Mesos

sharing the cluster.


University of California, Berkeley
observation. Many distributed apps; no single one optimal for all use cases.
Want to run multiple applications in a single cluster
...to maximize utilization
...to share data
solution. Mesos, a common layer over which diverse applications can run.
Multiple instances of the *same* application

Build *specialized applications* targeting particular problem domains
Build specialized applications
goals.

High utilization of resources

Diverse applications

Scalability to 50-100K nodes

Fault tolerance
resulting design. Small microkernel-like core
**design element 1.** Fine-grained sharing
Coarse-Grained Sharing (HPC):

Framework 1

Framework 2

Framework 3

Storage System (e.g. HDFS)

Fine-Grained Sharing (Mesos):

Fw. 1

Fw. 2

Fw. 3

Storage System (e.g. HDFS)

+ Improved utilization, responsiveness, data locality
**design element 2.** Resource offers (vs. global scheduler)
global scheduler.
Frameworks express needs in a specification language, global scheduler matches them to resources

+ Can make optimal decisions
– Complex: language must support all framework needs
– Difficult to scale and to make robust
– Future frameworks may have unanticipated needs
mesos: resource offers.
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
- Decentralized decisions might not be optimal
Pick framework to offer resources to

Framework-specific scheduling

Mesos master

Resource offer

Mesos slave

MPI executor

MPI executor

Task

Task
Resource offer =
list of (node, availableResources)

E.g. { (node1, <2 CPUs, 4 GB>),
       (node2, <3 CPUs, 2 GB>) }
Pick framework to offer resources to

Launches and isolates executors

Framework-specific scheduling
200+ nodes running ~12 production services (stream processing)

Genomics researchers using Hadoop and Spark on Mesos

Spark in use by Y! research

Spark for Analytics

Hadoop, running Experiments, Spark for ML Grad students for research

Cutting Edge Research!
evidence that sharing helps.
vs. static allocation (each app gets 25% of nodes)

<table>
<thead>
<tr>
<th>Framework</th>
<th>Speedup on Mesos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook Hadoop Mix</td>
<td>1.14×</td>
</tr>
<tr>
<td>Large Hadoop Mix</td>
<td>2.10×</td>
</tr>
<tr>
<td>Spark</td>
<td>1.26×</td>
</tr>
<tr>
<td>Torque / MPI</td>
<td>0.96×</td>
</tr>
</tbody>
</table>
let’s talk!

(actually we have been)

about...

1. Lessons learned: ours, Tashi’s, others
2. Long running services
3. Socialized services (HDFS, HBase)
4. Priority scheduling
mesos-project.org
Backup Slides
Deployments
What’s running?

Stream processing (tweets from the “firehose”)
  » Detecting spam and malicious users
    - Duplicate detection
    - Repeat offender detection
    - Spam account creation banning
    - ... more
  » Trend identification
Conviva

35% of all analytics jobs in Spark on Mesos
Spark jobs running on data in Hive
Porting Hive on top of Spark (Shark)
UCSF

Goal: run Hadoop and Spark to analyze genomic data for UCSF Medical Center

Scale out from 10 dedicated nodes onto ~1000 node HPC cluster running Sun Grid Engine

» Nodes allocated through SGE join Mesos cluster
Data Locality with Resource Offers

Ran 16 instances of Hadoop on a shared HDFS cluster

Used delay scheduling [EuroSys ’10] in Hadoop to get locality (wait a short time to acquire data-local nodes)

**Local Map Tasks (%)**

<table>
<thead>
<tr>
<th></th>
<th>Static Partitioning</th>
<th>Mesos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Map Tasks (%)</td>
<td>0%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**Job Duration (s)**

<table>
<thead>
<tr>
<th></th>
<th>Static Partitioning</th>
<th>Mesos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Duration (s)</td>
<td>600</td>
<td>1.7  ×</td>
</tr>
</tbody>
</table>
Scalability

Mesos only performs *inter-framework* scheduling (e.g. fair sharing), which is easier than intra-framework scheduling.

Result:
Scaled to 50,000 emulated slaves, 200 frameworks, 100K tasks (30s len)
Fault Tolerance

Mesos master has only *soft state*: list of currently running frameworks and tasks

Rebuild when frameworks and slaves re-register with new master after a failure

**Result:** fault detection and recovery in ~10 sec
Related Work

HPC schedulers (e.g. Torque, LSF, Sun Grid Engine)
  » Coarse-grained sharing for inelastic jobs (e.g. MPI)

Virtual machine clouds
  » Coarse-grained sharing similar to HPC

Condor
  » Centralized scheduler based on matchmaking

Parallel work: Next-Generation Hadoop
  » Redesign of Hadoop to have per-application masters
  » Also aims to support non-MapReduce jobs
  » Based on resource request language with locality prefs
Conclusion

Mesos shares clusters efficiently among diverse frameworks thanks to two design elements:

» **Fine-grained sharing** at the level of tasks
» **Resource offers**, a scalable mechanism for application-controlled scheduling

Enables co-existence of current frameworks and development of new specialized ones

In use at Twitter, UC Berkeley, Conviva and UCSF
Framework Isolation

Mesos uses OS isolation mechanisms, such as Linux containers and Solaris projects.

Containers currently support CPU, memory, IO and network bandwidth isolation.

Not perfect, but much better than no isolation.
Analysis

Resource offers work well when:
  » Frameworks can scale up and down elastically
  » Task durations are homogeneous
  » Frameworks have many preferred nodes

These conditions hold in current data analytics frameworks (MapReduce, Dryad, ...)
  » Work divided into short tasks to facilitate load balancing and fault recovery
  » Data replicated across multiple nodes
Revocation

Mesos allocation modules can revoke (kill) tasks to meet organizational SLOs

Framework given a grace period to clean up

“Guaranteed share” API lets frameworks avoid revocation by staying below a certain share
# 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>