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)

```sql
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

<table>
<thead>
<tr>
<th>Tasks runs from <strong>seconds to hours</strong></th>
<th>Tasks can require from 100MB to more than 10GB of memory</th>
<th>Short tasks are sensitive to <strong>scheduling latency</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks can be <strong>IO bound</strong> or <strong>CPU bound</strong></td>
<td></td>
<td>Long IO bound tasks are sensitive to <strong>locality</strong></td>
</tr>
<tr>
<td>Challenge: Make <strong>optimal scheduling decisions</strong> for a complex workload</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tasks may run for seconds to hours and can be IO bound or CPU bound. They may require from 100MB to more than 10GB of memory. Short tasks are sensitive to scheduling latency, while long IO bound tasks are sensitive to locality.
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**

Number of concurrent jobs drops by **40%** on weekends
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
Queue allows to reason about future resource availability and to defer conflict resolution.

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

The wait time matrix allows to **reason about future resource availability**
Optimizing for various factors

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

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

Apollo: Consistent performance despite variation in load

>1.5x speedup over baseline
Does Apollo use resources efficiently?

**Opportunistic tasks** increase their share of utilization on **weekends**

**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

Hierarchical Schedulers
- Mesos
- Yarn
- Corona

Decentralized Schedulers
- Sparrow
- Omega
## Conclusion

### Apollo

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

22