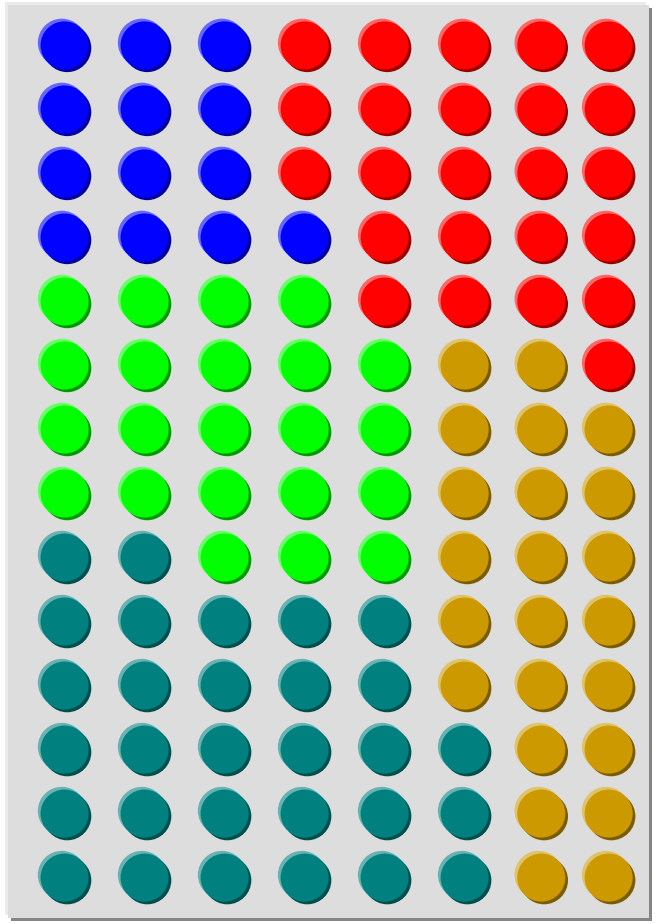
Adaptive Scheduling with Parallelism Feedback

Kunal Agrawal (MIT), Yuxiong He (NUS),
Wen Jing Hsu (NUS), Charles E. Leiserson (MIT)

Supported in part by NSF Grant CNF 0615215

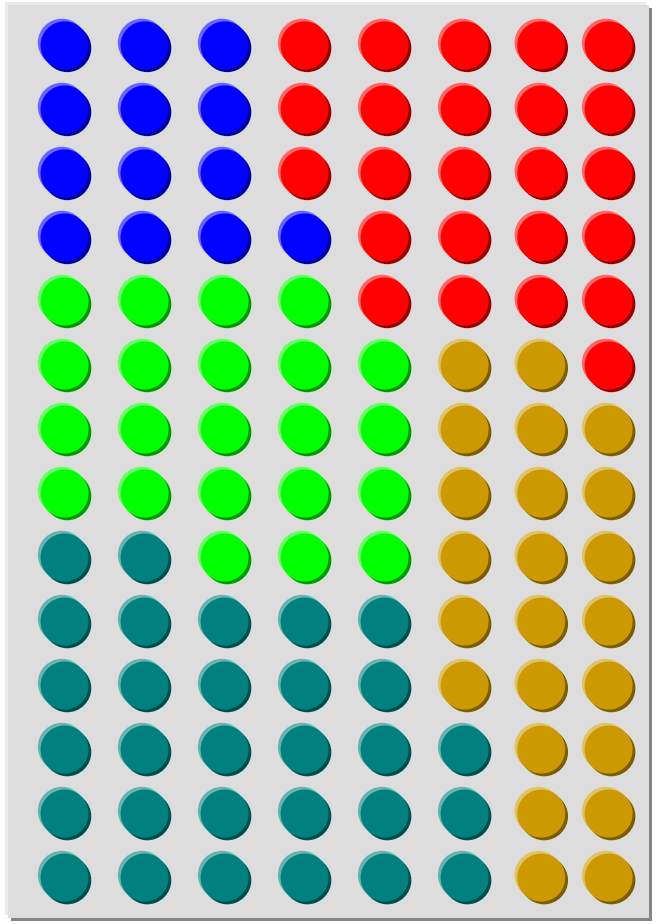
Adaptive Multiprocessor Scheduling

- Many jobs space-share a large multiprocessor, entering and leaving the system dynamically. The number of processors available to the job may, therefore, change during execution.

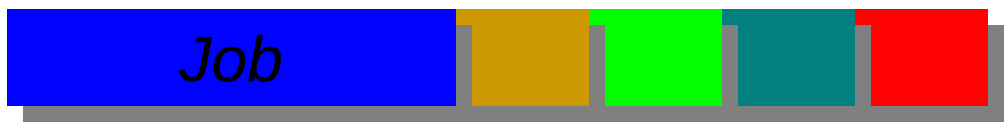


Adaptive Multiprocessor Scheduling

- Many jobs space-share a large multiprocessor, entering and leaving the system dynamically. The number of processors available to the job may, therefore, change during execution.
- The jobs are *adaptively parallel* — the parallelism of a jobs may change during execution, e.g., data parallel jobs [GC+94], dynamic multithreaded jobs [BL92]. The jobs' future parallelism is unknown.

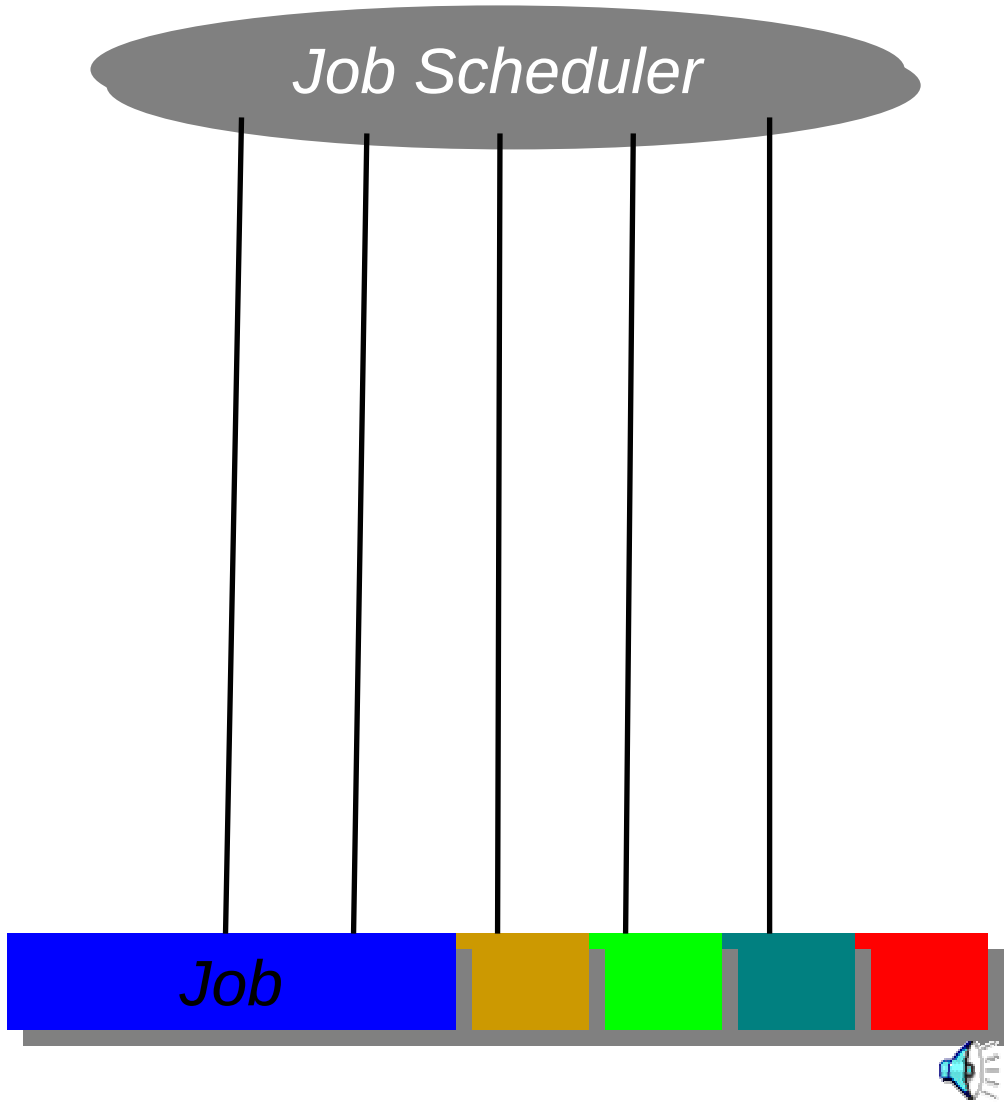


Two-Level Scheduling



Two-Level Scheduling

The *job scheduler* allots processors to jobs.

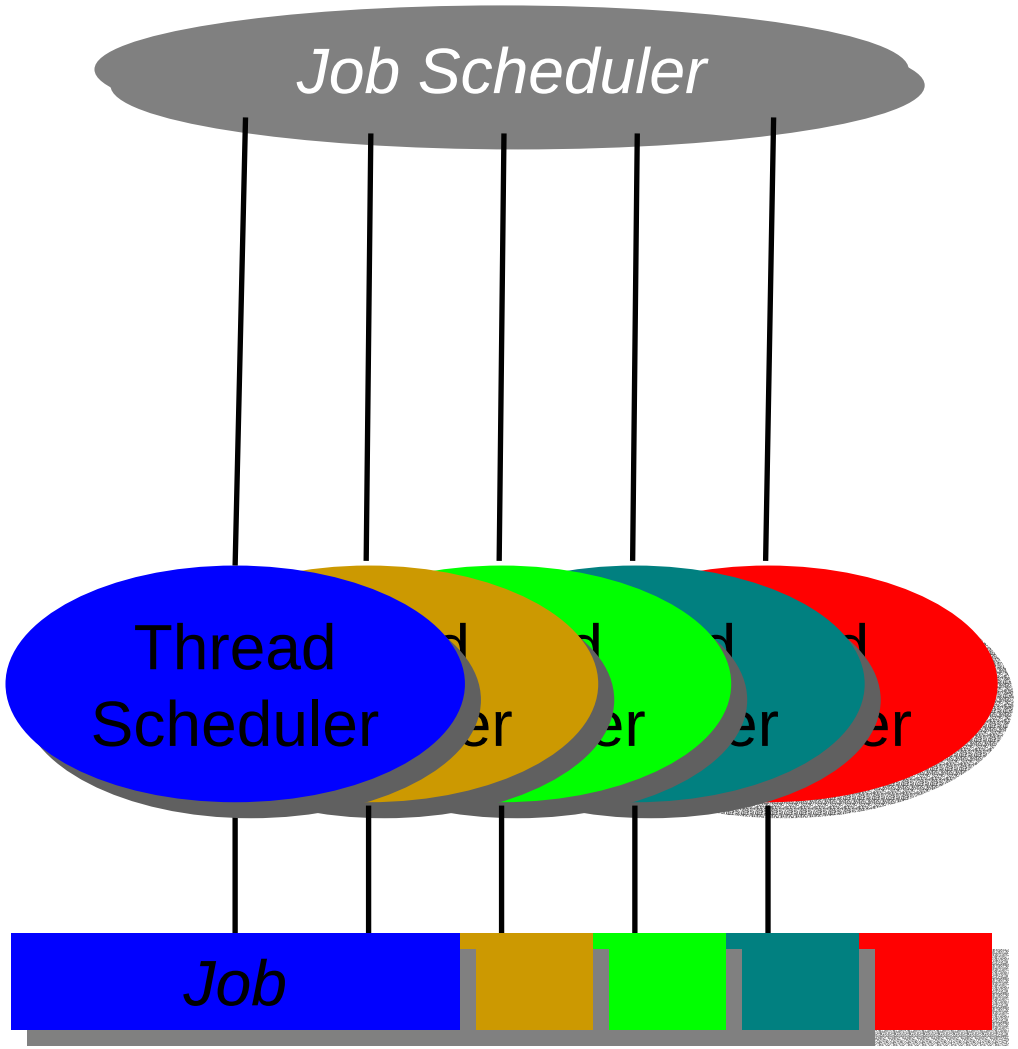


Two-Level Scheduling

The *job scheduler* allots processors to jobs.

The *thread scheduler*

- schedules the ready work of the job on the allotted processors,

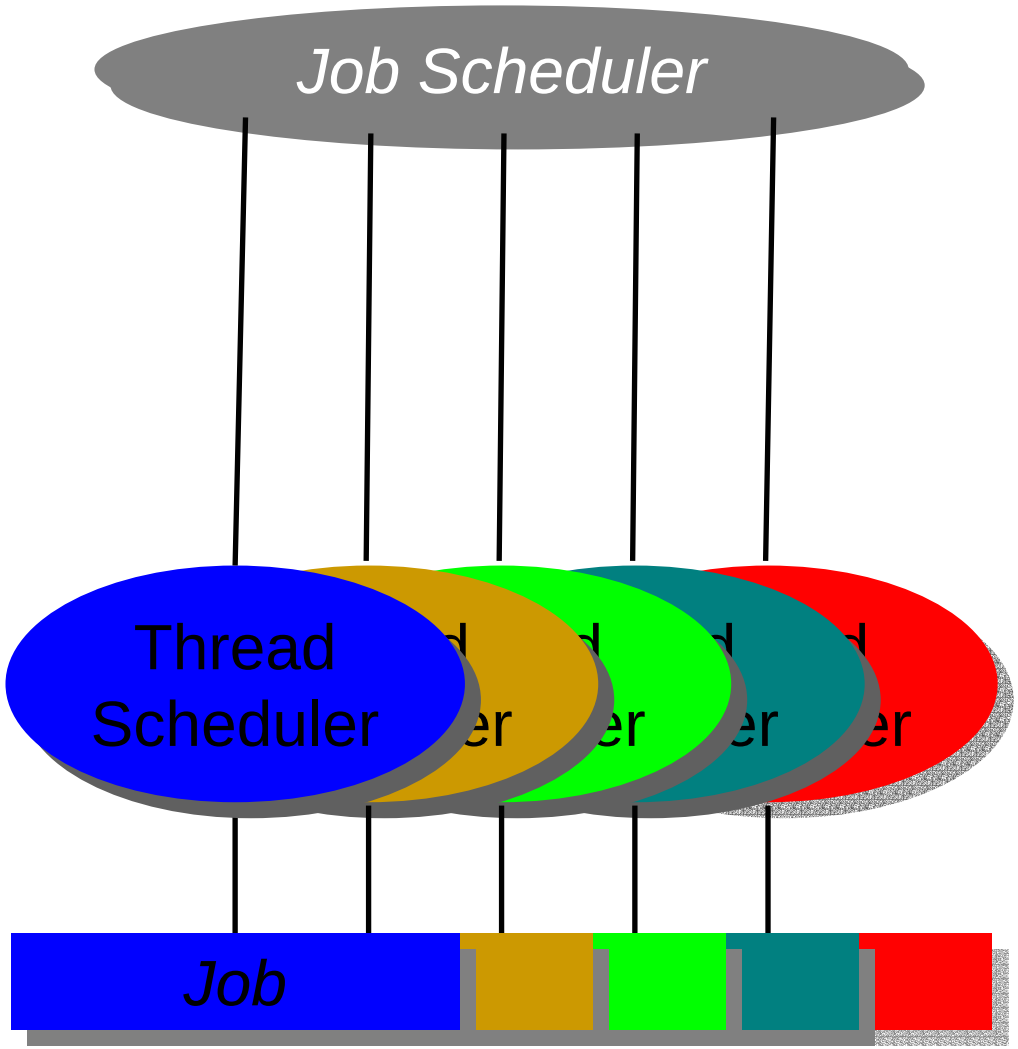


Two-Level Scheduling

The *job scheduler* allots processors to jobs.

The *thread scheduler*

- schedules the ready work of the job on the allotted processors, and
- between scheduling *quanta*, requests processors from the job scheduler for the next quantum *(parallelism feedback).*



Our Contributions

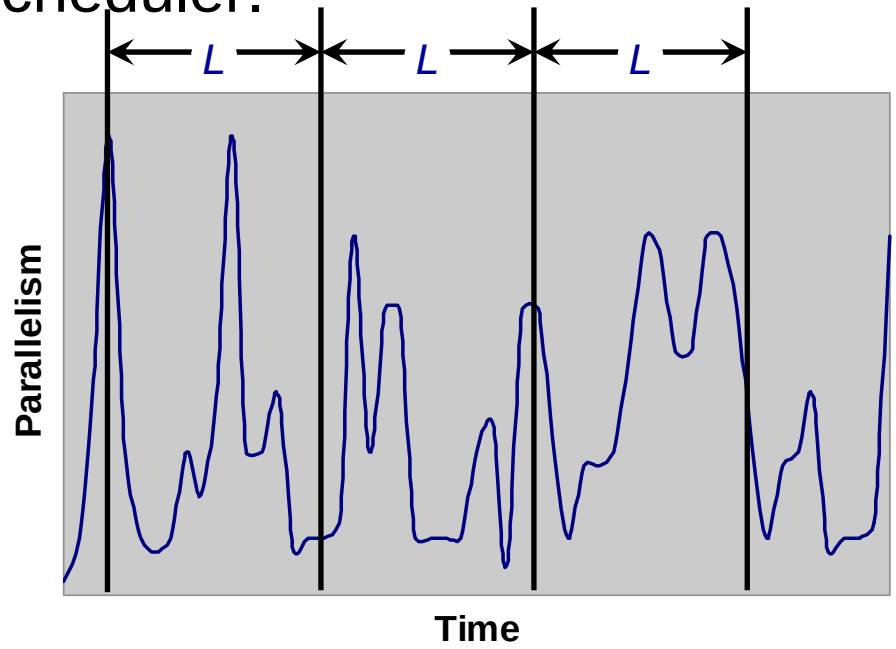
- Provably good thread schedulers that provide parallelism feedback for
  - data-parallel jobs, and
  - dynamic-multithreaded jobs.
- Provably good job schedulers for jobs that use these thread schedulers:
  - dynamic equipartitioning, and
  - dynamic equipartitioning augmented with round robin.
- Performance evaluation by simulations.

Outline

- Problem Definition
- Thread Schedulers
  - A-GREEDY
  - A-STEAL
- Job Schedulers
  - Dynamic Equipartitioning
  - Dynamic Equipartitioning + Round Robin
- Simulation Results
- Future Work

Thread Schedulers

- A thread scheduler (TS) is responsible for reporting the job's parallelism to the job scheduler.
- The TS does not know the job's future parallelism.
- The job's parallelism may change during the quantum. Therefore, using the *instantaneous parallelism* as feedback might be ineffective.



- If the TS requests (and receives) too few processors, then the job runs slowly. If the TS requests (and receives) too many processors, then the job wastes processor cycles that other jobs could have used more effectively.

TS 1: A-GREEDY

A-GREEDY uses *greedy scheduling* [G69, B72] to schedule the job on the allotted processors, and provides parallelism feedback using a multiplicative increase, multiplicative decrease algorithm. It guarantees that the jobs complete quickly and waste few processor cycles, even when the job scheduler is *adversarial*.

THEOREM: Consider a job with *work* T_1 and *span (critical-path length)* T_∞ running on P_{max}-processor machine and scheduling quanta of length L. A-GREEDY guarantees that the job

- wastes $O(T_1)$ processor cycles and
- attains linear speedup on all but $O(T_\infty + L \lg P_{max})$ time steps.

TS 2: A-STEAL

A-STEAL uses *work-stealing* [BS81, H84, BL98] to schedule the job on the allotted processors, and provides parallelism feedback using a multiplicative increase, multiplicative decrease algorithm. It guarantees that the jobs complete quickly and waste few processor cycles, even when the job scheduler is *adversarial*.

THEOREM: Consider a job with *work* T_1 and *span (critical-path length)* T_∞ running on P_{max}-processor machine and scheduling quanta of length L. A-STEAL guarantees that the job

- wastes $O(T_1)$ processor cycles and
- attains linear speedup on all but $O(T_\infty + L \lg P_{max} + L \ln 1/\varepsilon)$ time steps with probability $(1-\varepsilon)$.

Trim Analysis

- We assume that the job scheduler is an adversary of the thread scheduler.
  - Therefore, our results apply to any job scheduler.
- *No* parallelism feedback algorithm can guarantee linear speedup on all time steps, while still minimizing waste, against an adversarial job scheduler.
- To reduce the power of the adversary, we introduce a new technique called *trim analysis.*
- Trim analysis allows the online algorithm to ignore a few data points and guarantee tight bounds on the majority.

Outline

- Problem Definition
- Thread Schedulers
  - A-GREEDY
  - A-STEAL
- Job Schedulers
  - Dynamic Equipartitioning
  - Dynamic Equipartitioning + Round Robin
- Simulation Results
- Future Work

Dynamic Equipartitioning

A *dynamic equipartitioning* (DEQ) job scheduler tries to allot equal number of processors to each job, while never giving any job more processors than it requests.

THEOREM 1: If all jobs use A-GREEDY or A-STEAL in a multiprocessor system which uses DEQ as a job scheduler, then the *makespan* of the jobs is within a *constant factor* of the optimal makespan.

THEOREM 2: If all jobs use A-GREEDY or A-STEAL in a multiprocessor system which uses DEQ as a job scheduler and *all jobs arrive at the same time*, then the *mean completion time* of the jobs is within a *constant factor* of the optimal mean completion time.

DEQ + Round Robin

PROBLEM: Dynamic equipartitioning works only if the number of processors P is at least the number of jobs J.

SOLUTION: Combine DEQ with round-robin scheduling.

- When $P \geqslant J$, use DEQ.
- When $P \leqslant J$, use *round-robin scheduling*. Maintain jobs in a FIFO queue. Remove P jobs from the front of the queue and give them 1 processor each. At the end of the quantum, push these jobs back at the end of the queue and schedule the next P jobs.

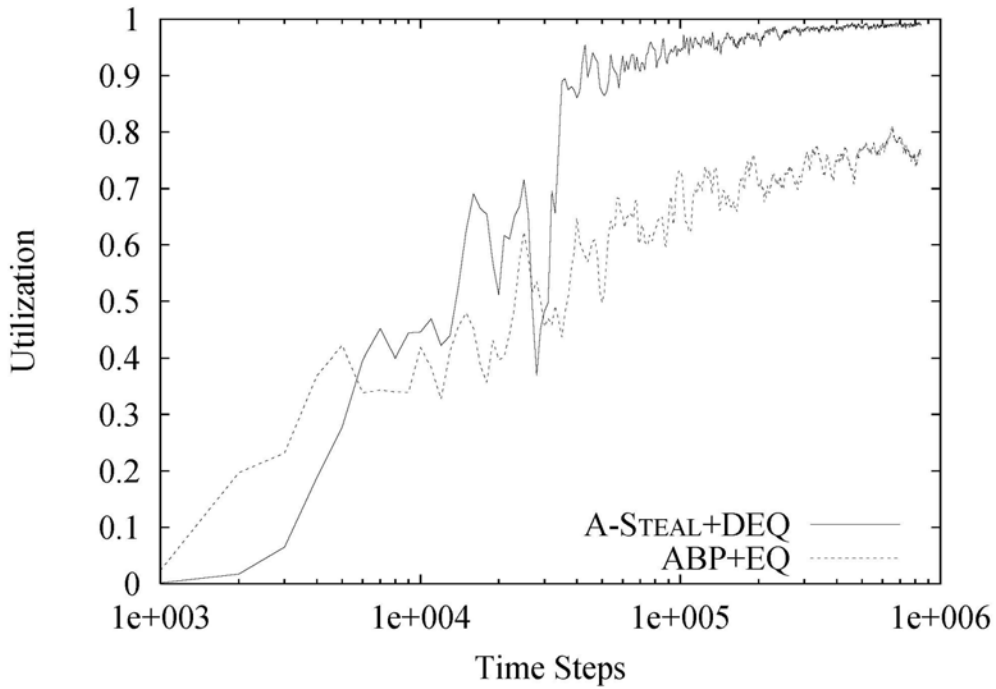
This algorithm provides results analogous to DEQ. When this algorithm is used in conjunction with A-GREEDY, then we get constant competitiveness with respect to makespan and mean completion time (if all jobs arrive at the same time).

Outline

- Problem Definition
- Thread Schedulers
  - A-GREEDY
  - A-STEAL
- Job Schedulers
  - Dynamic Equipartitioning
  - Dynamic Equipartitioning + Round Robin
- Simulation Results
- Future Work

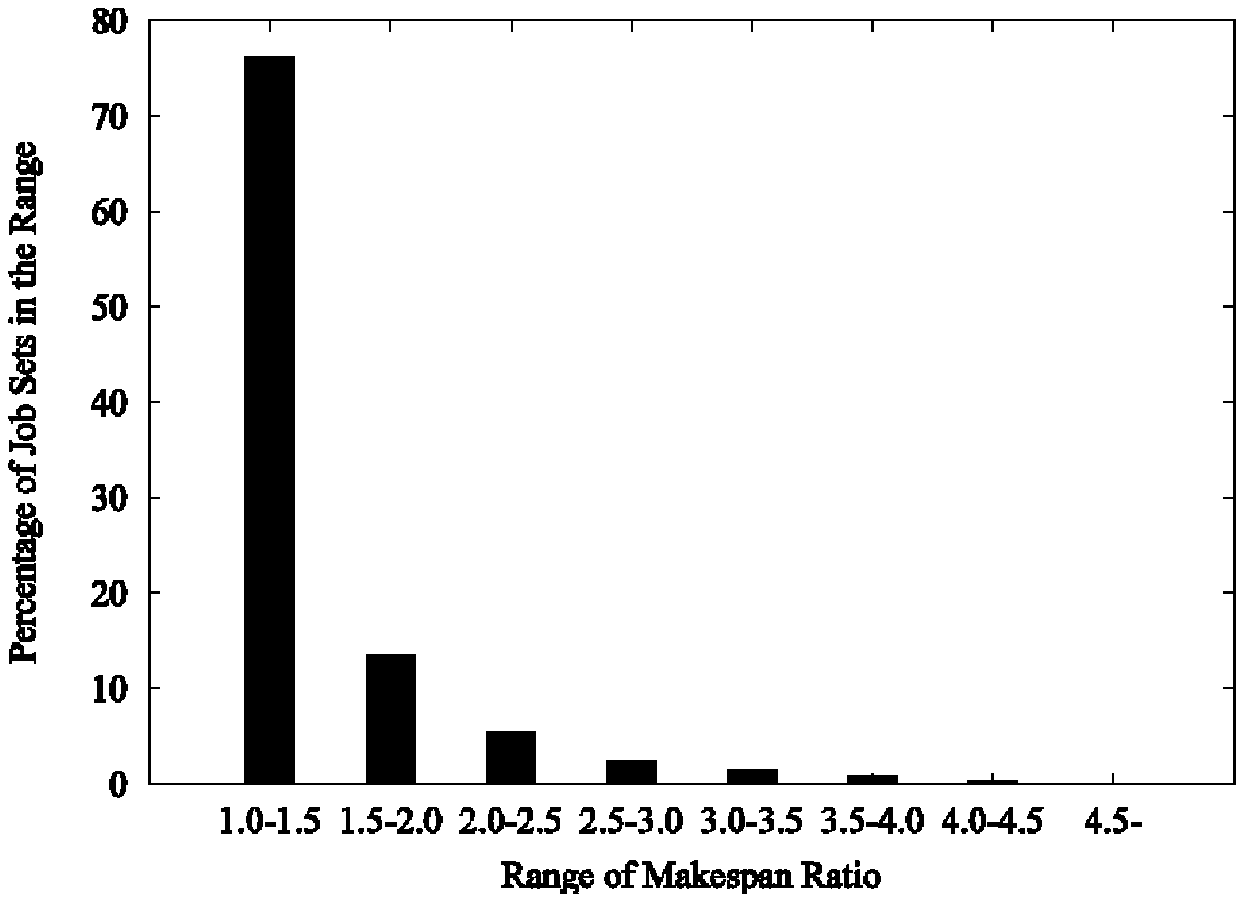
Simulation: A-STEAL vs. ABP

- A-STEAL provides nearly perfect linear speedup and wastes less than 20% of the allotted processor cycles.
- We compared the utilization provided by A-STEAL and *ABP* [ABP99], a work-stealing scheduler that does not employ parallelism feedback.
- The mean completion time of jobs under ABP is nearly 50% slower than that of jobs using A-STEAL.



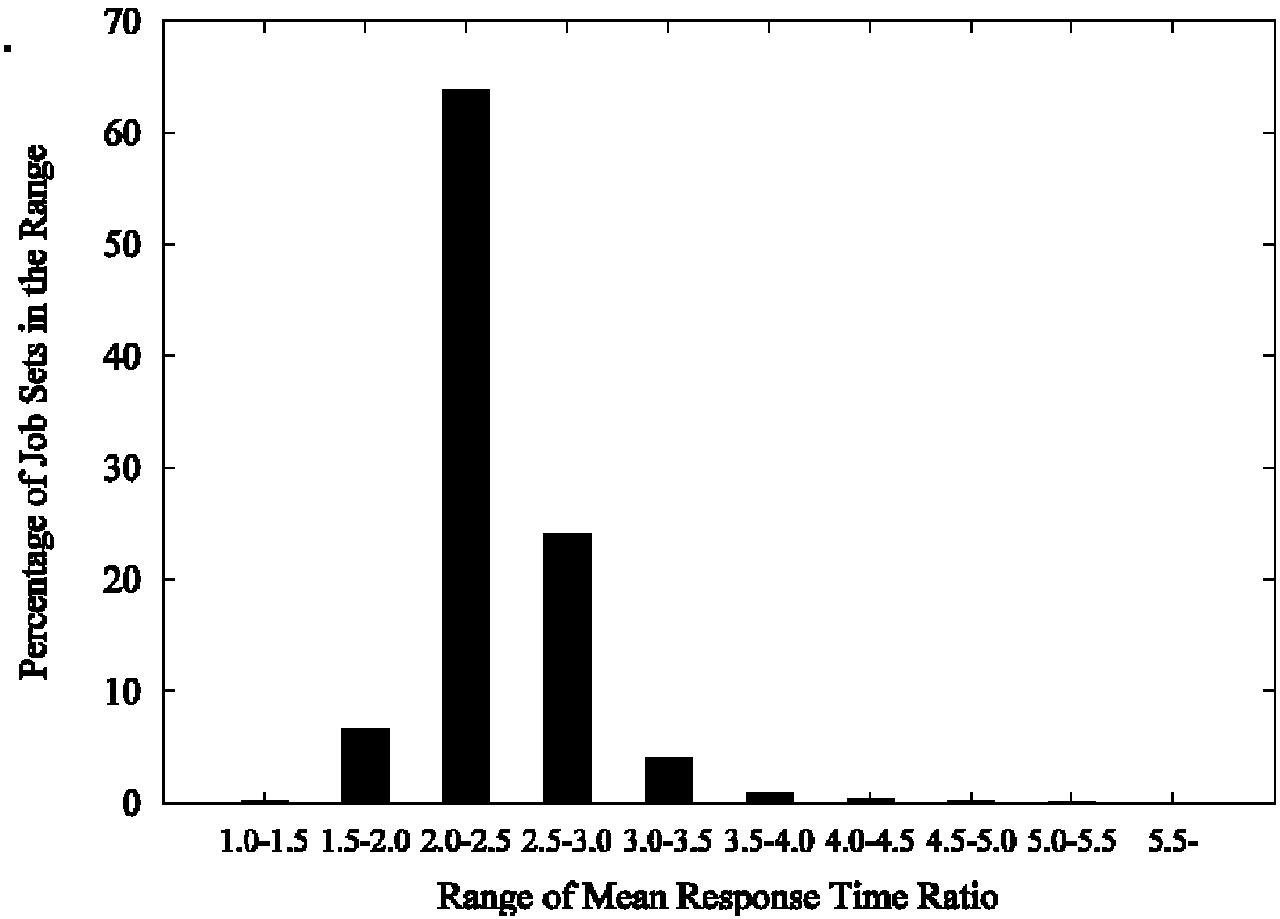
Simulation: Makespan

- The combination of A-GREEDY and DEQ+RR guarantees that the makespan is within a constant factor of the optimal makespan. But the theoretical constant is large.
- Simulations indicate that the constant should be less than 2 in practice.



Simulation: Mean Completion Time

- The combination of A-GREEDY and DEQ+RR guarantees that the mean completion time is within a constant factor of the optimal mean completion time. But the theoretical constant is large.
- Simulations indicate that the constant should be less than 3 in practice.



Current Research

- How do you provide parallelism feedback when the individual jobs are not independent?
- Example: MPEG Decoder.