Mesos

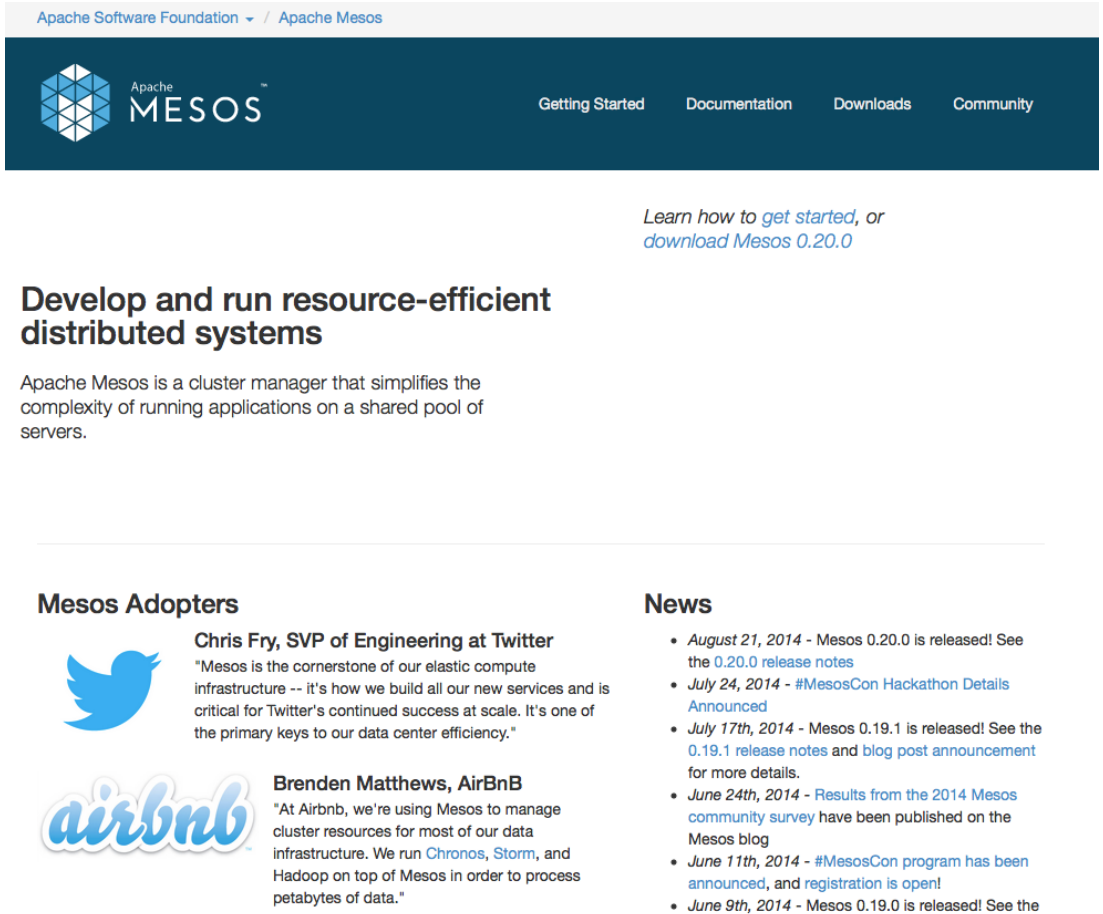
A platform for Fine-Grained Resource Sharing in the Data Center

Big Data and Cloud Systems
Paper Presentation
Yang Young Seok
September 3, 2014

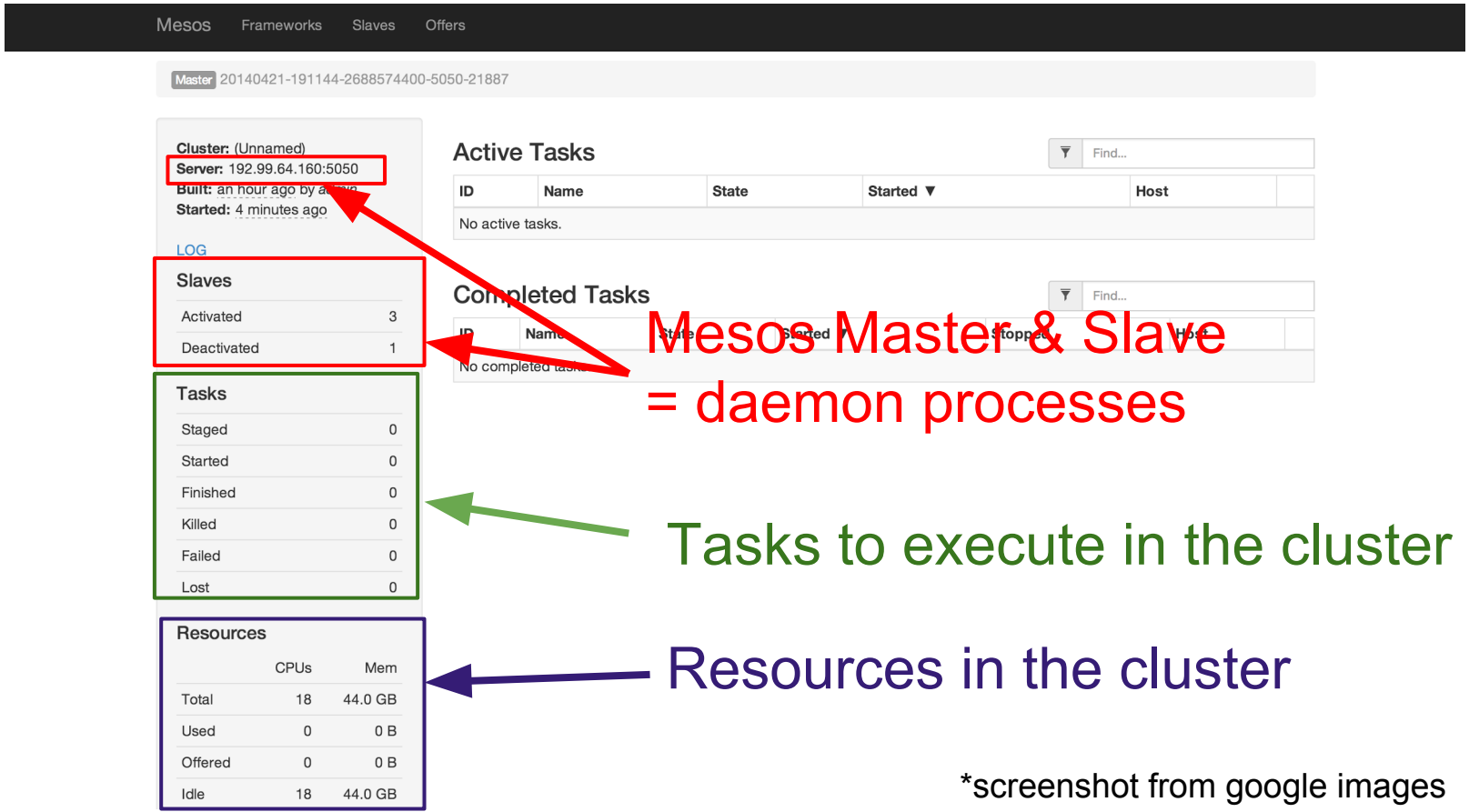
Table of Contents

1. Mesos Overview

Mesos Website



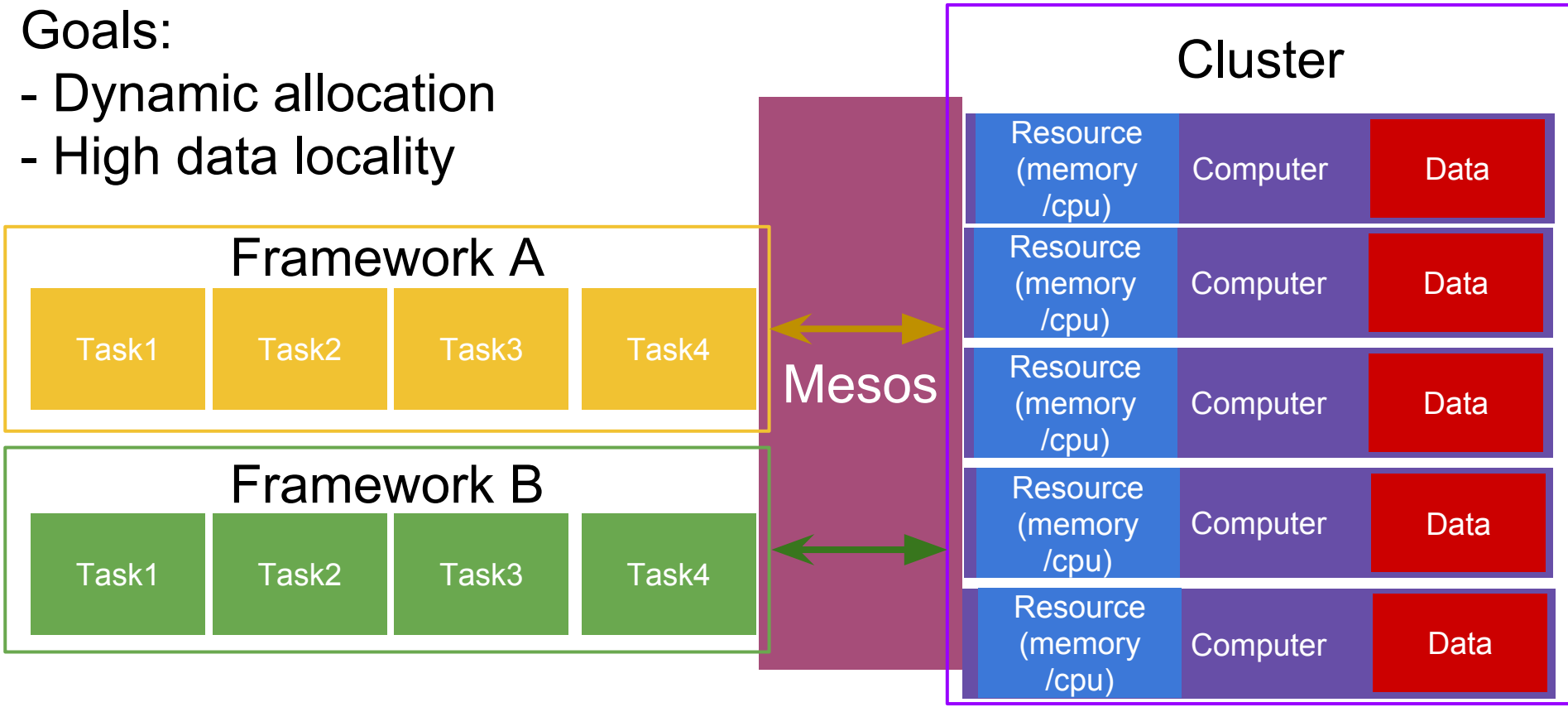
Mesos Master Dashboard



*screenshot from google images

(1)Fine-grained resource sharing

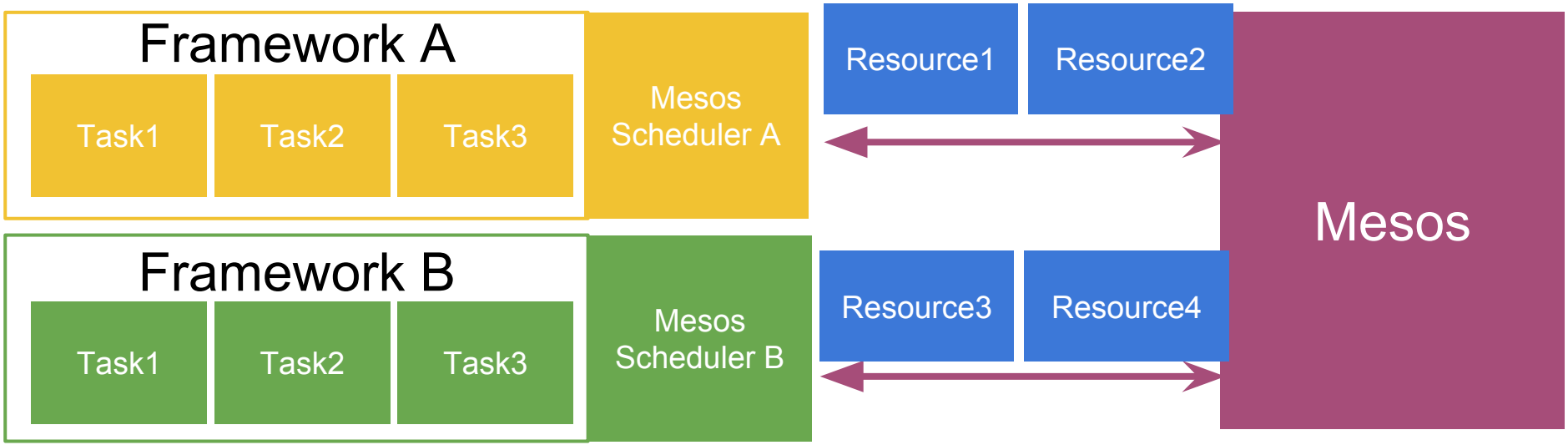
Goals:
- Dynamic allocation
- High data locality



(2)Distributed Scheduling

Goal:

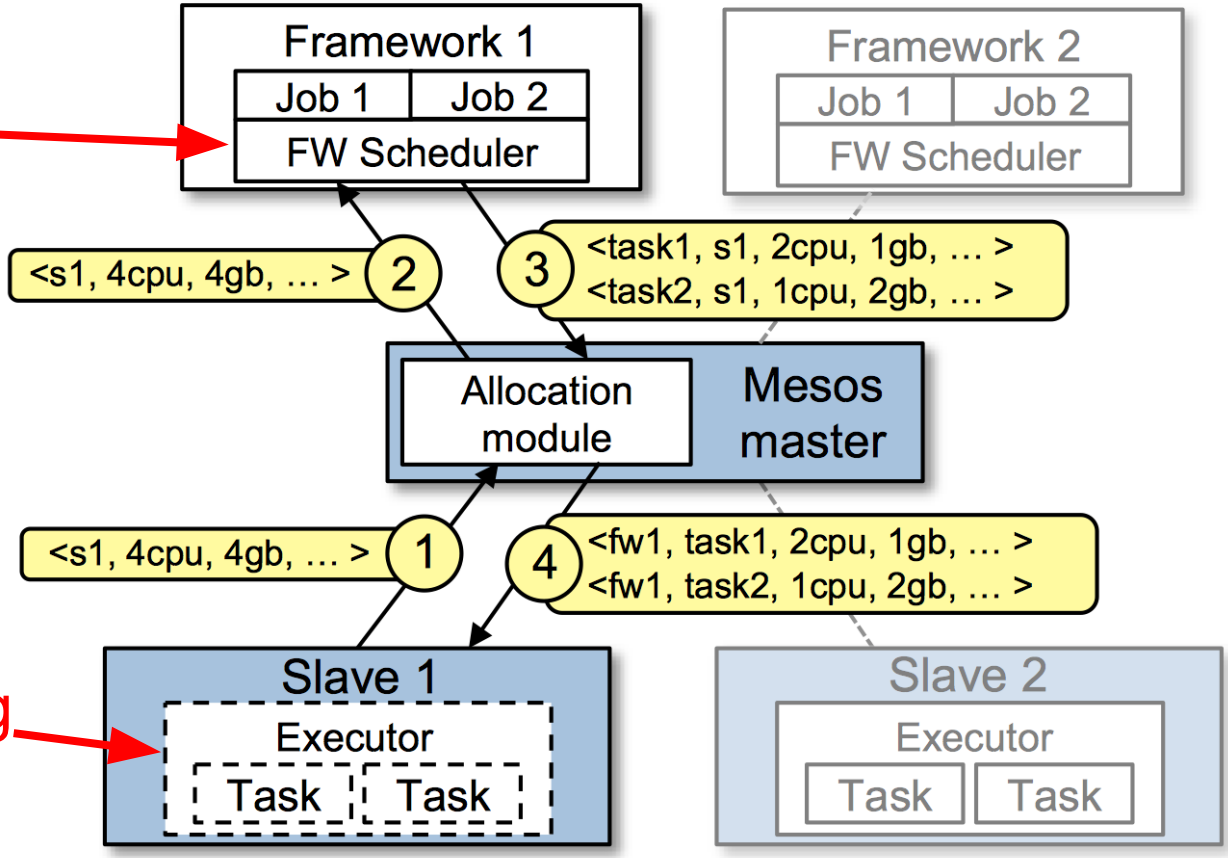
- Meet Different scheduling needs from different frameworks



Result: Resource Offer

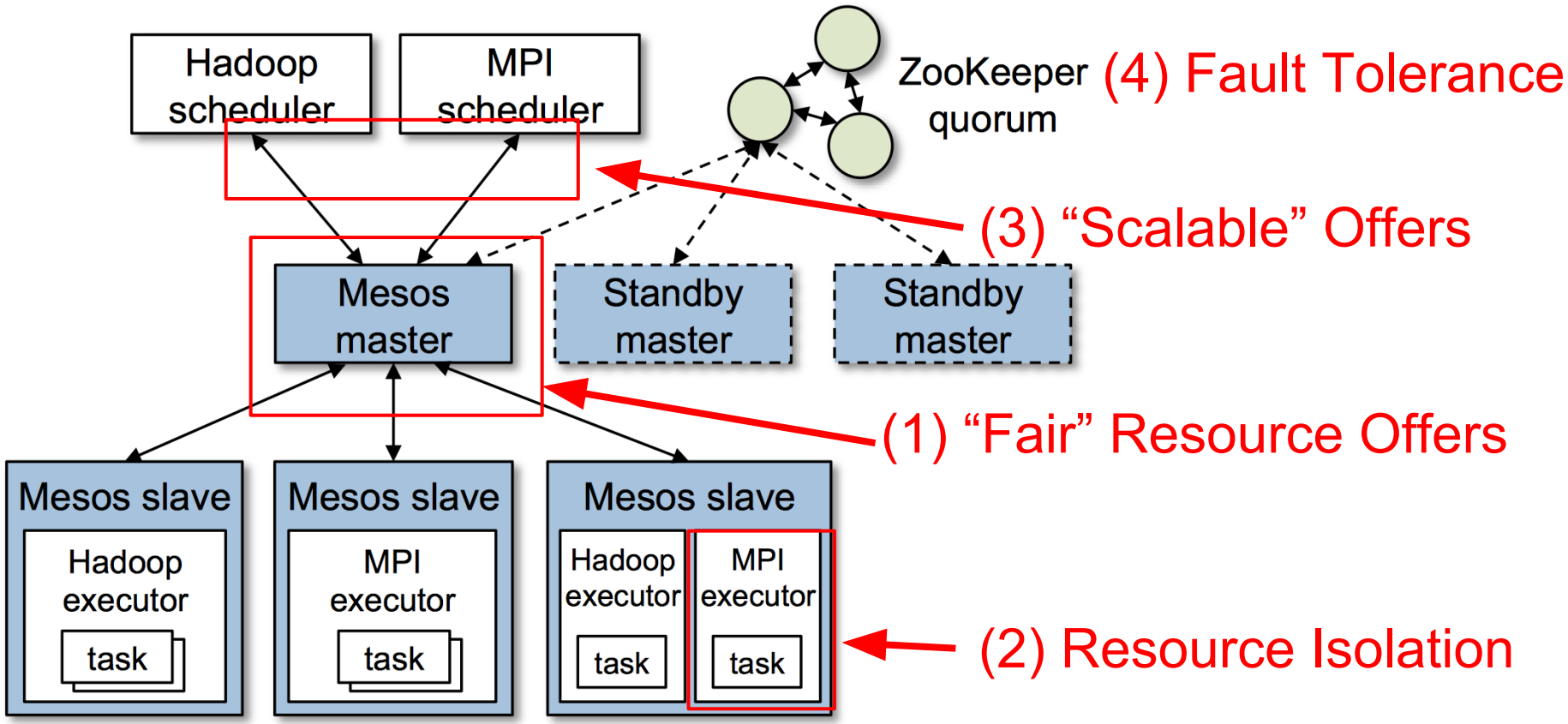
(2)Distributed Scheduling

(1)Fine-Grained Resource Sharing



2. Mesos Architecture

Mesos Architecture



(1) “Fair” Resource Offers

Dominant Resource Fairness algorithm:
Select user with least dominant share
*dominant share = max(resource shares)

Schedule	User A		User B		CPU	RAM
	res. shares	dom. share	res. shares	dom. share	total alloc.	total alloc.
User B	$\langle 0,\ 0\rangle$	**0**	$\langle 3/9,\ 1/18\rangle$	1/3	3/9	1/18
User A	$\langle 1/9,\ 4/18\rangle$	**2/9**	$\langle 3/9,\ 1/18\rangle$	1/3	4/9	5/18
User A	$\langle 2/9,\ 8/18\rangle$	4/9	$\langle 3/9,\ 1/18\rangle$	**1/3**	5/9	9/18
User B	$\langle 2/9,\ 8/18\rangle$	**4/9**	$\langle 6/9,\ 2/18\rangle$	2/3	8/9	10/18
User A	$\langle 3/9,\ 12/18\rangle$	**2/3**	$\langle 6/9,\ 2/18\rangle$	**2/3**	1	14/18

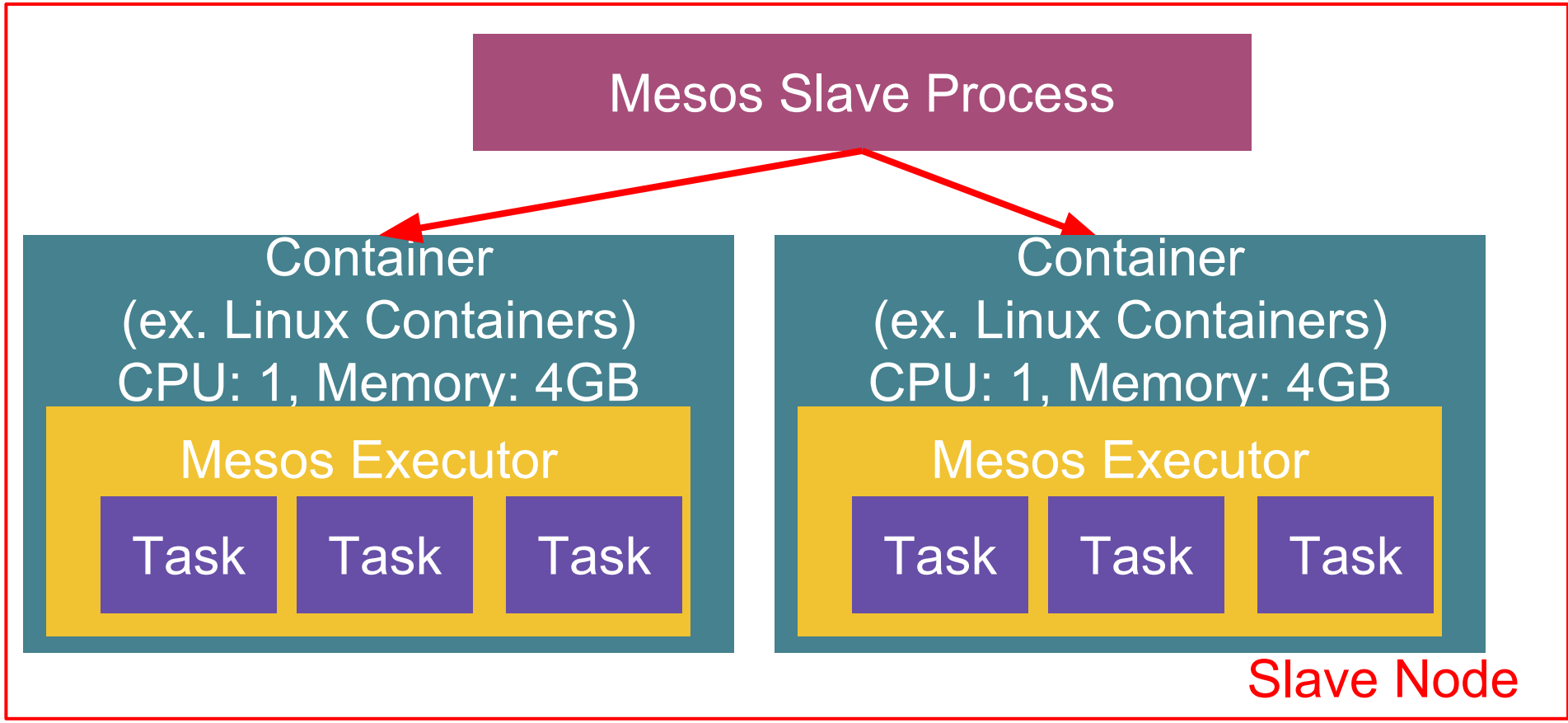
Reference: Dominant Resource Fairness: Fair Allocation of Multiple Resource Types

(1) “Fair” Resource Offers

The definition of "least share" is based on work we did at Berkeley called dominant resource fairness (DRF, see here<https://www.usenix.org/legacy/event/nsdi11/tech/full_papers/Ghodsi.pdf>). In a nutshell, DRF looks at the resources a framework cares the most about (it's dominant resource) and makes sure it has a "fair" share of that resource. Here's the intuition: imagine if we statically partitioned the cluster into 1/N sized smaller clusters for N frameworks. Then the amount of work that each framework could run in their cluster will be upper bounded by the resource they use up first, i.e., their dominant resource! Thus, at the very least, we should probably make sure we let a framework in a Mesos cluster be able to execute that much work at any point in time ... and of course the benefit to running in Mesos is the framework can also elastically scale to more than 1/N when those resources are idle.

Reference: Mesos Mailing List & Dominant Resource Fairness: Fair Allocation of Multiple Resource Types

(2) Resource Isolation



(3) Scalable Offers

- Filters at launchTask(ex. time to consider the unused resources refused)
→ avoids over-communication

- Counts resource offered as allocated
→ prompts quick response to the offers

- No response for a long time
→ rescinds offers

(4) Fault Tolerance(FT)

- HA Master → Soft state (only lists of active slaves, frameworks, schedulers) enables quick recovery via ZooKeeper

- Reports Node/Executor faults to Scheduler
→ enables error-handling by Scheduler

- HA Scheduler → A framework can register multiple schedulers (coordination is up to the user)

3. Mesos Behavior

Homogeneous Tasks

	Elastic Framework		Rigid Framework	
	Constant dist.	Exponential dist.	Constant dist.	Exponential dist.
Ramp-up time	T	$T \ln k$	T	$T \ln k$
Completion time	$(1/2 + \beta)T$	$(1 + \beta)T$	$(1 + \beta)T$	$(\ln k + \beta)T$
Utilization	1	1	$\beta/(1/2 + \beta)$	$\beta/(\ln k - 1 + \beta)$

*derivations in mesos_tech_report

Assuming infinite demand,
Elastic Framework achieves full utilization

Heterogeneous Tasks

Long tasks hurting short tasks?

- if ½ random tasks and 5 slots per node, the probability for a node filled with long tasks is (½)^5 = 0.4%

- Also, Mesos can “offer” maximum task duration

Placement Preferences

When Demand for node N exceeds the Supply
p slots preferred by m frameworks(intend total: s)

A central scheduler
- Weighted fair allocation policy: $p \cdot s_i / (\sum_{i=1}^{m} s_i)$

(Distributed) Mesos schedulers don't know s…
- Mesos Master make offers: $s_i / (\sum_{i=1}^{n} s_i)$

Framework Incentives

1. Short Tasks:
can use short slots, less loss at faults

2. Scale elastically:
start/complete job earlier, grab opportunities

3. Decline resources that won’t be used ASAP:
these will also be counted when offers are made

Limitations

1. Fragmentation
- Short tasks waiting for long tasks
- Long tasks waiting for many short tasks

*Solution: an allocation module enforcing a maximum offer size

2. Interdependent framework constraints
- For example, tasks from 2 frameworks cannot be colocated

*Solution: None (but these cases are rare)

3. Framework complexity due to resource offers
- 1st Argument: decision(distributed) vs. preference(central)
- 2nd Argument: Online-algorithms can be easily implemented

4. Implementation

Mesos Implementation

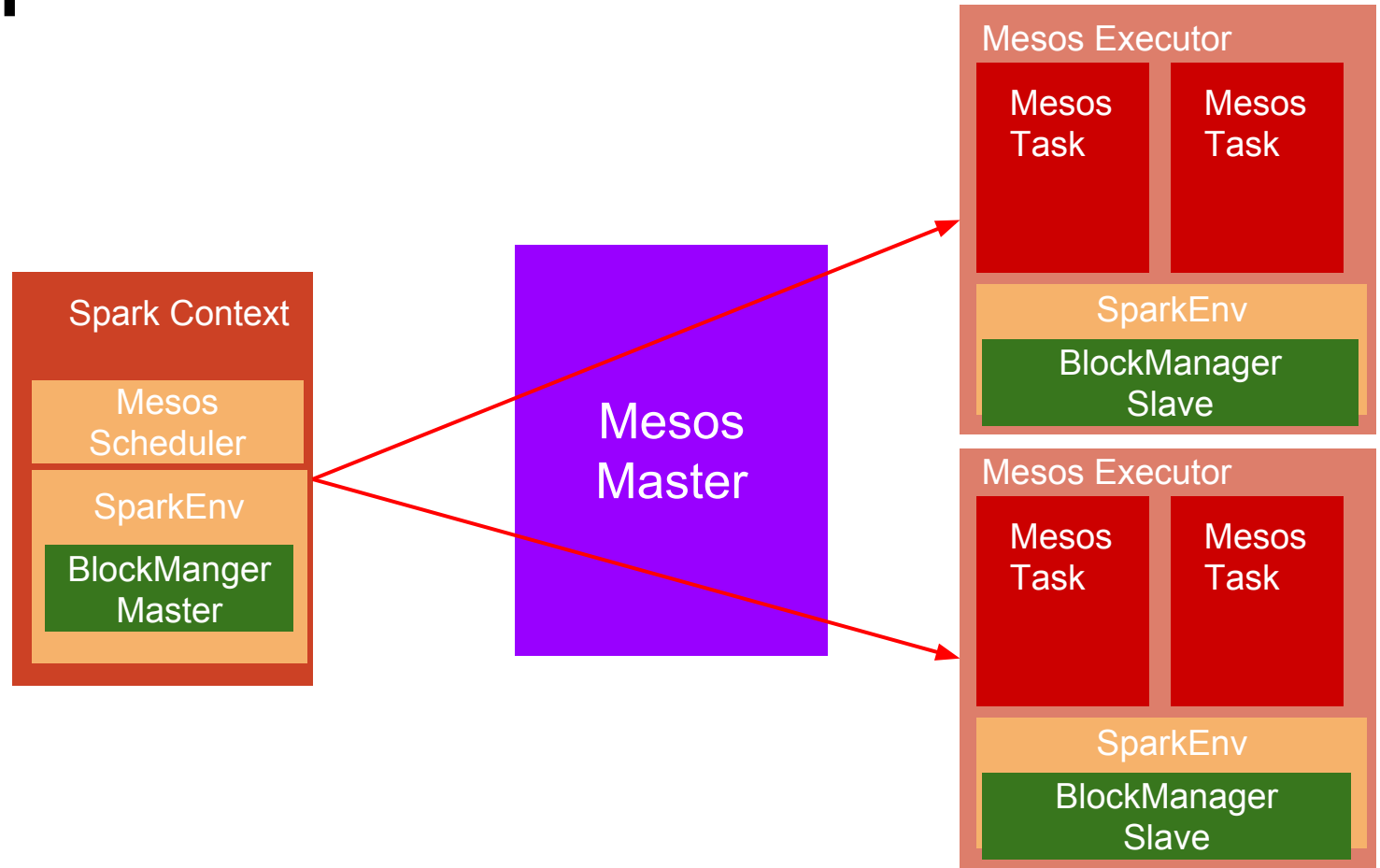
- 10,000 lines of C++

- libprocess for actor-based async I/O

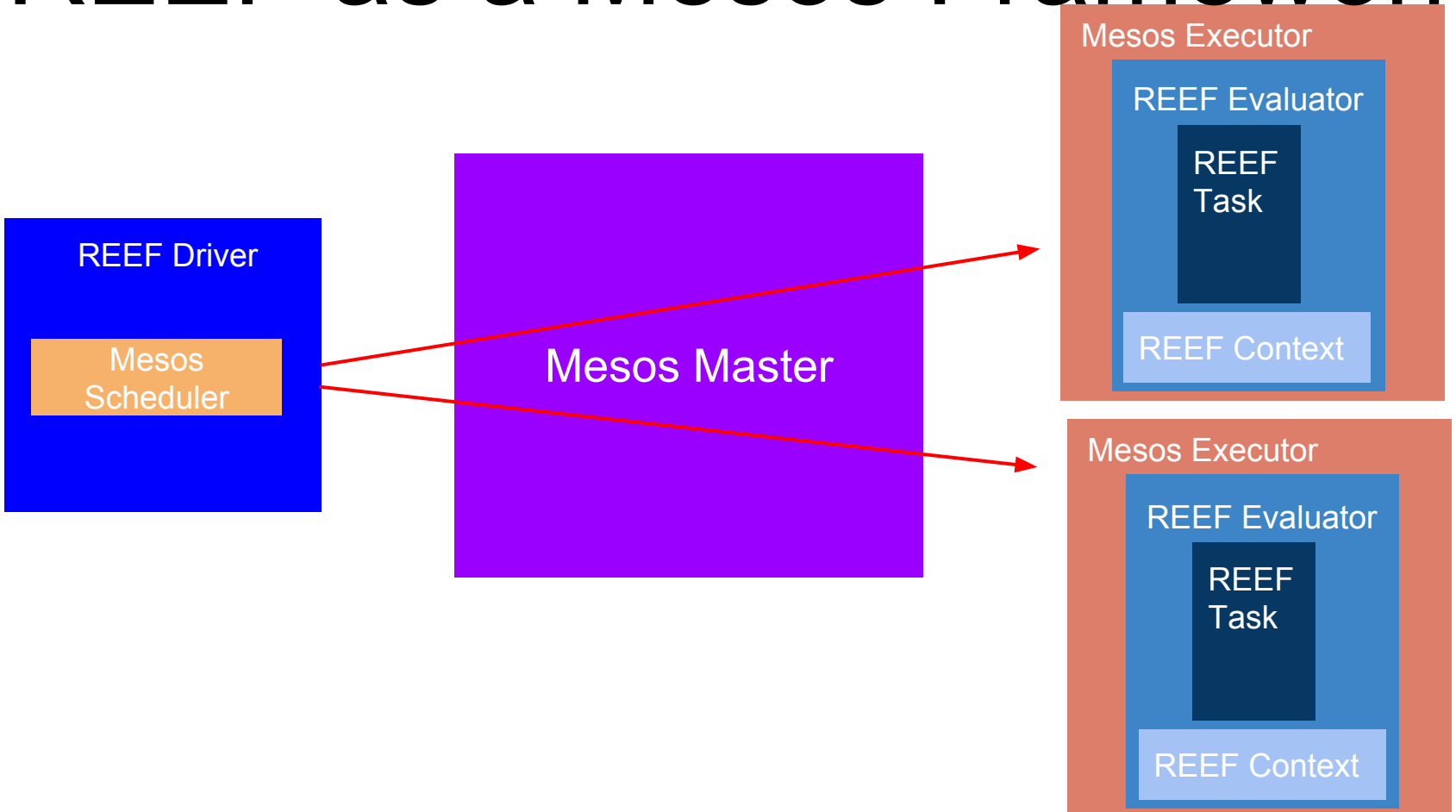
- Zookeeper for HA Mesos Master

- Linux Containers/Solaris projects for task Isolation(CPU Cores/Memory)

Spark as a Mesos Framework



REEF as a Mesos Framework



5. Evaluation

Macrobenchmark

macrobenchmark consisting of a mix of four workloads:

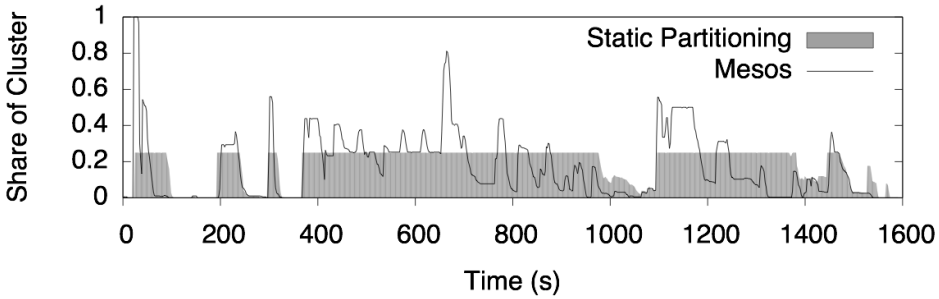
- A Hadoop instance running a mix of small and large jobs based on the workload at Facebook.
- A Hadoop instance running a set of large batch jobs.
- Spark running a series of machine learning jobs.
- Torque running a series of MPI jobs.

We compared a scenario where the workloads ran as four frameworks on a 96-node Mesos cluster using fair sharing to a scenario where they were each given a static partition of the cluster (24 nodes), and measured job response times and resource utilization in both cases. We used EC2 nodes with 4 CPU cores and 15 GB of RAM.

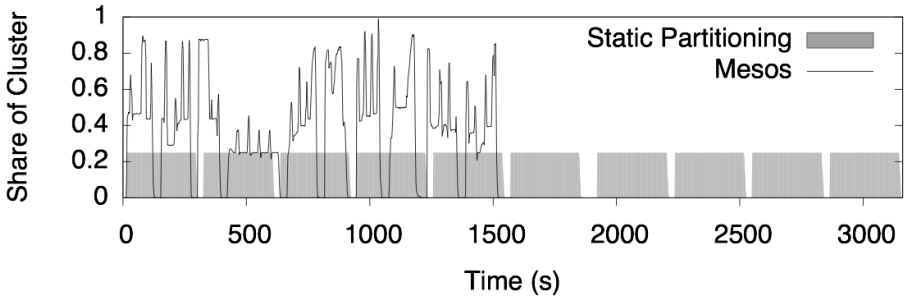
Macrobenchmark

1. higher total utilization 2. faster job execution

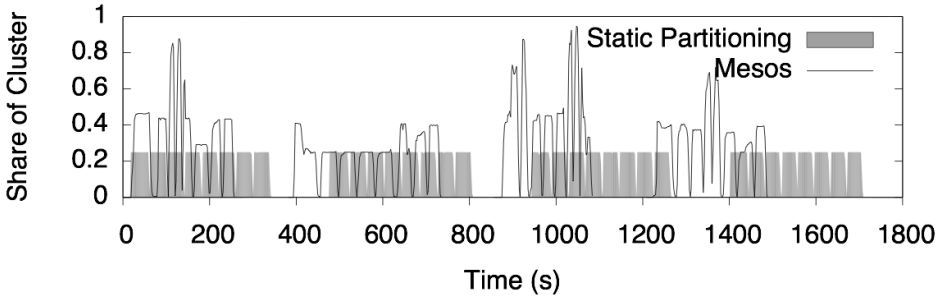
(a) Facebook Hadoop Mix



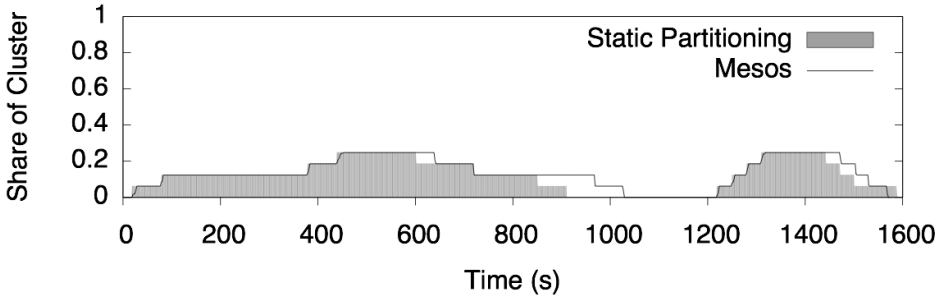
(b) Large Hadoop Mix



(c) Spark



(d) Torque / MPI



Mesos Scalability

- 200 frameworks running continuously
- Launch a test framework(10-sec task) and calculate overhead

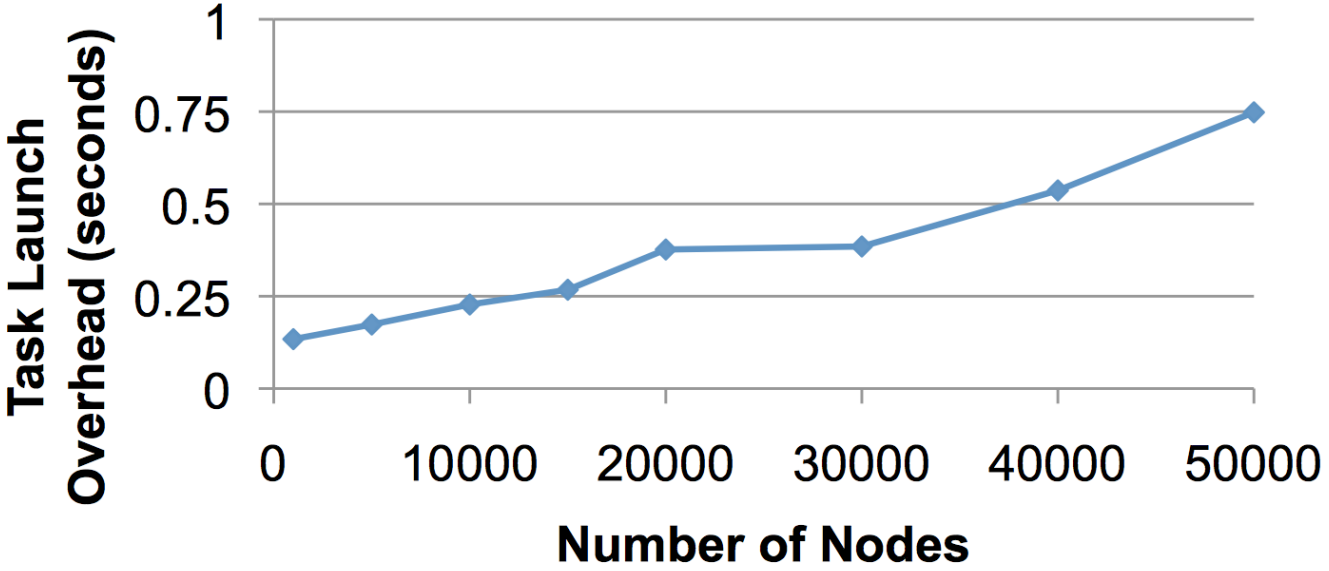


Figure 10: Mesos master's scalability versus number of slaves.

6. Comments

Comments

Pros:
- More Flexible interface than YARN
(YARN ApplicationMaster vs. Mesos Scheduler)
(YARN Container vs. Mesos Executor)
- Solid definition of “Fair Sharing”

Cons:
- No Benchmark against similar systems
(ex. Hadoop 1.0 MapReduce jobs)
- Why does Mesos Scheduler run locally?
(It could run on a slave node like YARN ApplicationMaster)