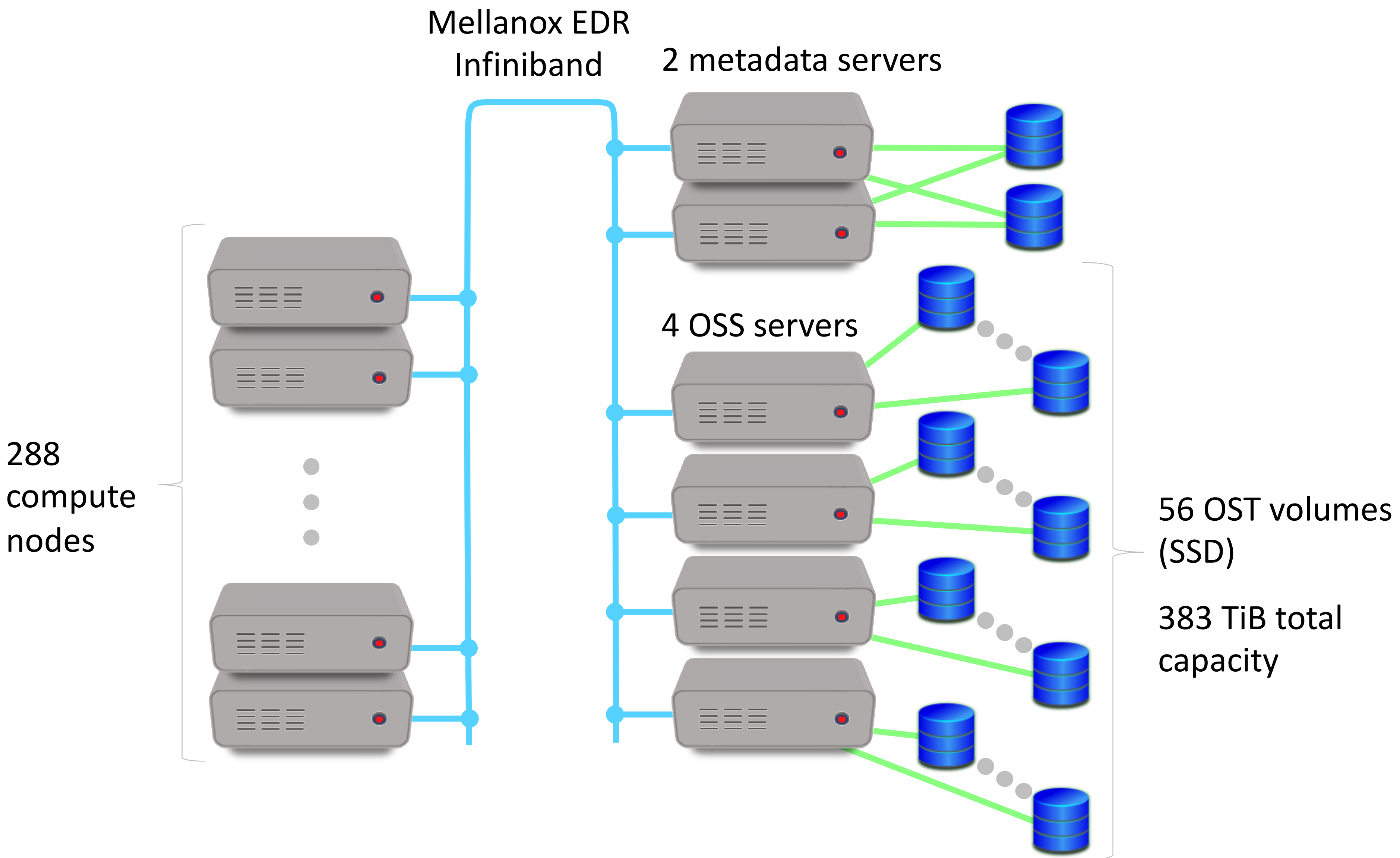








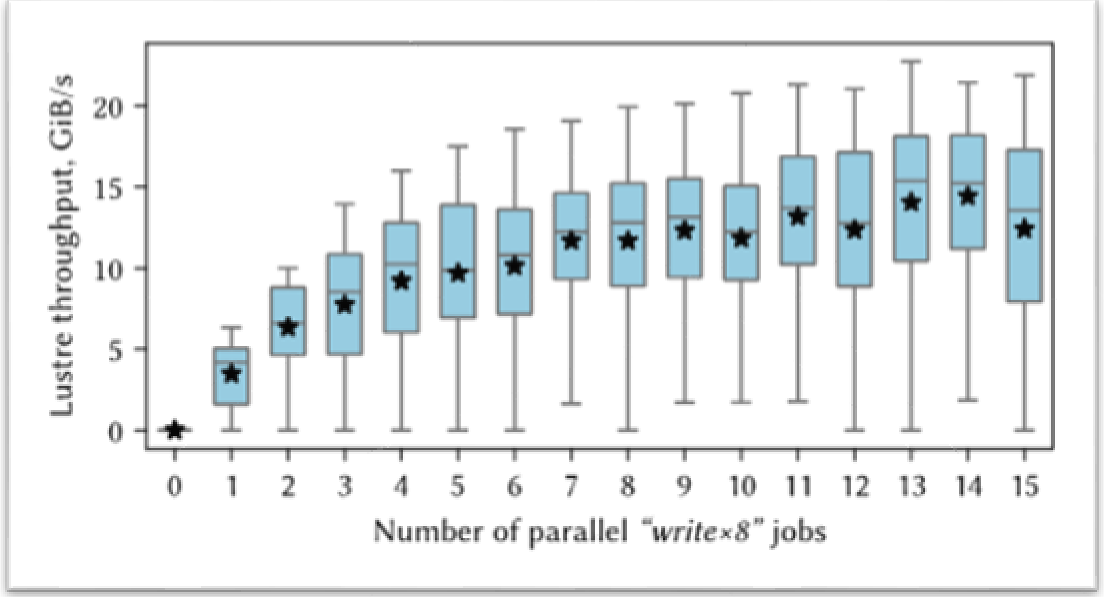
HPC cluster

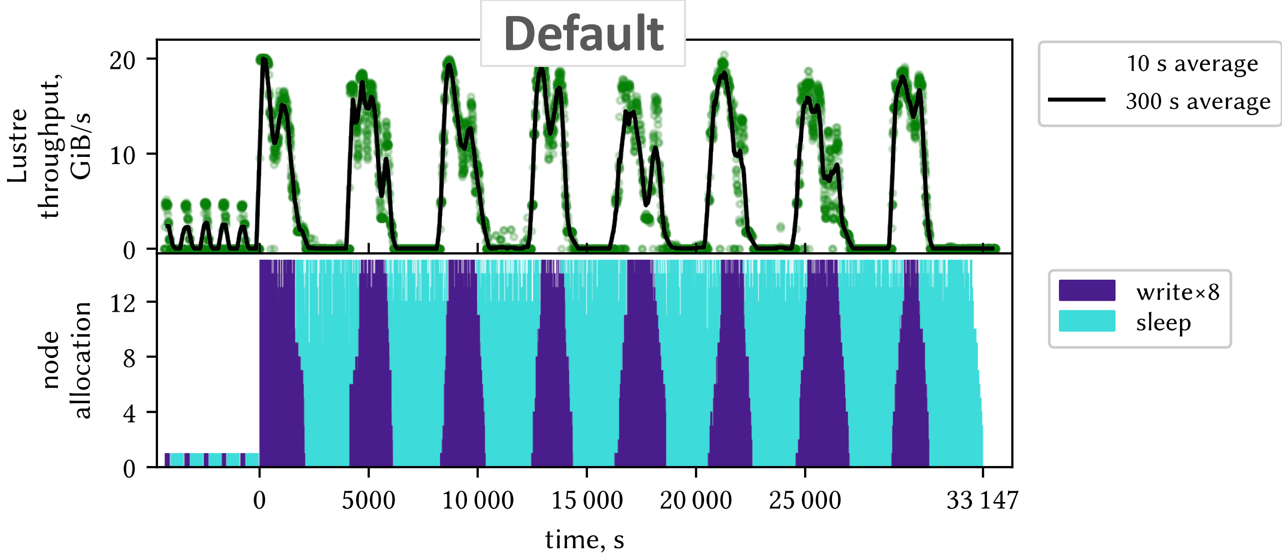


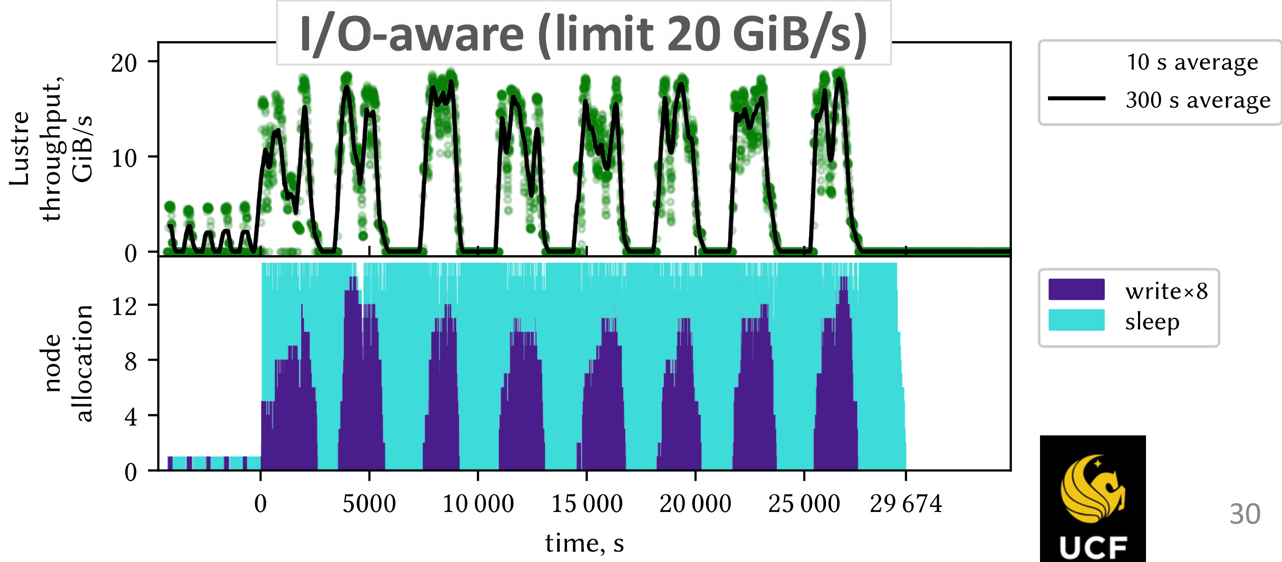
time, s

Robustness of I/O-aware scheduling: *"Workload 1"* (HPC cluster)

- Periodical pattern (8×)
  - 30 "write×8" jobs
  - 60 "sleep" jobs
- Bandwidth is 15-20 GiB/s

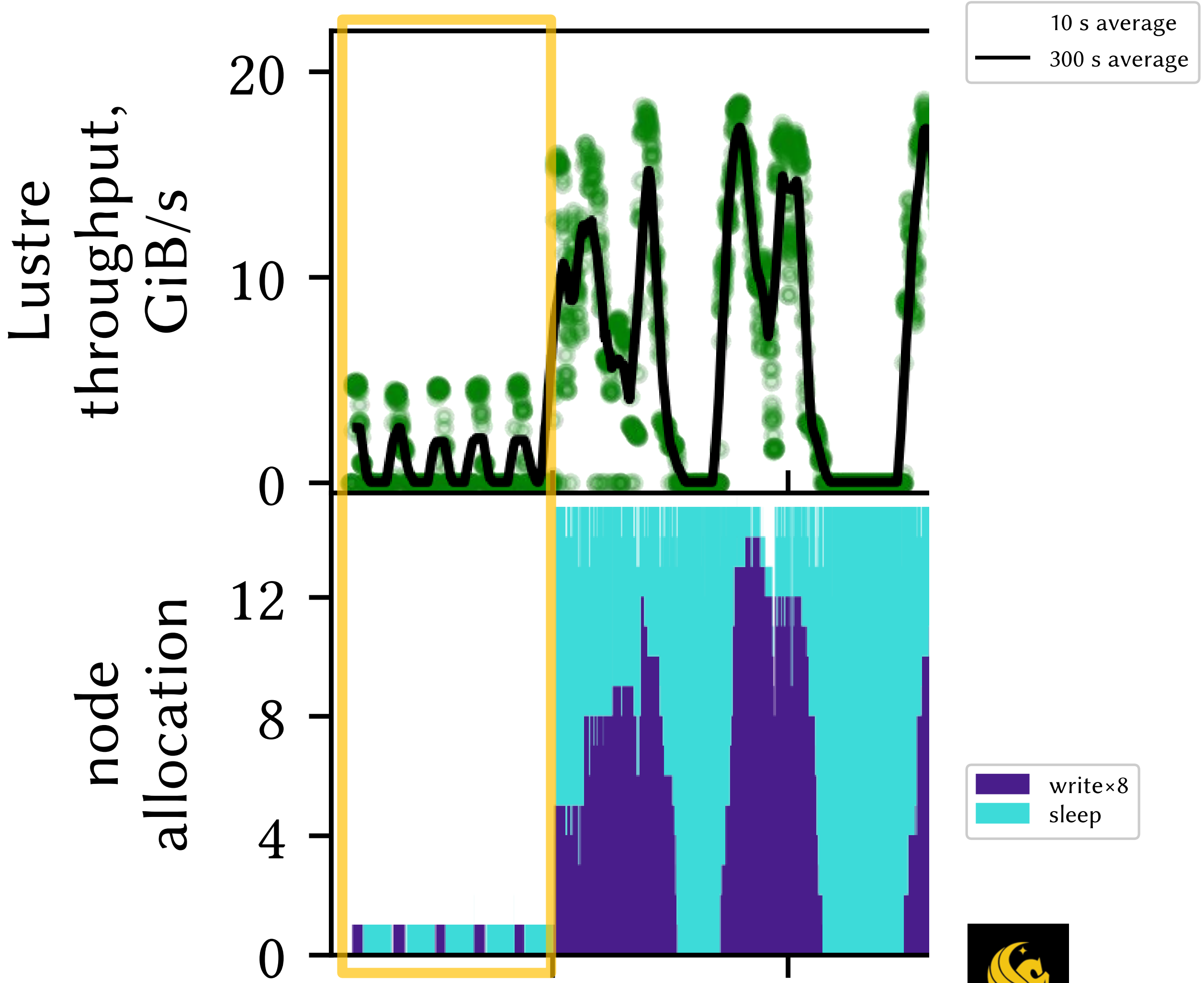






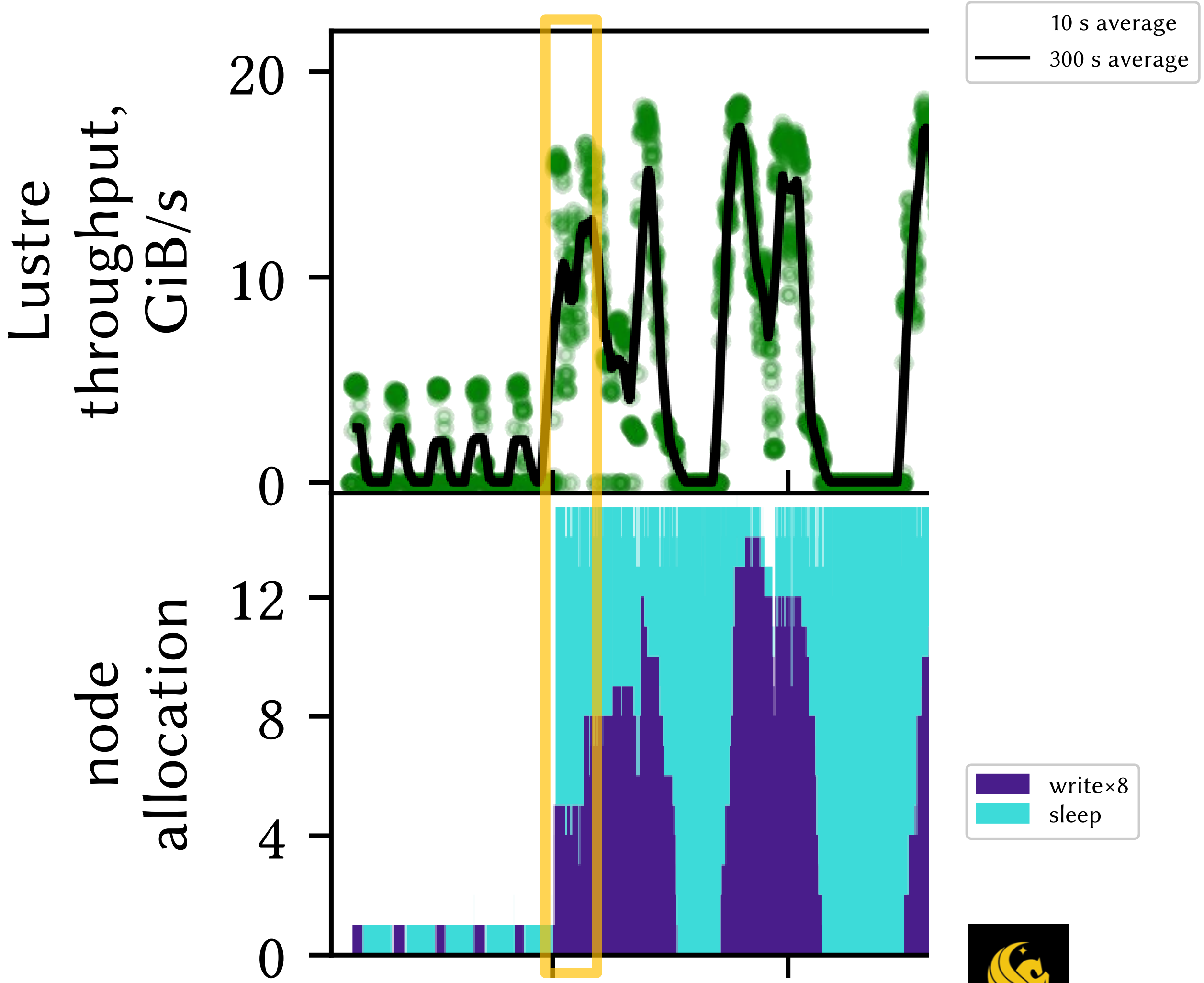
Robustness of I/O-aware scheduling:
"Workload 1" (HPC cluster)

- Periodical pattern (8×)
  - 30 "write×8" jobs
  - 60 "sleep" jobs
- Bandwidth is 15-20 GiB/s
- The estimator is pre-trained by running jobs in isolation



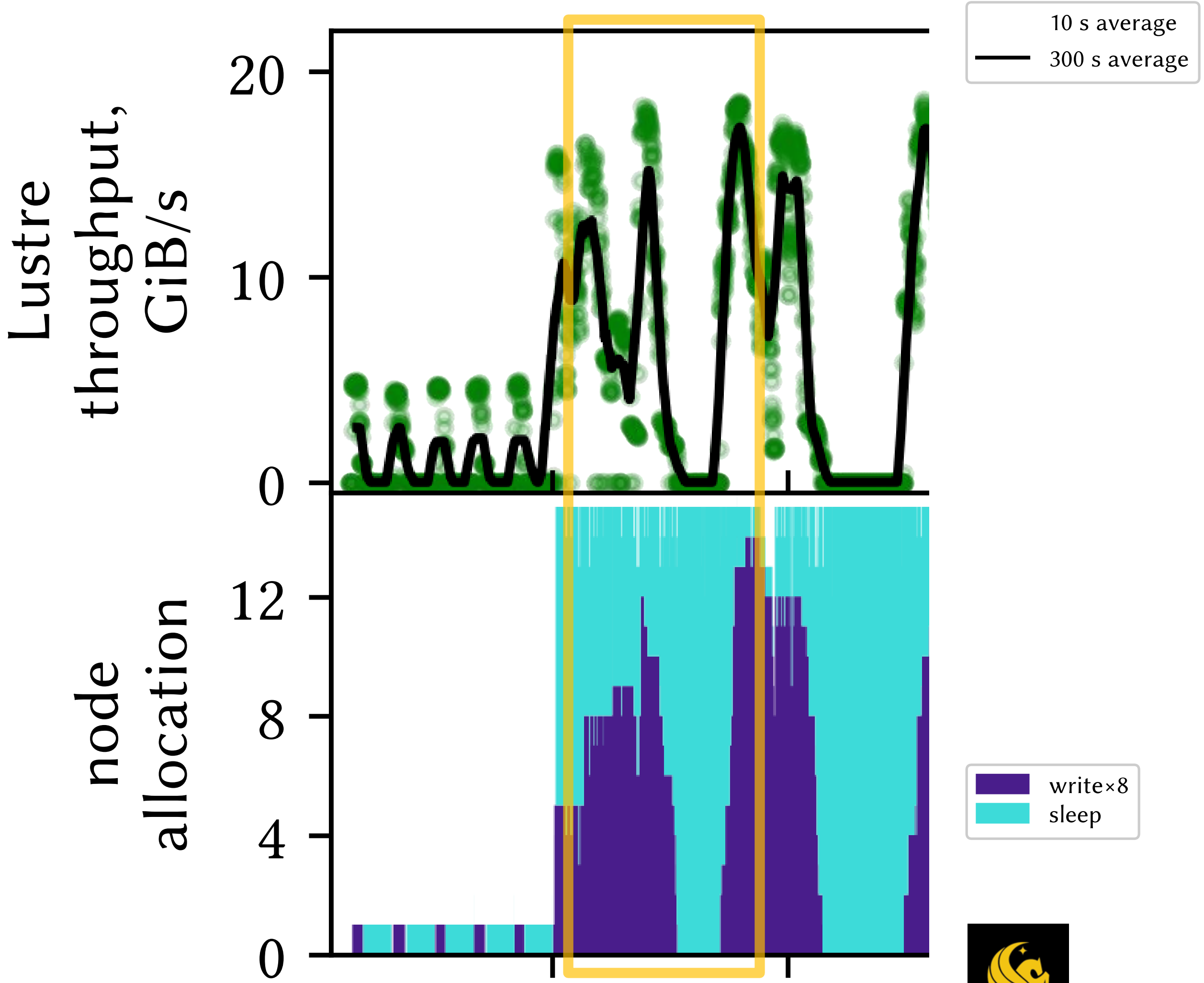
Robustness of I/O-aware scheduling:
"Workload 1" (HPC cluster)

- Periodical pattern (8×)
  - 30 "write×8" jobs
  - 60 "sleep" jobs
- Bandwidth is 15-20 GiB/s
- The estimator is pre-trained by running jobs in isolation
- I/O-aware scheduler initially schedules no more than 5 "write×8" jobs



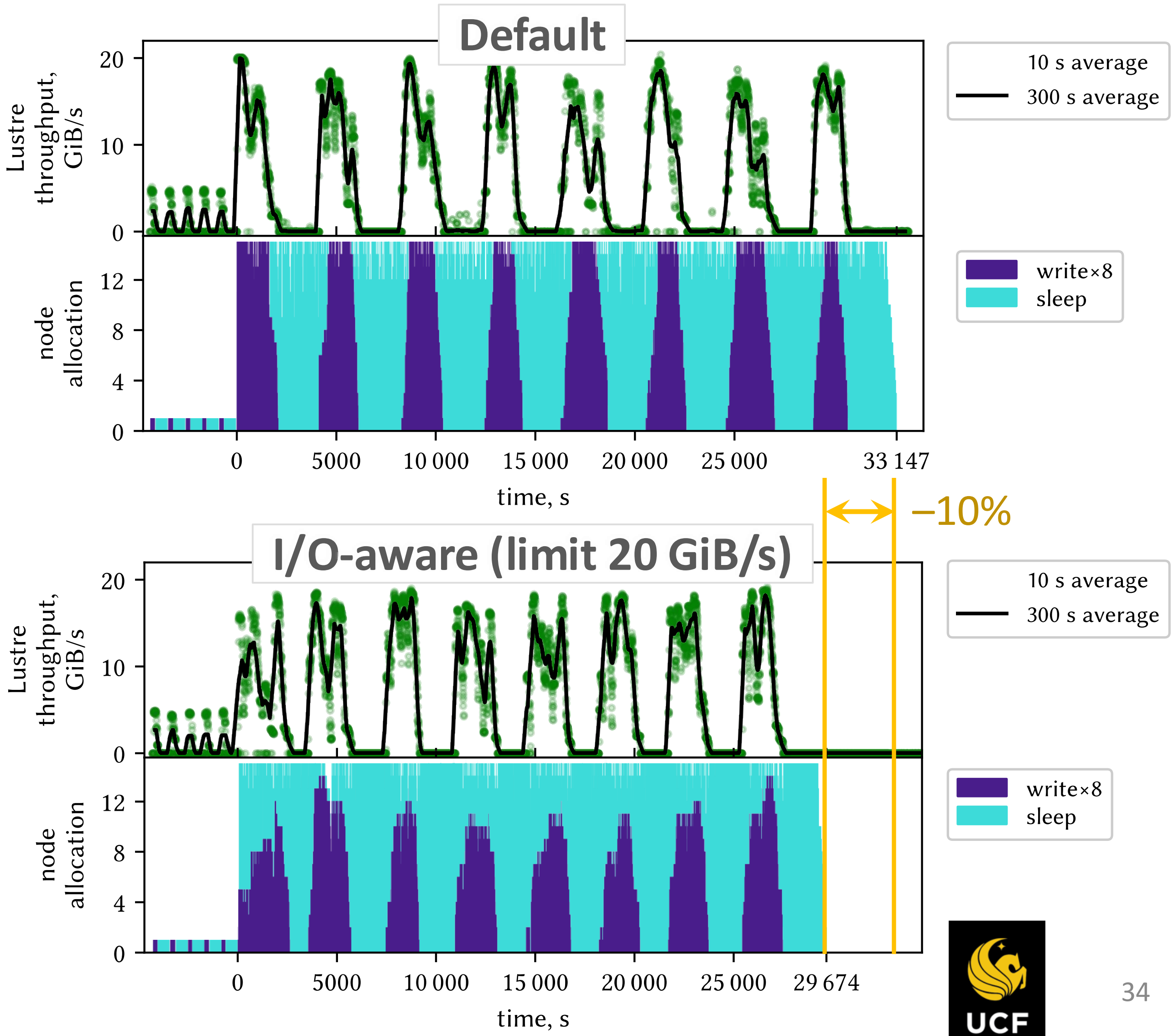
Robustness of I/O-aware scheduling:
"Workload 1" (HPC cluster)

- Periodical pattern (8×)
  - 30 "write×8" jobs
  - 60 "sleep" jobs
- Bandwidth is 15-20 GiB/s
- The estimator is pre-trained by running jobs in isolation
- I/O-aware scheduler initially schedules no more than 5 "write×8" jobs
- Later, I/O-aware scheduler allows as many as 12 "write×8" jobs



Robustness of I/O-aware scheduling:
"Workload 1" (HPC cluster)

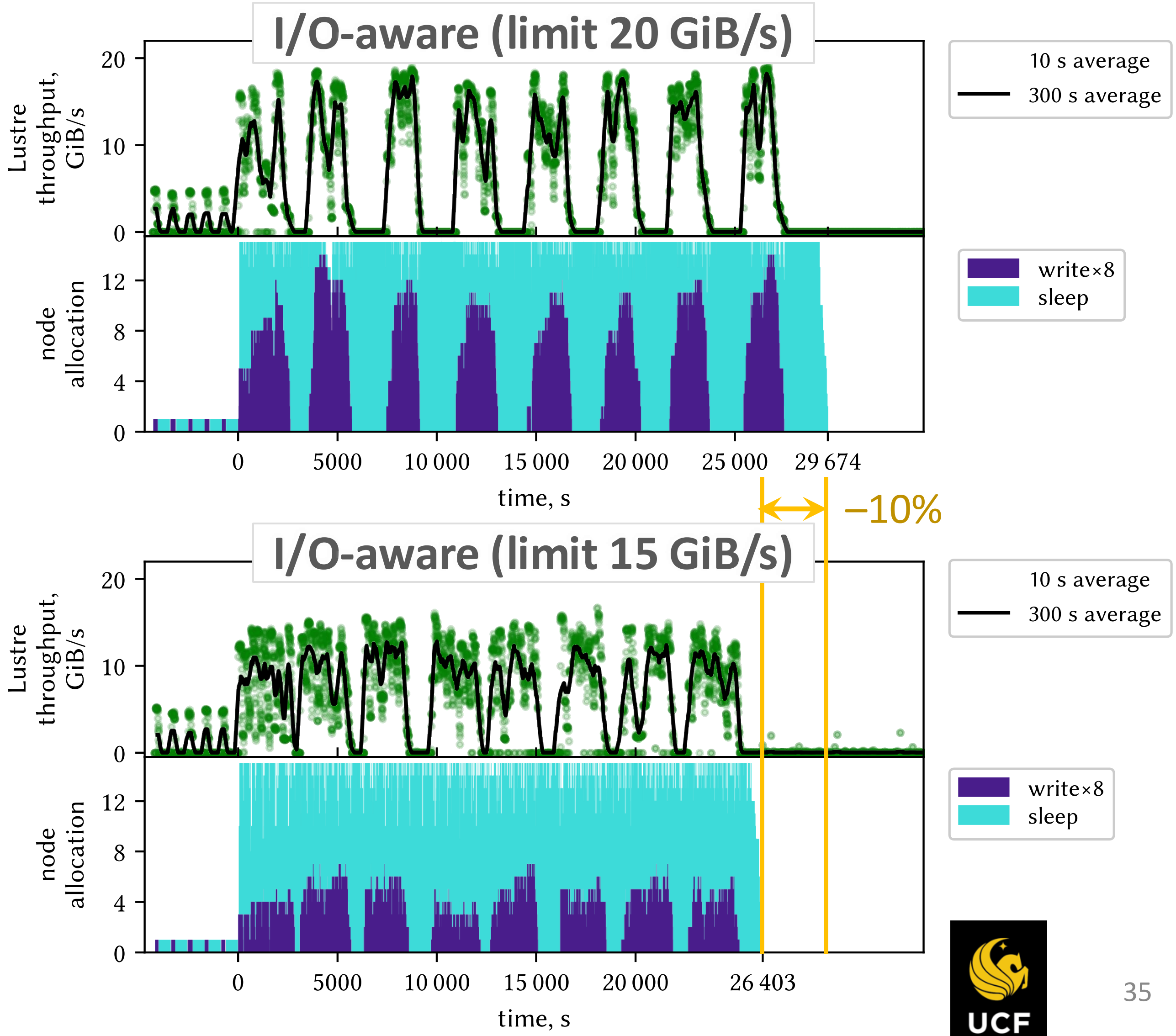
- 10% improvement using I/O-aware scheduler with 20 GiB/s throughput limit



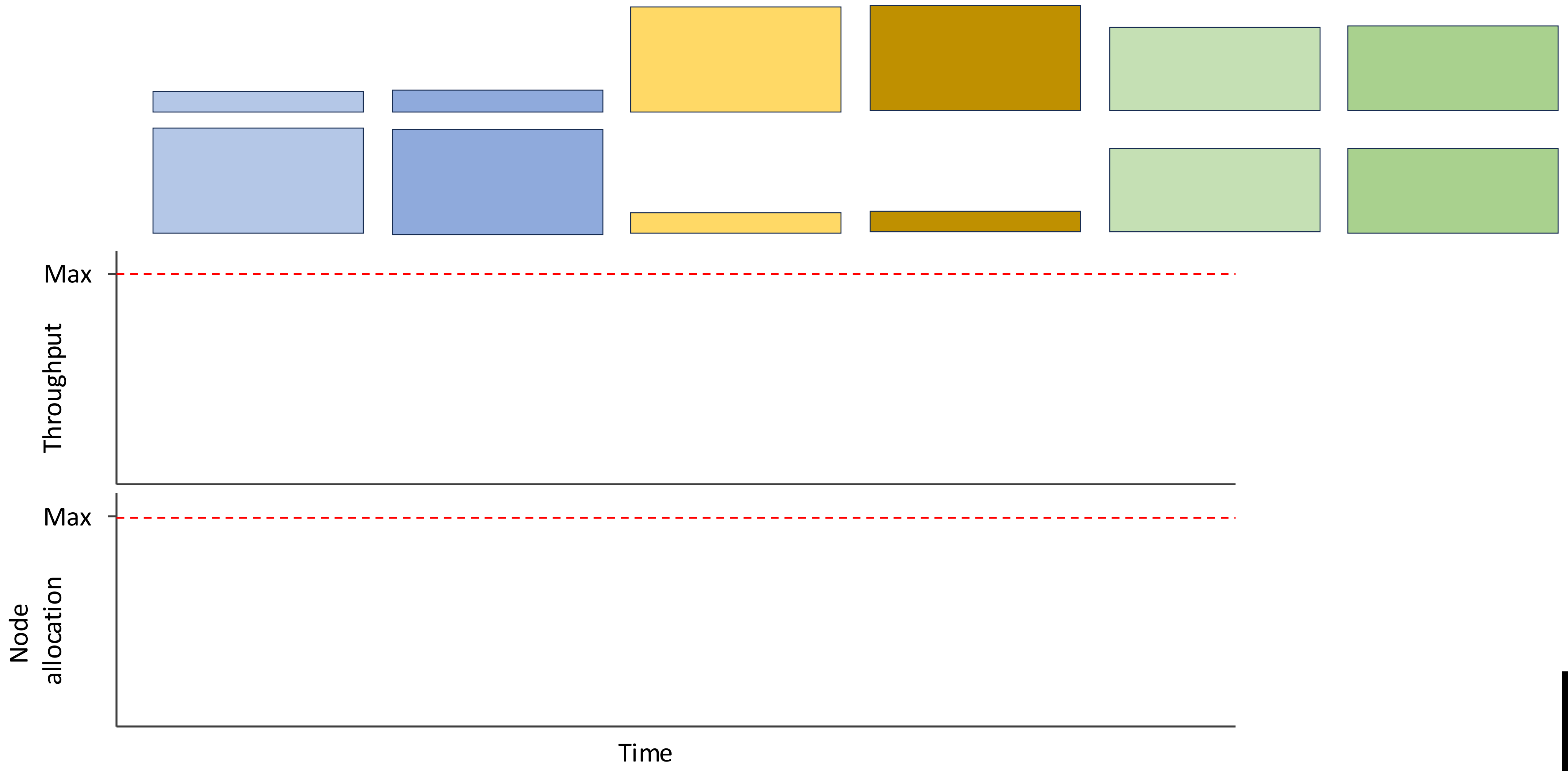
Robustness of I/O-aware scheduling:
"*Workload 1*" (HPC cluster)

- 10% improvement using I/O-aware scheduler with 20 GiB/s throughput limit
- additional 10% improvement using 15 GiB/s throughput limit
  - 20% improvement overall

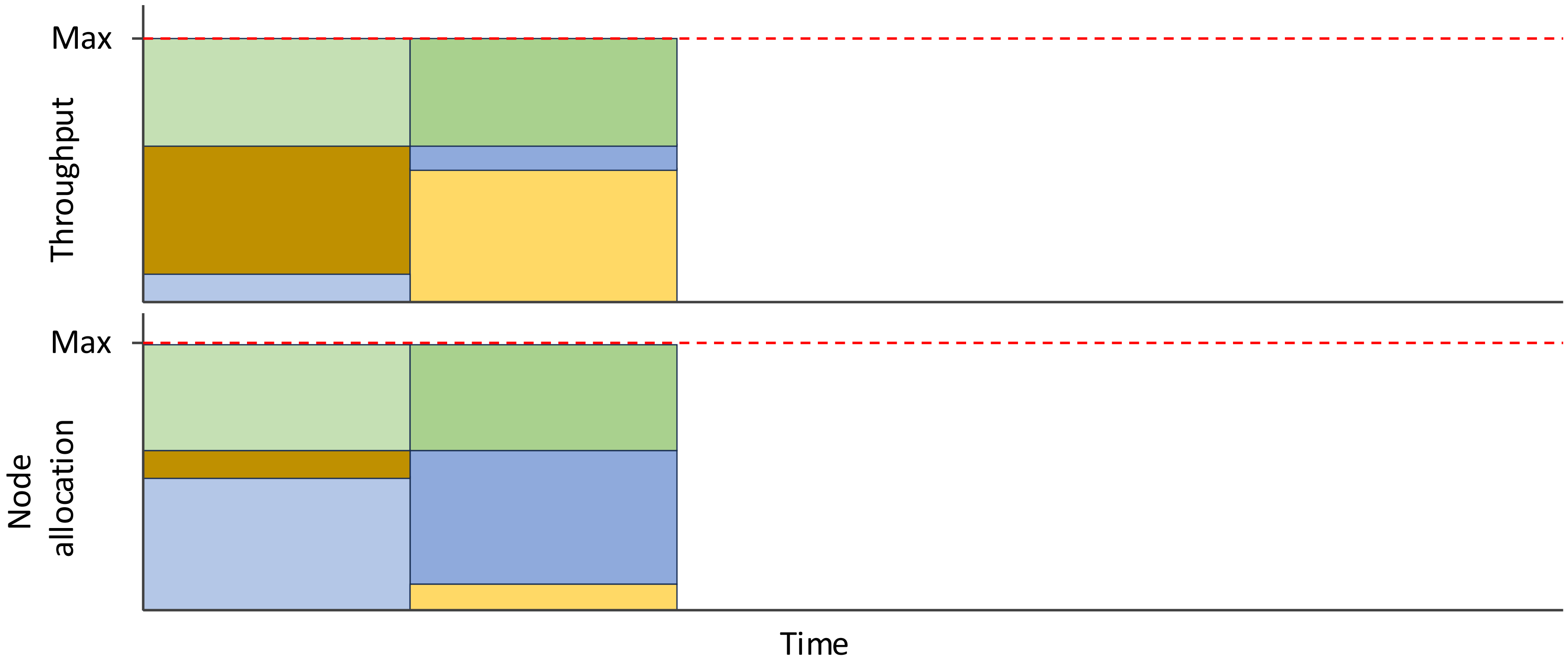
- **Overscheduling leads to further overscheduling**
- **Systematic error is not corrected**



Multi-resource scheduling: Idling in-demand resources



Multi-resource scheduling: Idling in-demand resources



Shortcomings of I/O-aware scheduling

- Better performance can be attained in case rate vs load dependence is concave
- I/O-aware scheduling is not robust when job loads are approximated by job rates
  - Overscheduling leads to further overscheduling
  - Systematic error is not corrected
- I/O-aware scheduling (as other multi-resource scheduling) increase possibly of resources being idle while they are in-demand

UCF

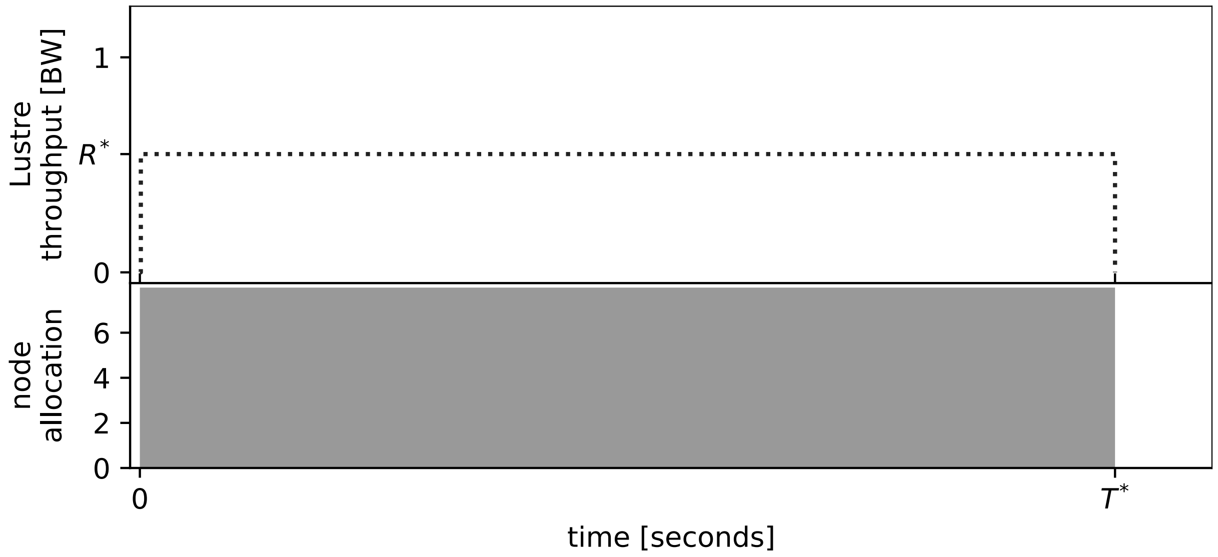
Workload-adaptive scheduling

I/O throughput is a special type of resource

- Cluster-wide, non-exclusive resource
  - The scheduler cannot prevent jobs from using more than allocated
  - Jobs that use the resource can impede each other progress
- Users may not know how their jobs use the resource
  - Measured resource utilization depends on job’s running conditions

Workload-adaptive scheduling: Ideal scenario

- Full utilization of nodes
- Stable I/O throughput



$$Ideal\ makespan = T^* = \frac{Total\ area\ of\ jobs}{Total\ number\ of\ nodes}$$

$$Ideal\ throughput = R^* = \frac{Total\ number\ of\ bytes\ read/written}{Ideal\ makespan}$$

Workload-adaptive scheduling: Practical objective

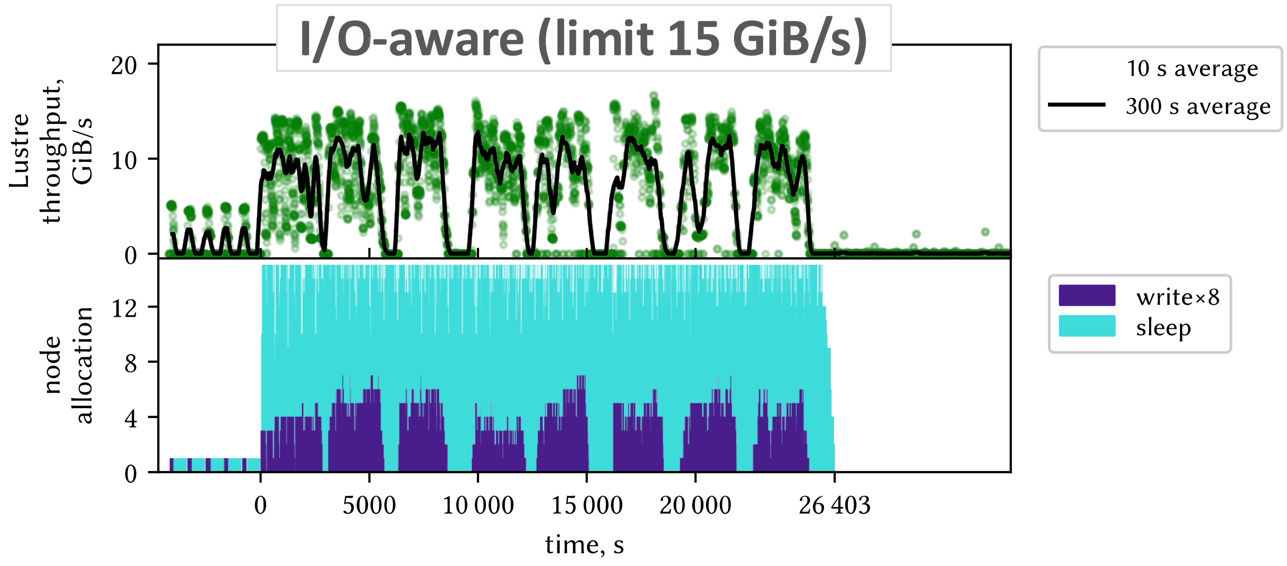
- Target throughput is estimated from pending jobs

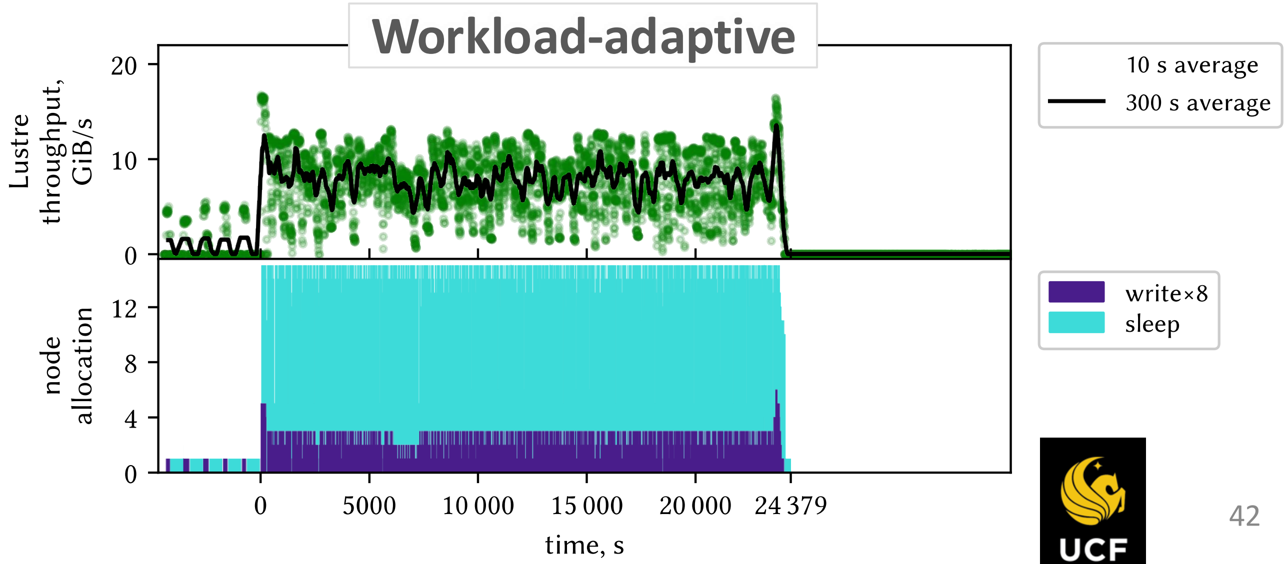
$$Target\ throughput = \frac{Estmated\ number\ of\ bytes\ read/written}{Estimated\ makespan}$$

- Scheduler attempts to maintain the throughput close to $Target\ throughput$ while keeping all nodes occupied
  - "Hard limit" (bandwidth) is still used to prevent overload
- Predictions of job parameters (and correspondingly $Target\ throughput$) are continuously updated

Workload-adaptive scheduling:
"*Workload 1*" (HPC cluster)

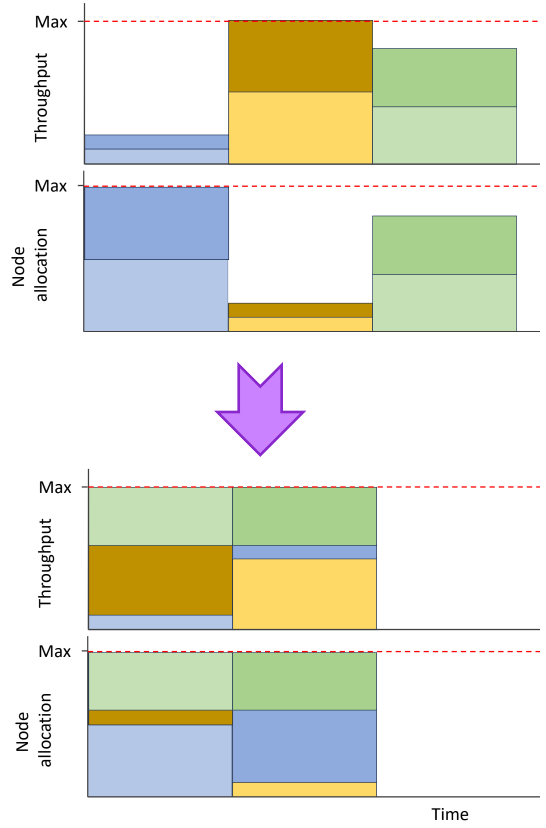
- Workload-adaptive I/O-aware scheduler (bottom) converges to optimal state
  - 5.5% better than "common" I/O-aware scheduler with 15 GiB/s limit
  - 25% better than the default Slurm scheduler





Reducing node idle time

- The algorithm described so far:
  - Jobs using the file system can't be scheduled at time slots for which $Target\ throughput$ has been reached
    - only jobs with "zero load" can still be scheduled
  - The algorithm may cause idle node time and performance degradation if "zero load" jobs are not available
- The algorithm should
  - Keep idle time of the nodes at minimum
  - Keep file system load reasonably close to $Target\ throughput$
- Solution: **Two-group approximation**



Two-group approximation

- Divide jobs into 2 groups according to r^*:

"zero jobs": $\{j: \quad r_j \leq n_j r^*\}$

"regular jobs" : $\{j: \quad r_j > n_j r^*\}$

job's estimated throughput

job's number of nodes

- r^* can be set, for instance, so that

$$\sum_{j \in \text{"zero jobs"}} n_j D_j \geq \sum_{j \in \text{"regular jobs"}} n_j D_j$$

job's estimated runtime

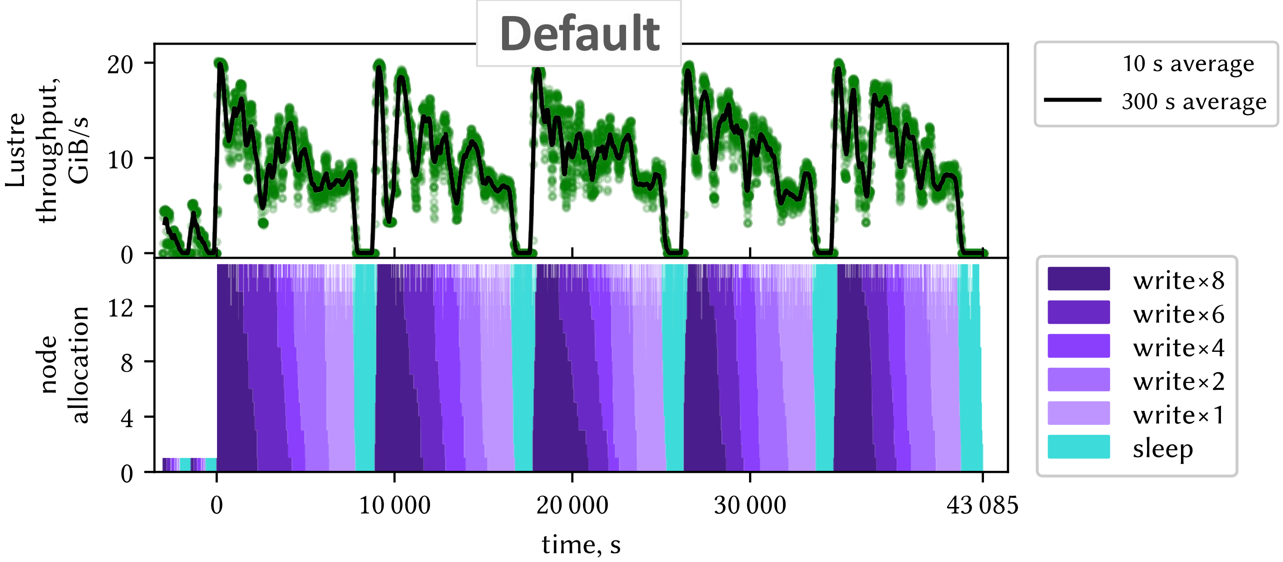
- Find the average load of "zero jobs"

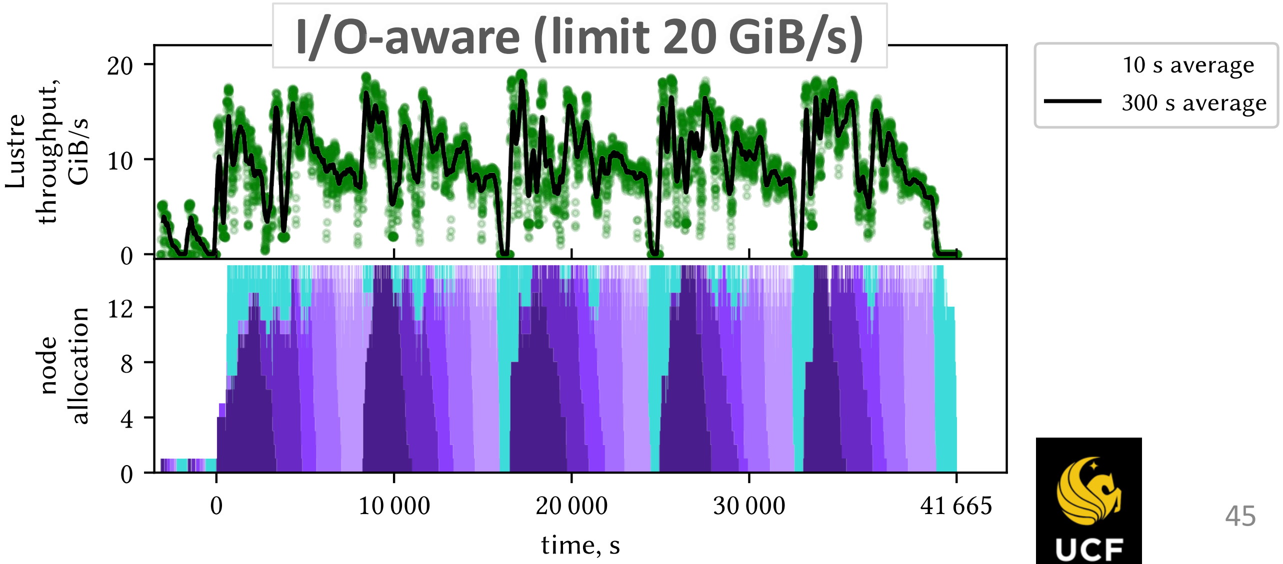
$$\overline{r^*} = \sum_{j \in \text{"zero jobs"}} r_j n_j D_j \Big/ \sum_{j \in \text{"zero jobs"}} n_j D_j$$

- Recalculate target $R^{*\prime} = Target\ throughput - N\overline{r^*}$
- Recalculate jobs' requirements $r_j' = \begin{cases} 0, & j \in \text{"zero jobs"} \\ r_j - n_j\overline{r^*}, & j \in \text{"regular jobs"} \end{cases}$

Two-group approximation:
"Workload 2" (HPC cluster)

- Periodical pattern (5×)
  - 30 "write×8" (8 threads × 10 GiB)
  - 30 "write×6" (6 threads × 10 GiB)
  - 30 "write×4" (4 threads × 10 GiB)
  - 70 "write×2" (2 threads × 10 GiB)
  - 120 "write×1" (1 thread × 10 GiB)
  - 60 "sleep" (10 min)
- 4% improvement using I/O-aware scheduler with 20 GiB/s throughput limit

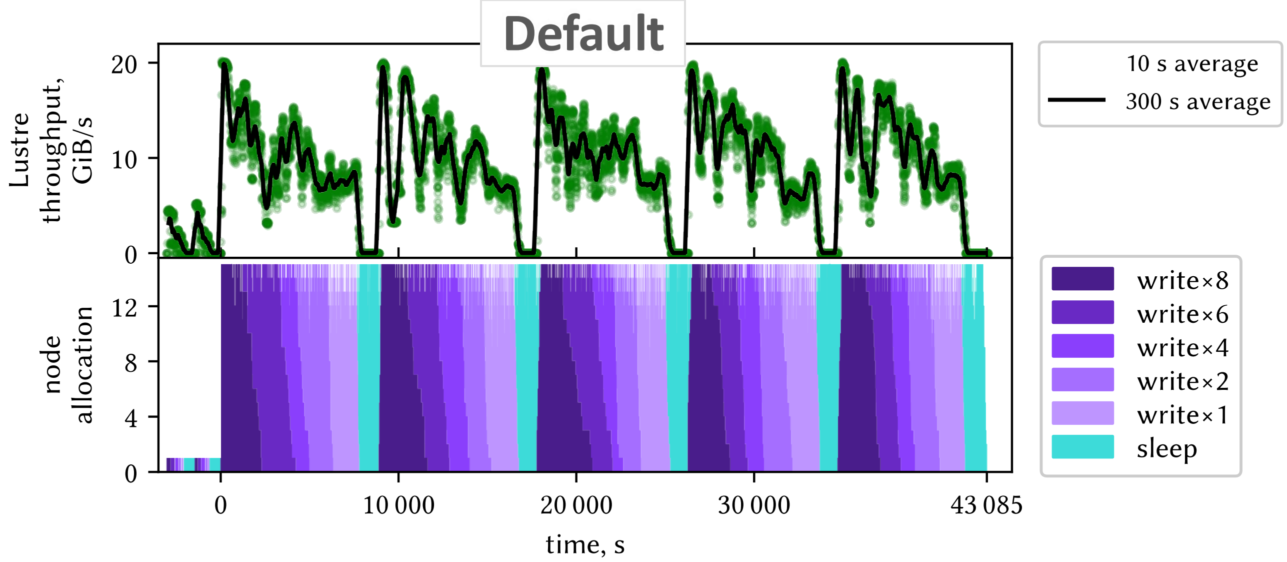


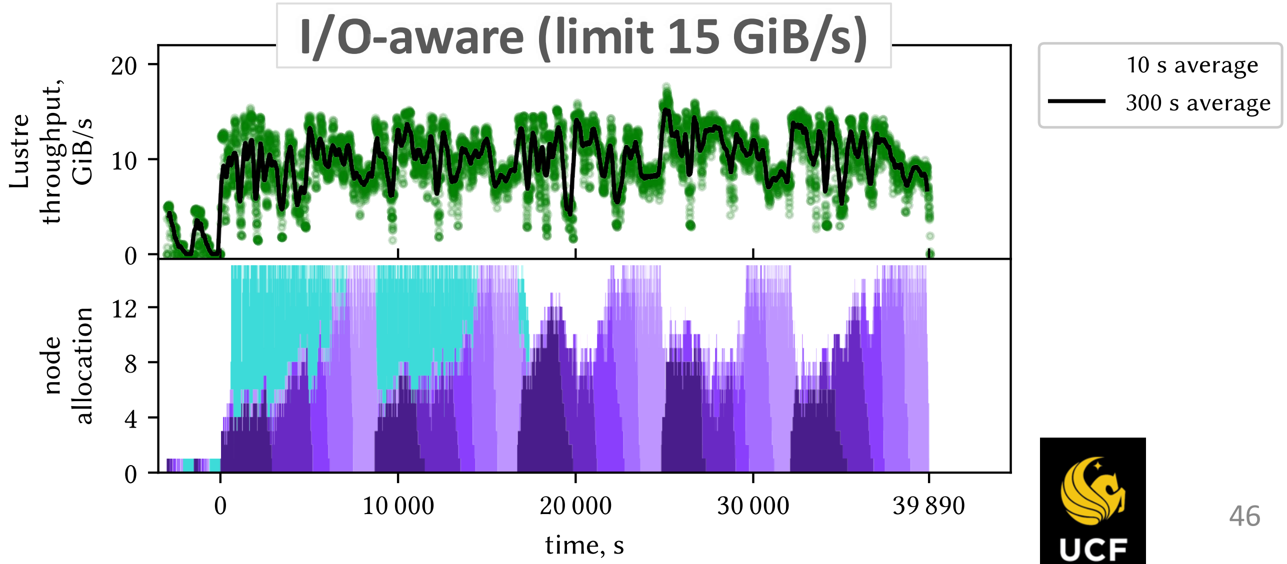


UCF

Two-group approximation:
"Workload 2" (HPC cluster)

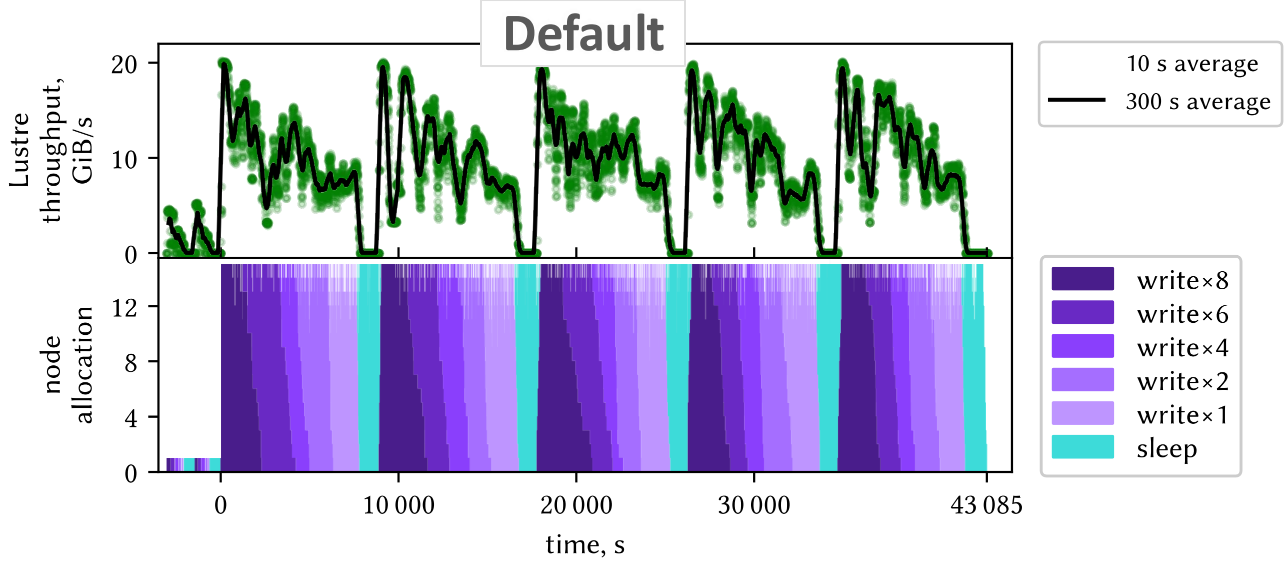
- Periodical pattern (5×)
  - 30 "write×8" (8 threads × 10 GiB)
  - 30 "write×6" (6 threads × 10 GiB)
  - 30 "write×4" (4 threads × 10 GiB)
  - 70 "write×2" (2 threads × 10 GiB)
  - 120 "write×1" (1 thread × 10 GiB)
  - 60 "sleep" (10 min)
- 7% improvement using I/O-aware scheduler with 15 GiB/s throughput limit
  - Idle nodes
  - Could have been worse than the default scheduling

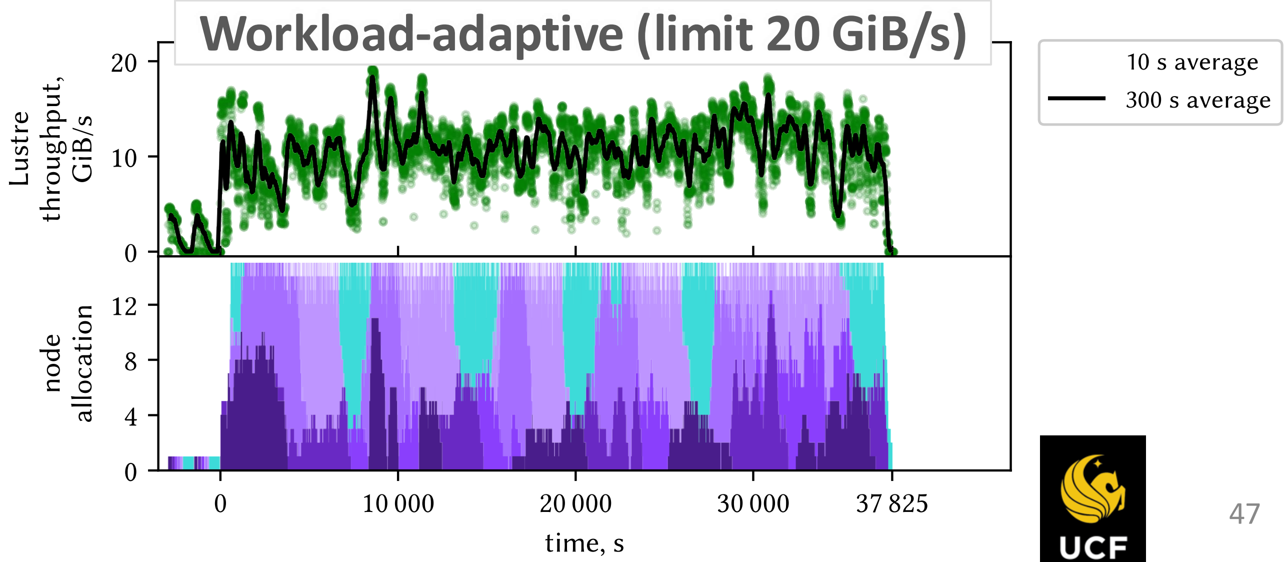




Two-group approximation:
"Workload 2" (HPC cluster)

- Workload-adaptive scheduler with 20 GiB/s limit (bottom) maintains constant throughput without causing idle nodes
  - 5% better than I/O-aware scheduler with 15 GiB/s limit
  - 12% better than the default Slurm scheduler





Conclusions

- We demonstrated a prototype of I/O-aware scheduler based on Slurm and LDMS
  - Predictions of resource requirement based on historical data
  - Ability to manage Lustre throughput
- We proposed Workload-adaptive scheduling approach
  - with "two-group" approximation
- We evaluated the feasibility of the approach
  - on a real HPC cluster
  - on a cluster of virtual machines
  - by simulations

UCF

Acknowledgements

- The works at the University of Central Florida were supported through contracts with Sandia National Laboratories
- Thanks to Benjamin Schwaller, Omar Aaziz, and others at SNL for valuable discussions and help throughout the project
- Thanks to Christina Peterson, Kenneth Lamar and the rest of Prof. Dechev team at UCF

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

UCF