UNIVERSITY OF CAMBRIDGE

Cloud Computing

Large-scale Resource Management II

Eva Kalyvianaki
ek264@cam.ac.uk

Contents

Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing,

by Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, and Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou, *in OSDI 2014*

Slides and material from the OSDI paper and its presentation from:

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/boutin

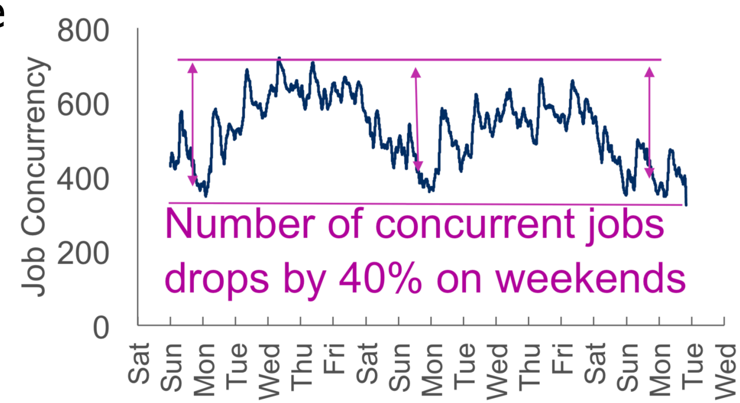
Background

- Cloud-scale jobs
  - Jobs are written using SCOPE, a SQL-like high-level scripting language, augmented with user-defined processing logic.
  - Job’s are represented by DAGs
  - Tasks are the basic unit of computation
  - Tasks are grouped in stages
  - Execution is driven by the scheduler

Scheduling at Cloud Scale

minimize job latency while maximizing cluster utilization

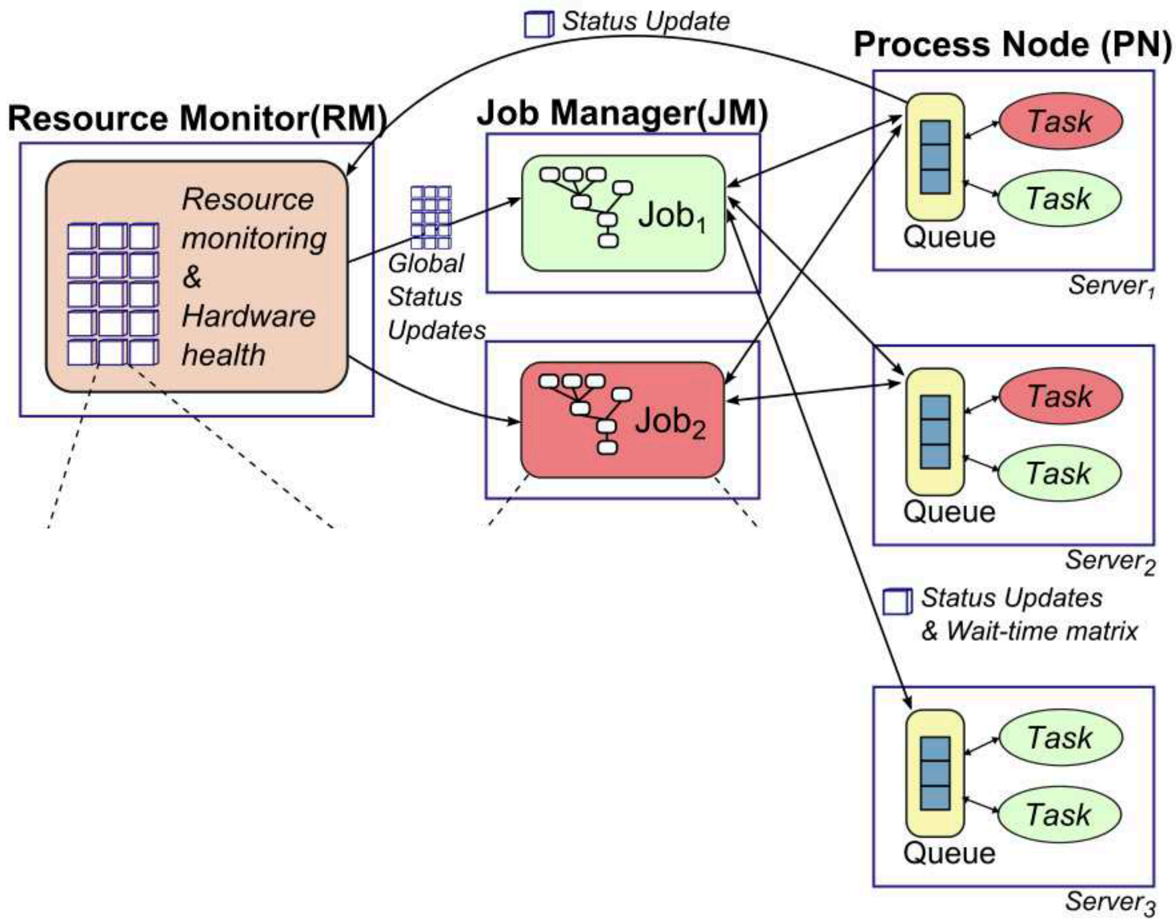
- Challenges:
  - Scale
    - Jobs process gigabytes to petabytes of data, 100K scheduling req/sec
    - Clusters run 170K tasks in parallel and each has over 20K servers
  - Heterogeneous workload
    - Tasks run secs to hours
    - Can be IO or CPU bound
    - Require 100MB to 10GB of memory
    - Short tasks are scheduling sensitive
    - Long tasks are locality sensitive
  - Maximize utilization
    - Workload fluctuates



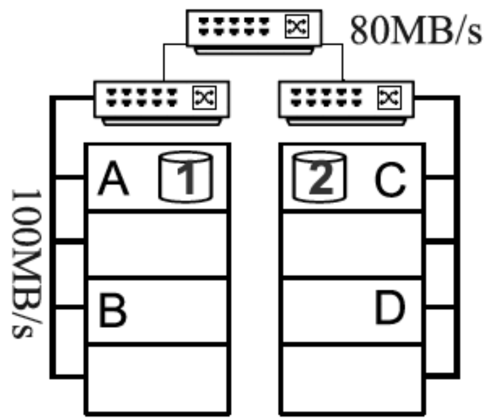
Apollo Overview

1. Distributed and coordinated architecture
2. Estimation-based scheduling
3. Conflict Resolution
4. Opportunistic Scheduling

Distributed and coordinated architecture



Different factors for optimization



(a) Server map.

Server	Wait	I/O	Wait+I/O
A	0s	63.13s	63.13s
B	0s	63.5s	63.5s
C	40s	32.50s	72.50s
D	5s	51.25s	56.25s

(b) Scheduling alternatives.

Figure 4: A task scheduling example.

Estimation-based scheduling

Estimated task completion time

$$E_{succ} = I + W + R$$

$$C = P_{succ} * E_{succ} + K_{fail} * (1 - P_{succ}) * E_{succ}$$

E: Estimated task completion time

I: initialization time: fetching files for the task

W: wait time: a lookup in the wait-time matrix of the target server

R: Runtime: both I/O and CPU time

Conflicts Resolution

1. Apollo defers the correction of conflicts
   → (vs Omega where conflicts are handled at scheduling time)

2. Re-evaluates prior decisions

3. Triggers a duplicate if the decision is not optimal with up-to-date data

Opportunistic Scheduling

- Maximize utilization
  - Use the remaining capacity
  - Dispatch more that the resource allocation
  - Tasks only consume idle resources
  - Tasks can be preempted or terminated
  - Tasks can be upgraded

  - Limit capacity share of each job
  - Random queueing

Evaluation

1. Apollo runs on Microsoft production clusters with over 20K servers each
2. It runs 170K tasks in parallel
3. Tracks 14M pending tasks

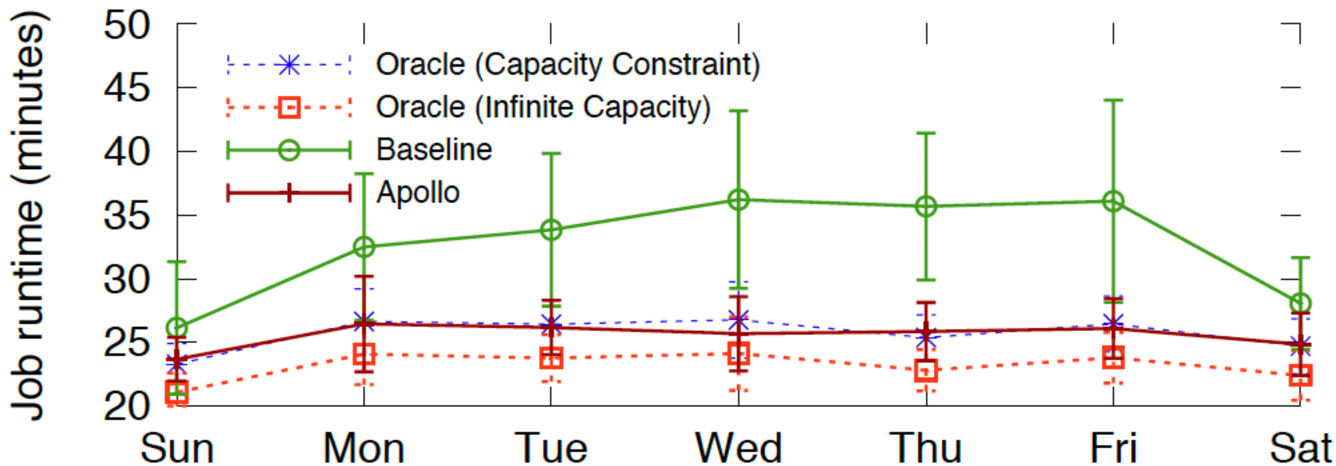
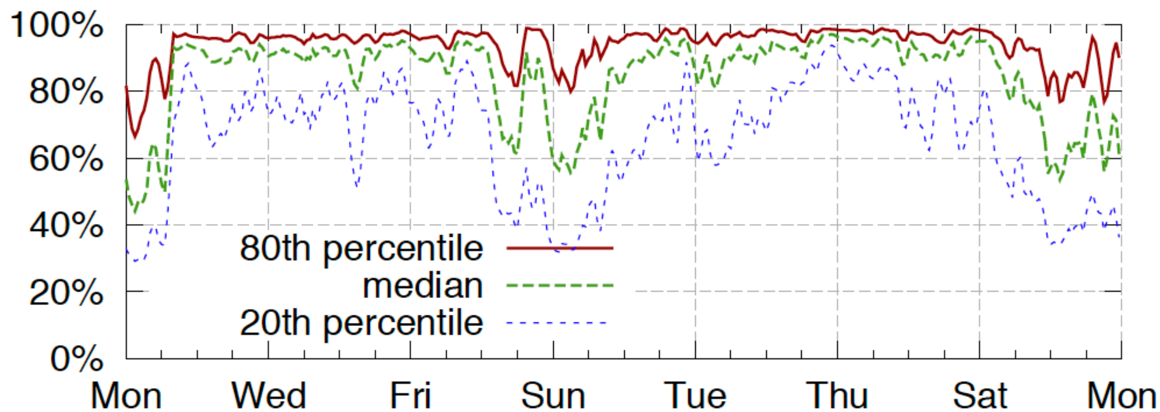
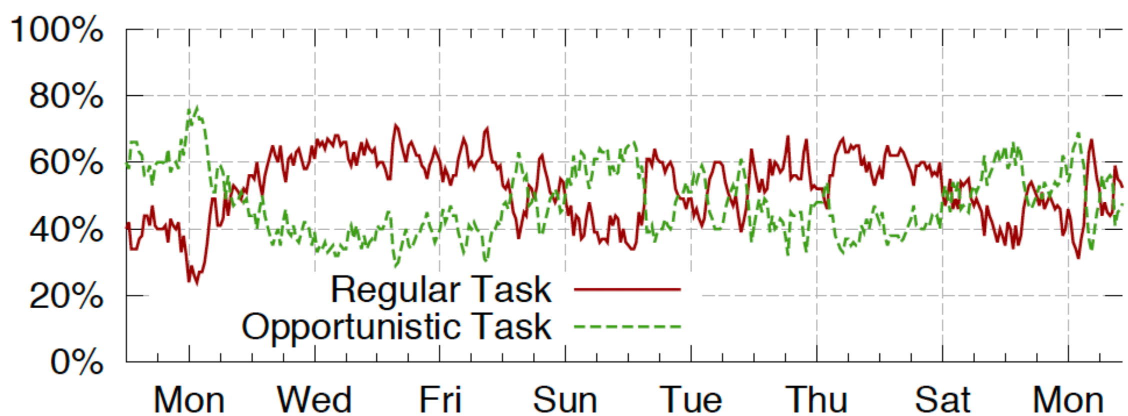


Figure 9: Job latencies with different schedulers.

Apollo's Resource Efficiency



(b) CPU utilization.



(c) CPU time breakdown: regular and opportunistic task.

Figure 7: Apollo in production.

Conclusions

1. Loosely Coordinated Distributed Architecture
2. Deployed to clusters with over 20K servers
3. High Quality Scheduling
4. Mininizes task completion time
5. Consistent performance
6. Maximizes resource utilization
7. Opportunistic scheduling
8. 90% median CPU utilization