Mesos

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

Benjamin Hindman, Andy Konwinski, **Matei Zaharia**,
Ali Ghodsi, Anthony Joseph, Randy Katz, Scott Shenker, Ion Stoica

University of California, Berkeley

Background

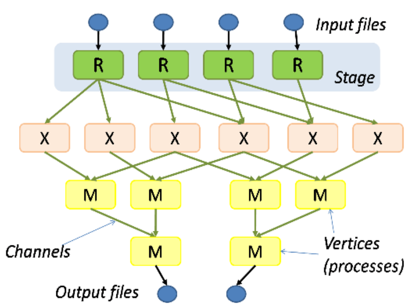
Rapid innovation in cluster computing frameworks

hadoop MapReduce

Google Pregel

Pig

RAILS



Dryad

CIEL

S4 distributed stream computing platform

Google Percolator

MPI-2

Problem

Rapid innovation in cluster computing frameworks

No single framework optimal for all applications

Want to run multiple frameworks in a single cluster

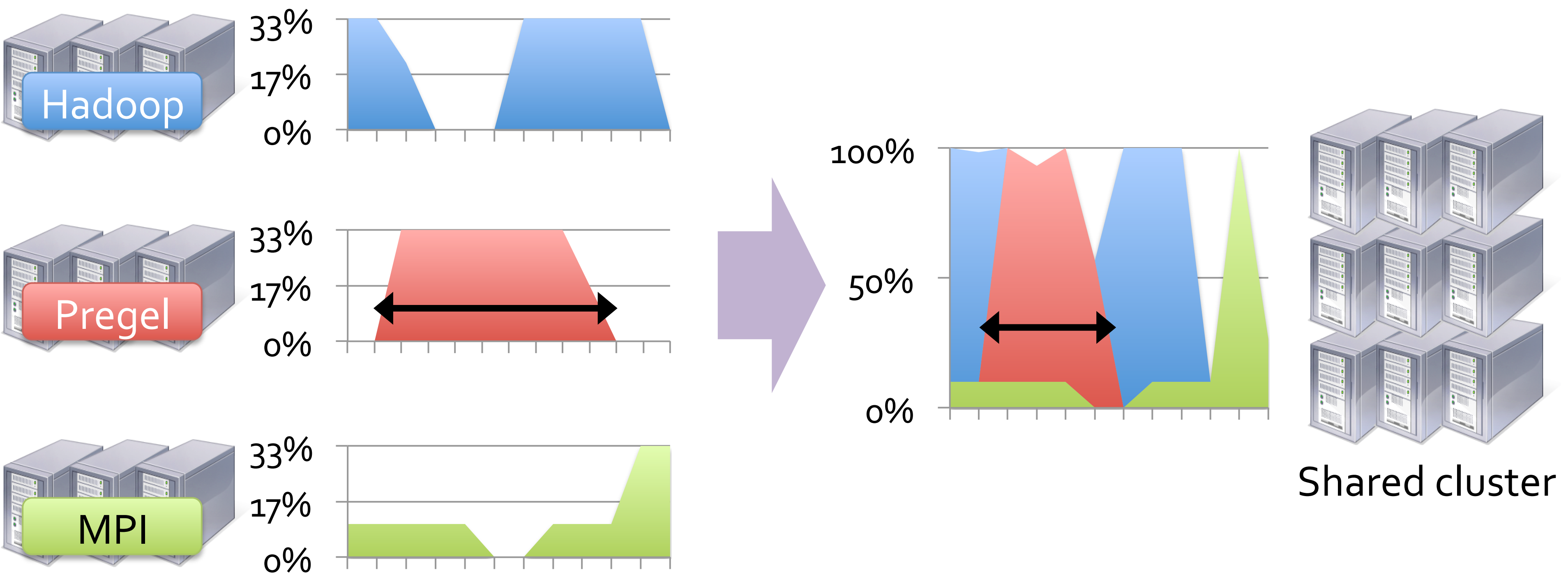
» …to *maximize utilization*

» …to *share data* between frameworks

Where We Want to Go

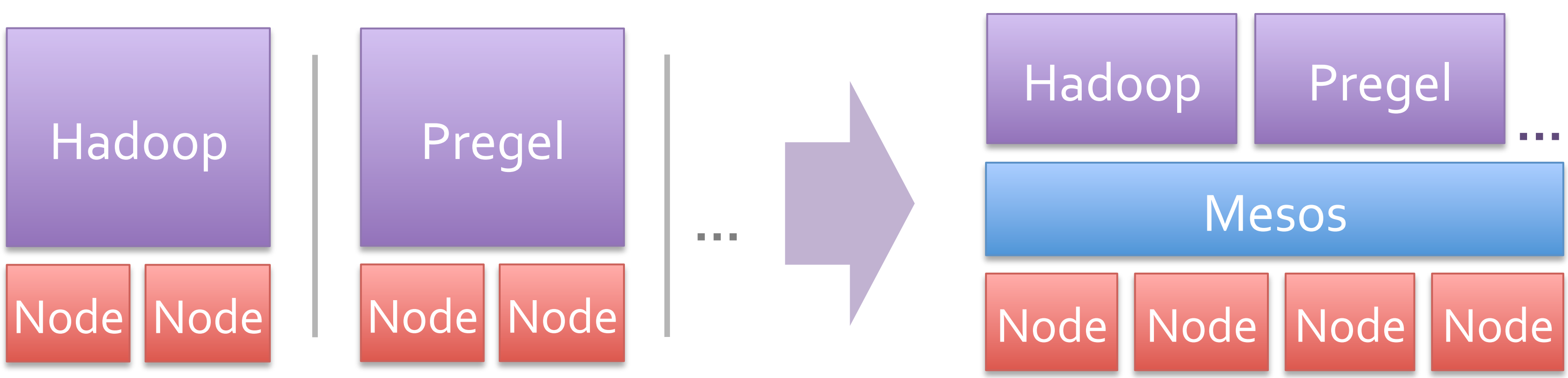
Today: static partitioning

Mesos: dynamic sharing



Solution

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



Other Benefits of Mesos

Run multiple instances of the *same* framework

» Isolate production and experimental jobs

» Run multiple versions of the framework concurrently

Build *specialized frameworks* targeting particular problem domains

» Better performance than general-purpose abstractions

Outline

Mesos Goals and Architecture

Implementation

Results

Related Work

Mesos Goals

High utilization of resources

Support diverse frameworks (current & future)

Scalability to 10,000's of nodes

Reliability in face of failures

Resulting design: Small microkernel-like core that pushes scheduling logic to frameworks

Design Elements

Fine-grained sharing:

» Allocation at the level of *tasks* within a job

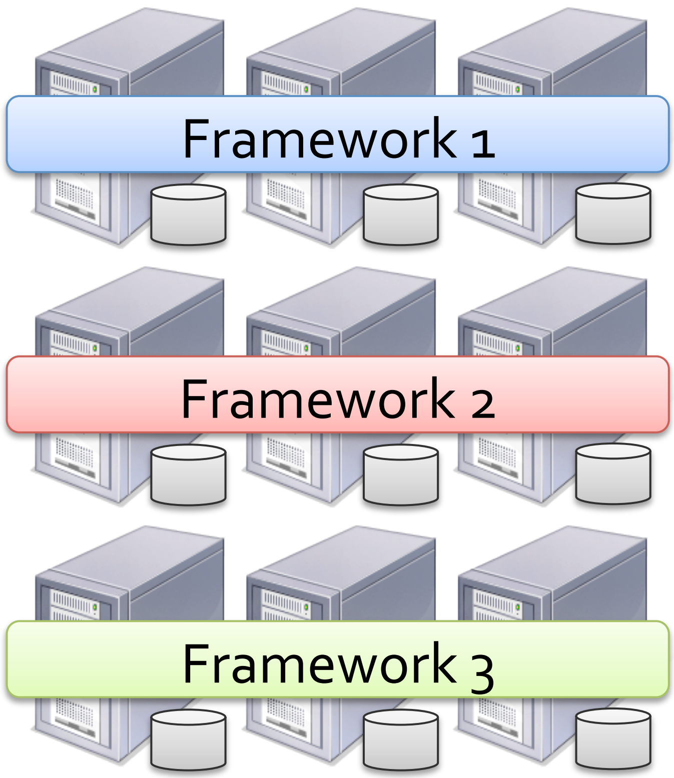
» Improves utilization, latency, and data locality

Resource offers:

» Simple, scalable application-controlled scheduling mechanism

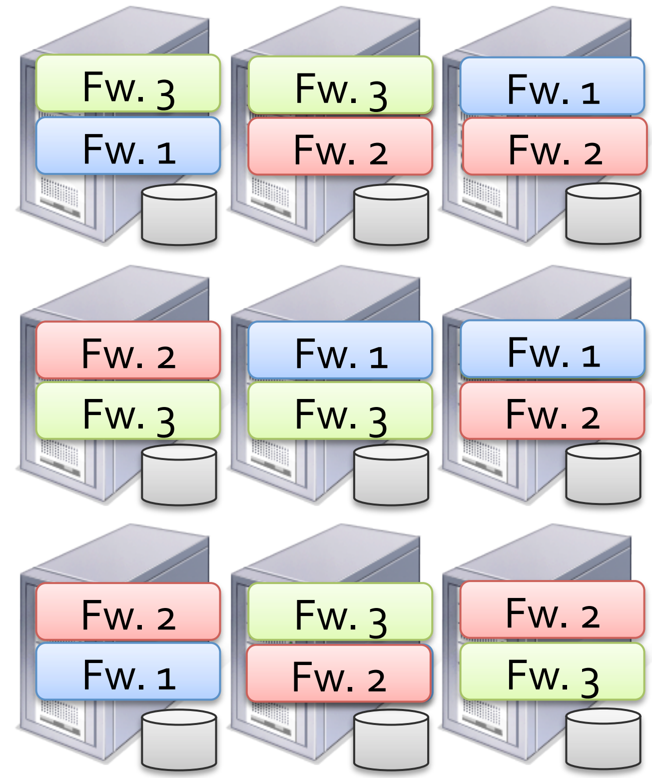
Element 1: Fine-Grained Sharing

Coarse-Grained Sharing (HPC):



Storage System (e.g. HDFS)

Fine-Grained Sharing (Mesos):



Storage System (e.g. HDFS)

+ Improved utilization, responsiveness, data locality

Element 2: Resource Offers

Option: 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

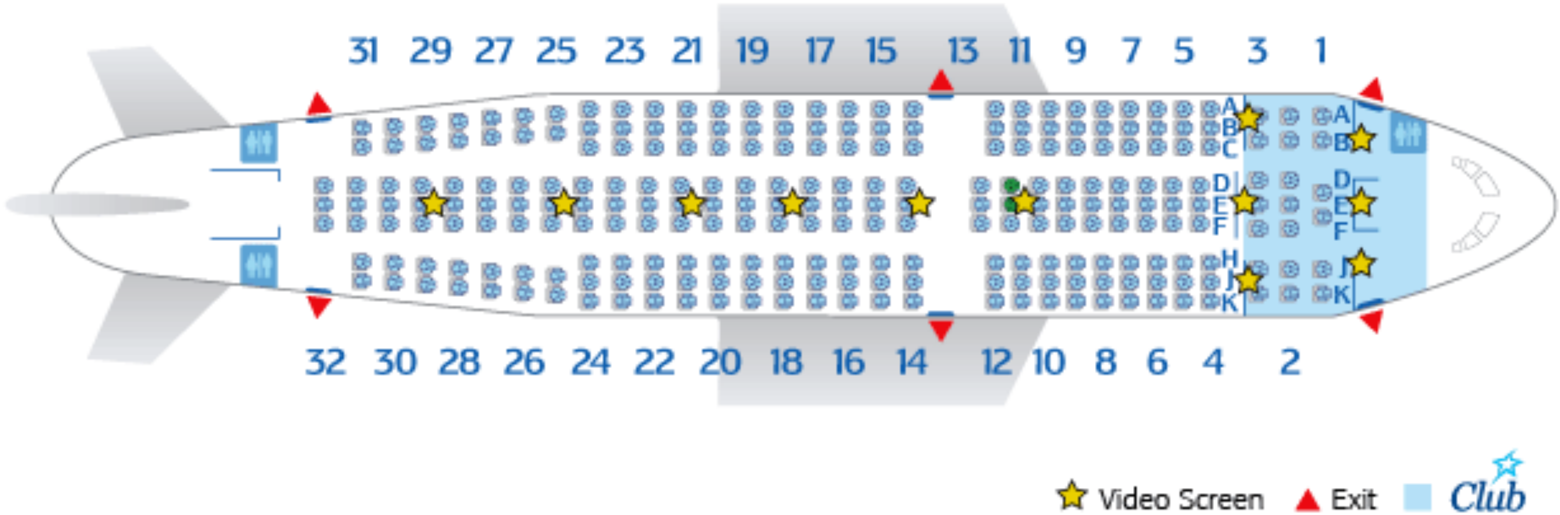
Element 2: Resource Offers

Mesos: Resource offers

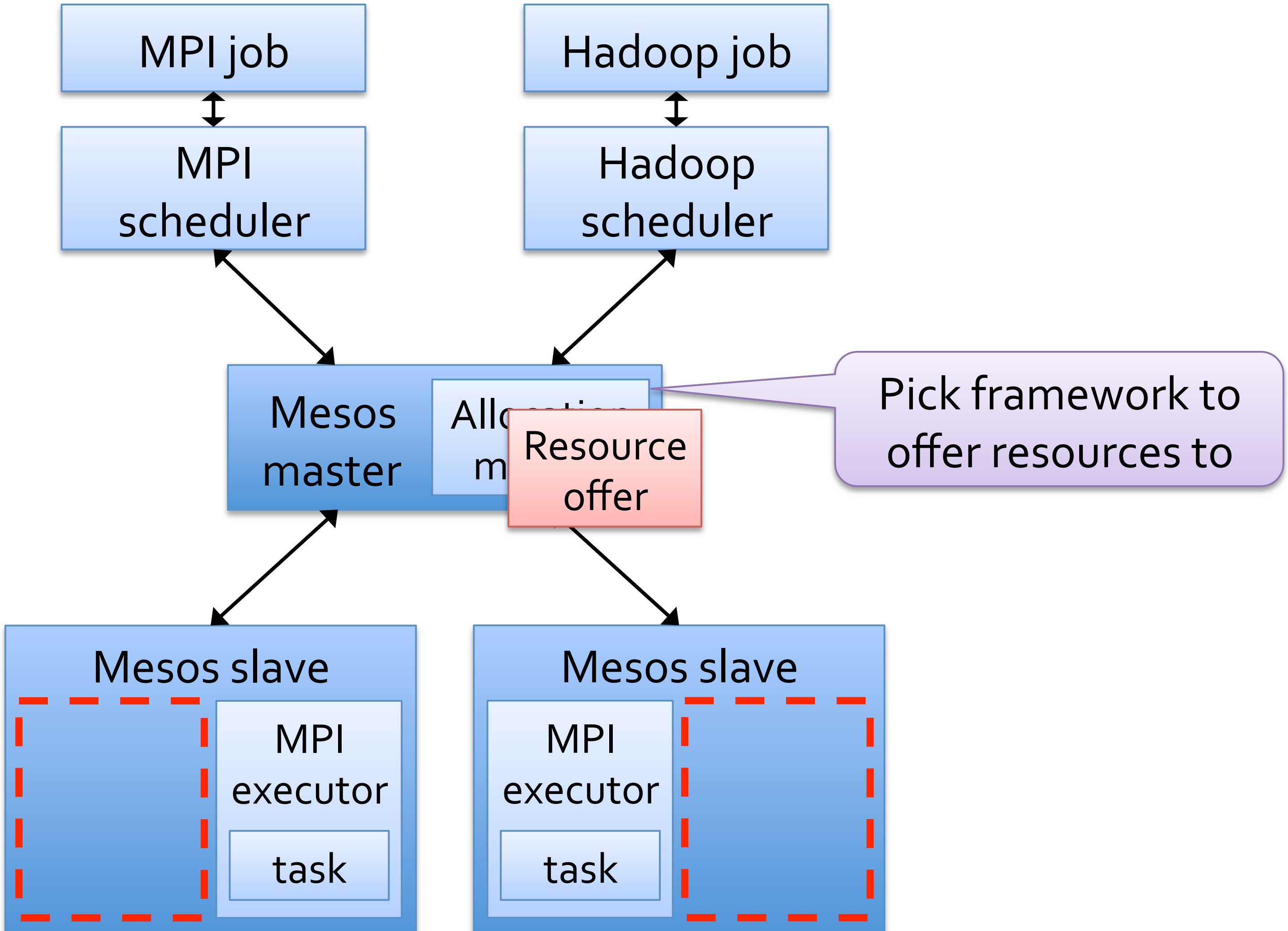
» Offer available resources to frameworks, let them pick which resources to use and which tasks to launch

+ Keeps Mesos simple, lets it support future frameworks

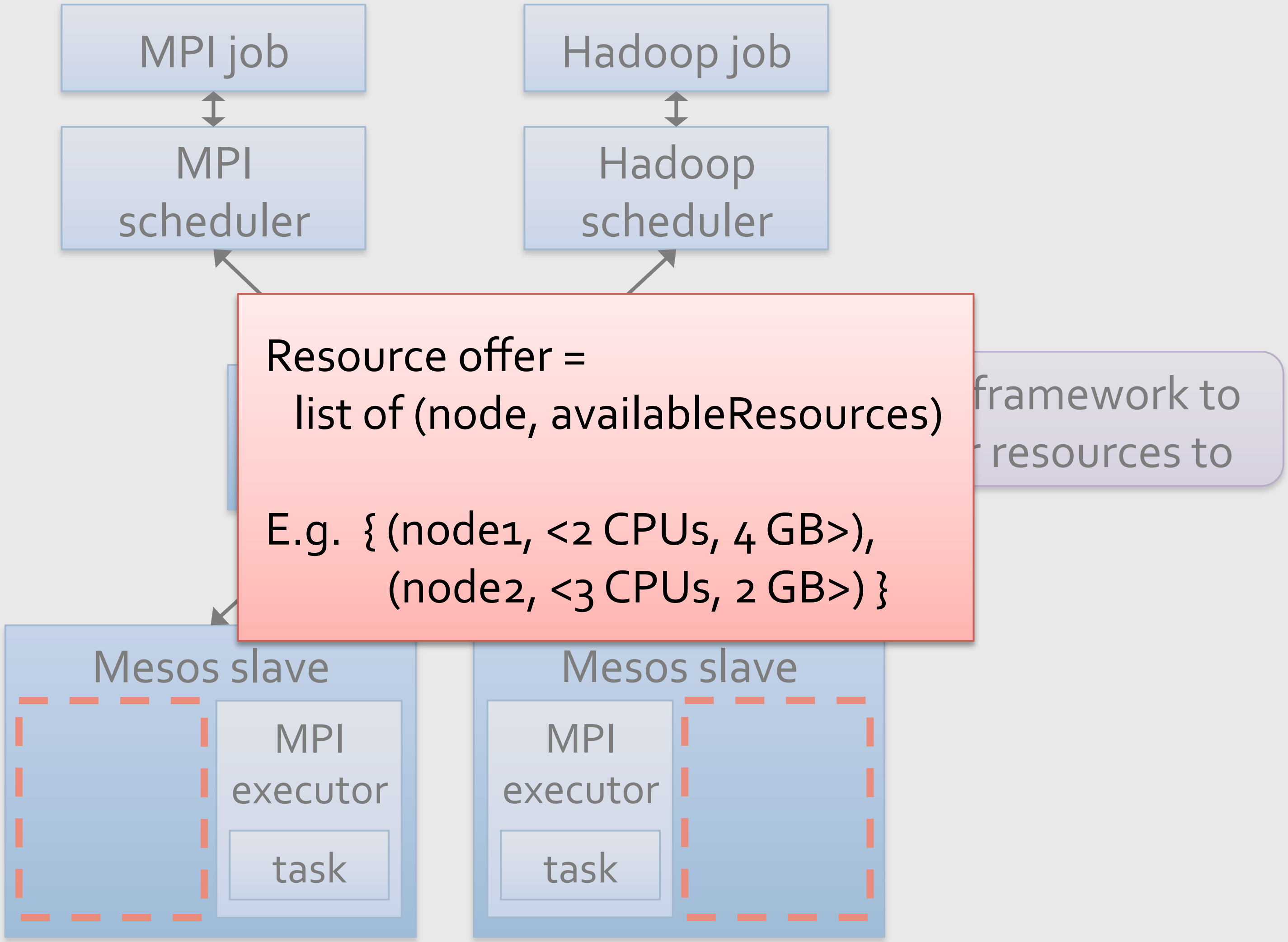
− Decentralized decisions might not be optimal



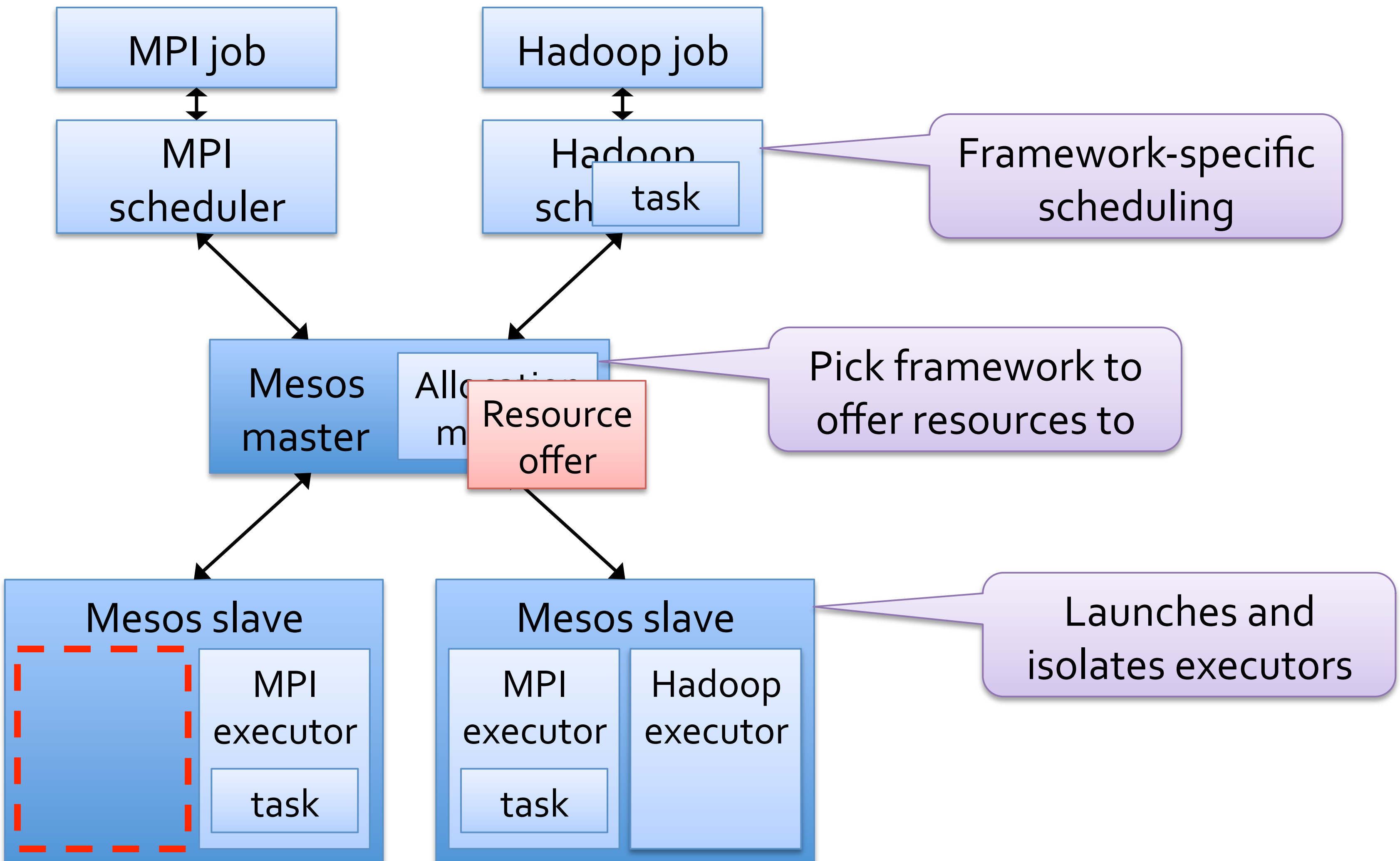
Mesos Architecture



Mesos Architecture



Mesos Architecture



Optimization: Filters

Let frameworks short-circuit rejection by providing a predicate on resources to be offered

» E.g. “nodes from list L” or “nodes with > 8 GB RAM”

» Could generalize to other hints as well

Ability to reject still ensures *correctness* when needs cannot be expressed using filters

Implementation

Implementation Stats

20,000 lines of C++

Master failover using ZooKeeper

Frameworks ported: Hadoop, MPI, Torque

New specialized framework: Spark, for iterative jobs (up to 20× faster than Hadoop)

Open source in Apache Incubator

Users

Twitter uses Mesos on > 100 nodes to run ~12 production services (mostly stream processing)

Berkeley machine learning researchers are running several algorithms at scale on Spark

Conviva is using Spark for data analytics

UCSF medical researchers are using Mesos to run Hadoop and eventually non-Hadoop apps

Results

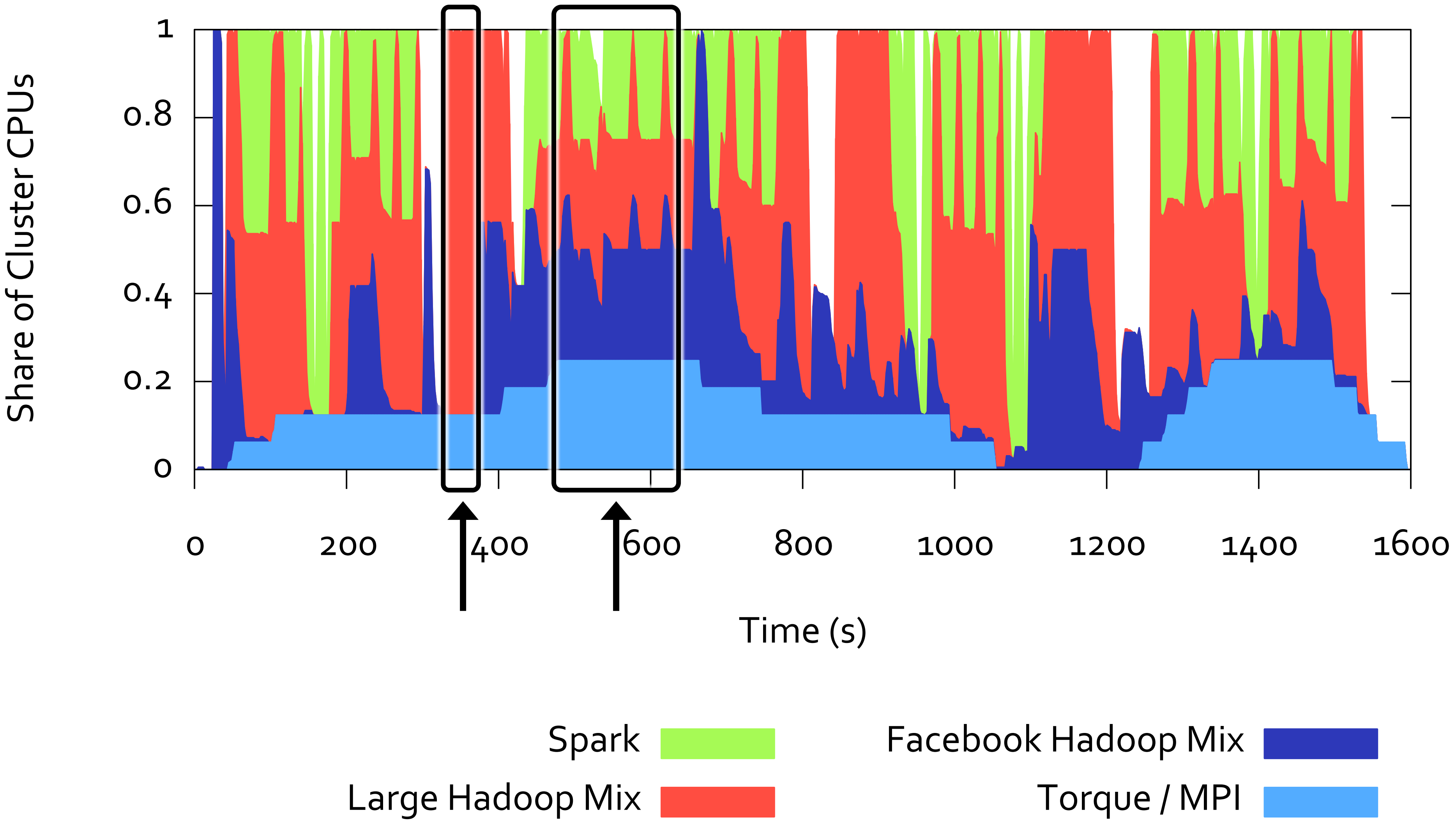
» Utilization and performance vs static partitioning

» Framework placement goals: data locality

» Scalability

» Fault recovery

Dynamic Resource Sharing



Mesos vs Static Partitioning

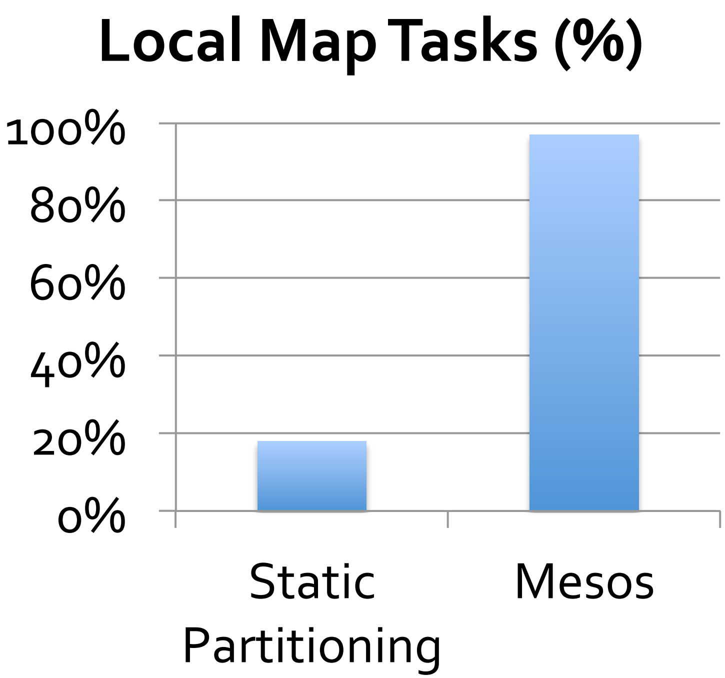
Compared performance with statically partitioned cluster where each framework gets 25% of nodes

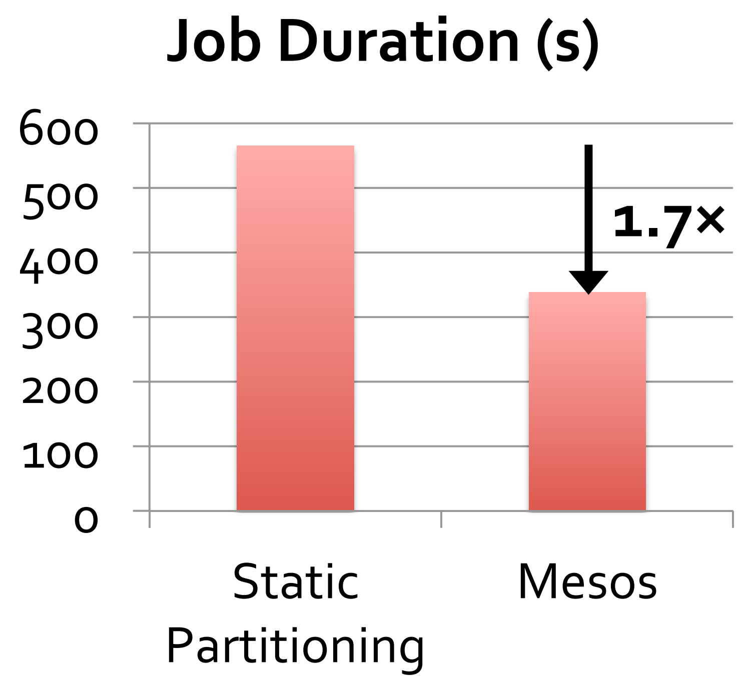
Framework	Speedup on Mesos
Facebook Hadoop Mix	1.14×
Large Hadoop Mix	2.10×
Spark	1.26×
Torque / MPI	0.96×

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)





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

Backup Slides

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

Scheduler Callbacks	Scheduler Actions
resourceOffer(offerId, offers) offerRescinded(offerId) statusUpdate(taskId, status) slaveLost(slaveId)	replyToOffer(offerId, tasks) setNeedsOffers(bool) setFilters(filters) getGuaranteedShare() killTask(taskId)

Executor Callbacks	Executor Actions
launchTask(taskDescriptor) killTask(taskId)	sendStatus(taskId, status)