Resource Management - Mesos, YARN, and Borg

Amir H. Payberah
payberah@kth.se

2023-10-02

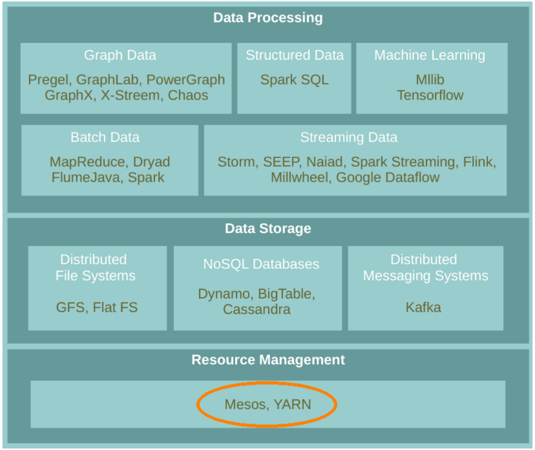
The Course Web Page

https://id2221kth.github.io

The Questions-Answers Page

https://tinyurl.com/hk7hzpw5

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.

Proposed Solution

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

Question?

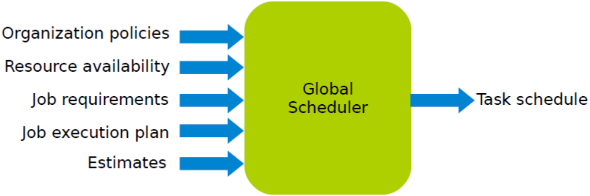
How to schedule resource offering among frameworks?

Schedule Frameworks

- Monolithic scheduler
- Two-Level scheduler

Monolithic Scheduler (1/2)

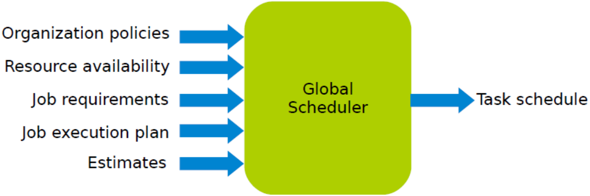
- Job requirements
  - Response time
  - Throughput
  - Availability



Monolithic Scheduler (1/2)

- Job requirements
  - Response time
  - Throughput
  - Availability

- Job execution plan
  - Task DAG
  - Inputs/outputs

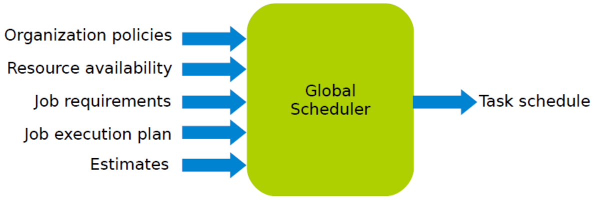


Monolithic Scheduler (1/2)

- Job requirements
  - Response time
  - Throughput
  - Availability

- Job execution plan
  - Task DAG
  - Inputs/outputs

- Estimates
  - Task duration
  - Input sizes
  - Transfer sizes



Monolithic Scheduler (2/2)

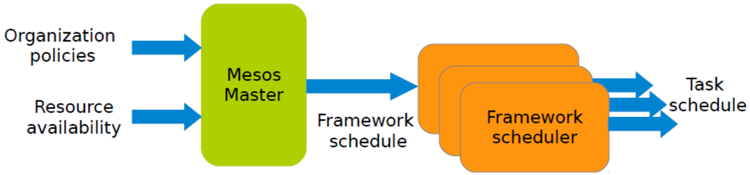
- Advantages
  - Can achieve optimal schedule.

Monolithic 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.

Two-Level Scheduler (1/2)



Two-Level Scheduler (2/2)

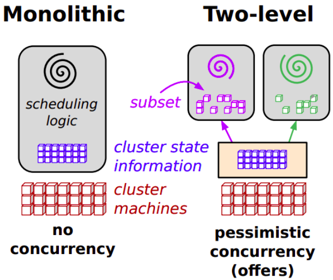
- Advantages
  - Simple: easier to scale and make resilient.
  - Easy to port existing frameworks, support new ones.

Two-Level Scheduler (2/2)

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

Two-Level vs. Monolithic

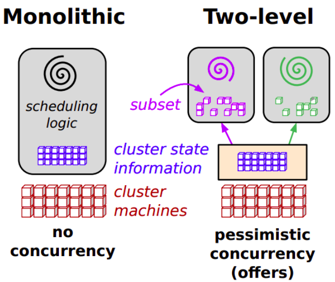
- Two-level schedulers: separate concerns of resource allocation and task placement.



[Schwarzkopf et al., Omega: flexible, scalable schedulers for large compute clusters, EuroSys'13.]

Two-Level vs. Monolithic

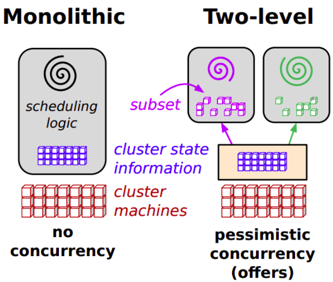
- Two-level schedulers: separate concerns of resource allocation and task placement.
  - An active resource manager offers compute resources to multiple parallel, independent scheduler frameworks.



[Schwarzkopf et al., Omega: flexible, scalable schedulers for large compute clusters, EuroSys'13.]

Two-Level vs. Monolithic

- Two-level schedulers: separate concerns of resource allocation and task placement.
  - An active resource manager offers compute resources to multiple parallel, independent scheduler frameworks.
  - Mesos and Yarn

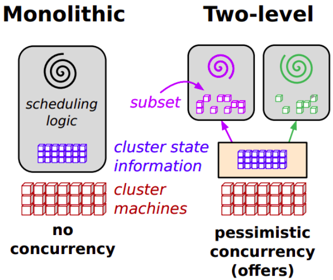


[Schwarzkopf et al., Omega: flexible, scalable schedulers for large compute clusters, EuroSys'13.]

Two-Level vs. Monolithic

- Two-level schedulers: separate concerns of resource allocation and task placement.
  - An active resource manager offers compute resources to multiple parallel, independent scheduler frameworks.
  - Mesos and Yarn

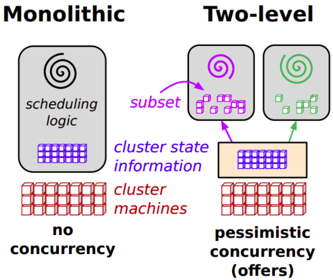
- Monolithic schedulers: use a single, centralized scheduling algorithm for all jobs.



[Schwarzkopf et al., Omega: flexible, scalable schedulers for large compute clusters, EuroSys'13.]

Two-Level vs. Monolithic

- Two-level schedulers: separate concerns of resource allocation and task placement.
  - An active resource manager offers compute resources to multiple parallel, independent scheduler frameworks.
  - Mesos and Yarn
- Monolithic schedulers: use a single, centralized scheduling algorithm for all jobs.
  - Borg

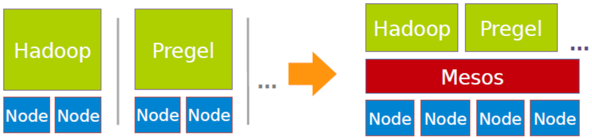


[Schwarzkopf et al., Omega: flexible, scalable schedulers for large compute clusters, EuroSys'13.]

Mesos

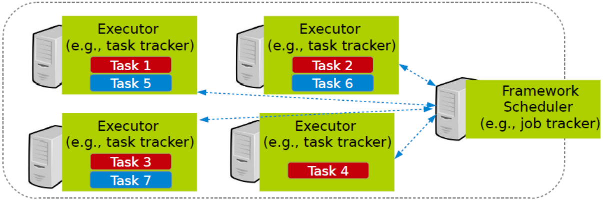
Mesos

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



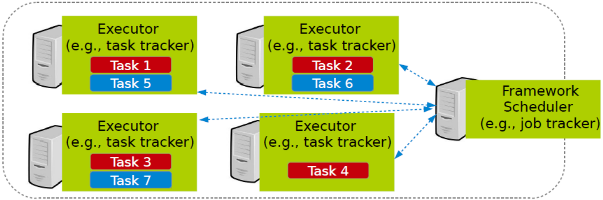
Computation Model

- A framework (e.g., Hadoop, Spark) manages and runs one or more jobs.



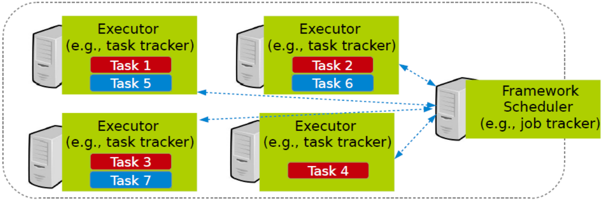
Computation Model

- A framework (e.g., Hadoop, Spark) manages and runs one or more jobs.
- A job consists of one or more tasks.



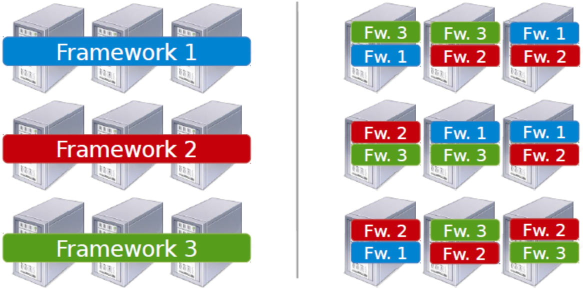
Computation Model

- 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.



Fine-Grained Sharing

- Allocation at the level of tasks within a job.



Coarse-grained sharing   Fine-grained sharing

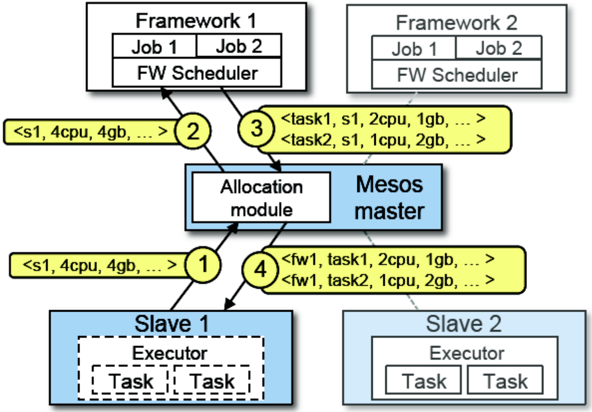
Mesos Scheduler

- Master sends resource offers to frameworks.
- Frameworks select which offers to accept and which tasks to run.

Mesos Scheduler

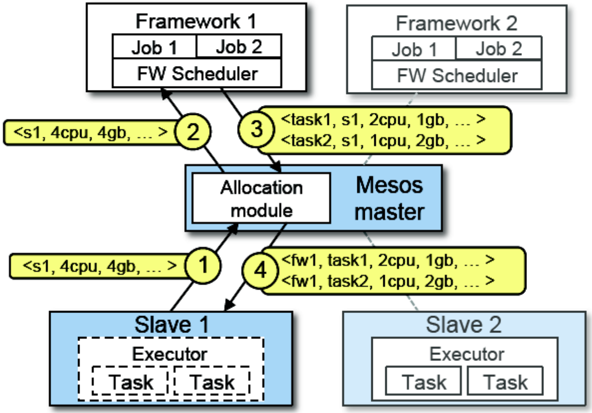
- Master sends resource offers to frameworks.
- Frameworks select which offers to accept and which tasks to run.
- Unit of allocation: resource offer
    - Vector of available resources on a node
    - For example, node1: $\langle \texttt{1CPU}, \texttt{1GB} \rangle$, node2: $\langle \texttt{4CPU}, \texttt{16GB}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

Mesos Architecture (1/4)



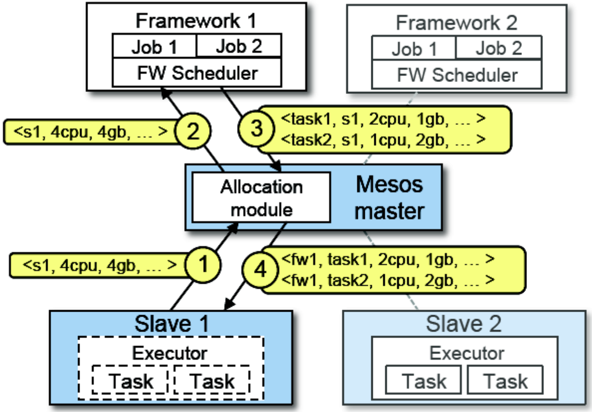
- Slaves continuously send status updates about resources to the Master.

Mesos Architecture (2/4)



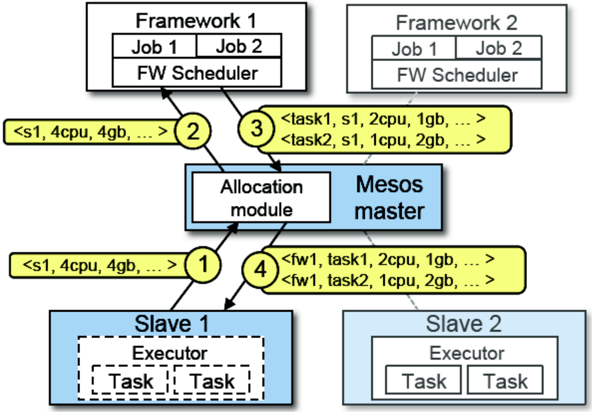
- Pluggable scheduler picks framework to send an offer to.

Mesos Architecture (3/4)



- Framework scheduler selects resources and provides tasks.

Mesos Architecture (4/4)



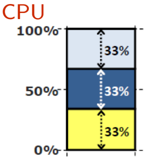
- Framework executors launch tasks.

Question?

How to allocate resources of different types?

Single Resource: Fair Sharing

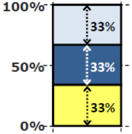
- n users want to share a resource, e.g., CPU.
  - Solution: allocate each $\frac{1}{n}$ of the shared resource.



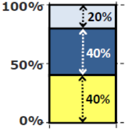
Single Resource: Fair Sharing

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

CPU



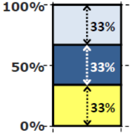
- Generalized by max-min fairness.
  - Handles if a user wants less than its fair share.
  - E.g., user 1 wants no more than 20%.



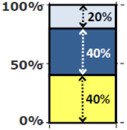
Single Resource: Fair Sharing

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

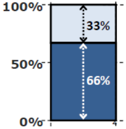
CPU



- 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 `x` tasks and wants $\langle$`1CPU`$\rangle$ per task
- User 2 has `y` tasks and wants $\langle$`2CPU`$\rangle$ per task

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)

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)
$\mathtt{x} = 2\mathtt{y}$

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.

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.

Question?

When is Max-Min Fairness NOT Enough?

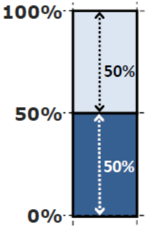
Question?

When is Max-Min Fairness NOT Enough?

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

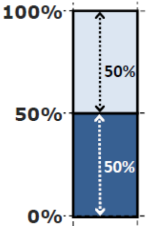
Problem

- Single resource example
  - 1 resource: CPU
  - User 1 wants ⟨1CPU⟩ per task
  - User 2 wants ⟨2CPU⟩ per task

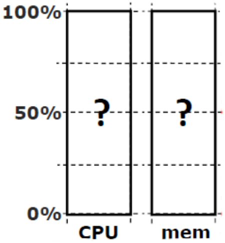


Problem

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

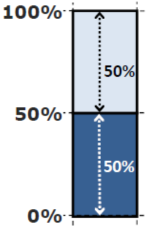


- Multi-resource example
  - 2 resources: CPUs and mem
  - User 1 wants $\langle 1CPU, 4GB \rangle$ per task
  - User 2 wants $\langle 2CPU, 1GB \rangle$ per task

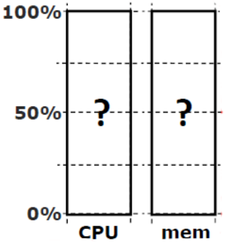


Problem

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



- Multi-resource example
  - 2 resources: CPUs and mem
  - User 1 wants $\langle 1CPU, 4GB \rangle$ per task
  - User 2 wants $\langle 2CPU, 1GB \rangle$ 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.

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 x tasks and wants $\langle 1CPU, 2GB \rangle$ per task
  - User 2 has y tasks and wants $\langle 1CPU, 4GB \rangle$ per task

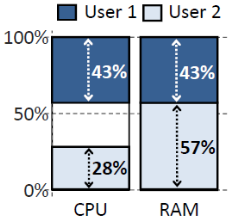
A Natural Policy (1/2)

- 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:

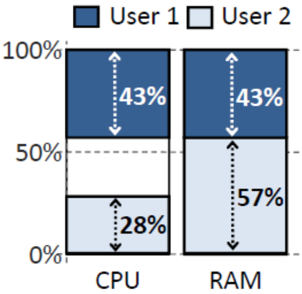
$$\mathtt{max(x, y)}$$
$$\mathtt{x} + \mathtt{y} \leq 28$$
$$2\mathtt{x} + 4\mathtt{y} \leq 56$$
$$2\mathtt{x} = 3\mathtt{y}$$

User 1: $\mathtt{x} = 12$: $\langle 43\%\mathtt{CPU}, 43\%\mathtt{GB} \rangle$ $(\sum = 86\%)$

User 2: $\mathtt{y} = 8$: $\langle 28\%\mathtt{CPU}, 57\%\mathtt{GB} \rangle$ $(\sum = 86\%)$



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 Resource Fairness (DRF) (1/2)

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

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: $\langle 9\text{CPU}, 18\text{GB} \rangle$
  - User 1 wants $\langle 1\text{CPU}, 4\text{GB} \rangle$; Dominant resource: RAM ($\frac{1}{9} < \frac{4}{18}$)
  - User 2 wants $\langle 3\text{CPU}, 1\text{GB} \rangle$; Dominant resource: CPU ($\frac{3}{9} > \frac{1}{18}$)

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: $\langle 9\text{CPU}, 18\text{GB}\rangle$
  - User 1 wants $\langle 1\text{CPU}, 4\text{GB}\rangle$; Dominant resource: RAM ($\frac{1}{9} < \frac{4}{18}$)
  - User 2 wants $\langle 3\text{CPU}, 1\text{GB}\rangle$; Dominant resource: CPU ($\frac{3}{9} > \frac{1}{18}$)

- $\max(x, y)$

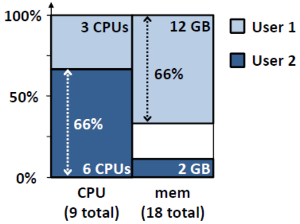
  $x + 3y \leq 9$

  $4x + y \leq 18$

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

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

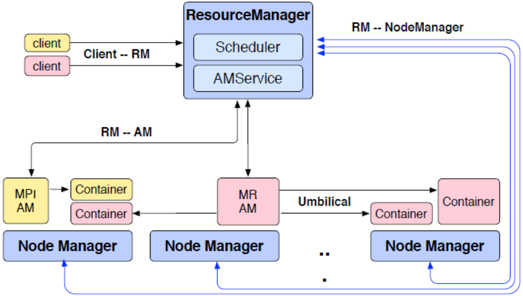
  User 2: $y = 2$: $\langle 66\%\text{CPU}, 16\%\text{GB}\rangle$



YARN

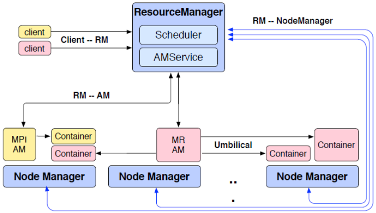
YARN Architecture

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



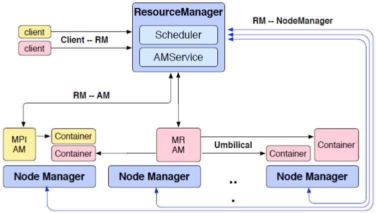
YARN Architecture - Resource Manager

- One per cluster (Central: global view)



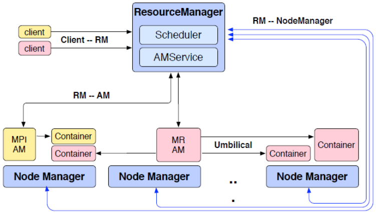
YARN Architecture - Resource Manager

- One per cluster (Central: global view)
- Job requests are submitted to RM.
  - To start a job, RM finds a container to spawn AM.



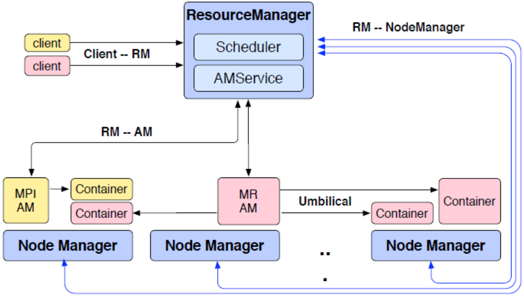
YARN Architecture - Resource Manager

- One per cluster (Central: global view)
- Job requests are submitted to RM.
    - To start a job, RM finds a container to spawn AM.
- Only handles an overall resource profile for each job.
    - Local optimization is up to the job.



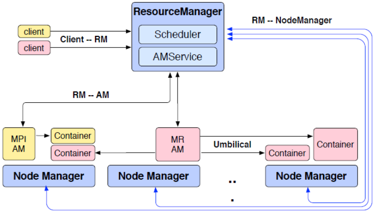
YARN Architecture - Application Manager

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



YARN Architecture - Node Manager

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



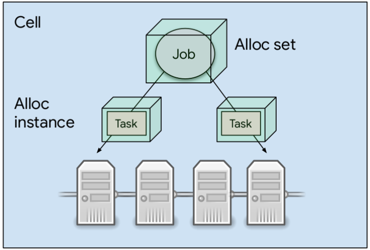
Borg

Borg

- Cluster management system at Google.

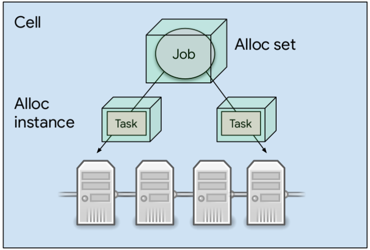
Borg Cell, Job, Task, and Alloc

- Cell: a set of machines managed by Borg as one unit.



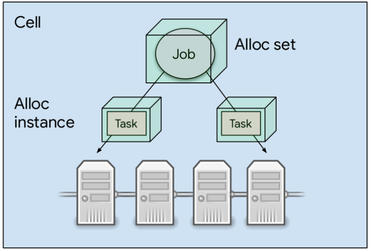
Borg Cell, Job, Task, and Alloc

- Cell: a set of machines managed by Borg as one unit.
- Job: users submit work in the form of jobs.



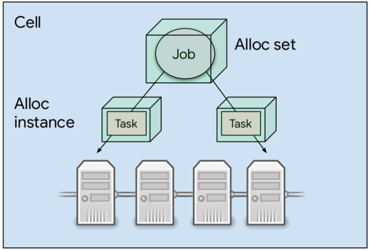
Borg Cell, Job, Task, and Alloc

- Cell: a set of machines managed by Borg as one unit.
- Job: users submit work in the form of jobs.
- Task: each job contains one or more tasks.



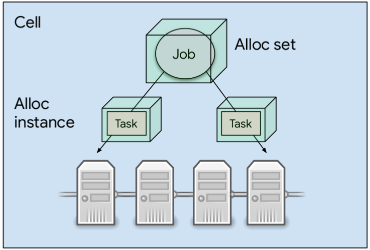
Borg Cell, Job, Task, and Alloc

- Cell: a set of machines managed by Borg as one unit.
- Job: users submit work in the form of jobs.
- Task: each job contains one or more tasks.
- Alloc: reserved set of resources and a job can run in an alloc set.



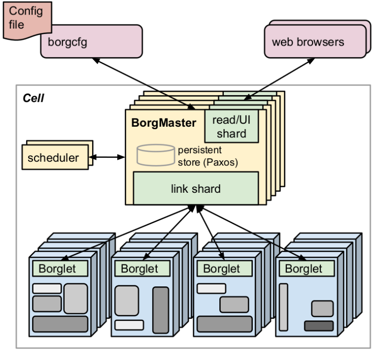
Borg Cell, Job, Task, and Alloc

- Cell: a set of machines managed by Borg as one unit.
- Job: users submit work in the form of jobs.
- Task: each job contains one or more tasks.
- Alloc: reserved set of resources and a job can run in an alloc set.
- Alloc instance: making each of its tasks run in an alloc instance.



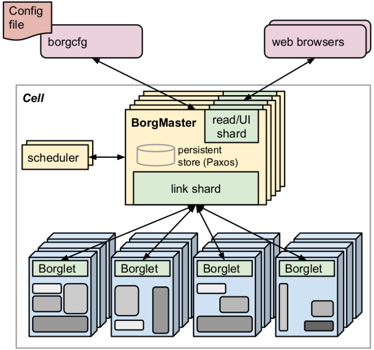
Borg Architecture

- BorgMaster
  - The central brain of the system
  - Holds the cluster state
  - Replicated for reliability (using paxos)
  - Scheduling: where to place tasks?



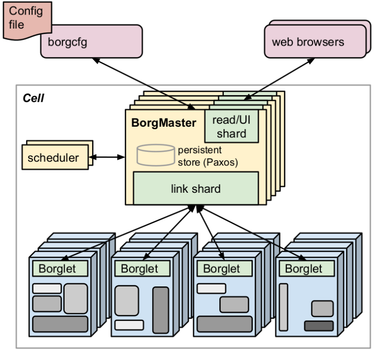
Borg Architecture

- BorgMaster
  - The central brain of the system
  - Holds the cluster state
  - Replicated for reliability (using paxos)
  - Scheduling: where to place tasks?
- Borglet
  - Manage and monitor tasks and resource
  - BorgMaster polls Borglet every few seconds



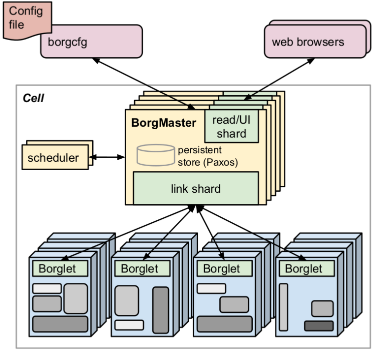
Borg Scheduler

- Feasibility checking: find machines for a given job



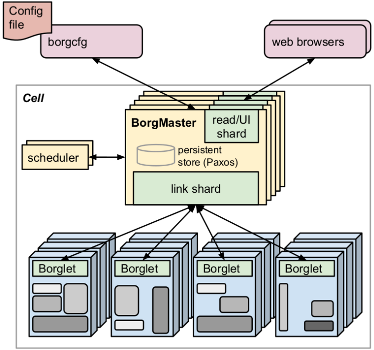
Borg Scheduler

- Feasibility checking: find machines for a given job
- Scoring: pick one machines



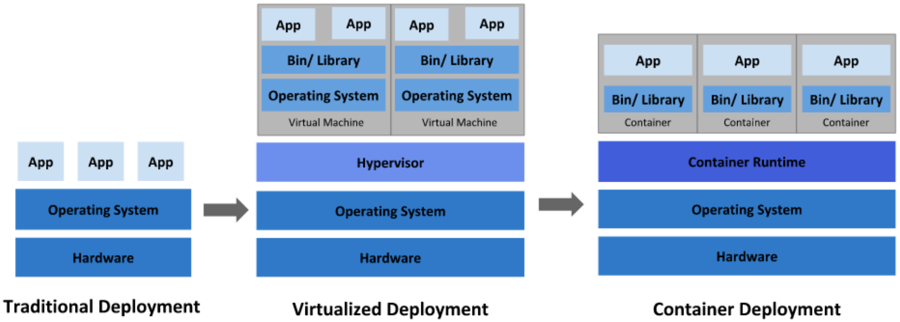
Borg Scheduler

- Feasibility checking: find machines for a given job
- Scoring: pick one machines
- According to the users prefs and built-in criteria



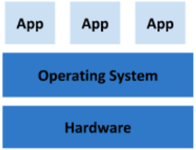
Docker and Kubernetes

Application Deployment



Traditional Deployment Era

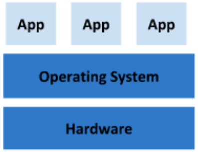
- Running applications on physical servers.



Traditional Deployment

Traditional Deployment Era

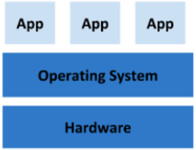
- Running applications on physical servers.
- No resource boundaries for applications in a physical server



Traditional Deployment

Traditional Deployment Era

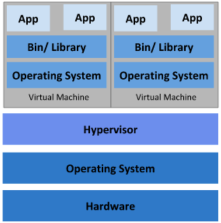
- Running applications on physical servers.
- No resource boundaries for applications in a physical server
- Resource allocation issues, e.g., one application would take up most of the resources, so the other applications would underperform.



Traditional Deployment

Virtualized Deployment Era

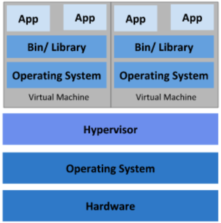
- Virtual Machines (VMs): a full machine running all the components, including its own operating system (OS), on top of the virtualized hardware.



Virtualized Deployment

Virtualized Deployment Era

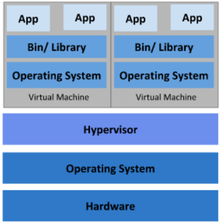
- Virtual Machines (VMs): a full machine running all the components, including its own operating system (OS), on top of the virtualized hardware.
- Virtualization allows to run multiple VMs on a single physical server's CPU.



Virtualized Deployment

Virtualized Deployment Era

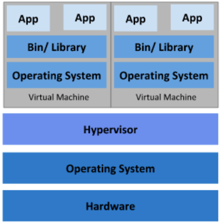
- Virtual Machines (VMs): a full machine running all the components, including its own operating system (OS), on top of the virtualized hardware.
- Virtualization allows to run multiple VMs on a single physical server's CPU.
  - Utilizes the resources of a physical server better.



Virtualized Deployment

Virtualized Deployment Era

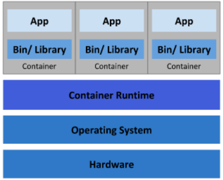
- Virtual Machines (VMs): a full machine running all the components, including its own operating system (OS), on top of the virtualized hardware.
- Virtualization allows to run multiple VMs on a single physical server's CPU.
  - Utilizes the resources of a physical server better.
  - Better scalability as applications can be added/updated easily.



Virtualized Deployment

Container Deployment Era

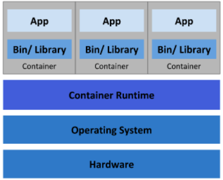
- Containers are similar to VMs, but they have relaxed isolation properties to share the OS among the applications.



Container Deployment

Container Deployment Era

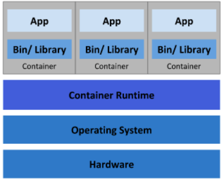
- Containers are similar to VMs, but they have relaxed isolation properties to share the OS among the applications.
- Similar to a VM, a container packages applications as images that contain everything needed to run them: code, runtime environment, libraries, and configuration.



Container Deployment

Container Deployment Era

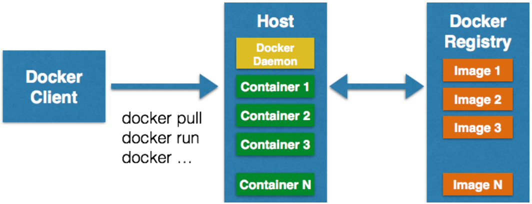
- Containers are similar to VMs, but they have relaxed isolation properties to share the OS among the applications.
- Similar to a VM, a container packages applications as images that contain everything needed to run them: code, runtime environment, libraries, and configuration.
- As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.



Container Deployment

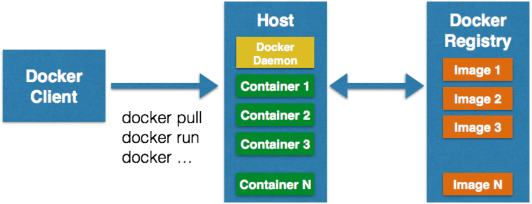
Docker

- Docker is a virtualization software.



Docker

- Docker is a virtualization software.
- A docker image is a template, and a container is a copy of that template.



Container Orchestration

- Container scalability is an operational challenge.

Container Orchestration

- Container scalability is an operational challenge.
- If we have 10 containers and four applications, it is not difficult to manage the deployment and maintenance of the containers.

Container Orchestration

- Container scalability is an operational challenge.
- If we have 10 containers and four applications, it is not difficult to manage the deployment and maintenance of the containers.
- But, what if we have 1000 containers and 400 services?

Container Orchestration

- Container scalability is an operational challenge.
- If we have 10 containers and four applications, it is not difficult to manage the deployment and maintenance of the containers.
- But, what if we have 1000 containers and 400 services?
- Container orchestration can help to manage the lifecycles of containers, especially in large and dynamic environments.

Container Orchestration

- Container scalability is an operational challenge.
- If we have 10 containers and four applications, it is not difficult to manage the deployment and maintenance of the containers.
- But, what if we have 1000 containers and 400 services?
- Container orchestration can help to manage the lifecycles of containers, especially in large and dynamic environments.
- Container orchestration tools: Kubernetes (based on Borg), Marathon (runs on Mesos)

Summary

Summary

- Mesos
  - Offered-based
  - Max-Min fairness: DRF
- YARN
  - Request-based
  - RM, AM, NM
- Borg
  - Request-based
  - BorgMaster, Borglet
  - Kubernetes

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
- A. Verma et al., "Large-scale cluster management at Google with Borg", EuroSys 2015

Questions?

Acknowledgements

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