UNIVERSITY OF CAMBRIDGE

Cloud Computing

Large-scale Resource Management

Eva Kalyvianaki
ek264@cam.ac.uk

Contents

1. Mesos: A Platform for Fine-Grained Resource Sharing in Data Center

   by B. Hindman, A. Knwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, I. Stoica from University of California, Berkeley, NSDI 2011.

1. Omega: flexible, scalable schedulers for large compute clusters

   by Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek and John Wilkes, EuroSys 2013.

Background

- Data centers are built from clusters of commodity hardware
- These clusters run a diverse set of applications: e.g., MapReduce, Data Streaming, Storm, S4, etc
- Each application has its own execution framework
- Multiplexing a cluster between frameworks improves resources utilisation and so reduces costs

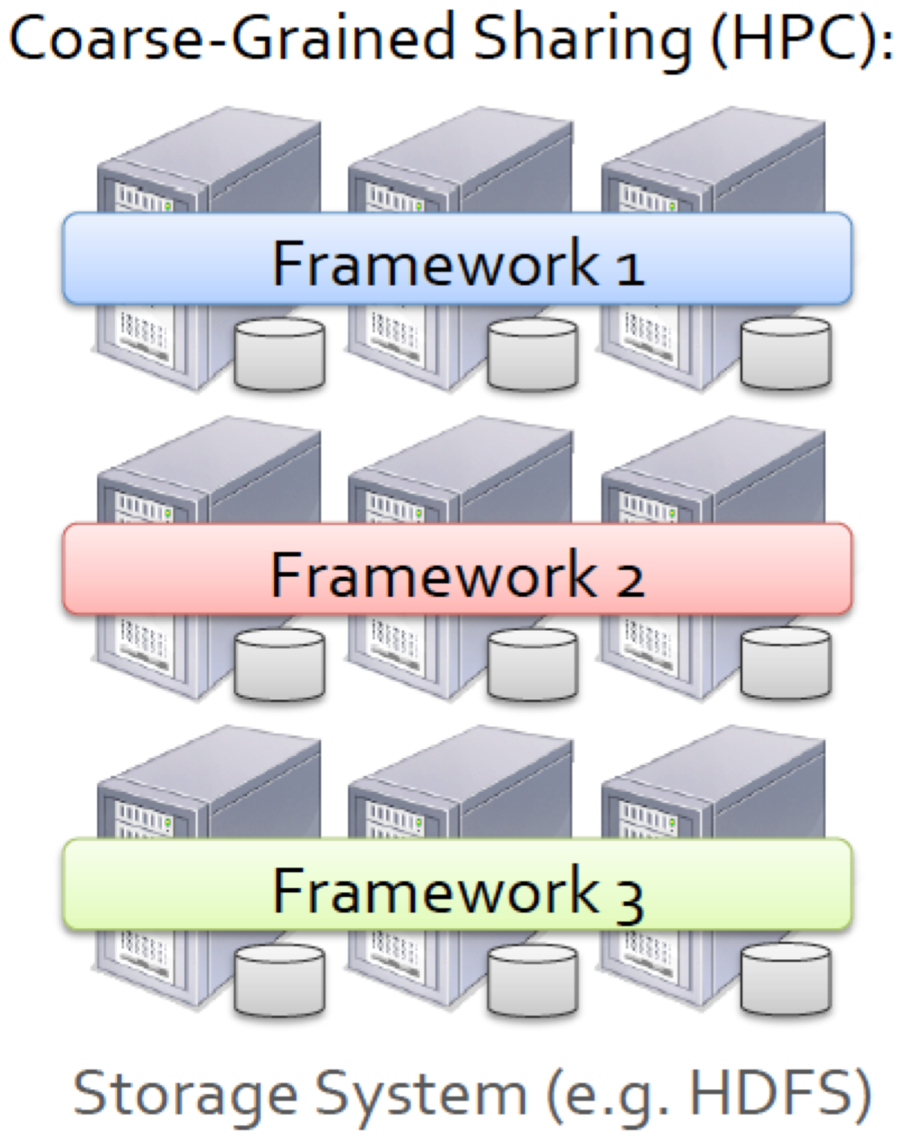
Problem Statement

- It is very difficult and challenging for a single framework (application-specific) to efficiently manage the resources of clusters

- The aim of the Mesos work is to be able to run multiple frameworks in a single cluster in order to:
  - maximise utilisation
  - share data between frameworks

Common solutions for sharing clusters

1. **Statically partition** the cluster and run one framework (or VMs) per partition. Disadvantages:
   1. Low utilisation
   2. Rigid partitioning which might not match the real-time changing application demands
   3. Mismatch between the allocation granularity of static partitioning and current application frameworks such as Hadoop.



Mesos

- Challenges and Goals:
  1. **High utilisation**
  2. **Support diverse frameworks**: each framework will have different scheduling needs
  3. **Scalability:** the scheduling system must scale to clusters of 1,000s of nodes running 100s of jobs with 1,000,000s of tasks
  4. **Reliability:** because all applications would depend on Mesos, the system must be fault-tolerant and highly available.

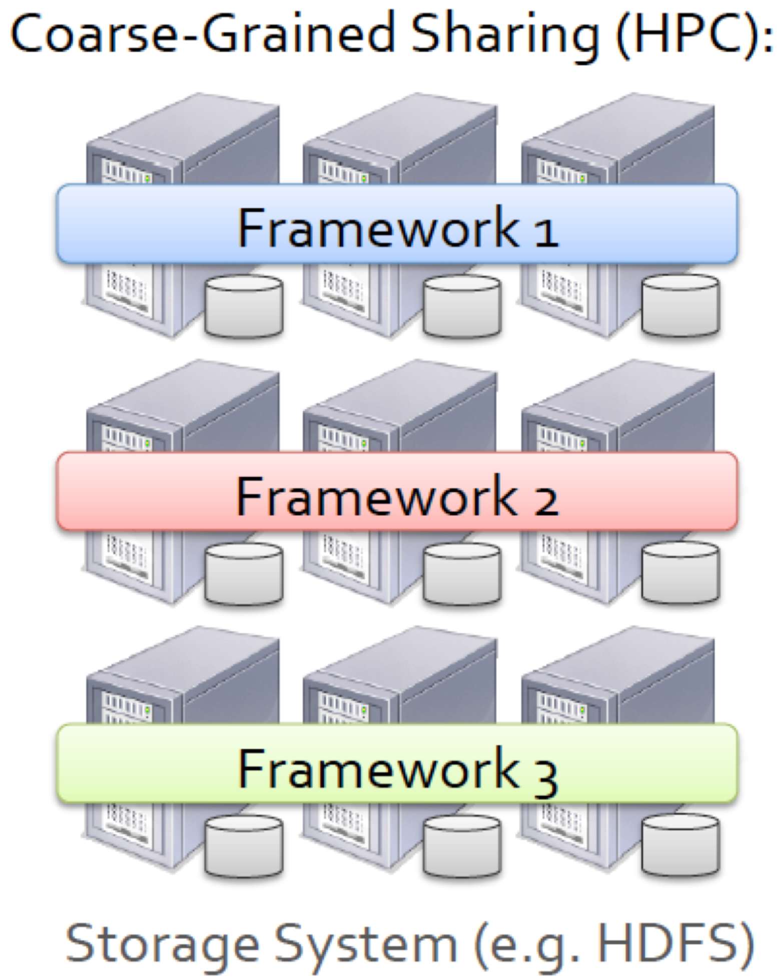
- **Mesos:** a thin resource sharing layer that enables fine-grained sharing across diverse cluster computing frameworks, by giving frameworks a common interface for accessing cluster resources.

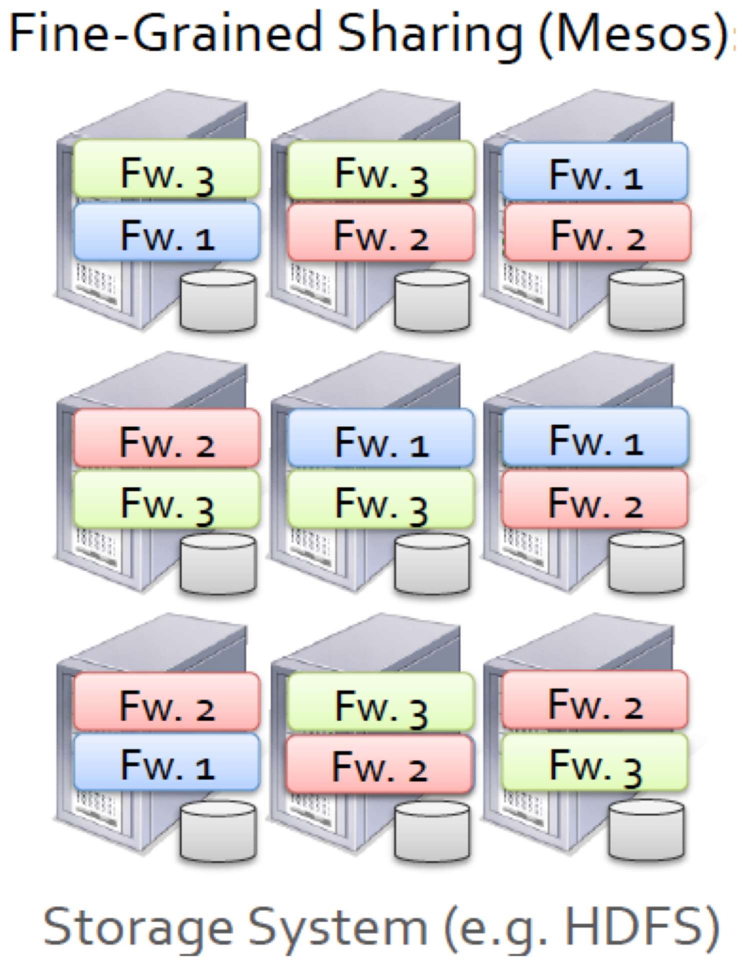
Design Elements

1. **Fine-grained sharing:**
   - Allocation at the level of tasks within a job
   - Improves utilisation, latency and data locality

2. **Resource offers:**
   - Offer available resources to frameworks
   - Simple, scalable application controlled scheduling mechanism

Design Element 1, Fine-Grained Sharing





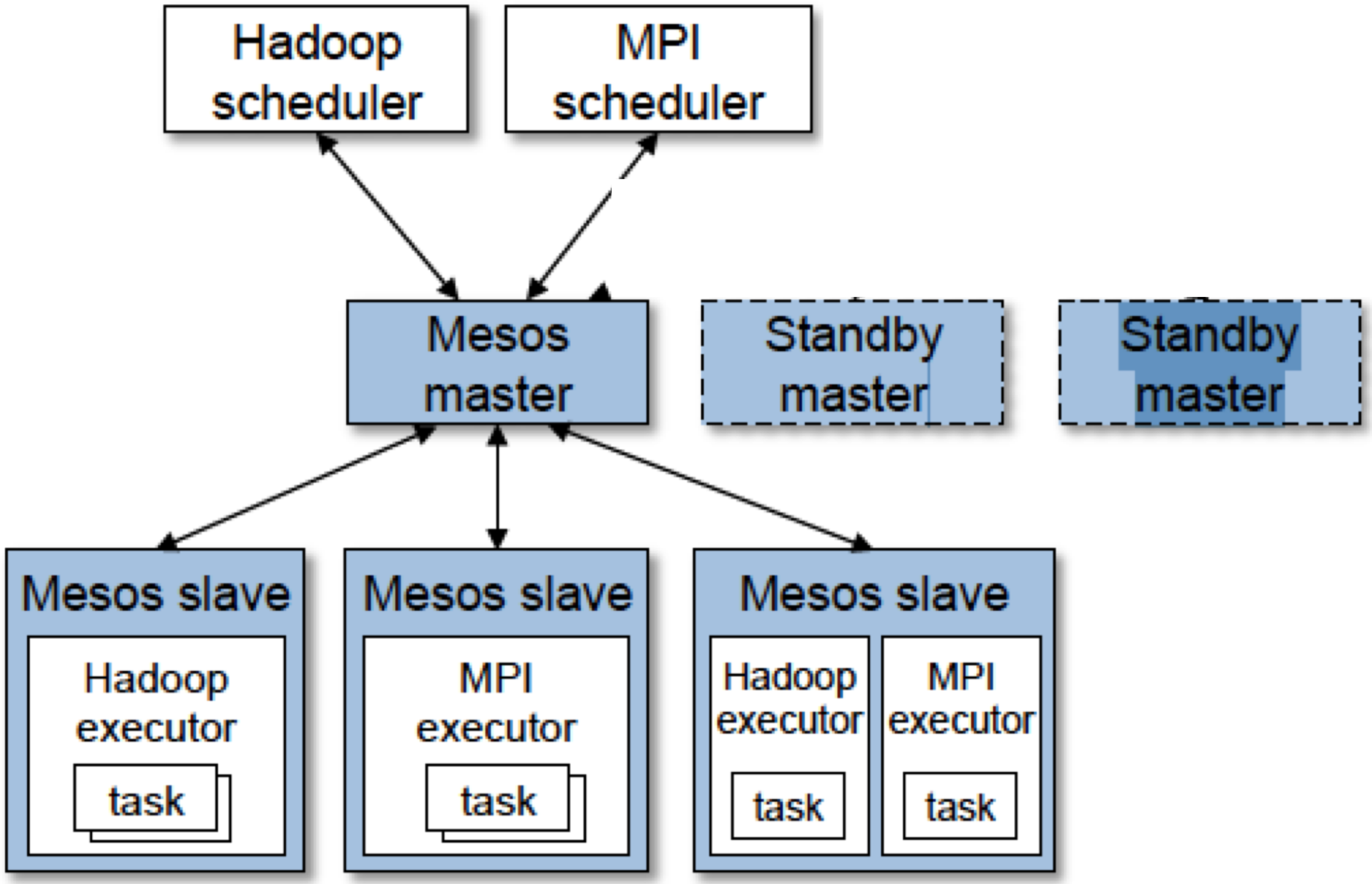
Design Element 2, Resource Offers

- Mesos proposes and uses **Resource Offers**:

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

- **Advantage:** keeps Mesos simple, expandable to future frameworks
- **Disadvantage:** Decentralised solutions are not optimal

Mesos Architecture

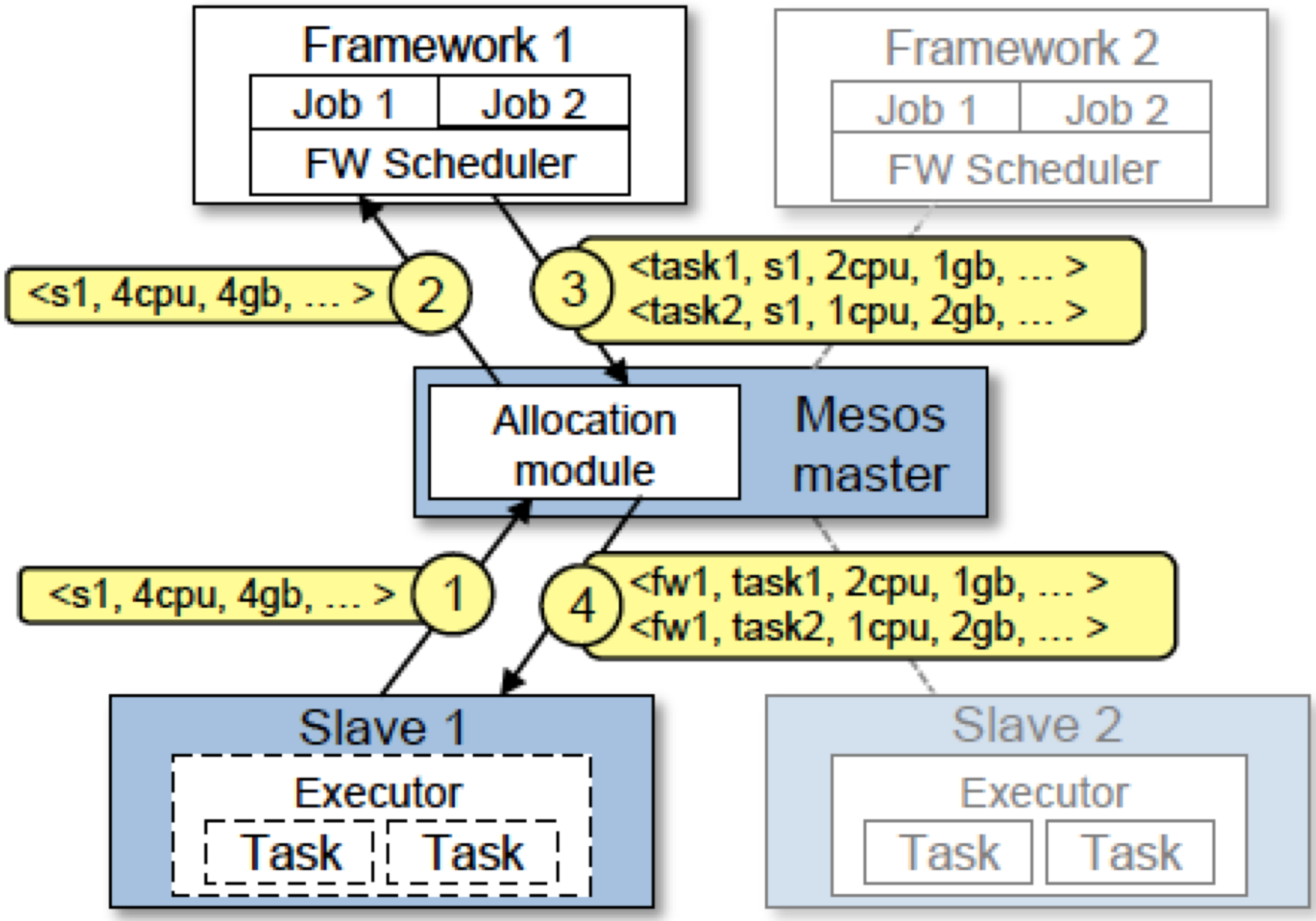


Mesos Architecture, Terminology

- Mesos master: fine-grained sharing across frameworks
- Mesos slave on each node
- Frameworks that run tasks on slave nodes
- Framework schedulers

- "While the Mesos master determines how many resources to offer to each framework, the frameworks' schedulers select which of the offered resources to use".

Mesos Resource Offer Example



Mesos Resource Offers

- Frameworks can reject offers when these do not satisfy their constraints in order to wait for ones that do.

- But, a framework might have to wait for a long time until it receives an offer that satisfies its constraints and Mesos might be sending these offers to many frameworks → time consuming.

- So, Mesos frameworks set their own *filters*; they specify offers that will always be rejected.

Mesos Architecture Details

- Resource Allocation Module

1. Fair Sharing based on a generalisation of max-min fairness for multiple resources
2. Strict priorities
3. Mesos can also (revoke) kill tasks when a greedy framework uses up lots of resources for a long time. Mesos allows the framework a greedy period.

- Isolation is achieved using Linux Containers

Mesos Architecture Details (con't)

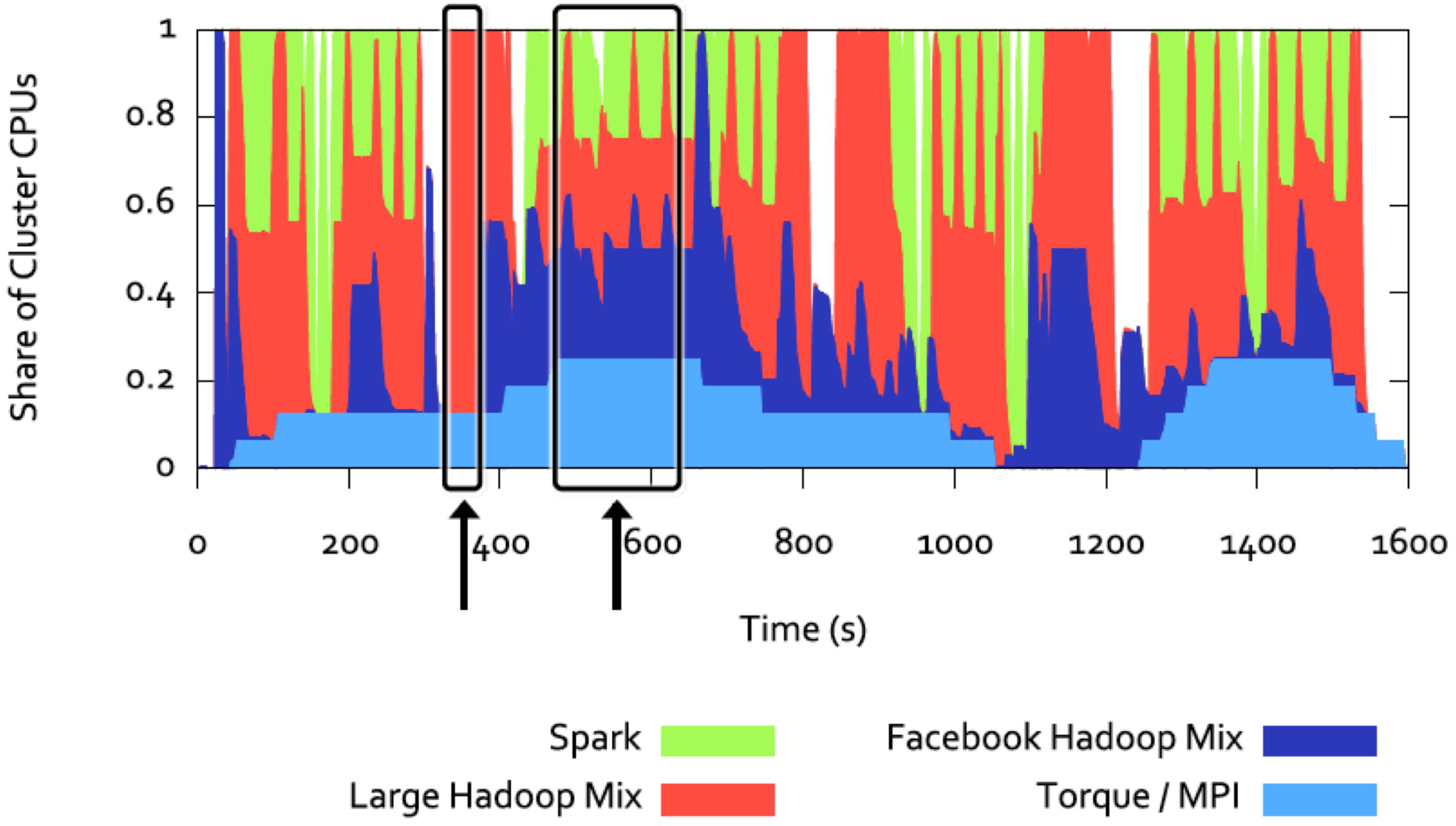
- Scalable and Robust Resource Offers
  - Use of filters: "*only offer nodes from list L*", "*only offer nodes with at least R resources free*", these can be evaluated quickly
  - Mesos counts resources offered to a framework towards its allocation of the cluster
  - If a framework takes a long time to respond, Mesos withdraws the offers and asks another framework

Mesos Evaluation

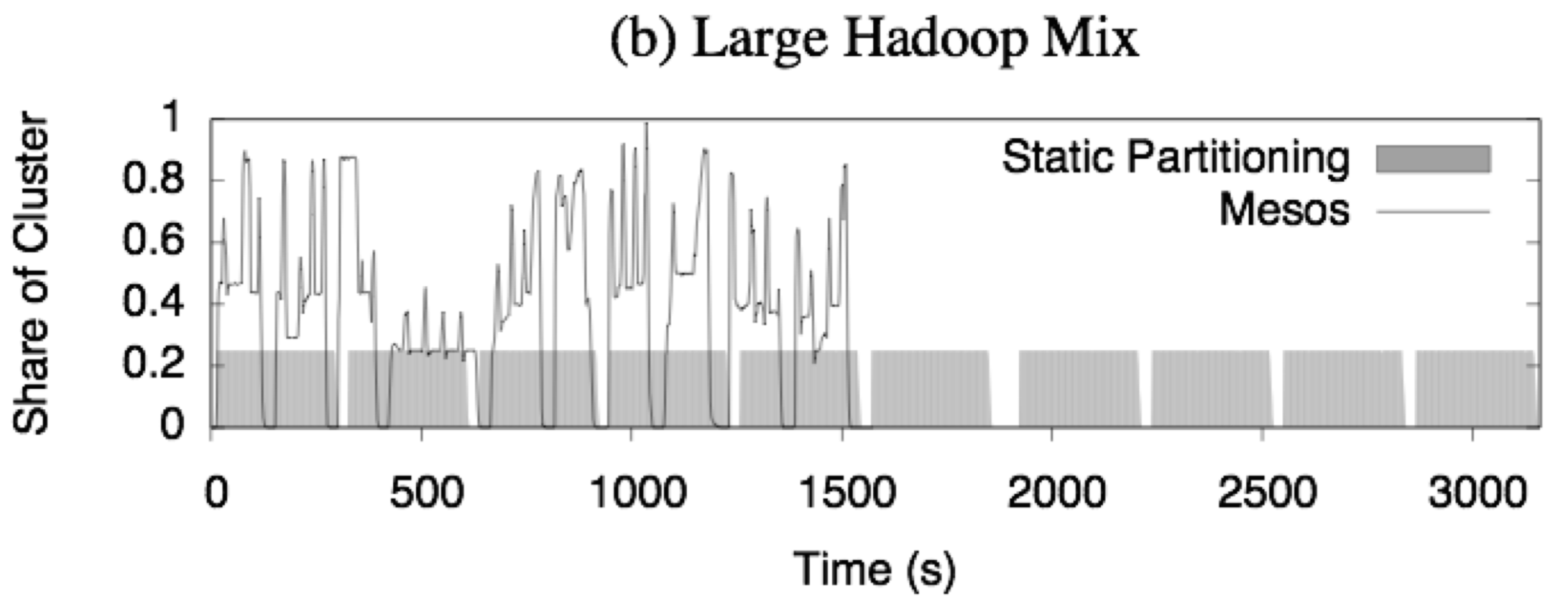
- Amazon EC2 with 92 Mesos nodes
- A mix of Frameworks:
  1. Hadoop running a mix of small and large jobs based on a Facebook workload
  2. A Hadoop instance running a set of large batch jobs
  3. Spark running machine learning jobs
  4. Torque running MPI jobs

  5. Mesos should: achieve higher utilisation and all jobs should finish at least as fast as in the static partitioning

Mesos Performance For all Frameworks



Mesos Performance vs Static Partitioning

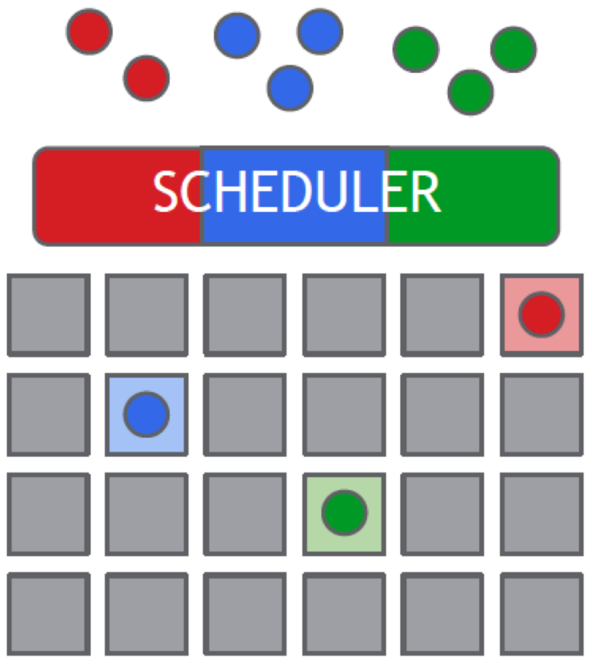


Omega: flexible, scalable schedulers for large compute clusters

- Challenges:
  - Google maintains different data centers around the world
  - Clusters and workloads are growing in size
  - Diverse workloads
  - Growing job arrival rates
- Need for a *scalable scheduler* to tackle these challenges

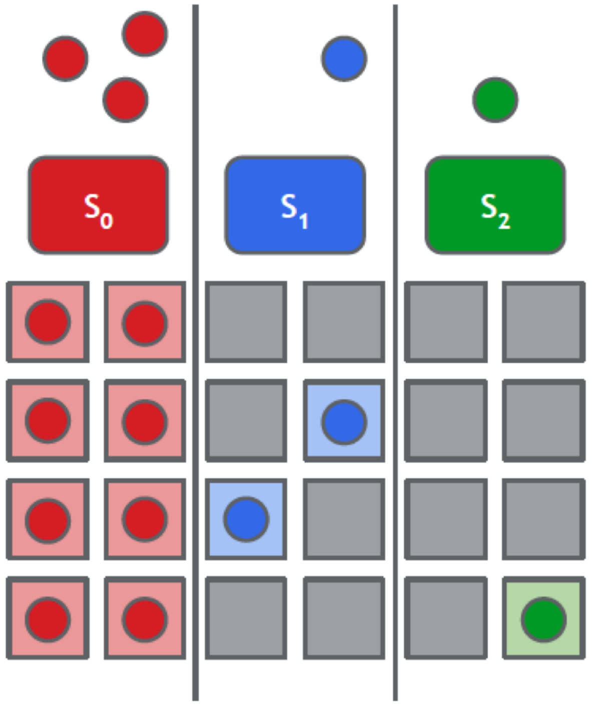
Existing Approaches

monolithic scheduler



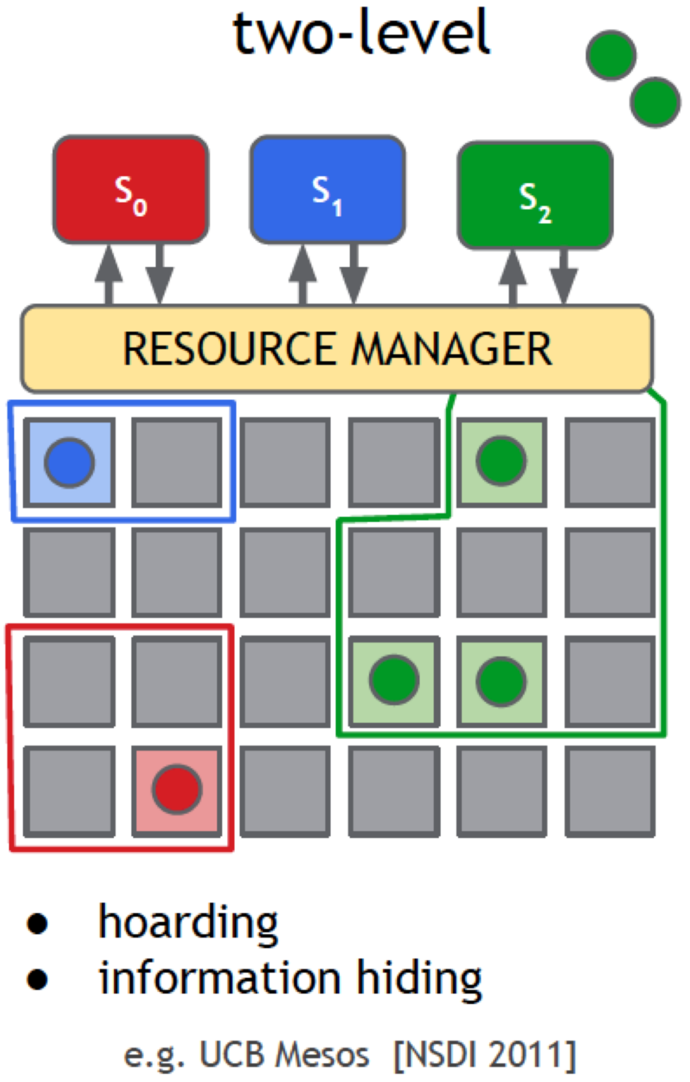
- hard to diversify
- code growth
- scalability bottleneck

static partitioning



- poor utilization
- inflexible

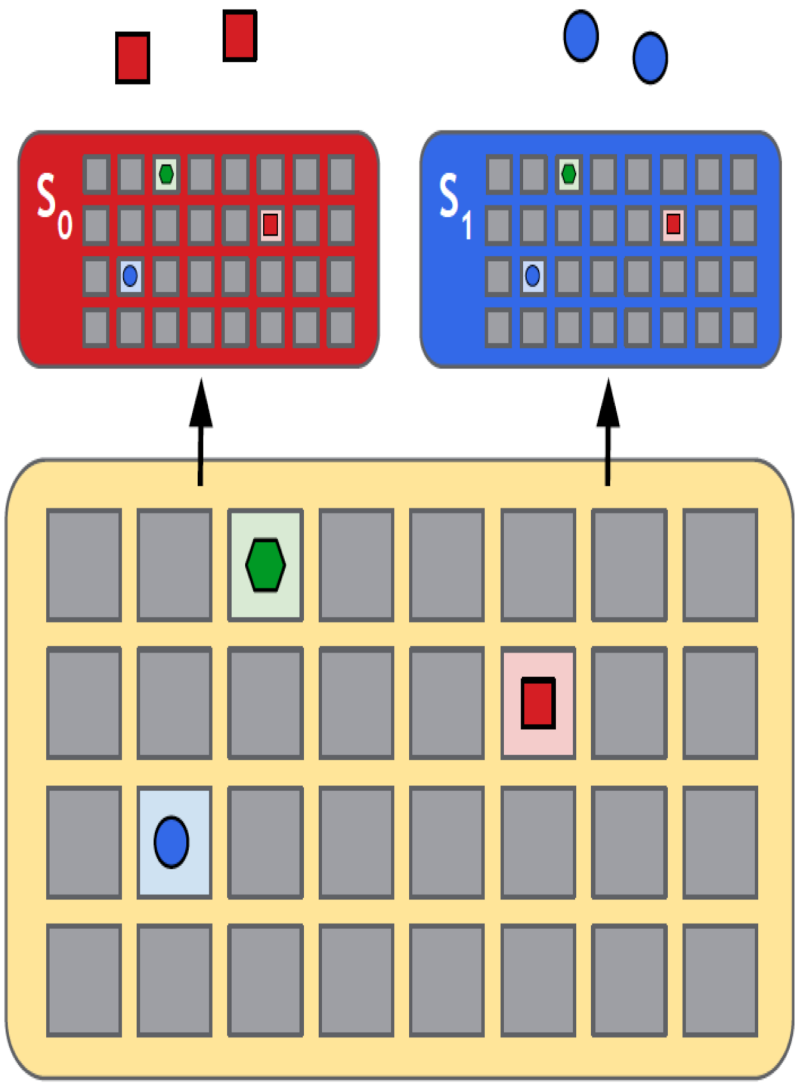
Existing Approach, Mesos revised



Disadvantages:

- Pessimistic Concurrency Control: Mesos avoids conflicts by offering a given resource to one framework at-a-time. Mesos chooses the order and the sizes of the offers. One framework holds the lock of the resources for the duration of the decision → slow.
- A framework does not have access to all the cluster state. So, it cannot support preemption or getting all resources.

Omega: Shared State, Overview



Cell State: a resilient master copy of the resource allocations of the cluster

No centralised scheduler. Each framework maintains its own scheduler. Each scheduler has its own private, local, frequently-updated copy of the cell state which uses to make decisions.

Omega Shared-state Scheduling, Steps

1. A framework’s scheduler makes a decision
2. Updates the shared cell state copy in an atomic commit
3. At most one such commit will succeed
4. The scheduler resyncs its local copy and if needed re-schedules

Omega Scheduling, Remarks

- Omega schedulers operation in parallel
- Schedulers typically do incremental transactions to avoid starvation
- Schedulers agree on common scale for expressing the relative importance of jobs
- Performance viability of the shared-state approach is determined by the frequency at which transactions fail and their costs!!!

- "*Our performance evaluation of the Omega model using both lightweight simulations with synthetic workloads, and high-fidelity, trace-based simulations of production workloads at Google, shows that optimistic concurrency over shared state is a viable, attractive approach to cluster scheduling.*"

Material

1. Mesos: A Platform for Fine-Grained Resource Sharing in Data Center

   by B. Hindman, A. Knwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, I. Stoica from University of California, Berkeley, NSDI 2011.

   Sections 1-3.5, 6-6.1.2

2. Omega: flexible, scalable schedulers for large compute clusters

   By Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek and John Wilkes, EuroSys 2013.

   Sections 1-3, 8