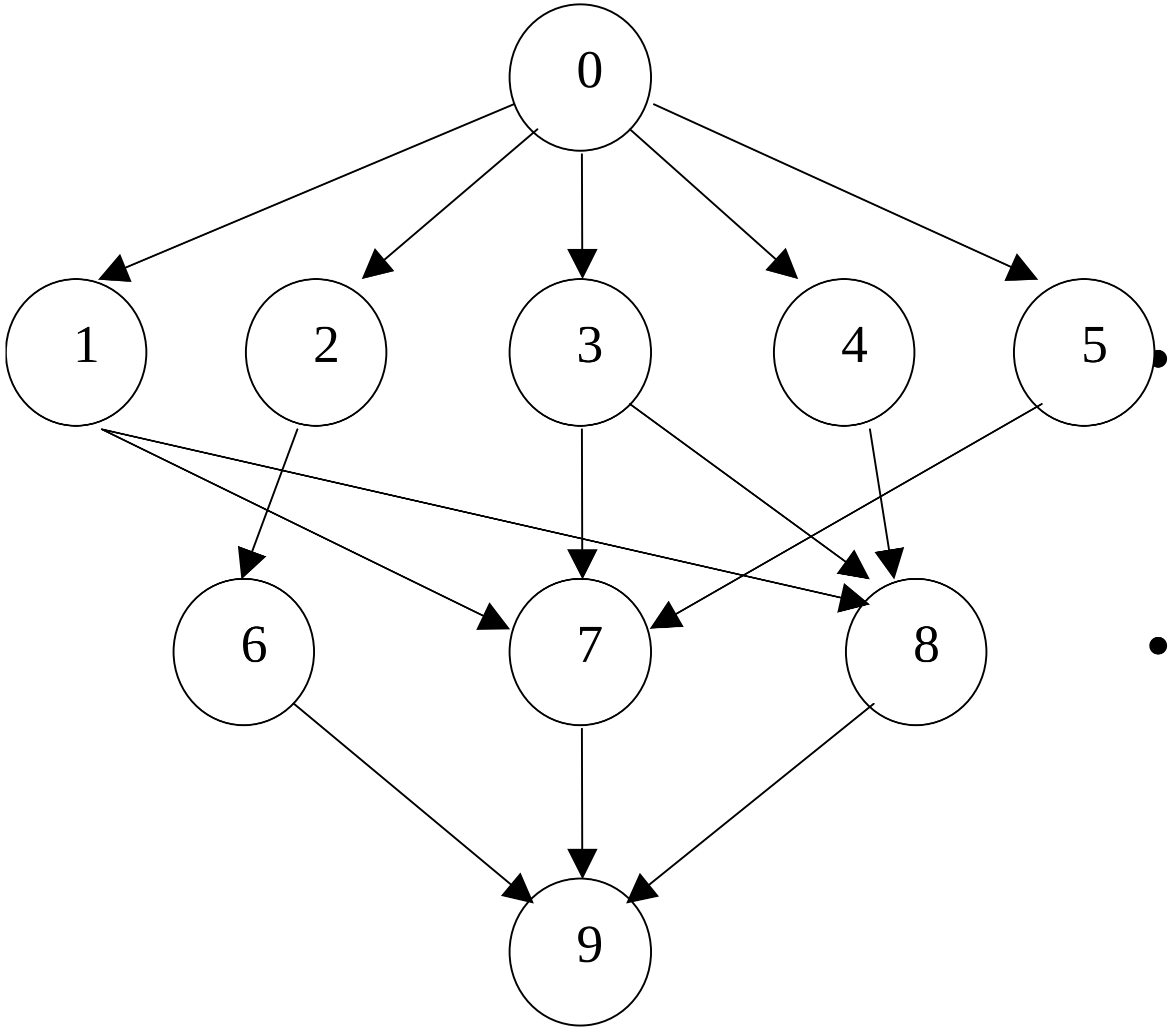
Parallel DAG Scheduling: Recent Results and New Directions

Rizos Sakellariou

University of Manchester

The Problem

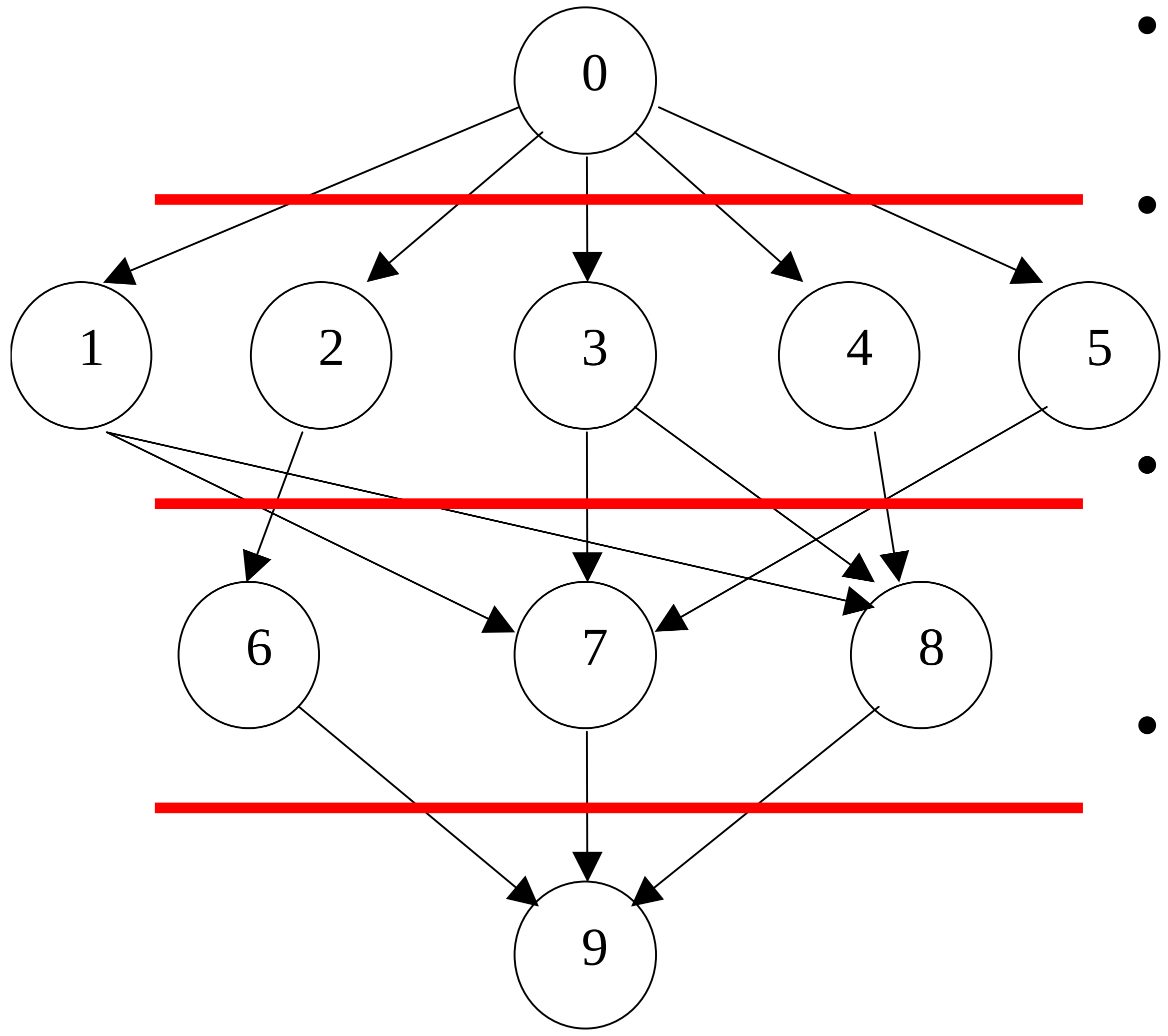


- Many applications can be represented by a *Directed Acyclic Graph* (DAG)
- Dependences between tasks (nodes) must be respected
- How do we allocate tasks onto machines to exploit parallelism and finish as soon as possible?

Some background

- Parallel processing of precedence graphs has been studied as a problem since the 1960s.
- Difficult to come up with generic solutions, we need heuristics with good performance.
- There are many variations to the problem.
- Good overview (of work until then) in ACM Computing Surveys (1999).
- In recent years, there has been an increased interest.

A simple solution

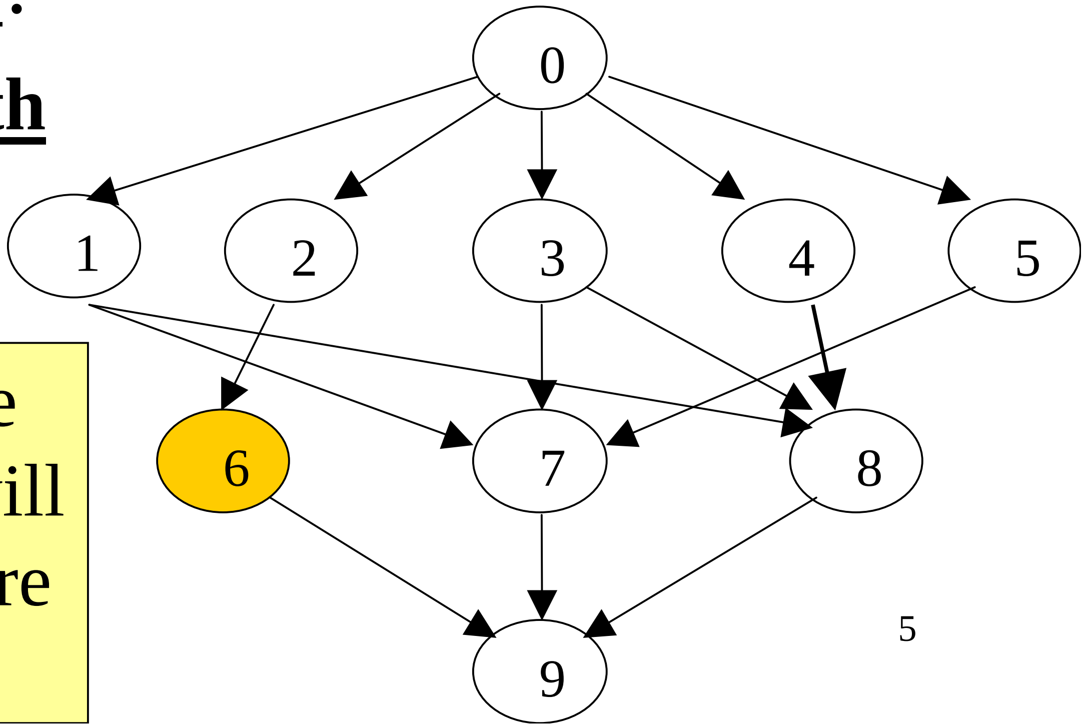


- Divide the graph into ‘levels’.
- Schedule each level as a ‘bag of independent tasks’
- May not always be possible to divide nicely.
- What if task (2) takes 1 time unit to run, task (6) takes 9 units and all the rest takes 5 units?

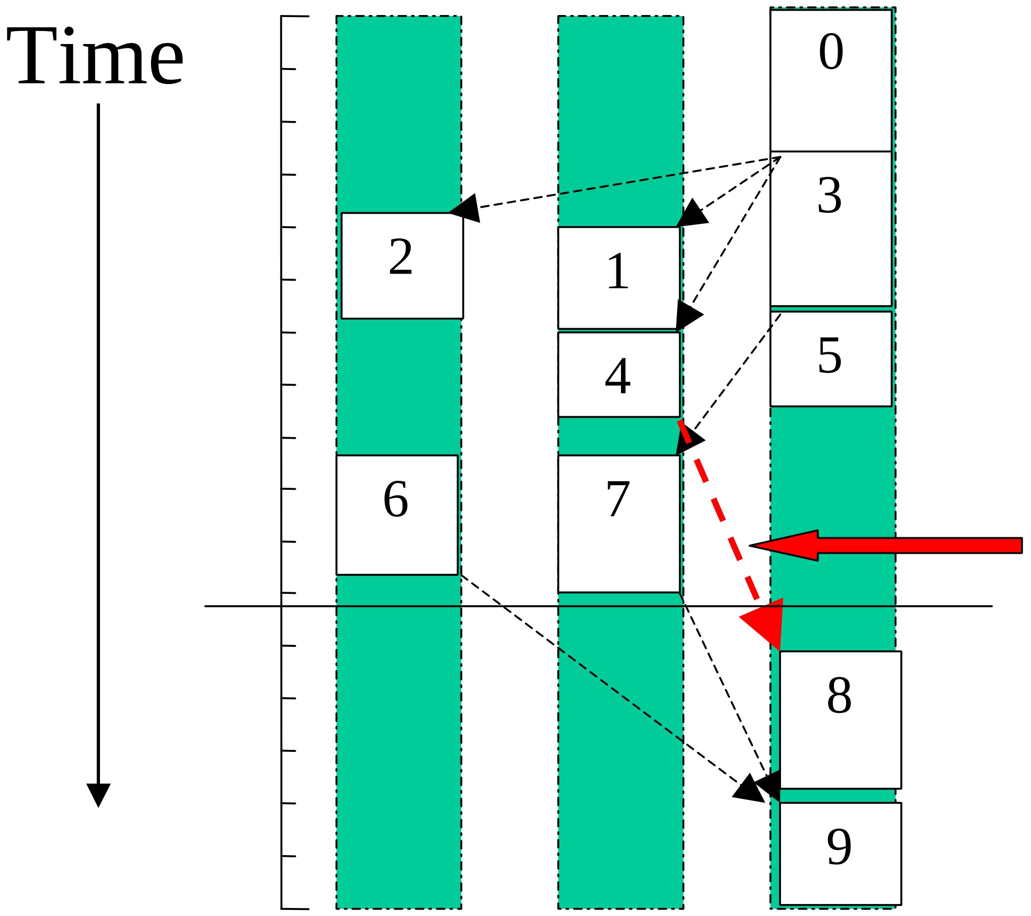
The key idea of a good solution

- Let’s assume we allocate tasks to machines one after the other, respecting dependences.
- How do we choose among multiple tasks?
  - E.g., after 0 finishes, any of 1,2,3,4,5 can be selected.
- <u>The order does matter</u>!
  - follow the <u>**critical path**</u>

Key: we need to minimize the chance that machines will remain idle. Make sure there are tasks available.



Communication does matter too…



Tasks on the same machine will need no communication.

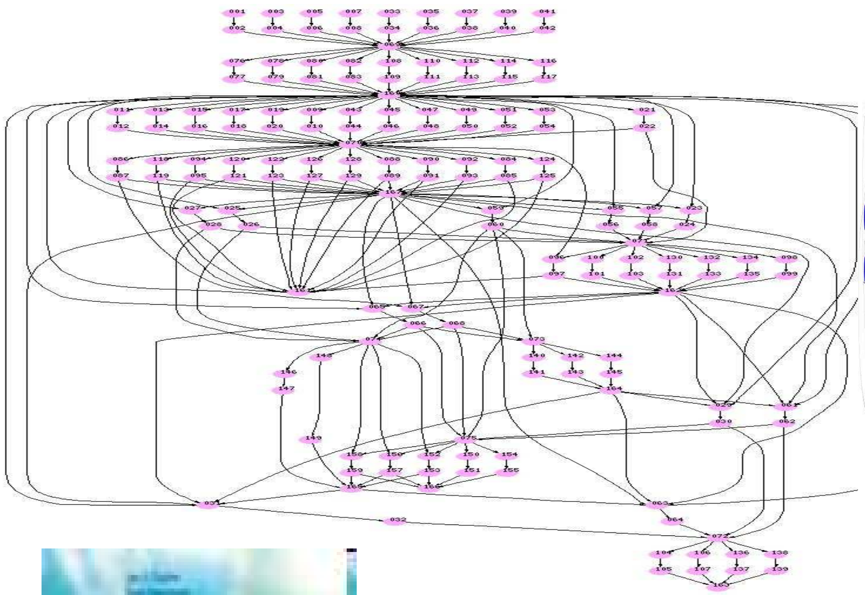
Note the role of the critical path. If the communication between nodes 4 and 8 takes way too long, the length of the schedule will keep increasing.

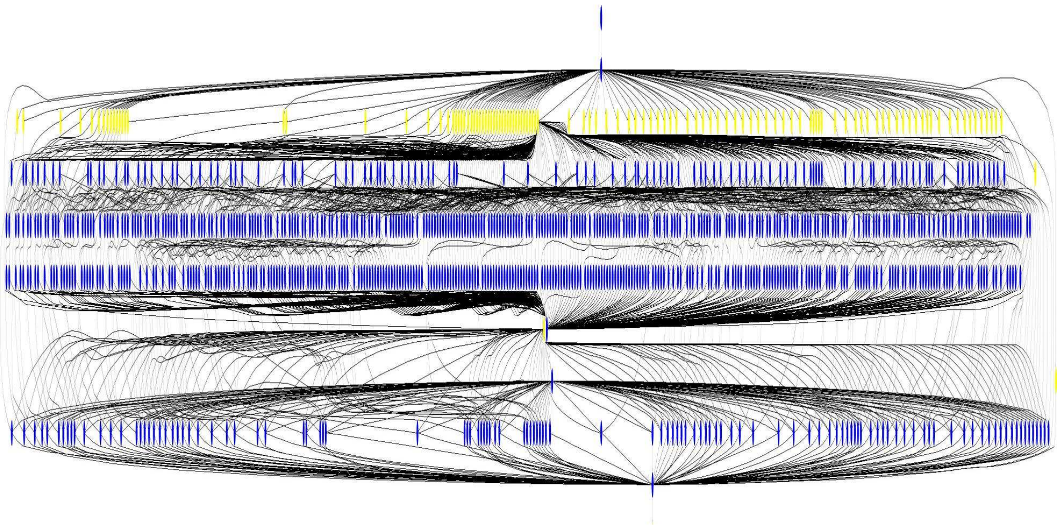
Increased interest in recent years

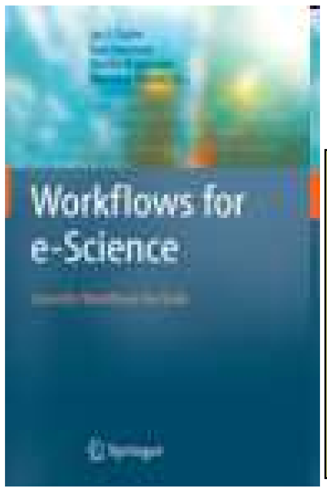
- Lots of interest in applications which can be represented as DAGs with tens of thousands of nodes.
- Large-scale distributed/heterogeneous platforms
- Additional interesting dimensions to an already difficult problem:
  - Execution environment (heterogeneous, queue-based)
  - The objective is not the minimization of execution time only but (monetary) cost, energy, …
- Practical solutions are needed!
  - Problems of more theoretical nature abound…

Scientific Workflows

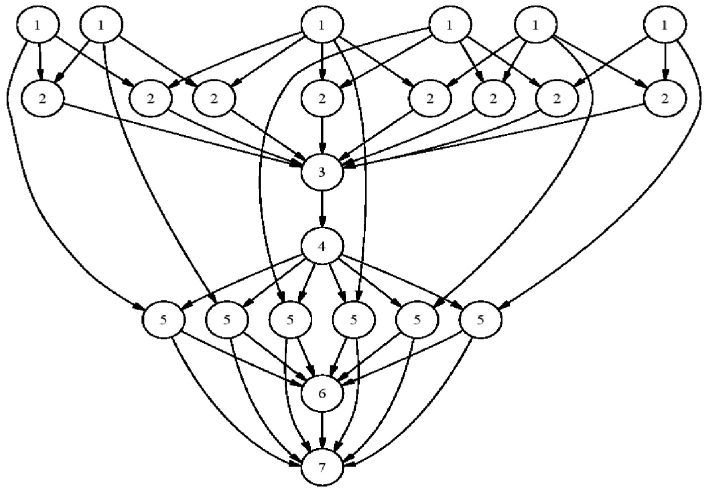
Many interesting scientific applications can be represented by DAGs





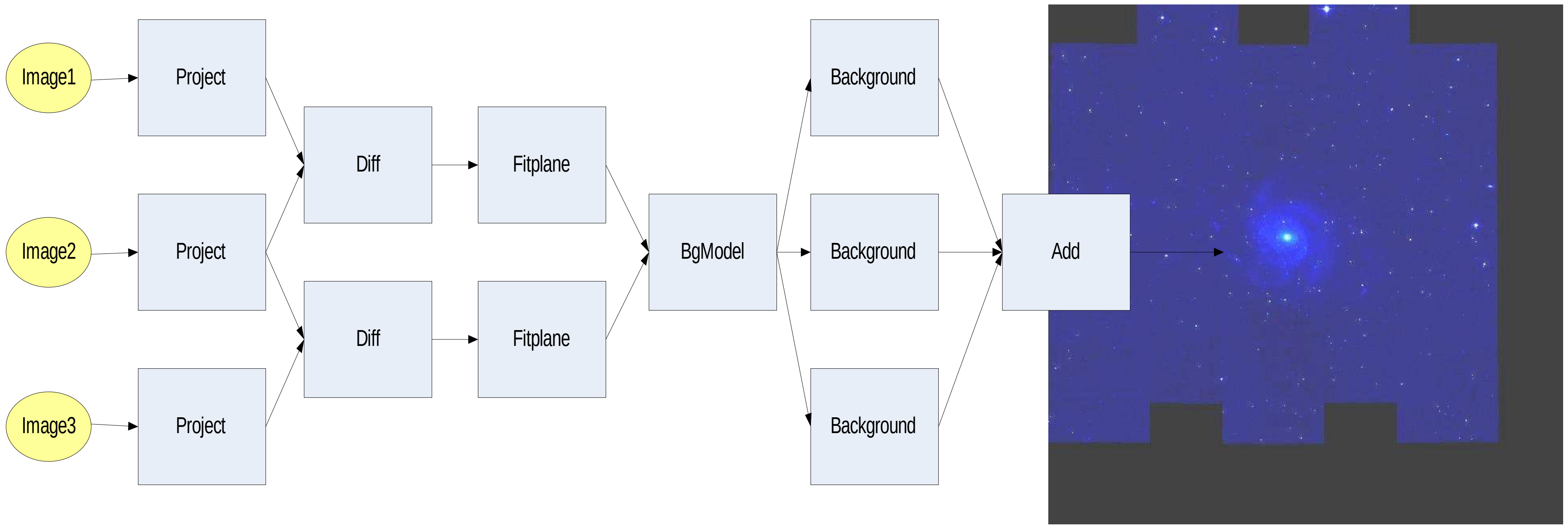


I. Taylor, E. Deelman, D. Gannon: Workflows for e-Science. Springer, 2007



The Montage Workflow

- Montage example: Generating science-grade mosaics of the sky
- http://montage.ipac.caltech.edu/



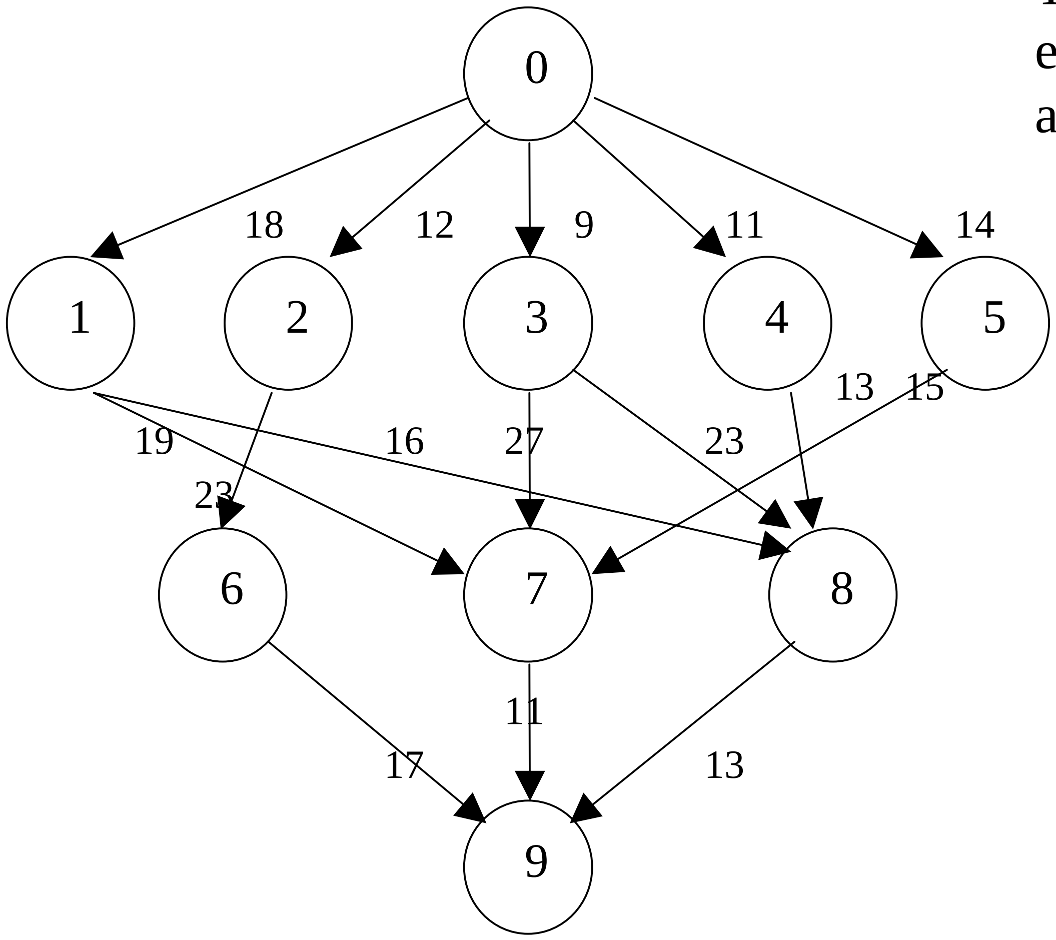
Outline

Parallel DAG scheduling:

- The ‘simple’ (static) case for heterogeneous environments
- Uncertainties in execution time
- Scheduling multiple DAGs
- Multi-criteria scheduling
- Dealing with queues
- Adapting the DAG

• Conclusion

How did it all start?



The model assumes that we know execution times on each machine and communication between them

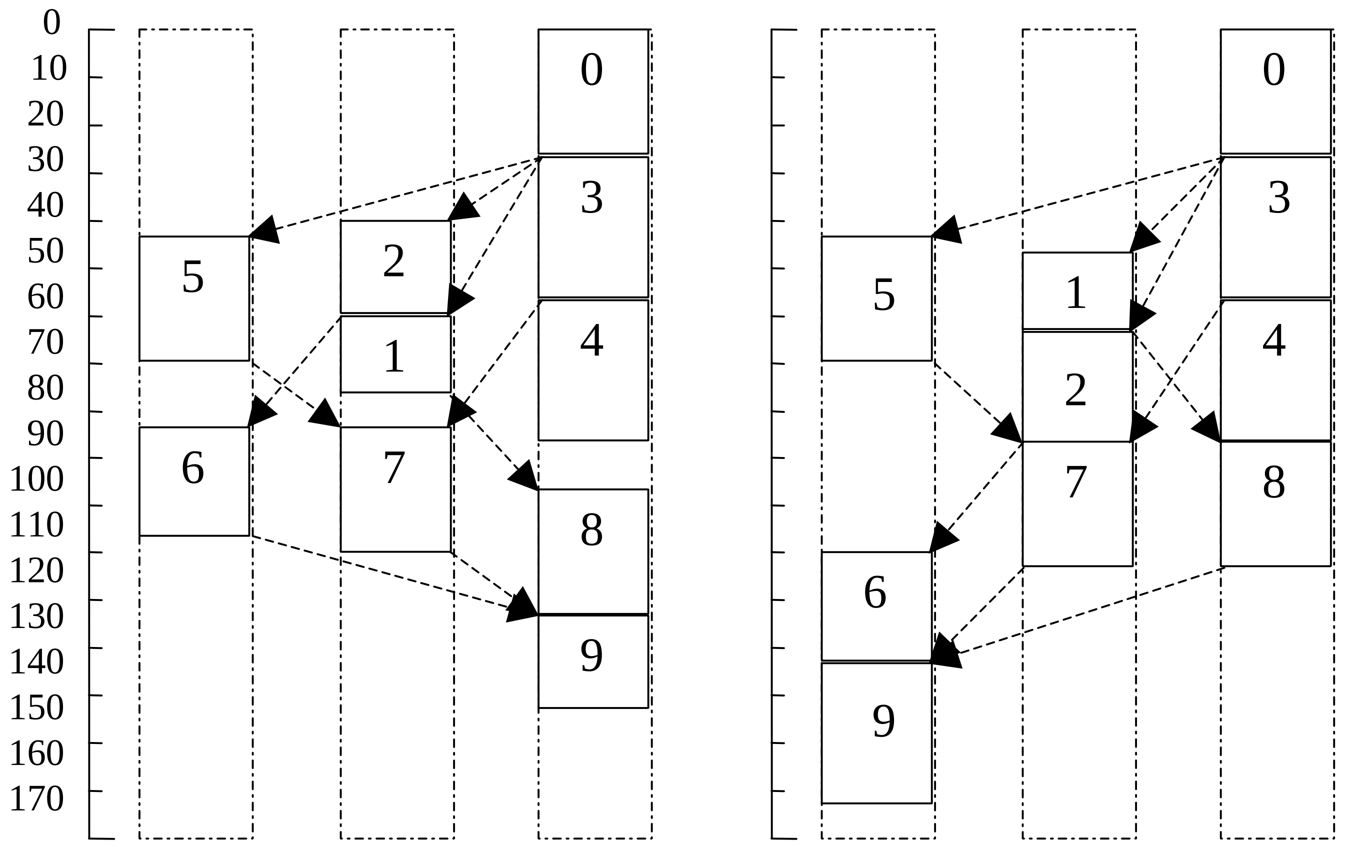
Task	M1	M2	M3
0	37	39	27
1	30	20	24
2	21	21	28
3	35	38	31
4	27	24	29
5	29	37	20
6	22	24	30
7	37	26	37
8	35	31	26
9	33	37	21

List scheduling is a well-known technique; we can make use of average values to compute a weight for each node and determine a scheduling order

Two Schedules – ~15% difference

- Two schedules for two different ways to calculate weights

Worst {0, 3, 5, 2, 1, 4, 7, 8, 6, 9} **Mean** {0, 3, 5, 1, 2, 4, 7, 8, 6, 9}

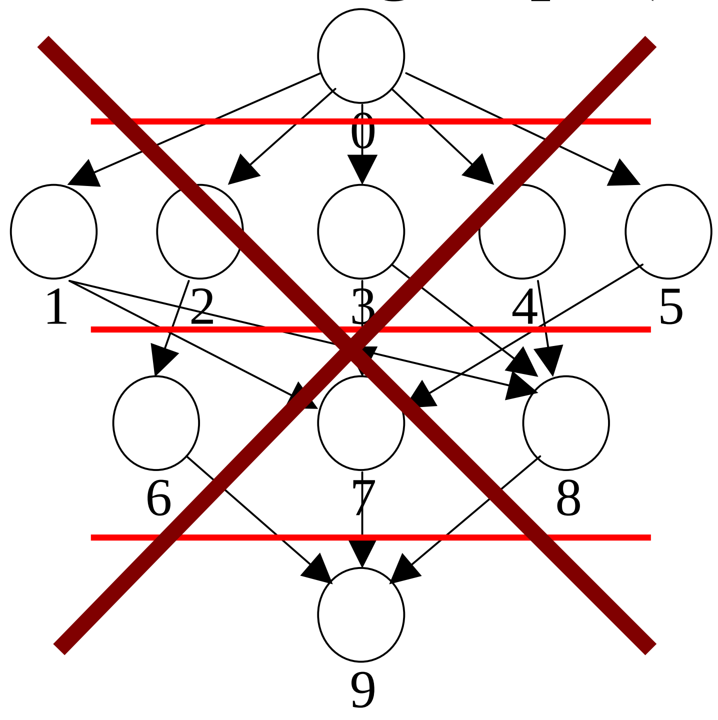


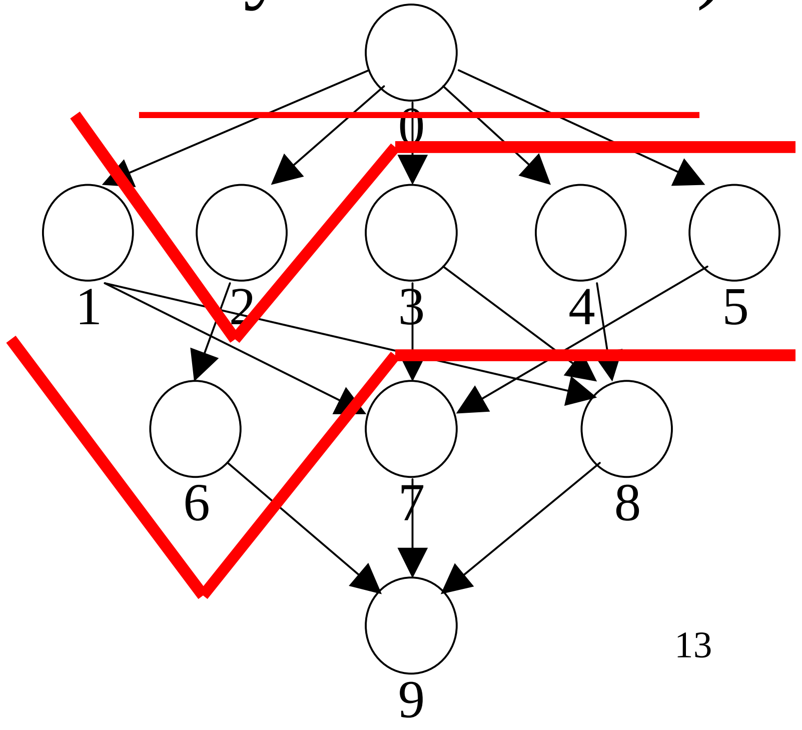
Makespan: **143**

Makespan: **164**

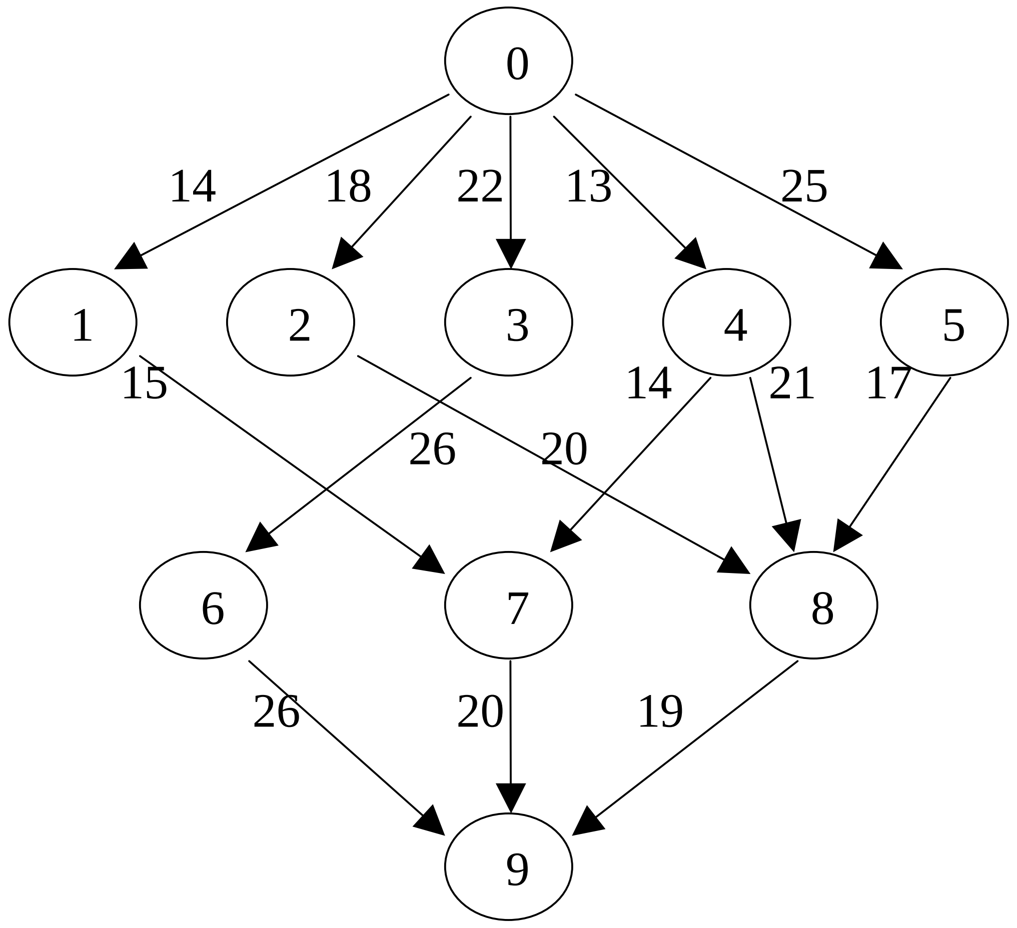
The observation leads to the idea

- Dividing tasks into levels may not be optimal…
- …but strictly ranking tasks may not be optimal either!
- The idea: combine list scheduling with the creation of groups (not necessarily level-based)





HBMCT: An Example



Node	M0	M1	M2
0	17	19	21
1	22	27	23
2	15	15	9
3	4	8	9
4	17	14	20
5	30	27	18
6	17	16	15
7	49	49	46
8	25	22	16
9	23	27	19

Machines	Time for a data unit
M0 – M1	0.9
M1 – M2	1.0
M0 – M2	1.4

HBMCT: An Example

- Phase 1: Rank the nodes

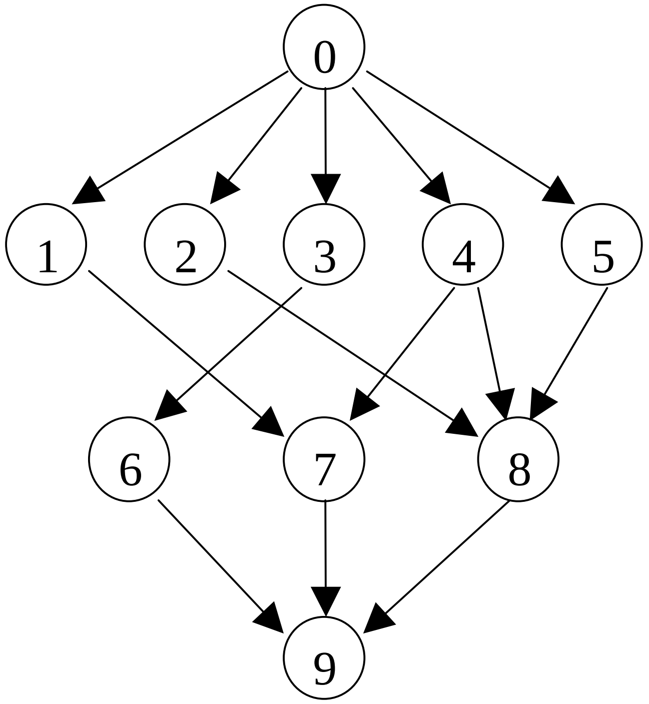
Node	Weight	Rank
0	19	149.93
1	24	120.66
2	13	85.6
3	7	84.13
4	17	112.93
5	25	95.39
6	16	58.06
7	16	85.66
8	21	57.93
9	23	23.0

Mean + Upward Ranking

The order is {0, 1, 4, 5, 7, 2, 3, 6, 8, 9}

HBMCT: An Example

- Phase 1: Rank the nodes
- Phase 2: Create groups of independent tasks

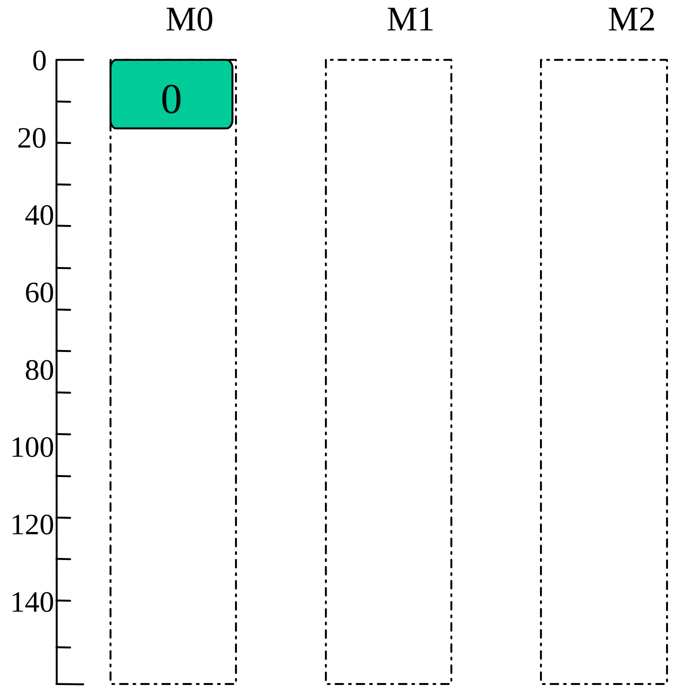


The order is {0, 1, 4, 5, 7, 2, 3, 6, 8, 9}

Group	Tasks
0	{0}
1	{1, 4, 5}
2	{7, 2, 3}
3	{6, 8}
4	{9}

HBMCT: An Example

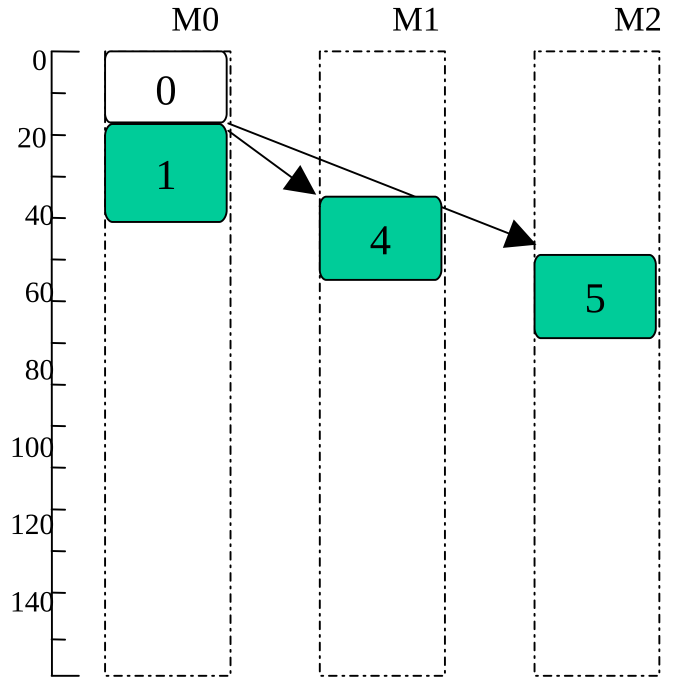
- Phase 3: Schedule Independent Tasks in Group 0
- Balanced Minimum Completion Time (BMCT)



- Initially assign each task in the group to the machine giving the fastest time
- No movement for the entry task

HBMCT: An Example

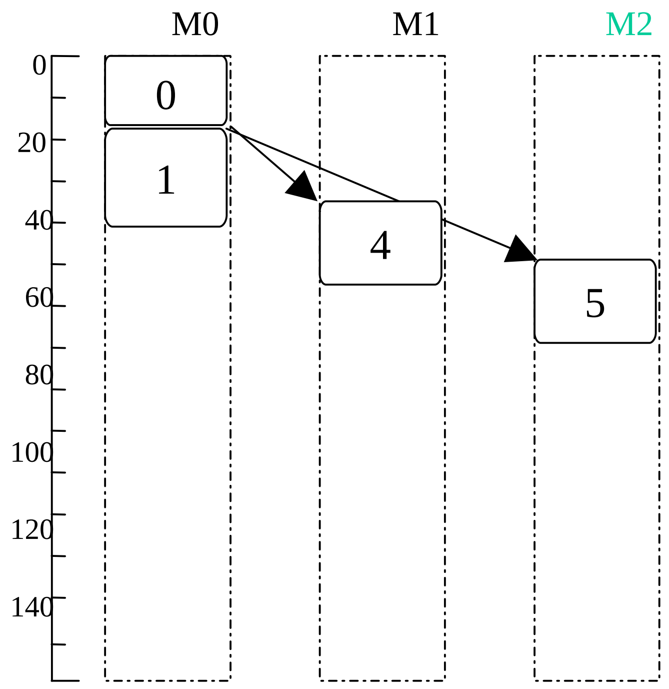
- Phase 3: Schedule Independent Tasks in Group 1



- Initially assign each task in the group to the machine giving the fastest time

HBMCT: An Example

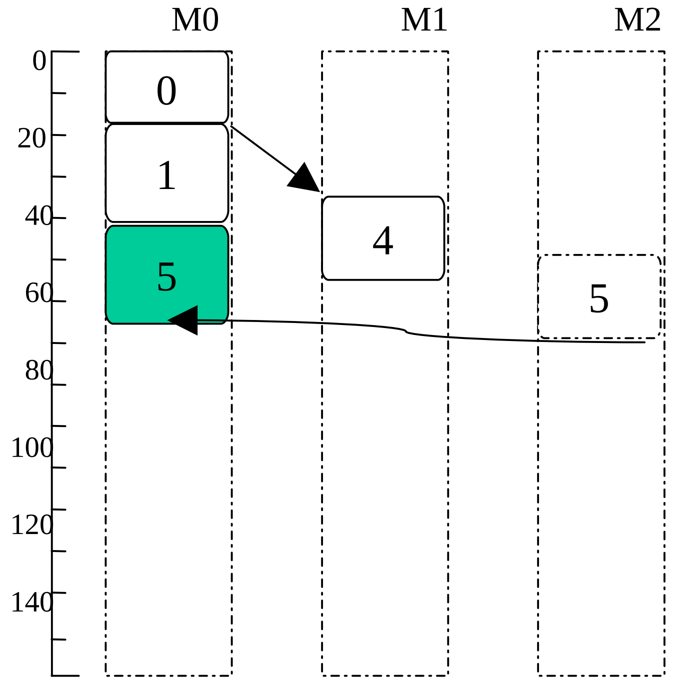
- Phase 3: Schedule Independent Tasks in Group 1



- Initially assign each task in the group to the machine giving the fastest time
- M2 is the machine with the Maximal Finish Time (70)

HBMCT: An Example

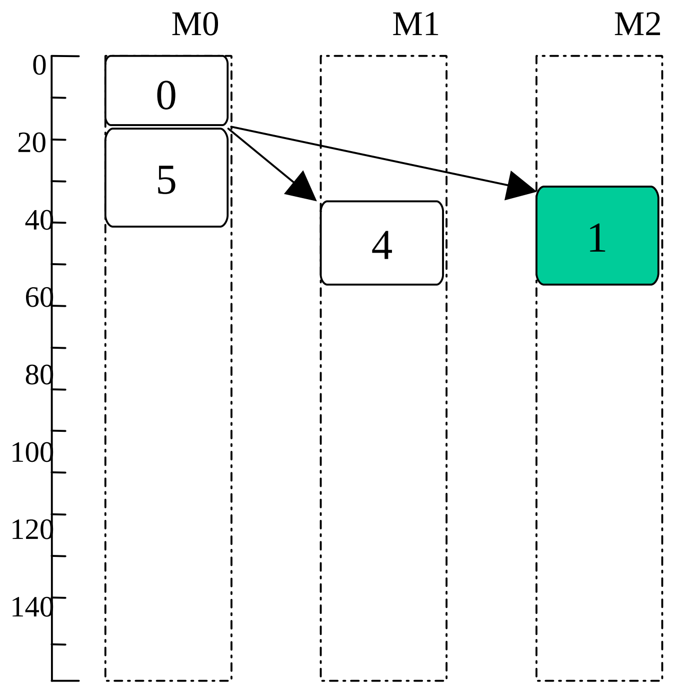
- Phase 3: Schedule Independent Tasks in Group 1



- Task 5 moves to M0 since it can achieve an earlier overall finish time
- Now M0 is the machine with the Maximal Finish Time (69)

HBMCT: An Example

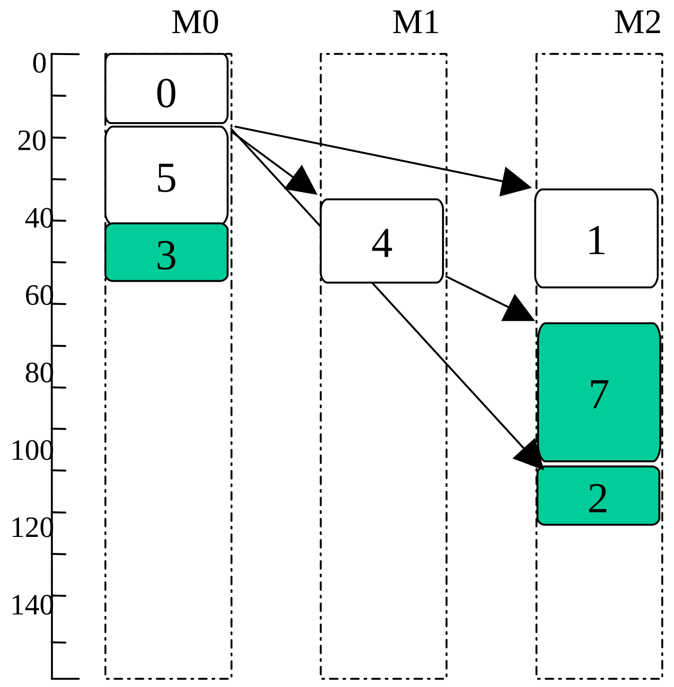
- Phase 3: Schedule Independent Tasks in Group 1



- Task 1 moves to M2 since it can achieve an earlier overall finish time
- Now M2 is the machine with the Maximal Finish Time (59)
- No task can be moved from M2, the movement stops.
- Schedule next group

HBMCT: An Example

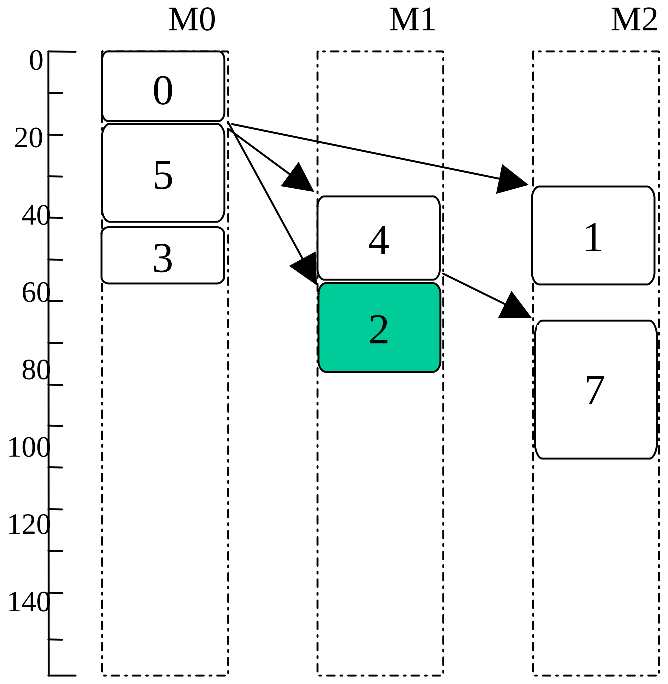
- Phase 3: Schedule Independent Tasks in Group 2



- Initially assign each task in this group to the machine giving the fastest time

HBMCT: An Example

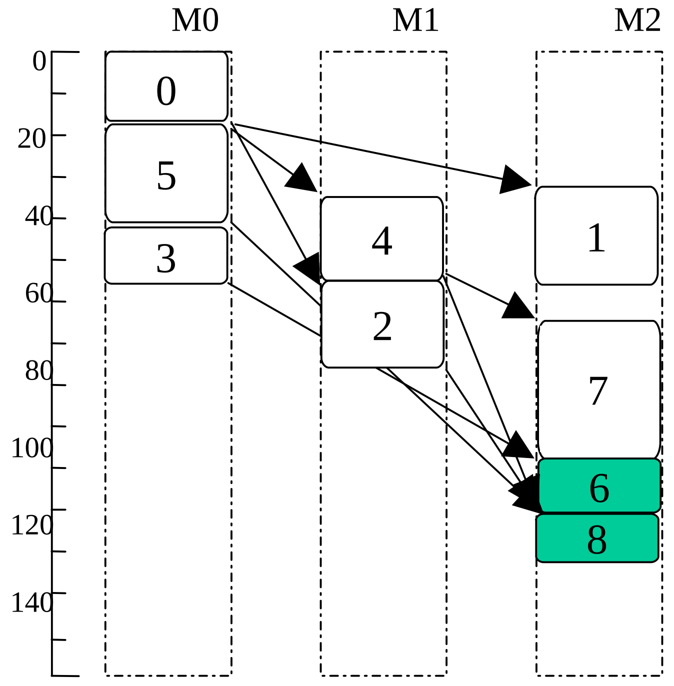
- Phase 3: Schedule Independent Tasks in Group 2



- Task 2 moves to M1 since it can achieve an earlier overall finish time
- M2 is the machine with the Maximal Finish Time
- No movement from M2
- Schedule next group

HBMCT: An Example

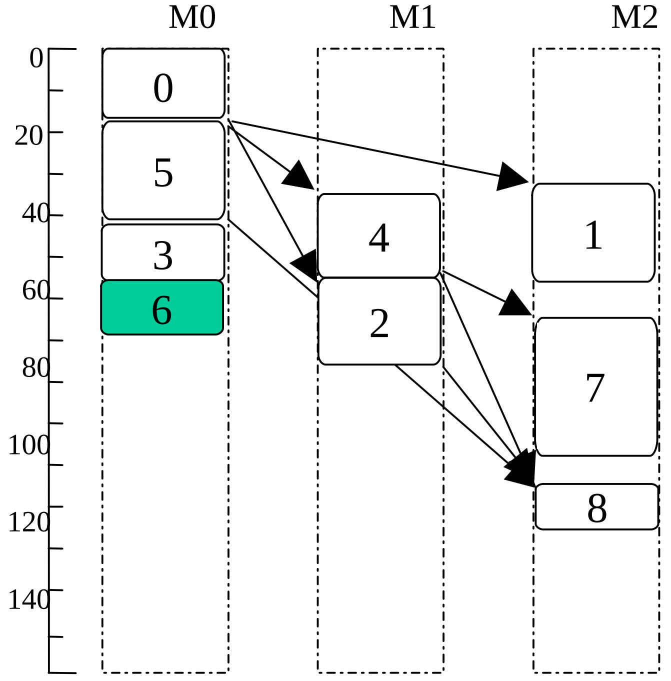
- Phase 3: Schedule Independent Tasks in Group 3



  - Initially assign each task in this group to the machine giving the fastest time

HBMCT: An Example

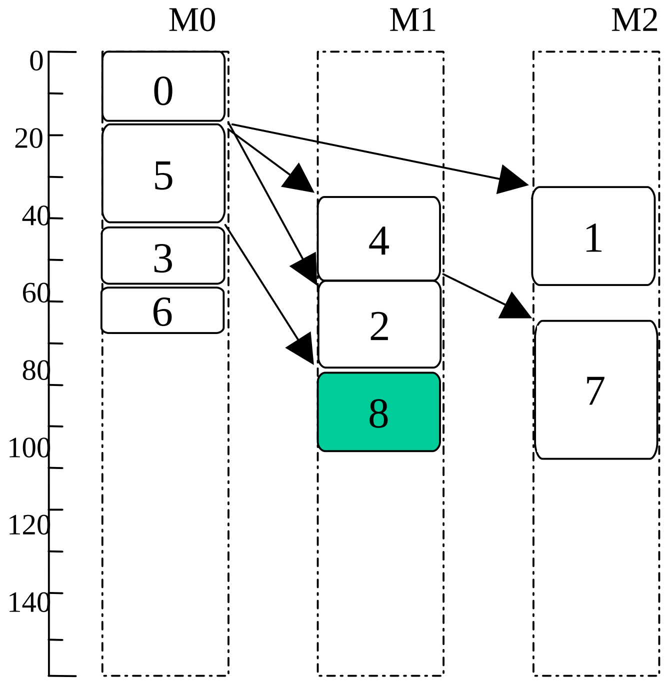
- Phase 3: Schedule Independent Tasks in Group 3



- Task 6 moves to M0 since it can achieve an earlier overall finish time
- M2 is the machine with the Maximal Finish Time

HBMCT: An Example

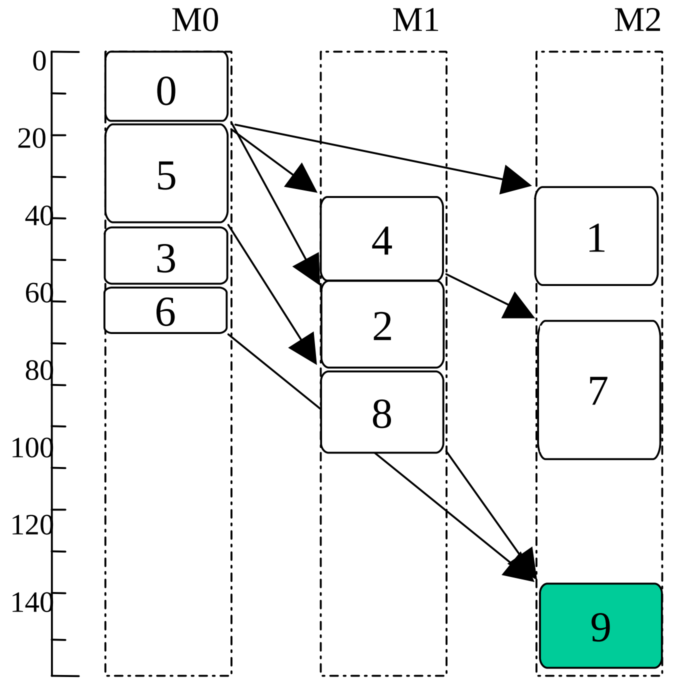
- Phase 3: Schedule Independent Tasks in Group 3



- Task 8 moves to M1 since it can achieve an earlier overall finish time
- M1 is the machine with the Maximal Finish Time
- No movement from M1
- Schedule next group

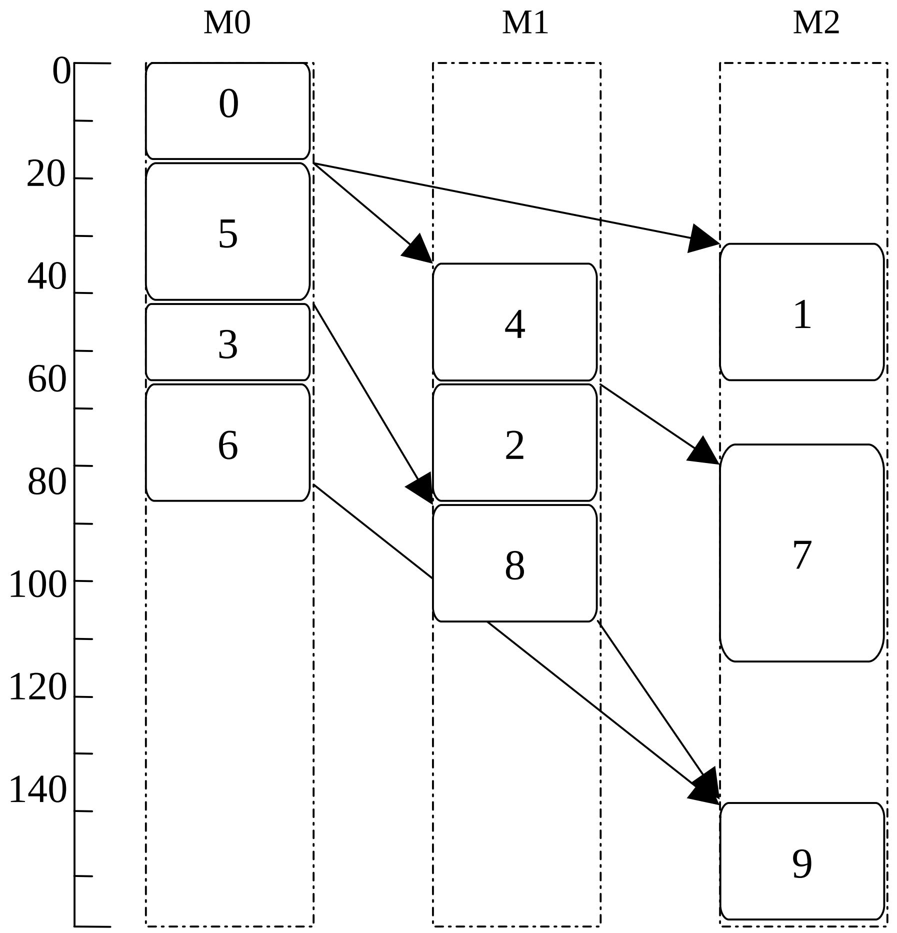
HBMCT: An Example

- Phase 3: Schedule Independent Tasks in Group 4



- Initially assign each task in this group to the machine giving the fastest time
- No movement for the exit task

The Final Schedule



Lessons learned…

- This is not a universally good solution, but it has good behaviour.
- A lot depends on the DAGs being used! (no benchmarks)
- We do not know how much space for improvement exists!
- Small tweaks may have a large impact.
- There is a trade-off between algorithm execution time and performance. Not easy to assess this trade-off.
- There are more than two dozen heuristics in the literature, all of them claiming some advantage!
- Personally, I wouldn’t see much space for another heuristic, unless there is a clear breakthrough…

Outline (cont.)

✓**The ‘simple’ (static) case for heterogeneous environments**

– Uncertainties in execution time

– Scheduling multiple DAGs

– Multi-criteria scheduling

– Dealing with queues

– Adapting the DAG

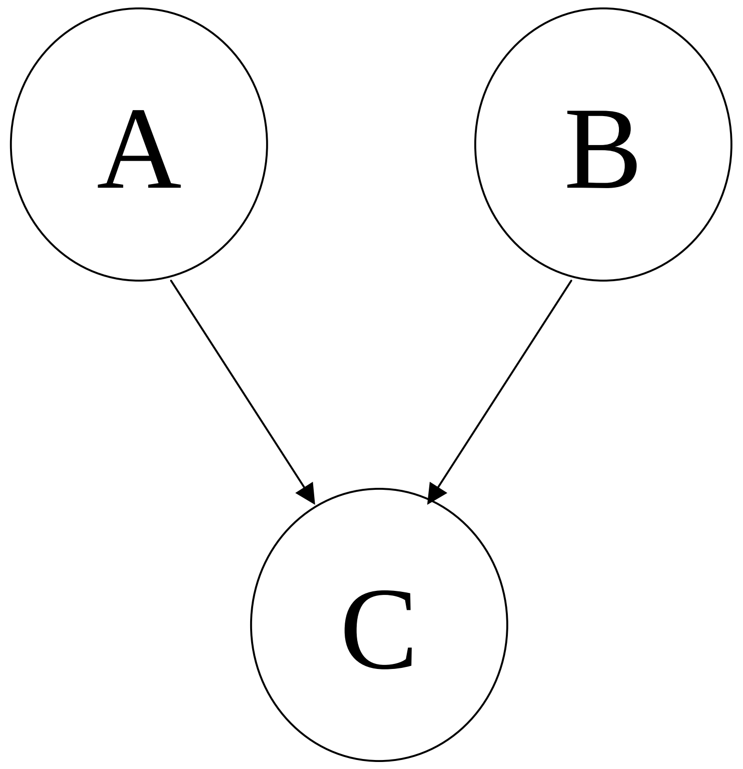
Dealing with uncertainties...

If predicted execution times are not 100% accurate, a possible solution is to **reschedule** as the DAG is being executed!

The problem: when exactly do we reschedule? (rescheduling has a cost, hence we want to avoid rescheduling too often)

The idea: characterize the DAG in terms of the delays it can absorb and monitor these delays at run-time; reschedule only when these delays will result in an overall delay exceeding some amount.

The basic idea: an example



- If task A needs 10 time units to execute and task B needs 2 time units to execute, task B has a slack of 8 time units (assuming A and B start execution at the same time).
- This can be estimated for the whole DAG.
- (highlights the importance of the critical path)

Lessons Learned…

- **Heuristics that perform better statically, perform better under uncertainties.**
- An initially bad schedule cannot improve much with rescheduling

(there is value in studying the problem in an idealized form)

- Some guarantees can be provided if the maximum deviation from the static estimate is known.
- Essentially, the approach suggests rescheduling when there are delays in the critical path
- Leaves space for further work – how do we model uncertainties?

Let’s assume everything is uncertain!

- No task execution times are known.
- Then, one approach is to schedule tasks in such a way that as many tasks as possible are enabled (following the dependences).
- There is some work and some theoretically interesting results are proven.
- Problem: if estimates or the execution times for a few tasks are known a much better schedule can be built using standard approaches.

Scheduling multiple DAGs

- A new mission: **fairness**
  - It is easy to schedule DAGs, say, one after the other; but what if we try to be fair to all DAGs?
- Every DAG will take longer to run, if it shares resources with other DAGs.
  - It will experience a slowdown proportional to the ratio of $\text{makespan}_{\text{own}}/\text{makespan}_{\text{multi}}$.
- For overall fairness we would like to minimize the absolute difference of each DAG’s slowdown from the average slowdown (average slowdown over all DAGs)

Scheduling for Fairness

Key idea: at each step (that is, every time a task is to be scheduled), select the most affected DAG (that is, the DAG with the lowest slowdown value as defined before) to allocate a task from.

What is the most affected DAG at any given point in time?

can be calculated based on the proportion of the DAG already executed.

Lessons Learned

- How do we define fairness?
- What about other objectives?
  - Some DAGs may get higher priority
  - Throughput as opposed to slowdown may be used (for example DAGs may need to produce results at a certain rate)

Surprisingly little in the literature on scheduling multiple DAGs!

Multi-criteria DAG scheduling

- Minimize at the same time:
  - Schedule length (makespan) and (monetary) cost, or
  - Schedule length and energy, or
  - Energy and cost, or
  - All three together!
- Can be combined with meeting deadlines!
- Topical problem!
  - Some work on makespan and money.
  - Paper on energy and cost/makespan to appear next month: I.Pietri et al, *Energy-Constrained Provisioning for Scientific Workflow Ensembles*. 3rd IEEE International Conference on Cloud and Green Computing (CGC'13), 2013.

Multi-criteria DAG scheduling

- The idea:
  - Start from a good solution for one objective and try to meet the other(s).
- Our first work on budget-deadline scheduling:
  - Start with a good schedule.
  - Repeatedly select tasks that can move to cheaper resources by checking for minimum values of:

$$(t\_new - t\_old) / (c\_old - c\_new)$$

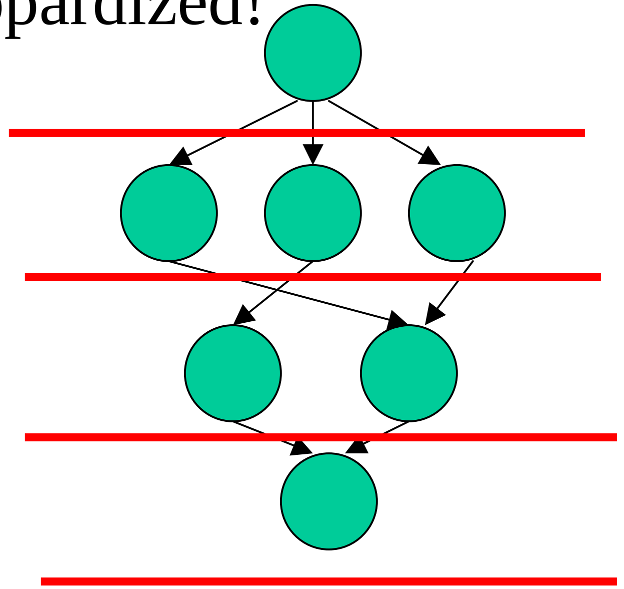
Surprisingly good!

Issues with multi-criteria scheduling

- Topical (cloud computing)
- Energy constraints become a big issue. Little work yet on the interplay between cost/energy/makespan.
- Preliminary results suggest that good resource utilization may minimize energy consumed too.
- Studying the problem and the possible solutions can help us understand how to (i) make good use of resources, (ii) derive good pricing models.
- Large variety of DAGs:
  - affects result
  - Solutions are sensitive to small tweaks
- Data transfer is important
  - Need to understand the platform

Understanding the platforms

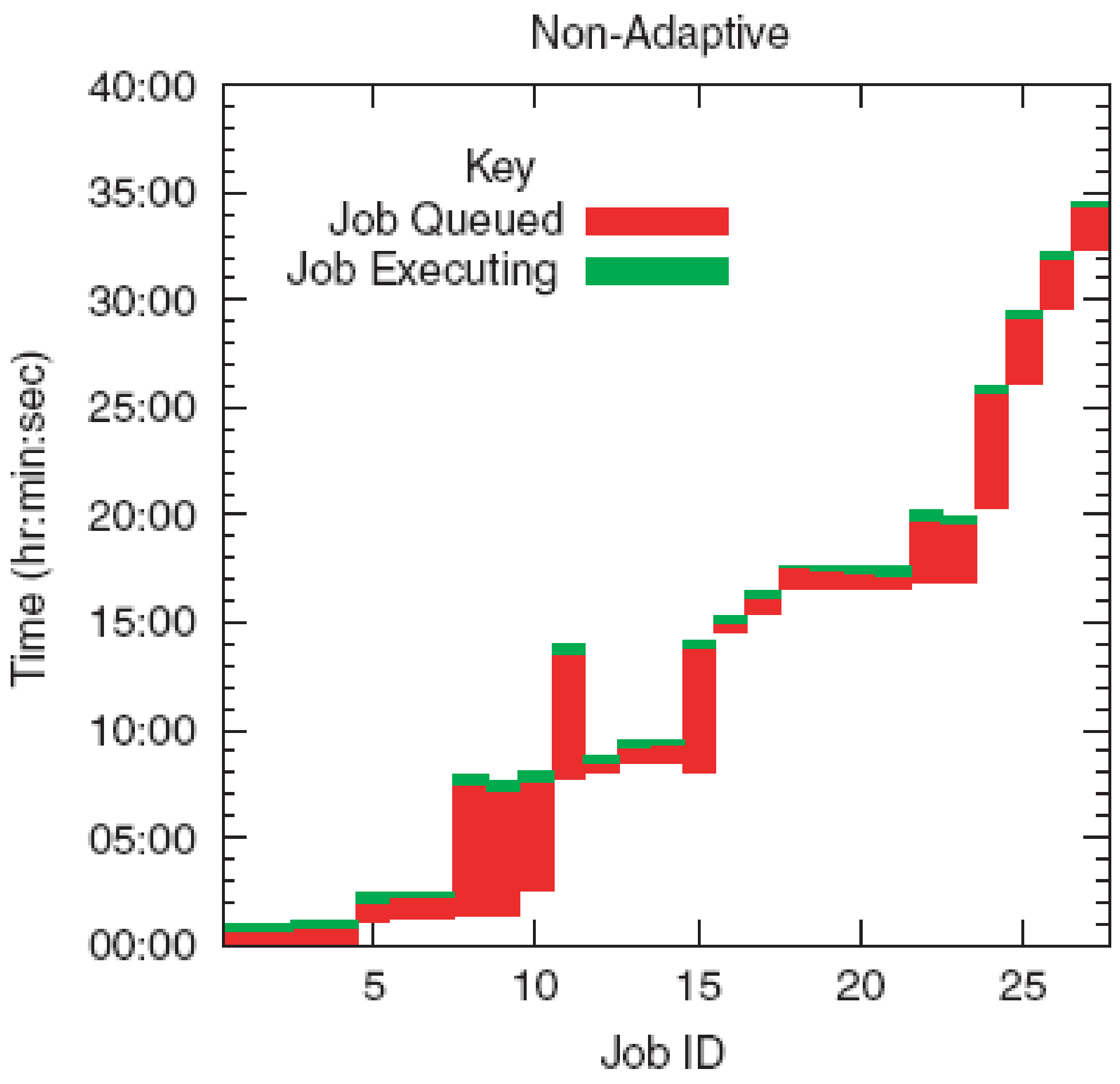
- Queue-based platforms create additional problems as they have different objectives. The carefully crafted schedule of the DAG may be jeopardized!



The execution model fails to hide the latencies resulting from the length of job queues: <u>these determine the execution time of the workflows</u>.

- It is not clear if parallelism will be fully exploited; e.g., if the three tasks above that can be executed in parallel are submitted to 3 different queues of different length, there is no guarantee that they will execute in parallel – **<u>job queues rule</u>**!

Queues may dominate!



Bypassing queues

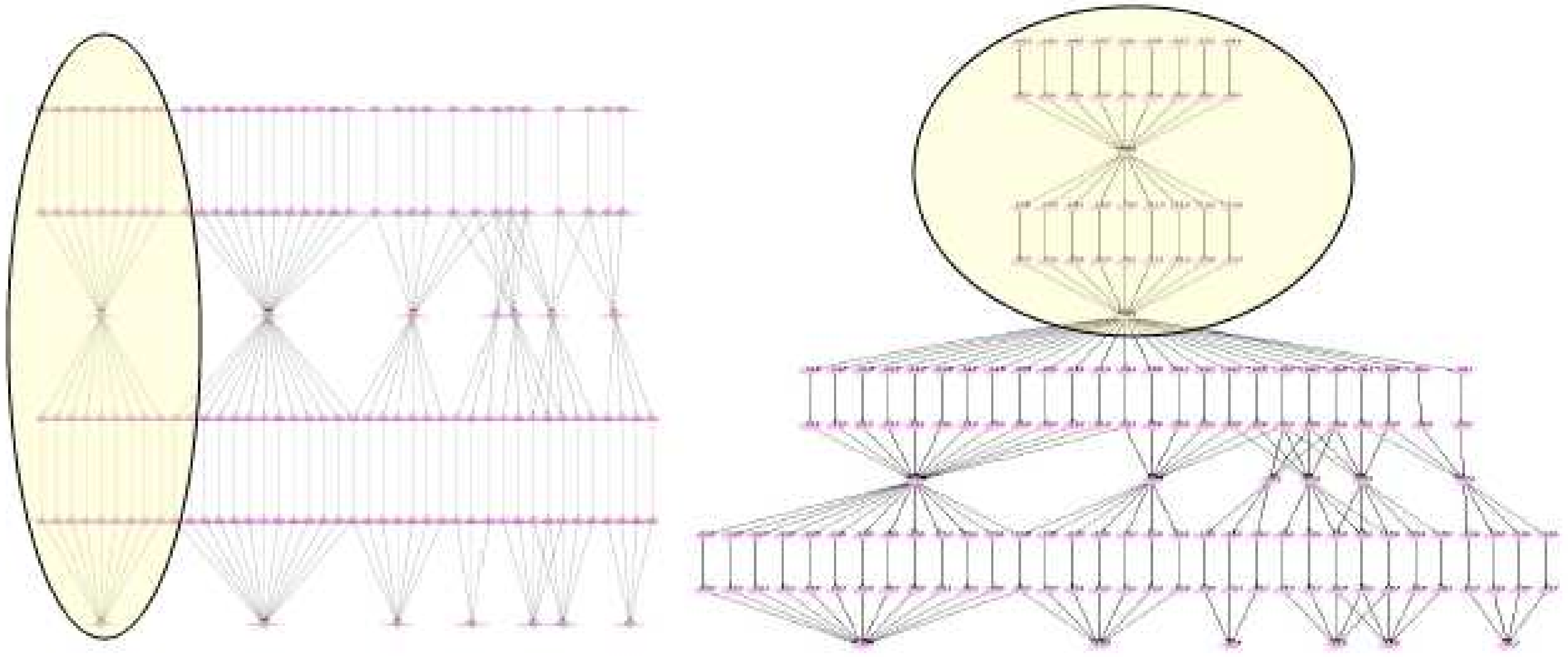
- Get rid of queues!
  - Advance reservation
- Adaptive execution (adjust to the length of the queues)
- In such cases, how to create an initial schedule of the DAG becomes less important. The latencies of the queues dominate!

A final aspect of the problem

- Adapt the DAG!
- Executing a large DAG in parallel may increase pressure on memory/storage.
- It may be useful to change from a wide and short DAG to a long and thin. It will take longer to run but it requires less resources (disk/memory)
- Requires some merging of tasks. Paper to appear: W. Chen et al. *Balanced task clustering in scientific workflows*. 9th IEEE International Conference on e-Science (next month).

An Example

(“Optimizing Workflow Data Footprint”, Scientific Programming)



Summary

DAG scheduling in large-scale systems

- Static case (only if breakthrough)
- Modelling uncertainties in execution time
  - Interplay with performance prediction, good work exists.
- Scheduling multiple DAGs
  - Not much is available.
- Multi-criteria DAG scheduling
  - Work needed to understand the trade-off between time, money, energy, resource utilization, etc… Topical (vis-à-vis Cloud)
- Adapting the DAG (and adaptive DAG scheduling)
  - Lots of potential, requires some understanding of the DAG

Lots of parameters affect the outcome: small changes may have a large impact.

An important problem for both theory and practice: keeps reappearing in different forms.

Thank you!

Work carried out in collaboration with a number of people (without whom it wouldn’t have been possible):

- Henan Zhao
- Ilia Pietri
- Ewa Deelman
- Kevin Lee
- Viktor Yarmolenko
- Wei Zheng

and many more!

And some promotion…

- **Summer School on Cloud Computing**
  - **September 23rd - 24th**, Karlsruhe, Germany
  - Speakers: Ewa Deelman, Omer Rana, Markus Bauer, Josef Spillner
  - Lectures and hands-on tasks
  - Funded through the SUCRE EU FP7 project

  - http://www.sucreproject.eu/?q=summer-school
  - http://service-summer.ksri.kit.edu/