

Resource Management in Big-Data Clusters

Mesos, YARN

Christoph Kessler

IDA, Linköping University

Multiple Big-Data Programming Models Co-Exist

- No single programming framework is optimal for all kinds of big-data applications



MPI

Pregel



S4 distributed stream computing platform



Apache Pig



Multiple Big-Data Programming Models Co-Exist

- Organizations would like to use the same cluster hardware for multiple programming frameworks, versions, and applications
- Sharing of data to be used across frameworks?
- **Jobs:** Both periodic production runs, development tests, and short ad-hoc queries
 - Most jobs are (relatively) short
 - Jobs consist of (many) **tasks** e.g. mappers and reducers
 - Most tasks are (relatively) short
- Need a "cluster-wide OS" for sharing a cluster among different big-data frameworks and jobs that know basically nothing about each other
 - Fairness, priorities, scalability, protection
= **Virtualization** of cluster resources

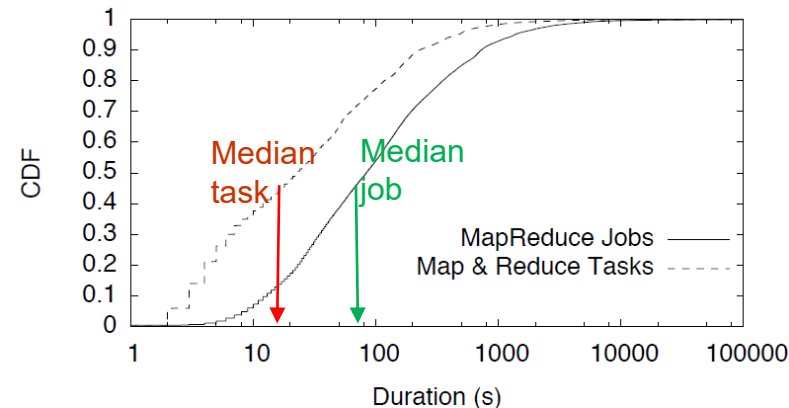


Figure 1: CDF of job and task durations in Facebook's Hadoop data warehouse (data from [38]). Image source: Mesos, 2011

Sharing a Cluster?

MR = Hadoop MapReduce

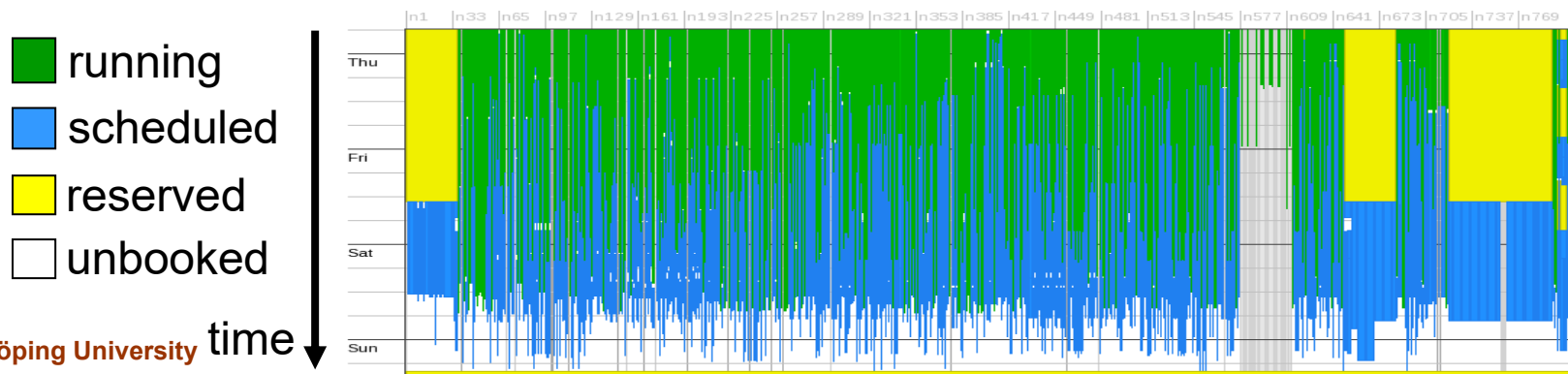
Dedicated cluster for a single user, single application (e.g. MR)?
→ Low utilization of expensive cluster resources

- **Idea 1:** Support *multiple users* and *multiple MR jobs* that *time-share the cluster*
 - Still one application type (here, MR) per node at a time
 - "Hadoop-on-demand"
 - Using a *batch scheduler* for cluster jobs

Background:

Batch Scheduling for Large Parallel Systems

- Batch queue systems (common in HPC)
 - e.g. Torque, Maui, Slurm
- Only 1 application (job) per node
- Parallel jobs
 - Job description:
 - ask for N nodes together for M minutes each to run program A
- Ahead-of-time reservation of system partitions and time
- Load balancing etc. over this fixed set of resources is up to the programming framework's runtime system e.g. of Hadoop, Spark

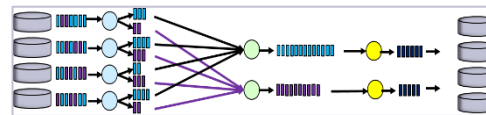


Sharing a Cluster?

MR = Hadoop MapReduce

Dedicated cluster for a single user, single application (e.g. MR)?
 → Low utilization of expensive cluster resources

- **Idea 1:** Support *multiple users and multiple MR jobs* that *time-share the cluster*
 - Still one application per node at a time
 - "Hadoop-on-demand"
 - ▶ Using a *batch scheduler* for cluster jobs



MapReduce computation structure

→ Utilization of cluster resources usually good during Map phases, but often not good during Reduce phases and I/O

- **Idea 2:** Time-share the *same* cluster nodes for *multiple applications* (*job types / programming frameworks*) to get better overall utilization (*multi-tenancy*)
 - Mesos, YARN

Mesos, YARN

- **Idea:**

Separate the resource management functionality from the programming model (and its runtime system)

- ☺ Can run *multiple applications* (e.g. multiple Hadoop MapReduce jobs) on the same cluster
- ☺ Can *mix* task executions from concurrent applications using *different frameworks* on same cluster, e.g. Hadoop MR (incl. different versions of it) and other frameworks (e.g. MPI)
 - Diversity of programming models
- ☺ Can *reuse the resource management subsystem* for different programming models
 - ▶ Cleaner software structure for the framework (e.g. Hadoop MapReduce) itself

Mesos

[Hindman *et al.* 2011]

- Mesos *master process* on one node manages all resources
- Mesos *slaves* (daemon processes) offer resources that are currently free
- *Frameworks* (e.g. Hadoop MR) submit requests for allocation and release of resources, to be approved/committed by Mesos master

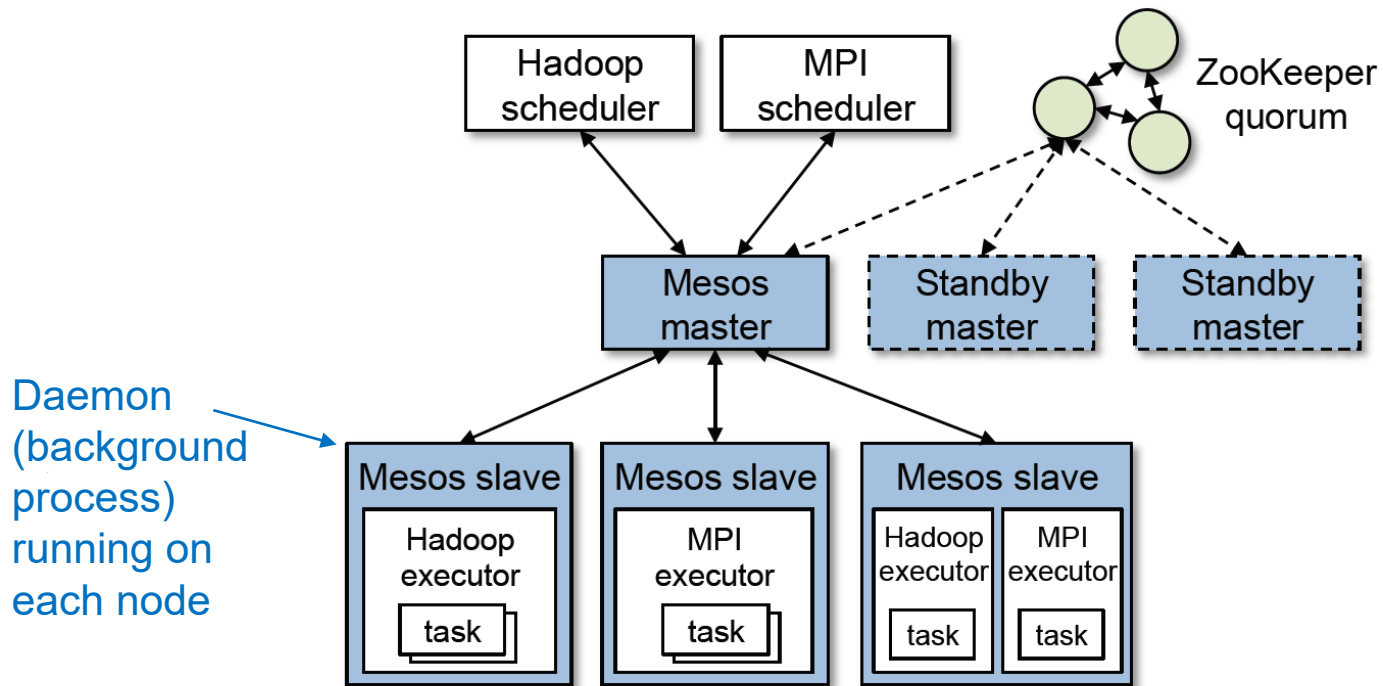
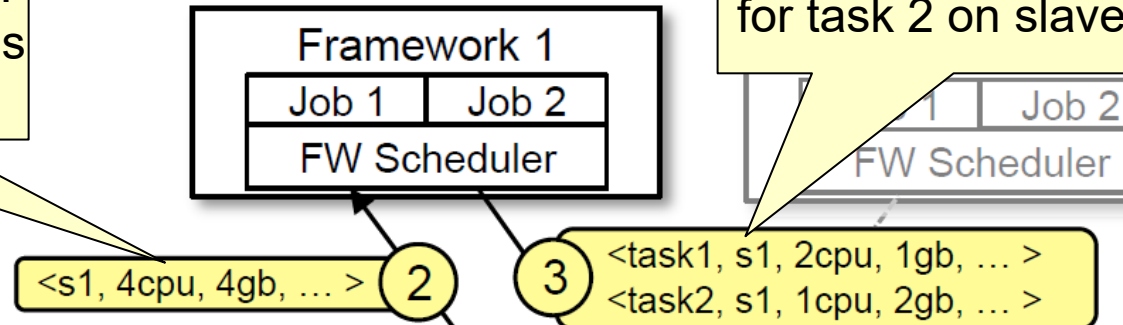


Figure 2: Mesos architecture diagram, showing two running frameworks (Hadoop and MPI).

Mesos Example

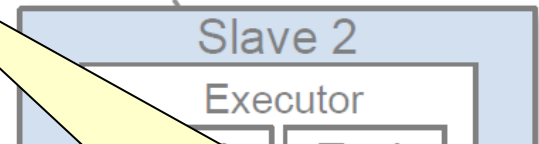
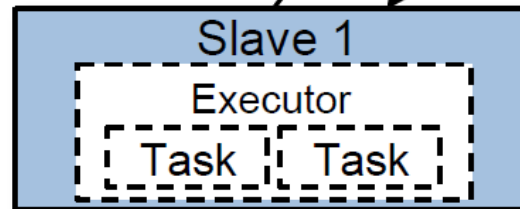
Master to Framework 1
(e.g. Hadoop MR master):
I have a node with 4 CPUs
and 4 GB RAM to offer

Framework 1 to Master:
Give 2 CPUs and 1 GB
to task 1 to run on slave1
and 1 CPU with 2 GB
for task 2 on slave 1



Slave (worker) 1 to Master:
I have 4 CPUs
and 4 GB RAM
currently free

Master launches tasks
on behalf of Framework1
on slave 1



YARN

YARN = Yet Another Resource Negotiator

Split up: resource management (global), scheduling/monitoring (application-specific)

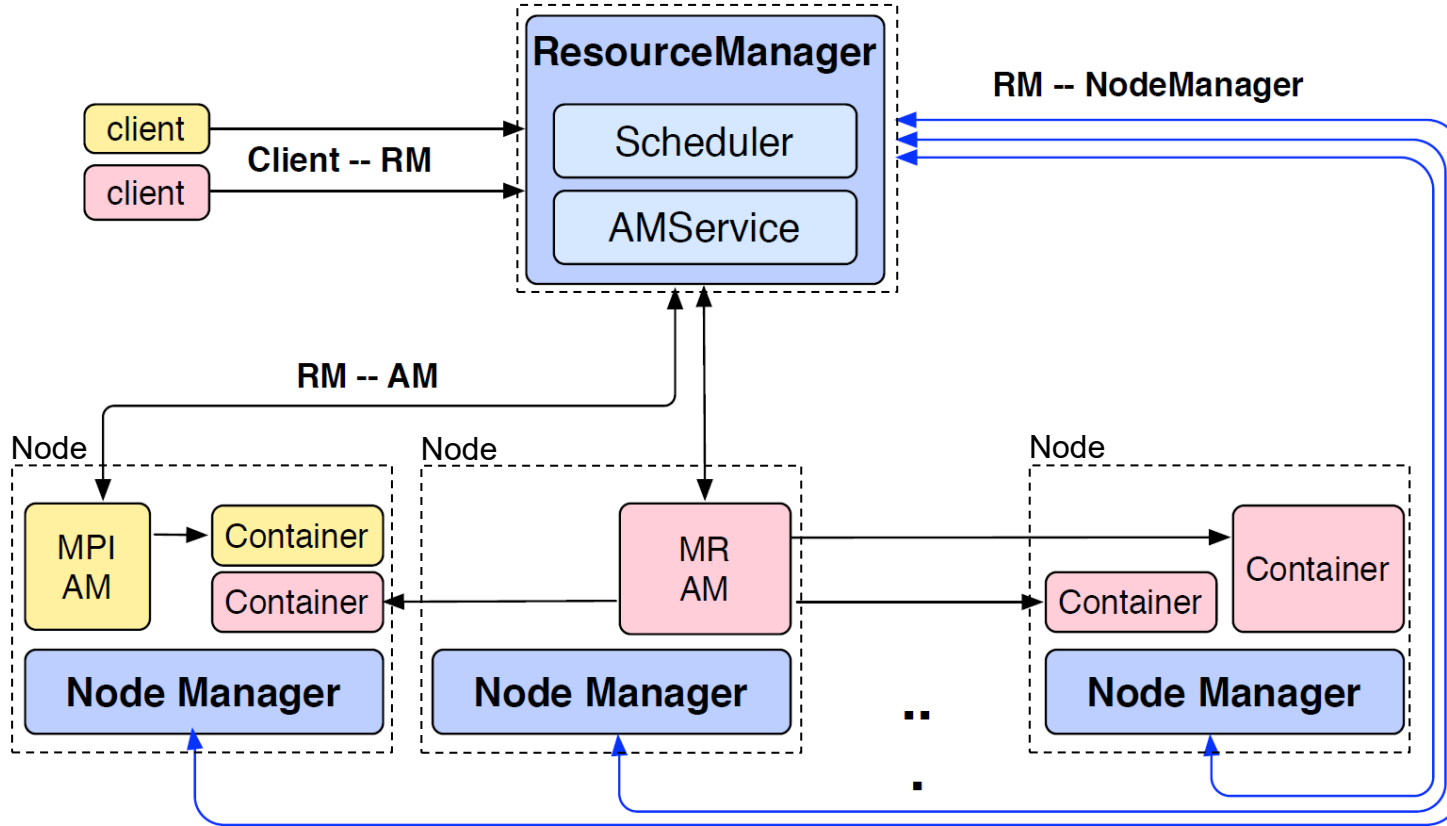
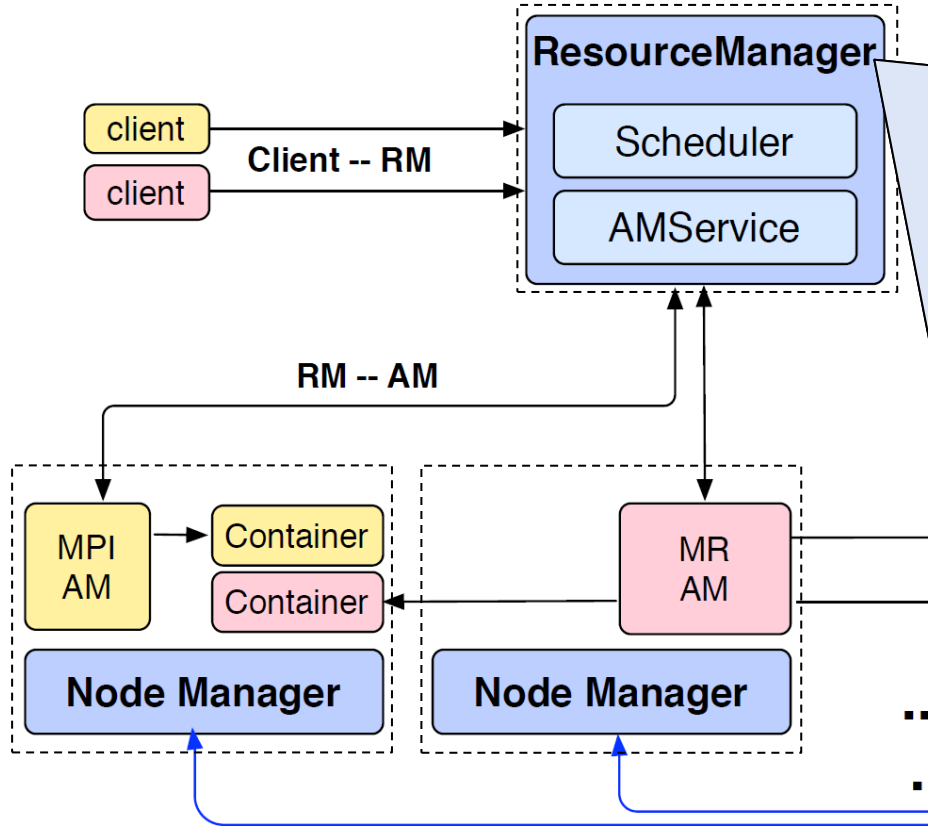


Image adapted from: Vavilapalli et al. 2013

- System components (blue)
 - RM – Resource Manager, AM – Application Master, NM – node mgr.
- Two applications running (yellow: using MPI, pink: using Hadoop MR)

YARN

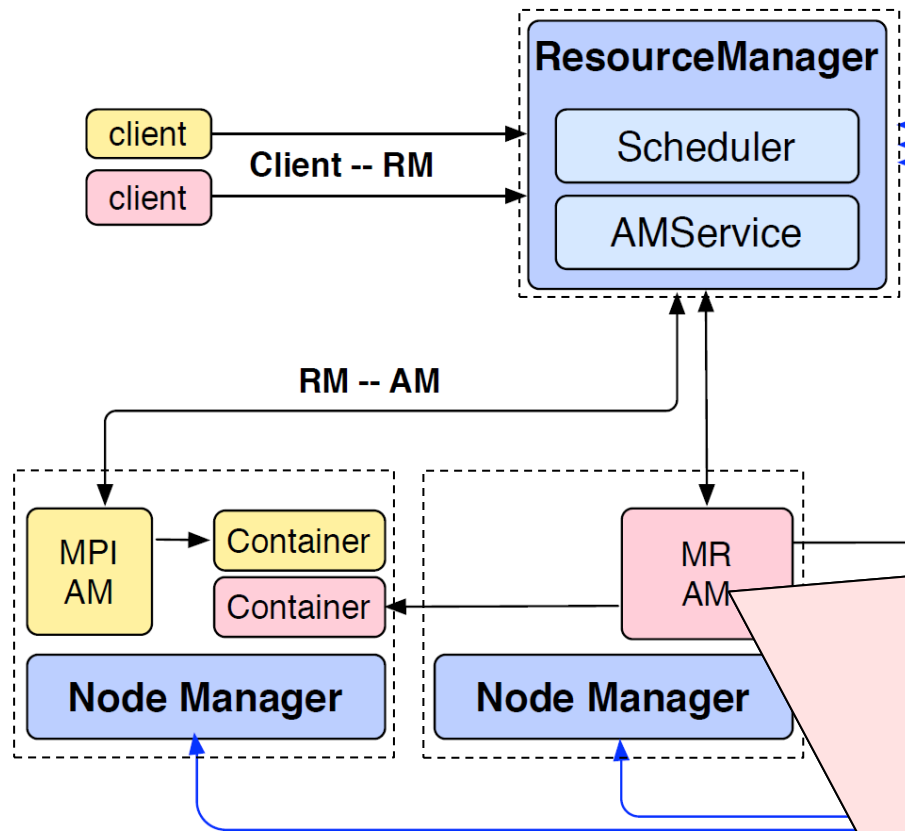


Resource Manager (RM):

- A daemon process running on a dedicated cluster node (e.g. node 0)
- Periodically polls the node managers to check for liveness
- **Jobs** are submitted to the RM with resource requirements (e.g. share of 20% CPU time, ...)
- RM performs admission control, checks security credentials, and allocates (leases) a bundle of free resources on particular nodes, e.g. <2GB RAM, 1 CPU>, to a **container** (= virtual node) for the job's execution on each of these nodes.
- RM informs the Node Managers of these nodes about the lease.
- Job **state** is moved from new to admitted to running.

- System components (blue)
 - RM – Resource Manager, AM – Application Master
- Two applications running (yellow: user application, pink: MapReduce application)

YARN

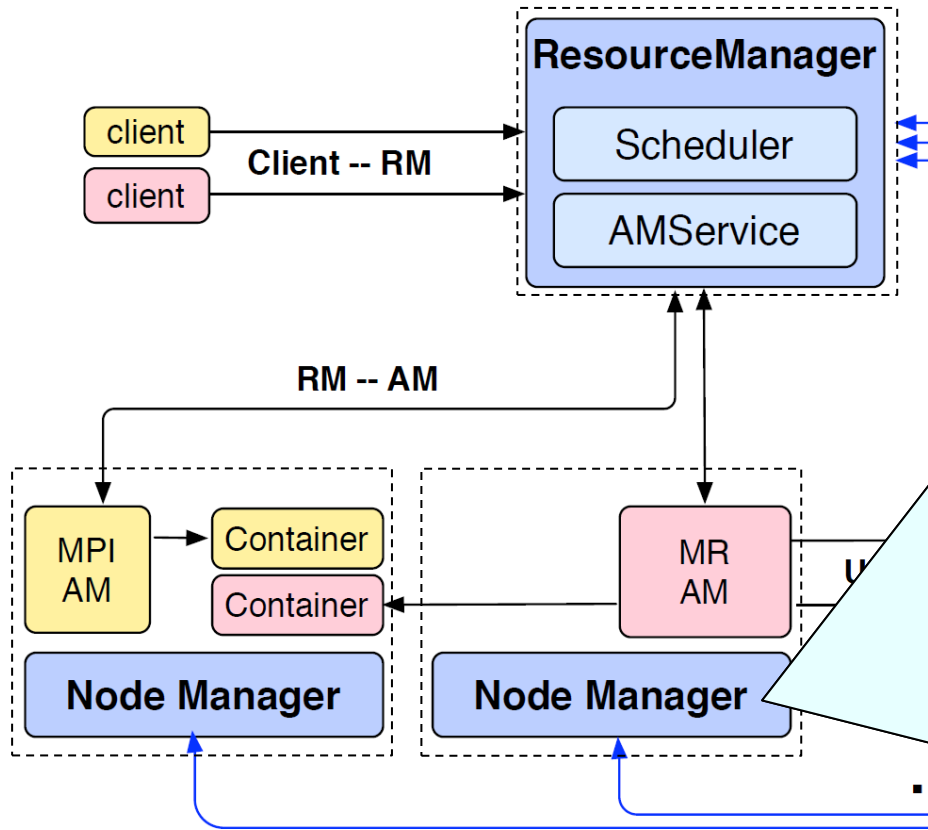


Application Master (AM):

- The "head" of a job (e.g., a MR job)
 - a daemon process (virtual master node)
 - application / framework - specific
- Manages all *intra-job lifecycle* aspects including dynamically increasing and decreasing resource allocation by issuing resource requests to the RM, *within* the lease obtained from RM
- Manages the flow of execution, e.g. dispatching Mapper and Reducer tasks in Hadoop MapReduce to the job's containers on the other nodes (= virtual worker nodes) assigned to it
- Handles faults and computation skew (straggling tasks) in a framework-specific way
- YARN itself makes no assumptions about the job's type of application (programming framework), this is entirely up to the AM
 - Only the protocols to communicate with RM and NM are fixed.

- System components (blue)
 - RM – Resource Manager, AM –
- Two applications running (yellow: us

YARN



Node Manager:

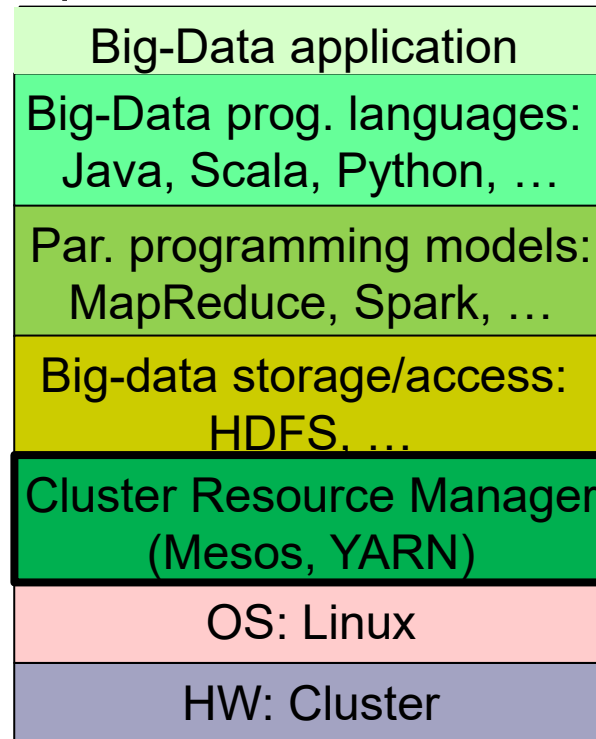
- The "worker" daemon on each node
- Authenticates container leases, manages container dependences, monitors their execution, provides services to containers
- Deallocates containers on request from AM or RM, when work is finished or when their resources must be preempted for a new higher-priority job
- Monitors health of the physical node
- Log aggregation for the containers
- Handles node-local persistence of data that escape the lifetime of a job.

- System components (blue)
 - RM – Resource Manager, AM – Application Master, NM – node mgr.
- Two applications running (yellow: using MPI, pink: using Hadoop MR)

Summary: Cluster Resource Management

- Cluster-wide "OS" for resource sharing across multiple concurrent jobs, programming frameworks, and their versions
- Virtualization of Cluster Resources
- 2-level resource allocation+management
 - Scheduling e.g. of mapper/reducer tasks to the resources assigned by the cluster manager is still done internally by each framework manager/master
- Mesos is offer-based, YARN is request-based

Big-Data System-Software Stack



References

- Benjamin Hindman *et al.*: Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. Proc. *NSDI'11*, USENIX, 2011.
- Apache Mesos: <http://mesos.apache.org/>
- V. Vavilapalli *et al.*: Apache Hadoop YARN: Yet Another Resource Negotiator. Proc. *SoCC'13*, ACM, 2013.
- Apache Hadoop YARN:
 - <https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.html>

Questions for Reflection

- Why is it reasonable that *Application Masters* can request and return resources dynamically from/to the *Resource Manager* (within the maximum lease initially granted to their job by the *RM*), instead of requesting their maximum lease on all nodes immediately and keeping it throughout the job's lifetime?
 - Contrast this mechanism to the resource allocation performed by batch queuing systems for clusters.
- Explain why the *Node Manager's* tasks are better performed in a daemon process controlled by the *RM* and not under the control of the framework-specific application.