Resource Management - Mesos and YARN

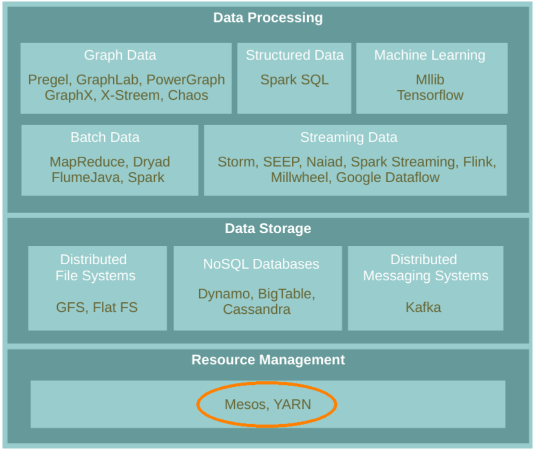
Amir H. Payberah
payberah@kth.se
2020-09-30

The Course Web Page

https://id2221kth.github.io

https://tinyurl.com/y4qph82u

Where Are We?


Data Processing
Graph Data
Pregel, GraphLab, PowerGraph
GraphX, X-Streem, Chaos
Structured Data
Spark SQL
Machine Learning
Mllib
Tensorflow
Batch Data
MapReduce, Dryad
FlumeJava, Spark
Streaming Data
Storm, SEEP, Naiad, Spark Streaming, Flink,
Millwheel, Google Dataflow
Data Storage
Distributed
File Systems
GFS, Flat FS
NoSQL Databases
Dynamo, BigTable,
Cassandra
Distributed
Messaging Systems
Kafka
Resource Management
Mesos, YARN

Motivation

- Rapid innovation in cloud computing.
- No single framework optimal for all applications.
- Running each framework on its dedicated cluster:
  - Expensive
  - Hard to share data

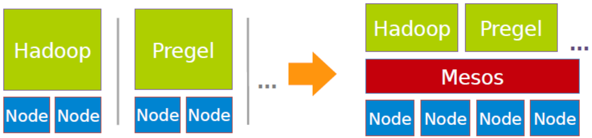
Proposed Solution

- Running multiple frameworks on a single cluster.
- Maximize utilization and share data between frameworks.
- Two resource management systems:
  - Mesos
  - YARN

Mesos

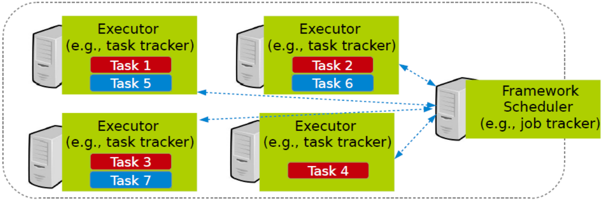
Mesos

- Mesos is a common resource sharing layer, over which diverse frameworks can run.



Computation Model

- A framework (e.g., Hadoop, Spark) manages and runs one or more jobs.
- A job consists of one or more tasks.
- A task (e.g., map, reduce) consists of one or more processes running on same machine.

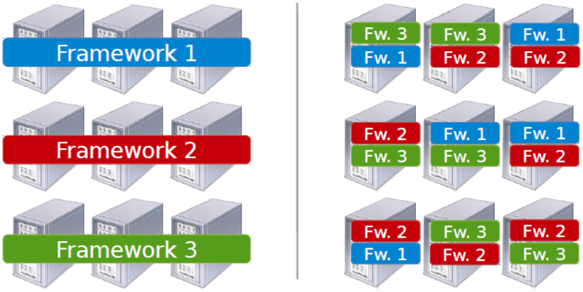


Mesos Design Elements

- Fine-grained sharing
- Resource offers

Fine-Grained Sharing

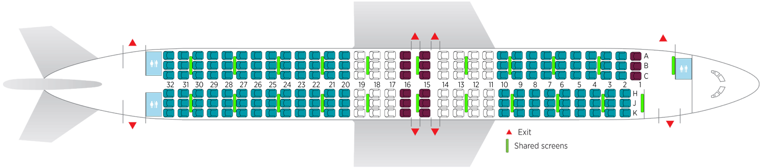
- Allocation at the level of tasks within a job.
- Improves utilization, latency, and data locality.



Coarse-grained sharing Fine-grained sharing

Resource Offer

- Offer available resources to frameworks, let them pick which resources to use and which tasks to launch.
- Keeps Mesos simple, lets it support future frameworks.



Question?

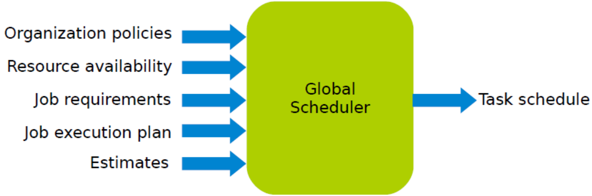
How to schedule resource offering among frameworks?

Schedule Frameworks

- Global scheduler
- Distributed scheduler

Global Scheduler (1/2)

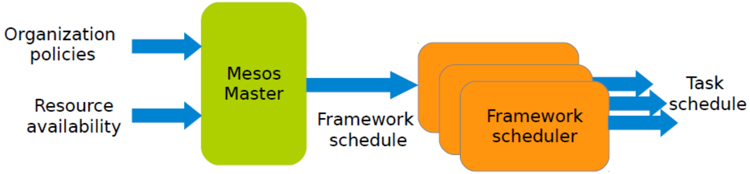
- **Job requirements**
  - Response time
  - Throughput
  - Availability
- **Job execution plan**
  - Task DAG
  - Inputs/outputs
- **Estimates**
  - Task duration
  - Input sizes
  - Transfer sizes



Global Scheduler (2/2)

- Advantages
  - Can achieve optimal schedule.
- Disadvantages
  - Complexity: hard to scale and ensure resilience.
  - Hard to anticipate future frameworks requirements.
  - Need to refactor existing frameworks.

Distributed Scheduler (1/3)



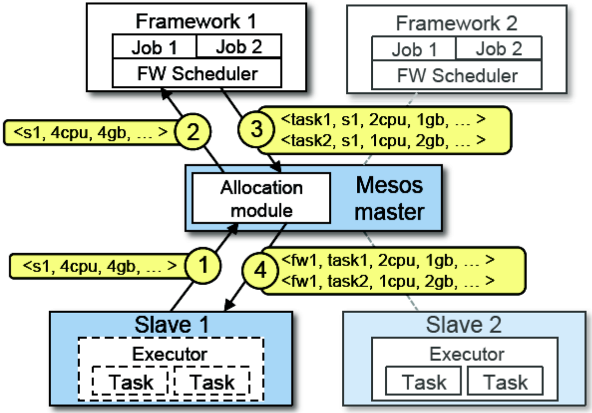
Distributed Scheduler (2/3)

- Master sends resource offers to frameworks.
- Frameworks select which offers to accept and which tasks to run.
- Unit of allocation: resource offer
  - Vector of available resources on a node
  - For example, node1: $\langle \mathtt{1CPU}, \mathtt{1GB} \rangle$, node2: $\langle \mathtt{4CPU}, \mathtt{16GB} \rangle$

Distributed Scheduler (3/3)

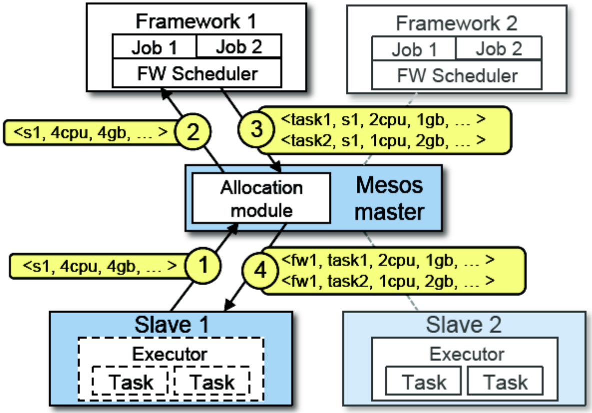
- Advantages
  - Simple: easier to scale and make resilient.
  - Easy to port existing frameworks, support new ones.
- Disadvantages
  - Distributed scheduling decision: not optimal.

Mesos Architecture (1/4)



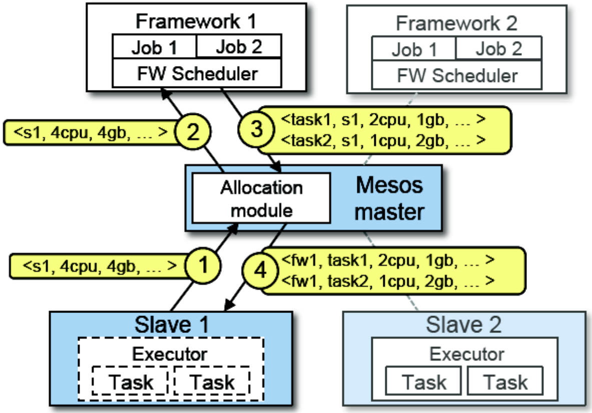
- Slaves continuously send status updates about resources to the Master.

Mesos Architecture (2/4)



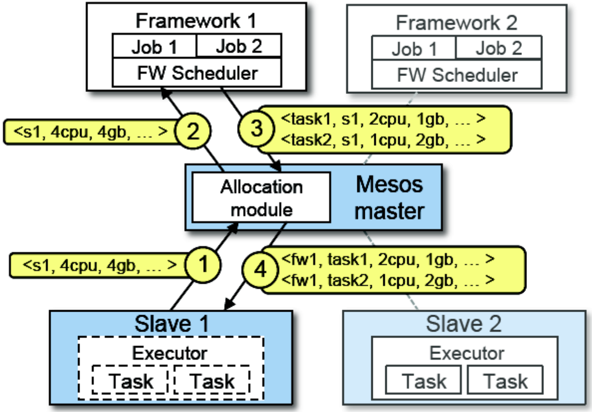
- Pluggable scheduler picks framework to send an offer to.

Mesos Architecture (3/4)



- Framework scheduler selects resources and provides tasks.

Mesos Architecture (4/4)



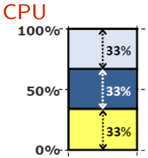
- Framework executors launch tasks.

Question?

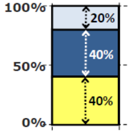
How to allocate resources of different types?

Single Resource: Fair Sharing

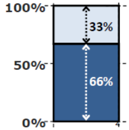
- n users want to share a resource, e.g., CPU.
  - Solution: allocate each $\frac{1}{n}$ of the shared resource.



- Generalized by max-min fairness.
  - Handles if a user wants less than its fair share.
  - E.g., user 1 wants no more than 20%.



- Generalized by weighted max-min fairness.
  - Give weights to users according to importance.
  - E.g., user 1 gets weight 1, user 2 weight 2.



Max-Min Fairness - Example

- 1 resource: CPU
- Total resources: 20 CPU
- User 1 has $\mathtt{x}$ tasks and wants $\langle \mathtt{1CPU} \rangle$ per task
- User 2 has $\mathtt{y}$ tasks and wants $\langle \mathtt{2CPU} \rangle$ per task

  $\mathtt{max}(\mathtt{x}, \mathtt{y})$ (maximize allocation)
  subject to
  $\mathtt{x} + 2\mathtt{y} \leq 20$ (CPU constraint)
  $\mathtt{x} = 2\mathtt{y}$
  so
  $\mathtt{x} = 10$
  $\mathtt{y} = 5$

Properties of Max-Min Fairness

- Share guarantee
  - Each user can get at least $\frac{1}{n}$ of the resource.
  - But will get less if her demand is less.
- Strategy proof
  - Users are not better off by asking for more than they need.
  - Users have no reason to lie.
- Max-Min fairness is the only reasonable mechanism with these two properties.
- Widely used: OS, networking, datacenters, ...

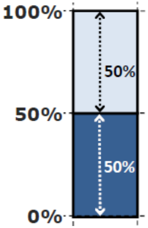
Question?

When is Max-Min Fairness NOT Enough?

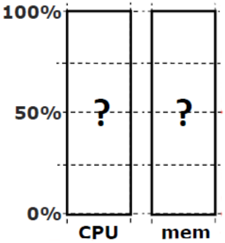
Need to schedule multiple, heterogeneous resources, e.g., CPU, memory, etc.

Problem

- Single resource example
  - 1 resource: CPU
  - User 1 wants $\langle 1\text{CPU} \rangle$ per task
  - User 2 wants $\langle 2\text{CPU} \rangle$ per task

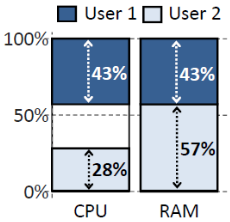


- Multi-resource example
  - 2 resources: CPUs and mem
  - User 1 wants $\langle 1\text{CPU}, 4\text{GB} \rangle$ per task
  - User 2 wants $\langle 2\text{CPU}, 1\text{GB} \rangle$ per task
  - What is a fair allocation?



A Natural Policy (1/2)

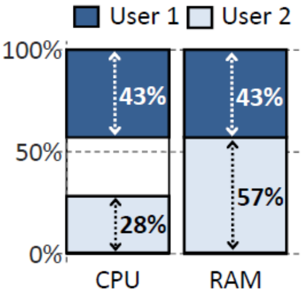
- Asset fairness: give weights to resources (e.g., 1 CPU = 1 GB) and equalize total value given to each user.
- Total resources: 28 CPU and 56GB RAM (e.g., 1 CPU = 2 GB)
  - User 1 has $\mathtt{x}$ tasks and wants $\langle \mathtt{1CPU}, \mathtt{2GB} \rangle$ per task
  - User 2 has $\mathtt{y}$ tasks and wants $\langle \mathtt{1CPU}, \mathtt{4GB} \rangle$ per task



- Asset fairness yields:

$\mathtt{max}(\mathtt{x}, \mathtt{y})$
$\mathtt{x} + \mathtt{y} \leq 28$
$2\mathtt{x} + 4\mathtt{y} \leq 56$
$2\mathtt{x} = 3\mathtt{y}$
User 1: $\mathtt{x} = 12$: $\langle \mathtt{43\%CPU}, \mathtt{43\%GB} \rangle$ $(\sum = 86\%)$
User 2: $\mathtt{y} = 8$: $\langle \mathtt{28\%CPU}, \mathtt{57\%GB} \rangle$ $(\sum = 86\%)$

A Natural Policy (2/2)



- Problem: violates share grantee.
- User 1 gets less than 50% of both CPU and RAM.
- Better off in a separate cluster with half the resources.

Challenge

- Can we find a fair sharing policy that provides:
  - Share guarantee
  - Strategy-proofness

- Can we generalize max-min fairness to multiple resources?

Proposed Solution

Dominant Resource Fairness (DRF)

Dominant Resource Fairness (DRF) (1/2)

- Dominant resource of a user: the resource that user has the biggest share of.
  - Total resources: $\langle 8\mathtt{CPU}, 5\mathtt{GB}\rangle$
  - User 1 allocation: $\langle 2\mathtt{CPU}, 1\mathtt{GB}\rangle$: $\frac{2}{8} = 25\%\ \mathtt{CPU}$ and $\frac{1}{5} = 20\%\ \mathtt{RAM}$
  - Dominant resource of User 1 is CPU ($25\% > 20\%$)

- Dominant share of a user: the fraction of the dominant resource she is allocated.
  - User 1 dominant share is 25%.

Dominant Resource Fairness (DRF) (2/2)

- Apply max-min fairness to dominant shares: give every user an equal share of her dominant resource.

- Equalize the dominant share of the users.
  - Total resources: $\langle 9\mathtt{CPU}, 18\mathtt{GB}\rangle$
  - User 1 wants $\langle 1\mathtt{CPU}, 4\mathtt{GB}\rangle$; Dominant resource: RAM ($\frac{1}{9} < \frac{4}{18}$)
  - User 2 wants $\langle 3\mathtt{CPU}, 1\mathtt{GB}\rangle$; Dominant resource: CPU ($\frac{3}{9} > \frac{1}{18}$)

- $\max(\mathtt{x}, \mathtt{y})$

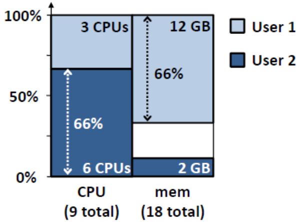
  $\mathtt{x} + 3\mathtt{y} \leq 9$

  $4\mathtt{x} + \mathtt{y} \leq 18$

  $\frac{4\mathtt{x}}{18} = \frac{3\mathtt{y}}{9}$

  User 1: $\mathtt{x} = 3$: $\langle 33\%\mathtt{CPU}, 66\%\mathtt{GB}\rangle$

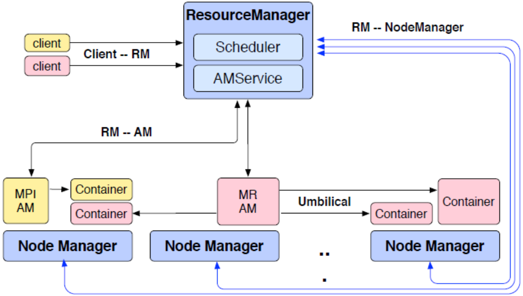
  User 2: $\mathtt{y} = 2$: $\langle 66\%\mathtt{CPU}, 16\%\mathtt{GB}\rangle$



YARN

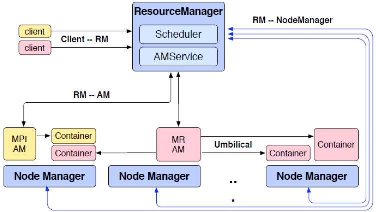
YARN Architecture

- Resource Manager (RM)
- Application Master (AM)
- Node Manager (NM)



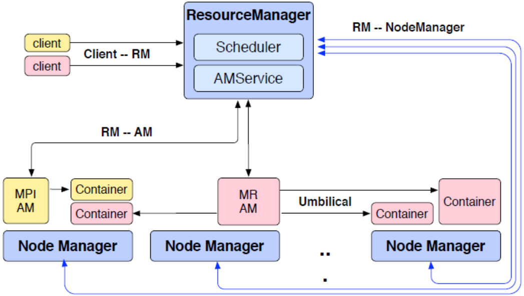
YARN Architecture - Resource Manager (1/2)

- One per cluster
  - Central: global view
- Job requests are submitted to RM.
  - To start a job, RM finds a container to spawn AM.
- Container: logical bundle of resources (CPU/memory)



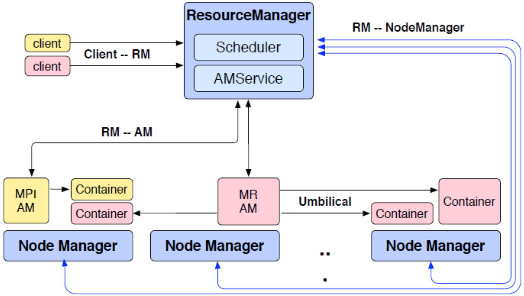
YARN Architecture - Resource Manager (2/2)

- Only handles an overall resource profile for each job.
  - Local optimization is up to the job.
- Preemption
  - Request resources back from an job.
  - Checkpoint jobs



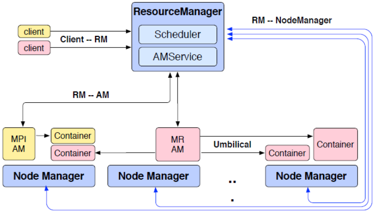
YARN Architecture - Application Manager

- The head of a job.
- Runs as a container.
- Request resources from RM (num. of containers/resource per container/locality ...)



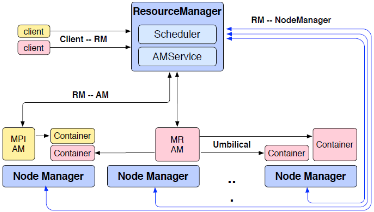
YARN Architecture - Node Manager (1/2)

- The worker daemon.
- Registers with RM.
- One per node.
- Report resources to RM: memory, CPU, ...



YARN Architecture - Node Manager (2/2)

- Configure the environment for task execution.
- Garbage collection.
- Auxiliary services.
  - A process may produce data that persist beyond the life of the container.
  - Output intermediate data between map and reduce tasks.



YARN Framework (1/2)

- Containers are described by a Container Launch Context (CLC).
  - The command necessary to create the process
  - Environment variables
  - Security tokens
  - ...
- Submitting the job: passing a CLC for the AM to the RM.
- When RM starts the AM, it should register with the RM.
  - Periodically advertise its liveness and requirements over the heartbeat protocol.

YARN Framework (2/2)

- Once the RM allocates a container, AM can construct a CLC to launch the container on the corresponding NM.
  - It monitors the status of the running container and stop it when the resource should be reclaimed.

- Once the AM is done with its work, it should unregister from the RM and exit cleanly.

Summary

Summary

- Mesos
  - Offered-based
  - Max-Min fairness: DRF
- YARN
  - Request-based
  - RM, AM, NM

References

- B. Hindman et al., "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center", NSDI 2011
- V. Vavilapalli et al., "Apache hadoop yarn: Yet another resource negotiator", ACM Cloud Computing 2013

Questions?

Acknowledgements

Some slides were derived from Ion Stoica and Ali Ghodsi slides (Berkeley University), and Wei-Chiu Chuang slides (Purdue University).