Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center

Ashvin Goel
Electrical and Computer Engineering
University of Toronto
ECE1724

Authors: Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica

Many slides adapted from Ion Stoica
Goals of Mesos

• Fine-grained data sharing
• Support diverse frameworks within a cluster
• High resource utilization
• High scalability, reliability
Fine-Grained Data Sharing

• Multiplex tasks on physical nodes
  • Tasks share CPU, memory, disk of physical node
  • Typical tasks are 10s of seconds to minutes

• Tasks may belong to different jobs

• Jobs may run on different frameworks
Fine-Grained Data Sharing Example

Today: static partitioning

- Hadoop
  - 33%
  - 17%
  - 0%

- Pregel
  - 33%
  - 17%
  - 0%

- MPI
  - 33%
  - 17%
  - 0%

Mesos: dynamic sharing

Shared cluster
Baseline Approach: Global Scheduler

Organization policies
Resource availability

Job requirements
- Response time
- Throughput
- Availability
- ...

Global Scheduler
Baseline Approach: Global Scheduler

Organization policies
Resource availability
Job requirements
Job execution plan
- Task DAG
- Inputs/outputs

Global Scheduler
Baseline Approach: Global Scheduler

Organization policies → Global Scheduler
Resource availability → Global Scheduler
Job requirements → Global Scheduler
Job execution plan → Global Scheduler
Estimates → Global Scheduler

- Task durations
- Input sizes
- Transfer sizes
Baseline Approach: Global Scheduler

- **Advantages:**
  - Can achieve optimal schedule

- **Disadvantages:**
  - Complex, hard to scale and ensure resilience
  - Hard to anticipate future framework requirements
  - Need to refactor existing frameworks
Mesos: Hierarchical Scheduler

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

- **Disadvantages:**
  - Distributed scheduling decision is not optimal
Mesos Approach: Resource Offers

- Master sends resource offers to frameworks
  - A resource offer is a vector of available resources on a node
    - E.g., node1: <1CPU, 1GB>, node2: <4CPU, 16GB>
- Frameworks choose whether to accept offer or not
  - On accepting offer, framework decides which tasks to run
  - Approach pushes task scheduling to frameworks
Mesos Example

Slaves continuously send status updates about resources

Slave S1
- 8CPU, 8GB

Slave S2
- 8CPU, 16GB

Slave S3
- 16CPU, 16GB

Mesos Master
- S1: <8CPU, 8GB>
- S2: <8CPU, 16GB>
- S3: <16CPU, 16GB>

Allocation Module

Pluggable scheduler, chooses framework to send an offer

Hadoop Scheduler

MPI Scheduler
Mesos Example

Slaves continuously send status updates about resources.

- **Slave S1:** 8CPU, 8GB
- **Slave S2:** 8CPU, 16GB
- **Slave S3:** 16CPU, 16GB

**Mesos Master**

<table>
<thead>
<tr>
<th>Slave</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>&lt;8CPU, 8GB&gt;</td>
</tr>
<tr>
<td>S2</td>
<td>&lt;8CPU, 16GB&gt;</td>
</tr>
<tr>
<td>S3</td>
<td>(S1:&lt;8CPU, 8GB&gt;, S2:&lt;8CPU, 16GB&gt;)</td>
</tr>
</tbody>
</table>

**Allocation Module**

- Pluggable scheduler, chooses framework to send an offer.

**Framework scheduler** selects resources and provides tasks.

- **Hadoop Scheduler**
- **MPI Scheduler**
Mesos Example

Slaves continuously send status updates about resources

Slave S1

8CPU, 8GB

Slave S2

8CPU, 16GB

Slave S3

16CPU, 16GB

Mesos Master

<table>
<thead>
<tr>
<th>Slave</th>
<th>Offer</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>&lt;6CPU,4GB&gt;</td>
</tr>
<tr>
<td>S2</td>
<td>&lt;4CPU,12GB&gt;</td>
</tr>
<tr>
<td>S3</td>
<td>&lt;16CPU,16GB&gt;</td>
</tr>
</tbody>
</table>

Allocation Module

Pluggable scheduler, chooses framework to send an offer

Framework scheduler selects resources and provides tasks

MPI Scheduler

(task1:{S1:<2CPU,4GB>}); task2:{S2:<4CPU,4GB>})
Slaves continuously send status updates about resources

Framework executor launch tasks and may persist across tasks

Framework scheduler selects resources and provides tasks

Mesos Example

<table>
<thead>
<tr>
<th>Slave S1</th>
<th>Executor</th>
</tr>
</thead>
<tbody>
<tr>
<td>task 1</td>
<td>8CPU, 8GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Slave S2</th>
<th>Executor</th>
</tr>
</thead>
<tbody>
<tr>
<td>task 2</td>
<td>8CPU, 16GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Slave S3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16CPU, 16GB</td>
<td></td>
</tr>
</tbody>
</table>

Mesos Master

| S1 | <6CPU, 4GB> |
| S2 | <4CPU, 12GB> |
| task 1 | | task 2: <4CPU, 4GB> |

Allocation Module

Pluggable scheduler, chooses framework to send an offer

Hadoop Scheduler

MPI Scheduler
Mesos Example

Slaves continuously send status updates about resources.

Framework executors launch tasks and may persist across tasks.

Framework scheduler selects resources and provides tasks.

Slaves S1, S2, and S3:
- **Slave S1**
  - Task 1
  - Executor
  - Resources: 8 CPU, 8 GB
- **Slave S2**
  - Task 2
  - Executor
  - Resources: 8 CPU, 16 GB
- **Slave S3**
  - Resources: 16 CPU, 16 GB

Mesos Master:
- Allocation Module
  - Resources:
    - S1: 6 CPU, 4 GB
    - S2: 4 CPU, 12 GB
    - S3: 8 CPU, 4 GB, 8 GB

Pluggable scheduler, chooses framework to send an offer.

Hadoop Scheduler

MPI Scheduler
Mesos Example

Slaves continuously send status updates about resources

Framework executors launch tasks and may persist across tasks

Framework scheduler selects resources and provides tasks

Pluggable scheduler, chooses framework to send an offer

| Slave S1 | task 1 Executor | 8CPU, 8GB |
| Slave S2 | task 2 Executor | 8CPU, 16GB |
| Slave S3 | | 16CPU, 16GB |

Mesos Master

| S1 | <2CPU,2GB> |
| S2 | <4CPU,12GB> |
| S3 | <16CPU,16GB> |

Allocation Module

Hadoop Scheduler

MPI Scheduler

((task1:S1:<4CPU,2GB>))
Mesos Example

Slaves continuously send status updates about resources.

Framework executors launch tasks and may persist across tasks.

Framework scheduler selects resources and provides tasks.

Slave S1
- task 1 Executor
  - 8CPU, 8GB

Slave S2
- task 2 Executor
  - 8CPU, 16GB

Slave S3
- 16CPU, 16GB

Allocation Module

Pluggable scheduler, chooses framework to send an offer.

Hadoop Scheduler

MPI Scheduler
Why Do Resource Offers Work?

• A framework can just wait for an offer that matches its constraints!
  • It can reject offers it does not like
• Example: Hadoop’s job input is blue file

Accept: both S2 and S3 store the blue file
(task1:[S2:<...>]; task2:[S3:<...>])
Optimization: Filters

• Frameworks can short-circuit rejection by providing a predicate on resources to be offered
  • E.g., offer me “nodes from list L” or “nodes with > 8 GB RAM”
• Ability to reject still ensures correctness when needs cannot be expressed using filters
## Mesos API

<table>
<thead>
<tr>
<th>Scheduler Callbacks</th>
<th>Scheduler Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceOffer(offerId, offers)</td>
<td>replyToOffer(offerId, tasks)</td>
</tr>
<tr>
<td>offerRescinded(offerId)</td>
<td>setNeedsOffers(bool)</td>
</tr>
<tr>
<td>statusUpdate(taskId, status)</td>
<td>setFilters(filters)</td>
</tr>
<tr>
<td>slaveLost(slaveId)</td>
<td>getGuaranteedShare()</td>
</tr>
<tr>
<td></td>
<td>killTask(taskId)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Executor Callbacks</th>
<th>Executor Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>launchTask(taskDescriptor)</td>
<td>sendStatus(taskId, status)</td>
</tr>
<tr>
<td>killTask(taskId)</td>
<td></td>
</tr>
</tbody>
</table>
Failure Recovery

- Mesos master only has soft state
  - List of currently running frameworks, slave nodes and tasks
- Master uses zookeeper for leader election
- After master failure, new master rebuilds state when frameworks and slaves re-register with new master
- Fault detection and recovery in ~10 sec
Evaluation

100 node cluster

Time (s)

Share of Cluster CPUs

Spark

Facebook Hadoop Mix

Large Hadoop Mix

Torque / MPI
Scalability

- Mesos only performs inter-framework scheduling (e.g., fair sharing), easier than intra-framework scheduling
- Result: Scaled to 50,000 emulated slaves, 200 frameworks, 100K tasks (30s task length)
Conclusions

• Mesos shares cluster among different frameworks efficiently with two key design ideas
  • Fine-grained sharing at the level of tasks
  • Use hierarchical scheduling
    • Resource manager offers resources to frameworks
    • Frameworks control their own task scheduling

• Enables co-existence of current frameworks and development of newer ones

• Hundreds of deployments in productions
  • E.g., Twitter, GE, Apple
  • Managing 10K node datacenters!
Discussion
Q1

- Mesos offers resources to frameworks, and frameworks accept or reject these offers. For what types of frameworks will this approach work well?
Q2

• Will the Resource Offers approach work well for tasks with small versus large resource requirements? (think about a restaurant that needs to serve small versus large groups)
Q3

• How would you handle the problem in the previous slide?
Q4

• How can Mesos handle a framework that delays responding to a Resource Offer?
Q5

• How can Mesos handle a framework that never releases resources?