Mesos

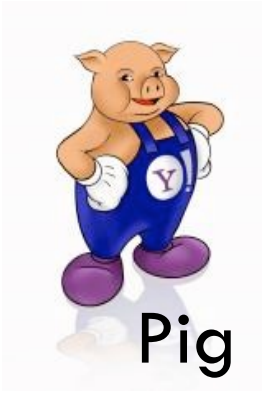
A Platform for Fine-Grained Resource Sharing in the Data Center

Benjamin Hindman, Andy Konwinski, Matei Zaharia

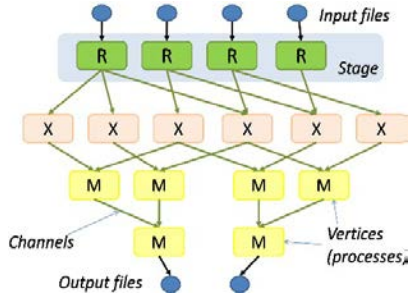
Background

Rapid innovation in cluster computing frameworks

Google Pregel



Pig



Dryad

CIEL

S4 distributed stream computing platform

Google Percolator

Problem

Rapid innovation in cluster computing frameworks

No single framework optimal for all applications

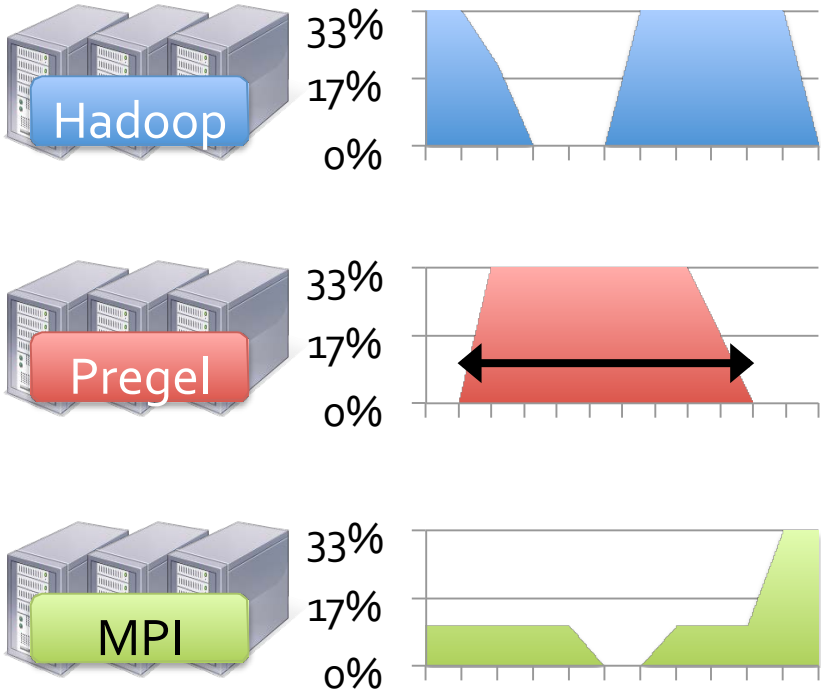
Want to run multiple frameworks in a single cluster

» to *maximize utilization*

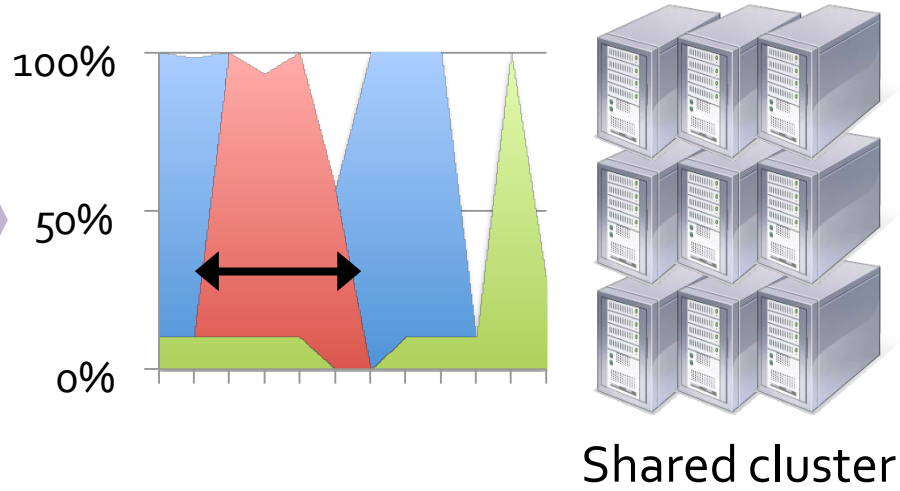
» to *share data* between frameworks

Improvement Direction

Today: static partitioning

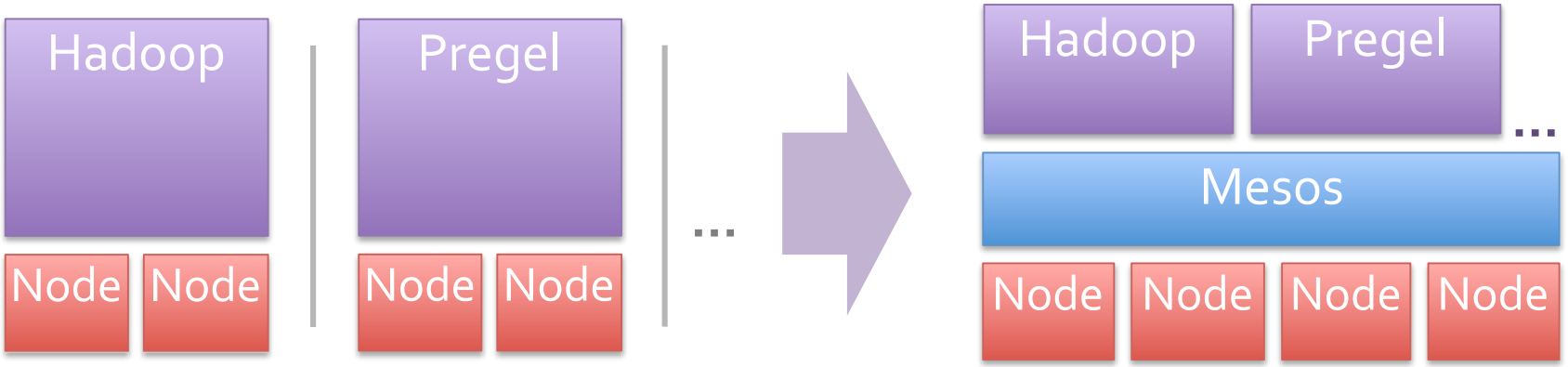


Mesos: dynamic sharing



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

Conclusion

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:

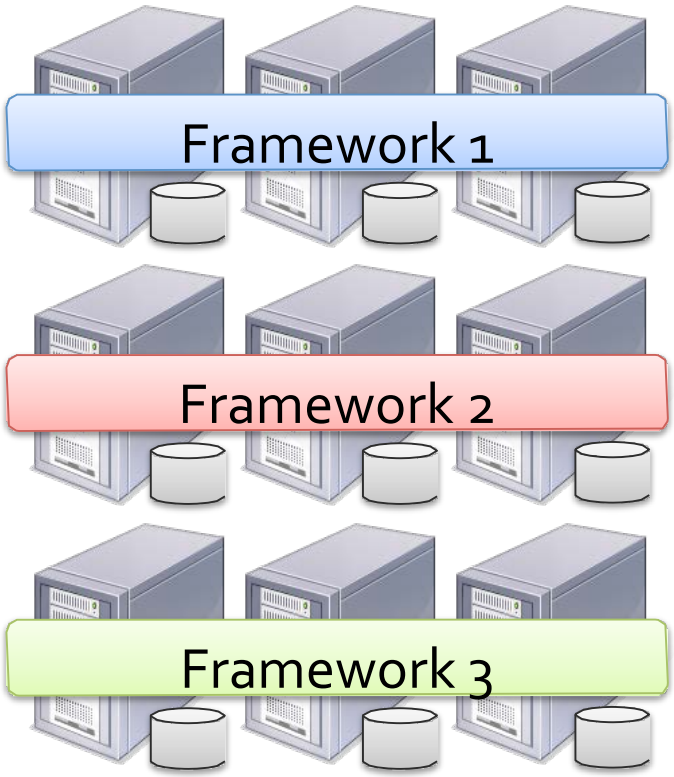
- » Allocation at the level of *tasks* within a job
- » Improves utilization, latency, and data locality

Resource offers:

- » Simple, scalable application-controlled scheduling mechanism

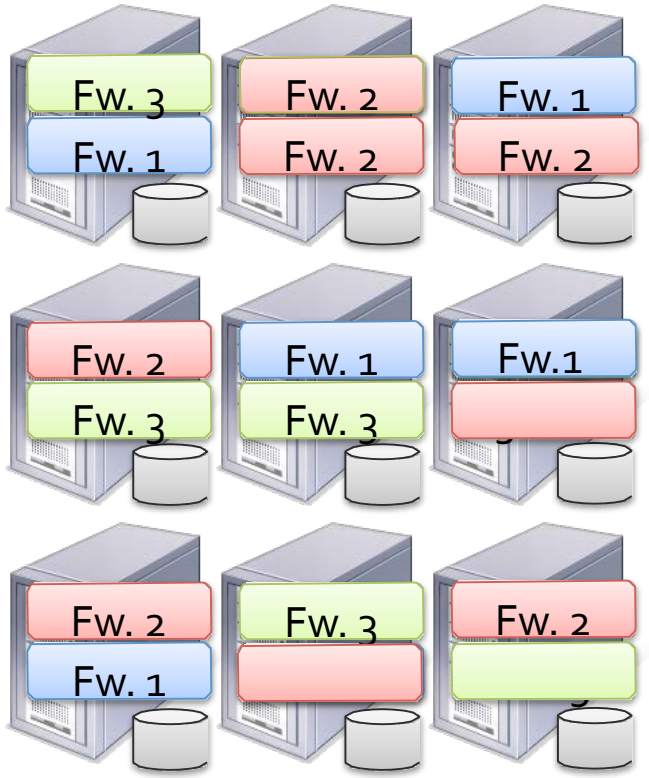
Element 1: Fine-Grained Sharing

Coarse-Grained Sharing (HPC):



Storage System (e.g. HDFS)

Fine-Grained Sharing (Mesos):



Storage System (e.g. HDFS)

+ Improved utilization, 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

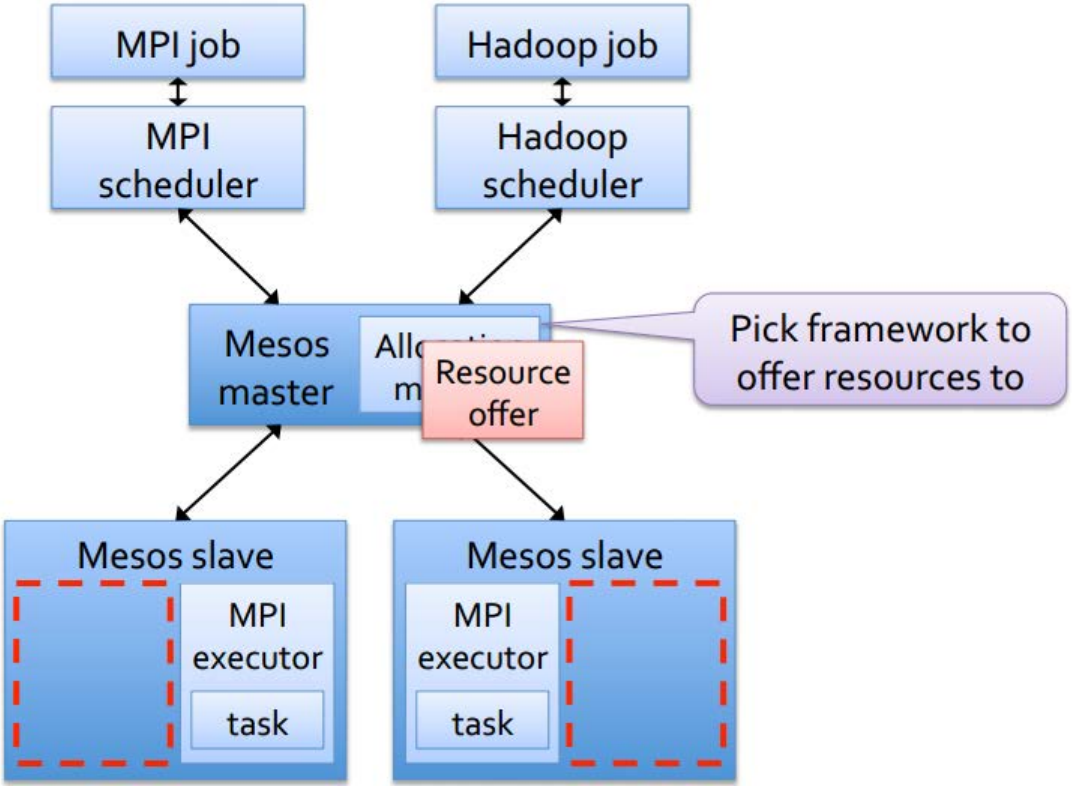
Mesos: Resource offers

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

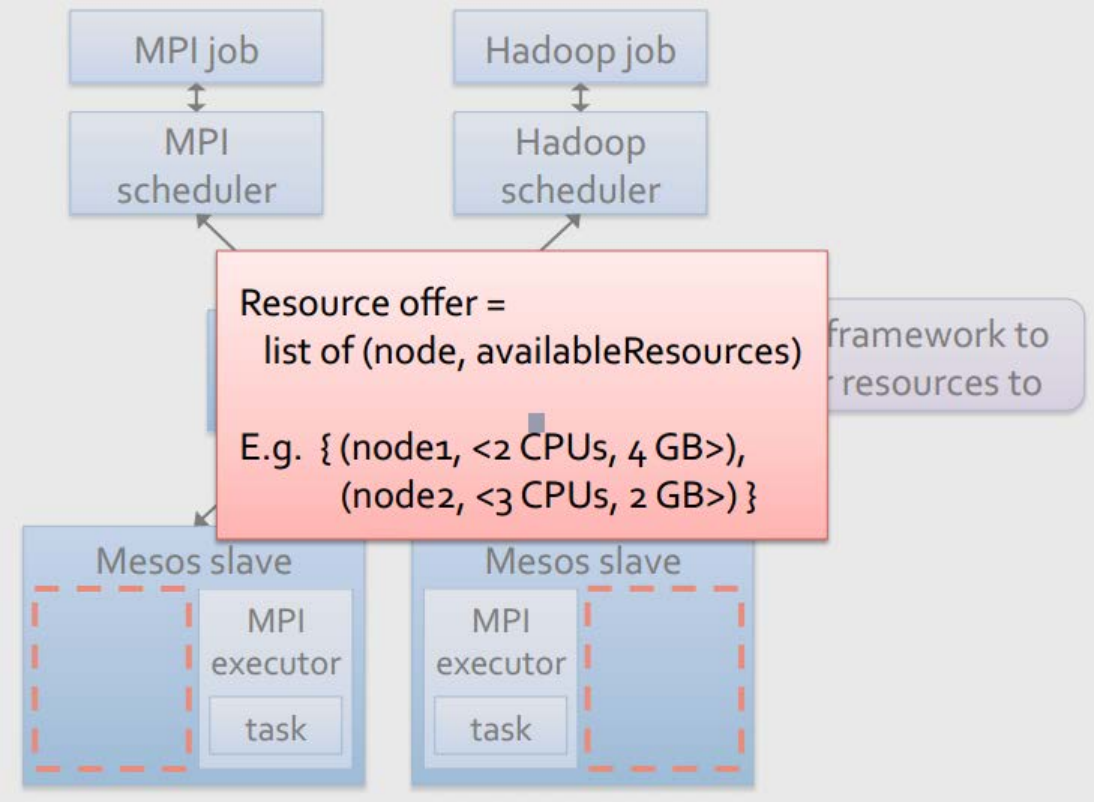
+ Keeps Mesos simple, lets it support future frameworks

– Decentralized decisions might not be optimal

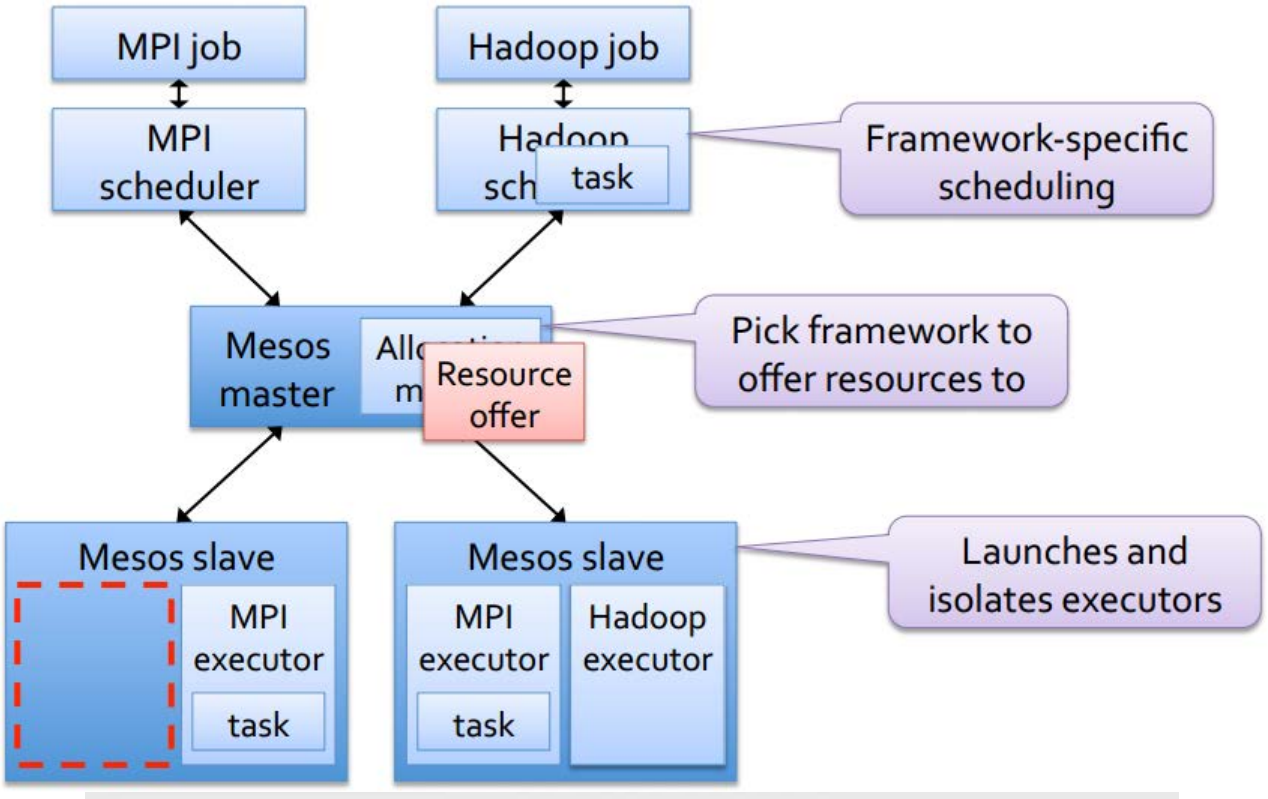
Mesos Architecture



Mesos Architecture



Mesos Architecture



Optimization: Filters

Let frameworks shorten 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 preferences 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

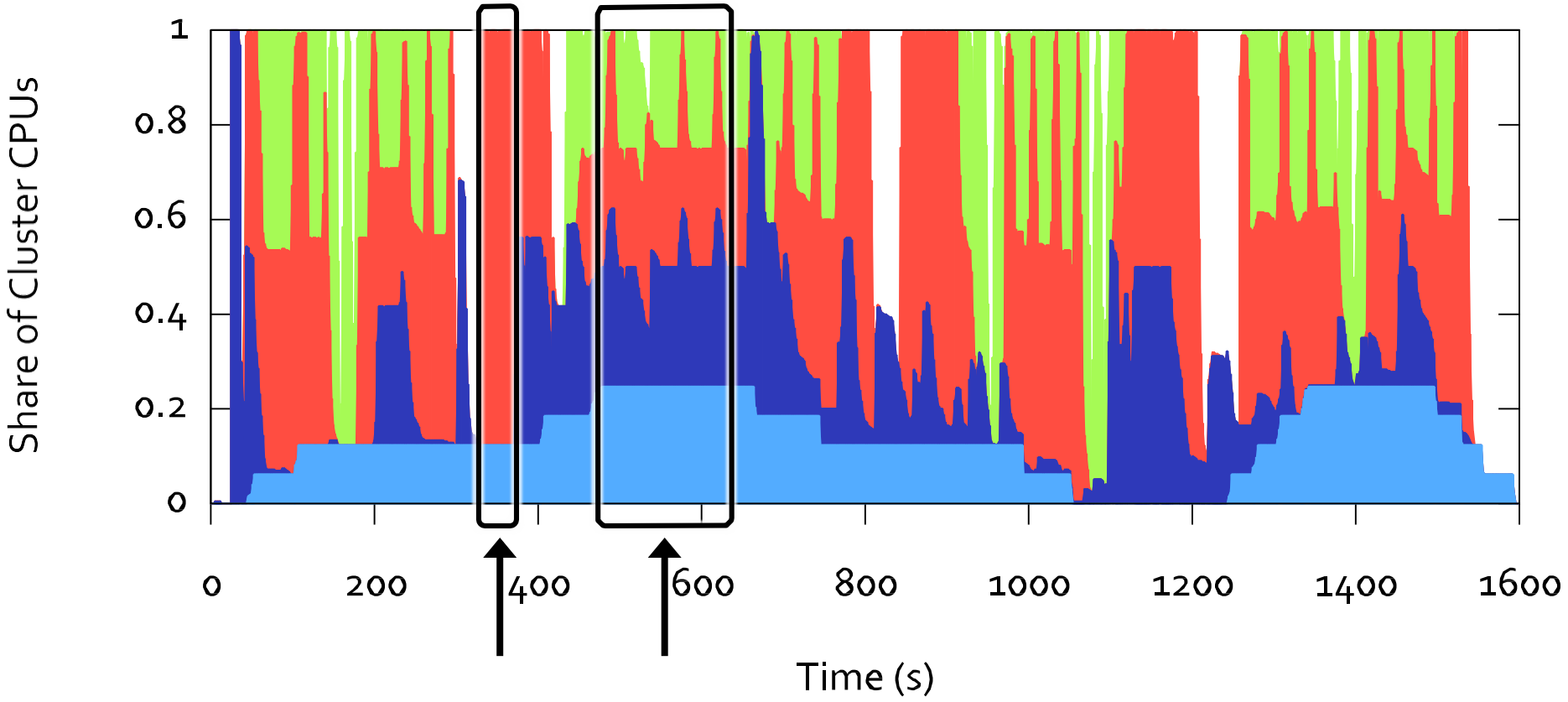
Frameworks ported: Hadoop, MPI, Torque

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

Results

- Utilization and performance vs static partitioning
- Framework placement goals: data locality
- Scalability
- Fault recovery

Dynamic Resource Sharing



Mesos vs Static Partitioning

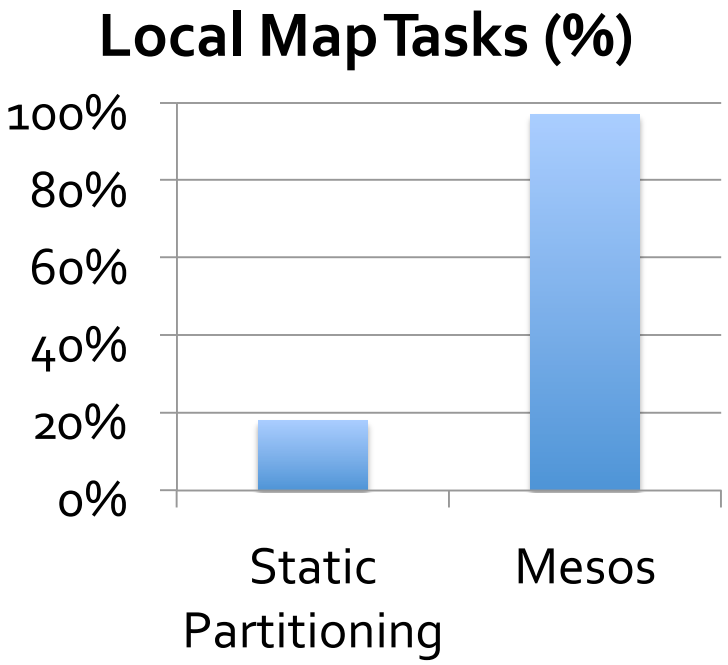
Compared performance with statically partitioned cluster where each framework gets 25% of nodes

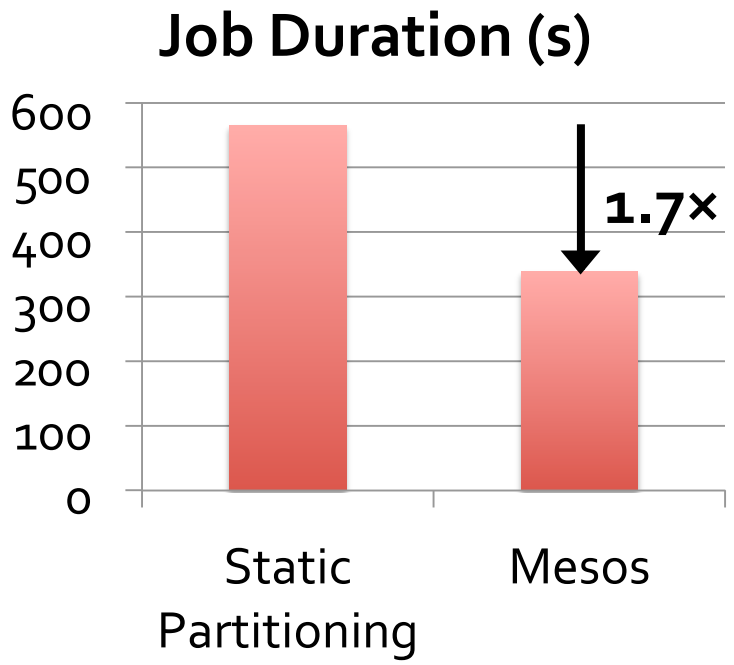
Framework	Speedup on Mesos
Facebook Hadoop Mix	1.14×
Large Hadoop Mix	2.10×
Spark	1.26×
Torque / MPI	0.96×

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)

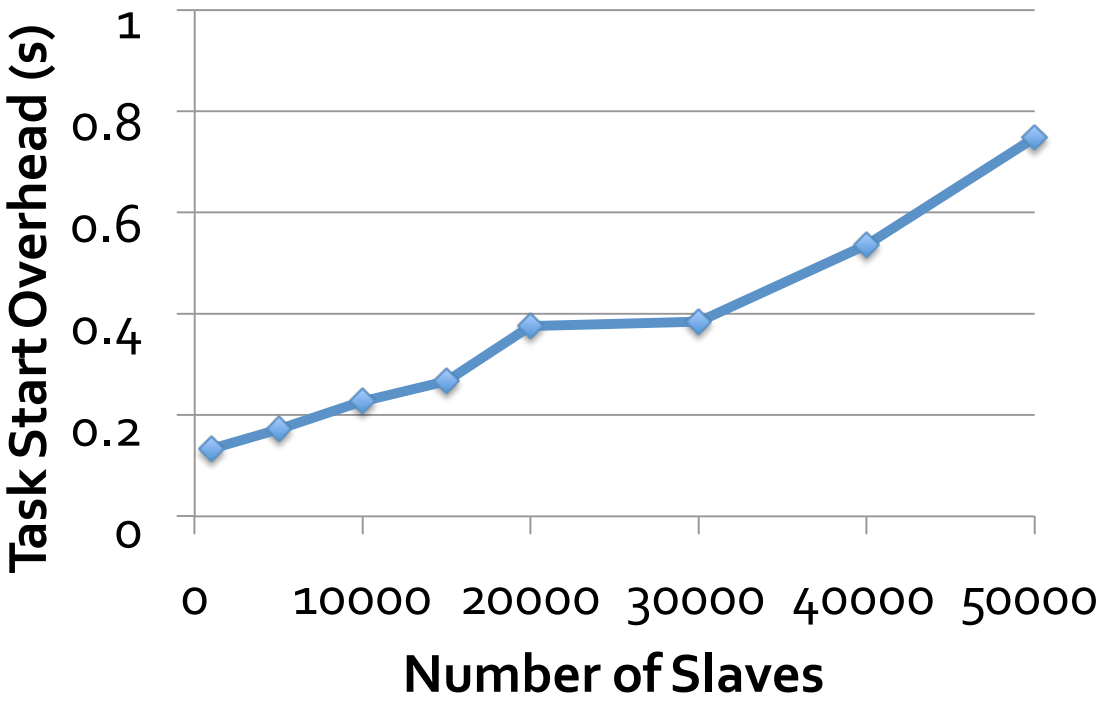




Scalability

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

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

My Summary

Light-weight core

Rely on external module to support properties

Too much power for frameworks may cause unfairness sharing

Not very good in workload isolation