Mesos: Multiprogramming for Datacenters

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

- Rapid innovation in cloud computing

Today

- No single framework optimal for all applications
- Each framework runs on its dedicated cluster or cluster partition
Computation Model: Frameworks

- A **framework** (e.g., Hadoop, MPI) manages one or more **jobs** in a computer cluster.
- A **job** consists of one or more **tasks**.
- A **task** (e.g., map, reduce) is implemented by one or more processes running on a single machine.

![Diagram of a computer cluster with tasks and jobs](image)

- **Job 1**: tasks 1, 2, 3, 4
- **Job 2**: tasks 5, 6, 7
One Framework Per Cluster Challenges

- Inefficient resource usage
  - E.g., Hadoop cannot use available resources from Pregel’s cluster
  - No opportunity for stat. multiplexing

- Hard to share data
  - Copy or access remotely, expensive

- Hard to cooperate
  - E.g., Not easy for Pregel to use graphs generated by Hadoop

Need to run multiple frameworks on same cluster
Solution: Mesos

- Common resource sharing layer
  - abstracts (“virtualizes”) resources to frameworks
  - enable diverse frameworks to share cluster
Mesos Goals

- High utilization of resources
- Support diverse frameworks (existing & future)
- Scalability to 10,000’s of nodes
- Reliability in face of node failures

- Focus of this talk: resource management & scheduling
Approach: Global Scheduler

Organization policies
Resource availability

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

Global Scheduler
Approach: Global Scheduler

- Organization policies
- Resource availability
- Job requirements
  - Job execution plan
    - Task DAG
    - Inputs/outputs
Approach: Global Scheduler

- Organization policies
- Resource availability
- Job requirements
- Job execution plan
- Estimates:
  - Task durations
  - Input sizes
  - Transfer sizes
Approach: Global Scheduler

- Advantages: can achieve optimal schedule
- Disadvantages:
  - Complexity → hard to scale and ensure resilience
  - Hard to anticipate future frameworks’ requirements
  - Need to refactor existing frameworks
Our Approach: Distributed Scheduler

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

- **Disadvantages:**
  - Distributed scheduling decision → not optimal
Resource Offers

- Unit of allocation: *resource offer*
  - Vector of available resources on a node
  - E.g., node1: <1CPU, 1GB>, node2: <4CPU, 16GB>
- Master sends resource offers to frameworks
- Frameworks select which offers to accept and which tasks to run
Mesos Architecture: 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:**
- **Slave S1:**
  - Task 1: Hadoop Executor
  - 8CPU, 8GB

- **Slave S2:**
  - Task 1: Hadoop Executor
  - 8CPU, 16GB

- **Slave S3:**
  - Task 2: Hadoop Executor
  - 16CPU, 16GB

**AllocaEon Module**

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

**Task Map:**
- (task1:[S1:<2CPU,4GB>]; task2:[S2:<4CPU,4GB>])
- ([task1:S1:<4CPU,2GB>]; task1:[S1:<2CPU,4GB>]; task2:[S2:<4CPU,4GB>])

**Pluggable scheduler**
- To pick framework to send an offer to.
Why does it Work?

- A framework can just wait for an offer that matches its constraints or preferences!
  - 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:<...>])
Two Key Questions

- How long does a framework need to wait?
- How do you allocate resources of different types?
Two Key Questions

- How long does a framework need to wait?
- How do you allocate resources of different types?
Single Resource: Fair Sharing

- $n$ users want to share a resource (e.g. CPU)
  - Solution: give each $1/n$ 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%
Why Max-Min Fairness?

<table>
<thead>
<tr>
<th>Policy</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proportional Allocation</td>
<td>User 1 gets weight 2, user 2 weight 1</td>
</tr>
<tr>
<td>Priority</td>
<td>Give user 1 weight 1000, user 2 weight 1</td>
</tr>
<tr>
<td>Reservation</td>
<td>Ensure user 1 gets 10% of a resource</td>
</tr>
<tr>
<td></td>
<td>Give user 1 weight 10, sum weights ≤ 100</td>
</tr>
<tr>
<td>Deadline Guarantees</td>
<td>Given a user job’s demand and deadline, compute user’s reservation/weight</td>
</tr>
</tbody>
</table>

**Isolation:** Users cannot affect others beyond their share
Widely Used

- **OS**: proportional sharing, lottery, Linux’s cfs, ...

- **Networking**: wfq, wf2q, sfq, drr, csfq, ...

- **Datacenters**: Hadoop’s fair sched, capacity sched, Quincy
Why is Max-Min Fairness Not Enough?

- Job scheduling is not only about a *single* resource
  - Tasks consume CPU, memory, network and disk I/O
Problem

- 2 resources: CPUs & mem
- User 1 wants <1 CPU, 4 GB> per task
- User 2 wants <3 CPU, 1 GB> per task
- What’s a fair allocation?
A Natural Policy

- **Asset Fairness**
  - Equalize each user’s *sum of resource shares*

- Cluster with 28 CPUs, 56 GB RAM
  - $U_1$ needs <1 CPU, 2 GB RAM> per task, or <3.6% CPUs, 3.6% RAM> per task
  - $U_2$ needs <1 CPU, 4 GB RAM> per task, or <3.6% CPUs, 7.2% RAM> per task

**Problem:** violates share guarantee
User 1 has < 50% of both CPUs and RAM
Better off in a separate cluster with half the resources

- **Asset fairness yields**
  - $U_1$: 12 tasks: <43% CPUs, 43% RAM> ($\Sigma=86\%$)
  - $U_2$: 8 tasks: <28% CPUs, 57% RAM> ($\Sigma=86\%$)
Cheating the Scheduler

- Users willing to *game* the system to get more resources

- Real-life examples
  - A cloud provider had quotas on map and reduce slots
    - Some users found out that the map-quota was low
  - **Users implemented maps in the reduce slots!**

  - A search company provided dedicated machines to users that could ensure certain level of utilization (e.g. 80%)
  - **Users used busy-loops to inflate utilization**
Challenge

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

- Can we generalize max-min fairness to multiple resources?
Dominant Resource Fairness (DRF)

- A user’s dominant resource is the resource user has the biggest share of
  - Example:
    - Total resources: <8 CPUs, 5 GB>
    - User 1’s allocation: <2 CPUs, 1 GB>
    - Dominant resource of User 1 is CPU (as 25% > 20%)

- A user’s dominant share is the fraction of the dominant resource she is allocated
  - User 1’s dominant share is 25%
Dominant Resource Fairness (DRF)

- Apply max-min fairness to dominant shares
- Equalize the dominant share of the users

Example:
Total resources: <9 CPU, 18 GB>
User 1 demand: <1 CPU, 4 GB>; dom res: mem (1/9 < 4/18)
User 2 demand: <3 CPU, 1 GB>; dom res: CPU (3/9 > 1/18)
Online DRF Scheduler

Whenever there are available resources and tasks to run:
*Schedule a task to the user with smallest dominant share*
### Properties of Policies

<table>
<thead>
<tr>
<th>Property</th>
<th>Asset</th>
<th>CEEI</th>
<th>DRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Share guarantee</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Strategy-proofness</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Envy-freeness</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Single resource fairness</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Bottleneck res. fairness</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Population monotonicity</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Resource monotonicity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Conjecture:** Assuming non-zero demands, DRF is the *only* allocation that is strategy proof and provides sharing incentive *(Eric Friedman, Cornell)*
Implementation Stats

- 20,000 lines of C++
- Master failover using ZooKeeper
- Isolation using Linux Containers
- Frameworks ported: Hadoop, MPI, Torque
- New specialized framework: Spark, for iterative jobs (up to 30× faster than Hadoop)

Open source in Apache Incubator
Users

- **Twitter** uses Mesos on > 100 nodes in production to run ~12 production services
- **Berkeley** machine learning researchers are running various algorithms at scale on Spark
- **Conviva** is using Spark for data analytics
- **UCSF** medical researchers are using Mesos to run Hadoop for bioinformatics apps
Dynamic Resource Sharing

- 100 node cluster

![Graph showing share of cluster CPUs over time for different tasks: Spark, Facebook Hadoop Mix, Large Hadoop Mix, and Torque / MPI. The graph includes time (s) on the x-axis and share of cluster CPUs on the y-axis. Specific time points are marked at 400 and 600 seconds.]
Conclusion

- Mesos shares clusters efficiently and dynamically among diverse frameworks
- Enable co-existence of current frameworks and development of new specialized ones
- In use at Twitter, UC Berkeley, Conviva and UCSF

www.mesosproject.org