Cluster Scheduling at Microsoft Scale

Konstantinos Karanasos

MIT, 10/31/2019

Computer Networks (6.829)
GSL (CISL)

**Mission**: applied research lab working on systems for big data, cloud, and machine learning

Office of the CTO for Azure Data

~15 researchers/engineers in the Bay Area, Redmond, and Madison
The three hats we wear

Applied research group

Collaborating with database, big data, and AI infra groups at MS

Open-sourcing our code
   Apache Hadoop, ONNX Runtime, MlFlow, REEF, Heron
Current Focus Areas

- Resource management
- Systems for ML
- ML for Systems
- Query Optimization
- Provenance
What is a Resource Manager?

Jobs consist of tasks

The RM allows jobs to acquire cluster resources in the form of containers

Popular examples: YARN, Apollo, Mesos, Borg, Kubernetes

Same end goal, different designs

Centralized/distributed

Targeting batch/interactive jobs, services
Lessons learned: Abstracting out the RM layer

Initial big data systems were monolithic

Reuse of RM component by multiple applications

We focus on YARN, but most systems follow layering abstractions
Cosmos: Microsoft’s Analytics Stack

Application Engines
- CloudViews
  - Scope
  - U-SQL
  - Hive
  - Spark
- AzureML
- TLC++
- Dhalion
  - Heron
  - Samza
  - ASA

Per-job resource management
- MR/Tez
  - Scope on YARN (SoY)
- REEF
- Heron AM
- Spark Runtime

Hydra
- Morpheus (SLA)
- Medea (services)
- Rayon (reservations)
- Federation (scale)
- Mercury (utilization)

Storage
- Router-Based Federation (scale)
- NetCachier (I/O planning for SLAs)
- Tiered HDFS
- ADLS
- WASB
- S3
- HDFS

Service Analytics
- Helios Access Shim (HDFS Input Format)
- Anomaly Detection (Supernova)
- DistributedStore (Ingestion + Metadata Indexing)
- Provenance Telemetry Graph (PTG)
- Helios Agent
- Cosmos Logs
The Scale and Utilization Challenge

Cluster(s):
> 50K nodes

Job(s):
> 2M tasks,
> 5PB input

Scheduler:
> 70K QPS

Utilization:
~60% avg CPU util

Tasks:
10 sec 50\textsuperscript{th} %ile
Scheduling in Analytics Clusters: a Journey...

Scope, Centralized sched.  
[euros07, vldb08]

Hadoop MR, Centralized sched.

Distributed sched.  
+tooling/optimizer,  
+scale, +high utilization  
[osdi14]

Hydra

+multi-framework  
+security  
+scheduler expressivity  
[socc13]
Teaser...

>99% tenants migrated
>250K servers
>500k daily jobs
>1 Zetabyte data processed
>1 Trillion tasks scheduled
Agenda

Overview of legacy systems (Apollo, YARN)

Scale: Federated YARN

Resource utilization: Opportunistic Containers

Production Experience
Distributed Scheduling: Legacy Cosmos (Apollo)
Legacy Cosmos (Apollo)

Distributed scheduling got us *scale*

* JM: Job Manager

Distributed scheduling + node-level queuing
Legacy Cosmos

**Problem:** (required) static resource allocation yields low utilization resource fragmentation within and across queues

**Solution:** opportunistically share resources

"opportunistic" containers

"guaranteed" containers

opportunistic scheduling got us *utilization*
Legacy Cosmos: Pros/Cons

+ Great scalability

+ Good resource utilization

- No support for multiple frameworks

- Limited control on scheduling (e.g., fairness, load-balancing, locality, special HW)
Centralized Scheduling: Apache Hadoop YARN
Legacy YARN

1) Submit job

2) Admission control (fairness, quotas, constraints, SLOs, ...)

3) Schedule Application Master (AM)

4) Start AM container
Legacy YARN

5) RM grants “token”

6) Start container

7) AM-Task communication
Legacy YARN: Pros/Cons

+ Support for arbitrary frameworks

+ Rich scheduling invariants

+ Non-tech advantage: OSS/mind-share

- Scale limits (~5K nodes in 2013)

- Utilization: heartbeats lead to idle resources
Main challenges

- High resource utilization: Mercury [atc15, euros16]
- Scalability: Federation [nsdi19]
- Rich placement constraints: Medea [euros18]
- Production jobs and predictability: Morpheus [osdi16]
Improving YARN’s scalability: Federated Architecture
Need **scale and utilization**?
Go distributed!

Need **scheduling control and multi-framework**?
Go centralized!

Want it all?

YOU GOTTA FEDERATE!
Hydra Architecture

Router

State Store

StateStore Proxy

RM

sub-cluster 1

RM

sub-cluster 2
Hydra Architecture
Hydra Architecture

[Diagram showing Hydra Architecture with components such as StateStore, Proxy, AM, RM, Global Policy Generator, and sub-clusters 1 and 2.]

State Store

Global Policy Generator

AM

AMRM Proxy

RM

NM

sub-cluster 1

sub-cluster 2
Scheduling Desiderata: *Global* Goals

High utilization

Scheduling invariants (e.g., fairness)

Locality (e.g., machine preferences)
Policies

AMRMProxy routing of requests
Enforce locality?

Per-cluster RM scheduling decisions
Enforce quotas?

Locality, Fairness
Utilization

Utilization, locality
Fairness

Utilization
Fairness
Locality

root

$Q_A$ 50% $Q_B$ 50%

demand: 12 demand: 4 demand: 12 demand: 12
Key Idea

Decouple:

**Share determination**
- How many resources should a queue get?

**Placement**
- On which machines should each task run?
Proposed Solution*

1. Periodically gather queue information at GPG
2. Determine resources for each queue at each sub-cluster centrally
   Logically reassign all resources, accounting for demand skew (and already assigned resources)
3. Propagate capacity decisions to each sub-cluster’s RM, which perform local task allocation

* More advanced than what in prod. (details in paper)
Handling GPG downtime

If GPG is down, we would fallback to local decisions
   Problematic if they “diverge” too much from global one

Leverage LP-based “tuning” of local queue allocation
   Historical demand as a predictor of future demand
Improving YARN’s Utilization: Opportunistic Containers
Suboptimal Utilization in Scope-on-YARN

Due to:

Gang scheduling
Feedback delays

<table>
<thead>
<tr>
<th></th>
<th>5 sec</th>
<th>10 sec</th>
<th>50 sec</th>
<th>Mixed-5-50</th>
</tr>
</thead>
<tbody>
<tr>
<td>60.59%</td>
<td>78.35%</td>
<td>92.38%</td>
<td>78.54%</td>
<td></td>
</tr>
</tbody>
</table>

Average slot utilization for varying workloads in YARN.
Opportunistic Containers in YARN

Mask feedback delays
Only OPP containers can be queued
Promotion/demotion of containers
Utilization Gains with Node-Side Queuing

Sufficiently long queues lead to optimal utilization

The shorter the tasks the longer the queues need to be

But are long queues all we need?
Naïve node-side queuing can be detrimental for job completion times.

Proper queue management techniques are required.
Problems with Node-Side Queuing

Load imbalance across nodes
  Suboptimal task placement

Head-of-line blocking
  Especially for heterogeneous tasks

Early binding of tasks to nodes
Queue management techniques

- Place tasks to node queues
- Prioritize task execution (queue reordering)
- Bound queue lengths
Placement of Tasks to Queues

Placement based on **queue length**
- Agnostic of task characteristics
- Suboptimal placement for heterogeneous workloads

Placement based on **queue wait time**
- Better for heterogeneous workloads
- Requires task duration estimates
Task Prioritization

Queue reordering strategies
- Shortest Remaining Job First (SRJF)
- Least Remaining Tasks First (LRTF)
- Shortest Task First (STF)
- Earliest Job First (EJF)

SRJF and LRTF are **job-aware**
- Dynamically reorder tasks based on job progress

Starvation freedom
- Give priority to tasks waiting more than $X$ secs
Bounding Queue Lengths

Determine max number of tasks at a queue
   Trade-off between short and long queues

Short queues
   Resource idling
   → lower throughput

Long queues
   High queuing delays, early binding of tasks to queues
   → longer job completion times

Static and dynamic queue bounding
Queuing Techniques: Evaluation

1.7x improvement in median JCT over YARN

Both bounding and reordering are crucial
Production Experience
Workload

- ECDF plots for average input size (bytes) across different conditions (C1, C2, C3, C4, C5).
- ECDF plots for average job runtime (sec) across different conditions.
- ECDF plots for average number of tasks (#) across different conditions.
- Bar charts showing the percentage of machines used for batch jobs per cluster (C1, C2, C3, C4, C5).
Scale

High scheduling rate at low allocation latency
Utilization

Federated design improves load balancing, while retaining utilization
Task Performance (comparison)
Performance

Jobs perform just as well (and tasks are as efficient)
Qualitative Experience

In-place migration: changing plane engine mid-flight

Happy customers can now play with OSS tech + MS stack!

Federated design improved operability:
  - Experiment at sub-cluster granularity
  - OSS innovation is easier to leverage

Policy-driven design:
  - Allows us to dynamically adapt and experiment
Recap

Overview of legacy systems (Apollo, YARN)

Scale: Federated YARN

Resource utilization: Opportunistic Containers

Production Experience
Conclusion

Research in Industry is a lot of fun

Fewer bigger projects, with massive wins
Big force multipliers
Failure is expected (otherwise we are not trying hard enough)

Modus Operandi

Be picky in choosing problems
Engage early, engage deep
Be inclusive with prod/OSS counterparts
Thank you!

Konstantinos.Karanasos@microsoft.com