Parallel Job Scheduling and Workloads

Dror Feitelson

Hebrew University

Parallel Jobs

- A set of processes that cooperate to solve a problem
  - Example: weather forecast, industrial/military simulation, scientific discovery
- Processes run in parallel on distinct processors, communicate using high-speed network
- Run to completion on dedicated processors to avoid memory problems
- Require rectangle in processorsXtime space

Parallel Job Scheduling

- Each job is a rectangle
- Given many jobs, we must schedule them to run on available processors
- This is like packing the rectangles
- Want to minimize space used, i.e. minimize used resources and fragmentation
- On-line problem: don’t know future arrivals or runtimes

Workloads

- System performance depends on the workload
  - Analogy: algorithm performance depends on the input
- Evaluation workload should be representative of real workloads
- In our case, the workload is a sequence of jobs to run
- Can use data from system accounting logs

Workloads

- System performance depends on the workload
  - Analogy: algorit the input
- Evaluation e of real work
- In our case, the wo equence of jobs to run
- Can use data from system accounting logs

- Job arrival patterns
- Job resource demands (processors and runtime)

Parallel Workloads Archive

- All large scale supercomputers maintain accounting logs
- Data includes job arrival, queue time, runtime, processors, user, and more
- Many are willing to share them

  (and shame on those who are not)
- Collection at www.cs.huji.ac.il/labs/parallel/workload/
- Uses standard format to ease use

Example: NASA iPSC/860 trace

user	*cmd*	*proc*	*runtm*	*date*	*time*
user8	cmd33	1	31	10/19/93	18:06:10
sysadmin	pwd	1	16	10/19/93	18:06:57
sysadmin	pwd	1	5	10/19/93	18:08:27
intel0	cmd11	64	165	10/19/93	18:11:36
user2	cmd2	1	19	10/19/93	18:11:59
user2	cmd2	1	11	10/19/93	18:12:28
user2	nsh	0	10	10/19/93	18:16:23
user2	cmd1	32	2482	10/19/93	18:16:37

Using Traces

- In simulations, traces can be used directly to generate the input workload
  - Jobs arrive according to timestamps in the trace
  - Each job requires the number of processors and runtime specified in the trace
- Used to evaluate new schedulers
- Can also be used as data for workload models

Outline

- Background: backfilling
- Conflicting performance results
- Explanation of results
- Accuracy of user runtime estimates
- Effect of inaccurate estimates

Outline

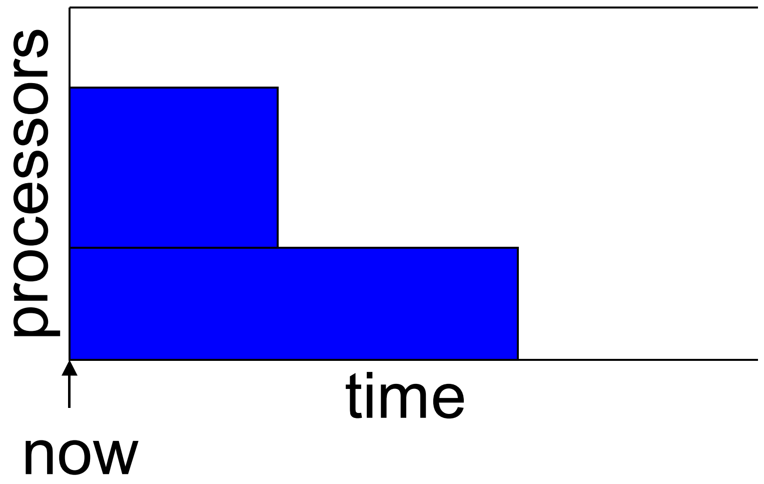
- Background: backfilling
- Conflicting performance results
- Explanation of results
- Accuracy of user runtime estimates
- Effect of inaccurate estimates

EASY Backfilling

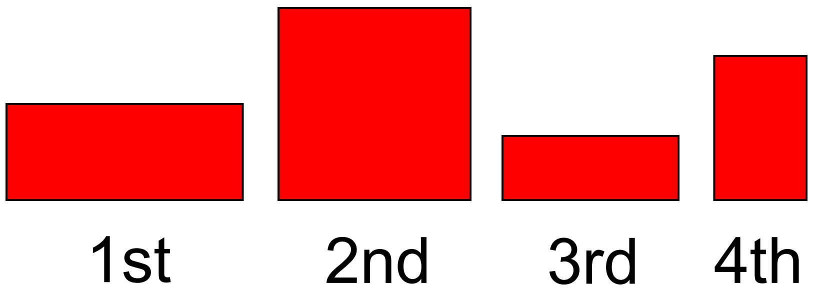
- Early parallel machines used FCFS scheduling
  - Large jobs need to wait for many processors to become available
  - Leads to low utilization
- In 1993 Argonne Natl. Lab. receives IBM SP machine with 128 nodes
  - Develop extensible Argonne scheduling system
  - Uses backfilling to fill in holes in the schedule

EASY Inputs

- List of running jobs
  - Number of processors they use
  - Expected termination

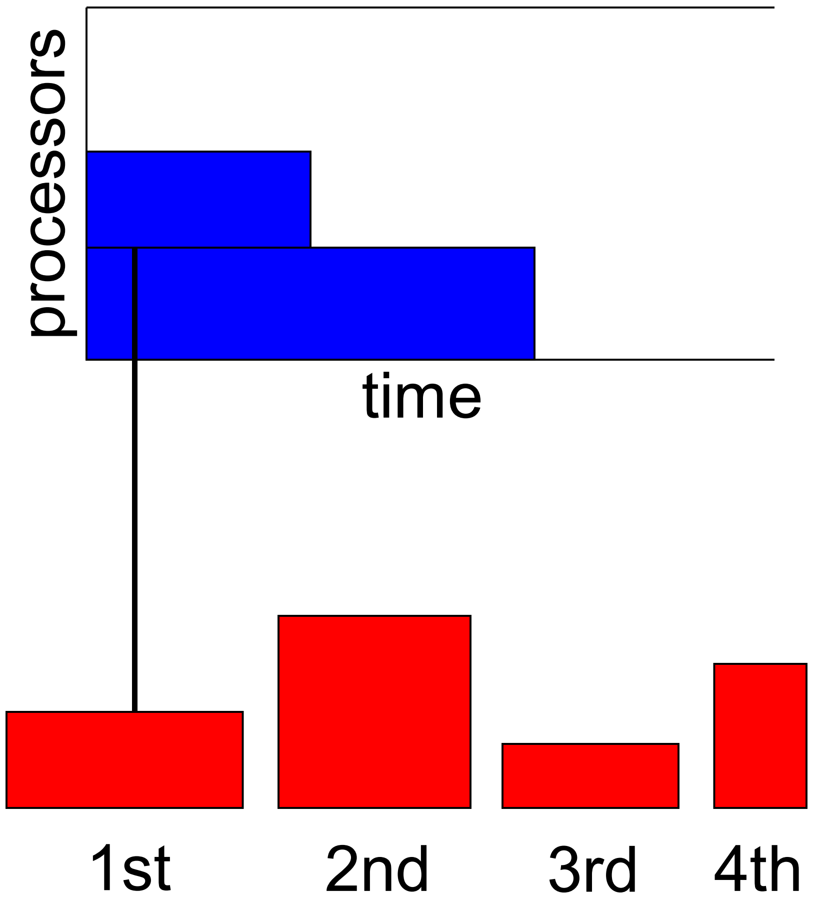


- List of queued jobs
  - How many processors they need
  - How long they are expected to run
  - Sorted in order of arrival



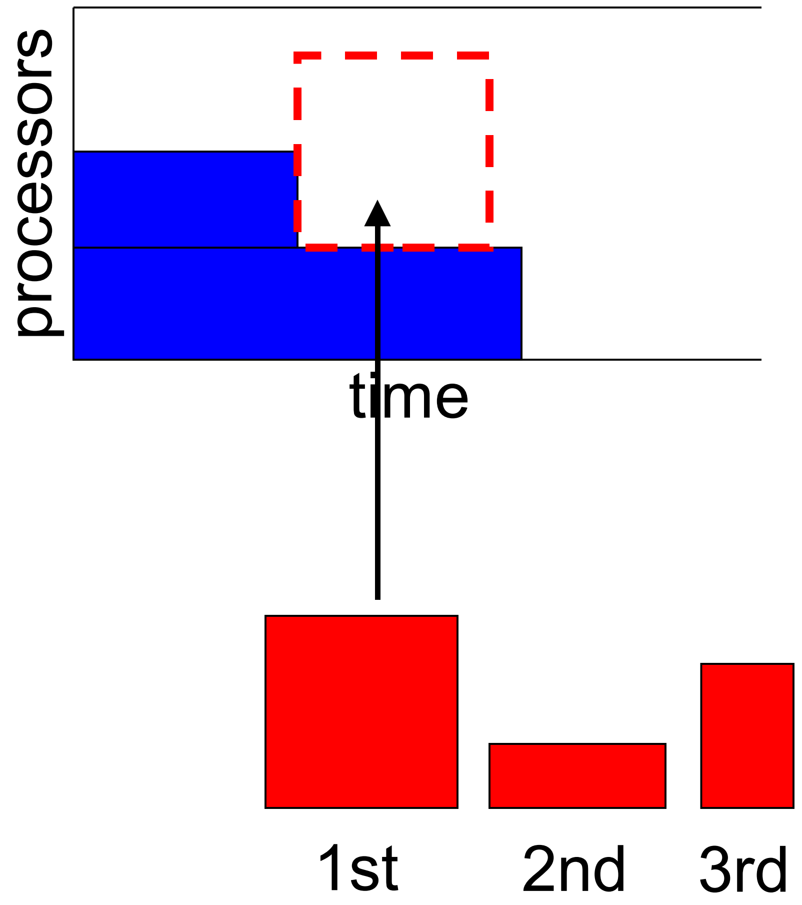
EASY Operation

- Schedule jobs on available processors in FCFS order
- Make reservation for first job that cannot run
- Schedule additional jobs provided they do not conflict with this reservation



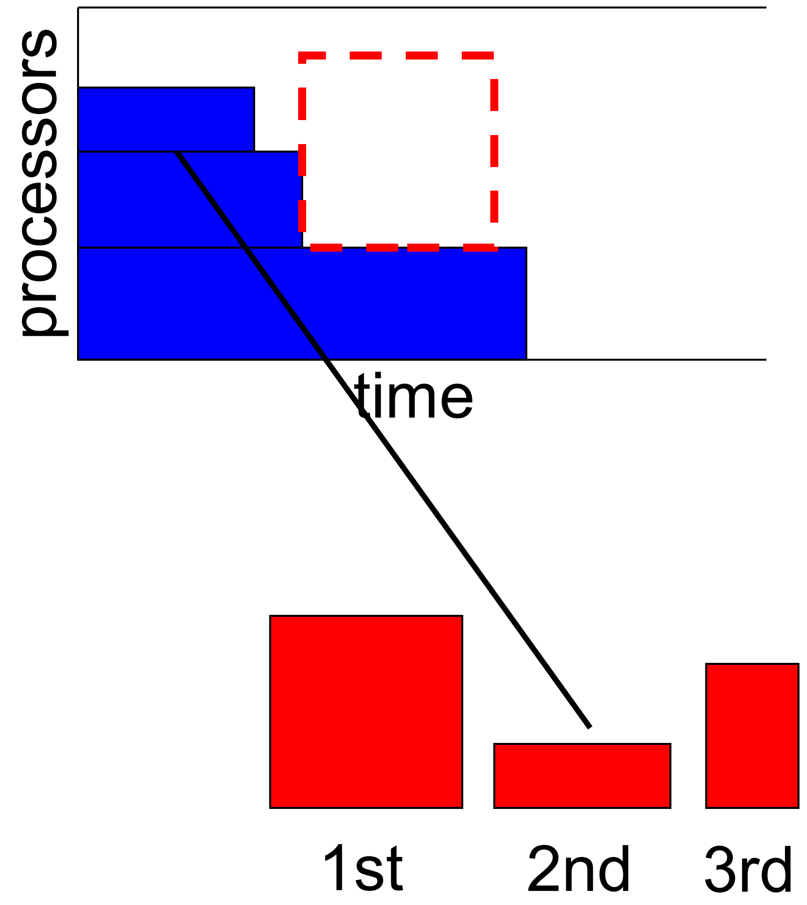
EASY Operation

- Schedule jobs on available processors in FCFS order
- Make reservation for first job that cannot run
- Schedule additional jobs provided they do not conflict with this reservation



EASY Operation

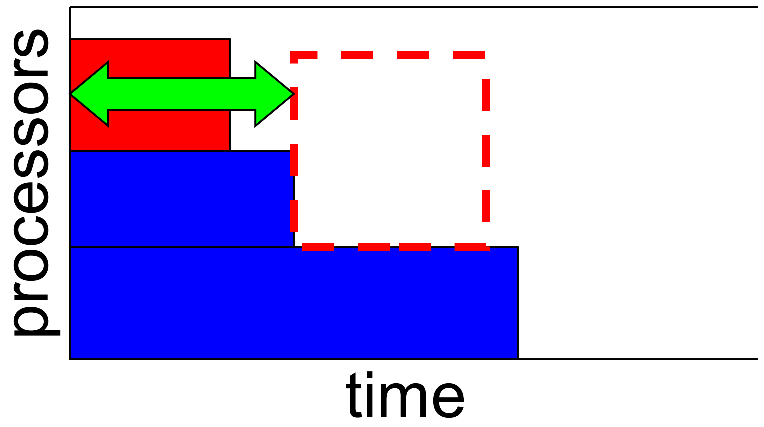
- Schedule jobs on available processors in FCFS order
- Make reservation for first job that cannot run
- Schedule additional jobs provided they do not conflict with this reservation



This is called "backfilling"

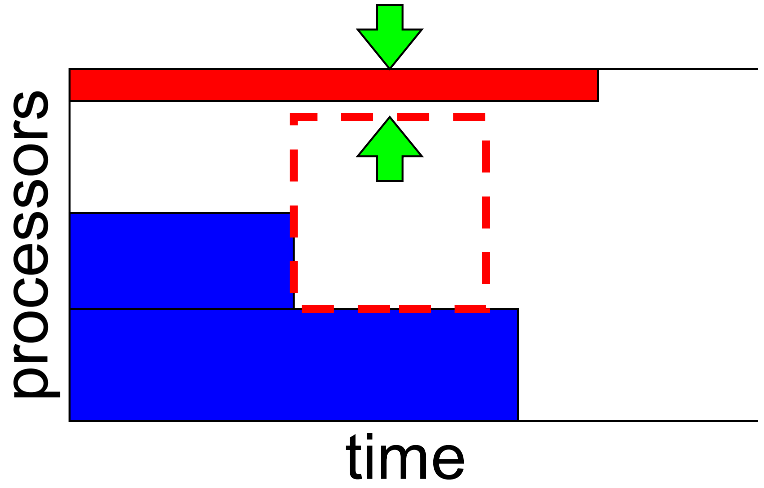
Backfilling Conditions

1. Backfill job will terminate before reservation time



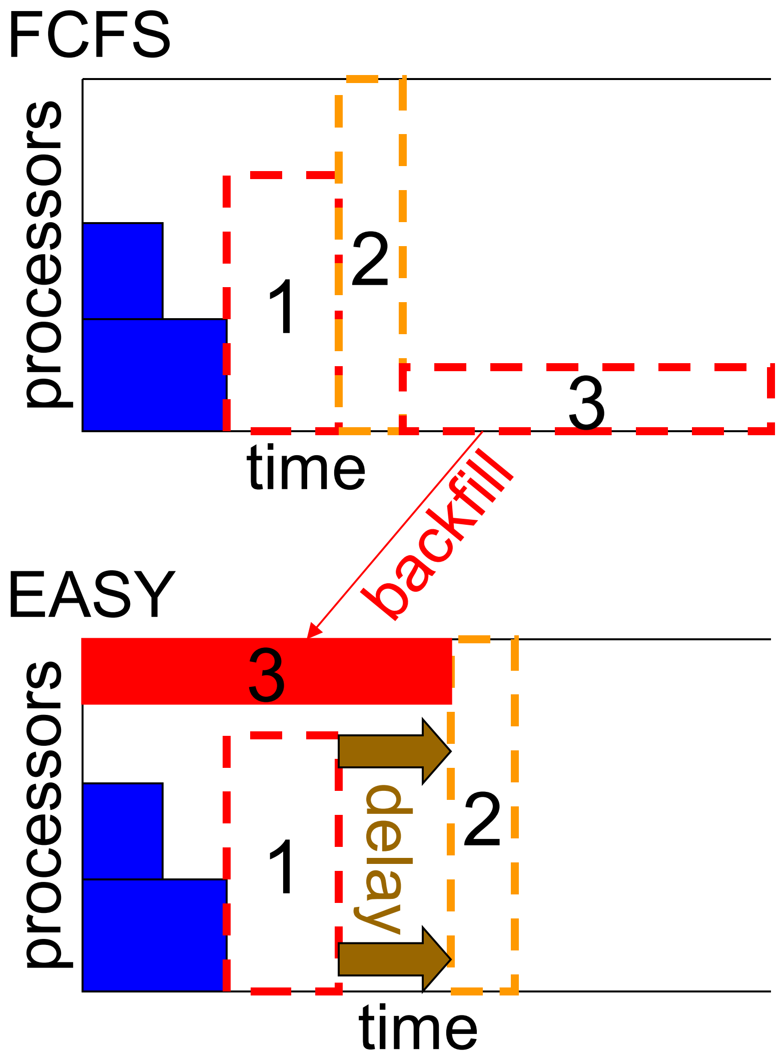
OR

5. Backfill job uses only “extra” processors



Delay

- EASY only makes 1 reservation
- Other jobs may be delayed without bound
- But there is no starvation
  - Each job eventually becomes first in queue

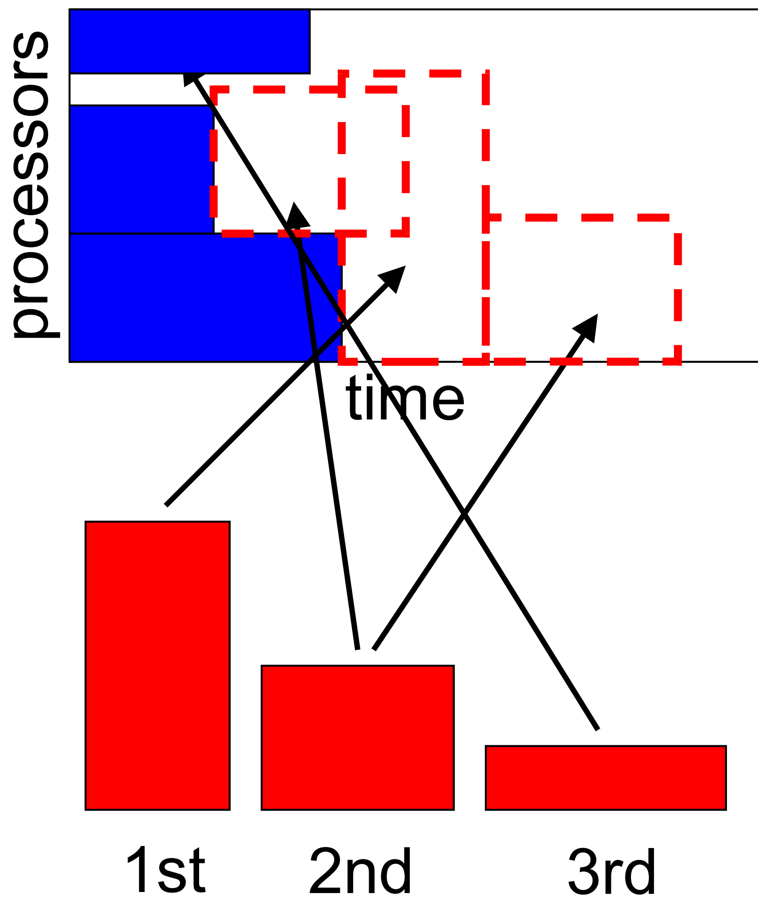


Conservative Backfilling

- Reservations for all jobs
  - Provide guaranteed start times for all jobs
  - Later jobs cannot delay earlier jobs
- Maintain profile of planned schedule
- New jobs need to fit in the profile

Conservative Operation

1. Make reservation for each job that cannot start now
2. Avoid conflict with previous reservations
3. Backfill jobs that can start and have no conflicts



Outline

- Background: backfilling
- Conflicting performance results
- Explanation of results
- Accuracy of user runtime estimates
- Effect of inaccurate estimates

Alternatives

- EASY uses aggressive backfilling
  - Backfilled jobs run earlier
  - But may cause delays for other jobs
- Conservative backfilling is an alternative
  - Less benefits from backfilling
  - But avoid delays for other jobs

Which approach is better?

Simulations

- Compare EASY and conservative
- Use different workload traces
  - CTC
  - SDSC
- Also use workload models
  - Jann (based on CTC)
  - Feitelson
- And different metrics of performance
  - Response time
  - Bounded slowdown

Workloads

- CTC: IBM SP at Cornell
  - 430 node machine
  - June 1996 to May 1997
  - 79,000 jobs
- SDSC: IBM SP at San Diego
  - 128 node machine
  - April 1998 to April 2000
  - 73,500 jobs

Models

- Jann
  - Based on CTC data
  - Log-uniform distribution of sizes
  - Hyper-Erlang distribution of runtimes
- Feitelson
  - Based on several traces
  - Modal distribution of sizes
  - Hyper-exponential distribution of runtimes
  - Correlation between them

Metrics

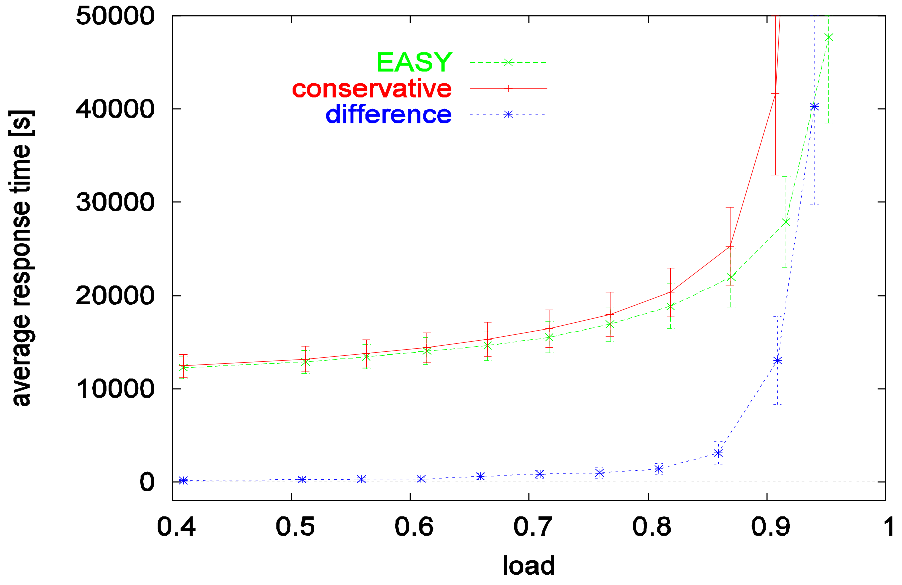
- Response time: from arrival to termination
- Bounded slowdown:

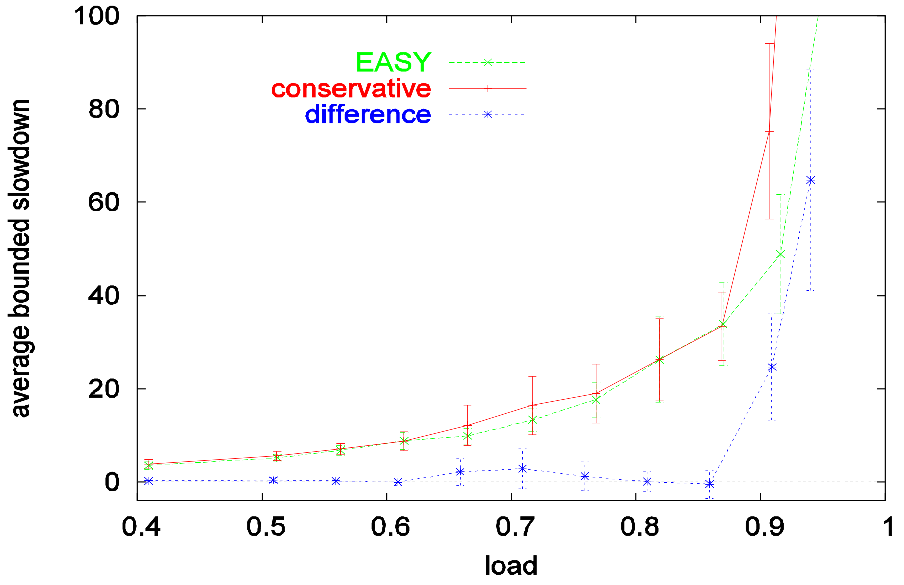
$$b_{sld} = \begin{cases} \dfrac{w+r}{r} & r > 10 \\ \dfrac{w+r}{10} & r < 10 \end{cases}$$

Simulation Results

- Results depend on interaction of
  - Metrics
  - Workloads
  - Schedulers
- Specifically, comparing EASY and conservative backfilling with CTC and Jann workloads using response time and slowdown metrics produces conflicting results
- Recall that Jann model is based on CTC data…

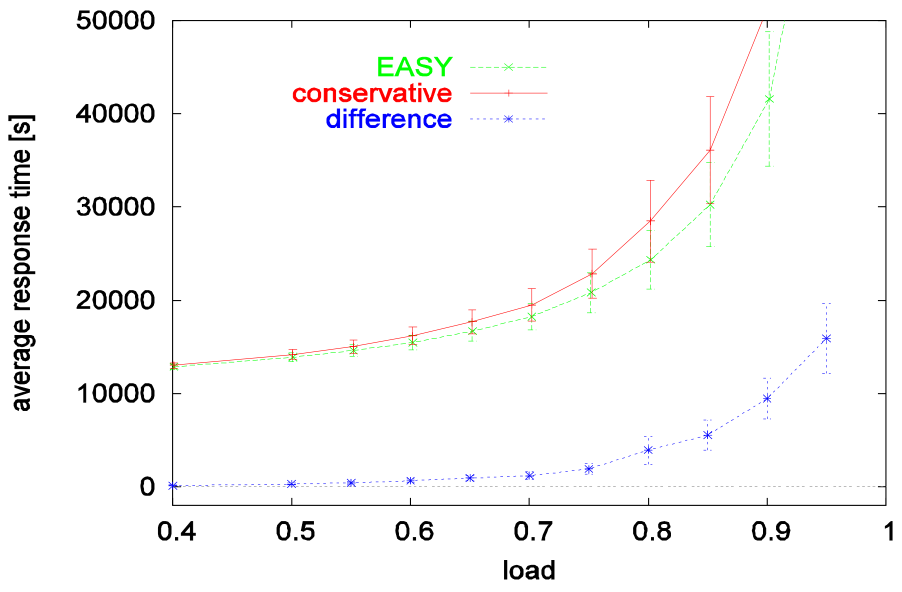
Results CTC

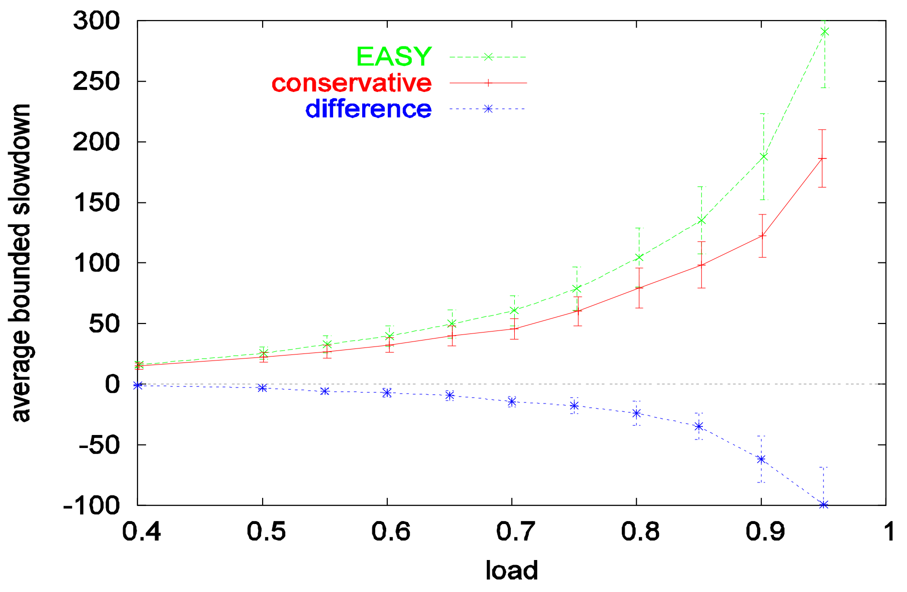




- EASY better for response time
- Largely the same for slowdown

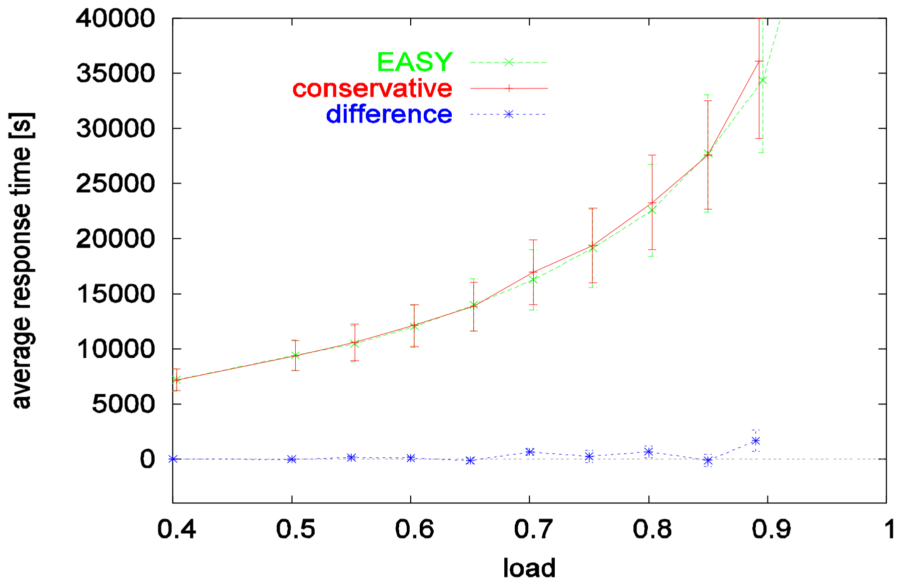
Results Jann

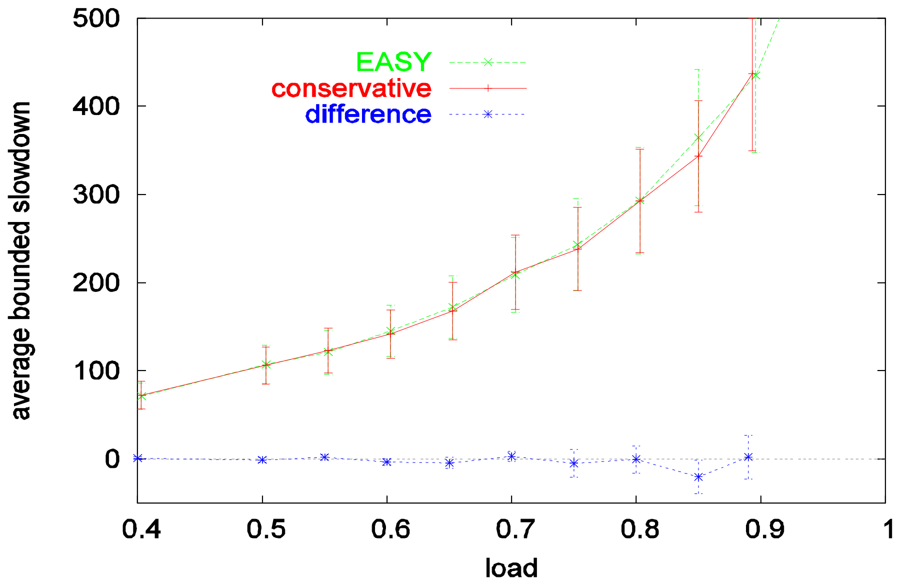




- EASY better for response time
- Conservative better for slowdown

Results Feitelson





- Largely the same for both metrics
- Similar results for SDSC

Job Classes

- Backfilling depends on job characteristics
  - Size
  - (Expected) length
- Slowdown is sensitive to short jobs
- So let’s look at different job classes independently
  - Short jobs <= 1 hour
  - Long jobs > 1 hour

Jann vs. CTC

Work-load	Job class	Response time		Slowdown	
		EASY	cons	EASY	cons
Jann	Short	8015	6404	143	109
	Long	52655	65173	1.85	2.32
	All	23901	27313	92.8	71.1
CTC	Short	3785	4632	22.8	45.4
	Long	33867	37763	1.47	1.65
	All	15465	17497	14.5	28.4

Jann vs. CTC

Work-load	Job class	Response time		Slowdown	
		EASY	cons	EASY	cons
Jann	Short	8015	6404	143	109
	Long	52655	65173	1.85	2.32
	All	23901	27313	92.8	71.1
CTC	Short	3785	4632	22.8	45.4
	Long	33867	37763	1.47	1.65
	All	15465	17497	14.5	28.4

Explanation

- With CTC all job classes behave consistently, and EASY is better
- With Jann job classes are different:
  - Conservative is better for short jobs
  - EASY is better for long jobs
- Overall average of slowdowns dominated by short jobs -> favors conservative
- Overall average of response times dominated by long jobs -> favors EASY

But...

- We found the *mechanism* that produces conflicting results from consistent data
- But why the difference between short and long jobs with Jann?
- Look at details of workload

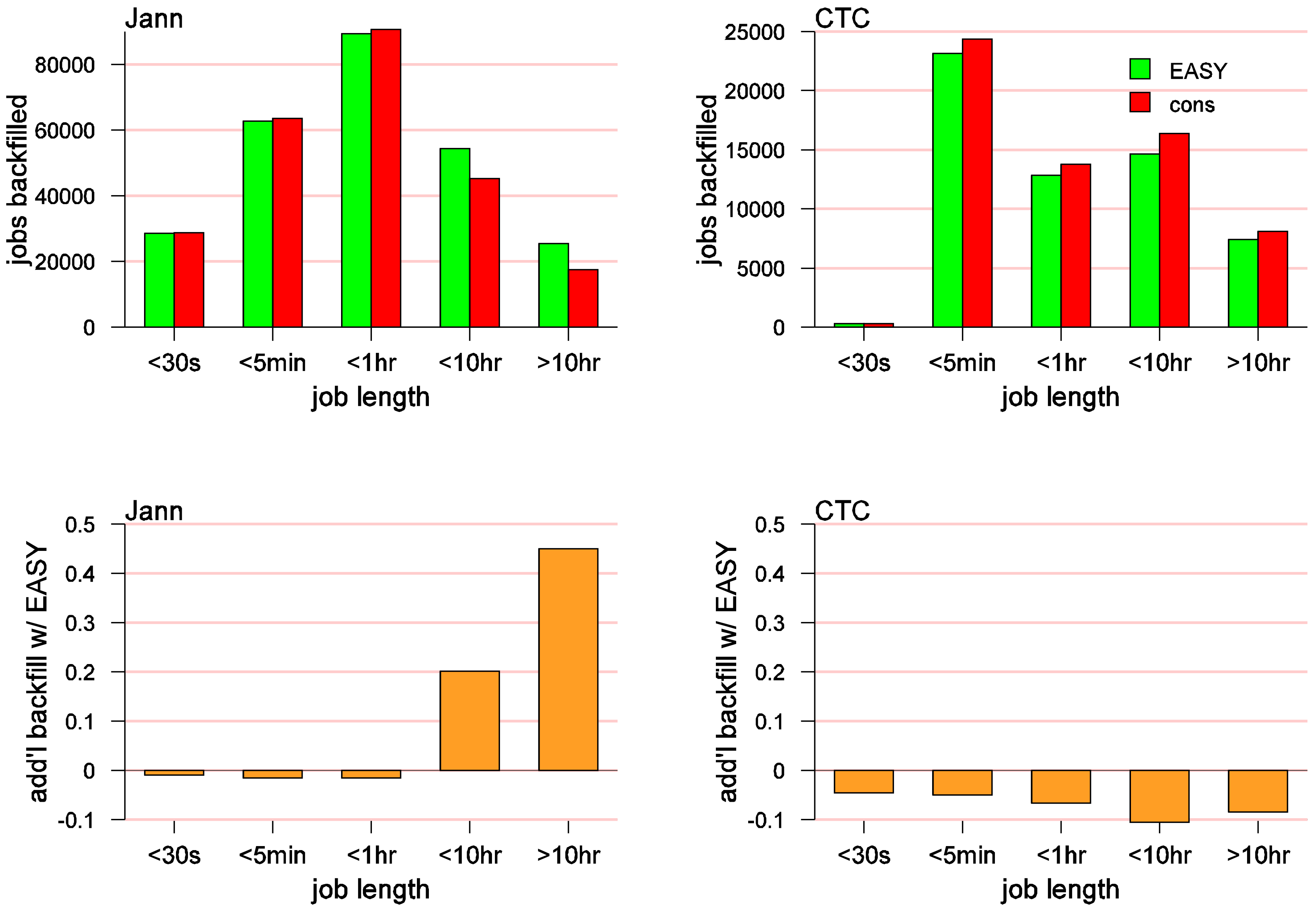
Workload Details

- Jann has very short jobs that suffer very high slowdowns
- Jann has very long jobs that cause longer delays to other jobs
- Job sizes in Jann are not powers of 2, so do not pack so well
- System size for Jann is not power of 2

(BTW, most long jobs are serial)

Root Causes

- All these hypotheses were checked by controlled modifications to the workload
  - E.g., remove all very short/long jobs
- Cause of results is <u>not</u> differences in the workload
- So look for clues in backfilling behavior


Jann
jobs backfilled
80000
60000
40000
20000
0
<30s
<5min
<1hr
<10hr
>10hr
job length
CTC
25000
20000
15000
10000
5000
EASY
cons
add'l backfill w/ EASY
0.5
0.4
0.3
0.2
0.1
-0.1

Results and Questions

- Under Jann, EASY does more backfilling of long jobs
- Conservative does less backfill of long jobs
- Why does this happen?
- How does this lead to better performance of short jobs?

Outline

- Background: backfilling
- Conflicting performance results
- Explanation of results
- **Accuracy of user runtime estimates**
- Effect of inaccurate estimates

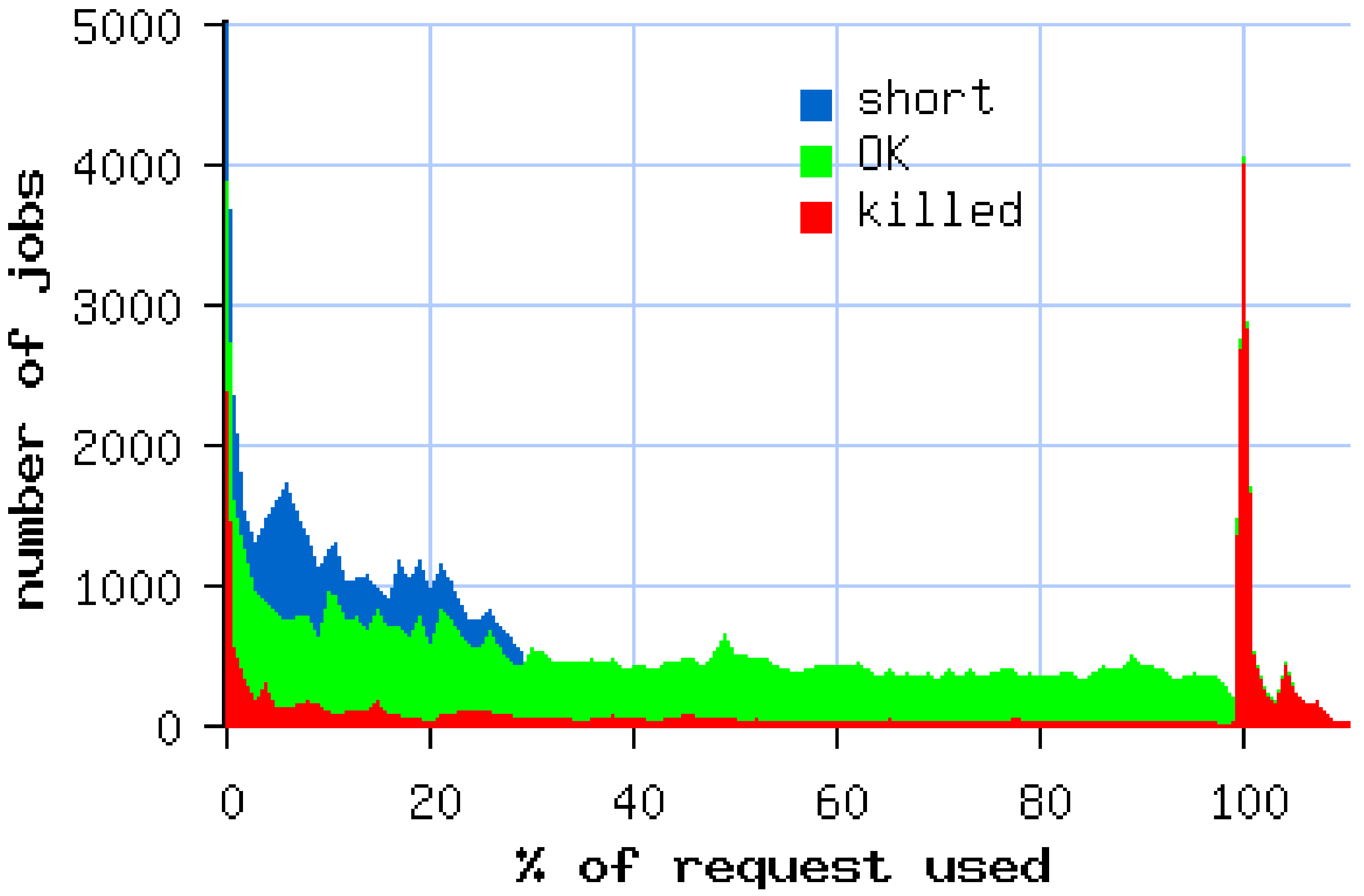
Runtime Estimates

- When users submit jobs, they provide
  - The number of processors to use
  - An estimate of the job runtime
- Estimates are used to predict when processors will become free for reservation
- Also used to verify that backfill job will terminate before reservation
- If it does not, it will be killed

The Theory

- If runtime estimate is low, job has a better chance to backfill
- If it is too low, job will be killed
- So users are motivated to provide accurate estimates

The Reality



short
OK
killed
number of jobs
5000
4000
3000
2000
1000
0
0
20
40
60
80
100
% of request used

Explanation of Results

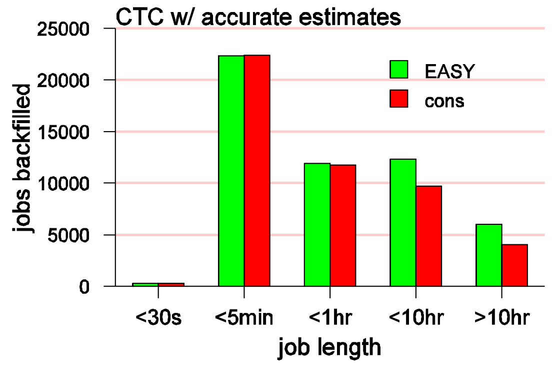
- The Jann model assumes accurate runtime estimates
  - Leaves few holes in the schedule
  - Harder to backfill long serial jobs
  - EASY backfills anyway and delays later short jobs; slowdown is sensitive to such delays
  - Conservative cannot delay and therefore does not backfill; this leads to lower slowdowns

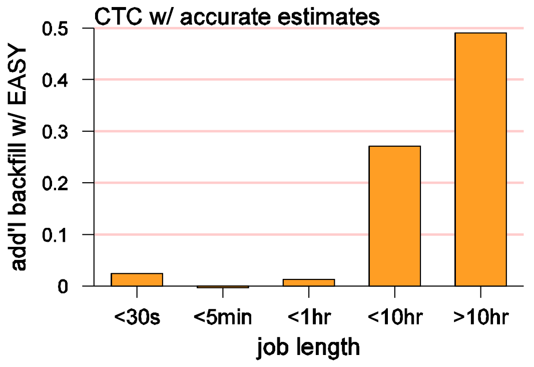
Explanation of Results

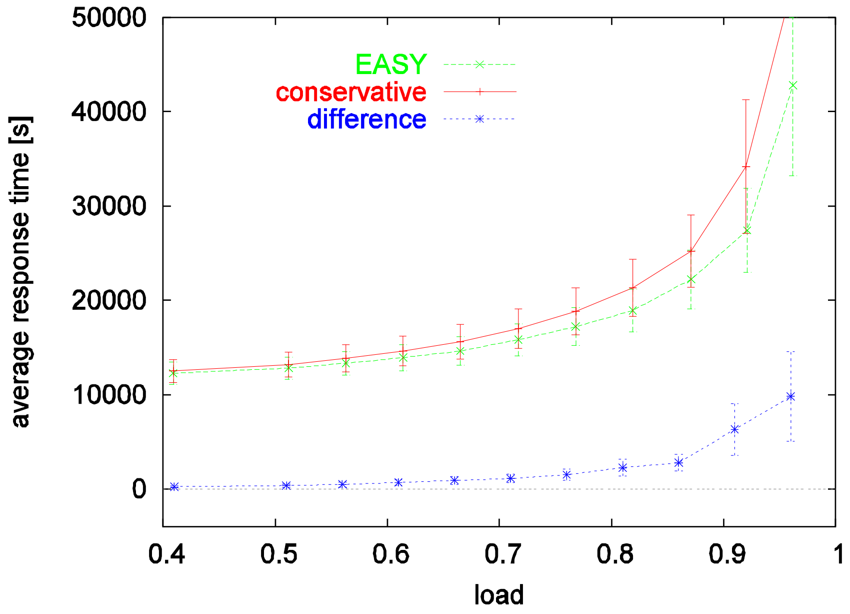
- In CTC estimates are grossly inaccurate
  - This leads to holes in the schedule and additional backfilling opportunities
  - Both EASY and conservative achieve similar backfilling

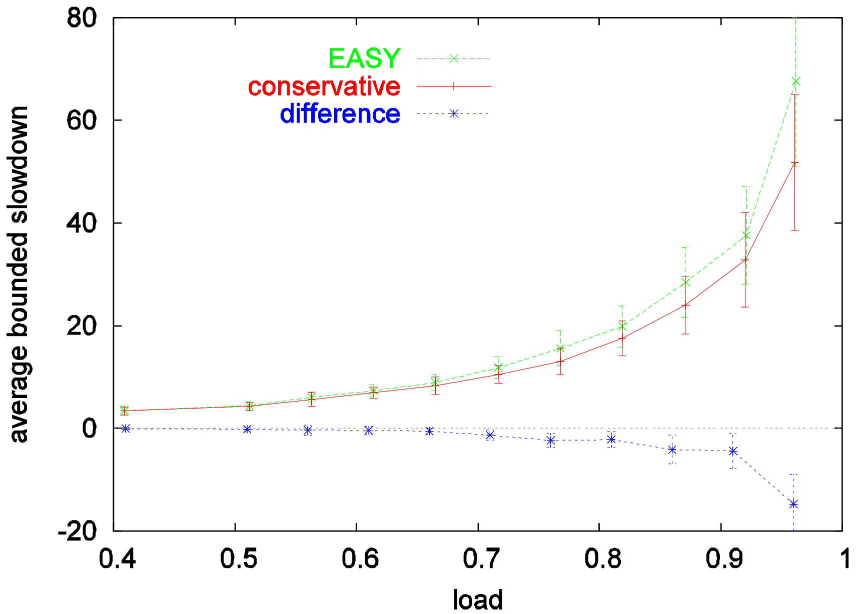
Verification

- Re-run CTC simulations using accurate runtime estimates
- Leads to results like Jann









Summary

A triple interaction:

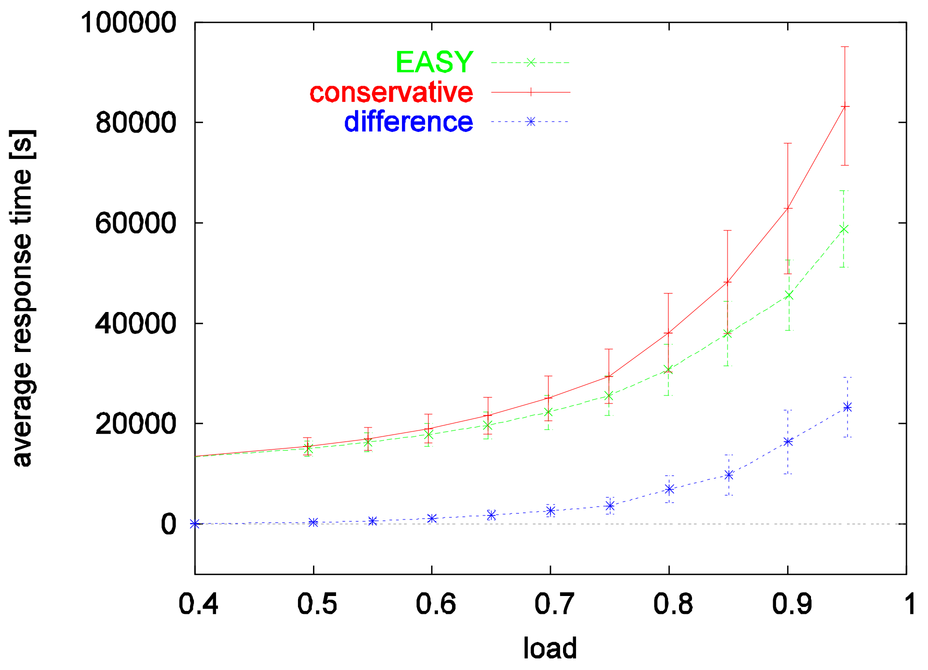
- The CTC workload includes user estimates of runtime. The Jann model does not.
- Using accurate estimates in Jann causes conservative to achieve less backfilling of long serial jobs. This is good for short jobs that do not get delayed by them.
- Response time is dominated by long jobs, so favors EASY. Slowdown is dominated by short jobs, so favors conservative.

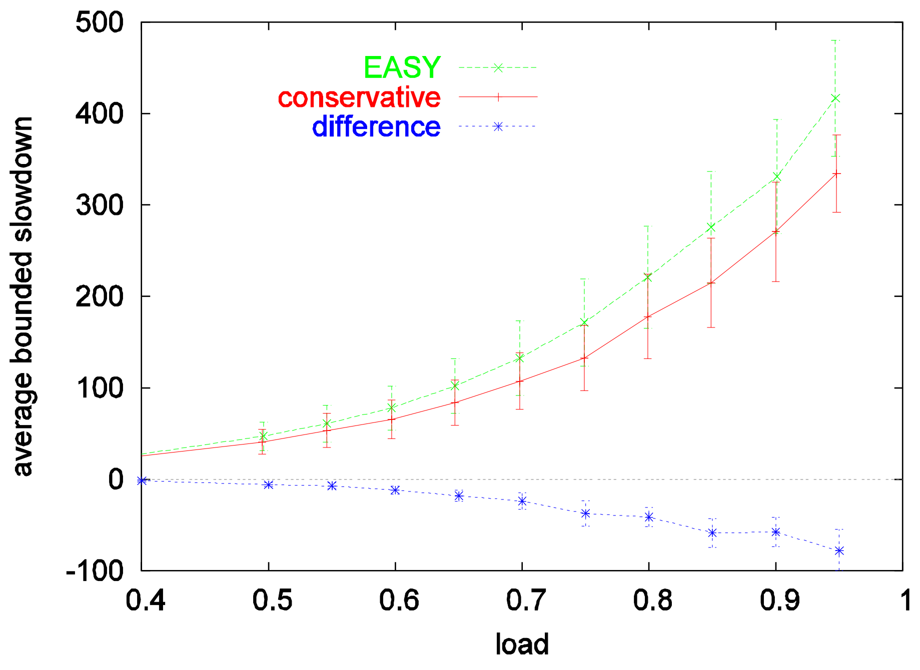
What about Feitelson?

- Jann is specifically modelled after CTC
- Shares the same distributions, especially the long single-node jobs
- Feitelson does not have many such jobs
- So the whole interaction does not occur

Verification

- Modify Feitelson model to create many single-node long jobs
- Also use non-power-of-two nodes





Summary of All Results

- No long serial jobs:

  => EASY and conservative are similar
- Have long serial jobs (as in CTC):

  => EASY and conservative are different
  - User runtime estimates inaccurate:

    => EASY better
  - User runtime estimates accurate:

    => EASY better for response time

    => conservative better for slowdown
  - difference grows when machine size not power of 2

Conclusions

- Workloads and metrics may play a larger role than expected
- Interactions can be complicated
- Seemingly benign assumptions can be crucial

Outline

- Background: backfilling
- Conflicting performance results
- Explanation of results
- Accuracy of user runtime estimates
- Effect of inaccurate estimates

The Questions

What is the effect of inaccurate user runtime estimates?

Can we generate more accurate predictions?

Will it improve performance?

The Experiments

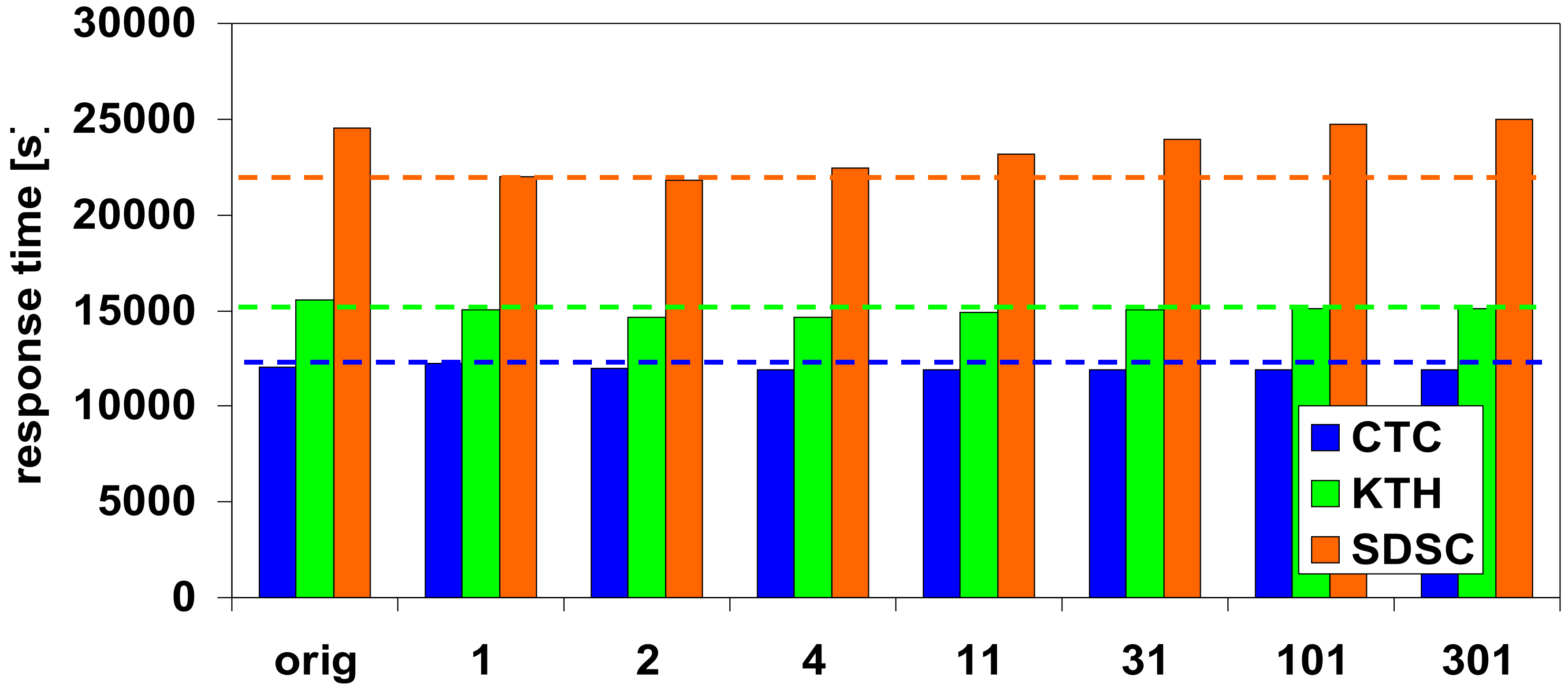
- Replace user estimates with synthetic estimates with controlled accuracy
  - For a given inaccuracy factor $f \geq 1$,

    and a job with real runtime r,

    generate an estimate in the range $[\, r, f{*}r \,]$
- Run the simulations again to assess effect

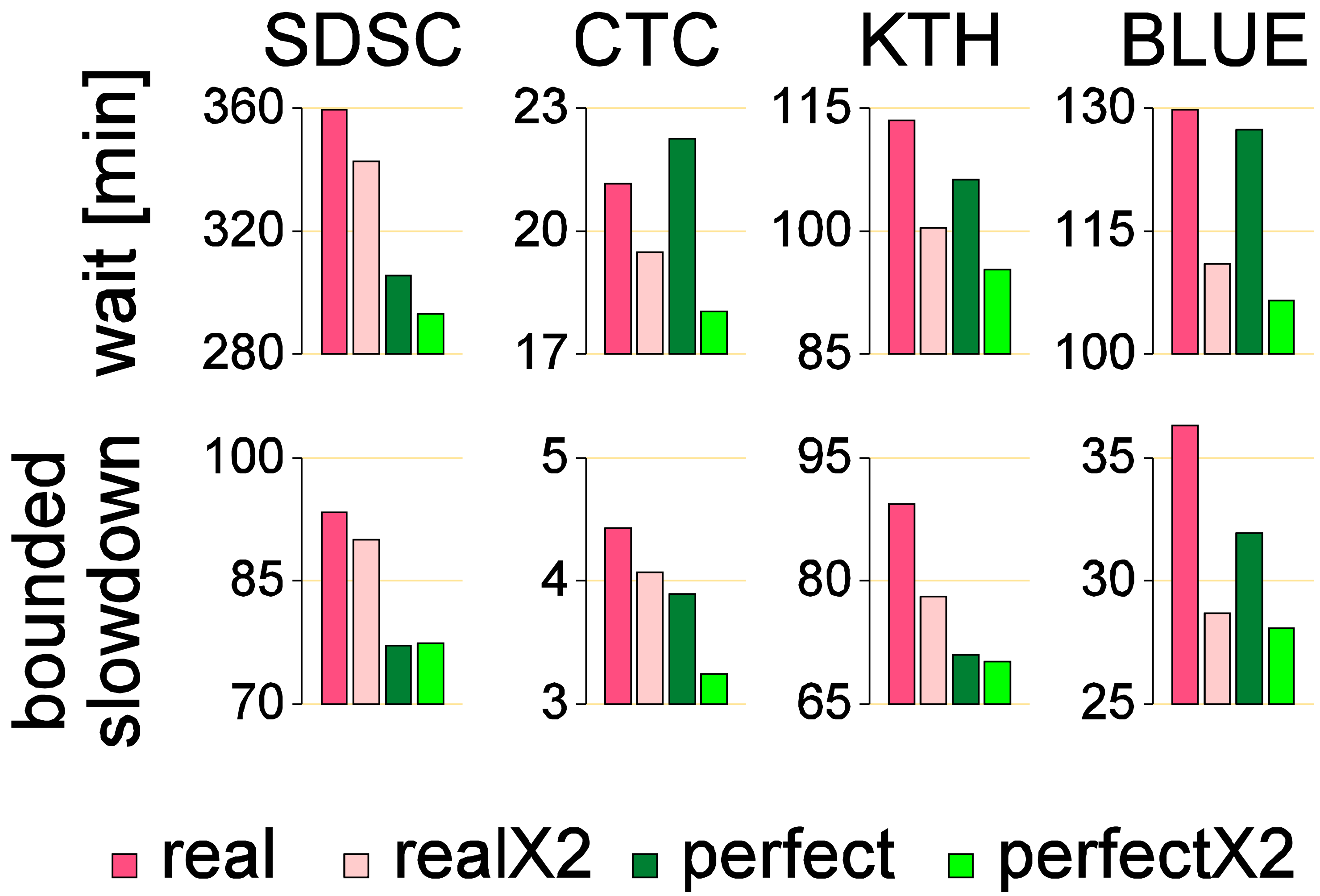
The Surprise

- Inaccurate runtime estimates lead to improved performance
- Also better than the original user estimates



Exploiting this Result

- User estimates are inaccurate
- Inaccurate estimates lead to better performance
- So why not make user estimates even less accurate?
- Idea: multiply user estimates by 2

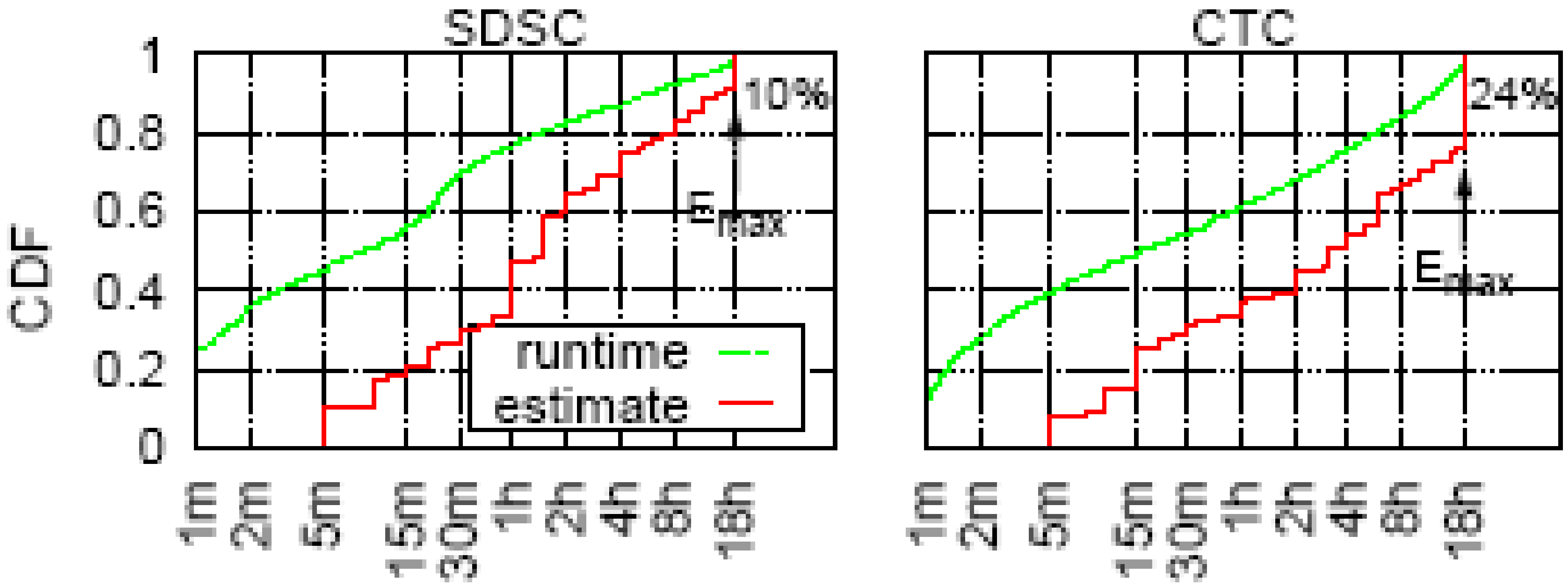


Outcome: indeed improves performance…

But why does it work?

A Defect in the Model

- Synthetic estimates based on the real runtime convey considerable information about the real runtime
  - Even with *f=10*, a short job will have a short estimate
  - And a long job will have a long estimate
- Real estimates convey much less information
  - Many simply use the maximal allowed value
  - They are rounded to the nearest 5/15/30/60 min, so jobs that are actually different become indistinguishable



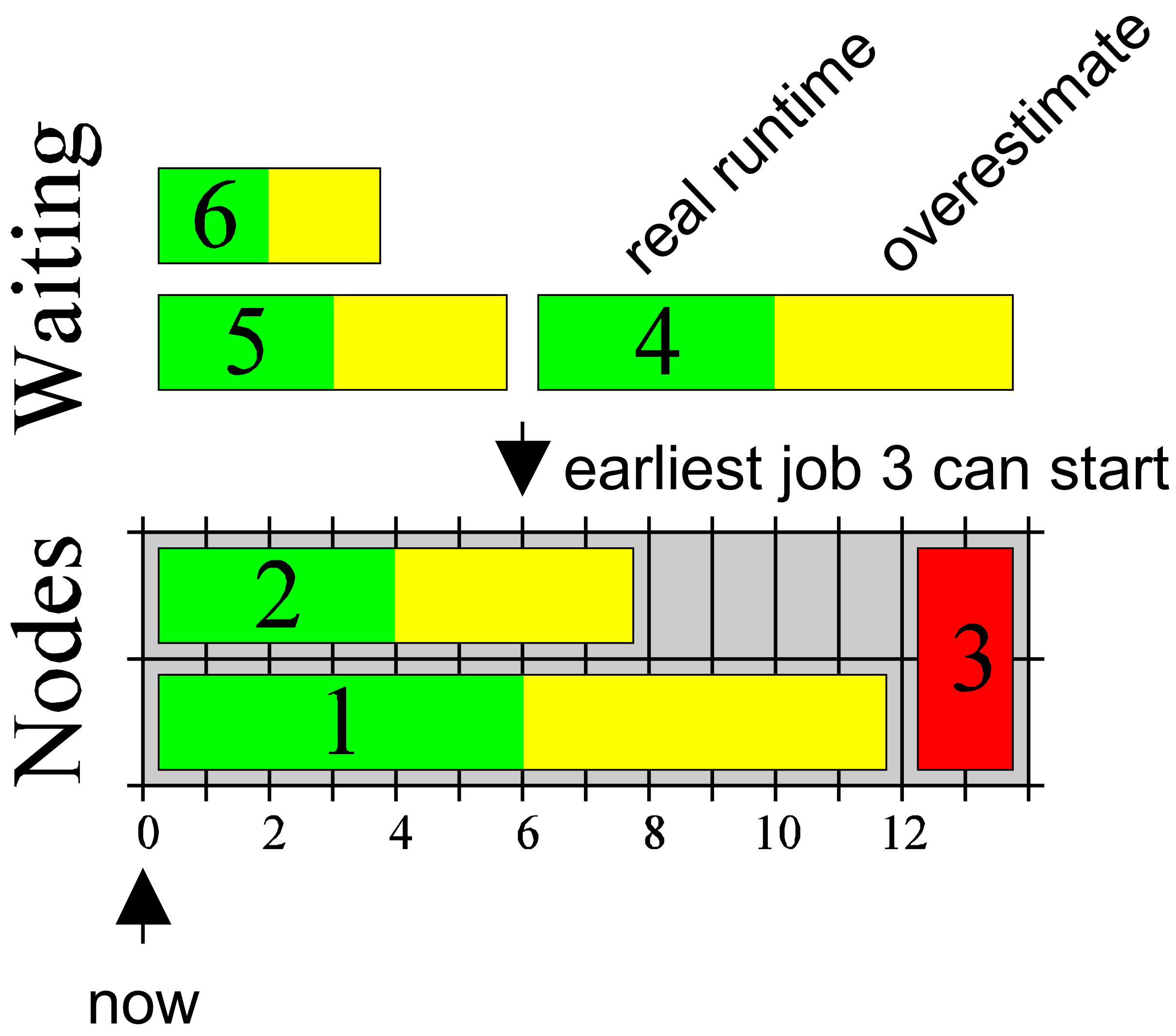
SDSC
CTC
CDF
1
0.8
0.6
0.4
0.2
0
1m
2m
5m
15m
30m
1h
2h
4h
8h
18h
10%
24%
E_{max}
runtime
estimate

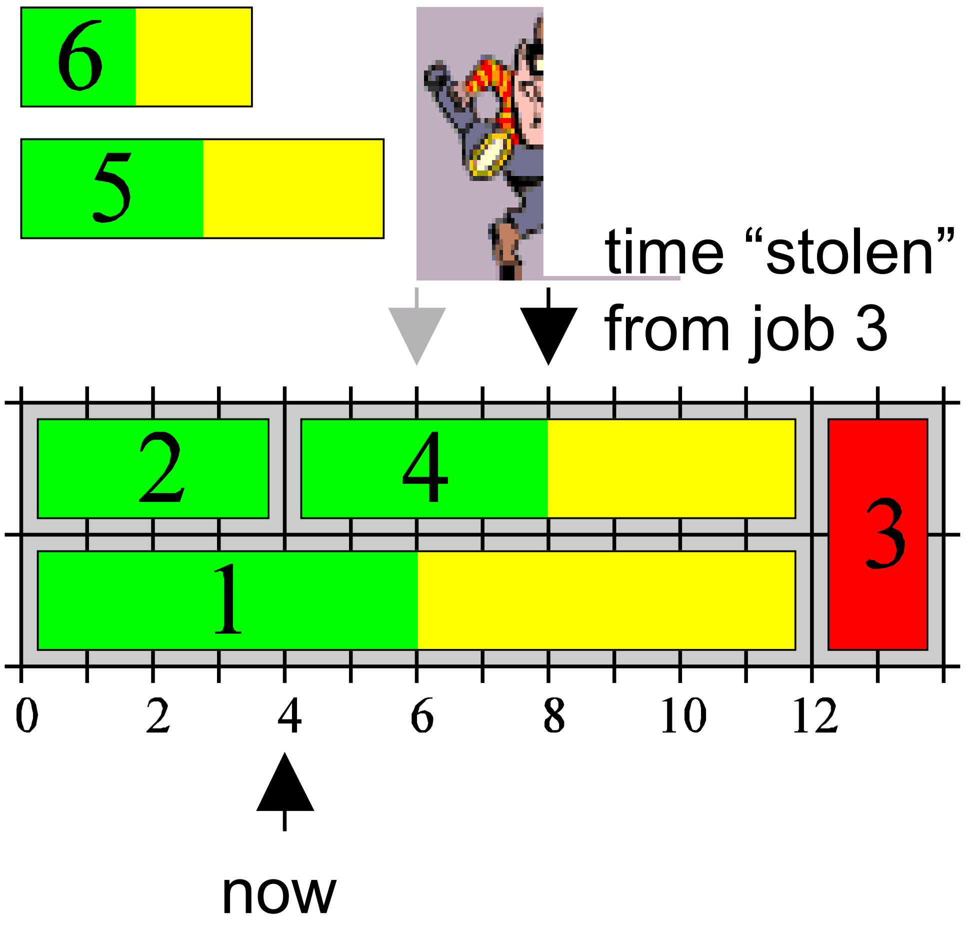
A Better Model

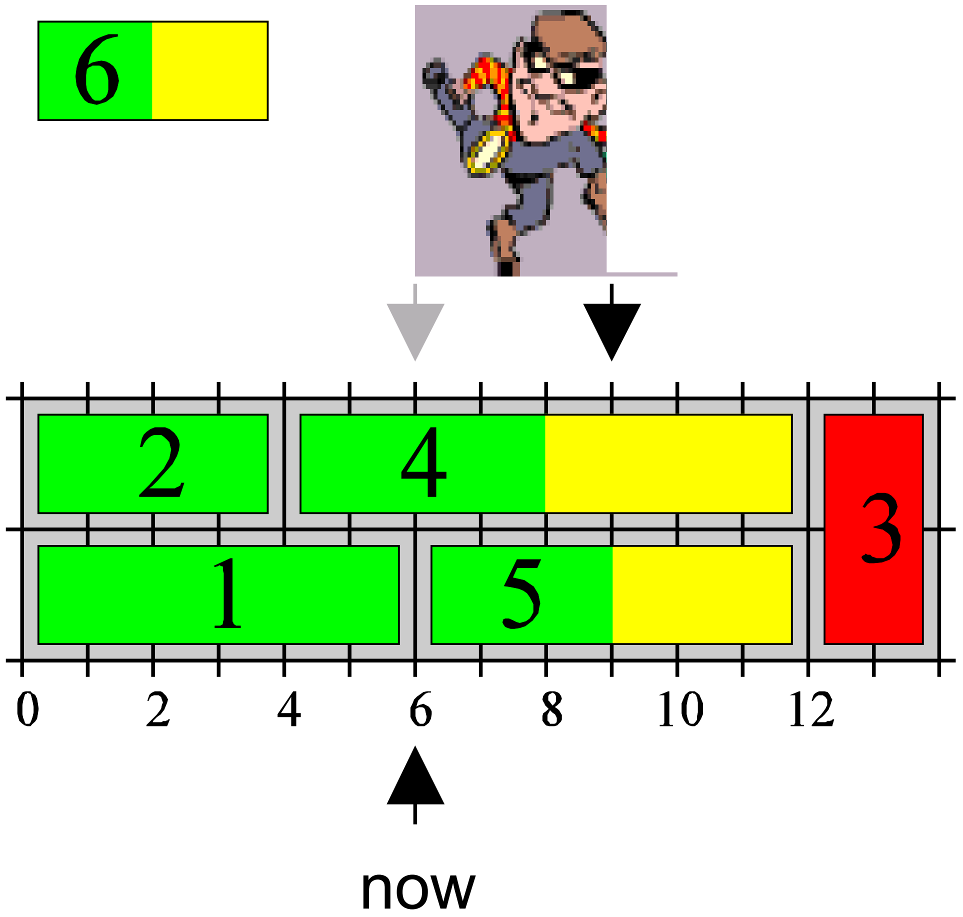
- Increased inaccuracy does *not* mean multiplying by a larger factor
- Instead, it means more modal estimates
  - More jobs use the maximal estimate value
  - More jobs use estimates like 30 min or 2 hr
- This provides less information
- And indeed leads to degraded performance

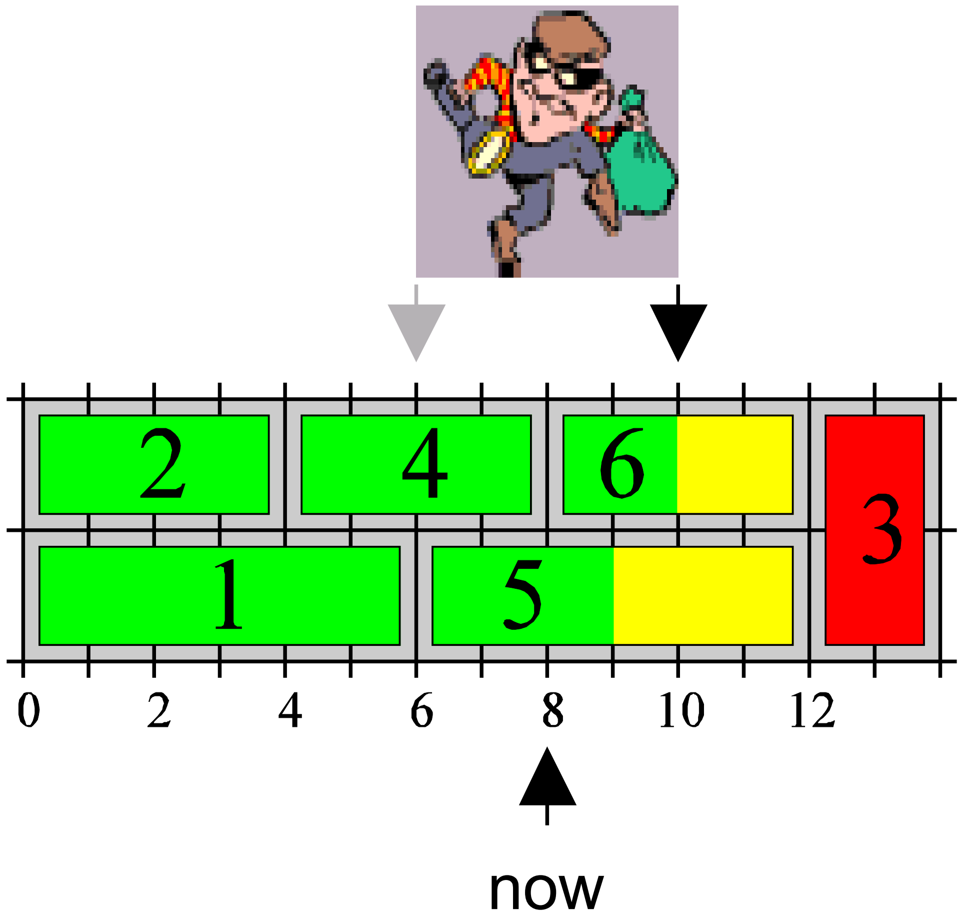
The Dynamics of Backfilling

- With inaccurate estimates jobs will terminate much earlier than expected
- Thus there will often be holes in the schedule
- These holes can be used for backfilling
- As we near the reservation for the first queued job, the holes will become shorter
- This leads to preferential backfilling of short jobs, or an SJF-like schedule
- Which leads to better performance


Waiting
6
5
4
real runtime
overestimate
earliest job 3 can start
Nodes
2
1
3
0
2
4
6
8
10
12
now


6
5
time “stolen” from job 3
2
4
3
1
0
2
4
6
8
10
12
now


6
2
4
3
1
5
0
2
4
6
8
10
12
now


2
4
6
1
5
3
0
2
4
6
8
10
12
now

A Better Scheduler

- Doubling user estimates is a scheduler policy that helps because it implicitly leads to an SJF-like schedule
- This implicitly trades off fairness for performance
- It would be better to do this explicitly
  - Explicit SJF to reduce average response time
  - Acknowledge the effect on fairness, and decide whether it is worth it
- Compromise: backfill short jobs first

Results

Trace	EASY avg wait time	SJBF avg wait time	Better by
SDSC	363	361	–
CTC	21	19	-10%
KTH	114	102	-11%
BLUE	130	102	-21%

Even better results achieved with historically based runtime predictions

History-Based Predictions

- Users often repeat work
- Successive jobs are similar
- So historical data is useful for predictions
  - E.g. use average of last 2 jobs by same user
- Problem: this could be an underestimate
  - System will kill job
  - Users will be mad

The Solution

- Estimates have two uses:
  1. Convey information about expected resource usage to scheduler
  2. Contract with user: job will be killed after this time
- Predictions based on history replace only the first use
- Also use prediction correction: if prediction is too short, replace with original estimate

Results

Trace	EASY avg wait time	EASY++ avg wait time	Better by
SDSC	363	327	-10%
CTC	21	14	-33%
KTH	114	95	-17%
BLUE	130	87	-33%

Even better results for slowdown