Competitive Online Adaptive Scheduling for Sets of Parallel Jobs with Fairness and Efficiency

Hongyang Sun @ Institut de Recherche en Informatique de Toulouse (IRIT)

Joint work with

Wen-Jing Hsu, Yangjie Cao @ Nanyang Technological University, Singapore

Introduction

- Multi-core processors and supercomputers led to massive development of parallel applications.
- Efficient scheduling of parallel jobs is an important and challenging task for high-performance computing environments.

Introduction

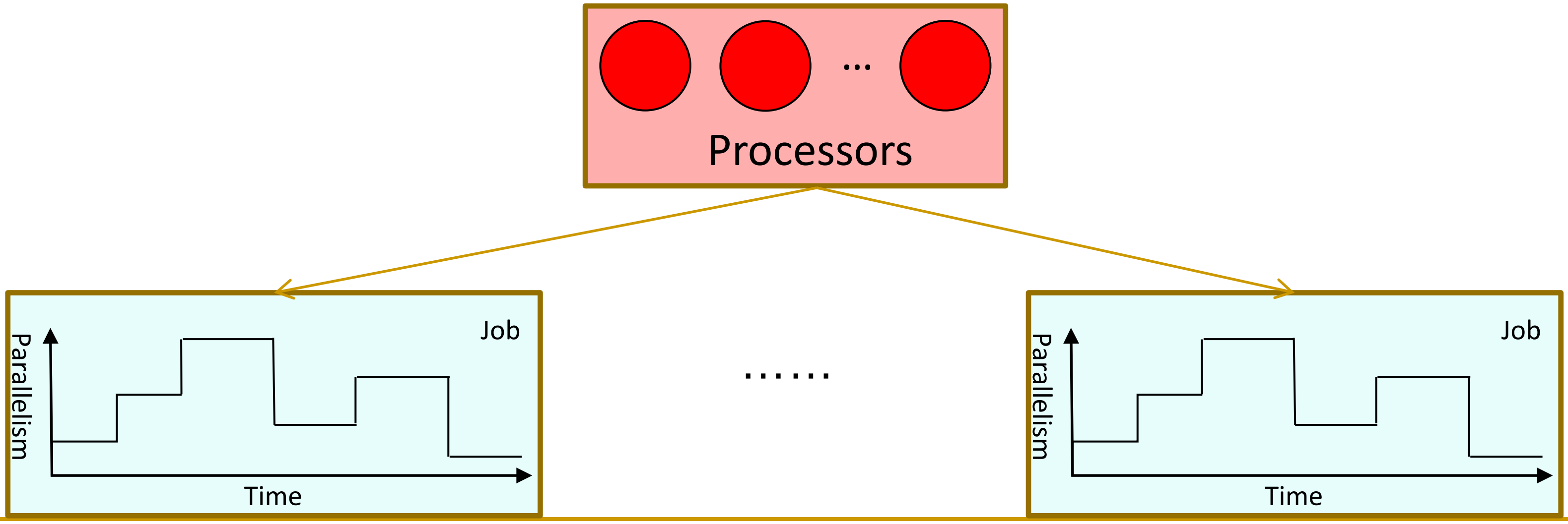
- Multi-core processors and supercomputers led to massive development of parallel applications.
- Efficient scheduling of parallel jobs is an important and challenging task for high-performance computing environments.
- This work studies scheduling on multiprocessor-based platforms
  - for ***multiple sets*** of parallel jobs
  - with ***fairness and efficiency***
  - in ***online nonclairvoyant adaptive*** manner

Outline

- Models and Objective
- Scheduling Algorithms
- Performance Analysis
- Simulation Results
- Remarks

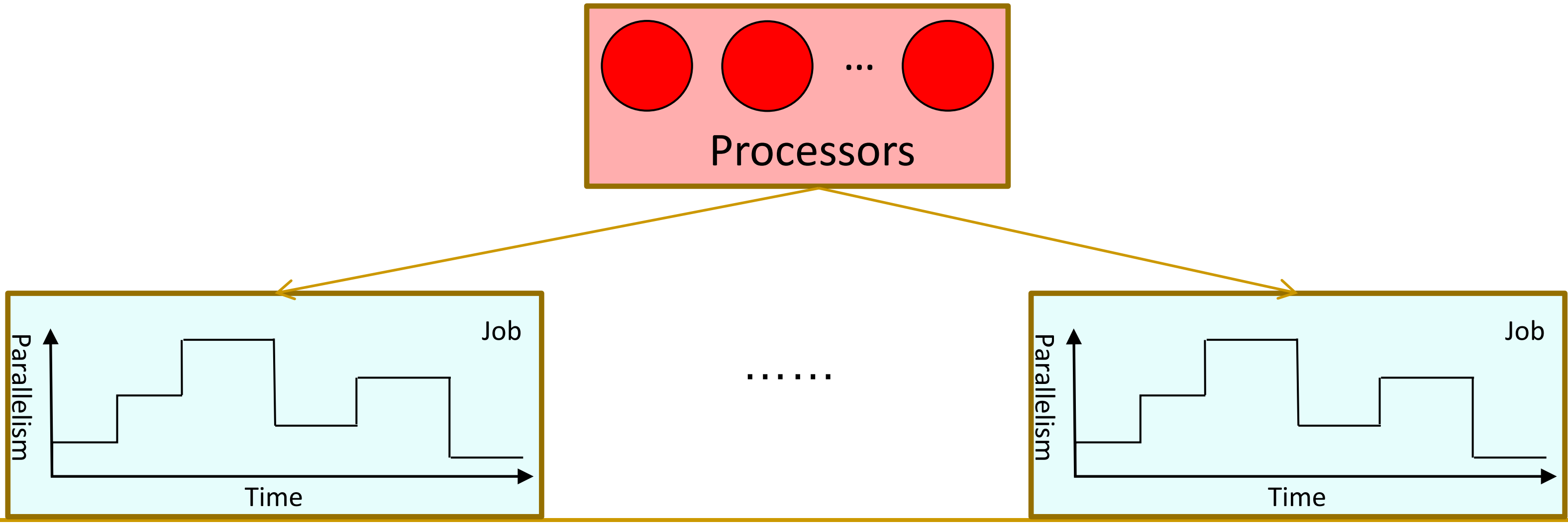
Basic Scheduling Model

- Each parallel job is assumed to have ***time-varying*** parallelism.
- Multiple jobs are to be scheduled on a set of identical processors.



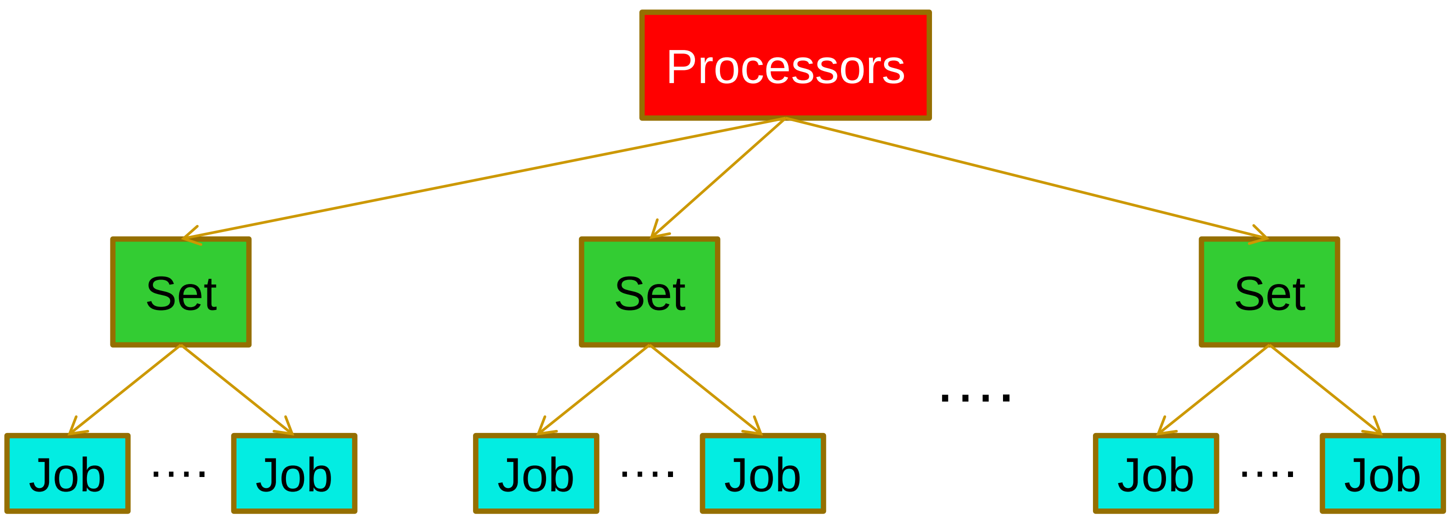
Basic Scheduling Model

- Each parallel job is assumed to have ***time-varying*** parallelism.
- Multiple jobs are to be scheduled on a set of identical processors.
- ***Problem***: Decide how many processors to allocate to each job.
  - ***Online*** – Don't know future job arrivals.
  - ***Nonclairvoyant*** – Don't know jobs' future parallelism & remaining work.
  - ***Adaptive*** – Processors allocations can be changed over time (*malleable*).



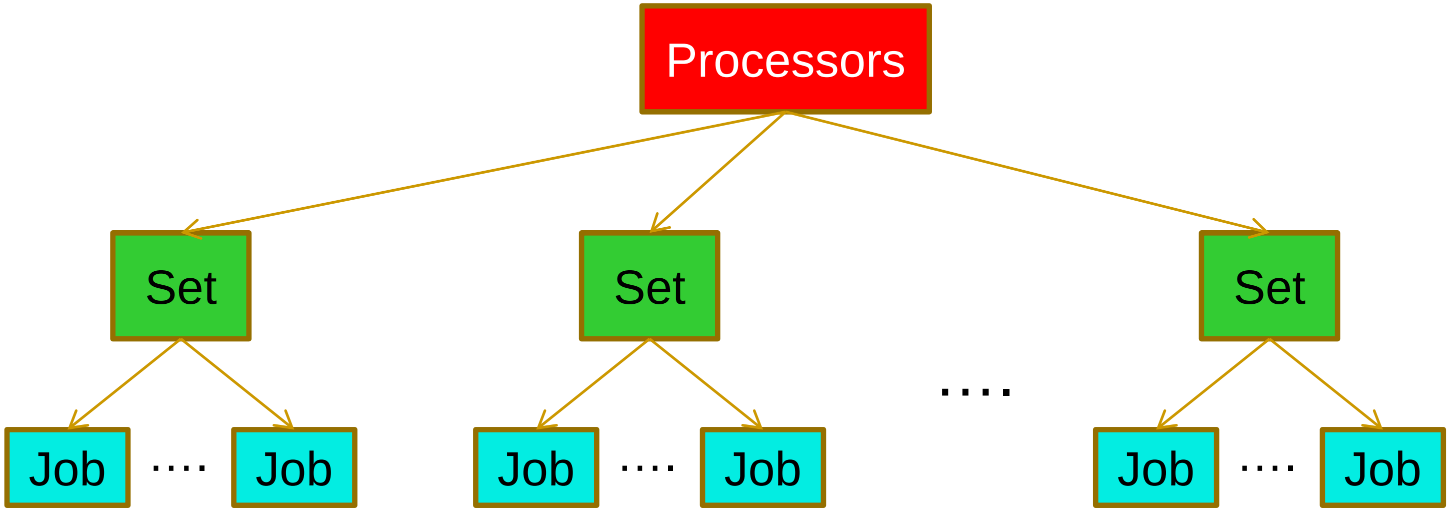
Two-Level Scheduling Model

- Jobs are further grouped into ***sets***.
  - A set of jobs may belong to a particular user.
- Multiple sets of jobs need to be scheduled on the processors.
- ***Problem***: Decide how many processors to allocate to each set and how many processors to allocate to each job within a set.



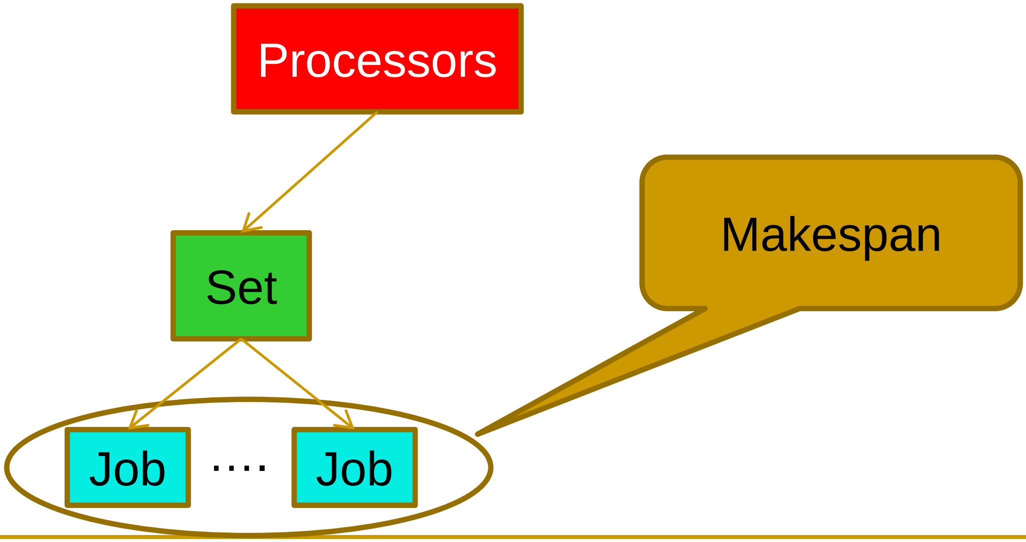
Objective Function

- Minimize total response time of all job sets, or ***set response time*** [Robert & Schabanel, 2007].
  - Response time of one set: duration between the release time of first job to the completion of last job in the set.
  - Combines two other popular metrics: makespan and total response time.



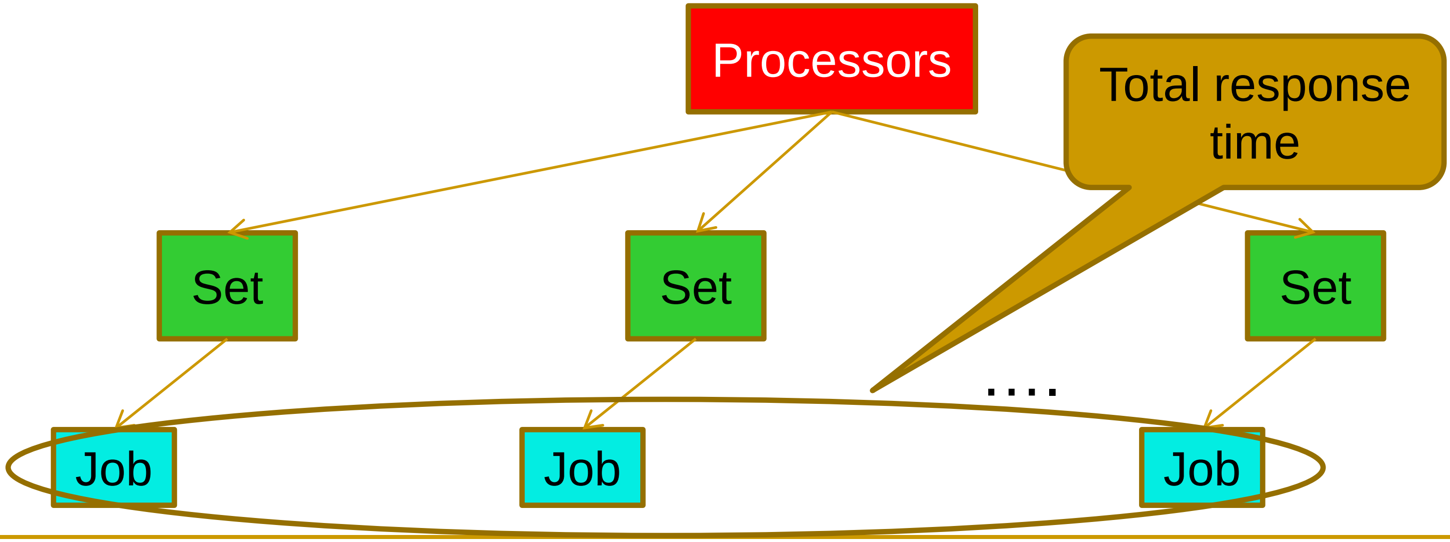
Objective Function

- Minimize total response time of all job sets, or ***set response time*** [Robert & Schabanel, 2007].
  - Response time of one set: duration between the release time of first job to the completion of last job in the set.
  - Combines two other popular metrics: makespan and total response time.
    - There is only 1 job set → ***Makespan*** of all jobs *(measures efficiency).*



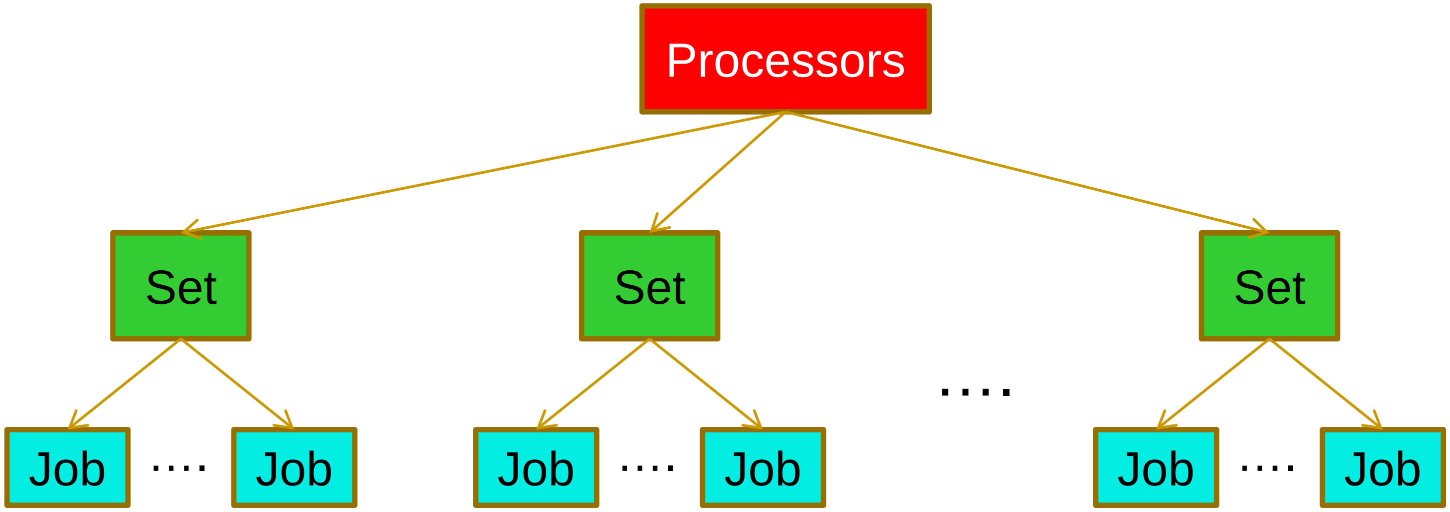
Objective Function

- Minimize total response time of all job sets, or ***set response time*** [Robert & Schabanel, 2007].
  - Response time of one set: duration between the release time of first job to the completion of last job in the set.
  - Combines two other popular metrics: makespan and total response time.
    - There is only 1 job set → ***Makespan*** of all jobs *(measures efficiency).*
    - Each set has only 1 job → ***Total response time*** of all jobs *(measures fairness).*



Objective Function

- Minimize total response time of all job sets, or ***set response time*** [Robert & Schabanel, 2007].
  - Response time of one set: duration between the release time of first job to the completion of last job in the set.
  - Combines two other popular metrics: makespan and total response time.
    - There is only 1 job set → ***Makespan*** of all jobs *(measures efficiency).*
    - Each set has only 1 job → ***Total response time*** of all jobs *(measures fairness).*
  - Benchmark for both ***fairness*** *(among sets)* and ***efficiency*** *(within sets)*.

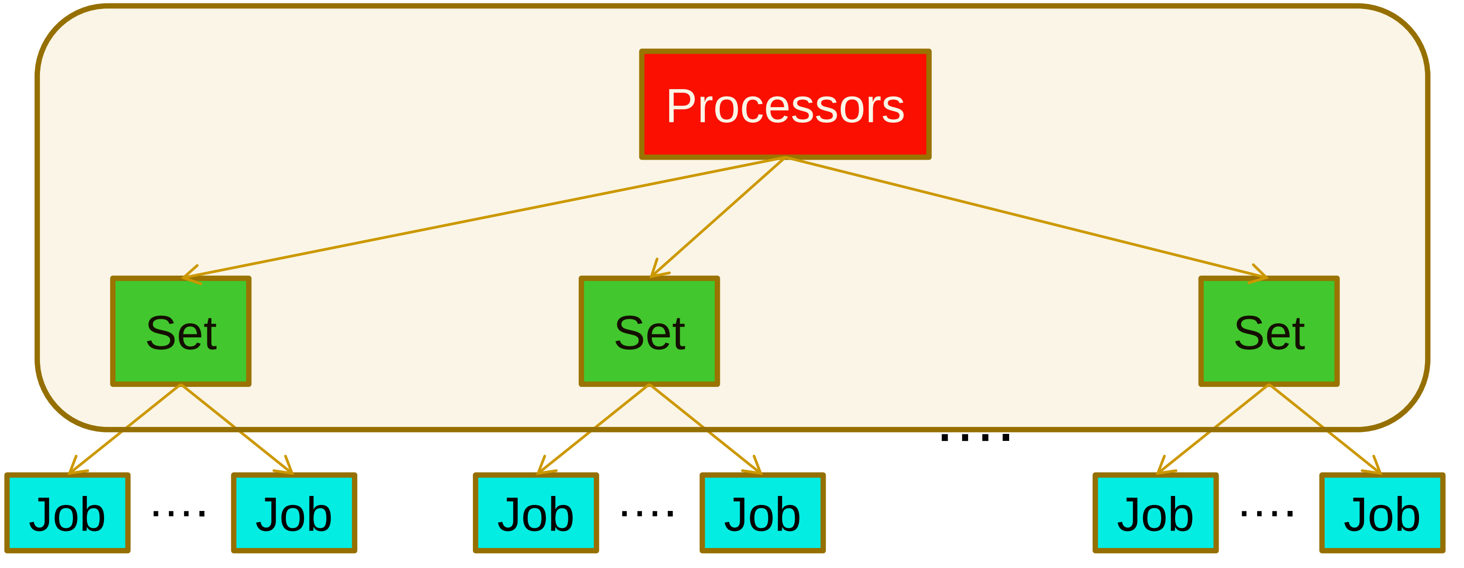


Outline

- Models and Objective
- Scheduling Algorithms
- Performance Analysis
- Simulation Results
- Remarks

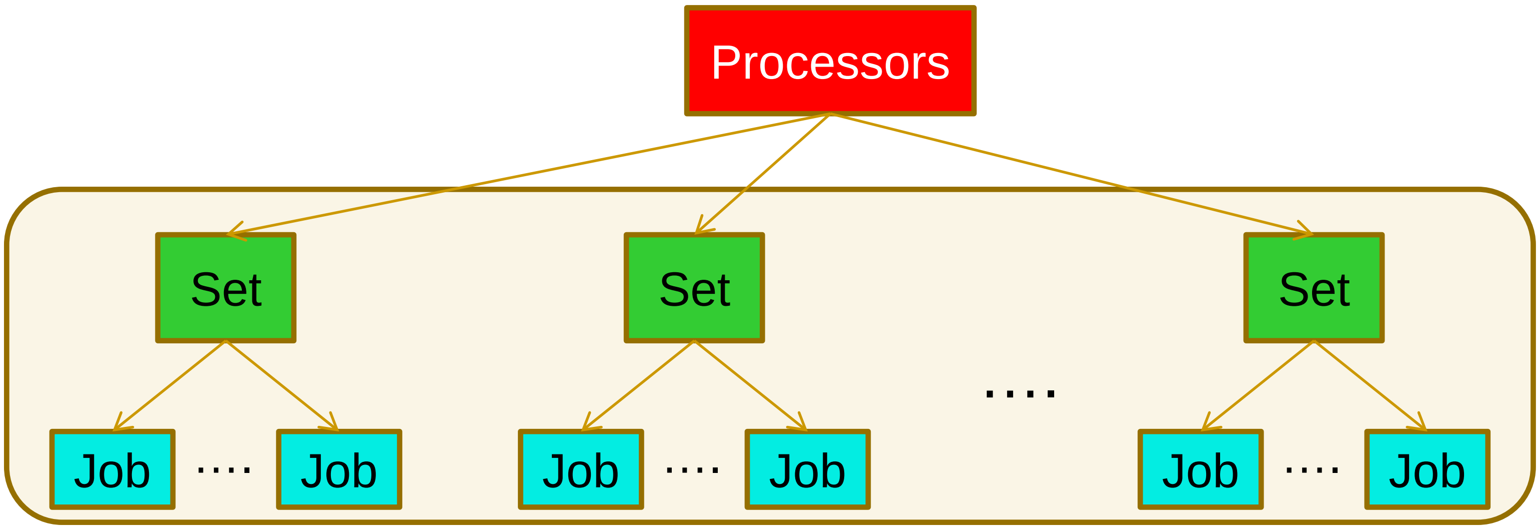
Overview of Scheduling Algorithm

- Scheduling for job sets
  - ***Equi-Partitioning (EQUI)*** algorithm: evenly divides all available processors among all active job sets at any time.
  - ***Fairness*** by simple definition with performance guarantees.



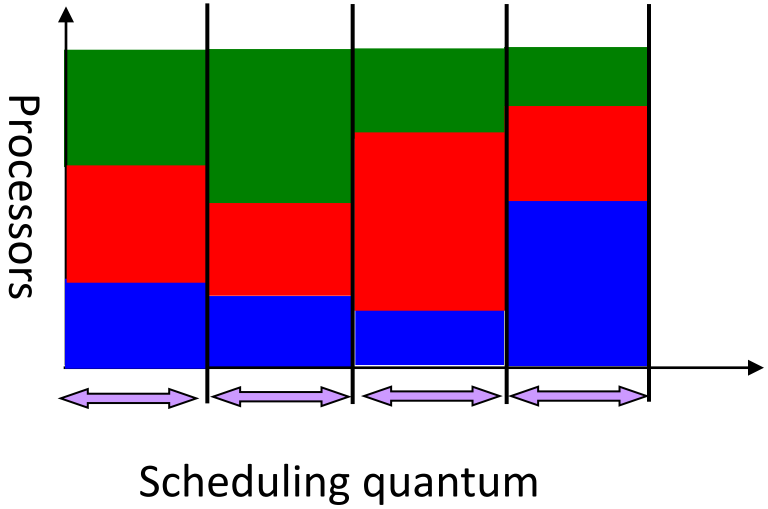
Overview of Scheduling Algorithm

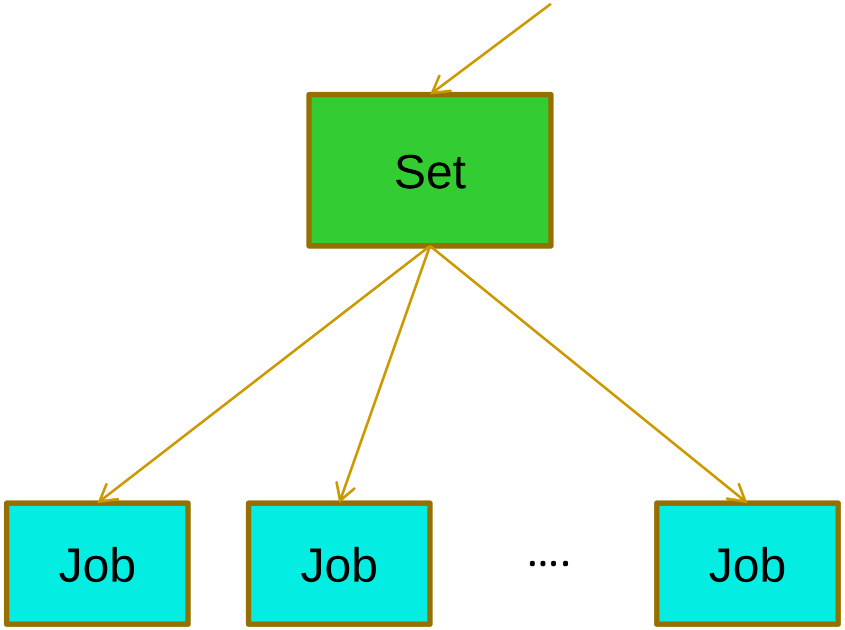
- Scheduling for jobs within each set
  - ***Feedback-driven adaptive*** algorithms: reallocates processors periodically among the jobs based on past execution of each job (feedbacks).
  - Provable ***efficiency*** in terms of processor utilization.



Feedback-Driven Adaptive Schedulers

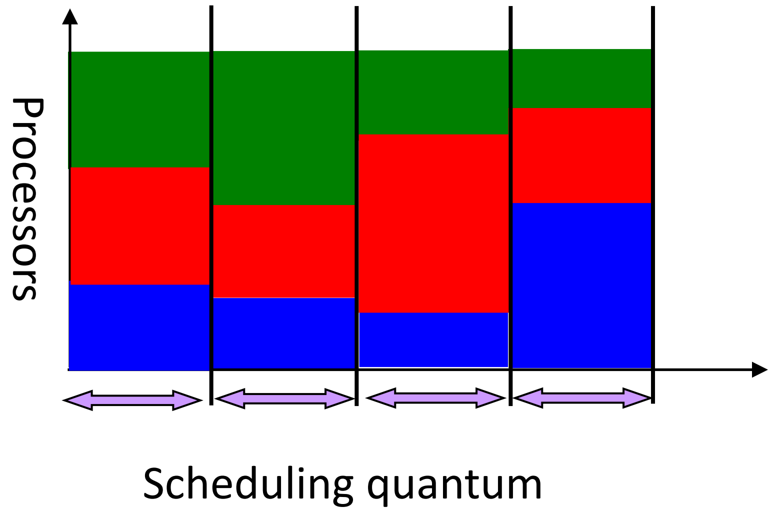
- Processors are reallocated after each ***scheduling quantum.***

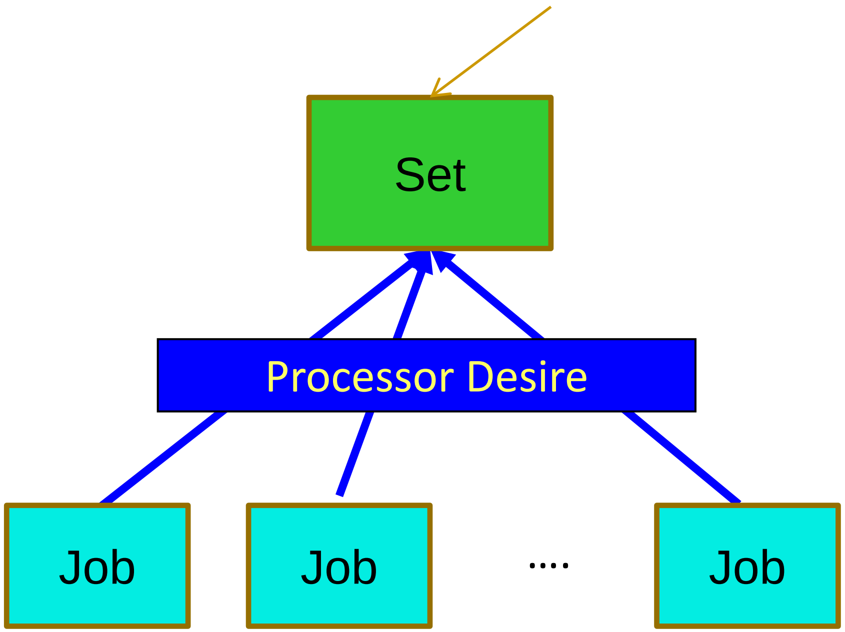




Feedback-Driven Adaptive Schedulers

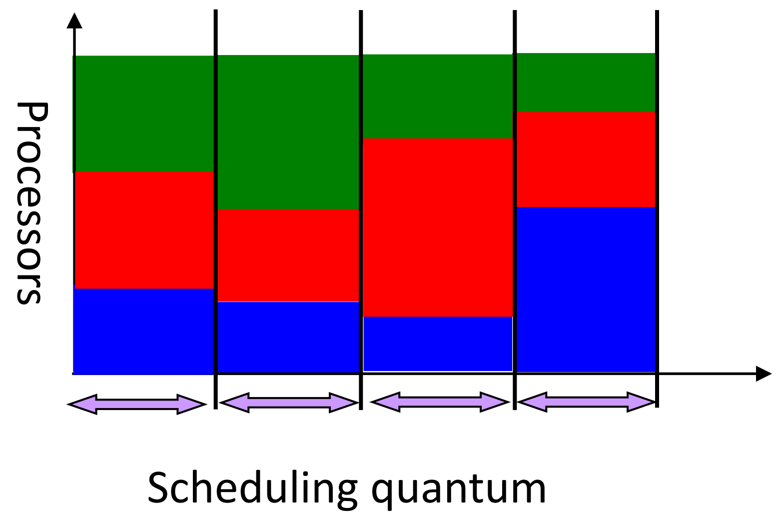
- Processors are reallocated after each ***scheduling quantum.***
- Each job computes its ***processor desire*** after a quantum as feedback.

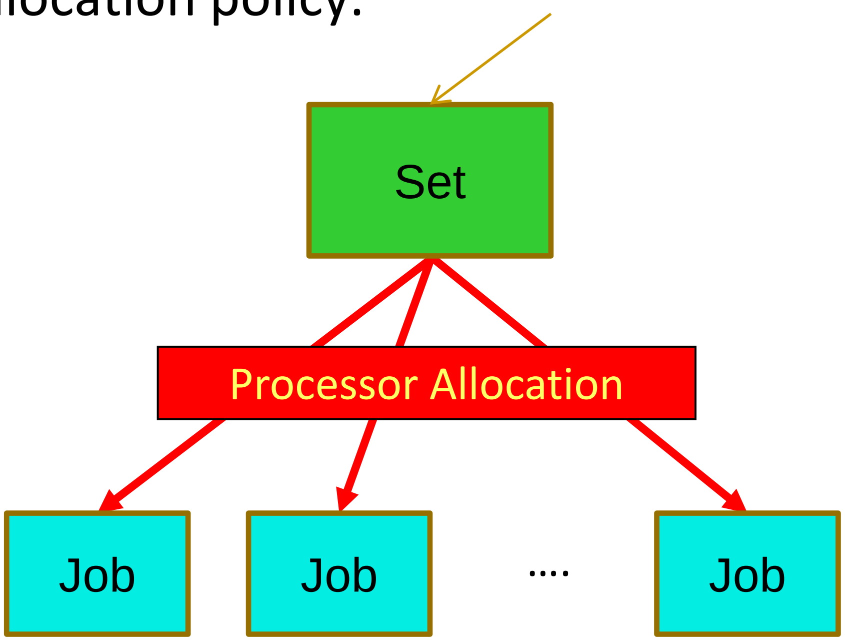




Feedback-Driven Adaptive Schedulers

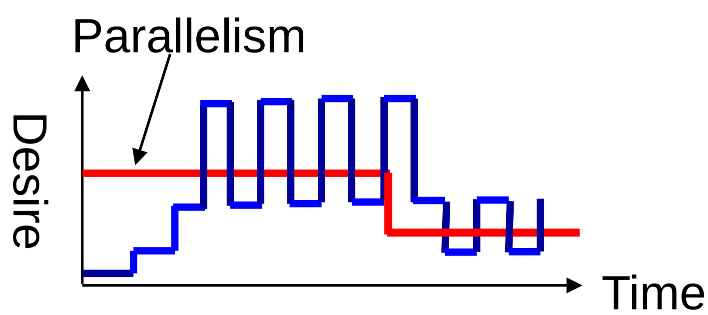
- Processors are reallocated after each ***scheduling quantum.***
- Each job computes its ***processor desire*** after a quantum as feedback.
- Set scheduler decides ***processor allocation*** for each job based on the feedbacks of all jobs and some allocation policy.



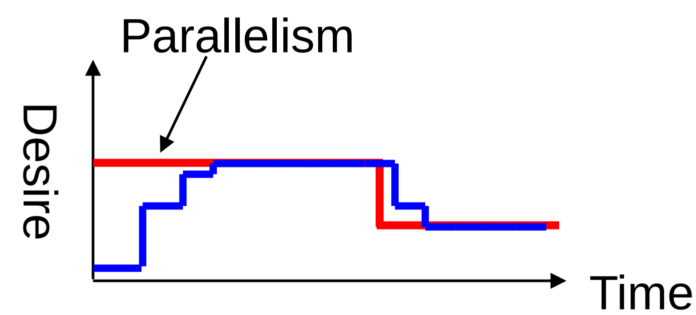


Processor Desire Calculations

- ***A-Greedy*** [Agrawal et al. 2006]
  - Uses a ***multiplicative-increase multiplicative-decrease*** approach based on processor utilization.
  - Increase processor desire if utilization is high (> a threshold); Decrease processor desire if utilization is low (< a threshold).
  - Provably ***high overall processor utilization***, but desires may be ***unstable***.

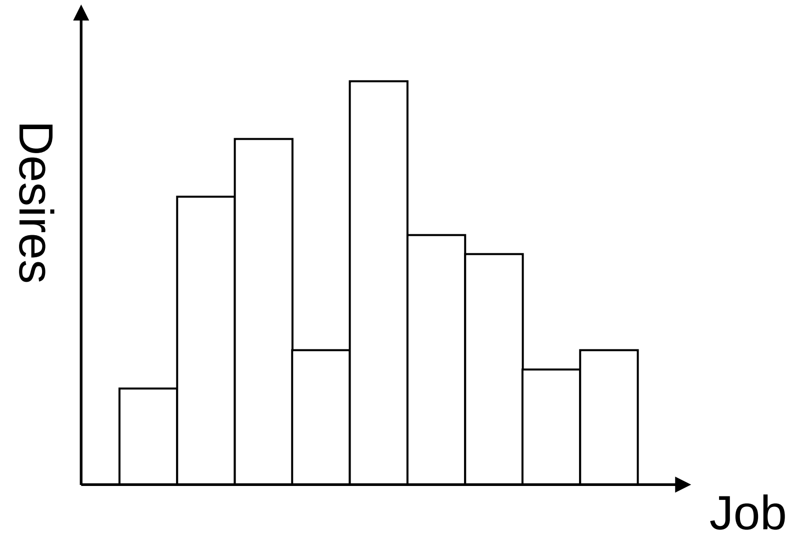


- ***A-Control*** [Sun et al. 2010]
  - Uses a ***control-theoretic approach*** based on both processor utilization and job progress.
  - Sets processor desire to be a linear combination of job's average parallelism and processor desire in previous quantum.
  - Settles the instability problem and has been shown to have ***better responsiveness.***



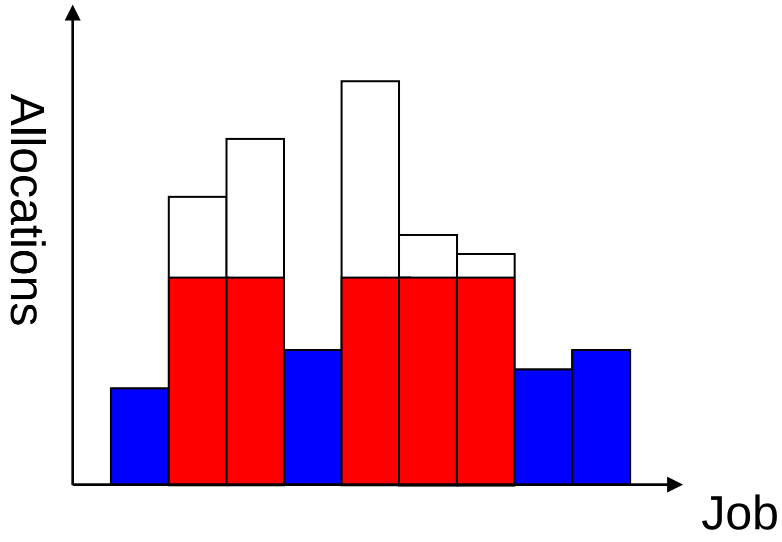
Processor Allocation Policy

- ***Dynamic Equi-Partitioning (DEQ)*** [McCann et al. 1993]
  - Give each unsatisfied job one processor in ***round-robin*** fashion until all processors are allocated or all jobs are satisfied.



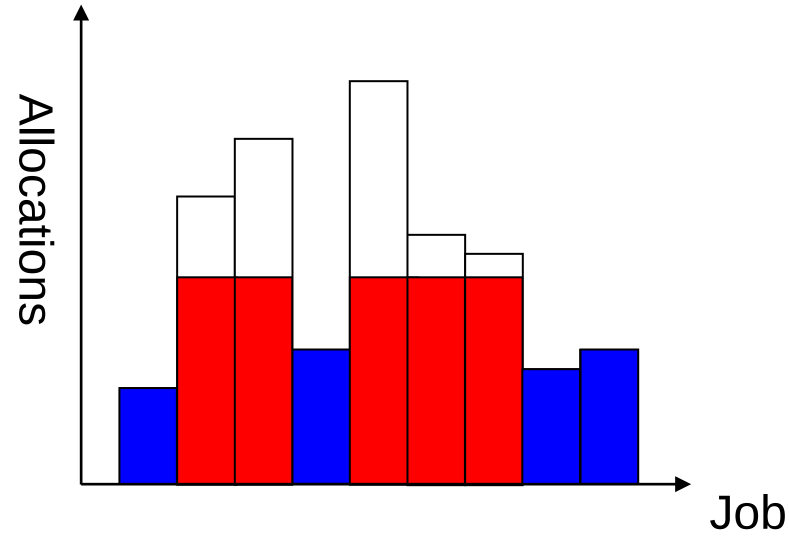
Processor Allocation Policy

- ***Dynamic Equi-Partitioning (DEQ)*** [McCann et al. 1993]
  - Give each unsatisfied job one processor in ***round-robin*** fashion until all processors are allocated or all jobs are satisfied.
  - Jobs with low desires will be ***satisfied*** and jobs with high desires will be ***deprived*** and get an approximate ***equal share***.



Processor Allocation Policy

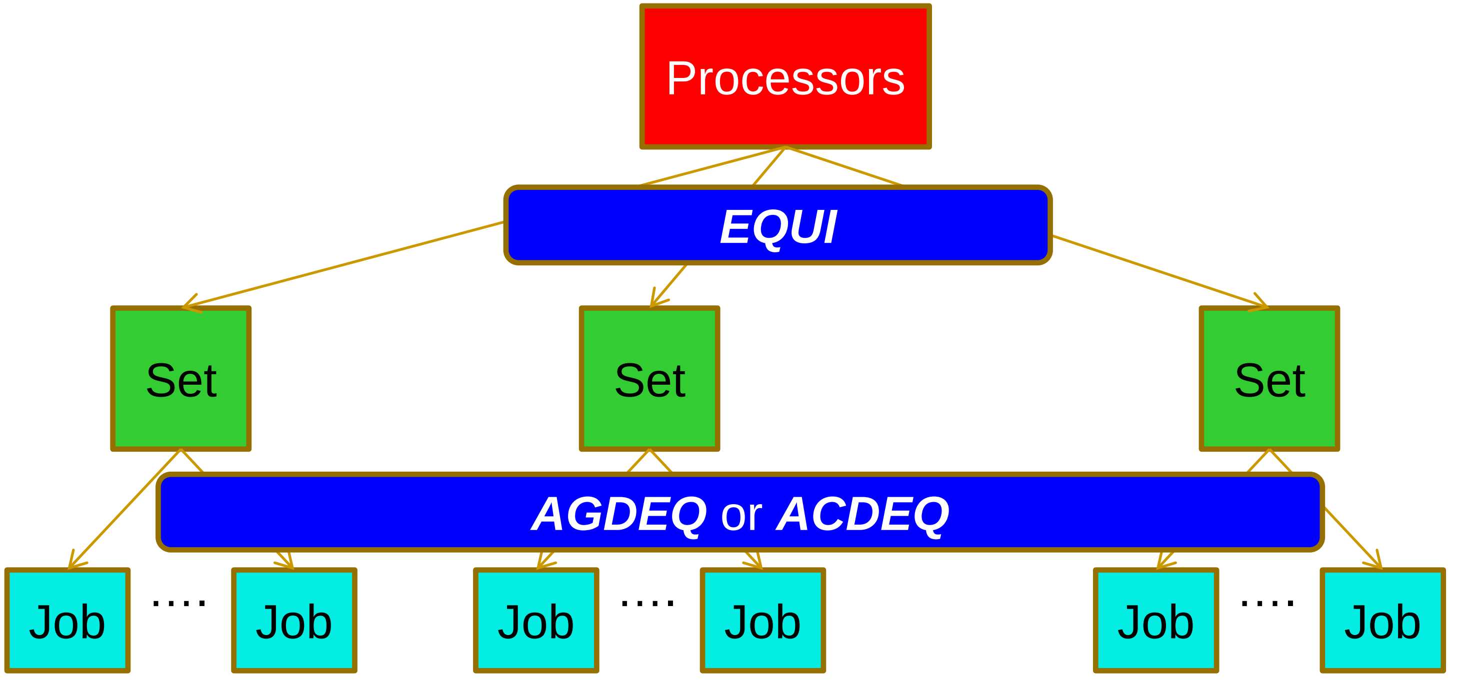
- ***Dynamic Equi-Partitioning (DEQ)*** [McCann et al. 1993]
  - Give each unsatisfied job one processor in ***round-robin*** fashion until all processors are allocated or all jobs are satisfied.
  - Jobs with low desires will be ***satisfied*** and jobs with high desires will be ***deprived*** and get an approximate ***equal share***.



- Combining ***A-Greedy*** or ***A-Control*** with ***DEQ***, we get ***feedback-driven adaptive*** schedulers ***AGDEQ*** or ***ACDEQ*** for scheduling each individual job set.

Fair and Efficient Adaptive Scheduler

- Combining ***AGDEQ*** or ***ACDEQ*** with ***EQUI***, we get ***fair and efficient*** schedulers ***EQUI-AGDEQ*** and ***EQUI-ACDEQ*** for scheduling multiple sets of jobs.



Outline

- Models and Objective
- Scheduling Algorithms
- Performance Analysis
- Simulation Results
- Remarks

Performance Analysis

- ***Competitive analysis***: An online algorithm is ***c-competitive*** if there is a constant ***b**, s.t.* the set response time is at most ***c times*** that of the optimal offline algorithm for all input instances:
  - $H_{alg}(J) \leq c \bullet H_{opt}(J) + b$
- When jobs can have ***different release times***, it is well-known that no good competitive ratio is achievable even for minimizing total response time (Lower Bound $\boldsymbol{\Omega(\sqrt{n})}$) .
- ***Resource augmentation analysis***: An online algorithm is ***s-speed c-competitive*** if its set response time when using ***s times*** faster processors is at most ***c times*** that of the optimal.
  - $H_{alg(s)}(J) \leq c \bullet H_{opt(1)}(J) + b$
- Competitive ratio is said to be ***strong*** if ***b = 0***; otherwise, it is achieved in the ***asymptotic*** sense.

Existing Results for **EQUI**-Based Algo.

- Total response time minimization.
  - ***EQUI*** is ***(2+√3)-comp.*** for batched jobs [Edmonds 1997].
  - ***EQUI*** is ***(2+ε)-speed (2+4/ε)-comp.*** [Edmonds 1999].
- Makespan minimization
  - ***EQUI*** is **O(lg*n*/lglg*n*)-*comp*.** for batched jobs, where ***n*** is the number of jobs in the set [Robert & Schabanel, 2007].
- Set response time minimization
  - ***EQUI-EQUI*** is **(*2*+√*3*+o(*1*))•O(lg*n*/lglg*n*)-*comp.*** for batched job sets, where *n* is the maximum number of jobs in any set [Robert & Schabanel, 2007].

Existing Results for **EQUI**-Based Algo.

- Total response time minimization.
  - ***EQUI*** is ***(2+√3)-comp.*** for batched jobs [Edmonds 1997].
  - ***EQUI*** is ***(2+ε)-speed (2+4/ε)-comp.*** [Edmonds 1999].
- Makespan minimization
  - ***EQUI*** is **O(lg*n*/lglg*n*)*-comp*.** for batched jobs, where ***n*** is the number of jobs in the set [Robert & Schabanel, 2007].
- Set response time minimization
  - ***EQUI-EQUI*** is ***(2+√3+o(1))•O(lgn/lglgn)-comp.*** for batched job sets, where *n* is the maximum number of jobs in any set [Robert & Schabanel, 2007].

Set response time ratio seems to combine the total response time ratio and the makespan ratio? Indeed!

Generalized Analysis for **EQUI-XY**

Generalized Analysis for **EQUI-XY**

- ***Property*** for feedback scheduler **X** that calculates jobs' processor desires:
  - $a \leq \alpha \cdot w$
  - $t_S \leq \beta \cdot l$

w : total work of job
l : total span of job
a : total processor allocation for the job
t_S : total processing time for the job when it is satisfied

Generalized Analysis for **EQUI-XY**

- ***Property*** for feedback scheduler **X** that calculates jobs' processor desires:
  - $a \leq \alpha \cdot w$
  - $t_S \leq \beta \cdot l$

w : total work of job
l : total span of job
a : total processor allocation for the job
t_S : total processing time for the job when it is satisfied

- ***Property*** for adaptive processor allocation policy **Y**:
  - ***Conservative***: never allocates more processors to a job than desired.
  - ***Non-idle***: never idles processors when a job set is deprived.

Generalized Analysis for **EQUI-XY**

- ***Property*** for feedback scheduler **X** that calculates jobs' processor desires:
  - $a \leq \alpha \cdot w$
  - $t_S \leq \beta \cdot l$

w : total work of job
l : total span of job
a : total processor allocation for the job
t_S : total processing time for the job when it is satisfied

- ***Property*** for adaptive processor allocation policy **Y**:
  - ***Conservative***: never allocates more processors to a job than desired.
  - ***Non-idle***: never idles processors when a job set is deprived.
- **Theorem. *EQUI-XY* achieves the following results:**
  - *2*(α+β)*-comp.* for batched job sets
  - (*2*α+*ε*)*-speed* (*2+2*(*2*α+β)/*ε*)*-comp.* for arbitrary released job sets
- The batched analysis relies on two lower bounds (*squashed* and *height* bounds). The general case uses the standard *potential function* argument.
- *Remarks.* (α+β) is the comp. ratio of XY for makespan.

Results for **EQUI-XY** Algo

	α	**β**
EQUI-AGDEQ	$(1+\rho)/\delta$	$2/(1-\delta)$
EQUI-ACDEQ	$c+1$	$c+1$
EQUI-EQUI	$O(\lg n/\lg\lg n)$	$O(\lg n/\lg\lg n)$

Achieved in amortized sense

Results for **EQUI-XY** Algo

	α	β
EQUI-AGDEQ	$(1+\rho)/\delta$	$2/(1-\delta)$
EQUI-ACDEQ	$c+1$	$c+1$
EQUI-EQUI	$O(\lg n/\lg\lg n)$	$O(\lg n/\lg\lg n)$

Achieved in amortized sense

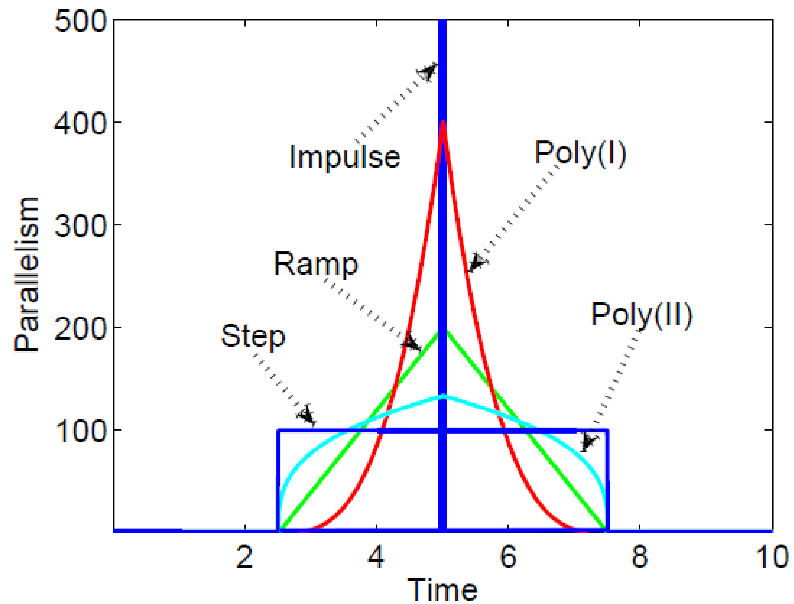
- **Theorem.** ***EQUI-AGDEQ*** and ***EQUI-ACDEQ*** are
  - $O(1)$-*competitive* for batched job sets.
  - $O(1)$-*speed* $O(1)$-*comp.* for arbitrary released job sets.
- The bounds are achieved in ***asymptotic*** sense when jobs are large enough; otherwise constant additive factor dominates, i.e., $H_{AGDEQ}(J) = O(1) \cdot H_{OPT}(J) + b$

Outline

- Models and Objective
- Scheduling Algorithms
- Performance Analysis
- Simulation Results
- Remarks

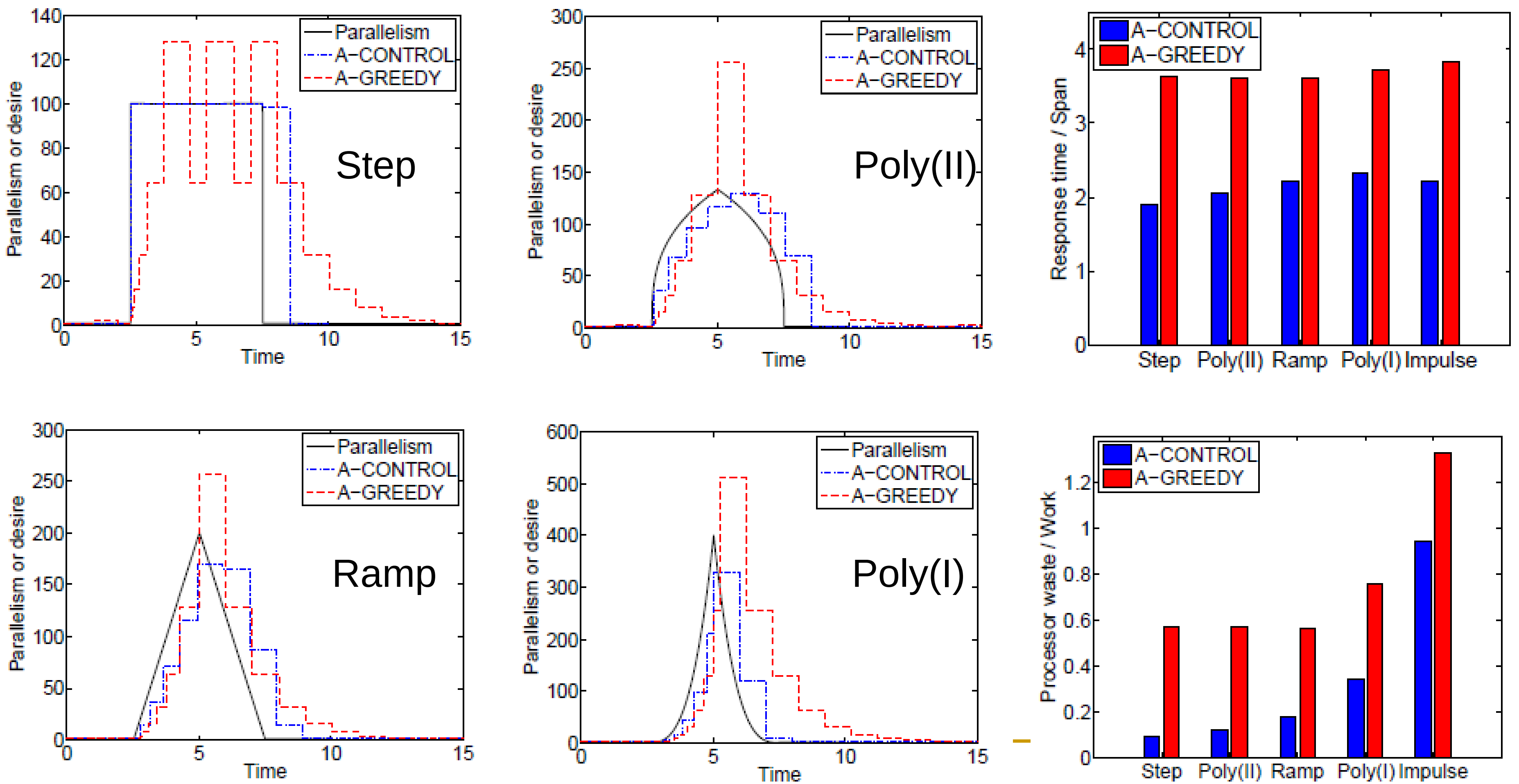
Simulation Setup

- Parallel Workloads:
  - Job model from the parallel workload archive.
  - The following regular patterns are used to generate jobs' internal parallelism variations.



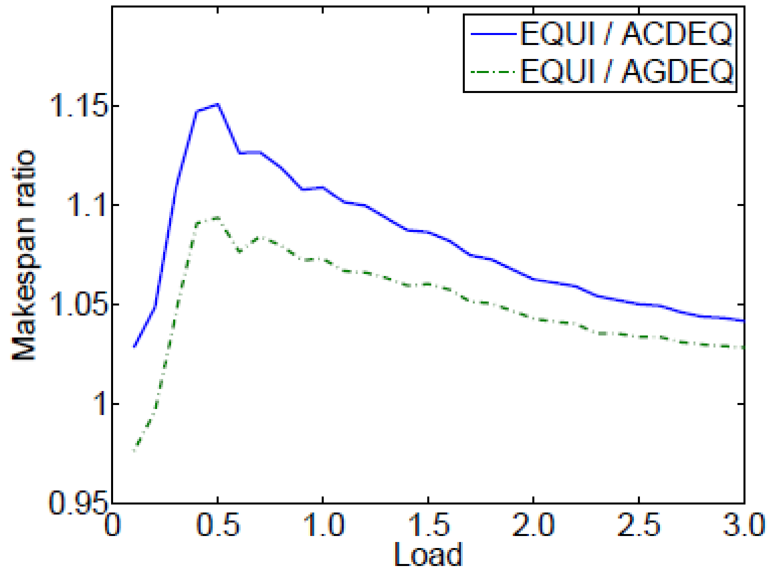
Results for Individual Job

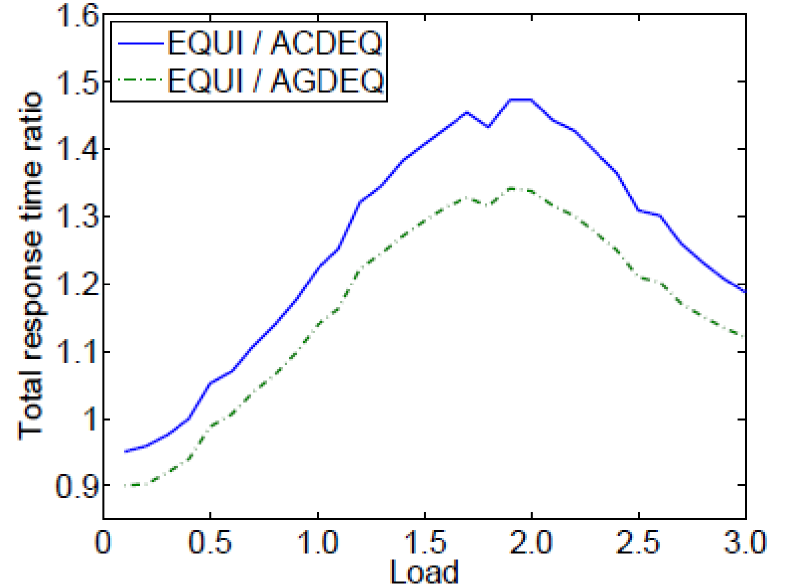
- A-Greedy v.s. A-Control on a single job set.



Results for a Single Job Set

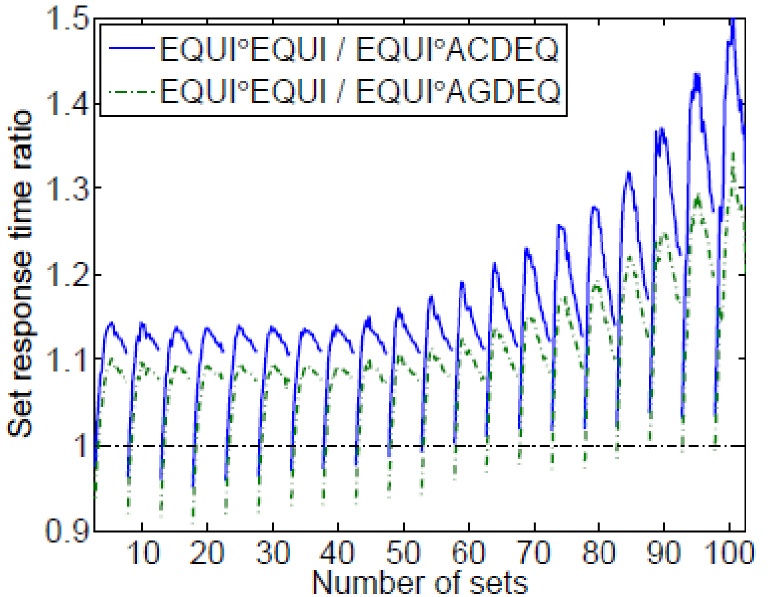
- Feedback-driven schedulers v.s. EQUI on a single job set.
  - Feedback-driven schedulers outperform EQUI at ***medium loads***.
  - At ***light loads,*** enough processors for every job; no feedback is needed.
  - At ***heavy loads,*** not enough processors even with feedbacks.



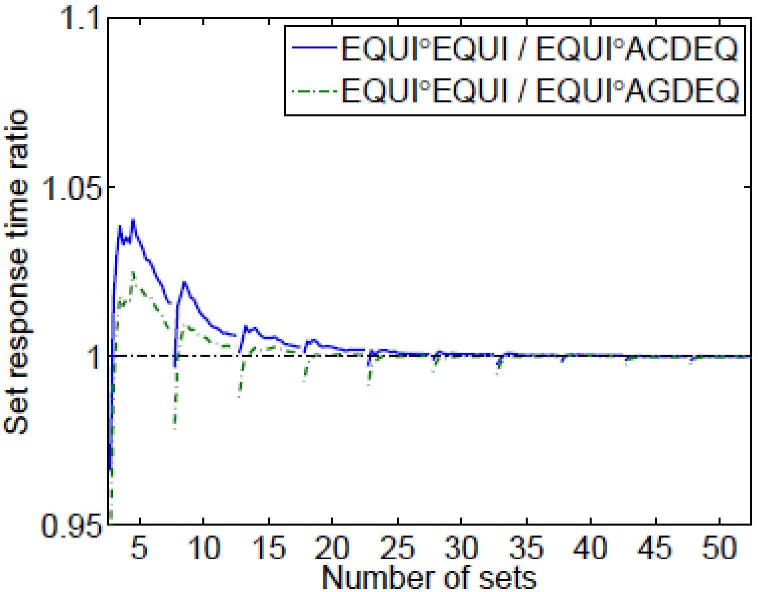


Results for Multiple Job Sets

- EQUI-Feedback v.s. EQUI-EQUI on multiple job sets. (64 procs)
  - Fair and efficient schedulers outperform EQUI-EQUI in most cases.
  - Combines results of makespan and total response time.



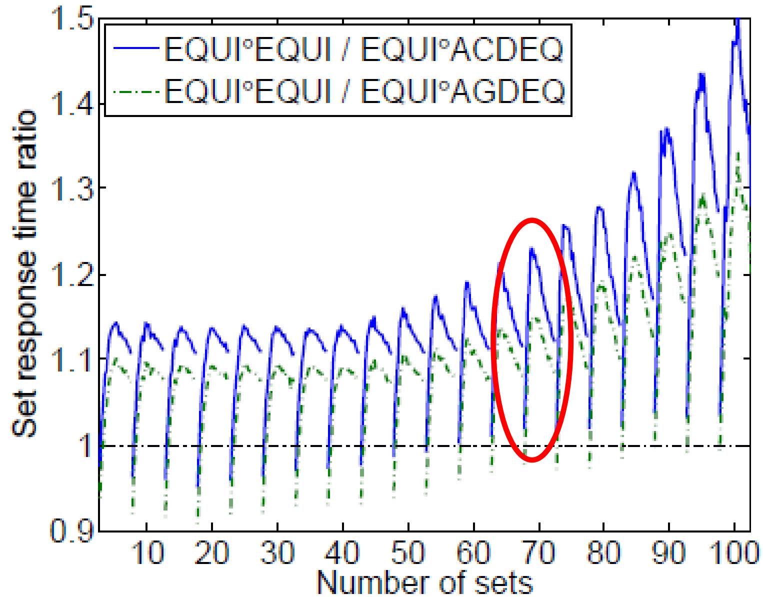
Following release time of the job model



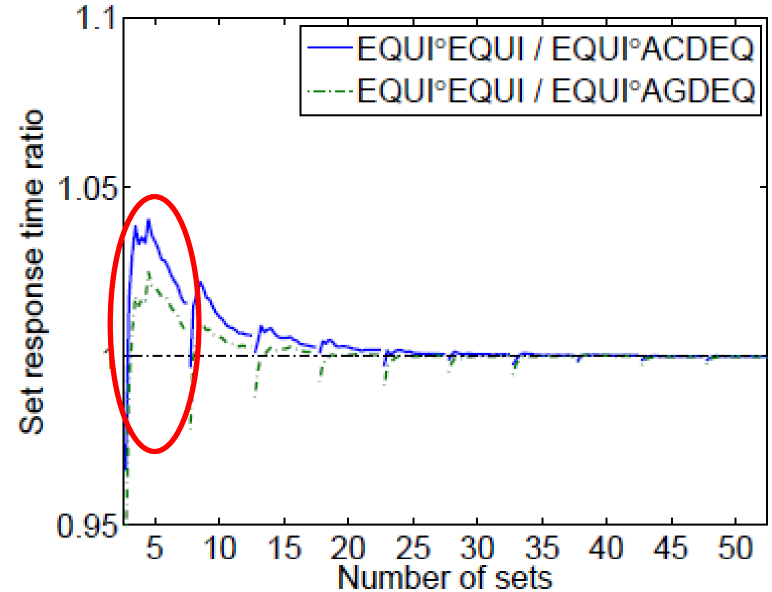
Identical release time

Results for Multiple Job Sets

- EQUI-Feedback v.s. EQUI-EQUI on multiple job sets. (64 procs)
  - Fair and efficient schedulers outperform EQUI-EQUI in most cases.
  - Combines results of makespan and total response time.



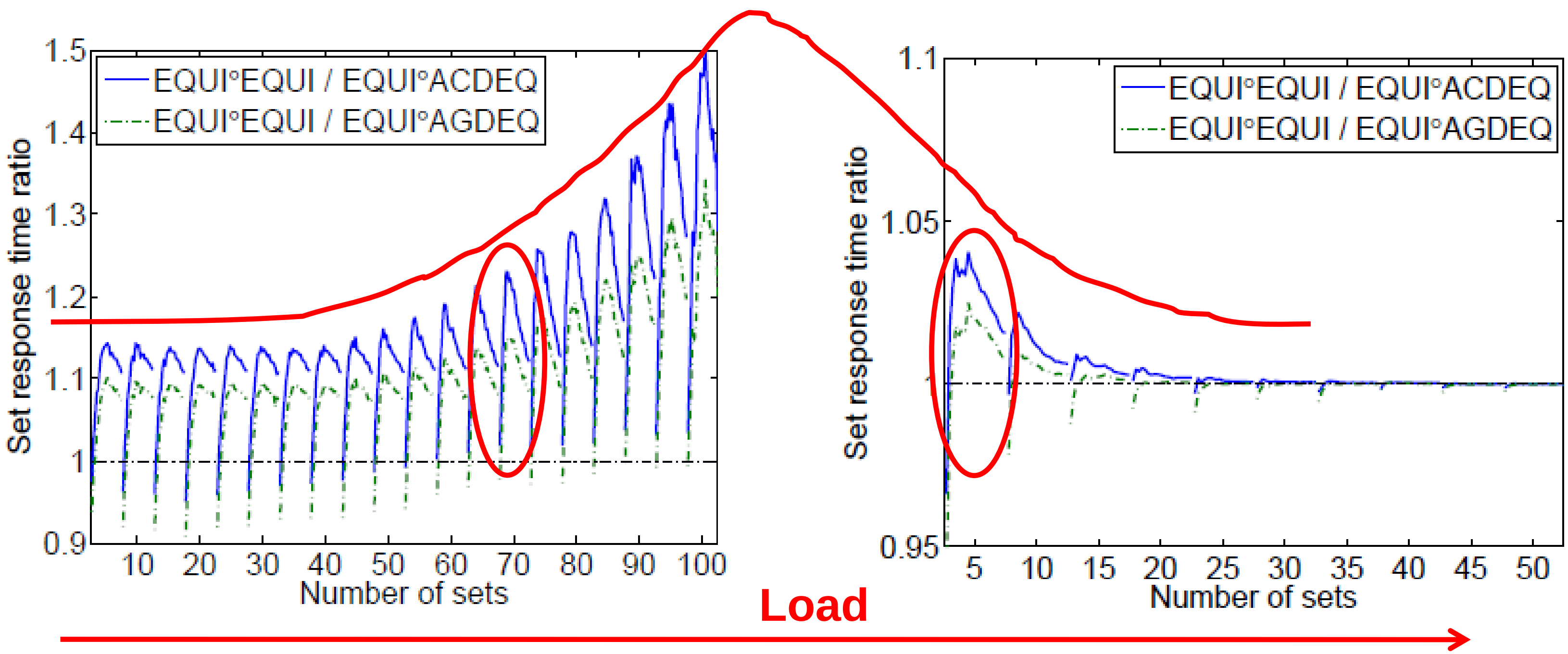
Following release time of the job model



Identical release time

Results for Multiple Job Sets

- EQUI-Feedback v.s. EQUI-EQUI on multiple job sets. (64 procs)
  - Fair and efficient schedulers outperform EQUI-EQUI in most cases.
  - Combines results of makespan and total response time.



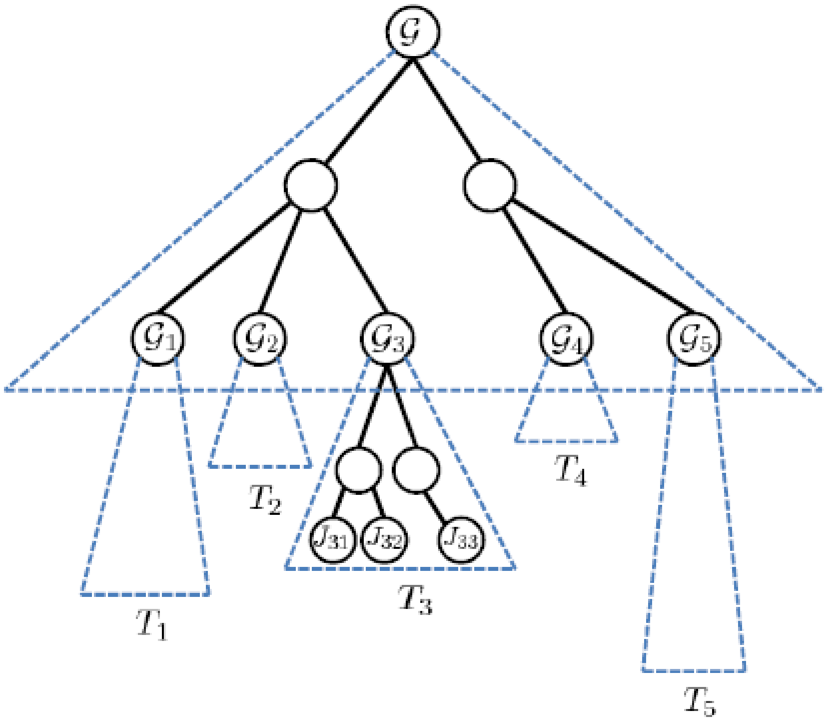
Following release time of the job model

Identical release time

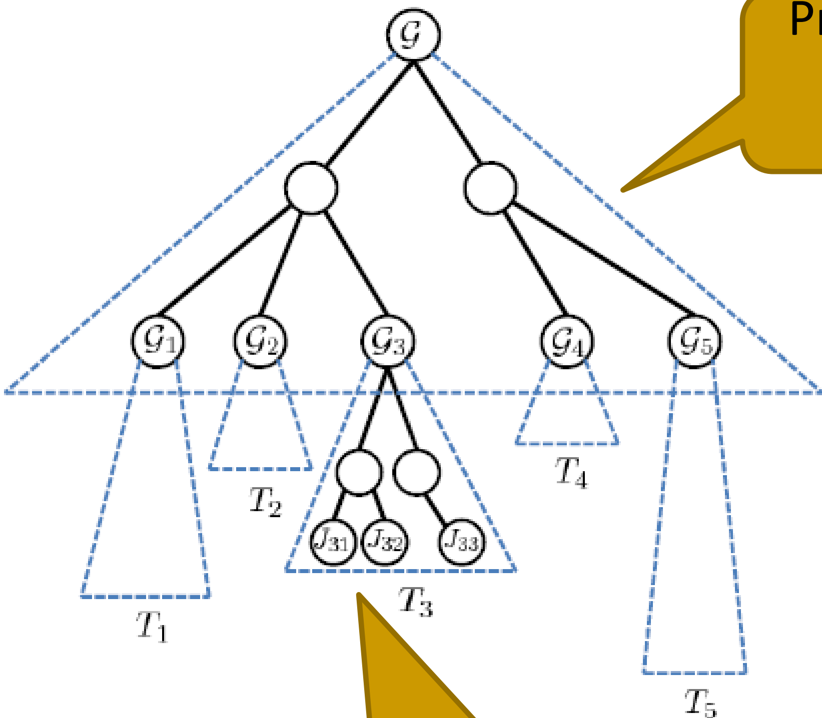
Outline

- Models and Objective
- Scheduling Algorithms
- Performance Analysis
- Simulation Results
- Remarks

Remark on Multi-Level Scheduling

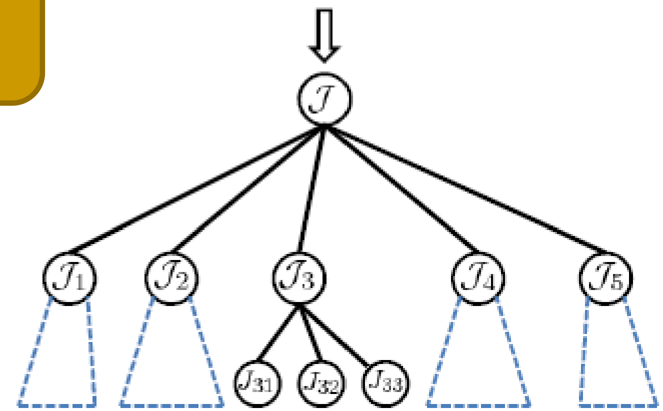


Remark on Multi-Level Scheduling

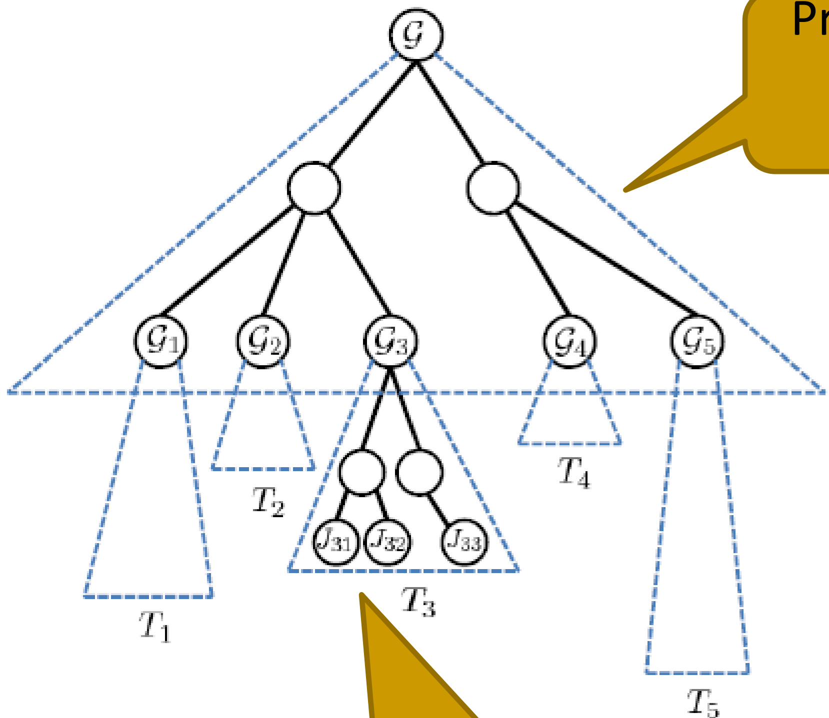


Proportional allocator = ***EQUI***

X + desire aggregator + *conservative* & *non-idle* allocator ***Y***

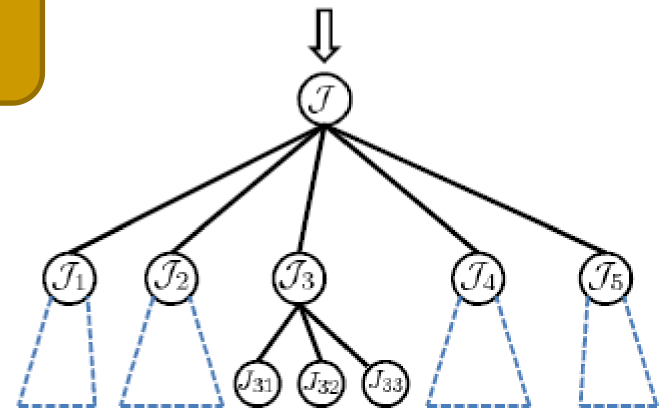


Remark on Multi-Level Scheduling



Proportional allocator = ***EQUI***

X + desire aggregator + *conservative* & *non-idle* allocator ***Y***



Reduced to ***EQUI-XY'*** in two-level scheduling, where property for ***X*** remains, and ***Y'*** is also *conservative* and *non-idle*.

Remark on Adaptive Parallel Job Scheduling

- **LAPS(β)** algorithm generalizes **EQUI** [Edmonds & Pruhs 2009].
  - Allocate processors to β fraction of jobs with latest arrival time.
  - ***s*=(1+β+*ε*)-*speed* (4*s*/β*ε*)-*comp.*** for total response time.
  - Provides tradeoff between speed augmentation and competitive ratio.

Remark on Adaptive Parallel Job Scheduling

- **LAPS(β)** algorithm generalizes **EQUI** [Edmonds & Pruhs 2009].
  - Allocate processors to β fraction of jobs with latest arrival time.
  - ***s*=(1+β+*ε*)-*speed* (4*s*/β*ε*)-*comp.*** for total response time.
  - Provides tradeoff between speed augmentation and competitive ratio.
- **LAPS(β)** can be combined with feedback-driven adaptive schedulers to improve the comp. ratios by constant factors.

Remark on Adaptive Parallel Job Scheduling

- **LAPS(β)** algorithm generalizes **EQUI** [Edmonds & Pruhs 2009].
  - Allocate processors to β fraction of jobs with latest arrival time.
  - ***s*=(1+β+*ε*)-*speed* (*4s*/β*ε*)-*comp.*** for total response time.
  - Provides tradeoff between speed augmentation and competitive ratio.
- **LAPS(β)** can be combined with feedback-driven adaptive schedulers to improve the comp. ratios by constant factors.
- **Open Question:** Does universally scalable algorithm exist? (i.e., **(1+*ε*)-*speed* O(1)-*comp.*** for any *ε>0* and the algorithm's parameter does not depend on *ε*!)

Thank you!

Questions?