Microsoft

Apollo

Scalable and Coordinated Scheduler for Cloud-Scale Computing

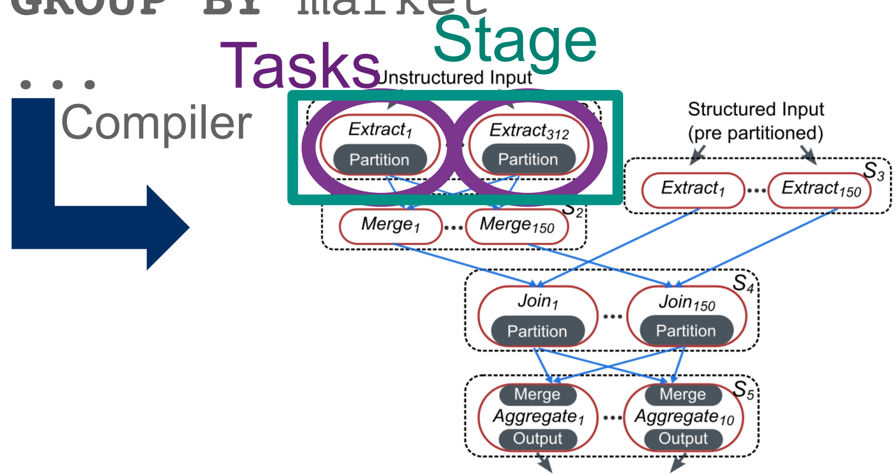
Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, **Microsoft**
Zhengping Qian, Ming Wu, Lidong Zhou, **Microsoft Research**

Cloud Scale Job

- High level SQL-Like language
- The job query plan is represented as a DAG
- **Tasks** are the basic unit of computation
- Tasks are grouped in **Stages**
- Execution is driven by a scheduler

Job sample: SCOPE (VLDBJ, 2012)

```
SELECT
AVG(DateTime.Parse(latency))
AS E2ELatency,
market
FROM
QueryLatencies
GROUP BY market
...
```



Scheduling at Cloud Scale

Minimize job latency while
maximizing cluster utilization

Challenges

1. Scale
2. Heterogeneous workload
3. Maximize utilization

Challenging Scale

Jobs process **gigabytes to petabytes** of data

and issue peaks of **100,000 scheduling requests/seconds**

Clusters run up to **170,000 tasks in parallel**

and each contains **over 20,000 servers**

Challenge: How to make optimal scheduling decisions at full production scale

Heterogeneous Load

Tasks runs from **seconds to hours**

Tasks can be **IO bound** or **CPU bound**

Tasks can require from 100MB to more than 10GB of memory

Short tasks are sensitive to **scheduling latency**

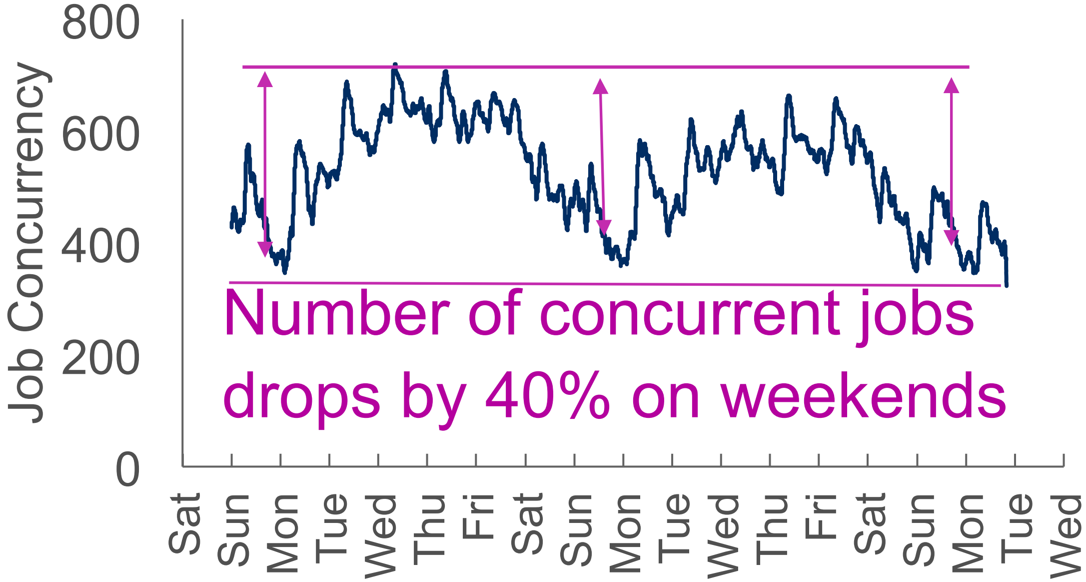
Long IO bound tasks are sensitive to **locality**

Challenge:

Make **optimal scheduling decisions** for a **complex workload**

Maximizing Utilization

We need to effectively use resources and
maintain performance guarantees
but the workload constantly fluctuates



Challenge: **Maximize utilization** while maintaining performance guarantees with a **dynamic workload**

Apollo

Background
Challenges
Overview
— Distributed and coordinated architecture
— Estimation-based scheduling
— Conflict resolution
— Opportunistic scheduling
Evaluation at scale
Related work
Conclusion

Distributed and Coordinated

To scale, Apollo adopts a distributed and coordinated architecture

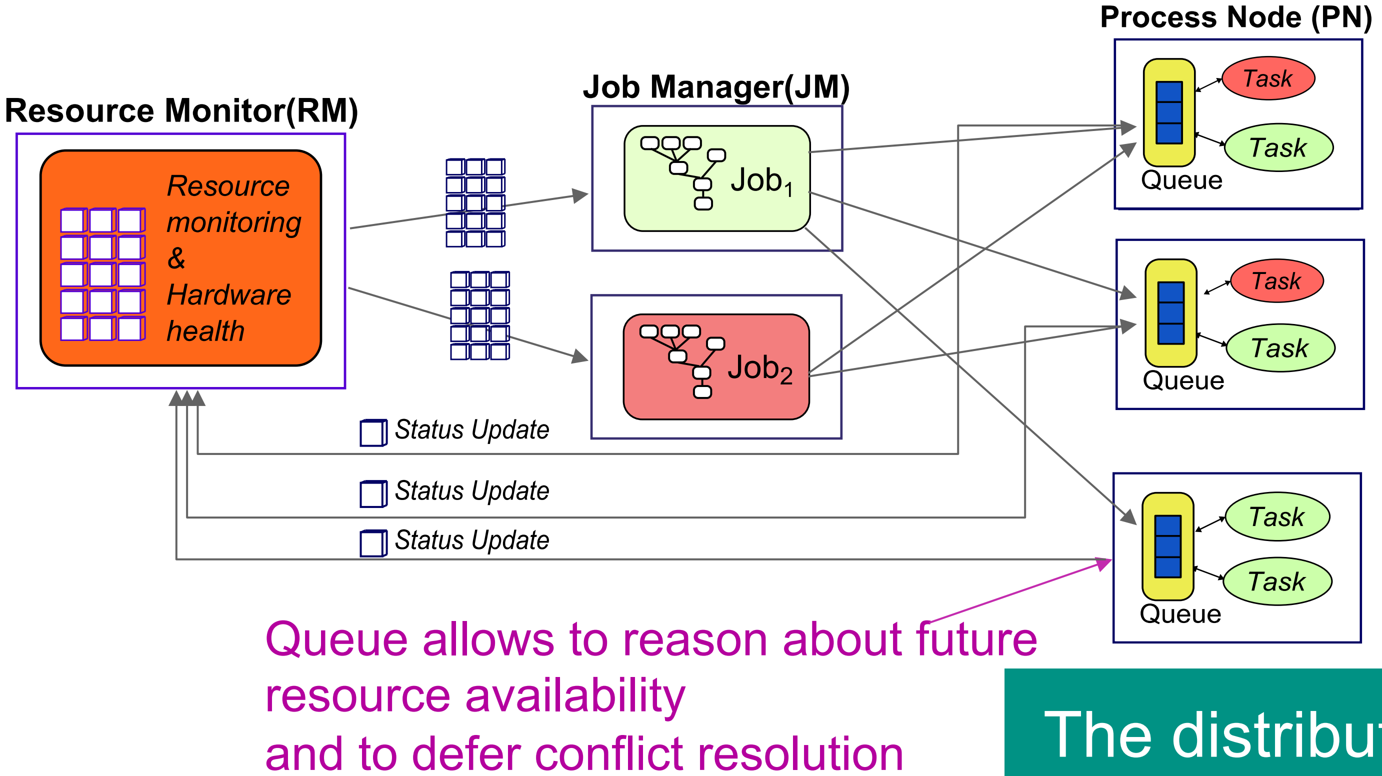
There is one scheduler per job

each making high quality decisions independently,

informed by global information

.

Distributed and Coordinated



The distributed architectures scales by allowing schedulers to **make independent decisions** with **global coordination**

Representing Load

The server load representation must

- Be hardware independent
- Be lightweight
- Supports heterogeneous workload

Apollo represents the load

- Using a wait-time matrix
- It represents the **expected wait time** to obtain resource of a certain size

→ *Memory*

↓ *CPU*

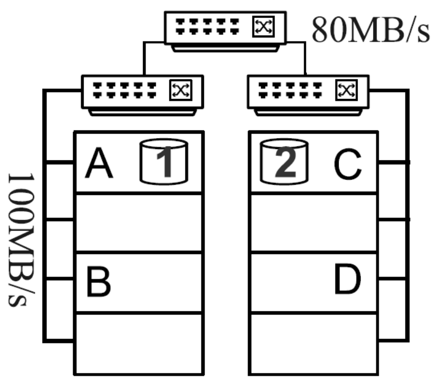
	4GB	8GB	12GB	16GB
2 core	0	10	10	16
4 core	0	10	10	16
6 core	5	10	10	16
8 core	10	15	15	25

Wait-time matrix (seconds)

Status Update

The wait time matrix allows to **reason about future resource availability**

Optimizing for various factors



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	

To optimize performance, the scheduler needs to simultaneously consider many conflicting factors

Estimation-Based Scheduling

Apollo minimizes the estimated task completion time

$$E = I + W + R$$

E:	Estimated task completion time
I:	Initialization time
W:	Wait time
R:	Runtime (including locality impact)

Apollo **minimize** the **task completion time** by considering relevant factors holistically

Correcting Conflicts

Cluster is dynamic

- Schedulers can have conflicts
- Apollo defers the correction of conflict

Apollo re-evaluates prior decisions

- Triggers a duplicate if the decision isn't optimal with up to date information

The correction mechanisms allows Apollo to **handle cluster dynamics**

Opportunistic scheduling

Maximize utilization

— Use the remaining capacity
— Dispatch more than the resource allocation
— Tasks only consume otherwise idle resources
— Tasks can be preempted or terminated
— Tasks can be upgraded

Opportunistic scheduling allows Apollo to **maximize utilization**

Additional techniques

— Limit capacity share of each job
— Random queuing

Apollo

Background

Challenges

Overview

— Distributed and coordinated architecture

— Estimation-based scheduling

— Conflict resolution

— Opportunistic scheduling

Evaluation at scale

Related work

Conclusion

How well does Apollo scale?

Apollo runs on Microsoft production clusters

— Incrementally rolled out from September to December 2013

— Each containing over **20,000 servers**

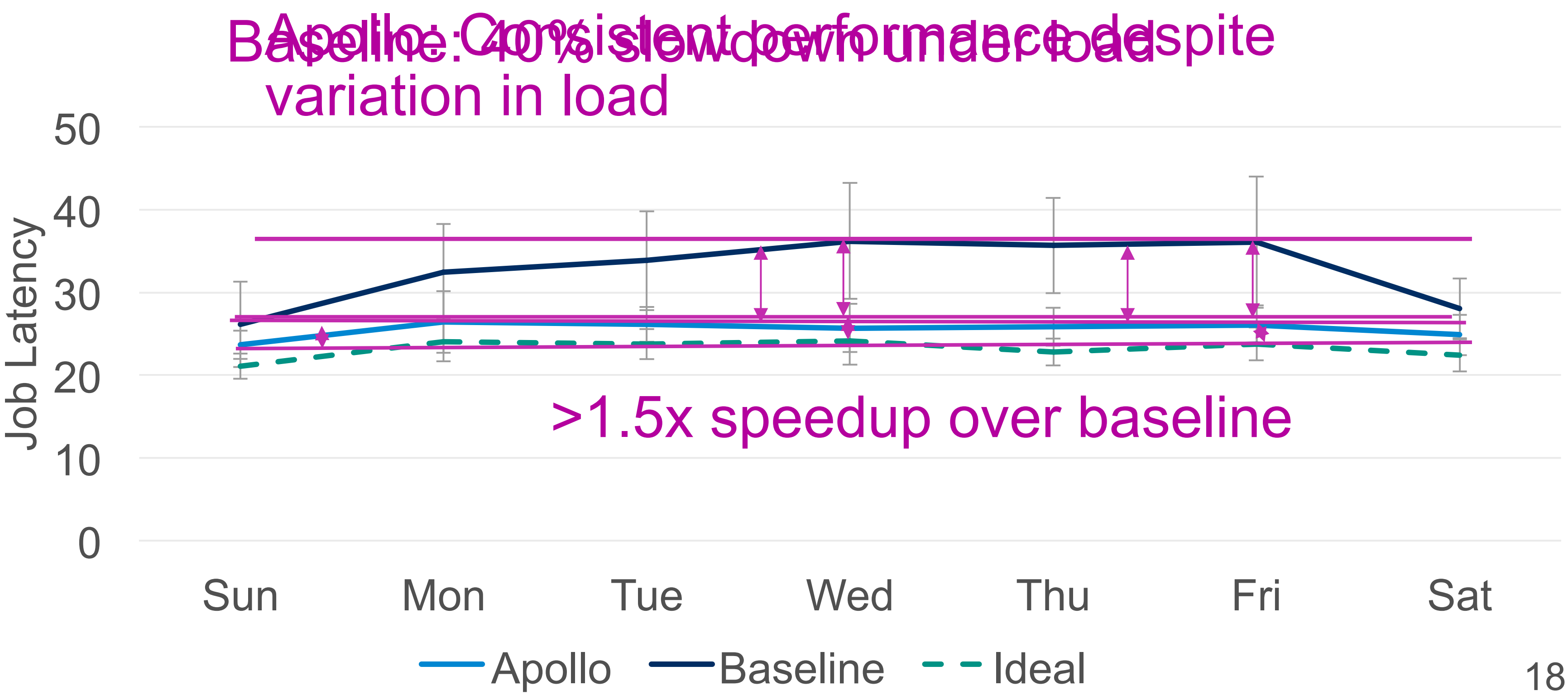
In one cluster, Apollo

— Runs **170,000 tasks** in parallel

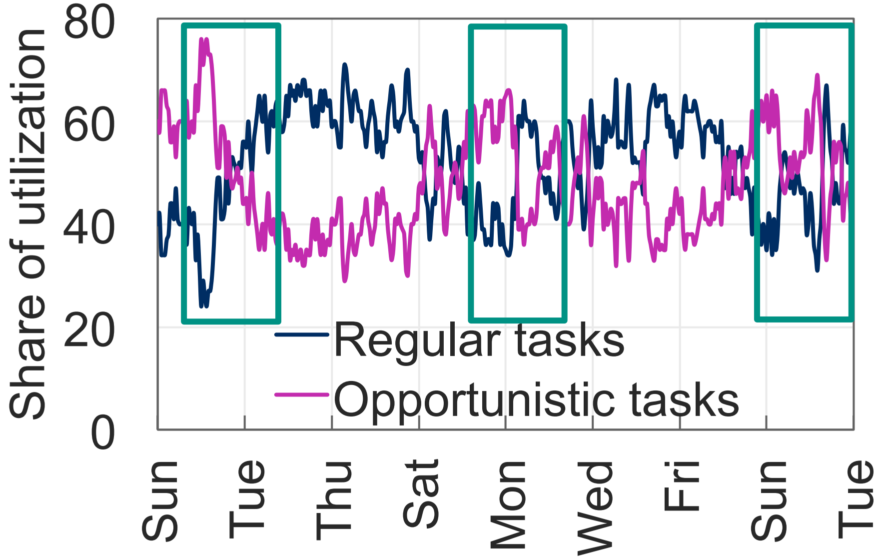
— Tracks **14,000,000 pending tasks**

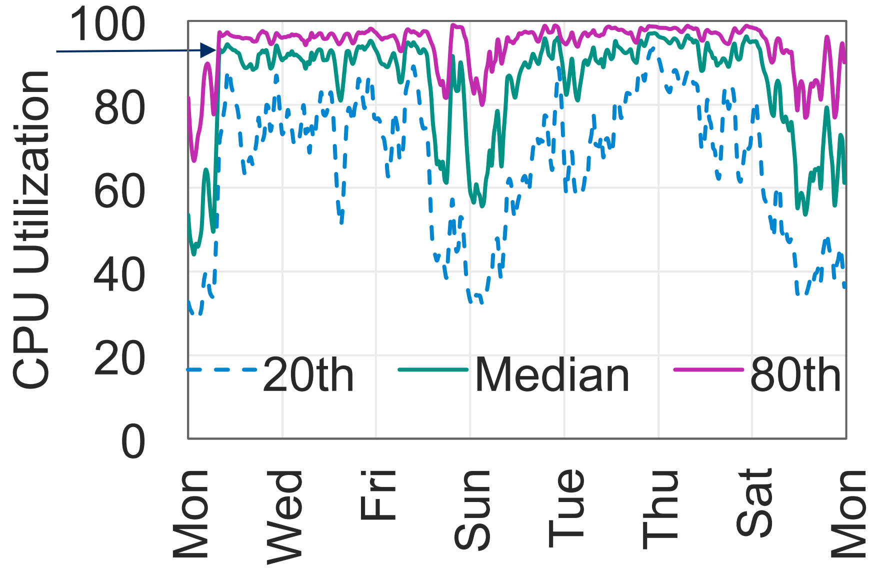
How does Apollo perform?

- **Baseline**: Previous production scheduler, lacking coordination and estimates
- **Ideal**: Trace-driven simulator with infinite capacity



Does Apollo use resources efficiently?





Opportunistic tasks increase their share of utilization on weekends

90% median CPU utilization under load

No impact to regular tasks

Regular tasks < 1 second queue time at the 95th percentile

Apollo

Background
Challenges
Overview
— Distributed and coordinated architecture
— Estimation-based scheduling
— Conflict resolution
— Opportunistic scheduling
Evaluation at scale
Related work
Conclusion

Related Work

Centralized Schedulers

Quincy

Delay Scheduler

Hierchical Schedulers

Mesos

Yarn

Corona

Decentralized Schedulers

Sparrow

Omega

Conclusion

Apollo

Loosely Coordinated Distributed architecture	High Quality Scheduling	Maximize resource utilization
Deployed to clusters with over 20,000 servers	Minimize task completion time Consistent performance	Opportunistic scheduling 90% median CPU utilization