Scheduling Many-Task Workloads on Supercomputers

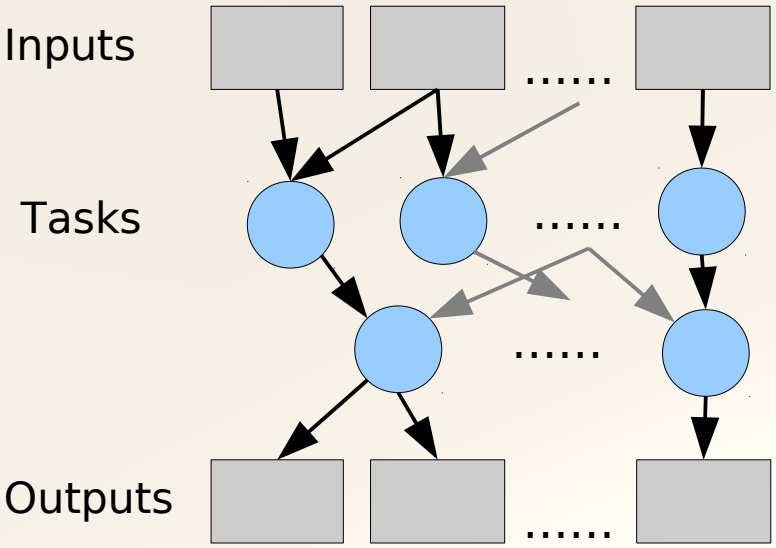
Dealing with Trailing Tasks

Timothy G. Armstrong, Zhao Zhang
Department of Computer Science
University of Chicago

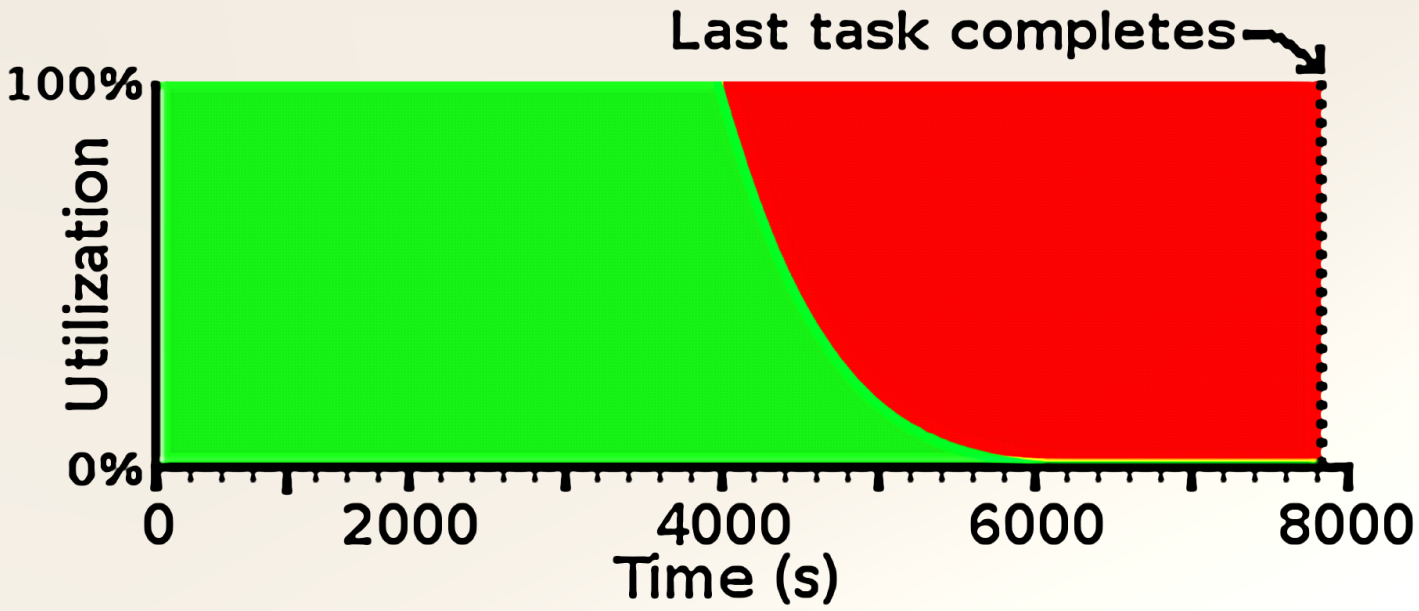
Daniel S. Katz, Michael Wilde, Ian T. Foster
Computation Institute
University of Chicago &
Argonne National Laboratory

Many-tasks on a Supercomputer

- Multi-level scheduling
- Metrics: ***time to solution*** and ***utilization***



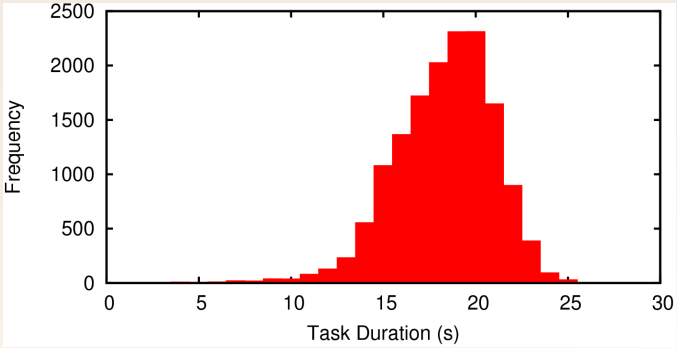
“Trailing Task” Problem



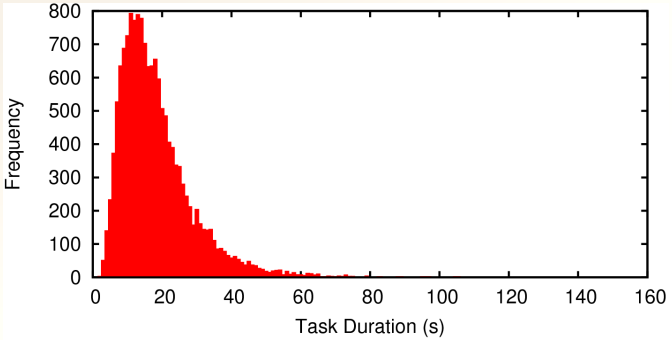
Utilization using 160,000 cores with molecular docking workload of 935,000 independent tasks. Last task completes at 7828 seconds

Runtime Distributions

- Task runtimes can follow various distributions
- Often highly skewed: maybe power law or heavy-tailed distribution
- Many-task computing systems should gracefully handle long-running tasks



Nearly symmetrical distribution of runtimes (another DOCK workload)



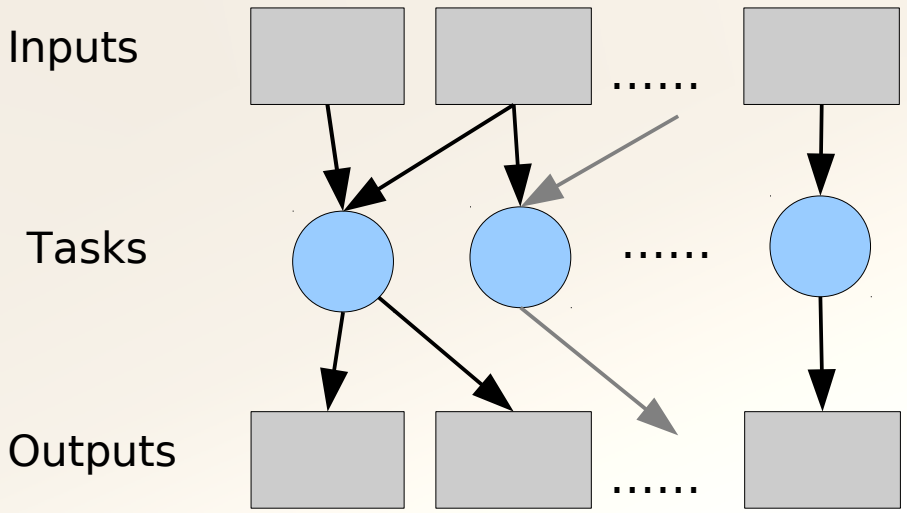
Runtimes with same mean as above but log-normal distribution

Obstacles to Shedding Workers

- Can't always just return unneeded worker CPUs to a pool
- Reasons:
  - Scheduler support
  - Policy restrictions
  - Scheduler not designed for tracking many small allocations
  - Schedule fragmentation
  - Network topology; spatial fragmentation of machine
- Resource provisioning granularity: thousands
- Task scheduling granularity: one

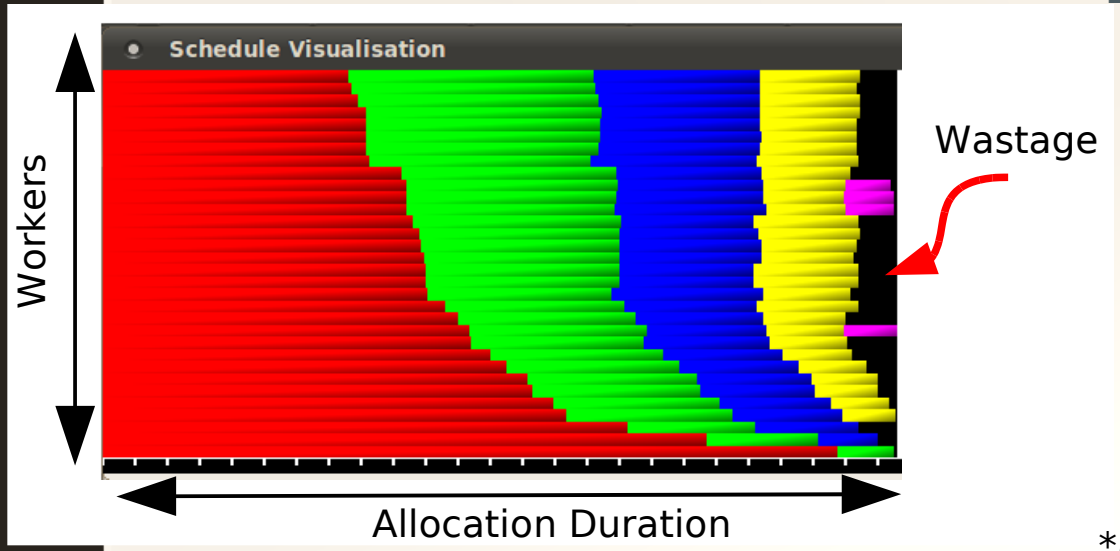
“Bag of Tasks” Workloads

- We only consider workloads with no dependencies: number of “ready” tasks decreases monotonically
  - Most many-task application like this when winding down
  - Similar to a stage of an many-task application with parallel barrier after stage. E.g. *MapReduce* pattern



Fixed Worker Count

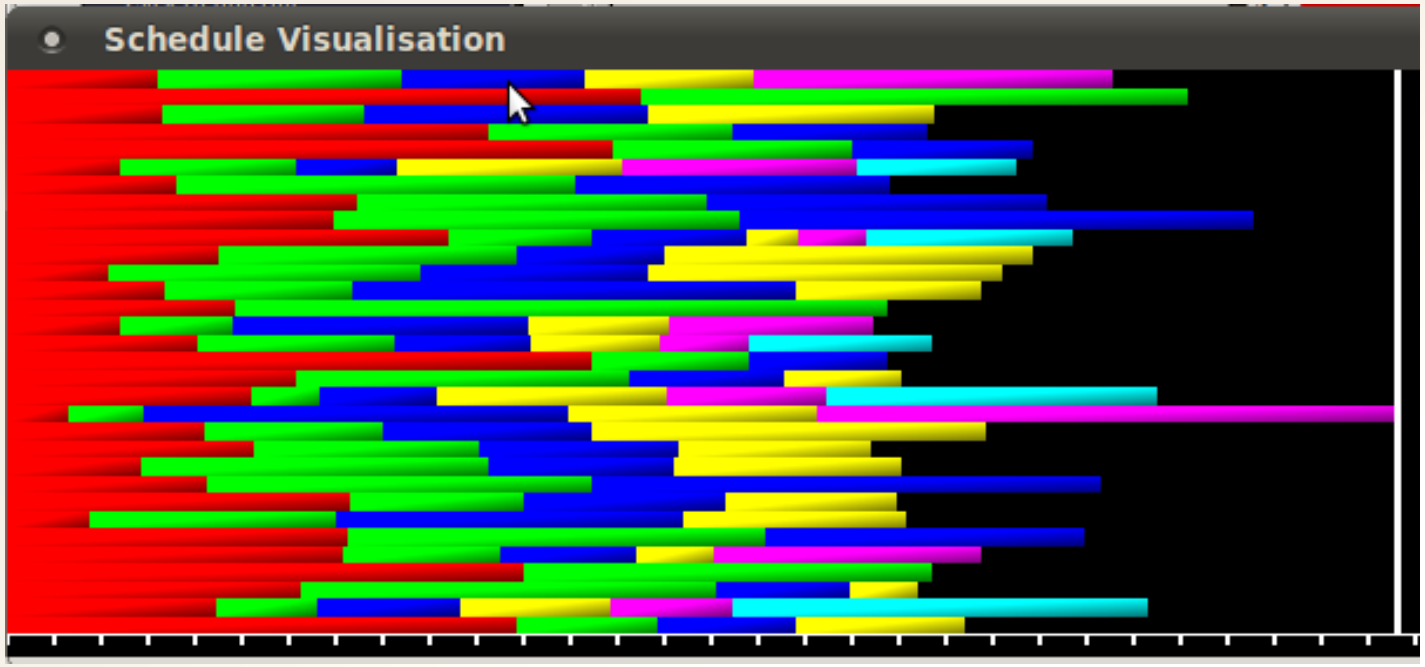
- Minimizing time to sol. leads to maximizing utilization
- NP-Complete optimization problem with heuristics*:
  - Arbitrarily assigning tasks to idle workers *(random)*: 2x opt.
  - Assigning longest running tasks first *(sorted)*: 4/3x opt.
  - Average case behavior for both is better



*Many more in scheduling literature

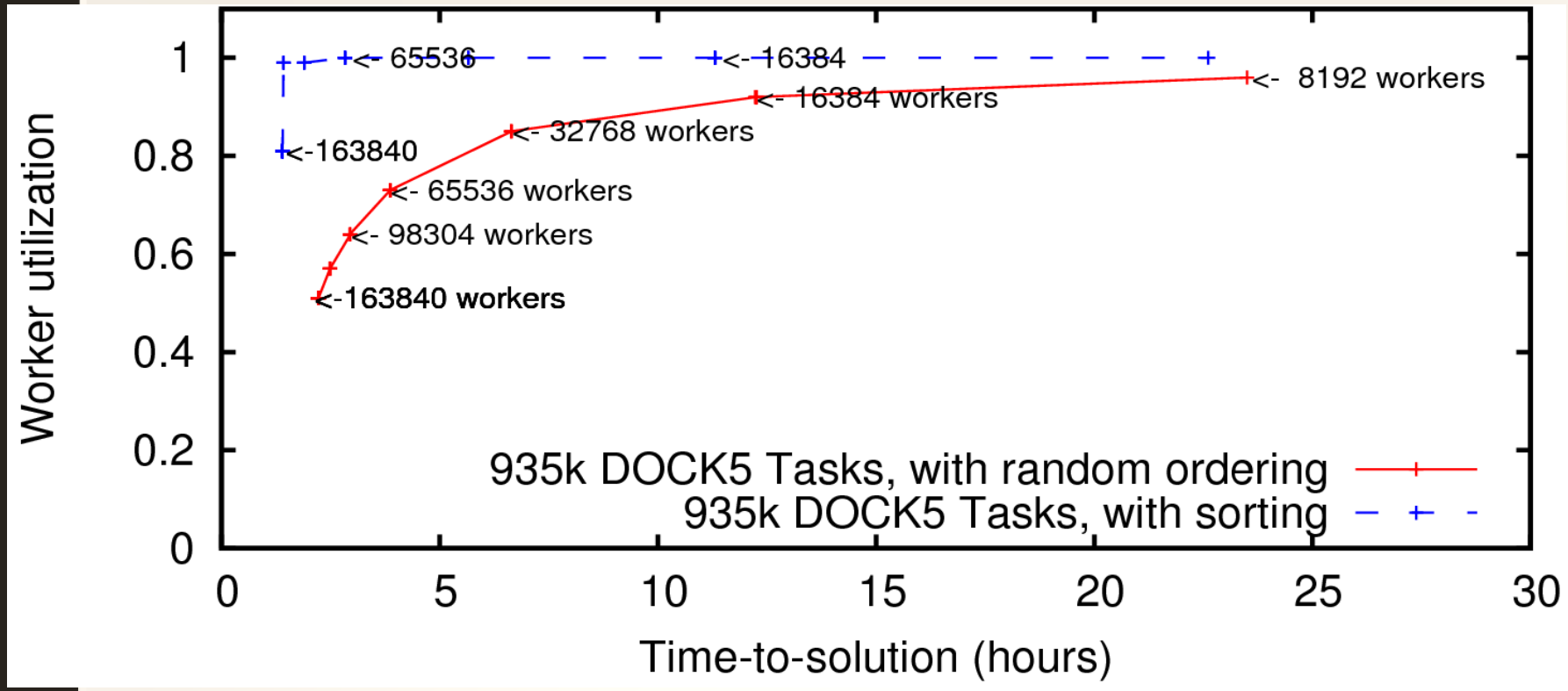
Living with Unknown Runtimes

- In an ideal world we would know the runtime of each task. In practice it is unrealistic assumption
- Random scheduling is what we must live with typically



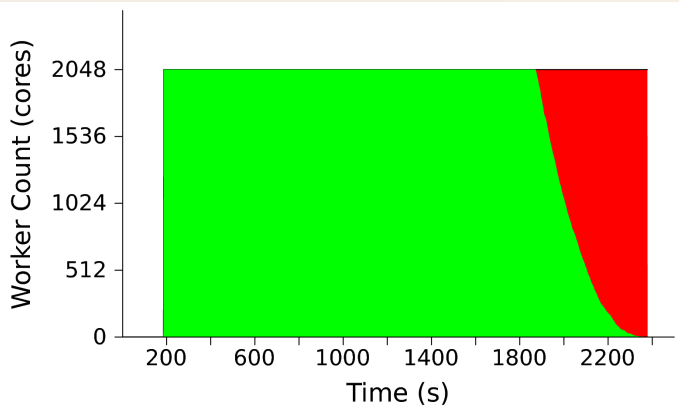
Trade-off with Fixed Worker Count

- With *random,* unavoidable trade-off between utilization and time to solution

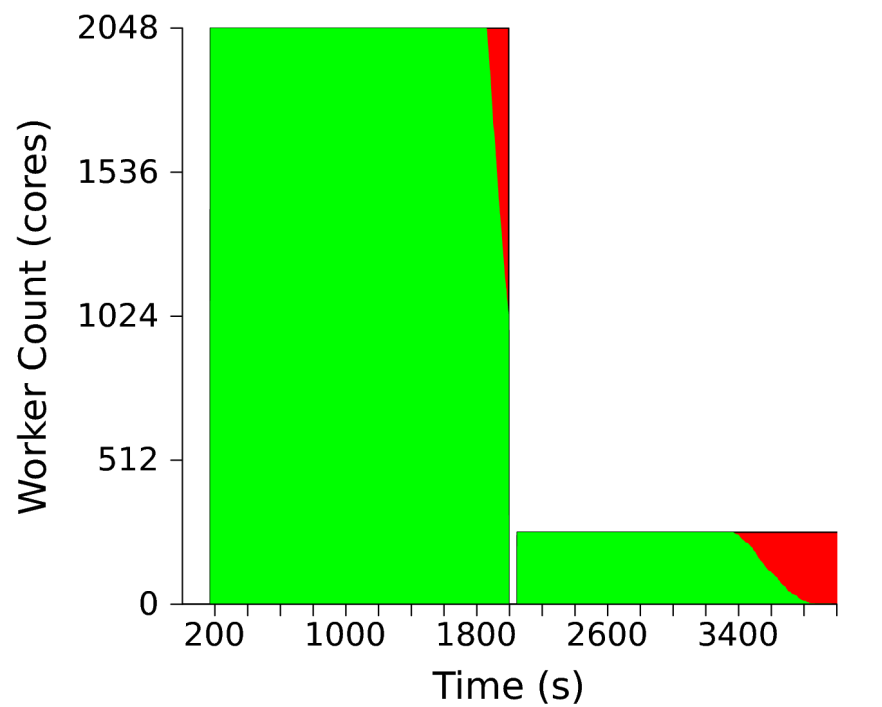


Chopping off the Tail of Tasks

- When utilization gets too low, switch to a smaller allocation
- No special scheduler/system support required



No tail-chopping



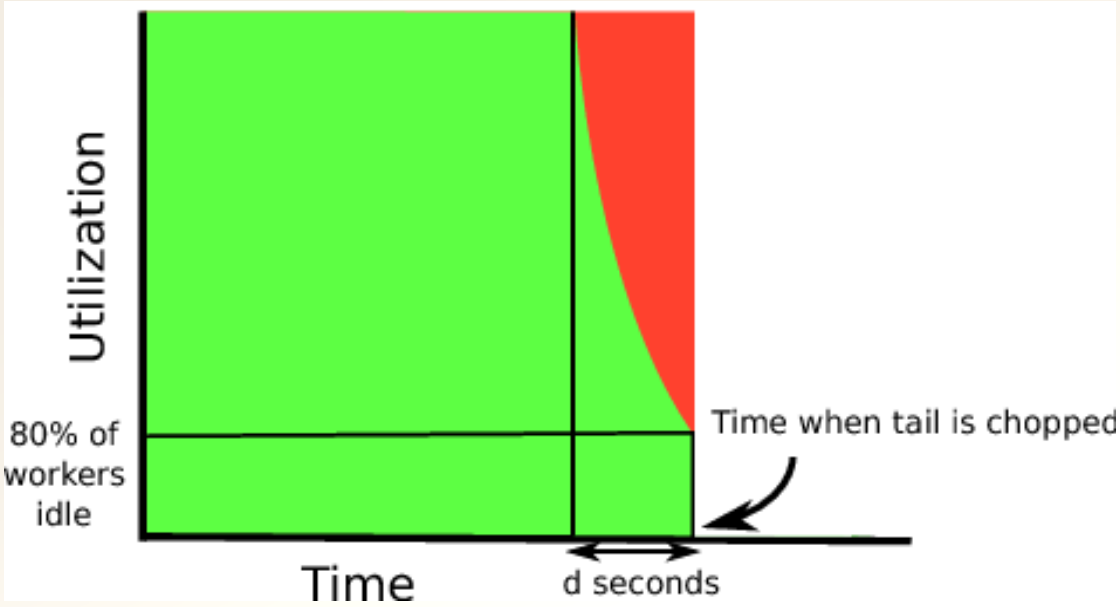
Chopping off the tail

Tail Chopping: Worthwhile?

- Tail-chopping promises to provide:
  - A better trade-off between TTS and Utilization
  - High utilization more robust to changes in worker count
- But has costs:
  - Overhead of migrating to new partition
  - Loss of task progress (unless tasks can be checkpointed)
  - Slower progress on smaller partition
  - Delays in requesting allocation
- Assumptions for study:
  - Tail-chopping means progress of incomplete tasks lost
  - Fixed delay in acquiring new partition

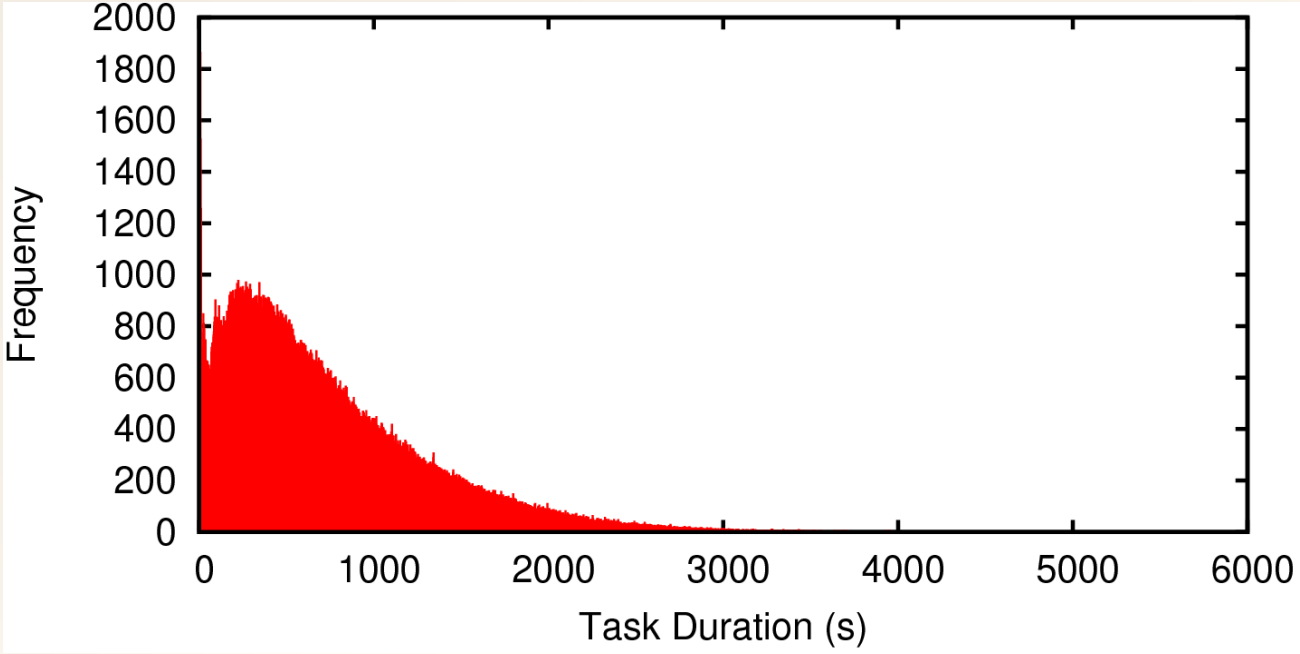
Simulation Design

- Task/worker ratio decides how many workers to request
- Threshold % of idle workers triggers tail-chopping
- Sweep over parameter values to find trade-off curves for different idle thresholds

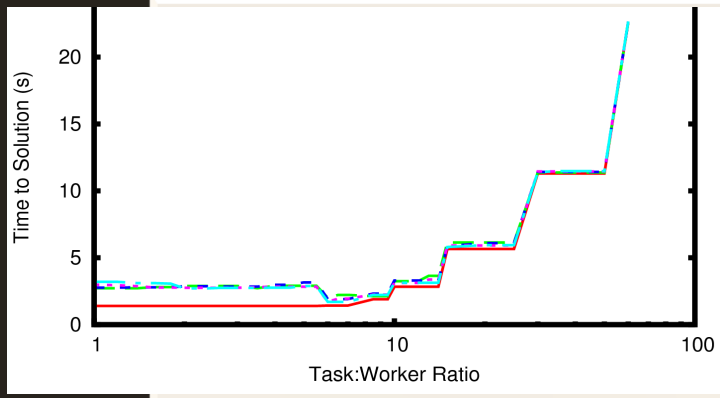


Simulation Data

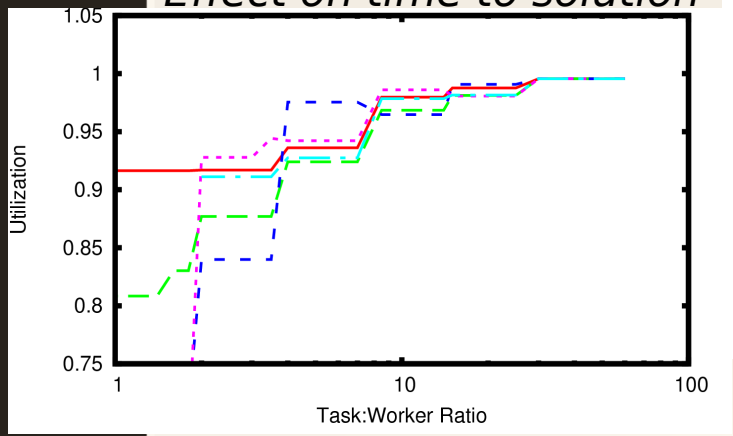
- Runtimes of 935,000 molecular docking tasks
- Skewed distribution of runtimes
- Available allocation sizes those on Blue Gene/P *Intrepid*



Simulation Results (1) - *Sorted*

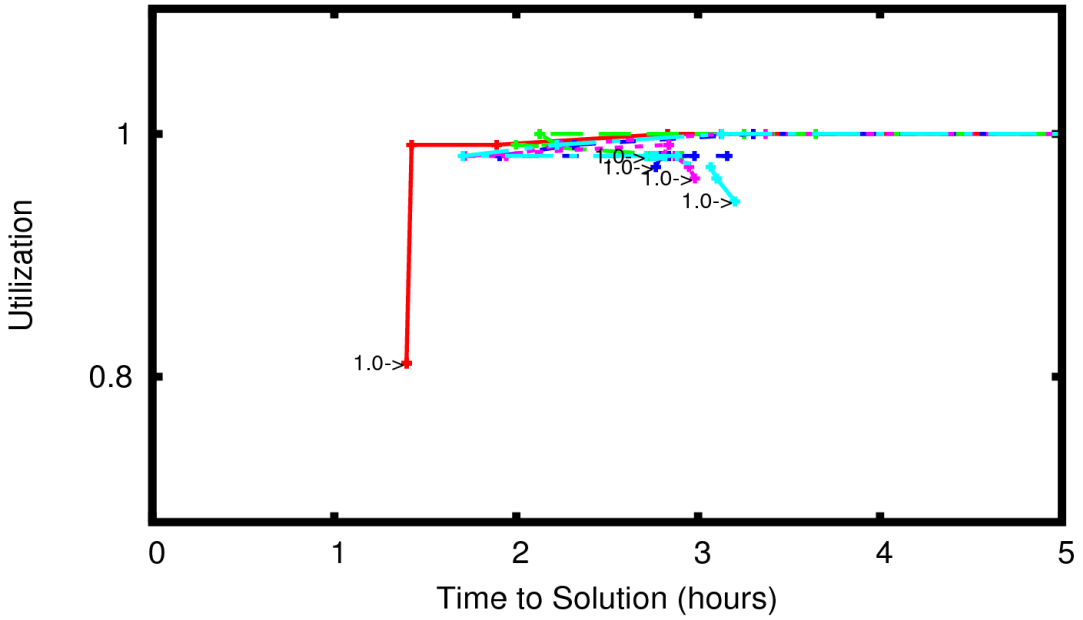


Effect on time to solution



Effect on utilization

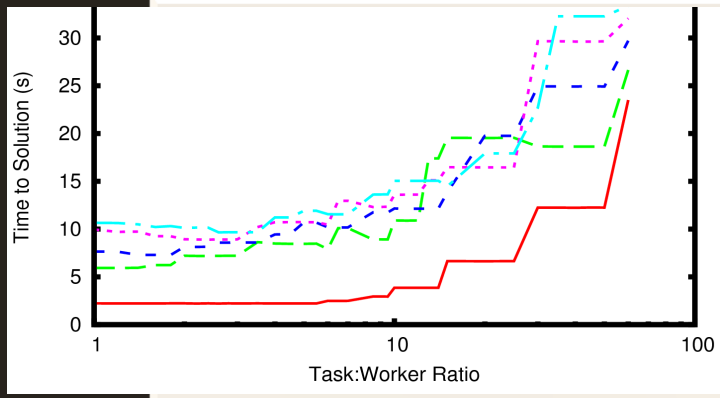
- *Sorted* scheduling



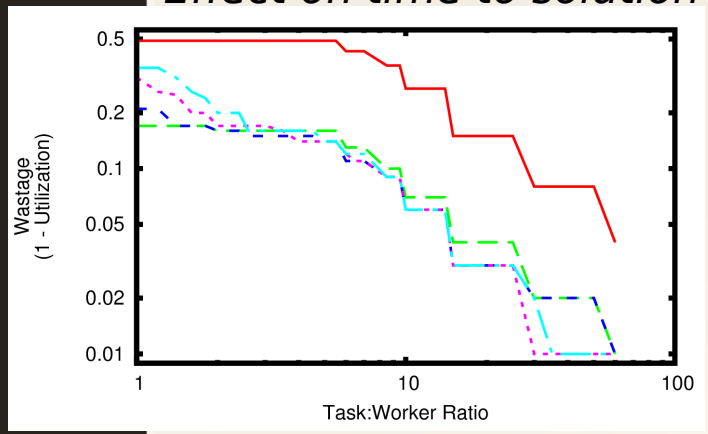
Effect on trade-off

No tail chopping
Max Fraction Idle 0.8
Max Fraction Idle 0.5
Max Fraction Idle 0.2
Max Fraction Idle 0.0

Simulation Results (2) - *Random*

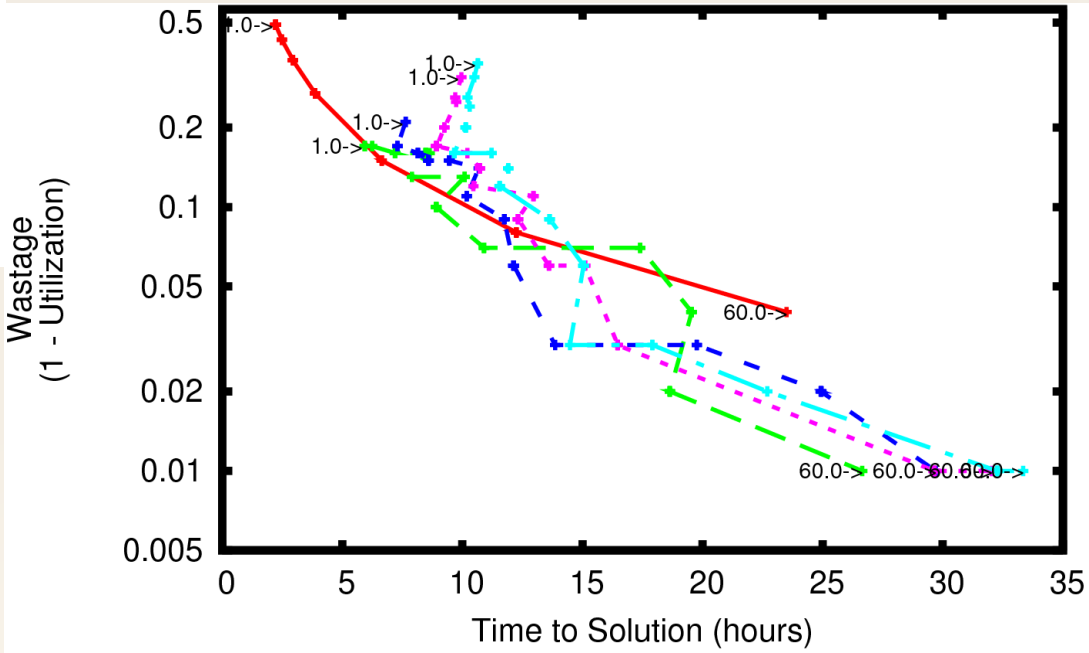


Effect on time to solution



Effect on wastage

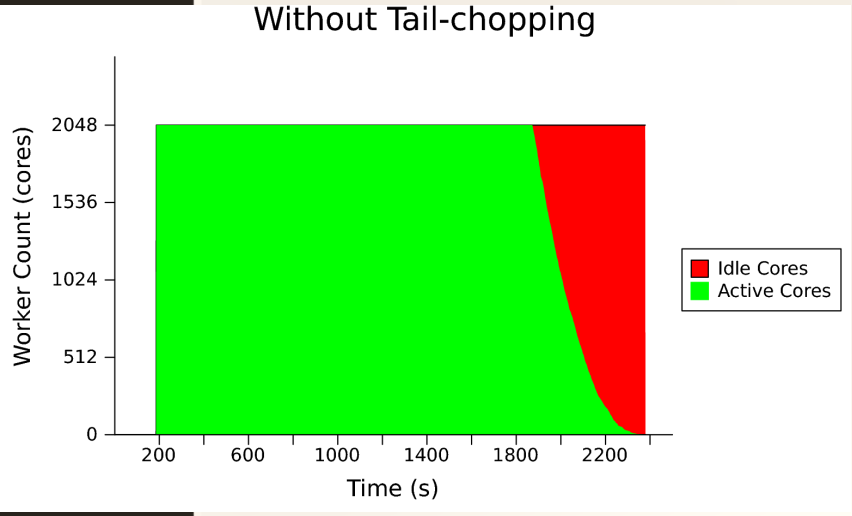
- *Random* scheduling

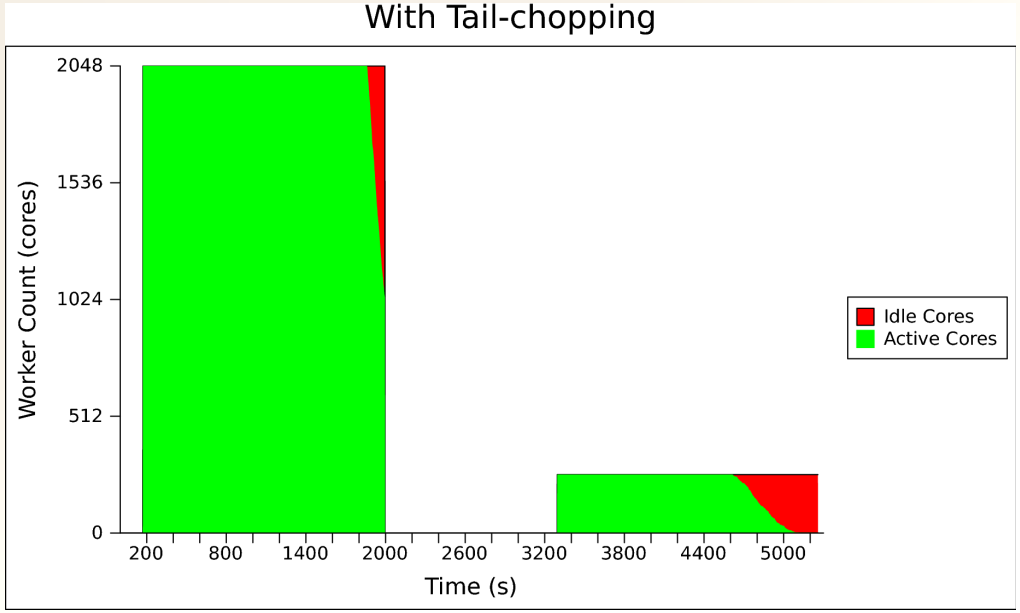


Effect on trade-off

Experiment on Blue Gene/P

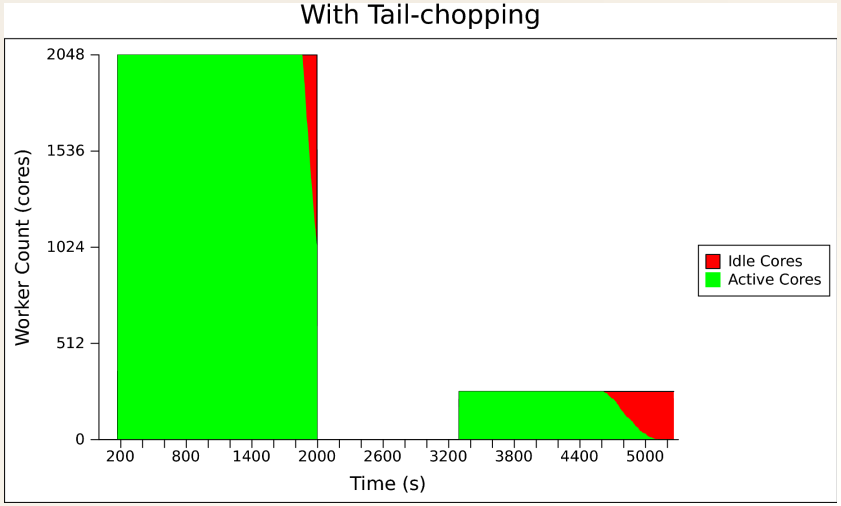
- Proof of concept on Blue Gene/P Intrepid at Argonne National Laboratory using Falkon task dispatcher
- Provisions machine partitions using task/worker ratio.
- Chops off tail when idle workers above 50% threshold





Possible Improvements

- “Warm up” partition before canceling old one
- Scheduler support for shedding workers
- Task migration
- Better heuristic for when to chop tail: use available information about task runtimes



Conclusions

- Need to consider scheduling when running many-task application on a supercomputer, especially if runtimes of tasks are highly variable
- Favorable utilization/time to solution trade-off not always possible with fixed worker count
- Tail-chopping can give better results for time to solution and utilization
- Tail-chopping delivers robustly high utilization

Questions?

The “Straggler” Problem

- Described in MapReduce literature; related but different
- “Straggler”: a task that is running slowly due to poor hardware/software performance
- Standard solution: replicate the task on other machines
  - Only works if the long-running tasks don't intrinsically involve more computation