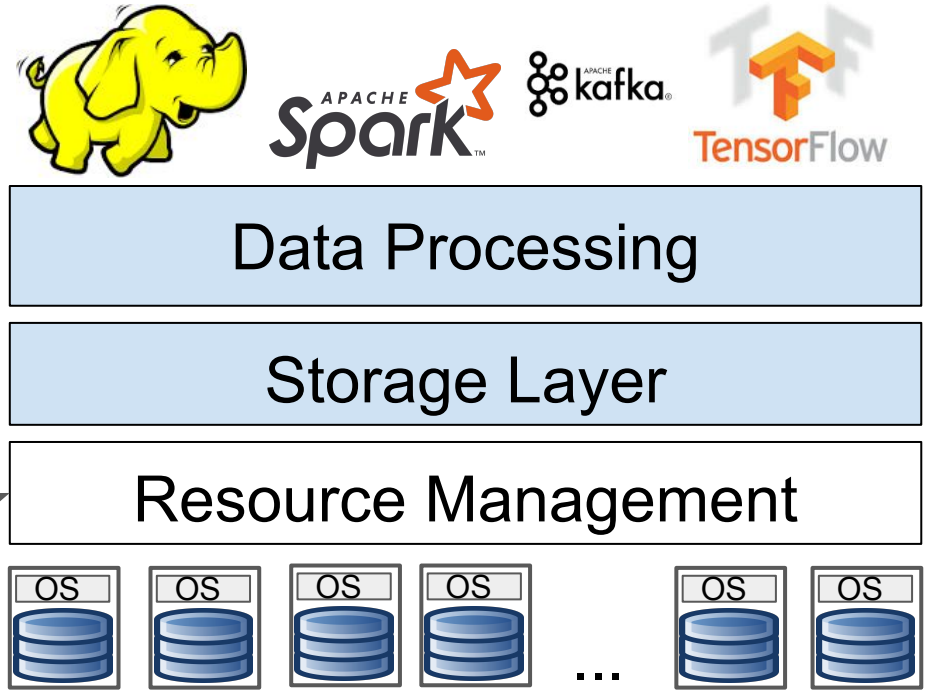
Databases and Big Data

Resource Management

Recap

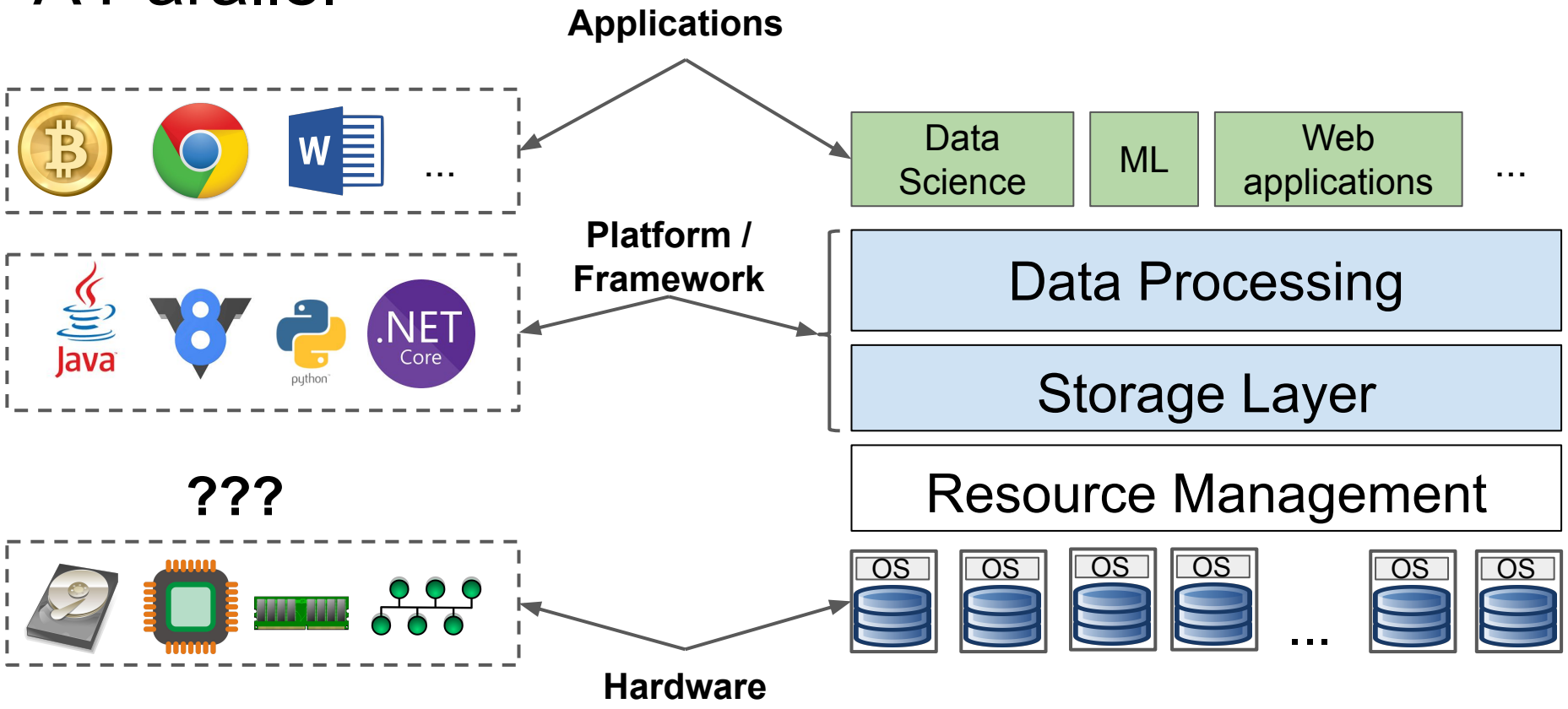
- Hadoop stack
  - HDFS: distributed file system
  - MapReduce: data processing
- See more data processing options this week
  - Even more next weeks



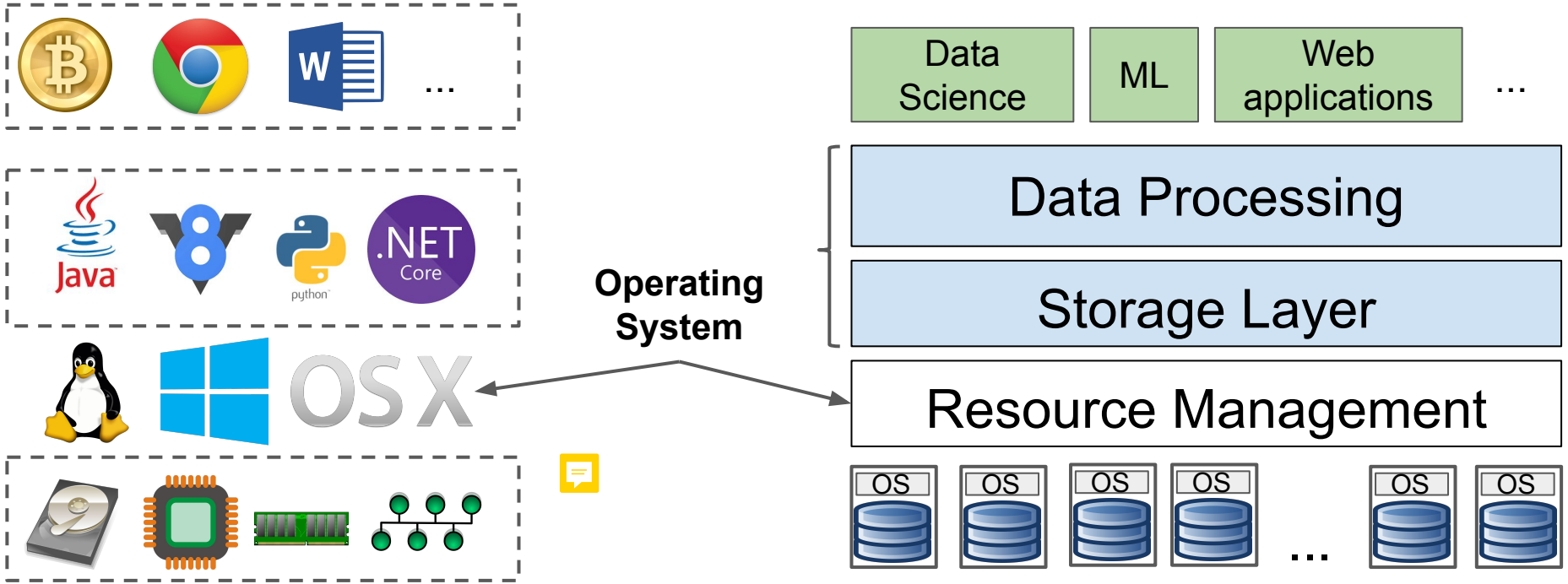
Agenda

- Why Resource Management
- YARN
- Mesos

A Parallel

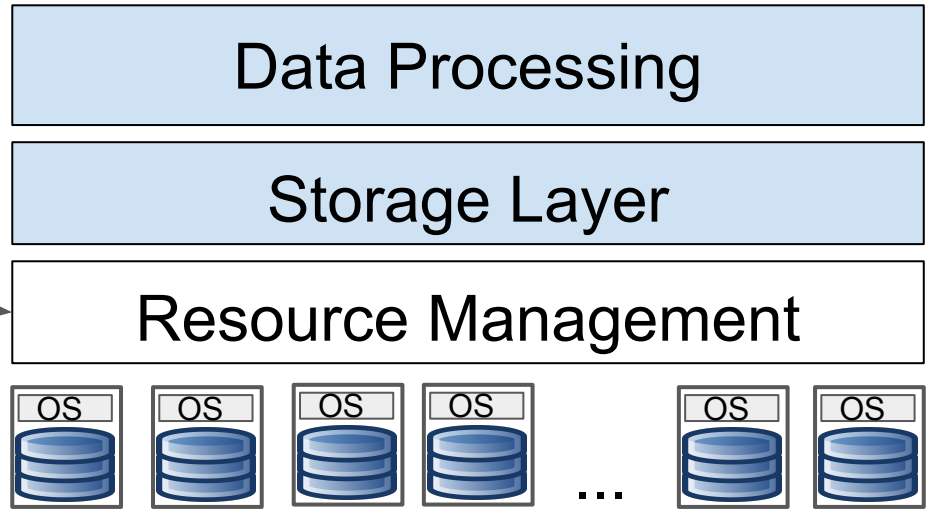

Applications
...
Data Science
ML
Web applications
...
Platform / Framework
Java
python
.NET Core
Data Processing
Storage Layer
Resource Management
???
OS
OS
OS
OS
...
OS
OS
Hardware

A Parallel


...
Java
.NET Core
python
OS X
Operating System
Data Science
ML
Web applications
...
Data Processing
Storage Layer
Resource Management
OS
OS
OS
OS
...
OS
OS

A Parallel

Does what your OS does, but for data center hardware.



What does **your** OS do?

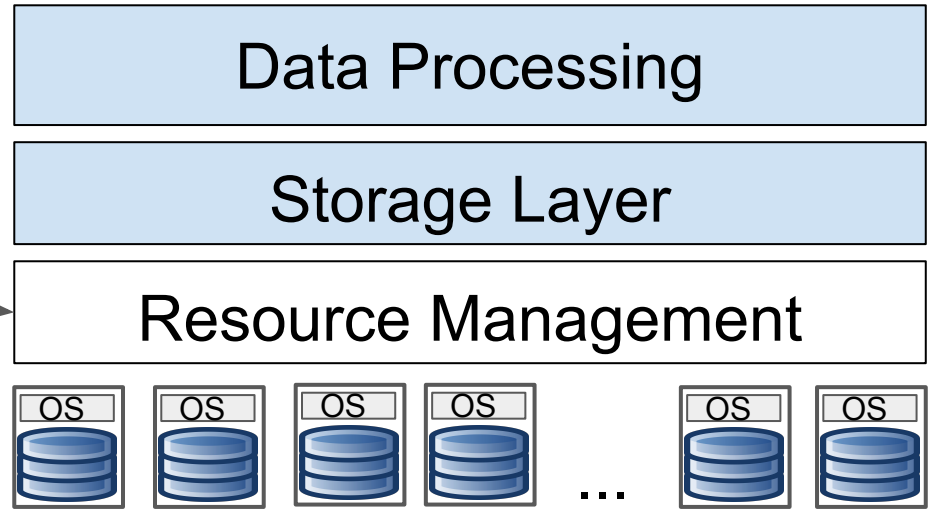
- Provide resource abstraction
- Manage resource sharing
- Isolate applications

A Parallel

when a hardware fail, you dk if it failed or just being slow

Does what **your OS** does,
but for data center hardware.

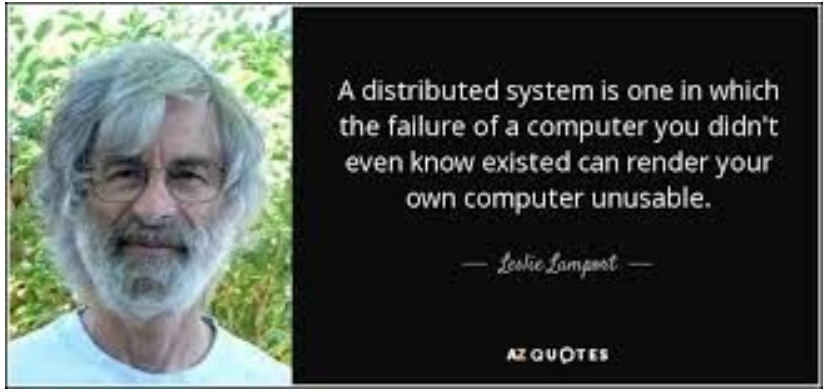
More difficult!



Failure

Failure

Failure



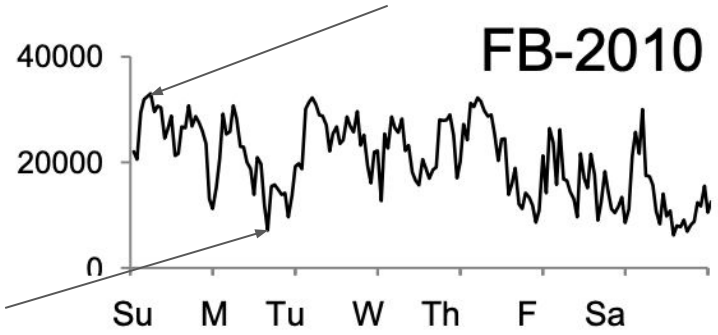
Why Difficult

What your OS provides, But people who use data centres are engineers who do not need this. They want to increase utilisation instead

- Goal:
  - Your OS:
    - Easy, and safe, to program
  - Data center:
    - **Utilization**
      - Idle resources waste money!
      - Or: increase utilization **saves** money!

One of Borg's primary goals is to make efficient use of Google's fleet of machines, which represents a significant financial investment; increasing utilization by a few percentage points can save millions of dollars. This section dis-

Provision for the peak!



Has to have enough HW to provide for the peak utilisation. But HW is just idle when utilisation is lower than the peak value.

What to fill the valleys?

Why Difficult

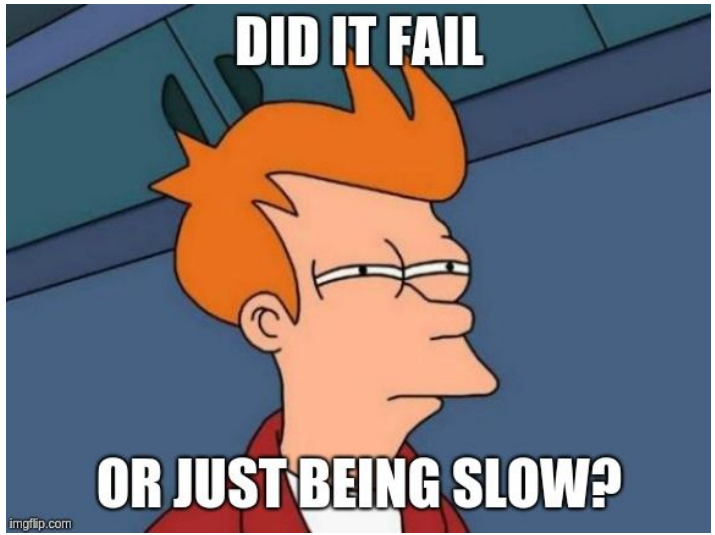
- Scale
  - Your OS:
    - 100s of processes before overheated
  - Data center:
    - Thousands of jobs per day

Trace	Machines	Length	Date	Jobs	Bytes moved
CC-a	<100	1 month	2011	5759	80 TB
CC-b	300	9 days	2011	22974	600 TB
CC-c	700	1 month	2011	21030	18 PB
CC-d	400-500	2+ months	2011	13283	8 PB
CC-e	100	9 days	2011	10790	590 TB
FB-2009	600	6 months	2009	1129193	9.4 PB
FB-2010	3000	1.5 months	2010	1169184	1.5 EB
Total	>5000	≈ 1 year	-	2372213	1.6 EB

Table 1: Summary of traces. CC is short for "Cloudera Customer". FB is short for "Facebook". Bytes moved is computed by sum of input, shuffle, and output data sizes for all jobs.

Why Difficult

- Failures
  - Your OS:
    - You know
  - Data center:
    - You don’t know if it really fails, or just slow

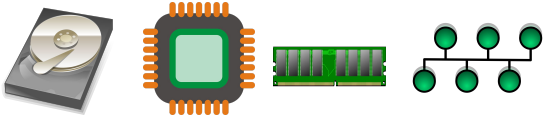


Why Difficult

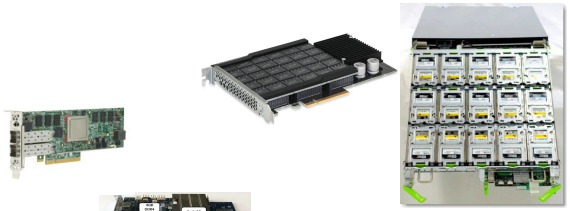
- Sharing
  - Your OS:
    - Your applications share resources
  - Data center:
    - Strangers share resources
    - Can’t rely on people not hogging resources.

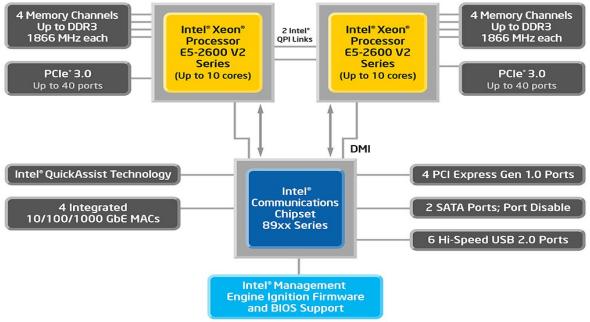
Why Difficult

- Type of resources
  - Your OS



  - Data center



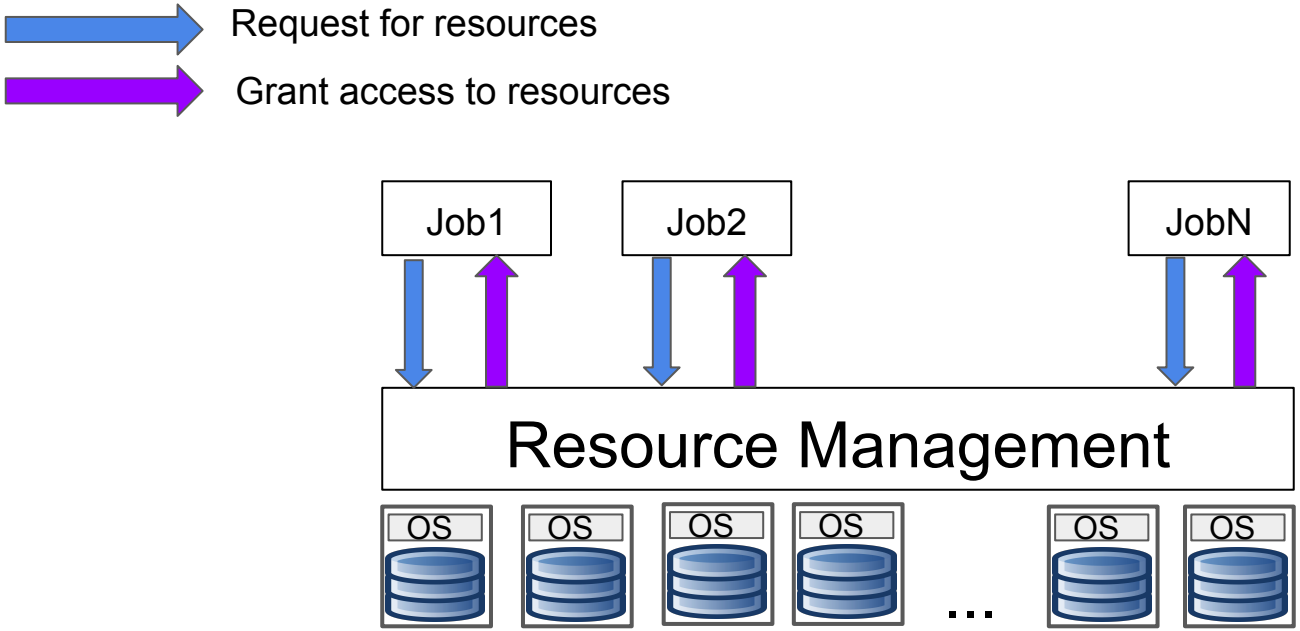


Summary

	Your Local OS	**Data Center's Resource Management System**
Goal	Easy to use. Isolation	Utilization Isolation
Scale	10s of processes	1000s of jobs Each job consists of many tasks
Failure	Fail together	Fail independently
User (sharing)	Single user	Multi-tenant
Resources	Small set of resources	Large set of resources

Summary

- Resource management for data center



Google’s Worst-Kept Secret

“I prefer to call it the system that will not be named.”

John Wilkes

CADE METZ BUSINESS 03.05.13 06:30 AM

Return of the Borg: How Twitter Rebuilt Google's Secret Weapon

History

- 200x: no Borg!
- 2011: Mesos published
- 2012: YARN released
- …
- 2015: Borg published

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

Benjamin Hindman, Andy Konwinski, Matei Zaharia,
Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica
University of California, Berkeley

23 May, 2012: Release 2.0.0-alpha available

This is the first (alpha) version in the hadoop-2.x series.

This delivers significant major features over the currently stable hadoop-1.x series including:

- HDFS HA for NameNode (manual failover)
- YARN aka NextGen MapReduce
- HDFS Federation
- Performance
- Wire-compatibility for both HDFS and YARN/MapReduce (using protobufs)

Please see the Hadoop 2.0.0-alpha Release Notes for details.

Large-scale cluster management at Google with Borg

Abhishek Verma† Luis Pedrosa‡ Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.

YARN Yarn - **Y**et **A**nother **R**esource **N**egotiator

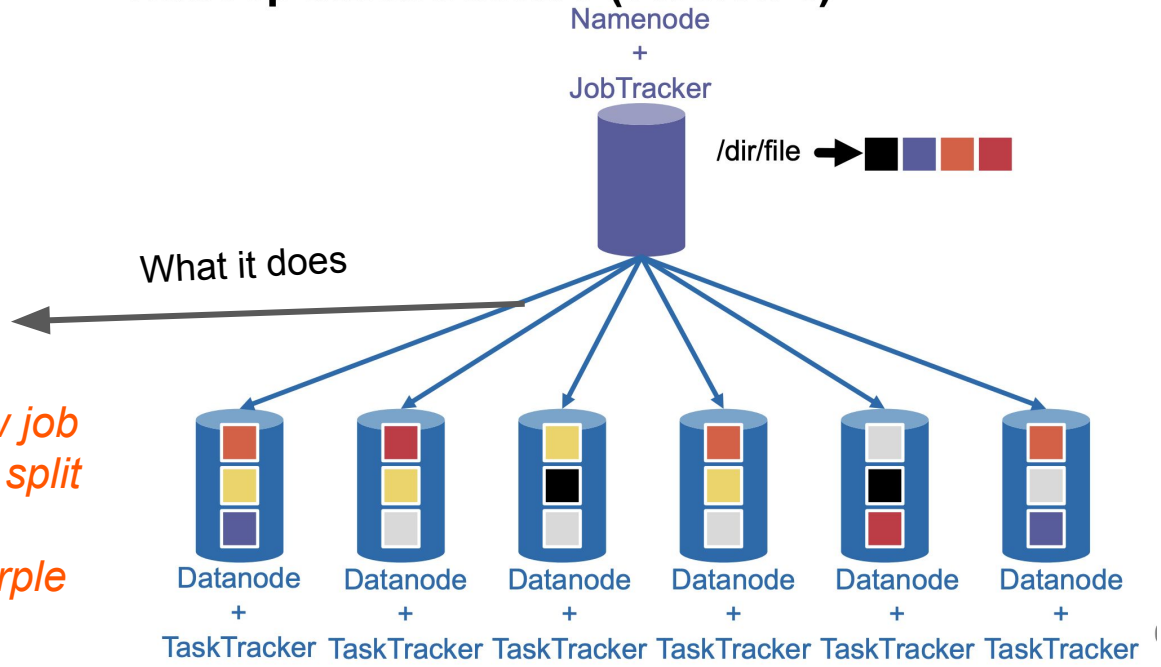
- General resource management system
- But designed with Hadoop in mind
  - Synonymous to Hadoop v2
  - Solve specific problems with Hadoop v1

Problems with Hadoop v1

namenode has the meta data to access the data nodes.

- Recall the architecture

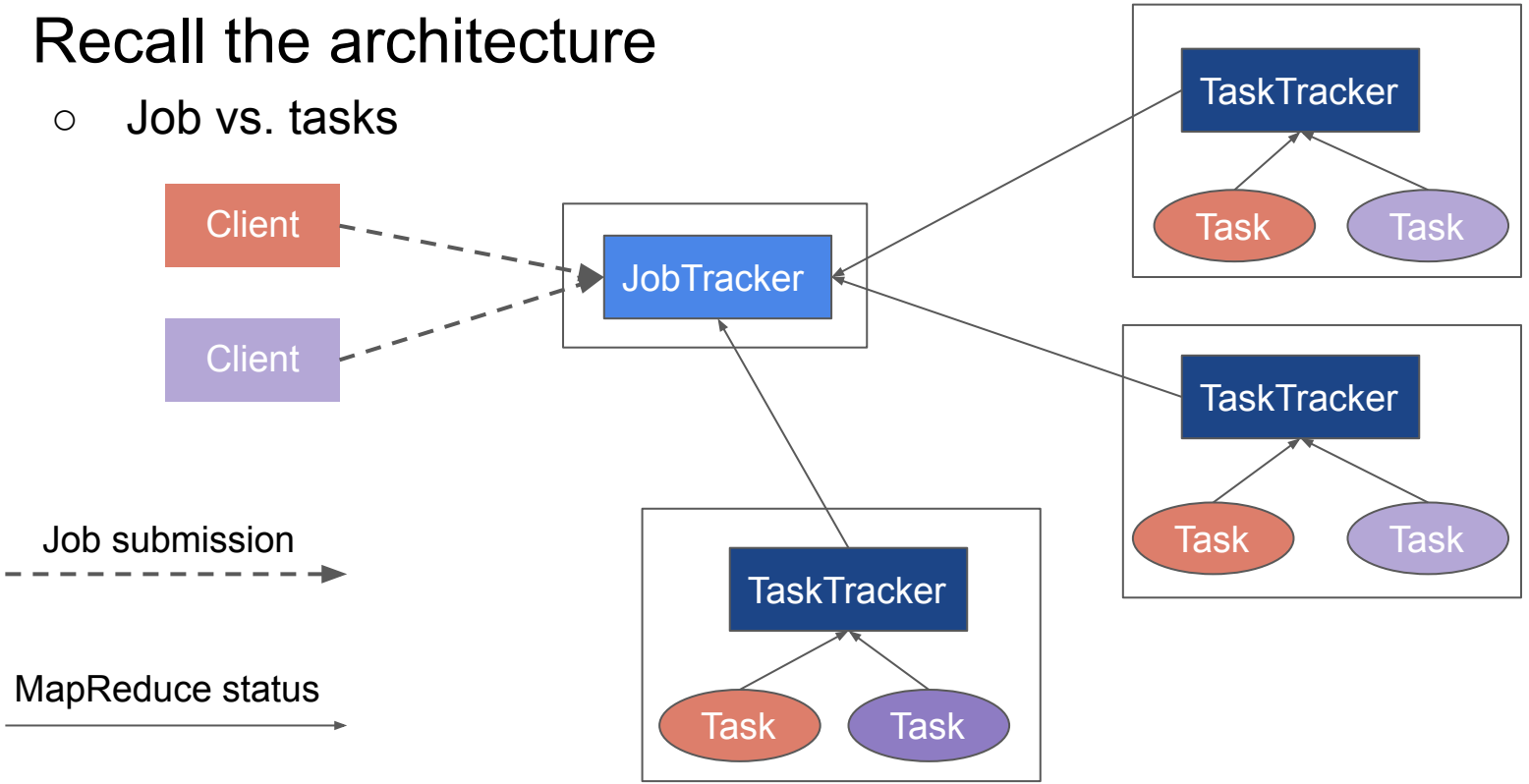
Hadoop infrastructure (version 1)



Client contact Jobtracker to start a new job
JT starts slave nodes (tasktrackers) to split the jobs into tasks.
JT knows and monitor red jobs and purple jobs where are they running (in which nodes)
Monitor by communicating with tasktracker (next page)

Problems with Hadoop v1

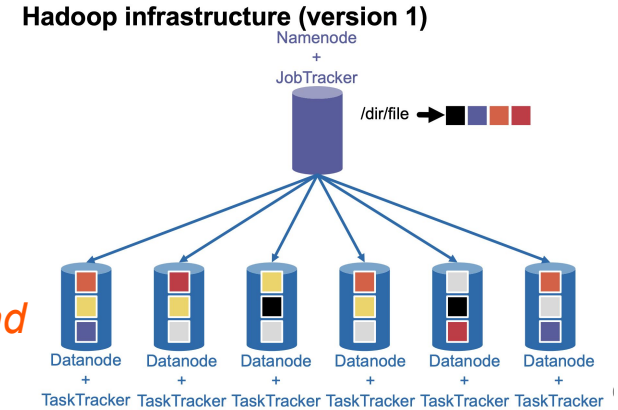
- Recall the architecture
  - Job vs. tasks



Problems with Hadoop v1

- JobTracker does too many things
  - Only scale to < 4K nodes, 40K tasks
  - Performance bottleneck
    - (Almost) All requests go through it
  - Jack of all trades:
    - Scheduling
    - And monitoring

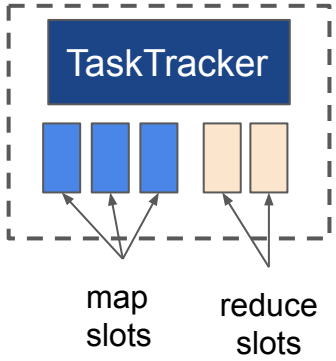
whenever you wanna read some data or wanna run some MapReduce jobs

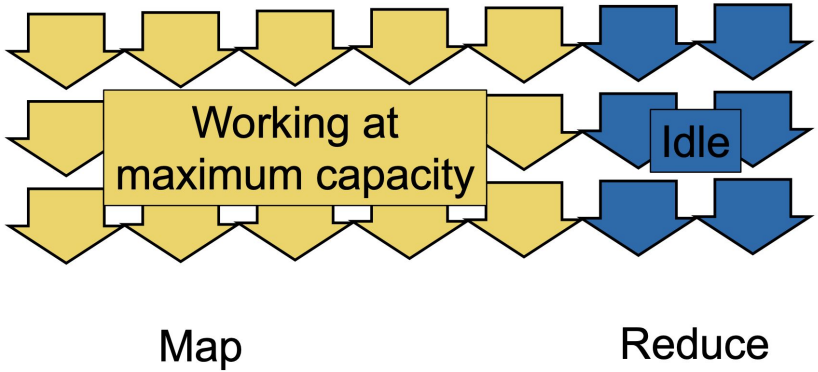


Scheduling: whenever request come, needs to know where to distribute the task to
Monitoring: detect failure and boot up from other nodes

Problems with Hadoop v1

- Utilization
  - Fixed slots: map slots and reduce slots
  - Configured at the beginning
  - Non-fungible: map tasks cannot run on reduce slots

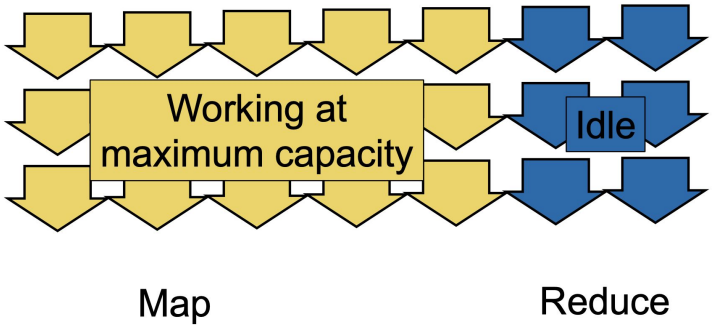




most of time you dont start reduce task until map tasks are finished

Problems with Hadoop v1

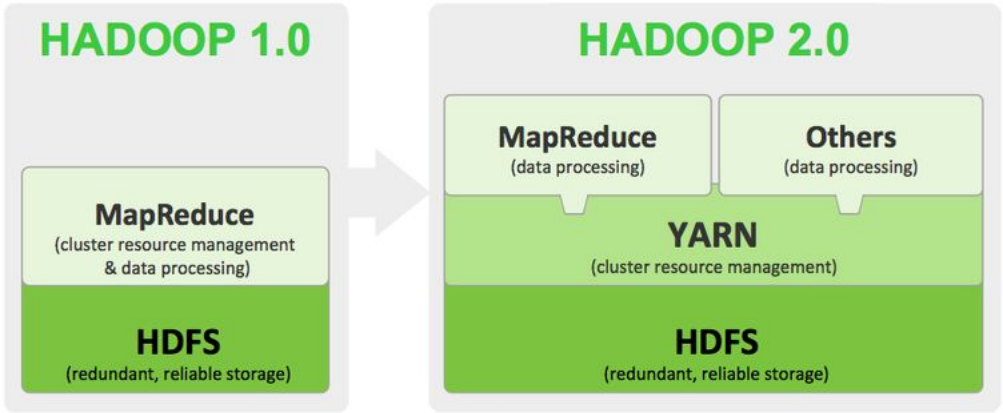
- Example
  - Job has 100 map tasks, 100 reduce tasks
  - Cluster has 10 servers
  - Each server has 10 cores:
    - 5 map slots (max)
    - 5 reduce slots (max)
  - Under-utilized:
    - 50 map tasks running at a time
    - Server is only 50% utilized



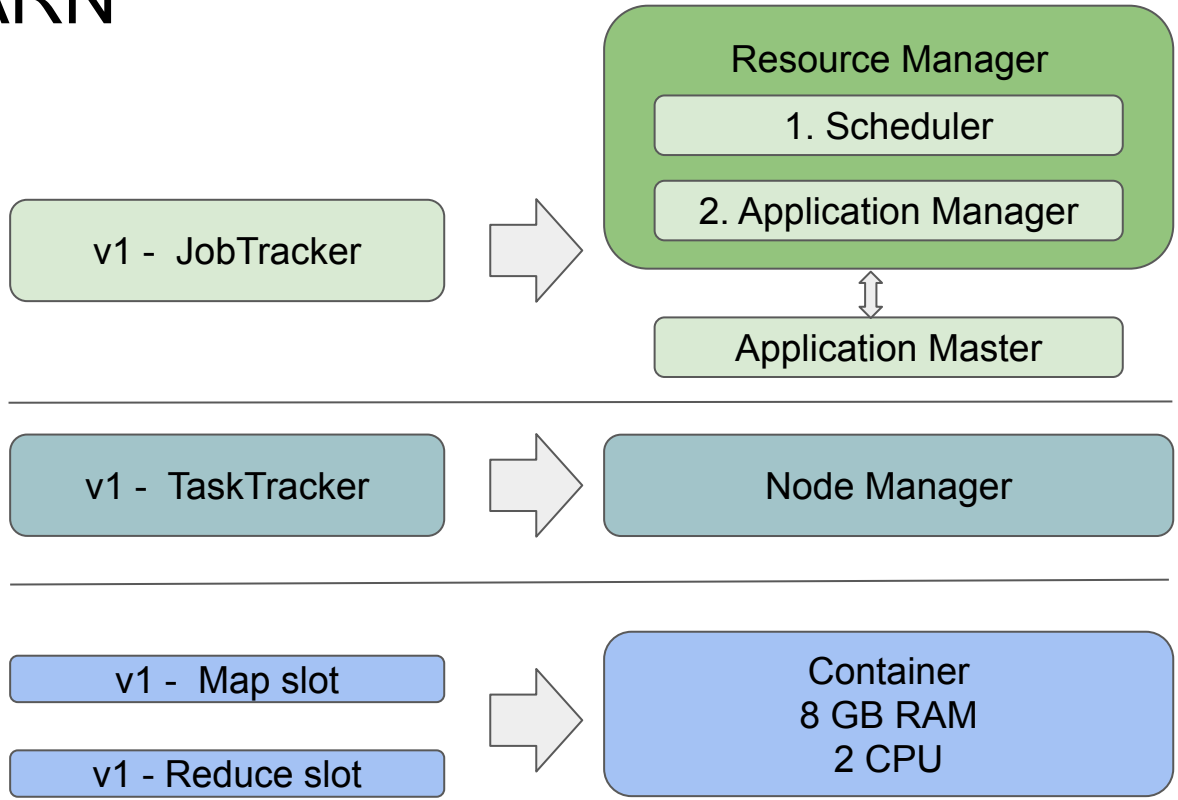
50 map tasks running
50 reduce slots are still free

Enter Hadoop v2.0

- YARN solves:
  - Jack-of-all-trades problem
  - Utilization problem
- And more features:
  - Better sharing of the cluster
    - User-defined sharing policies



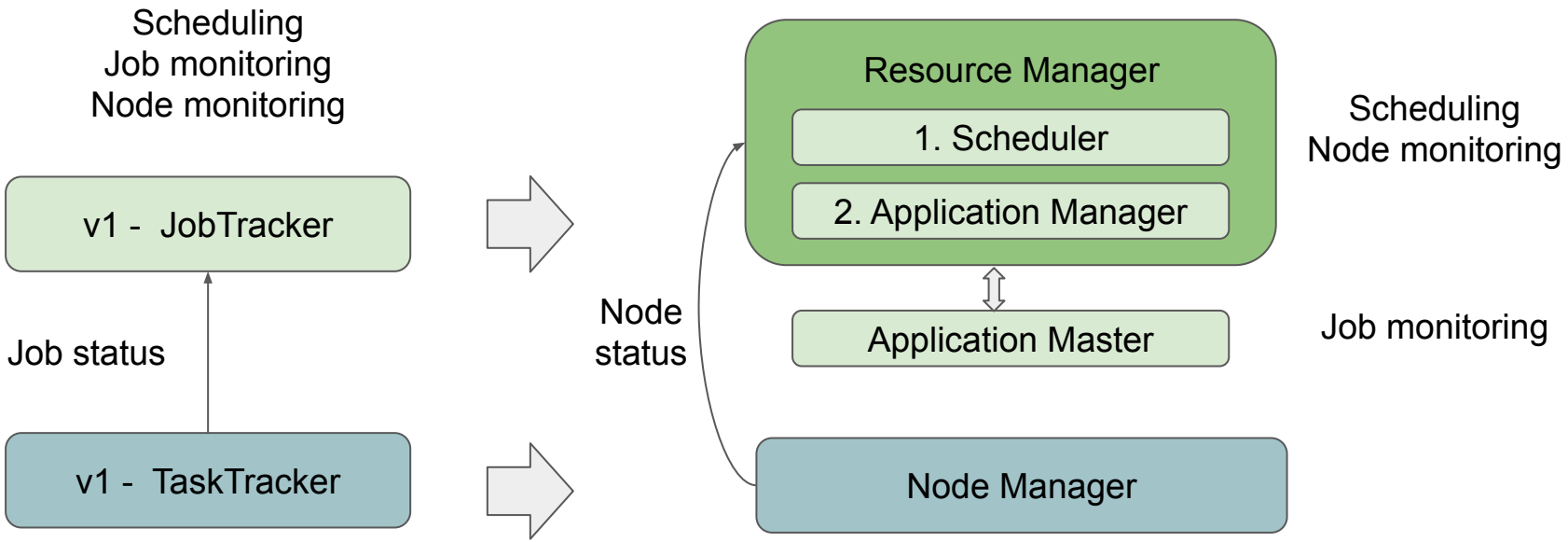
YARN



- dont need to specify how many map/ reduce slots
- now more flexible

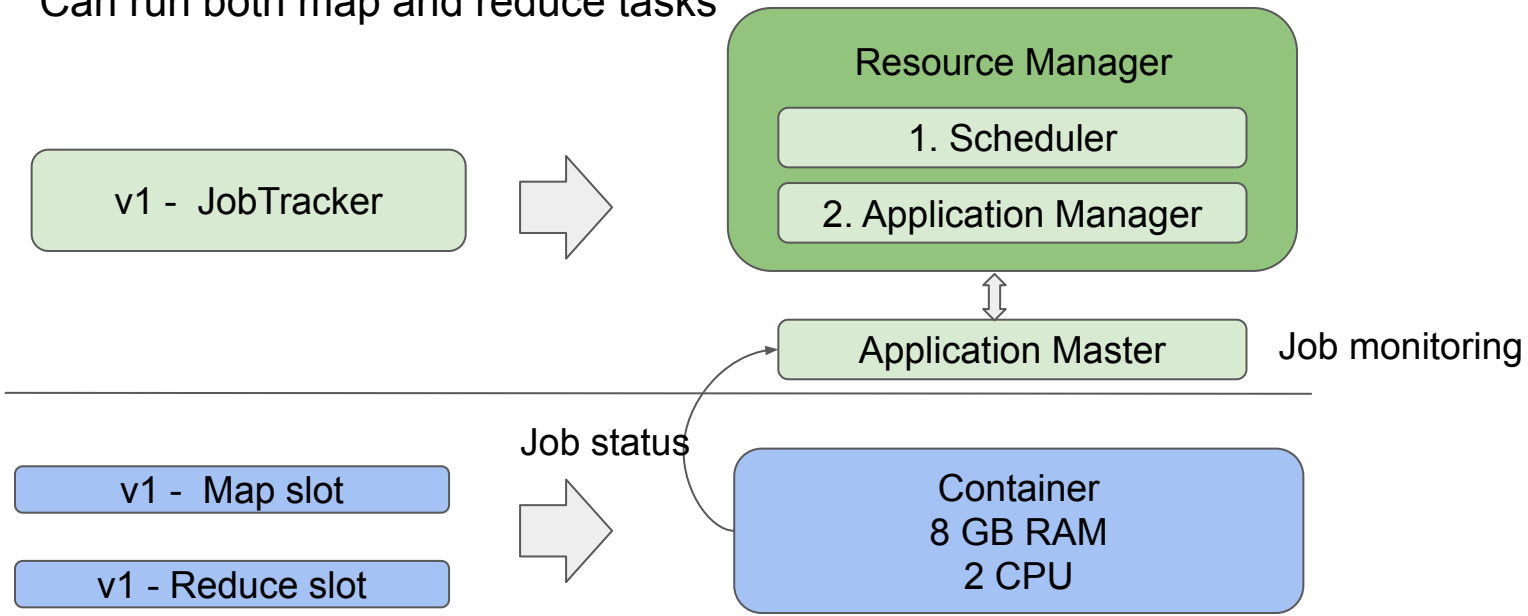
YARN

- Jack-of-all-trades → separate functionalities



YARN

- Utilization → *fungible* **containers**
  - Can run both map and reduce tasks

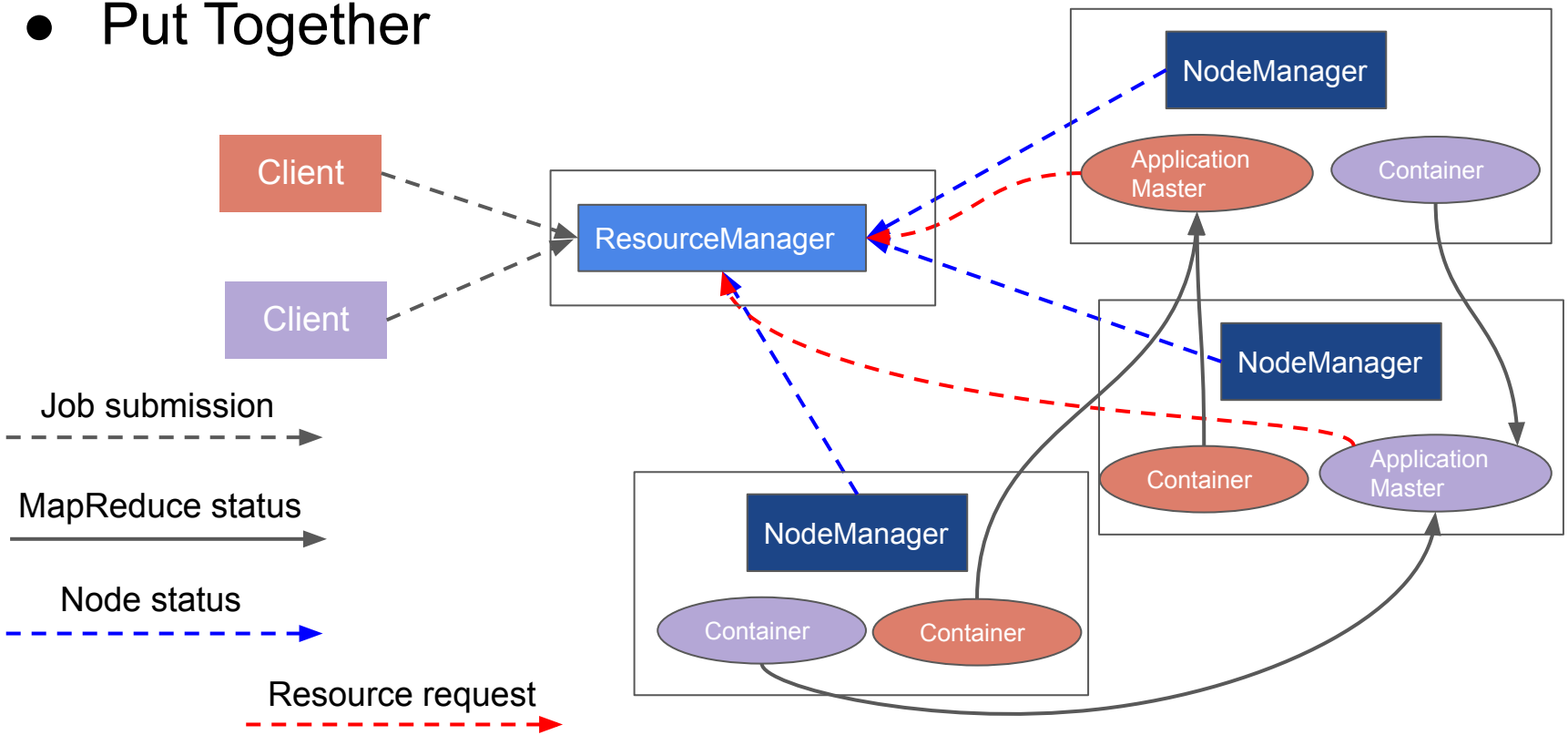


YARN

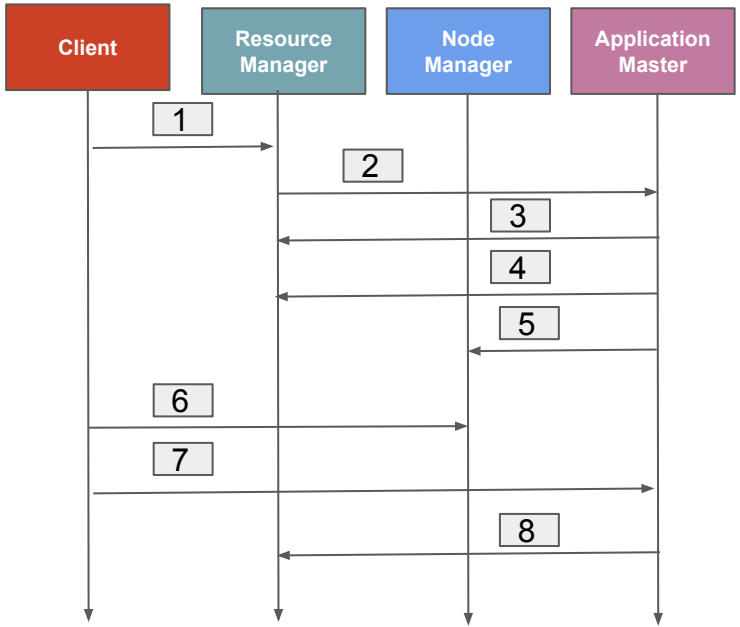
- Put Together

2 Application masters *-> one for red, one for purple; stays in one of the data node*

Nodemanager *-> only know how many resource still running, dont know about the job*



YARN Workflow



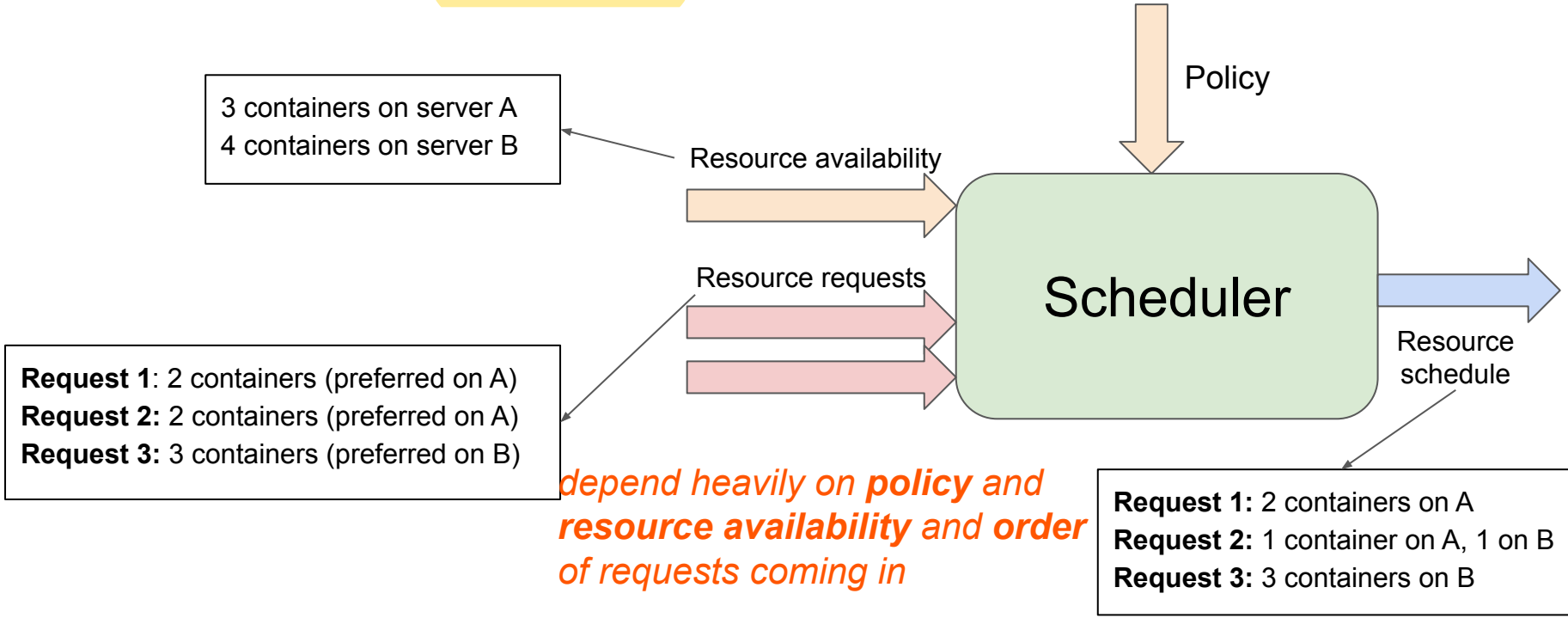
1. Client submits an application
2. Resource Manager (RM) allocates a container to start Application Master
3. Application Master registers with RM
   (start asking for resources from RM)
4. AM ask containers from RM
5. AM notifies Node Manager to launch containers
   (i have been granted these resources, gimme them)
6. Application code is executed in the container
7. Client contacts AM to monitor application status
8. AM unregisters with RM
   (release all the resources in step 8)

YARN Scheduler

Scheduler runs in the Resource Manager

(when request from App Manager goes to RM)

- What does a scheduler do?



YARN Resource Request

- Application Master asks for resources
  - Containers with X CPUs and Y memory
- When?
  - Up front, when the application starts
  - Or during execution, e.g:
    - At the beginning of map phase
    - At the beginning of reduce phase

Work in progress

Memory

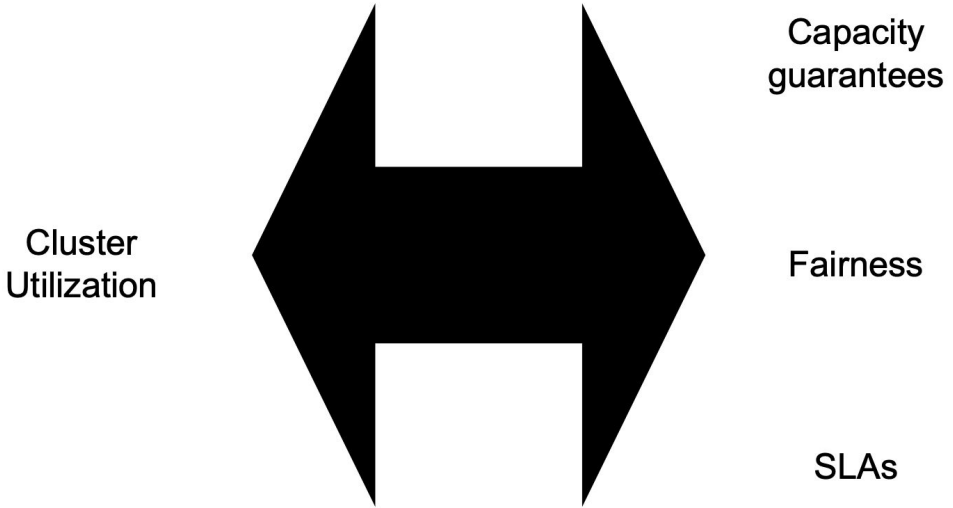
CPU

Disk

Network

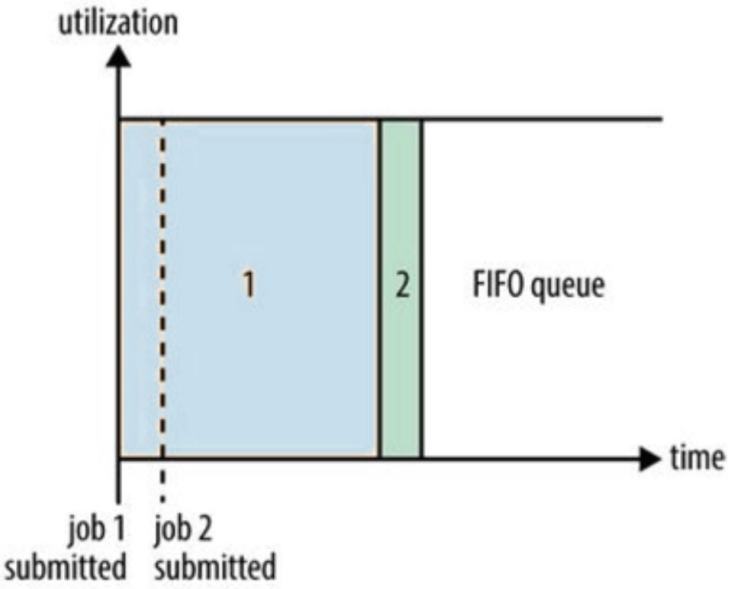
YARN Scheduler

- Why difficult?
  - Multi-objective optimization problem



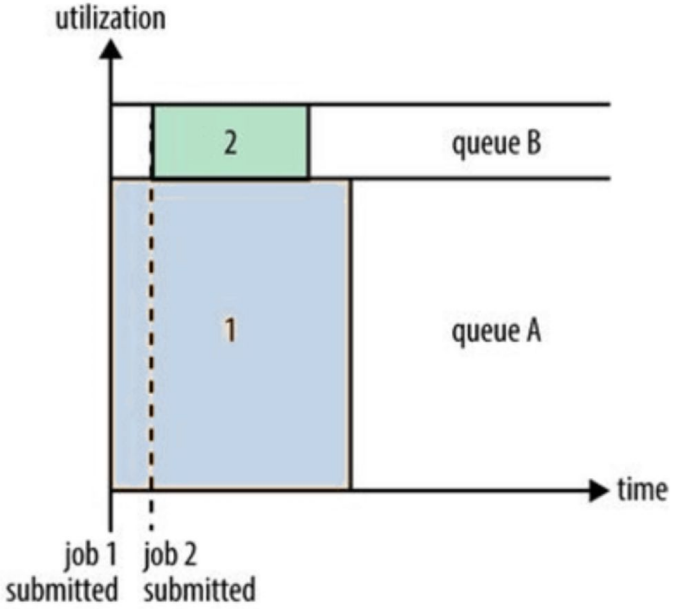
YARN Scheduler

- User can write his/her own
- YARN provides some
- FIFO
  - Simplest
  - Worst for short jobs
    - Queued behind long running jobs



YARN Scheduler

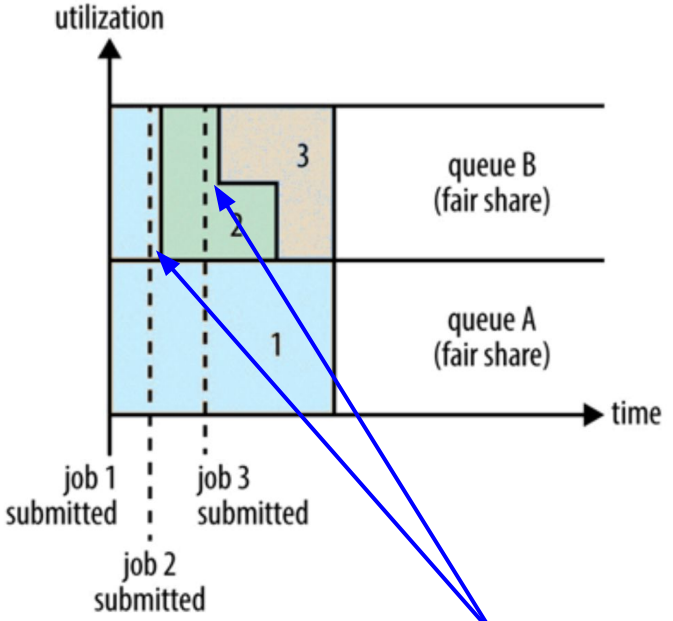
- Capacity Scheduler
  - Statically divide resources into multiple queues
    - E.g: one for short, another for long jobs
  - Request is submitted to one queue
  - Within queue, FIFO



job1 could use all resources in the cluster
then job 2 came in so they have to share

YARN Scheduler

- Fair Scheduler
  - No static queue
  - Simple policy: all jobs have equal shares
  - More complex:
    - Jobs in queue A has minimum of X%
    - Jobs in queue B has minimum of Y%
  - No other job running -> can use all resources



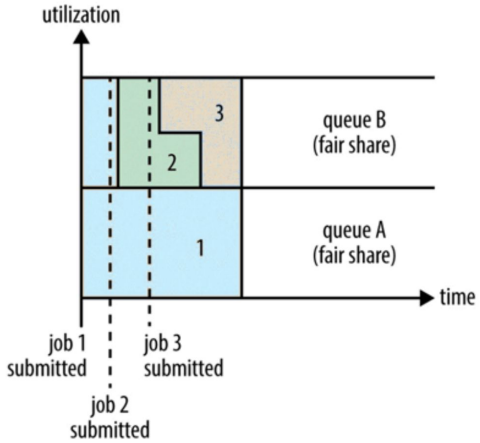
Why these lags?

got lags as compared to capacity scheduler

lags -> when u submit job 2, need to kill resources allocated to other jobs (job1) before you can allocate it to this job (job2)

YARN Scheduler

- How to enforce fair share?
  - Pre-emption: kill resources of existing job
- Multi-resource request?



Memory (total 1 TB)

CPU (total 100 cores)

Application A: 300 GB, 4 cores

30% Memory, 4% CPU

Application A: 10 GB, 50 cores

1% Memory, **50%** CPU

Some jobs are CPU heavy, some are memory heavy.

Add dominant resources tgt (30 + 50 = 80)

YARN Scheduler

Calculating the percentage for sharing
30 / 80 = 37.5%
50 / 80 = 62.5%

For Fair sharing where we need to know these percentages

- What's the share of A and B:
  - Dominant Resource Fairness (DRF)

Memory (total 1 TB)

CPU (total 100 cores)

Application A: 300 GB, 4 cores

30% Memory, 4% CPU

37.5%

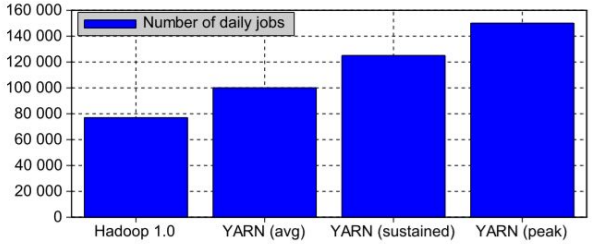
Application A: 10 GB, 50 cores

1% Memory, **50%** CPU

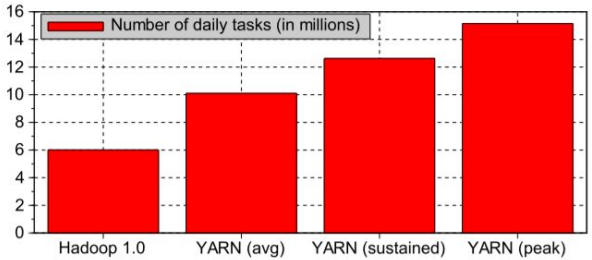
62.5%

YARN @ Yahoo!

- YARN processes 500,000 jobs daily
- When deployed on 2500 nodes
  - Jobs increased from **77k to 150k (peak)**
  - Tasks across jobs from **4M to 10M**
  - CPU utilization almost **doubled**
- Summary from Yahoo's operational team
  - "*Upgrading to YARN was equivalent to adding 1000 machines [to this 2500 machines cluster]*"



(a) Daily jobs



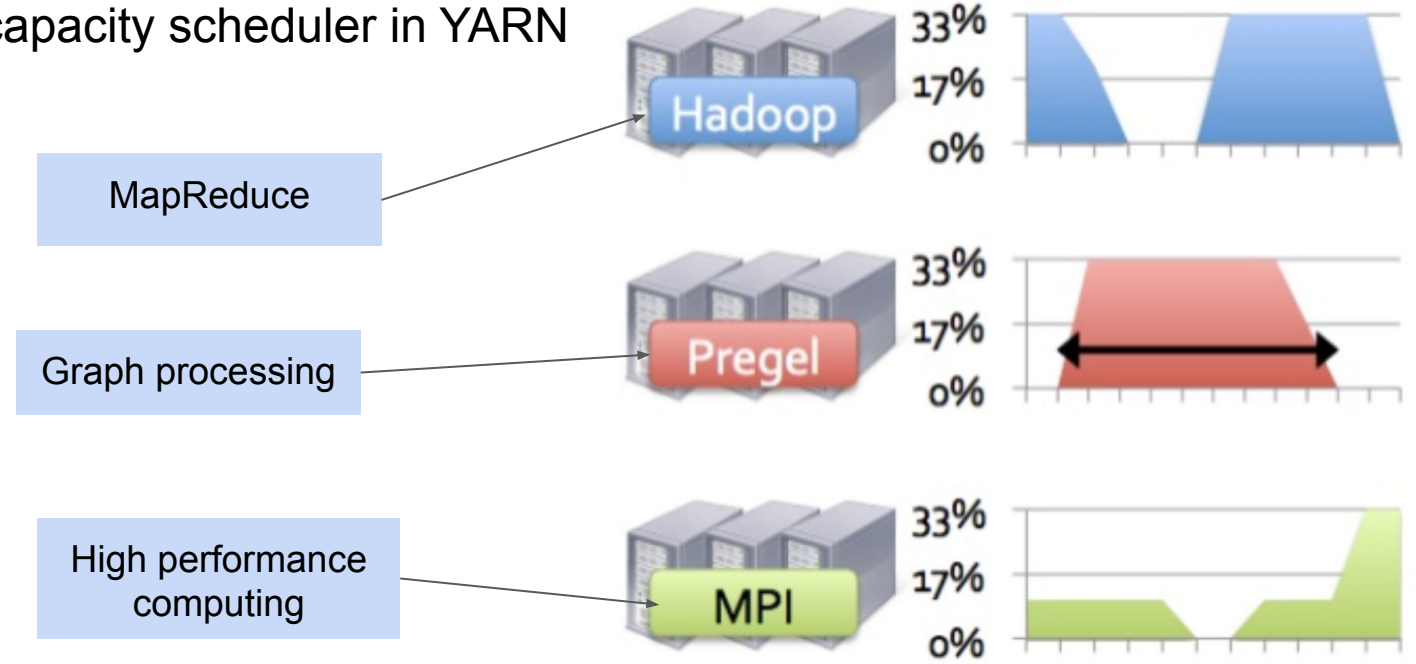
(b) Daily tasks

Mesos

- First of its kind
  - When Borg was still a secret
- Early 2010s:
  - Many frameworks sharing a cluster
    - MapReduce-effect

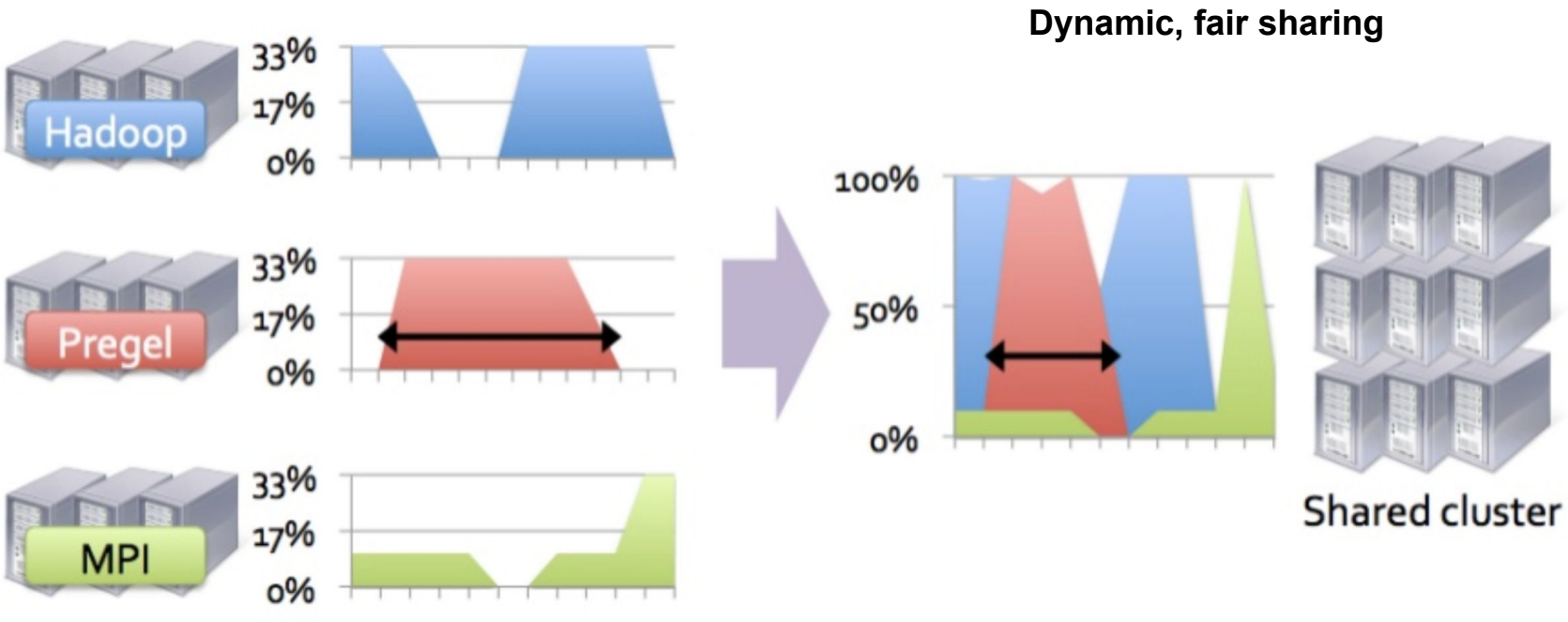
Mesos

- Static partitioning of resources
  - Recall capacity scheduler in YARN



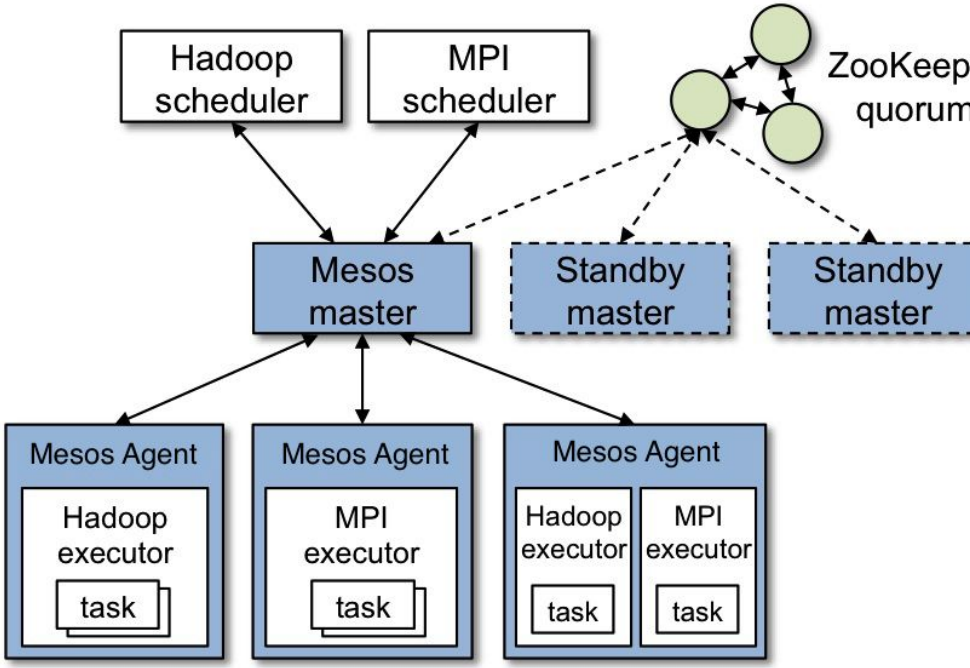
Mesos

- Goal


Dynamic, fair sharing
Hadoop
Pregel
MPI
33%
17%
0%
100%
50%
0%
Shared cluster

Mesos Architecture

- Master and slaves
- Framework:
  - Manage jobs
  - Like a JobTracker in Hadoop v1
- Executor:
  - One per framework
  - Launch & execute tasks
  - Like a TaskTracker in Hadoop v1

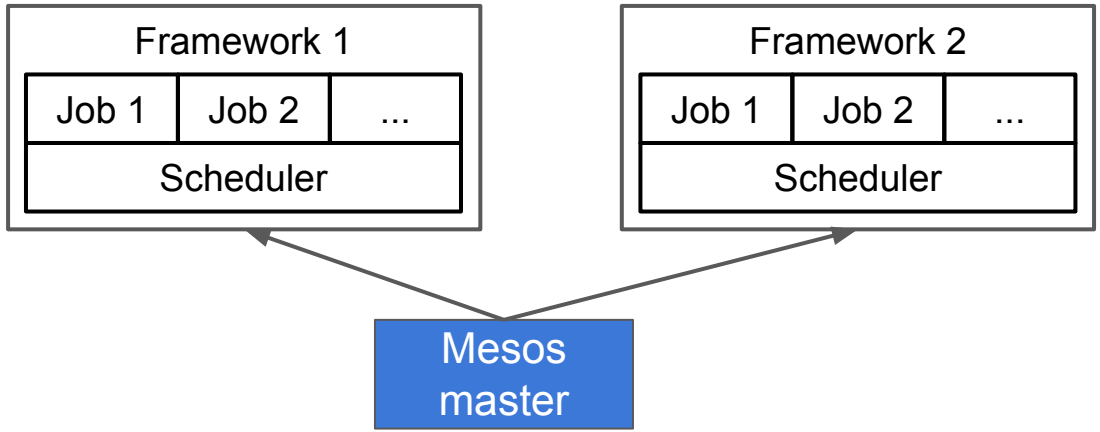


Mesos

- Framework gets resources from master
- Uses the assigned resources to schedule its jobs
  - Different to YARN?

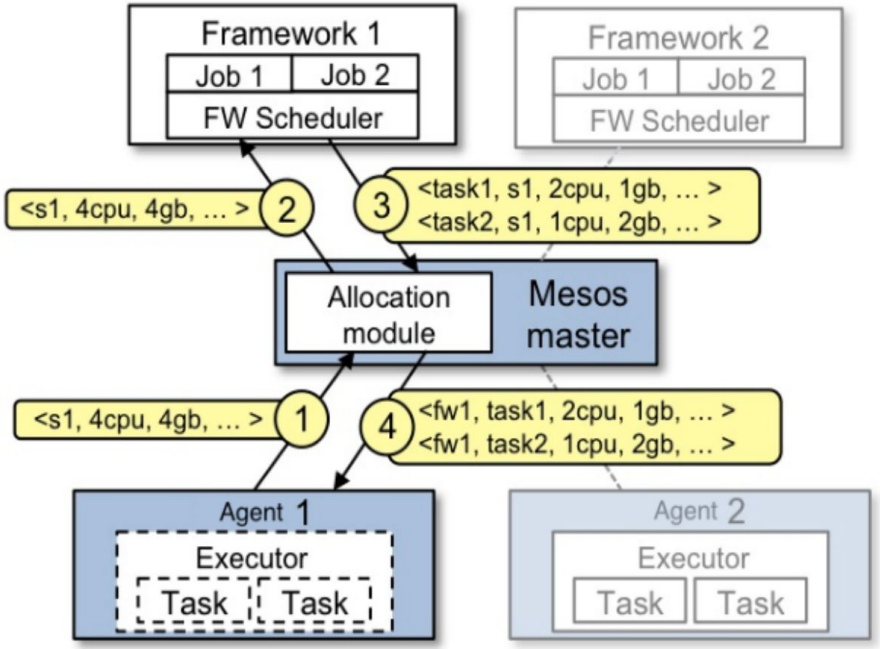
framework itself need to do the scheduling
Master does not do the scheduling!

*unlike **YARN** -> only hadoop so everything goes to the application manager (1 layer)*



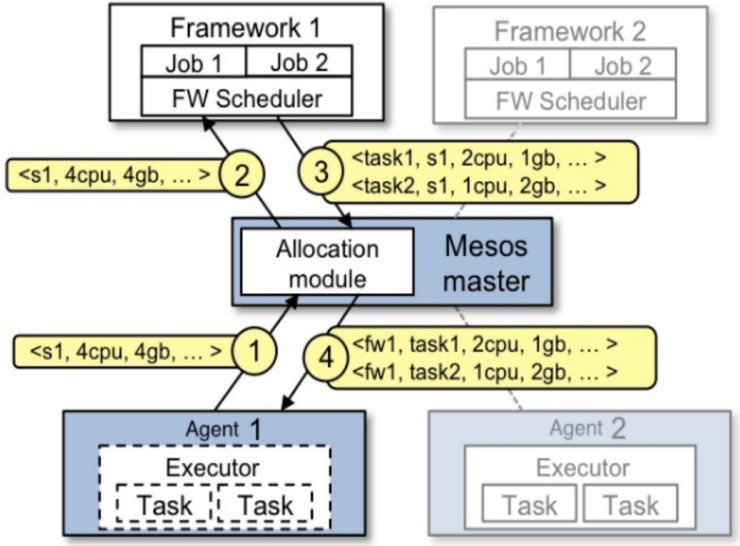
Mesos

- Resources are offered
  - Not requested
- Offers can be rejected!



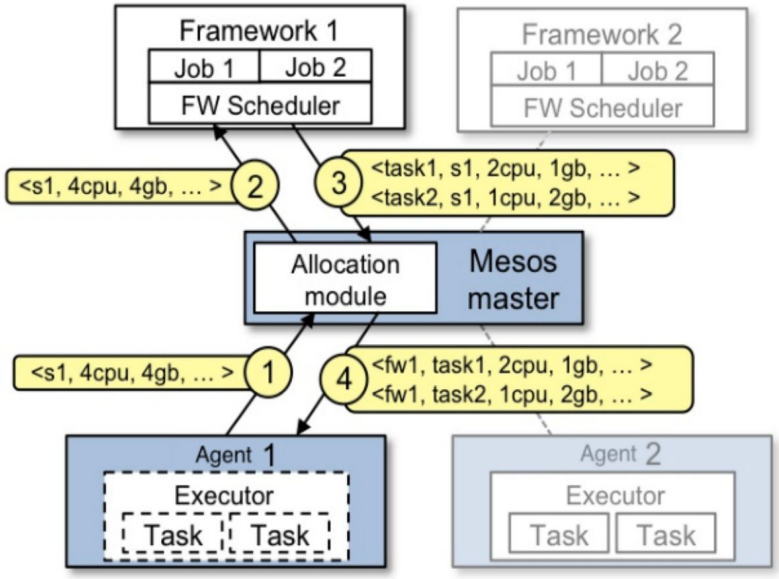
Mesos

- Offers:
  - Vectors of resources
  - E.g <machineID, #CPUs, RAM, Disk,..>
- How it works:
  - Step 1: nodes report available resources
  - Step 2: master make offers to framework
  - Step 3: framework accepts and create tasks
  - Step 4: master forwards tasks to slaves



Mesos

- Master is simple:
  - Collect available resources
  - Offer to framework
  - Forward tasks
- Coarse-grained scheduling:
  - Between frameworks
- Complexity pushed to Frameworks



Mesos

- Easy to add a new framework
  - Implement these APIs
- Spark is added like this

We have implemented four frameworks on top of Mesos. First, we have ported three existing cluster computing systems: Hadoop [2], the Torque resource scheduler [33], and the MPICH2 implementation of MPI [16]. None of these ports required changing these frameworks' APIs, so all of them can run unmodified user programs. In addition, we built a specialized framework for iterative jobs called Spark, which we discuss in Section 5.3.

Method and Description
disconnected(**SchedulerDriver** driver) Invoked when the scheduler becomes "disconnected
error(**SchedulerDriver** driver, java.lan Invoked when there is an unrecoverable error in the
executorLost(**SchedulerDriver** driver, **P** Invoked when an executor has exited/terminated.
frameworkMessage(**SchedulerDriver** drive Invoked when an executor sends a message.
offerRescinded(**SchedulerDriver** driver, Invoked when an offer is no longer valid (e.g., the sl
registered(**SchedulerDriver** driver, **Pro** Invoked when the scheduler successfully registers w
reregistered(**SchedulerDriver** driver, **P** Invoked when the scheduler reregisters with a newl
resourceOffers(**SchedulerDriver** driver, Invoked when resources have been offered to this fr
slaveLost(**SchedulerDriver** driver, **Prot** Invoked when a slave has been determined unreach
statusUpdate(**SchedulerDriver** driver, **P** Invoked when the status of a task has changed (e.g.,

Mesos

- When does a Framework reject offers?
  - Not enough resources
  - Or, because of locality constraints
    - E.g: MapReduce colocates tasks with HDFS blocks
- Delay Scheduling:
  - If doesn't satisfy locality constraints, waits a little bit (few seconds)

Data locality *refers to the ability to move the computation close to where the actual data resides on the node, instead of moving large data to computation. This minimizes network congestion and increases the overall throughput of the system.*

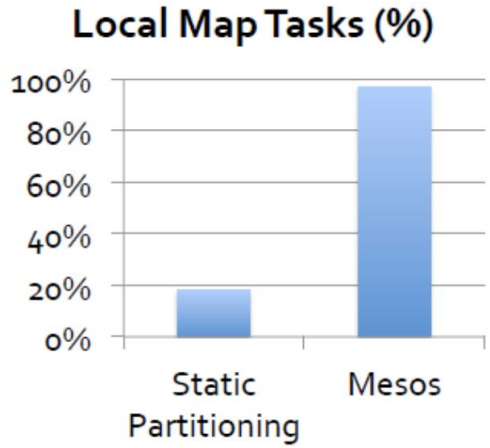
Instead of making decision right away, wait a lil bit then make decision.

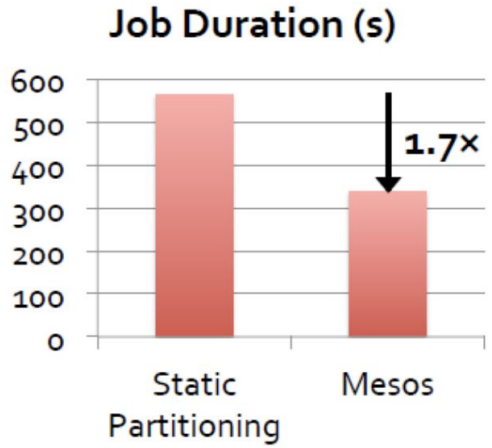
Mesos

- Delay scheduling
  - Surprisingly effective

Ran 16 instances of Hadoop on a shared HDFS cluster

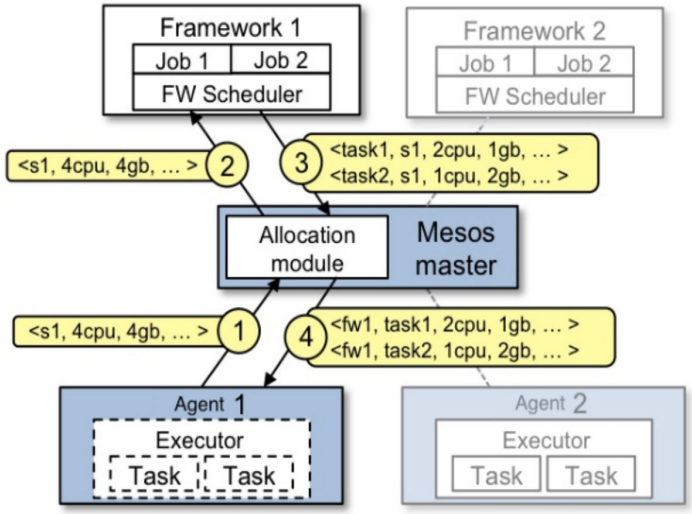
Used delay scheduling [EuroSys '10] in Hadoop to get locality (wait a short time to acquire data-local nodes)





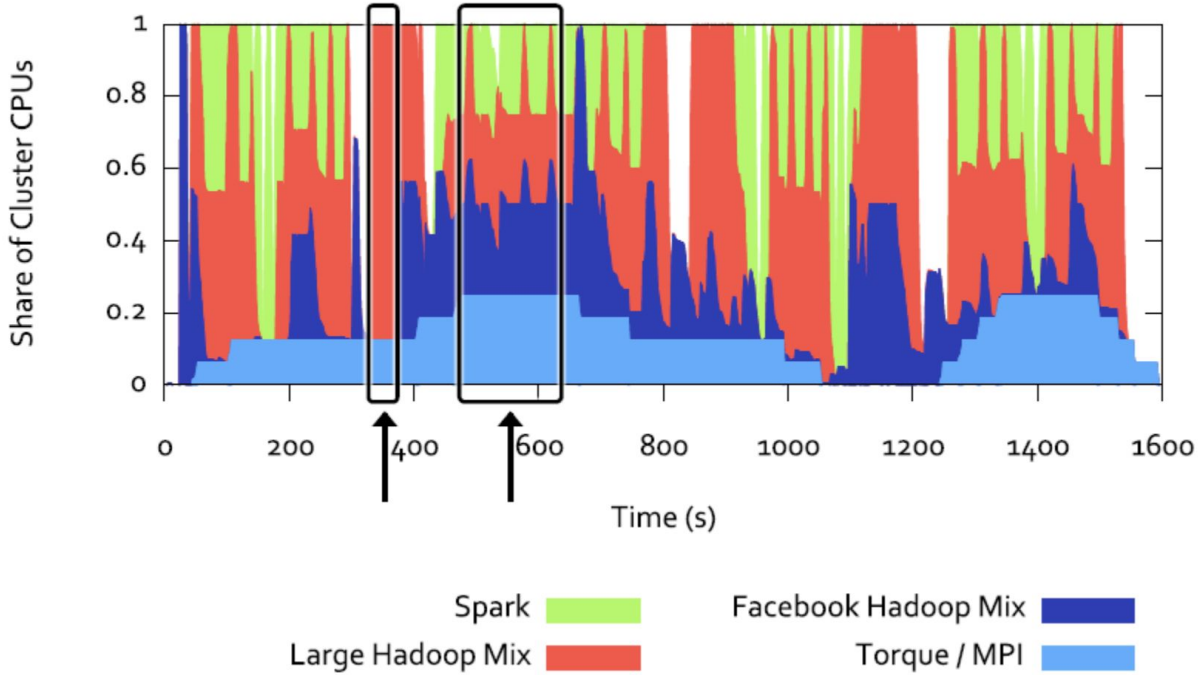
Mesos

- Allocation module:
  - Be default, broadcast all resources to Frameworks
  - Whoever accepts first get them
- But customizable:
  - Fair sharing
  - Priority
  - Non-revocable
  - etc.



Mesos

- Does it work?
  - Sharing



Mesos

- Does it work?
  - Scalability

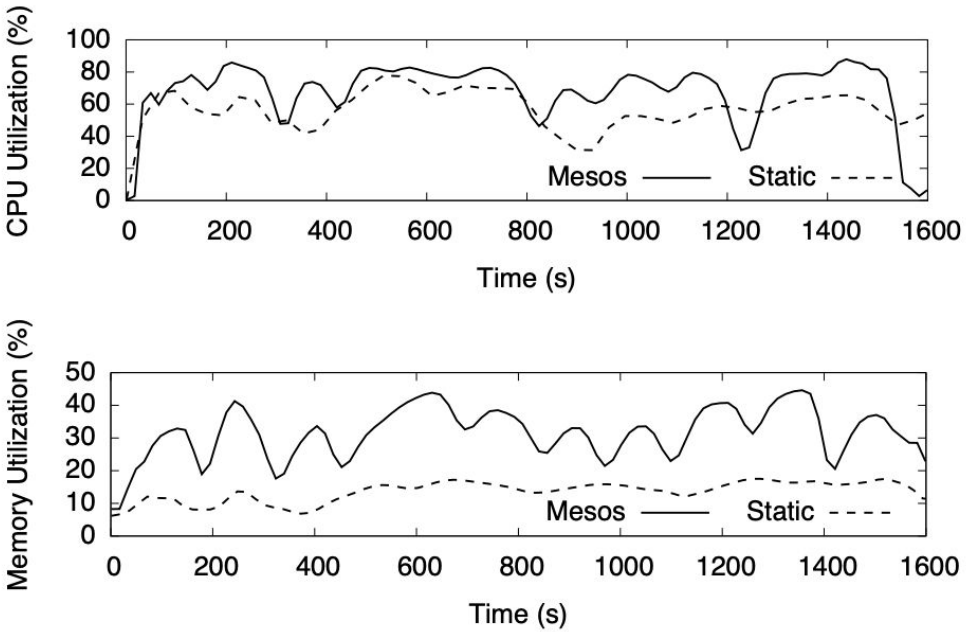
Mesos only performs *inter-framework* scheduling (e.g. fair sharing), which is easier than intra-framework scheduling

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

Mesos

- Does it work?
  - High utilization

esp in memory utilisation
if you were to use mesos, avg 30% utilisation which is higher already



Mesos

- Who’re using it?

MAY 03, 2015 • 4 MIN READ

by

Daniel Bryant

FOLLOW

Apple have announced that the company’s popular Siri iOS-based intelligent personal assistant application is powered on the backend by Apache Mesos, the open source cluster manager. The Mesosphere blog states that Apple have created a proprietary PaaS-like scheduler framework named J.A.R.V.I.S., which allows developers to deploy Siri services in a scalable and highly available manner.

Netflix Technology Blog Follow

Jul 29, 2016 · 6 min read

Netflix uses Apache Mesos to run a mix of batch, stream processing, and service style workloads. For over two years, we have seen an increased usage for a variety of use cases including real time anomaly detection,

Mesos

- But...

What Can We Learn from Twitter's Move to Kubernetes?

If you haven't heard the news yet, Mesos is out and Kubernetes is in. Check out what the implications of this switch are.

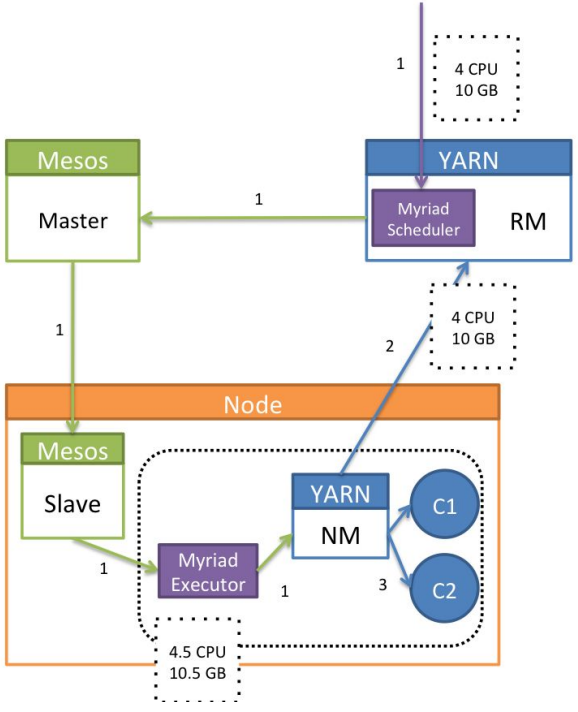
by Eva Tang · Aug. 28, 19 · Cloud Zone · Opinion

Mesos vs. YARN

- Per-framework scheduling
  - vs. per-job scheduling
- YARN more complex:
  - Monolith: master does scheduling for all frameworks
  - vs. 2-level resource management
- Mesos more suitable for multi-frameworks

Mesos vs. YARN

- Mesos said to influence YARN design
  - Because it came first
- YARN on Mesos



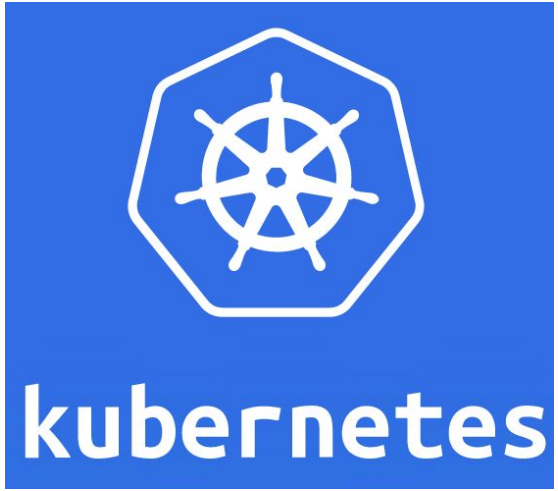
Summary

- Resource management layer is like your OS
- High utilization is its most important goal
- Mesos (2011):
  - 2-level scheduling: per-framework, and per-job
- YARN (2012):
  - Mainly designed for Hadoop
  - 1-level scheduling *only per-job*
- But:
  - Half-realize vision of datacenter OS

Summary

- OS has many more things beside a scheduler
  - Processes
  - File systems
  - IPC
  - Caches
  - etc.

Introduce



Question 1

- Consider YARN’s fair scheduler
  - Is it good if each job consists of only 1 long-running task?
  - For what type of jobs is fair scheduler most suitable?

Question 2

- Consider YARN’s capacity scheduler:
  - Give an example when it is worse than fair scheduler.
  - Give an example when it is better than fair scheduler.