A Low-Cost Rescheduling Policy for Dependent Tasks on Grid Computing Systems

Henan Zhao and *Rizos Sakellariou*

Department of Computer Science

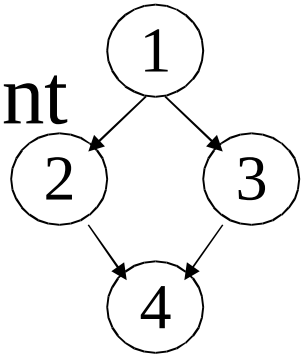
University of Manchester, UK

Aspects of Scheduling on the Grid

- Grid ~ Heterogeneous Distributed System
- Independent jobs (=tasks)
- Static predictions for job execution time
    - Build a schedule
    - What happens at run-time? Rescheduling?
- How to deal with dependent jobs?
    - A job needs to be executed before another
    - Workflows (e.g., Pegasus)
    - Directed Acyclic Graphs (DAGs)

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

DAG Scheduling

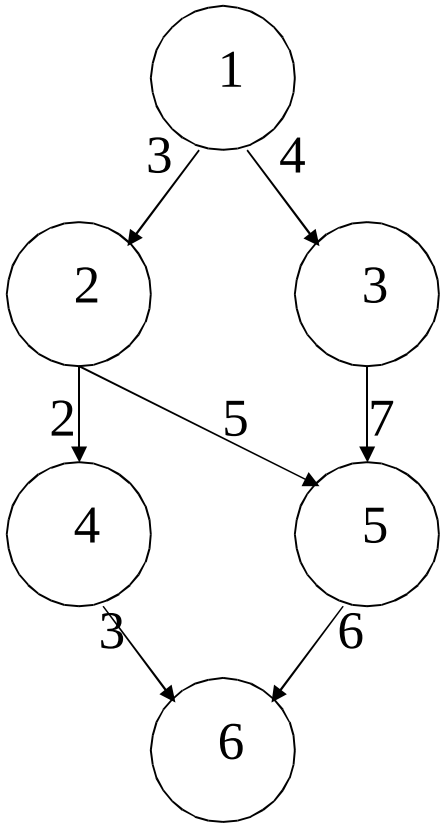


- A directed acyclic graph (DAG) may represent applications with dependent jobs.
- Each node (job) has an execution cost (different on different machines)
- Each edge corresponds to data transfer
- How to assign jobs onto machines, so that:
  - Dependence constraints are respected
  - Overall time (makespan) is minimized.
- One job is executed at a time on a machine.

Many heuristics for DAG scheduling exist - but…!

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

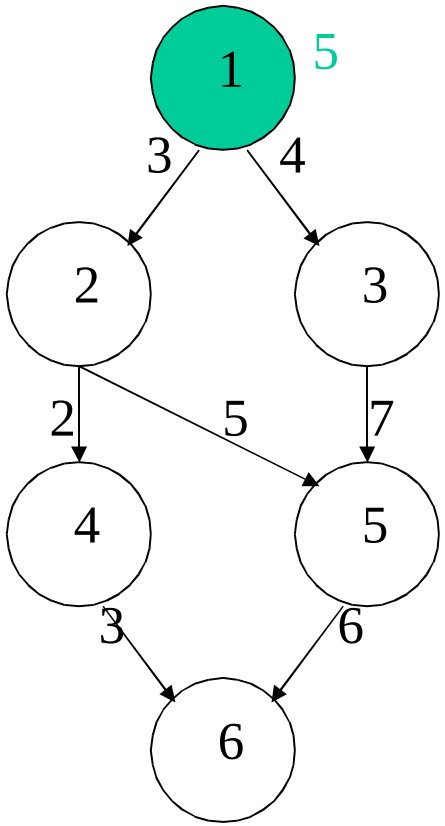
An Example of HEFT



Task	P1	P2	P3
1	3	5	7
2	4	8	3
3	3	4	11
4	5	8	2
5	10	3	5
6	5	8	8

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

An Example of HEFT

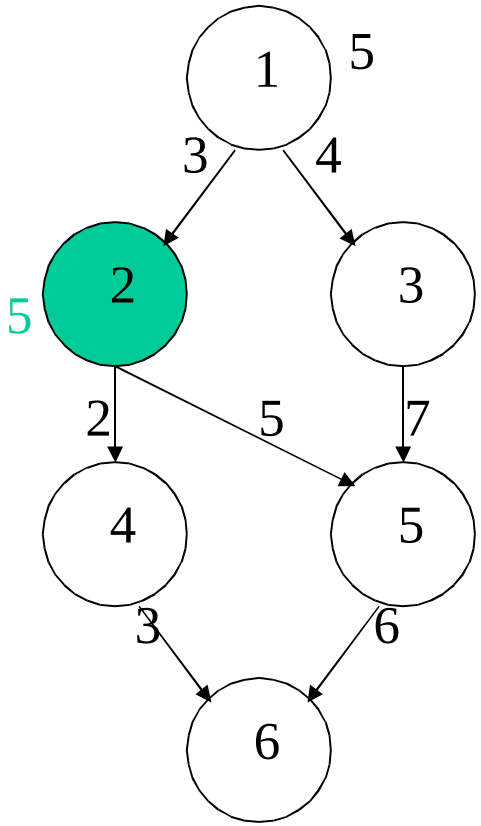


Task	P1	P2	P3
1	3	5	7
2	4	8	3
3	3	4	11
4	5	8	2
5	10	3	5
6	5	8	8

= 5

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

An Example of HEFT



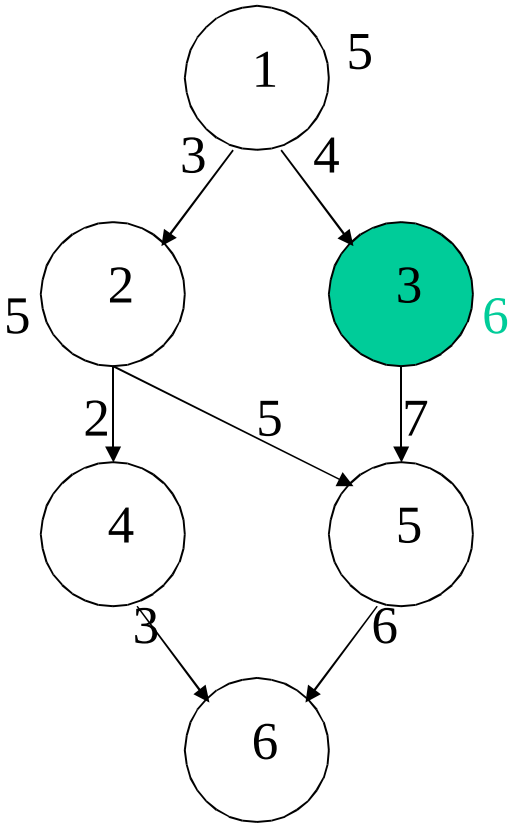
Task	P1	P2	P3
1	3	5	7
2	4	8	3
3	3	4	11
4	5	8	2
5	10	3	5
6	5	8	8

= 5

= 5

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

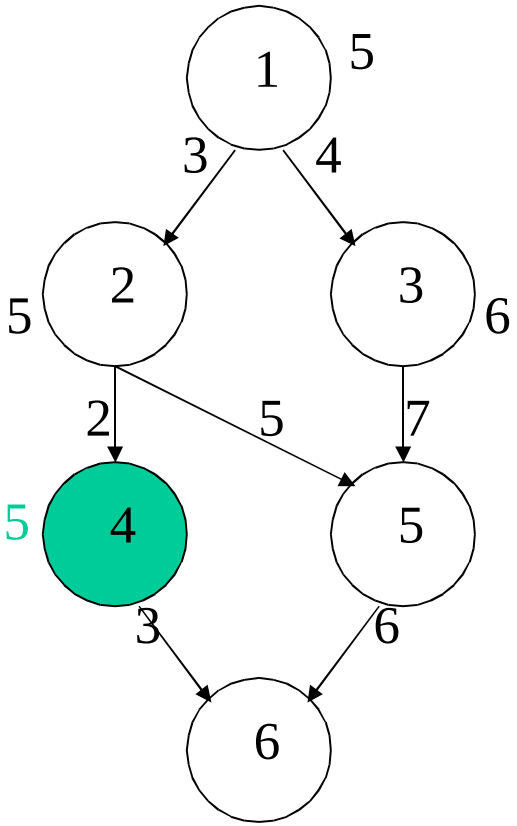
An Example of HEFT



Task	P1	P2	P3	
1	3	5	7	= 5
2	4	8	3	= 5
3	3	4	11	= 6
4	5	8	2	
5	10	3	5	
6	5	8	8	

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

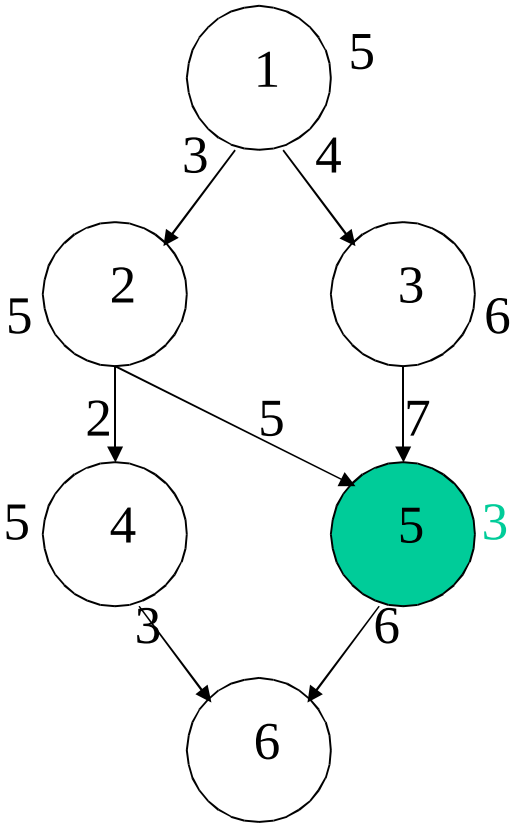
An Example of HEFT



Task	P1	P2	P3	
1	3	5	7	= 5
2	4	8	3	= 5
3	3	4	11	= 6
4	5	8	2	= 5
5	10	3	5	
6	5	8	8	

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

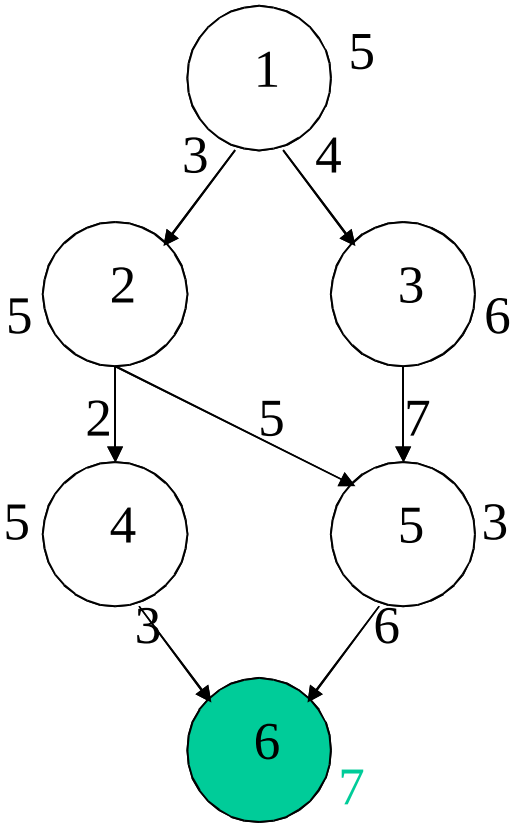
An Example of HEFT



Task	P1	P2	P3	
1	3	5	7	= 5
2	4	8	3	= 5
3	3	4	11	= 6
4	5	8	2	= 5
5	10	3	5	= 3
6	5	8	8	

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

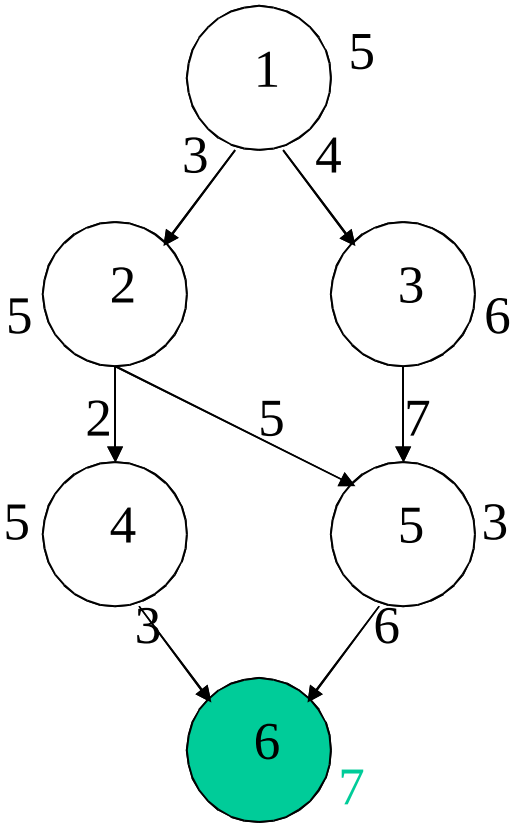
An Example of HEFT



Task	P1	P2	P3	
1	3	5	7	= 5
2	4	8	3	= 5
3	3	4	11	= 6
4	5	8	2	= 5
5	10	3	5	= 3
6	5	8	8	= 7

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

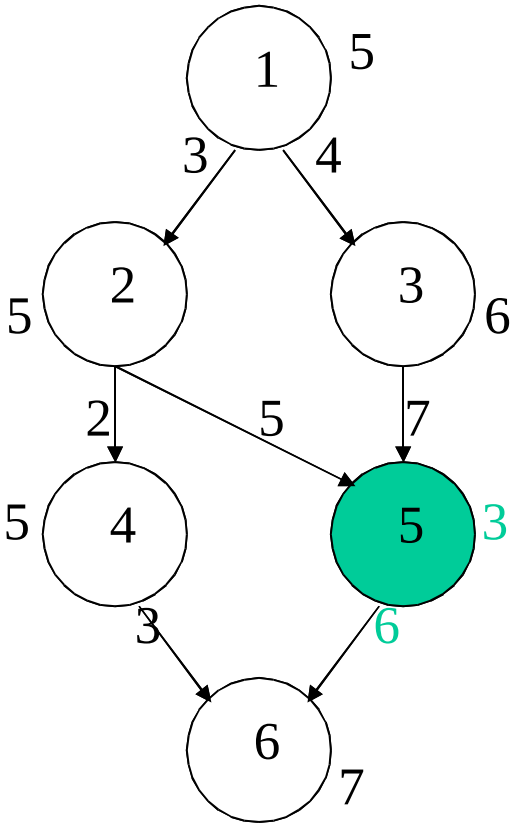
An Example of HEFT



Task	Rank
1	
2	
3	
4	
5	
6	7

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

An Example of HEFT

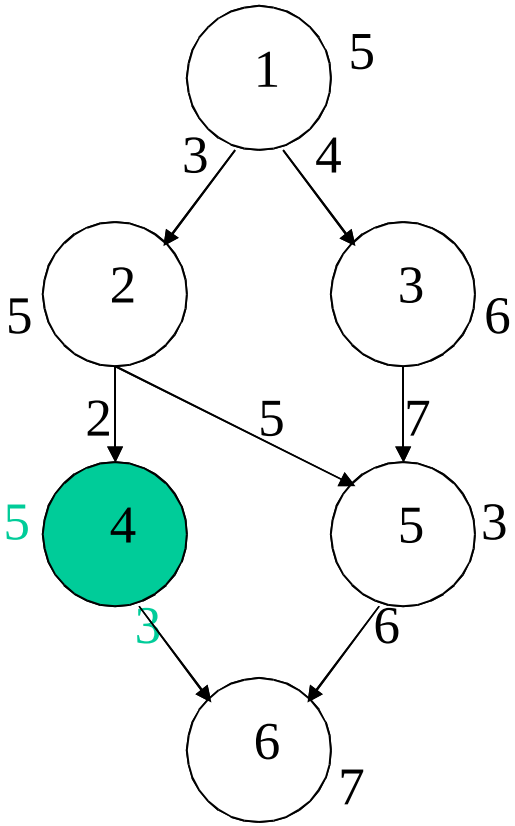


7 + 6 + 3 = 16

Task	Rank
1	
2	
3	
4	
5	16
6	7

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

An Example of HEFT

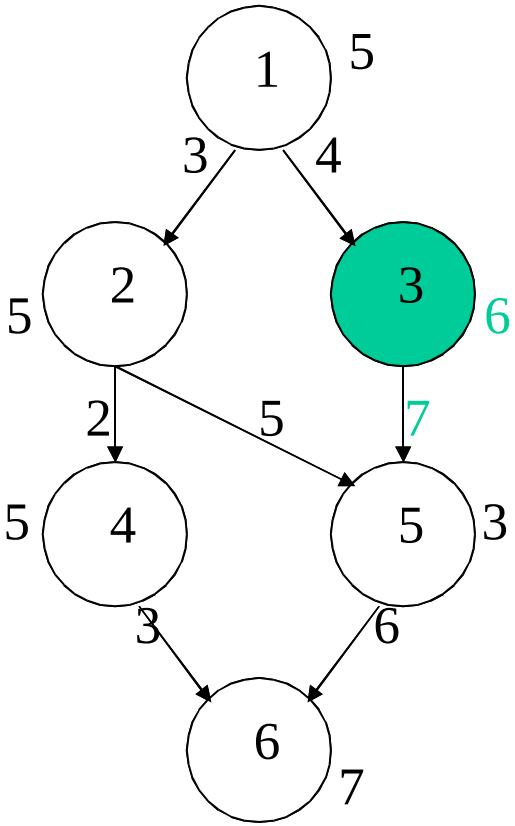


7 + 3 + 5 = 15

Task	Rank
1	
2	
3	
4	15
5	16
6	7

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

An Example of HEFT

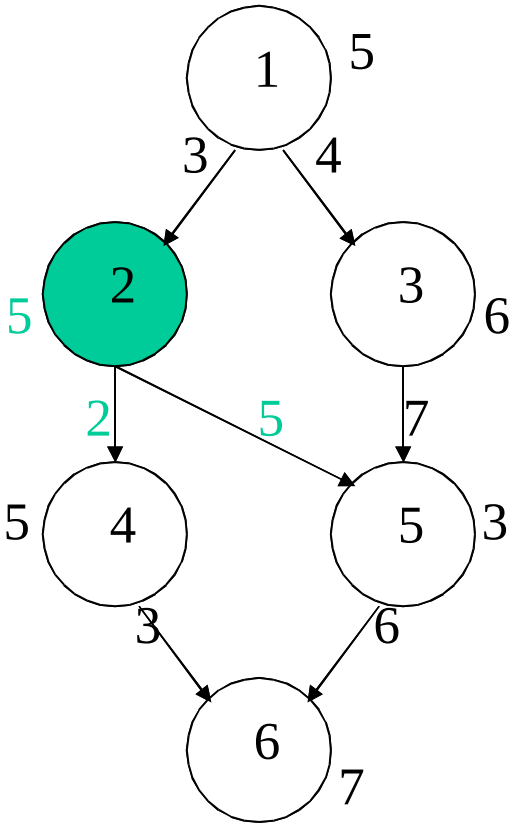


16 + 7 + 6 = 29

Task	Rank
1	
2	
3	29
4	15
5	16
6	7

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

An Example of HEFT

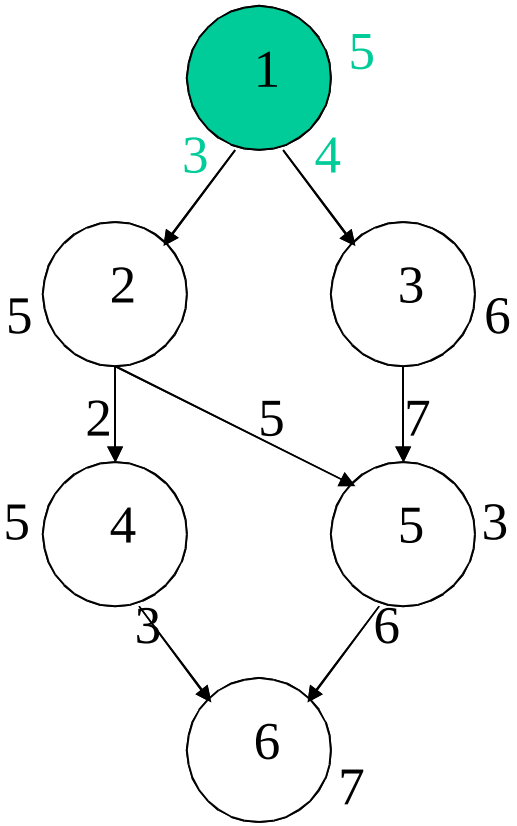


max (15 + 2 + 5 ,
16 + 5 + 5) = 26

Task	Rank
1	
2	26
3	29
4	15
5	16
6	7

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

An Example of HEFT



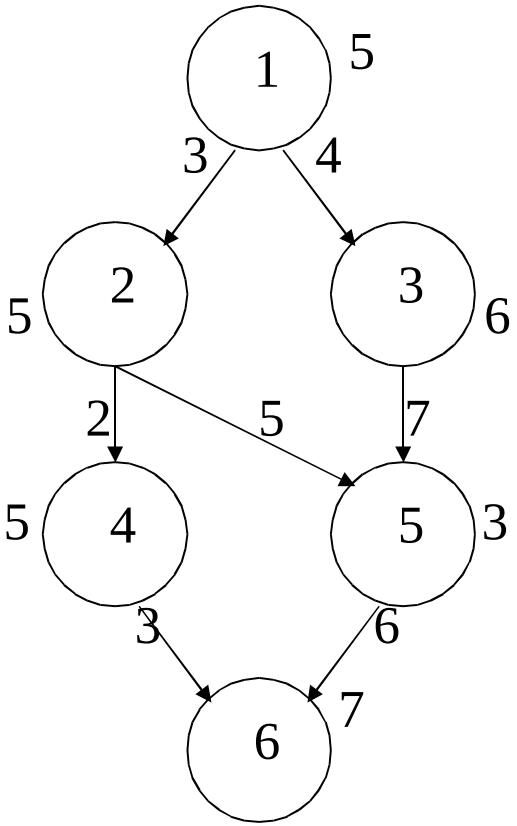
max (26 + 3 + 5 ,
29 + 4 + 5) = 38

Task	Rank
1	38
2	26
3	29
4	15
5	16
6	7

Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

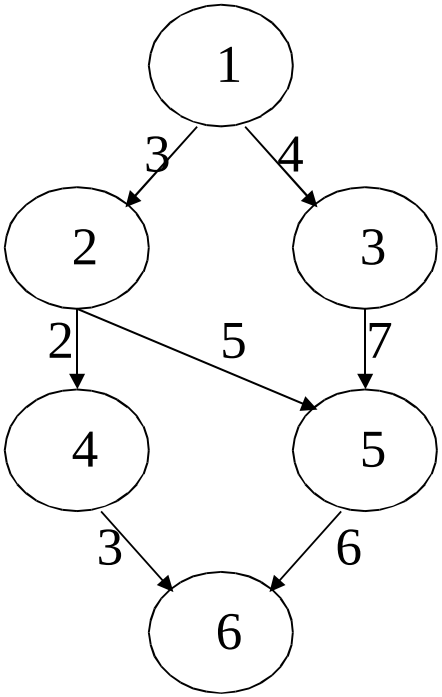
An Example of HEFT

{1, 3, 2, 5, 4, 6}



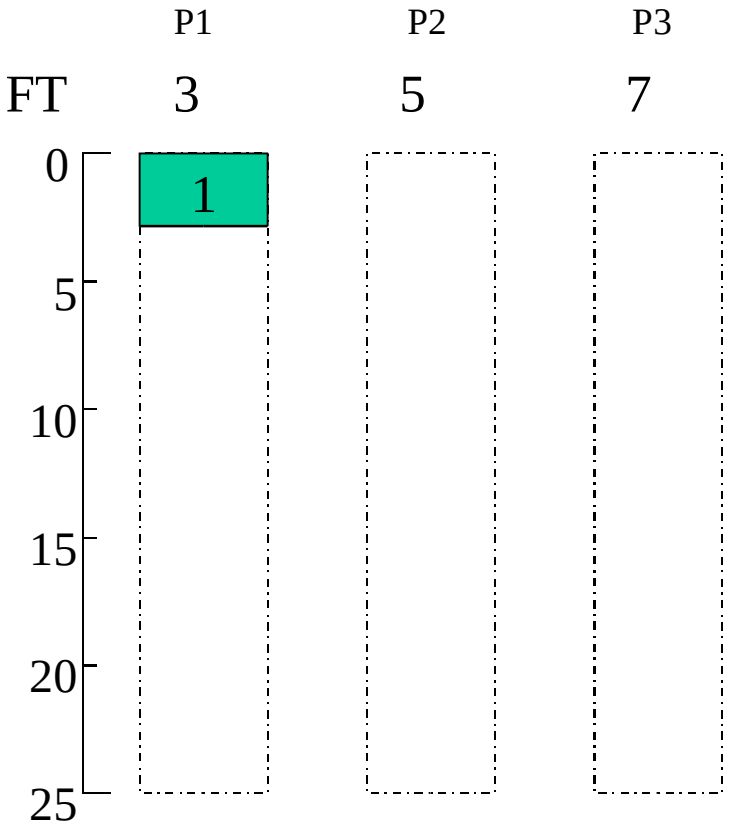
Task	Rank
1	38
2	26
3	29
4	15
5	16
6	7

An Example of HEFT

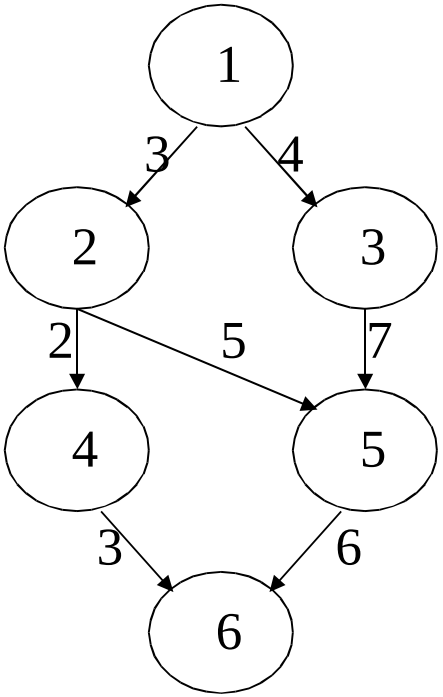


Task	P1	P2	P3
1	3	5	7
2	4	8	3
3	3	4	11
4	5	8	2
5	10	3	5
6	5	8	8

{1, 3, 2, 5, 4, 6}

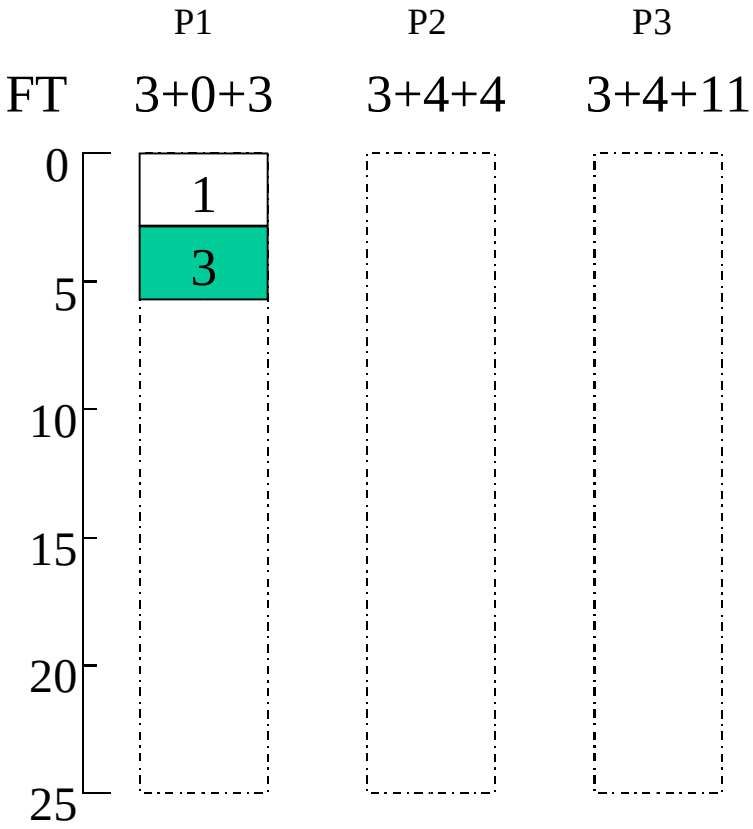


An Example of HEFT

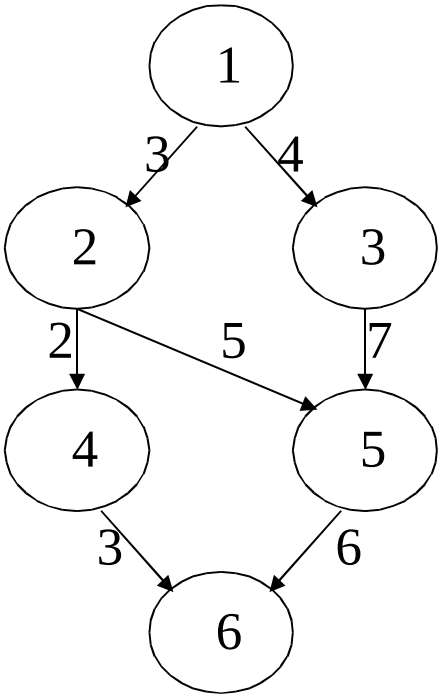


Task	P1	P2	P3
1	3	5	7
2	4	8	3
3	3	4	11
4	5	8	2
5	10	3	5
6	5	8	8

{1, 3, 2, 5, 4, 6}

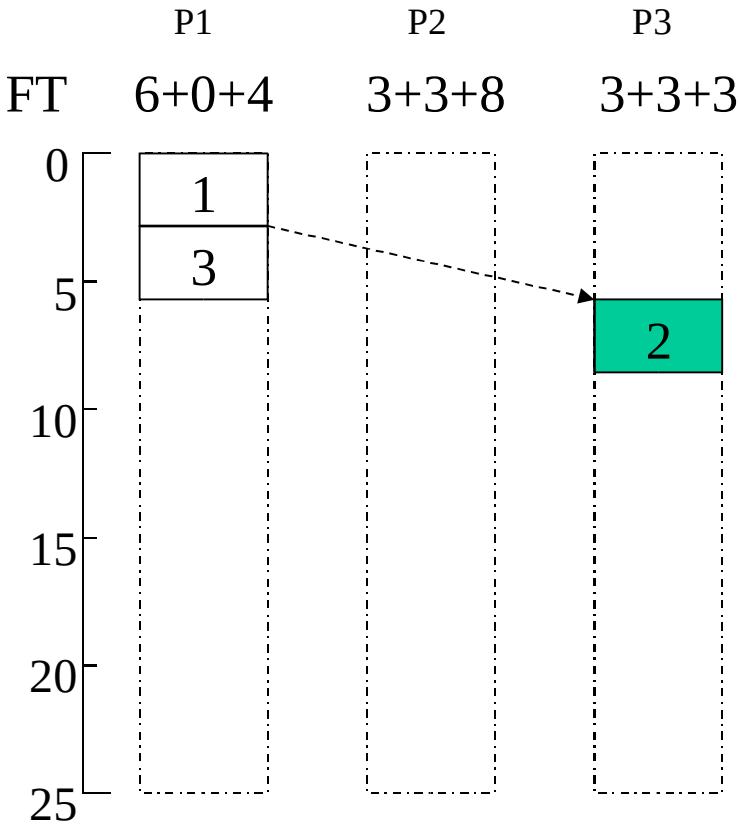


An Example of HEFT

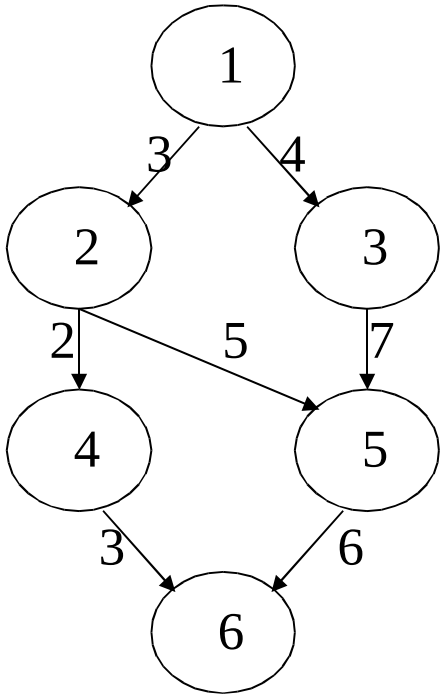


Task	P1	P2	P3
1	3	5	7
2	4	8	3
3	3	4	11
4	5	8	2
5	10	3	5
6	5	8	8

{1, 3, 2, 5, 4, 6}

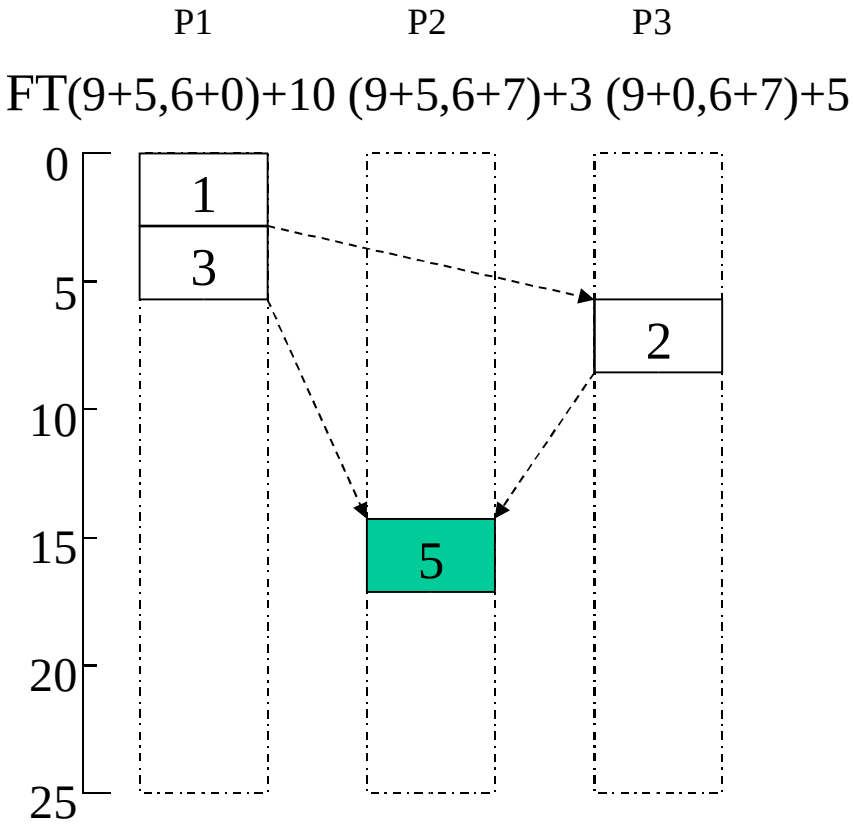


An Example of HEFT

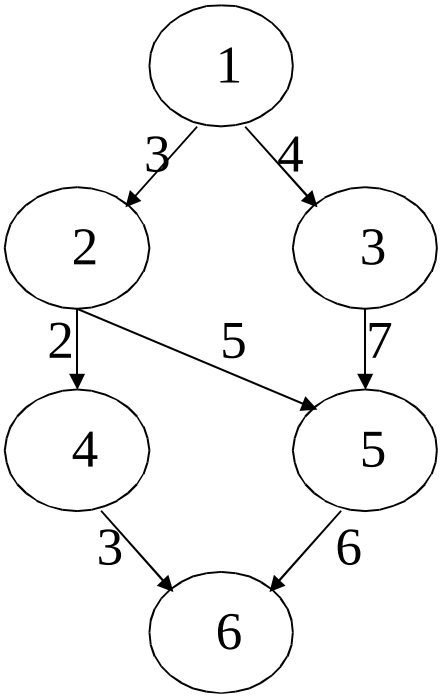


Task	P1	P2	P3
1	3	5	7
2	4	8	3
3	3	4	11
4	5	8	2
5	10	3	5
6	5	8	8

{1, 3, 2, 5, 4, 6}

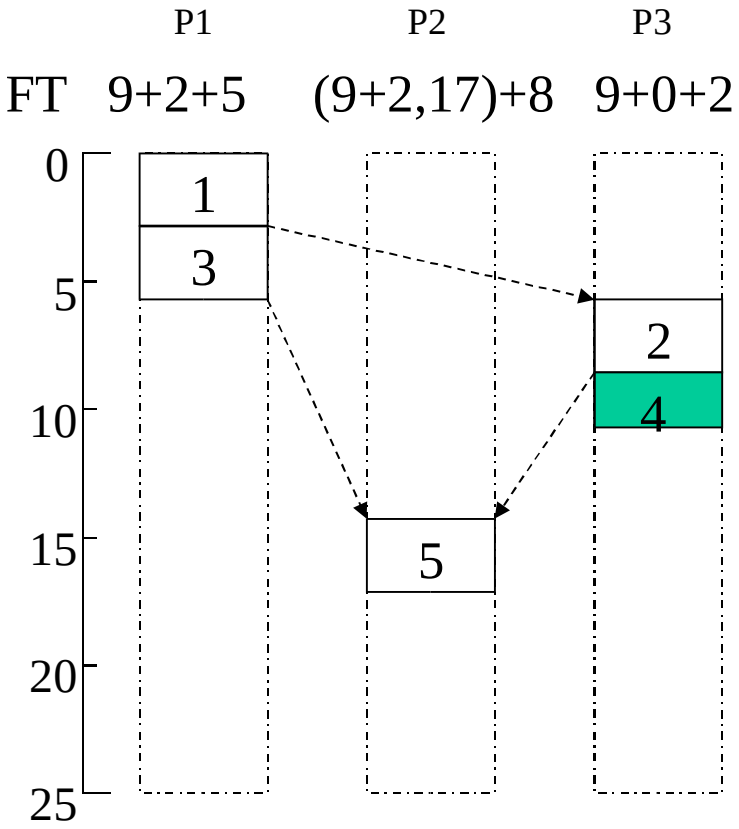


An Example of HEFT

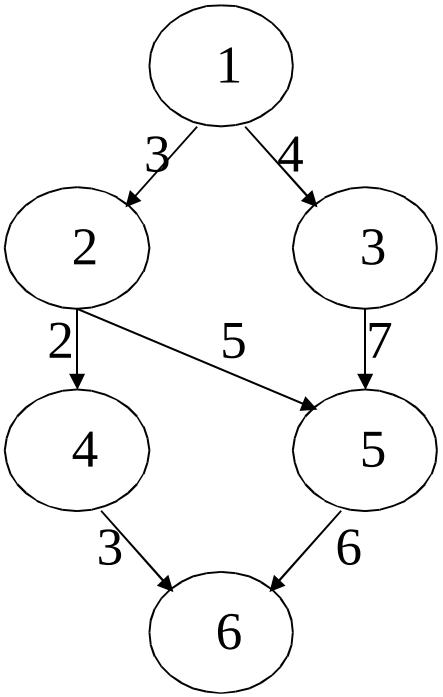


Task	P1	P2	P3
1	3	5	7
2	4	8	3
3	3	4	11
4	5	8	2
5	10	3	5
6	5	8	8

{1, 3, 2, 5, 4, 6}

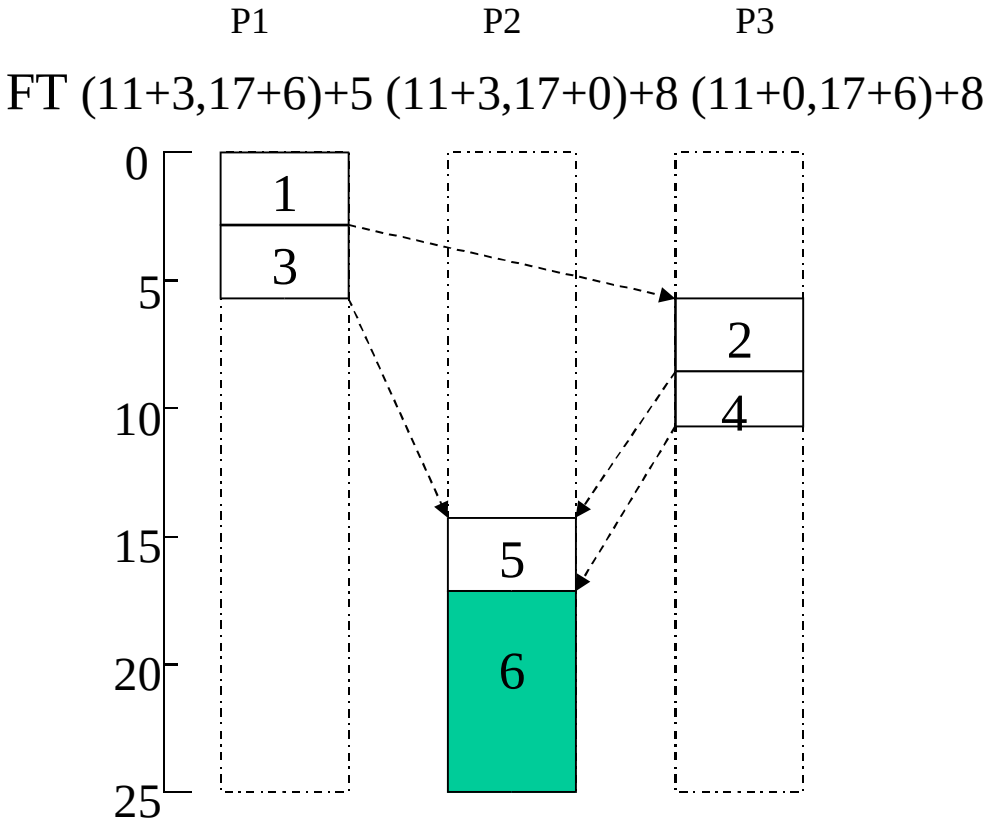


An Example of HEFT



Task	P1	P2	P3
1	3	5	7
2	4	8	3
3	3	4	11
4	5	8	2
5	10	3	5
6	5	8	8

{1, 3, 2, 5, 4, 6}



So far, everything is static…

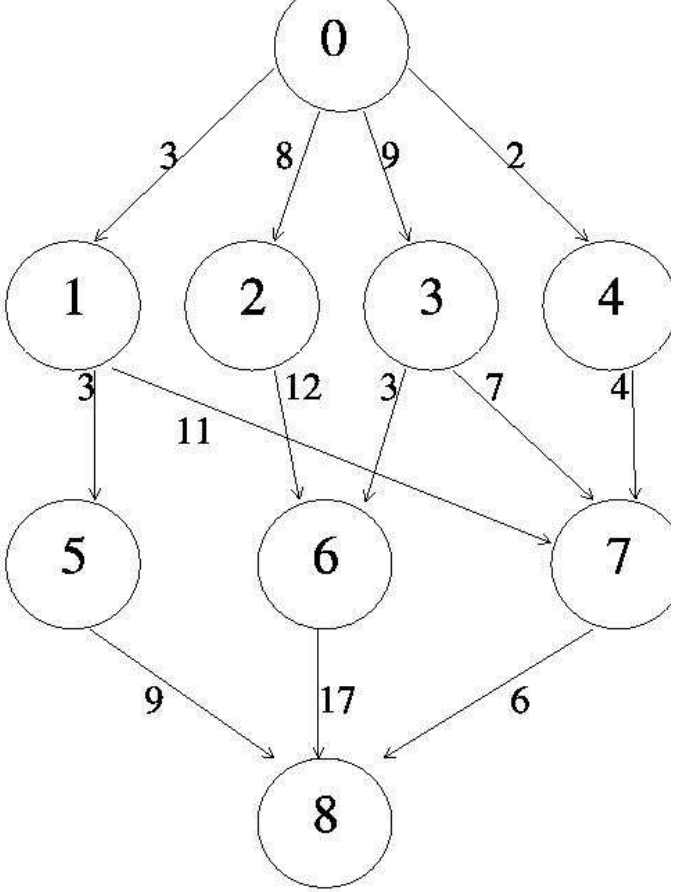
- DAG scheduling on heterogeneous systems is very sensitive to the weights of the nodes and edges…
- Little work takes into account dynamic changes…
- How good would be the schedule if the real execution time of the jobs was different?
- How robust? (i.e., not changes needed in the schedule for small variations?)
- An obvious solution:
  - rescheduling at run-time - how to minimize the number of times rescheduling is performed? Is repeated rescheduling good?

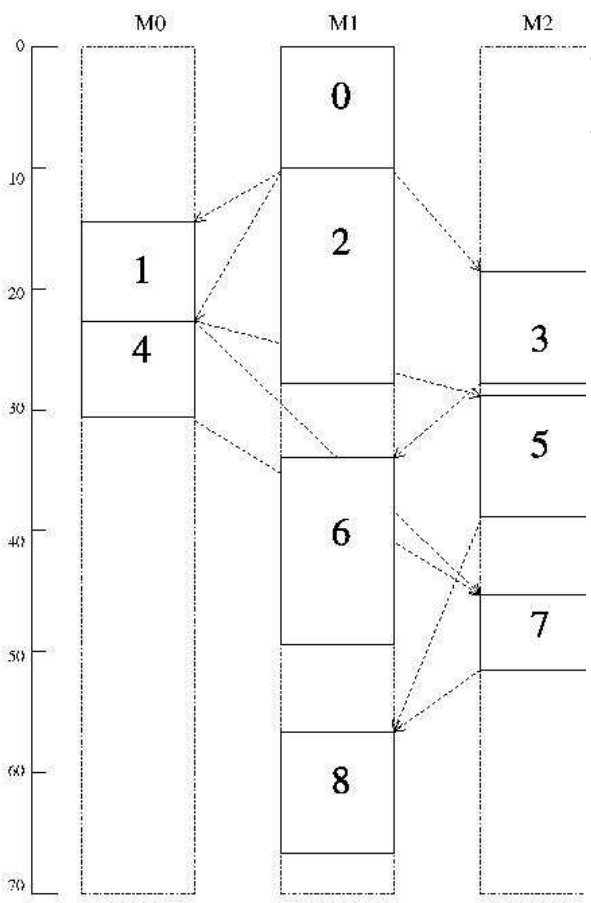
Henan Zhao, Rizos Sakellariou. A Low-Cost Rescheduling Policy...

Characterizing the Schedule

- **Spare time**
  - A node i with an immediate successor j: ST(j)-DAT(i,j)
  - A node i with the next node on the same machine: ST(j)-FT(i)
  - The minimum of all the above: MinSpare, indicates the maximal value that won't affect the start of a successor.
- **Slack** of a node i with all its immediate successors j: indicates the maximal delay of a task.
  - Slack(i)=min(slack(j)+spare(i,j))

Example





FT(4)=32.5, DAT(4,7)=40.5, ST(7)=45.5 – spare 5

FT(3)=28, ST(5)=29.5 – spare 1.5

Slack(8)=0; Slack(7)=Slack(8)+Spare(7)=0;
Slack(5)=6

The Idea

- Assume that the static execution time varies at run-time
- Assume that run-time rescheduling would take into account the latest available information.
  - Run-time rescheduling has a cost (we don't specify)
- Keep track of the values of slack (or, in a variant of the policy – MinSpare): perform rescheduling ***only*** when the start time of a task is greater than the value of slack (or MinSpare)

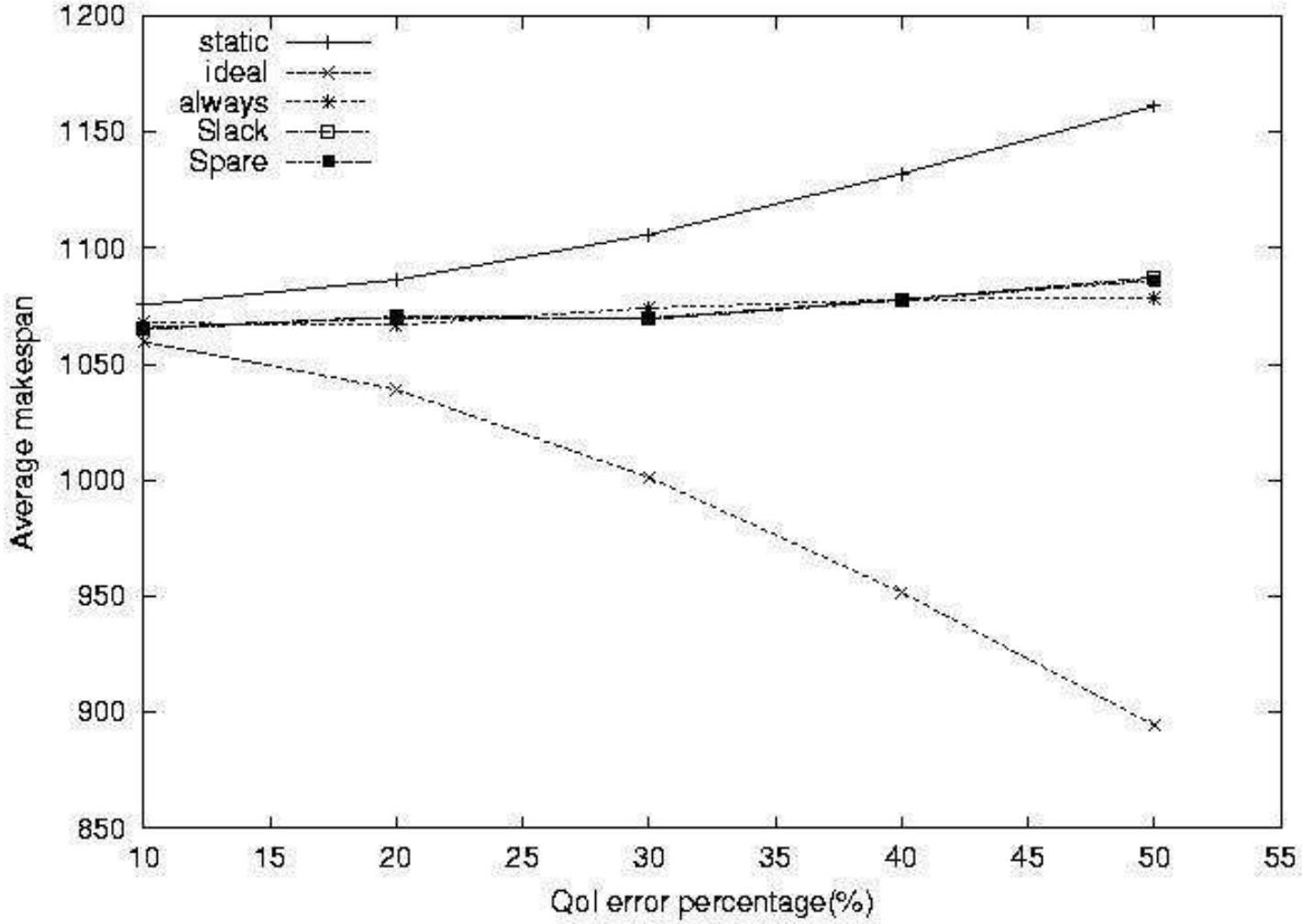
A Selective Rescheduling Algorithm

Input: an application graph G and a schedule S_1 produced by an algorithm A
Selective rescheduling policy:
(1) Mark all tasks in S_1 as unexecuted, *Unexecuted[]*
S_2 ← the real, post-execution schedule (initially empty)
(2) Compute for each task *i* from S_1, Slack(i) (or MinSpare(i))
(3) While (*Unexecuted[]* is not empty)
t ← first task in S_1, which is in *Unexecuted[]*
m ← the allocated machine for t in schedule S_1
if (t is not the entry task in G)
EST ← the expected start time of t in schedule S_1
RST ← the real start time of t on m in S_2
delay ← RST - EST
if (delay > Slack(t)) // or (delay > MinSpare(t))
S_1 = A(*Unexecuted[],* S_2)
compute MinSpare for all tasks in S_1, also in *Unexecuted[]* // or Slack
t ← first task in S_1, which is in *Unexecuted[]*
m ← the allocated machine for t in schedule S_1
endif
endif
execute task t on machine m
S_2 ← S_2 ∪ {(t, m)}
Unexecuted[] ← *Unexecuted[]* \ t
endwhile

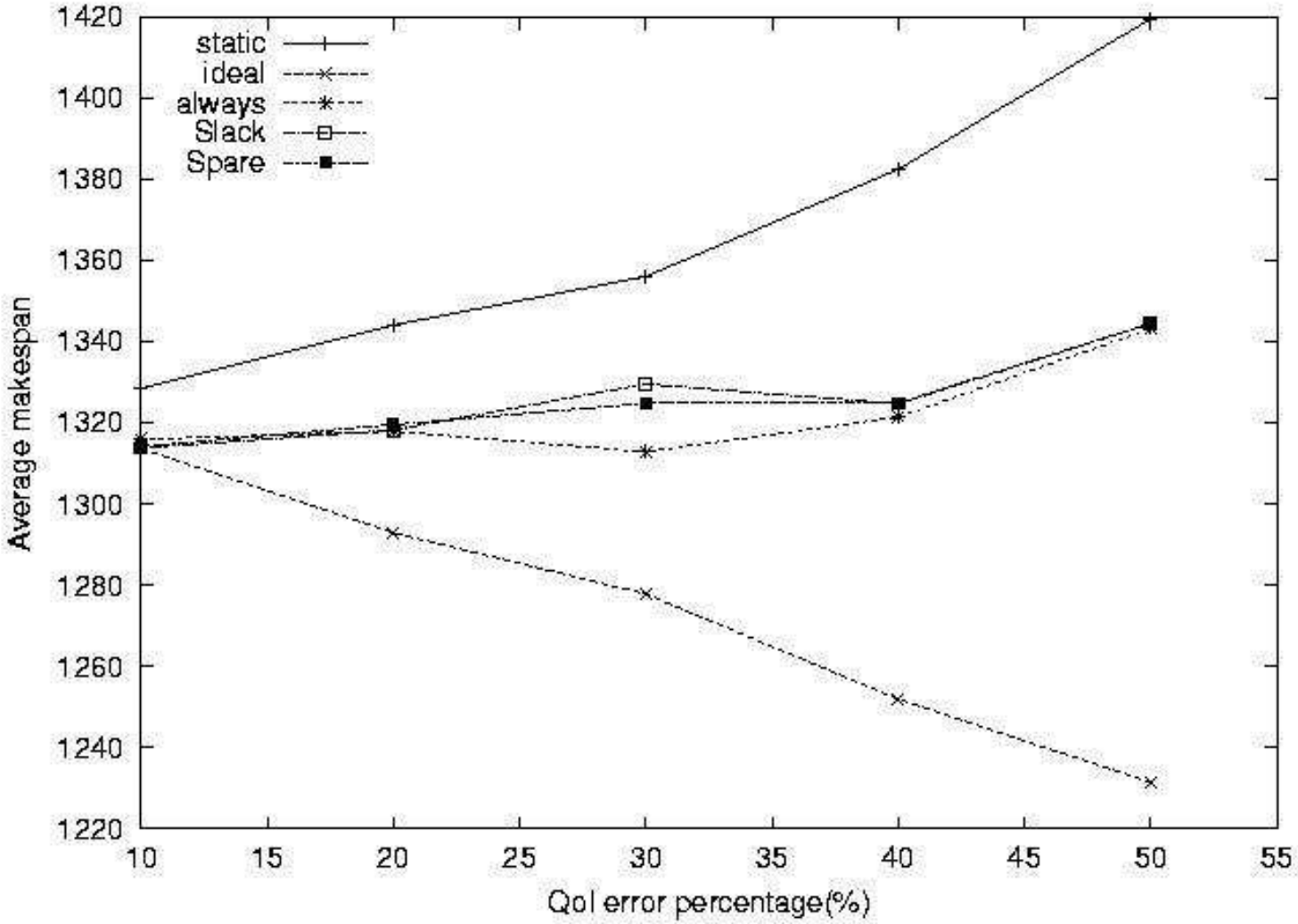
Performance Evaluation

- Randomly Generated DAGs (50-100 tasks)
- 3 to 8 machines
- Estimated Execution Time of each task: 50-100
- Error of execution time up to 50%
- 4 different static DAG scheduling heuristics: FCP, DLS, HEFT, LMT
- Comparison of schedules:
  - Static: based on estimated execution time
  - Always: reschedule before each task starts execution
  - Ideal: post mortem: i.e., knowing run-time values
  - Our selective rescheduling policy

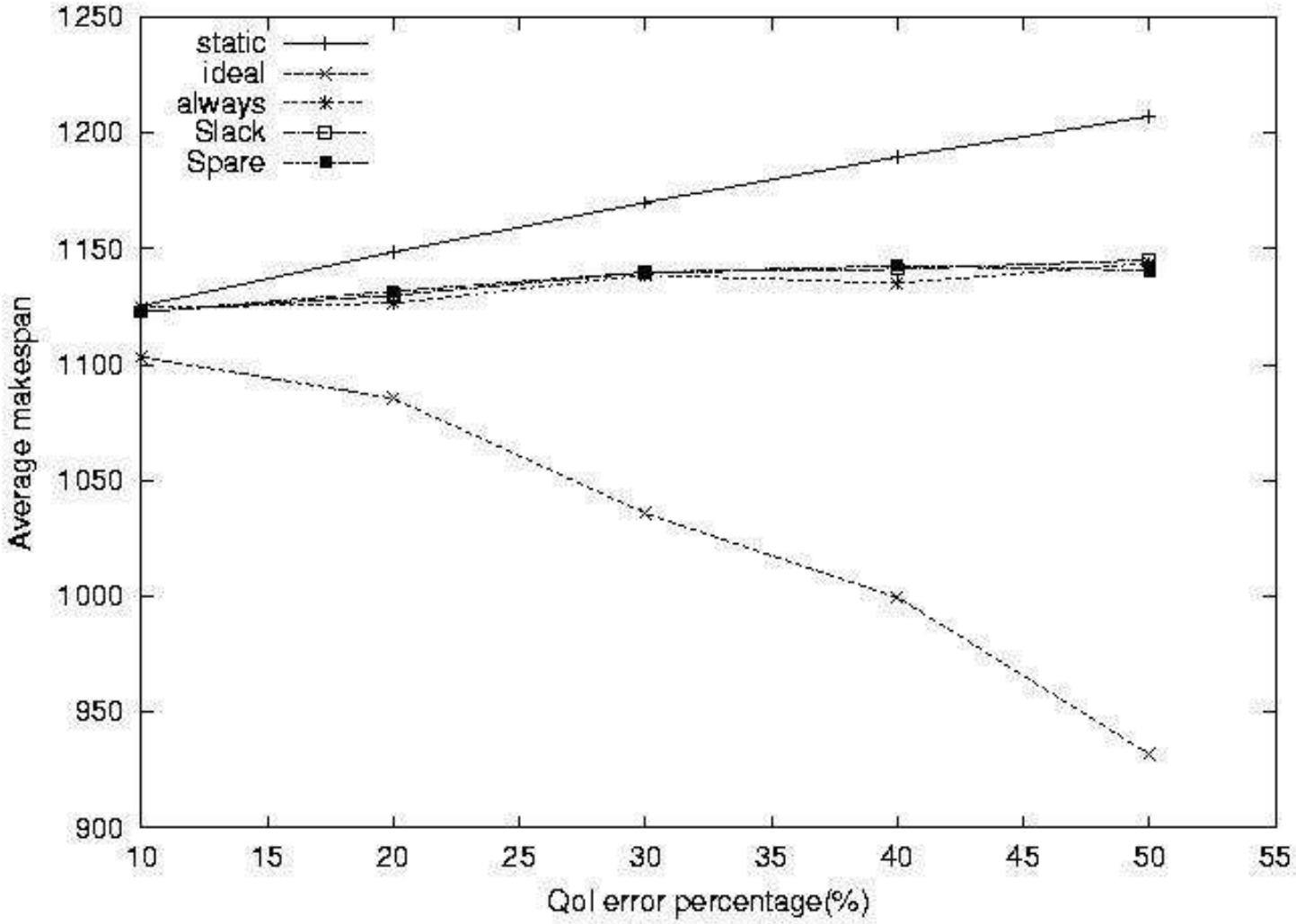
DLS


static
ideal
always
Slack
Spare
Average makespan
1200
1150
1100
1050
1000
950
900
850
10
15
20
25
30
35
40
45
50
55
QoI error percentage(%)

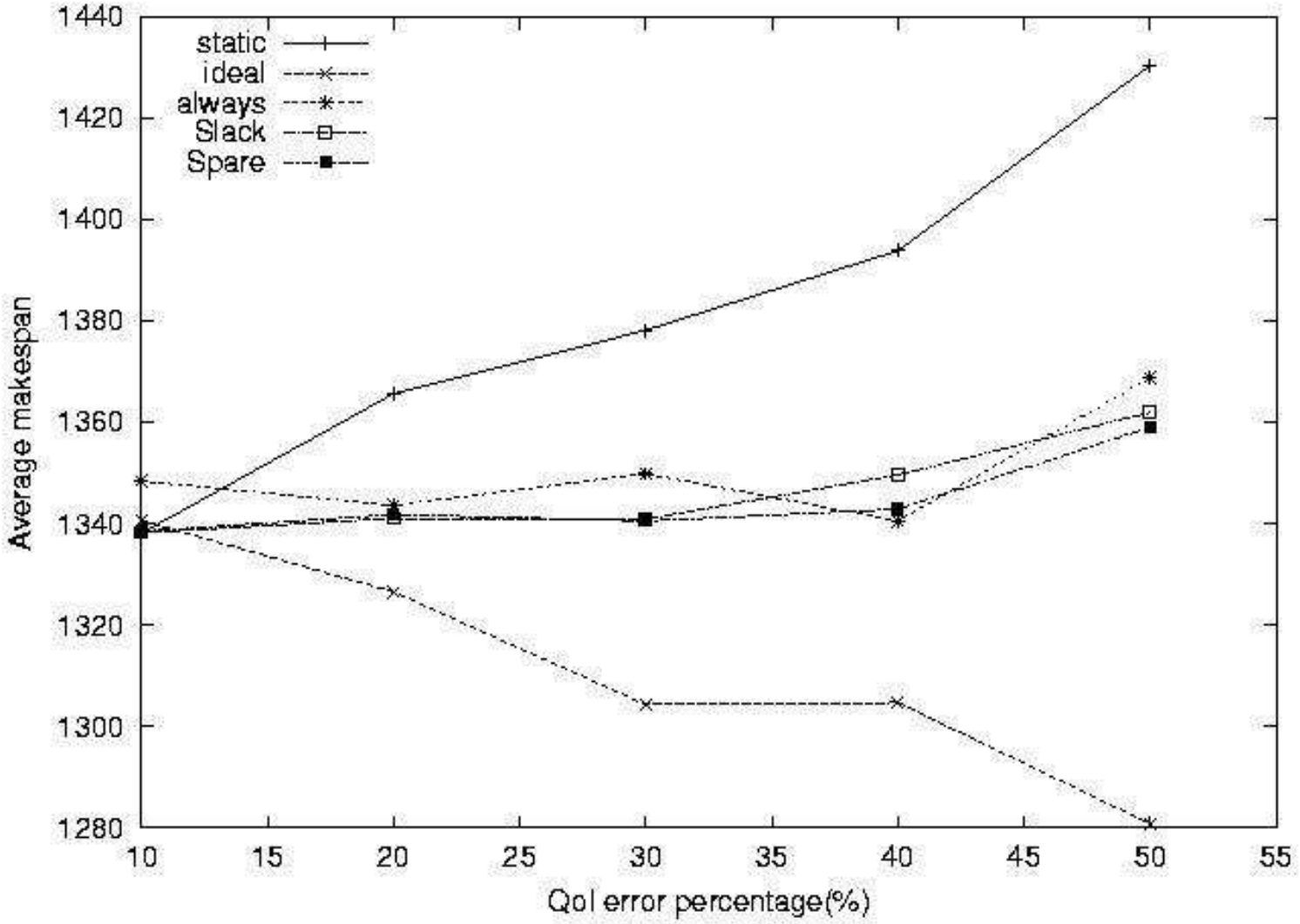
FCP


static
ideal
always
Slack
Spare
Average makespan
1420
1400
1380
1360
1340
1320
1300
1280
1260
1240
1220
10
15
20
25
30
35
40
45
50
55
QoI error percentage(%)

HEFT

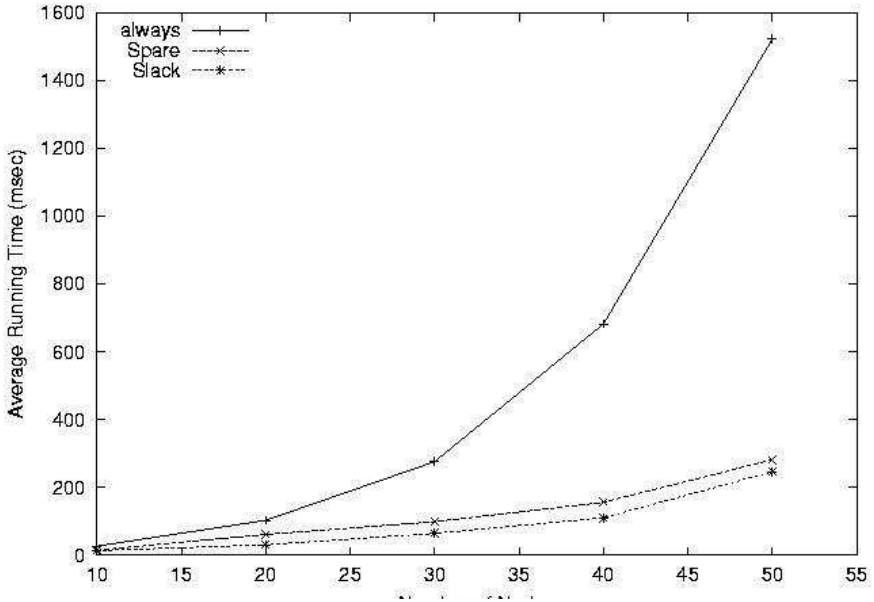

static
ideal
always
Slack
Spare
Average makespan
Qol error percentage(%)
1250
1200
1150
1100
1050
1000
950
900
10
15
20
25
30
35
40
45
50
55

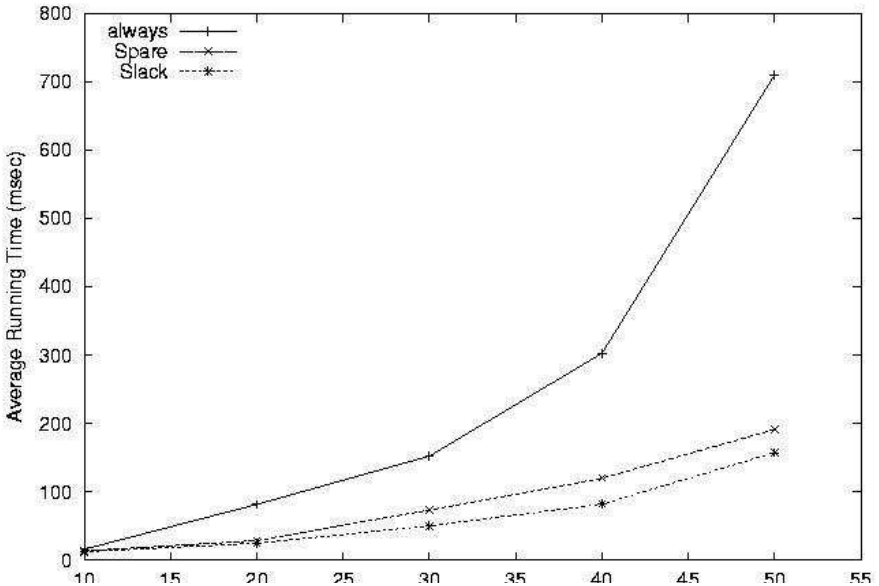
LMT

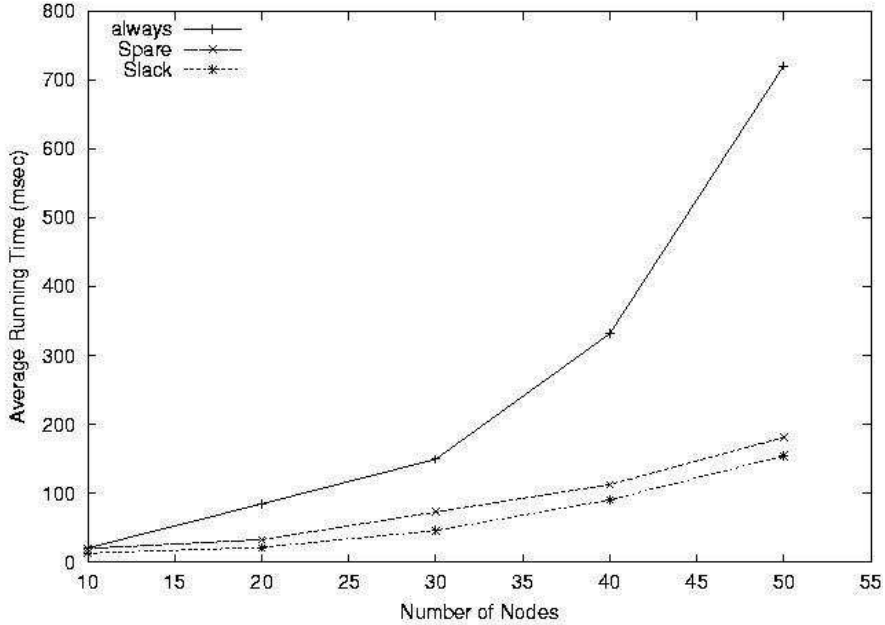

static
ideal
always
Slack
Spare
Average makespan
1440
1420
1400
1380
1360
1340
1320
1300
1280
10
15
20
25
30
35
40
45
50
55
QoI error percentage(%)

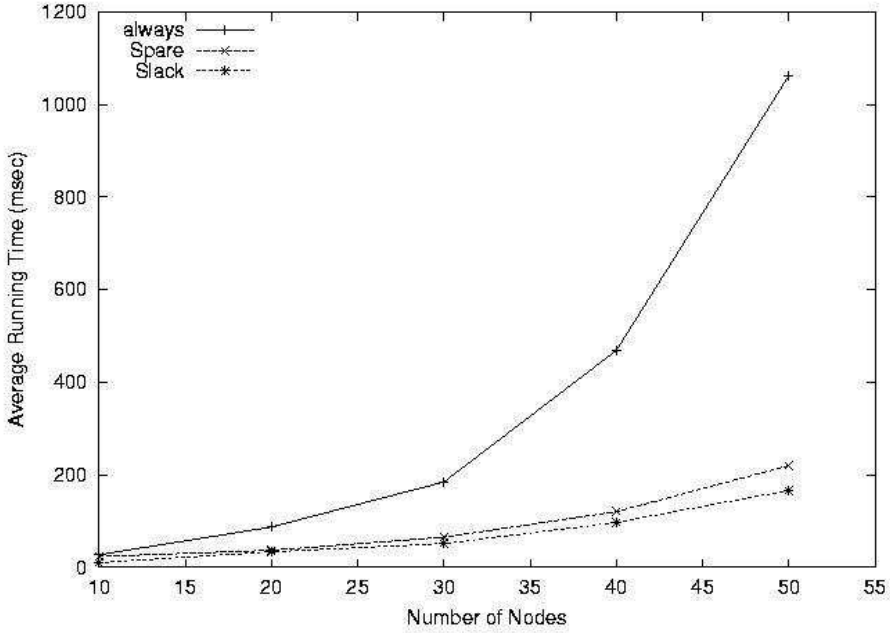
Running Time

- The rescheduling policies would attempt to reschedule at no more than 20% of the number of cases.
- As the number of nodes increases, differences in running time are more profound.
- (running time cost refers to the cost of running the algorithm)


always
Spare
Slack
Average Running Time (msec)
0
200
400
600
800
1000
1200
1400
1600
10
15
20
25
30
35
40
45
50
55


always
Spare
Slack
Average Running Time (msec)
0
100
200
300
400
500
600
700
800
10
15
20
25
30
35
40
45
50
55


always
Spare
Slack
Average Running Time (msec)
0
100
200
300
400
500
600
700
800
10
15
20
25
30
35
40
45
50
55
Number of Nodes


always
Spare
Slack
Average Running Time (msec)
0
200
400
600
800
1000
1200
10
15
20
25
30
35
40
45
50
55
Number of Nodes

Conclusions and Future Work

- An initial characterization of a schedule using spare time and slack is possible
- We don’t gain anything by repeated rescheduling!
- Work needs to be done
  - …robust heuristics for DAG scheduling (see hybrid heuristic to appear in the Heterogeneous Computing Workshop – HCW 2004)
  - …to evaluate static schedules in the presence of run-time dynamic changes
- See poster “*Service Level Agreement Based Scheduling Heuristics*” by Jon MacLaren *et al*