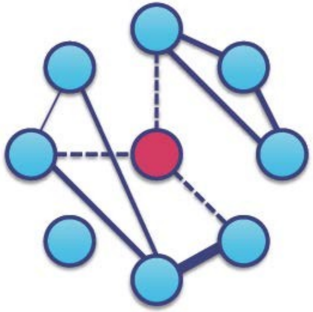
Twenty Years of Scheduling Research—Models, Methods, and Conclusions

Inauguration Seminar Alexandru Iosup

Dick Epema
Distributed Systems Group

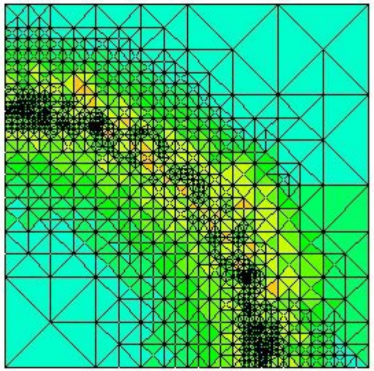
11 June 2018



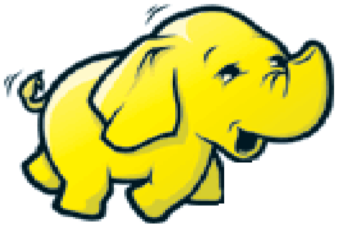
TU Delft

Four scheduling cases

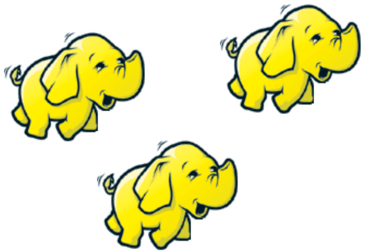
1



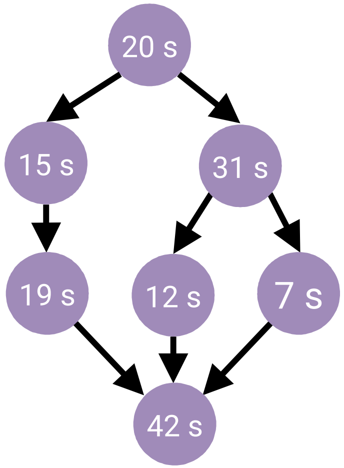
3



2



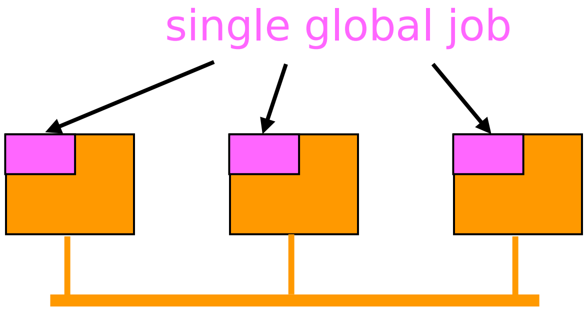
4



TU Delft

Case 1: co-allocation (1)

- Jobs may use resources in multiple sites: **co-allocation**
- **Reason**:
  - to benefit from distributed resources (e.g., processors, data, visualization)
- **Resource possession in different sites** can be:
  - simultaneous (e.g., parallel applications)
  - coordinated (e.g., workflows)
- **With co-allocation**:
  - more difficult **resource-discovery** process
  - need to **coordinate allocations** by autonomous resource managers



TUDelft

Co-allocation (2): slowdown

- Co-allocated applications are **less efficient** due to the relatively **slow wide-area communications**
- **Slowdown of a job**:

$$\frac{\text{execution time on multicluster}}{\text{execution time on single cluster}} \quad \textbf{(>1 usually)}$$

- Processor co-allocation is a **trade-off** between
  - **faster access to more capacity**
  - **shorter execution times**

TU Delft

Co-allocation (3): scheduling policies

- **Placement policies** dictate where the components of a job go
- **Examples of placement policies**:
  1. **Load-aware**: Worst Fit (**WF**)
     (balance load in clusters)
  2. **Input-file-location-aware**: Close-to-Files (**CF**)
     (reduce file-transfer times)
  3. **Communication-aware**: Cluster Minimization (**CM**)
     (reduce number of wide-area messages)

Co-allocation (4): simulations/analysis

Conclusions:

- There are fundamental problems to be derived from practical scheduling problems in distributed systems that have a general significance
- Combination of simulations and mathematical analysis gives more complete results and better understanding

Anca Bucur and Dick Epema, *HPDC 2003* and *IEEE TPDS 2007.*

TU Delft

Co-allocation (5): experiments on the DAS3

average execution time (s)

average execution time (s)

Conclusions:

- It may be very difficult to match simulations and experiments
- It is very difficult to do multiple experiments under the same conditions
- It is very difficult to identify (the influence of) “polluting elements”

Ozan Sonmez, Hashim Mohamed, and Dick Epema, *IEEE TPDS* 2010.

KOALA (1/2): a co-allocating grid scheduler

- **Original goals:**
  1. **processor co-allocation**: parallel applications
  2. **data co-allocation**: job affinity based on data locations
  3. **load sharing**: in the absence of co-allocation

  while being transparent for local schedulers

- **Additional goals:**
  – **research vehicle** for scheduling and RM research
  – support for (other) popular application types
- **KOALA has been deployed** on the DAS2 – DAS5 since september 2005
- Later versions: KOALA-C (clouds) and KOALA-F (frameworks)

TU Delft

Hashim Mohamed and Dick Epema, *CCPE* 2006.

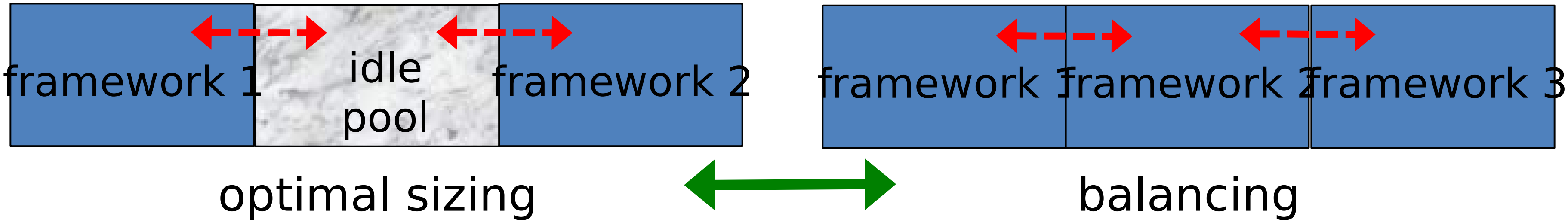
KOALA (2/2): the runners

Conclusions:

- Very beneficial to have a deployed research vehicle (DAS + KOALA) for
  - driving research
  - teaching distributed systems programming
  - doing experimentation
  - visibility
- Very time-consuming to make a scheduler “user proof” (never did a release)

Case 2: scheduling frameworks

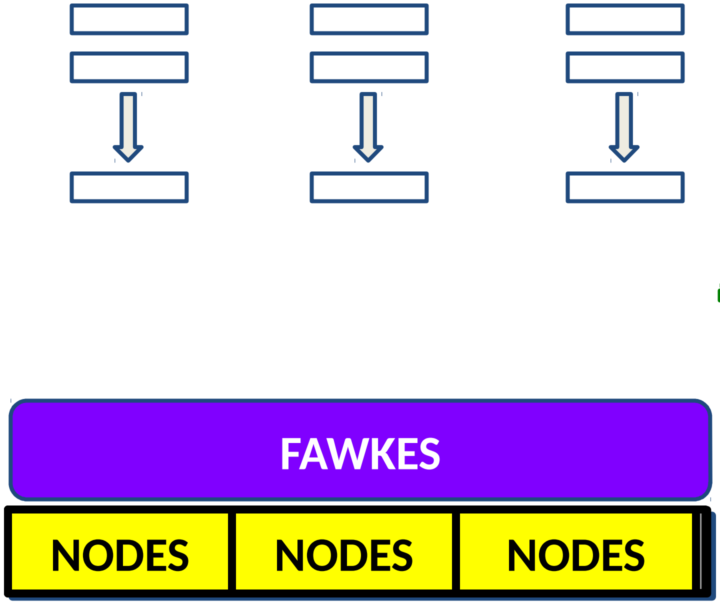
- **Reduce**
  - **scheduling overhead** of centralized scheduler
  - **complexity** of centralized scheduler
- **Provide isolation among frameworks**
- **Two models:**



TU Delft

Balancing allocations with FAWKES

Two-level scheduling architecture



Job submissions

Frameworks

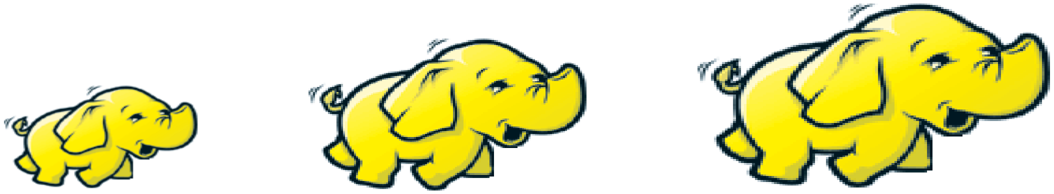
Resource manager

Infrastructure

Bogdan Ghiţ, Nezih Yiğitbaši, Alexandru Iosup, and Dick Epema, *ACM Sigmetrics* 2014.

TUDelft

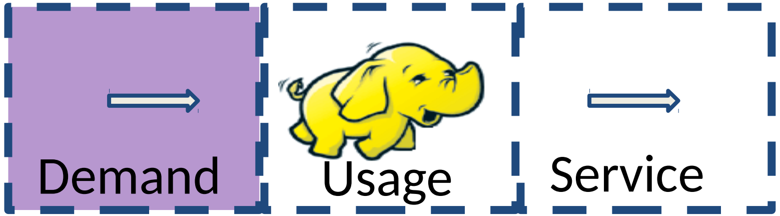
FAWKES in a nutshell

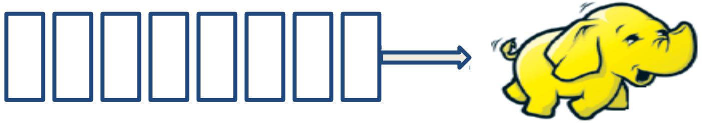


$$w_1 < w_2 < w_3$$

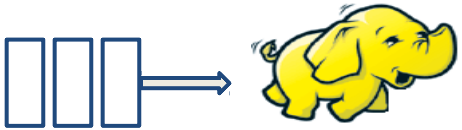
- Gives “fair” shares of the resources to frameworks
- Shares proportional to **dynamic weights**
- Updates weights when:
    - frameworks arrive or leave
    - framework states change

How to differentiate frameworks? (1/3)





versus

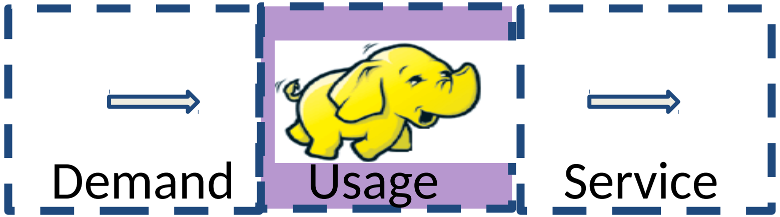


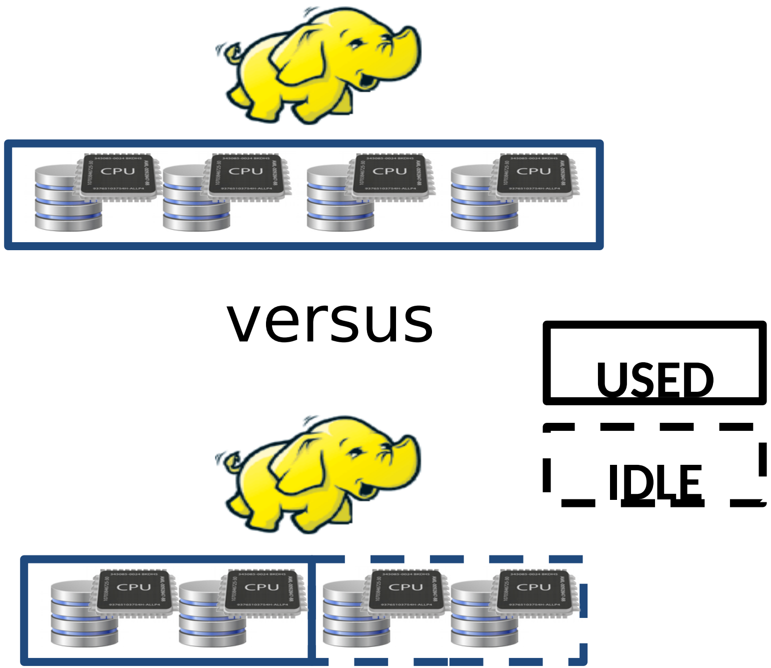
By **demand** – 3 policies:

- Job Demand (JD)
- Data Demand (DD)
- Task Demand (TD)

TUDelft

How to differentiate frameworks? (2/3)

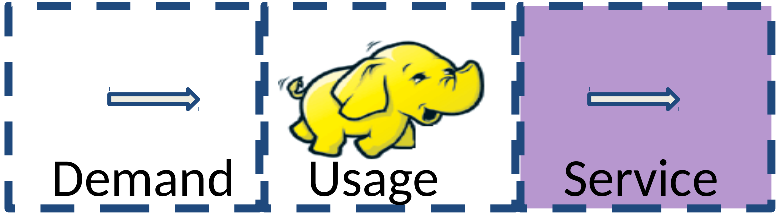


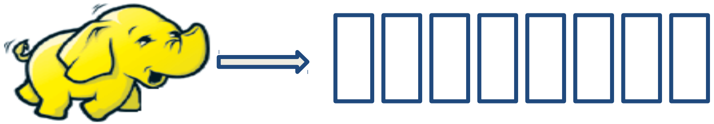


By **usage** – 3 policies:

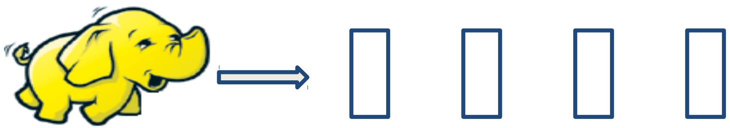
- Processor Usage (PU)
- Disk Usage (DU)
- Resource Usage (RU)

How to differentiate frameworks? (3/3)





versus



By **service** – 3 policies:

- Job Slowdown (JS)
- Job Throughput (JT)
- Task Throughput (TT)

Performance of FAWKES

um

Nodes	
Framev	
Minimu	
Datase	
Jobs su	

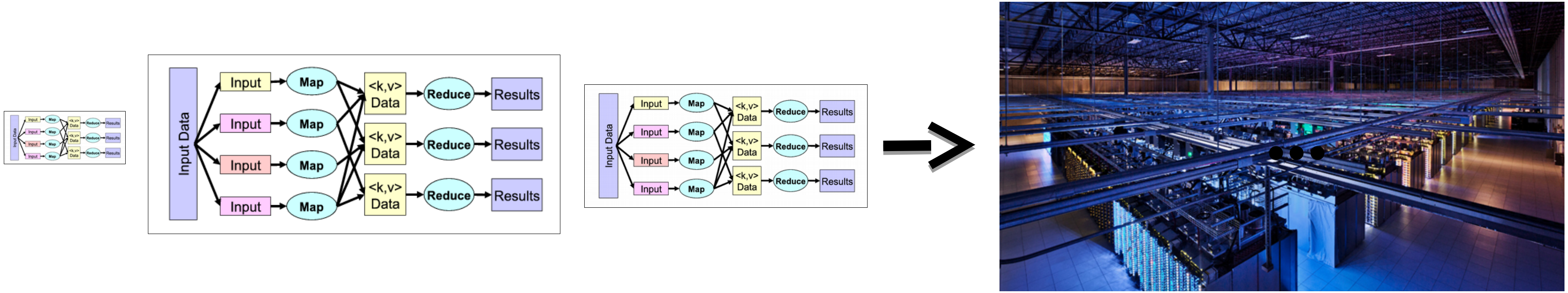
Conclusions:

- Studying queuing models is very beneficial for students for
  - a better understanding of practical performance problems
  - better problem formulation
  - better execution of research in scheduling in systems
- Simulations of scheduling frameworks is still required
- Experimentation with Spark and KOALA-F/Mesos are a nightmare

JS – Job Slowdown

TUDelft

Case 3: reducing slowdown variability in MapReduce



Bogdan Ghiţ and Dick Epema, *MASCOTS* 2015, *CCGrid* 2016.

TUDelft

Problem: “slowdown” due to big customers

20 seconds

3 minutes

$$\text{slowdown} = \frac{20 + 180}{20} = 10$$

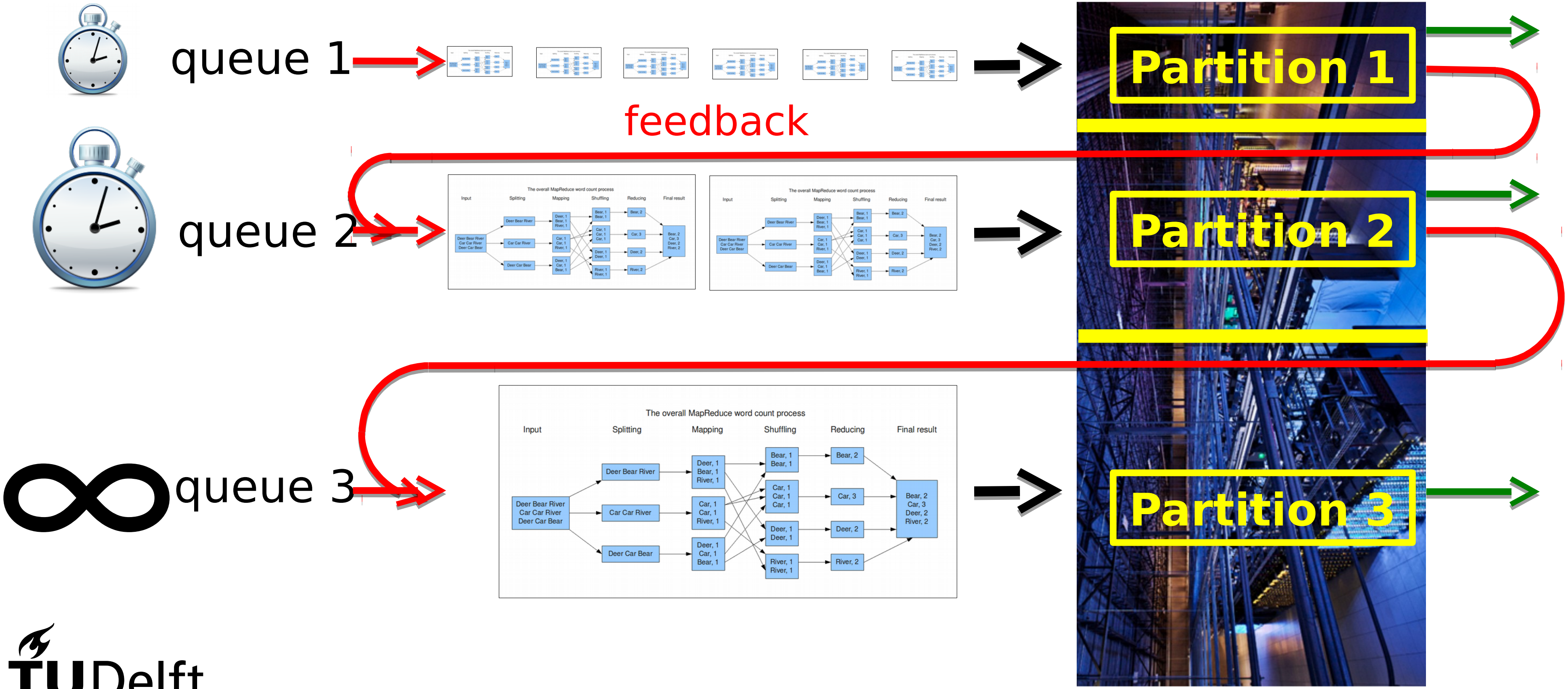
TU Delft

Solution: express lanes

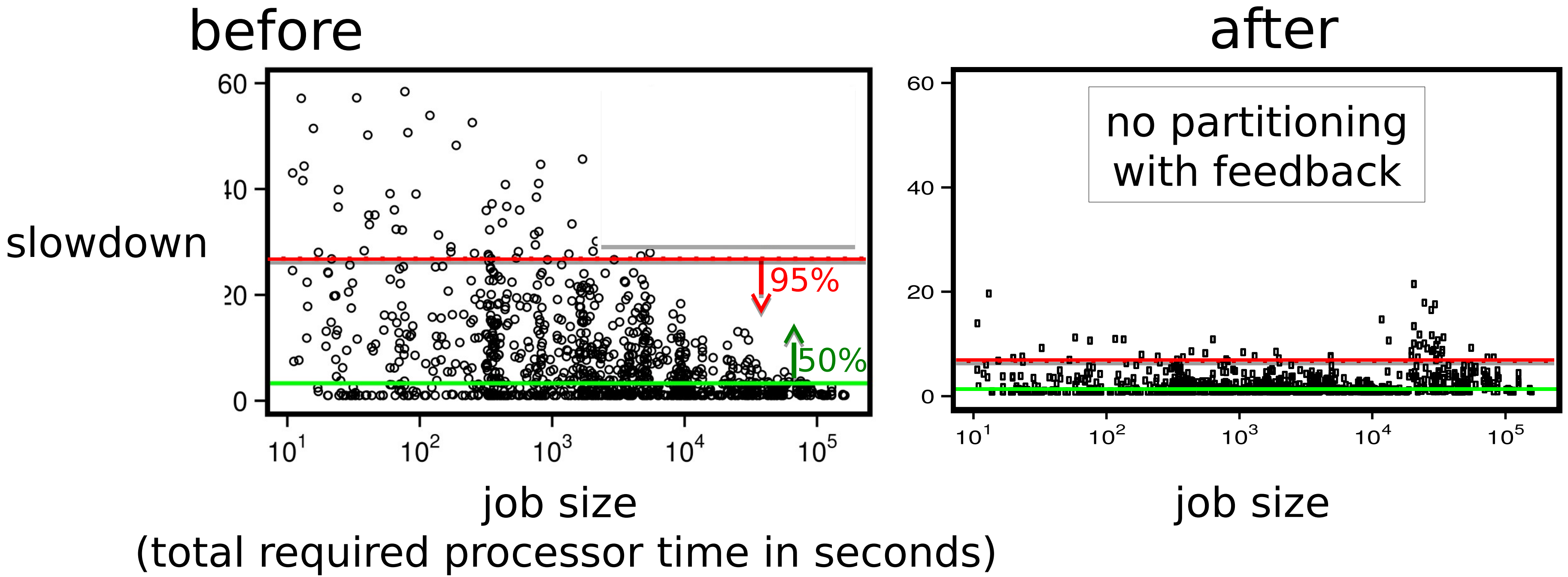
Size-based scheduling
Make jobs in a single queue homogeneous

TUDelft

Queues in datacenters

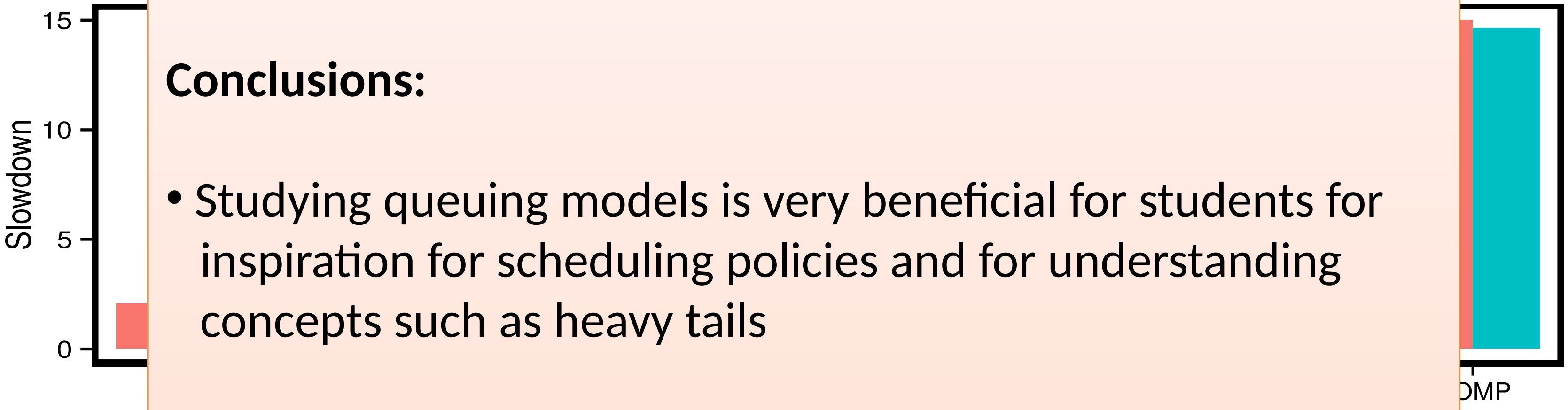


What is the improvement?



TUDelft

Simulator validation



Conclusions:

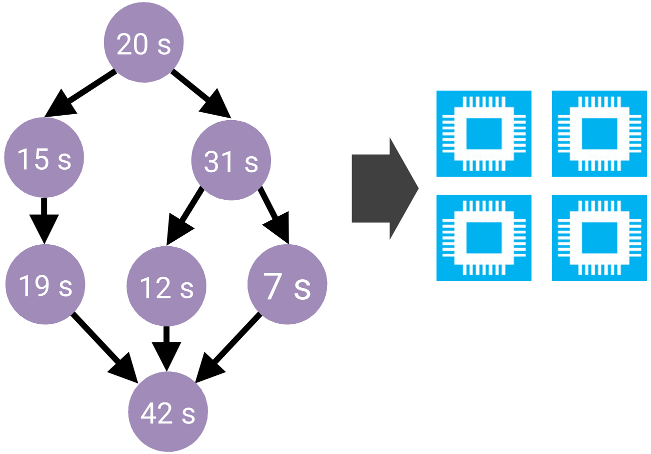
- Studying queuing models is very beneficial for students for inspiration for scheduling policies and for understanding concepts such as heavy tails

- Fundamental differences with original mathematical analysis of size-based scheduling (other job model, work-conserving pre-emption, partitioning)

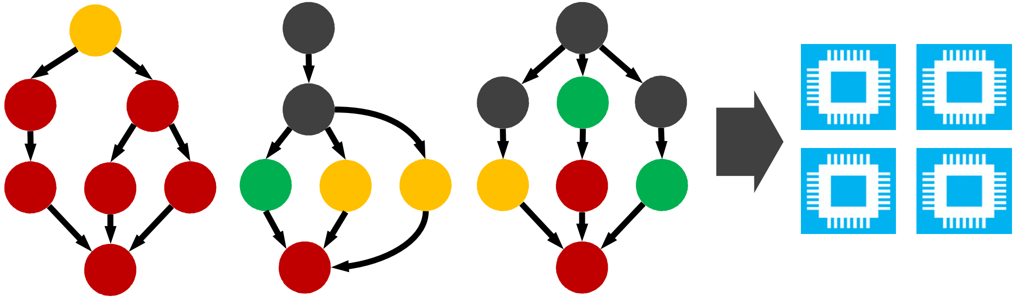
Less than 1% error between SIM and DAS.

Case 4: workloads of workflows

Previous work



Our work



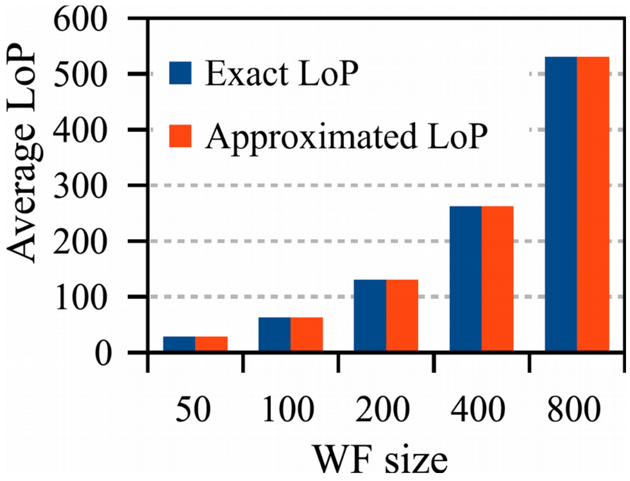
- Finished tasks
- Running tasks
- Eligible tasks
- Non-eligible tasks

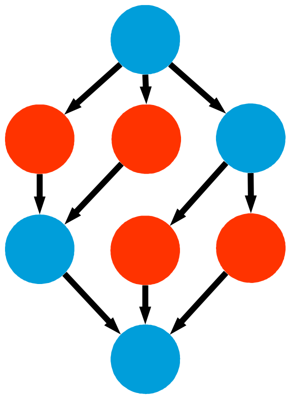
Alexey Ilyushkin, Bogdan Ghiţ, and Dick Epema, *CCGrid* 2015 and 2018.

Scheduling policies (1/2)

- **Greedy backfilling** versus some form of **reservation**
- For reservation, use Level of Parallelism (LoP)
- LoP is compute-intensive, use approximation

Quality of LoP Approximation for Montage workflow





LoP=4

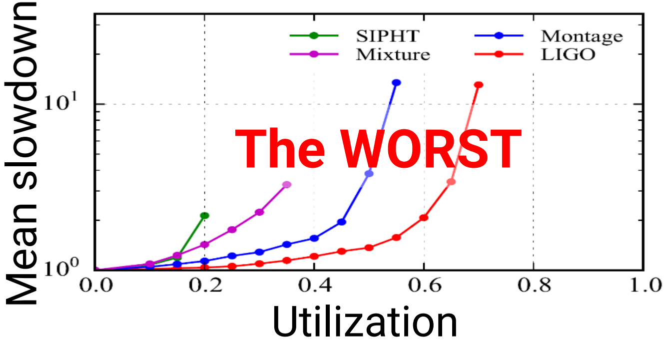
Scheduling policies (2/2)

1. Strict reservation: use LoP
2. Scaled LoP: use f x LoP, $0 \leq f \leq 1$
3. Consider future eligible sets
4. Greedy backfilling

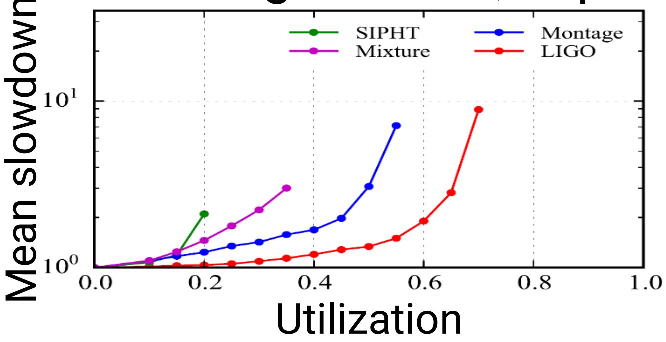
TU Delft

Simulation results

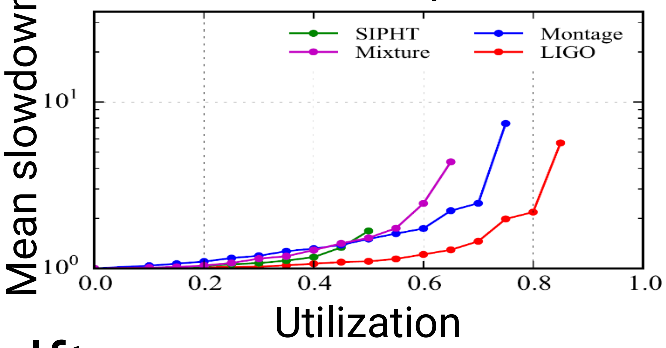
Strict Reservation



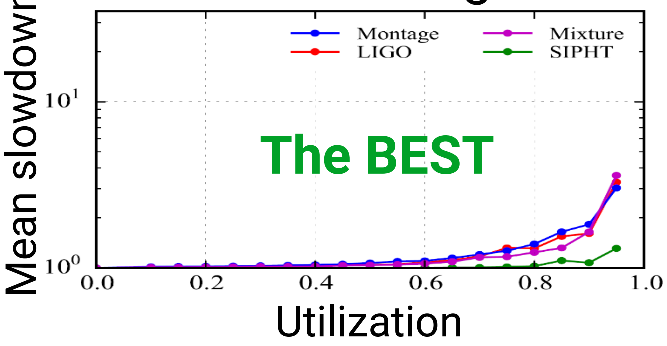
Future Eligible Sets, depth 2



Scaled LoP $f = 0.2$



Backfilling



What is the use of task runtime estimates?

- Suppose task runtimes are **known with some error**

Conclusions:

- Fills a gap in queueing models
- For these fundamental questions, no experiments are needed

is beneficial

- the sensitivity to inaccuracy of estimates increases at higher utilizations
- plan-based gives very much overhead and does not perform well

TUDelft

Acknowledgments

- Anca Bucur (co-allocation, 2004)
- Lipu Fei (KOALA-C)
- Bogdan Ghiţ (frameworks, MapReduce, 2017)
- Bart Grundeken (cycle scavenging)
- Alexey Ilyushkin (workflows, 201x)
- Alex Iosup (KOALA-C, frameworks, simulator, 2009)
- Aleksandra Kuzmanovska (KOALA-F, frameworks, 201x)
- Wouter Lammers (hardening KOALA)
- Hashim Mohamed (design KOALA, 2007)
- Ozan Sonmez (application types, 2010)
- Nezih Yiğitbaši (MapReduce, 2012)

TU Delft