### Where Are We?

<table>
<thead>
<tr>
<th>Data Processing</th>
<th></th>
<th>Machine Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Data</td>
<td>Pregel, GraphLab, PowerGraph GraphX, X-Streem, Chaos</td>
<td>Mlib Tensorflow</td>
</tr>
<tr>
<td>Structured Data</td>
<td>Spark SQL</td>
<td></td>
</tr>
<tr>
<td>Batch Data</td>
<td>MapReduce, Dryad FlumeJava, Spark</td>
<td>Storm, SEEP, Naiad, Spark Streaming, Flink, Millwheel, Google Dataflow</td>
</tr>
<tr>
<td>Streaming Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Storage</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed File Systems</td>
<td>GFS, Flat FS</td>
<td></td>
</tr>
<tr>
<td>NoSQL Databases</td>
<td>Dynamo, BigTable, Cassandra</td>
<td>Kafka</td>
</tr>
<tr>
<td>Distributed Messaging Systems</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resource Management</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesos, YARN</td>
<td></td>
</tr>
</tbody>
</table>
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 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 **framework** (e.g., Hadoop, Spark) manages and runs one or more **jobs**.

A **job** consists of one or more **tasks**.
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 vs. 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
Global Scheduler (1/2)

- **Job requirements**
  - Response time
  - Throughput
  - Availability

- **Job execution plan**
  - Task DAG
  - Inputs/outputs
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.
Unit of allocation: resource offer
- Vector of available resources on a node
- For example, node1: \((1\text{CPU}, 1\text{GB})\), node2: \((4\text{CPU}, 16\text{GB})\)

Master sends resource offers to frameworks.

Frameworks select which offers to accept and which tasks to run.
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.
Slaves continuously send status updates about resources to the Master.
Pluggable scheduler picks framework to send an offer to.
Framework scheduler selects resources and provides tasks.
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.

- 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.

- 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 $\langle 1 \text{CPU} \rangle$ per task
- User 2 has $y$ tasks and wants $\langle 2 \text{CPU} \rangle$ per task

\[
\begin{align*}
\text{max} & \ (x, y) \\
\text{subject to} & \ x + 2y \leq 20 \\
\text{so} & \ x = 10 \\
\text{and} & \ y = 5
\end{align*}
\]
Max-Min Fairness - Example

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

$$\max(x, y) \ (\text{maximize allocation})$$
Max-Min Fairness - Example

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

$$\max(x, y) \text{ (maximize allocation)}$$
subject to
\[ x + 2y \leq 20 \text{ (CPU constraint)} \]
\[ x = 2y \]
Max-Min Fairness - Example

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

$$\max(x, y) \text{ (maximize allocation)}$$
subject to
$$x + 2y \leq 20 \text{ (CPU constraint)}$$
$$x = 2y$$
so
$$x = 10$$
$$y = 5$$
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.
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?
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 $\langle 1\text{CPU} \rangle$ 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
Problem

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

▶ Multi-resource example
• 2 resources: CPUs and mem
• User 1 wants ⟨1CPU, 4GB⟩ per task
• User 2 wants ⟨2CPU, 1GB⟩ 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.

- For User 1:
  - Tasks: \(x\)
  - Per task: \(\langle 1\text{CPU}, 2\text{GB} \rangle\)

- For User 2:
  - Tasks: \(y\)
  - Per task: \(\langle 1\text{CPU}, 4\text{GB} \rangle\)

Asset fairness yields:

\[
\max(x, y) x + y \leq 28
\]

\[
2x + 4y \leq 56
\]

\[
2x = 3y
\]

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

User 2:
- \(y = 8\)
- \(\langle 28\%\text{CPU}, 57\%\text{GB} \rangle\) (\(\sum = 86\%\))
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 56 GB RAM (e.g., 1 CPU = 2 GB)
  - User 1 has \( x \) tasks and wants \( \langle 1 \text{CPU}, 2 \text{GB} \rangle \) per task
  - User 2 has \( y \) tasks and wants \( \langle 1 \text{CPU}, 4 \text{GB} \rangle \) per task

\[
\begin{align*}
\text{Asset fairness yields:} & \\
\max(x, y) & x + y \leq 28 \\
2x & + 4y \leq 56 \\
2x &= 3y
\end{align*}
\]

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

User 2: \( y = 8 \): \( \langle 28\% \text{CPU}, 57\% \text{GB} \rangle \) (\( \sum = 86\% \))
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 $\langle 1\text{CPU}, 2\text{GB} \rangle$ per task
  - User 2 has $y$ tasks and wants $\langle 1\text{CPU}, 4\text{GB} \rangle$ per task

- Asset fairness yields:

\[
\begin{align*}
\max(x, y) \\
x + y &\leq 28 \\
2x + 4y &\leq 56 \\
2x &= 3y \\
\text{User 1: } x &= 12: \langle 43\%\text{CPU}, 43\%\text{GB} \rangle \ (\sum = 86\%) \\
\text{User 2: } y &= 8: \langle 28\%\text{CPU}, 57\%\text{GB} \rangle \ (\sum = 86\%)
\end{align*}
\]
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: \(\langle 8\text{CPU}, 5\text{GB} \rangle\)
  - User 1 allocation: \(\langle 2\text{CPU}, 1\text{GB} \rangle\): \(\frac{2}{8} = 25\% \) CPU and \(\frac{1}{5} = 20\% \) 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: \( \langle 8 \text{CPU}, 5 \text{GB} \rangle \)
  - User 1 allocation: \( \langle 2 \text{CPU}, 1 \text{GB} \rangle \): \( \frac{2}{8} = 25\% \) CPU and \( \frac{1}{5} = 20\% \) 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.

```
• Total resources: ⟨9CPU, 18GB⟩
• User 1 wants ⟨1CPU, 4GB⟩; Dominant resource: RAM (1/9 < 4/18)
• User 2 wants ⟨3CPU, 1GB⟩; Dominant resource: CPU (3/9 > 1/18)
```

```
max(x, y) x + 3y ≤ 9
4x + y ≤ 18
9x = 3y
```

- User 1: x = 3: ⟨33% CPU, 66% GB⟩
- User 2: y = 2: ⟨66% CPU, 16% GB⟩

33 / 46
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 \(\frac{3}{9} > \frac{1}{18}\)
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 \( (\frac{3}{9} > \frac{1}{18}) \)

\[
\begin{align*}
\text{max}(x, y) \\
x + 3y &\leq 9 \\
4x + y &\leq 18 \\
\frac{4x}{18} &= \frac{3y}{9}
\end{align*}
\]

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 (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 ...)

![Diagram of YARN Architecture - Application Manager]
The worker daemon.

Registers with RM.

One per node.

Report resources to RM: memory, CPU, ...
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.
 Containers are described by a **Container Launch Context (CLC)**.

- The command necessary to create the process
- Environment variables
- Security tokens
- ...
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.
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.
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


Questions?

Acknowledgements

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