Resource Management

Patrick Eugster
Purdue University and TU Darmstadt

Outline

- Context
- Resource management concept
- Example: YARN
- Example: Borg
- Conclusions

Outline

- **Context**
- Resource management concept
- Example: YARN
- Example: Borg
- Conclusions

Context

- Consider Hadoop executing map and reduce tasks for different jobs
  - The Hadoop runtime is deployed on a cluster of *n* nodes
- Which node to deploy some new task *t* on?

Context

- Consider Hadoop executing map and reduce tasks for different jobs
  - The Hadoop runtime is deployed on a cluster of *n* nodes
- Which node to deploy some new task *t* on?

Context

- Consider Hadoop executing map and reduce tasks for different jobs
  - The Hadoop runtime is deployed on a cluster of *n* nodes
- Which node to deploy some new task *t* on?

Context

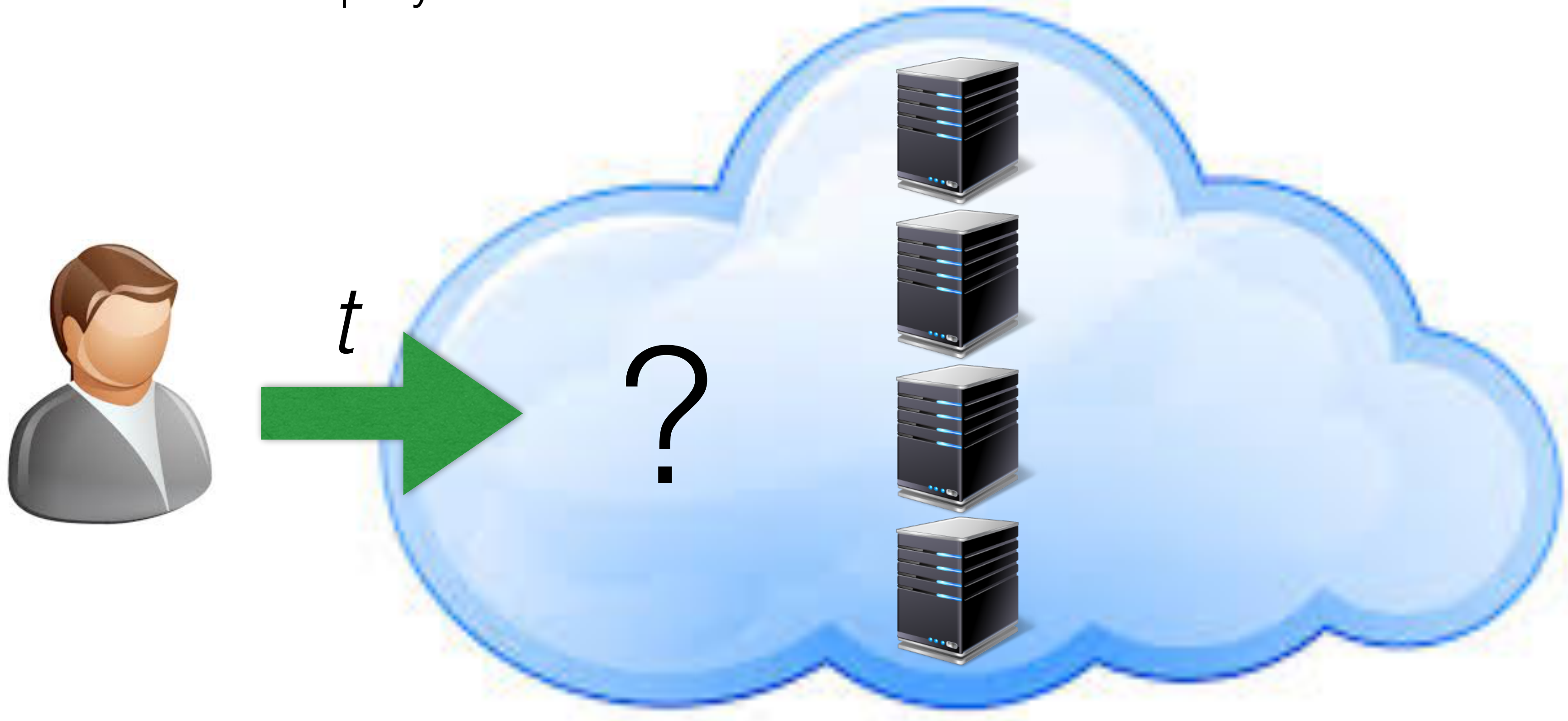
- Consider Hadoop executing map and reduce tasks for different jobs
  - The Hadoop runtime is deployed on a cluster of *n* nodes
- Which node to deploy some new task *t* on?

Context

- Consider Hadoop executing map and reduce tasks for different jobs
  - The Hadoop runtime is deployed on a cluster of *n* nodes
- Which node to deploy some new task *t* on?

Context

- Consider Hadoop executing map and reduce tasks for different jobs
    - The Hadoop runtime is deployed on a cluster of *n* nodes
- Which node to deploy some new task *t* on?



Simple Solutions?

- Why not simply round-robin?

Simple Solutions?

- Why not simply round-robin?
  - Tasks have different execution times
    - E.g., even mappers of same MR job can have very different execution times due to data skew
  - Tasks have different resource requirements
    - E.g., “CPU bound” vs “memory bound”
  - Nodes can have different HW configurations

Requirements

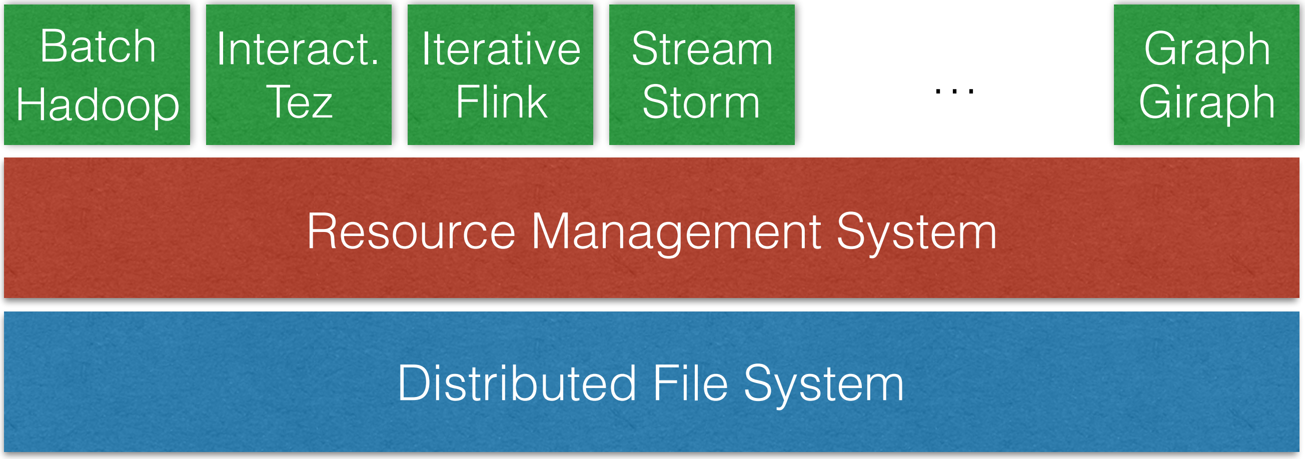
- Best usage of computational resources on cluster of heterogenous nodes for heterogenous jobs
- Yet another scheduling problem?
- Yes, but a complex one
    - Multi-dimensional: CPU, RAM, HD, (NW,) …
    - Multi-tenancy: different applications, in cloud also users
    - Fault tolerance (cluster nodes, app components, resource management components, …)
    - Security (…)
    - …

Outline

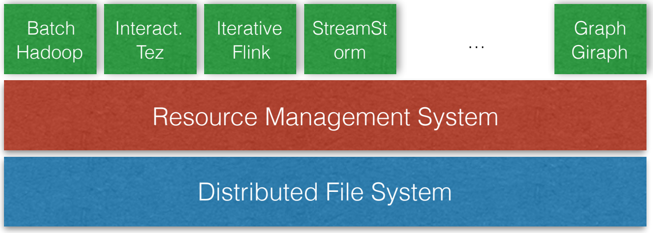
- Context
- **Resource management concept**
- Example: YARN
- Example: Borg
- Conclusions

Resource Management

- Typically implemented by a system deployed across nodes of a cluster
  - Layer below “frameworks” like Hadoop
  - On any node, the system keeps track of availabilities
  - Applications on top use information and estimations of own requirements to choose where to deploy something
    - RM systems (RMSs) differ in abstractions/interface provided and actual scheduling decisions

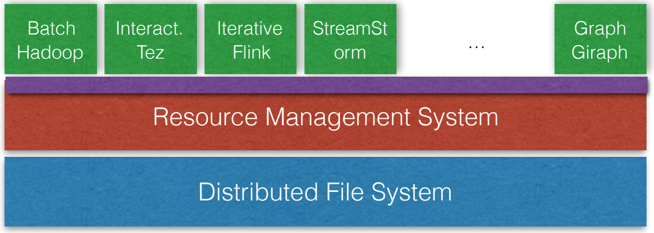


RMS Interfaces



- “Resource manager” (RM) interface
    - For applications to know **where** to deploy
- “Execution” interface
    - **How** to deploy application components
        - E.g. ssh
    - Execution is often **managed** by RMS
        - “Container” model (cf. app server)
        - Benefits: monitoring progress, debugging, fine-grained RM, fault tolerance, security

RMS Interfaces



- “Resource manager” (RM) interface
    - For applications to know **where** to deploy
- “Execution” interface
    - **How** to deploy application components
        - E.g. ssh
    - Execution is often **managed** by RMS
        - “Container” model (cf. app server)
        - Benefits: monitoring progress, debugging, fine-grained RM, fault tolerance, security

Scheduling

- Assignment of resources to applications/application instances
- Many different algorithms for scheduling based on different objectives
  - FIFO
  - Some notion of fairness
  - Max overall usage
- Internal or external (application) or mix
  - E.g. priorities

RM Interface and Architectures

- Interaction model
  - “Offer”: RM tells applications about available resources
  - “Pull”: application asks RM for resources, subsequently deploys
    - Sync vs async
  - “Push”: application directly submits request with resource description and task description (container)
- Coordination model
  - Via (logically centralized) resource manager “server” (“internal scheduling”)
  - Peer-based by interaction with individual nodes (“external scheduling”)

Fault Tolerance

- Node crashes?
    - Few RMSs provide support for this
        - Replication: costly and restrictive (e.g. determinism)
        - Checkpointing: application-specific
- Application component failures?
    - Due to OS etc. (not application code): see above
    - Due to application code: debugging information
- RMS component failures?
    - State can be restored from nodes
    - Unavailable to application

Security

- What if the RMS is compromised?
    - E.g. can claim that resources are out
        - “Denial of service” (availability)
        - Or inversely “make up” resources
    - If manages container deployment
        - Modify containers, tamper with execution and data (code/data integrity)
        - Inspect them or leak data (code/data privacy)
    - …?
- Very little to no support so far

RMS Examples

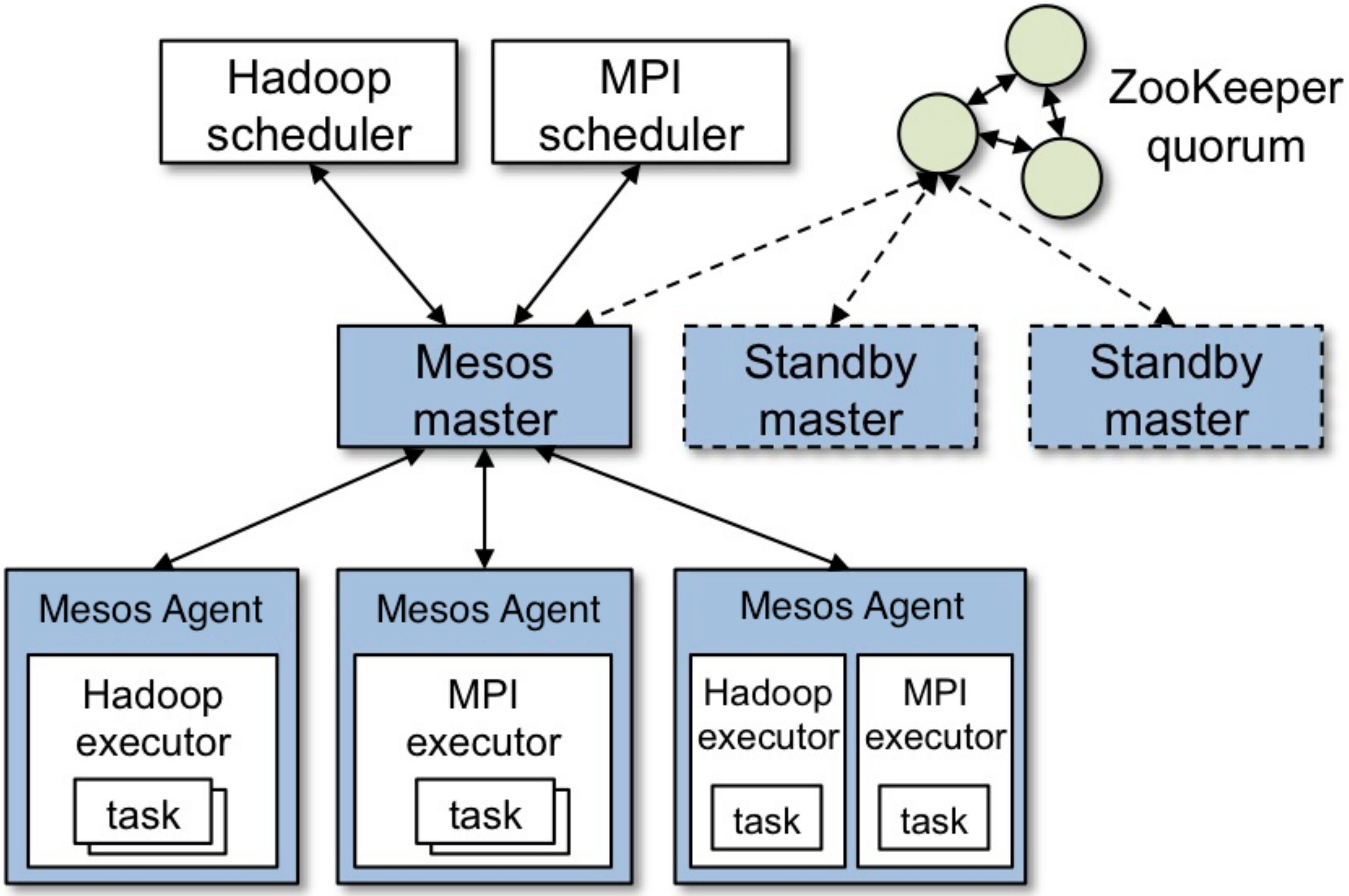
- Grid
    - Condor/HTCondor
    - Sun Grid Engine
- Cloud
    - Apache Mesos (Berkeley)
    - Apache YARN — yet another resource manager (Yahoo!)
    - Omega, Kubernetes, Borg (Google)
    - Apache Helix (LinkedIn)
    - Fuxi (Alibaba)
    - Tupperware (Facebook)
- Other
    - Apache Myriad: mediate between YARN and Mesos
    - Llama: mediate between YARN and Impala (Cloudera)

Apache Mesos

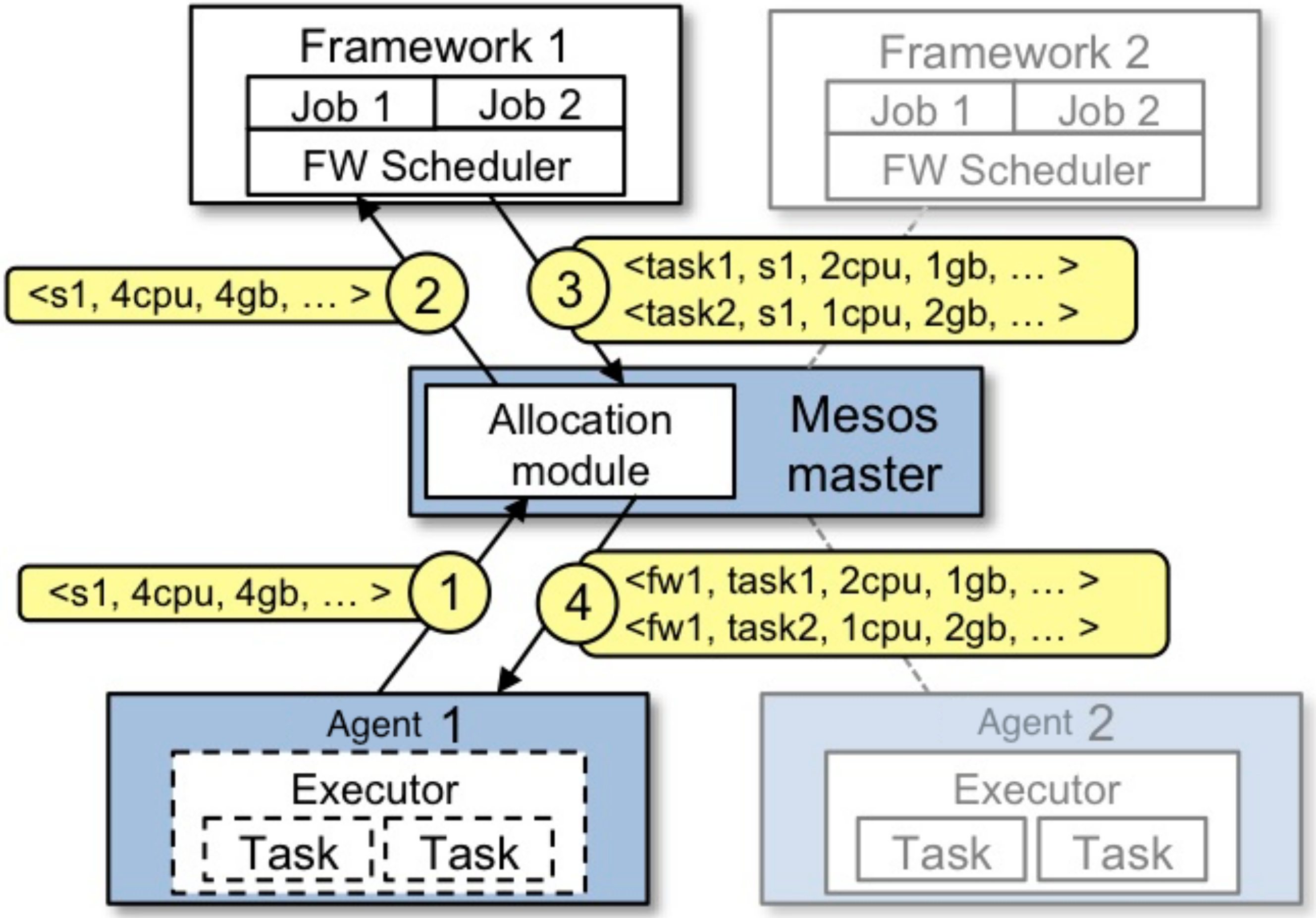
- Berkeley, 2007
  - Designed for global cluster management
- Two-level scheduling
  1. “Application”-level, pluggable
  2. Arbitration between different 1.
- Offers resources to applications, which accept/decline
- No security, FT for master (hot standby)

Overview

http://mesos.apache.org/documentation/latest/architecture/



Resource Offering

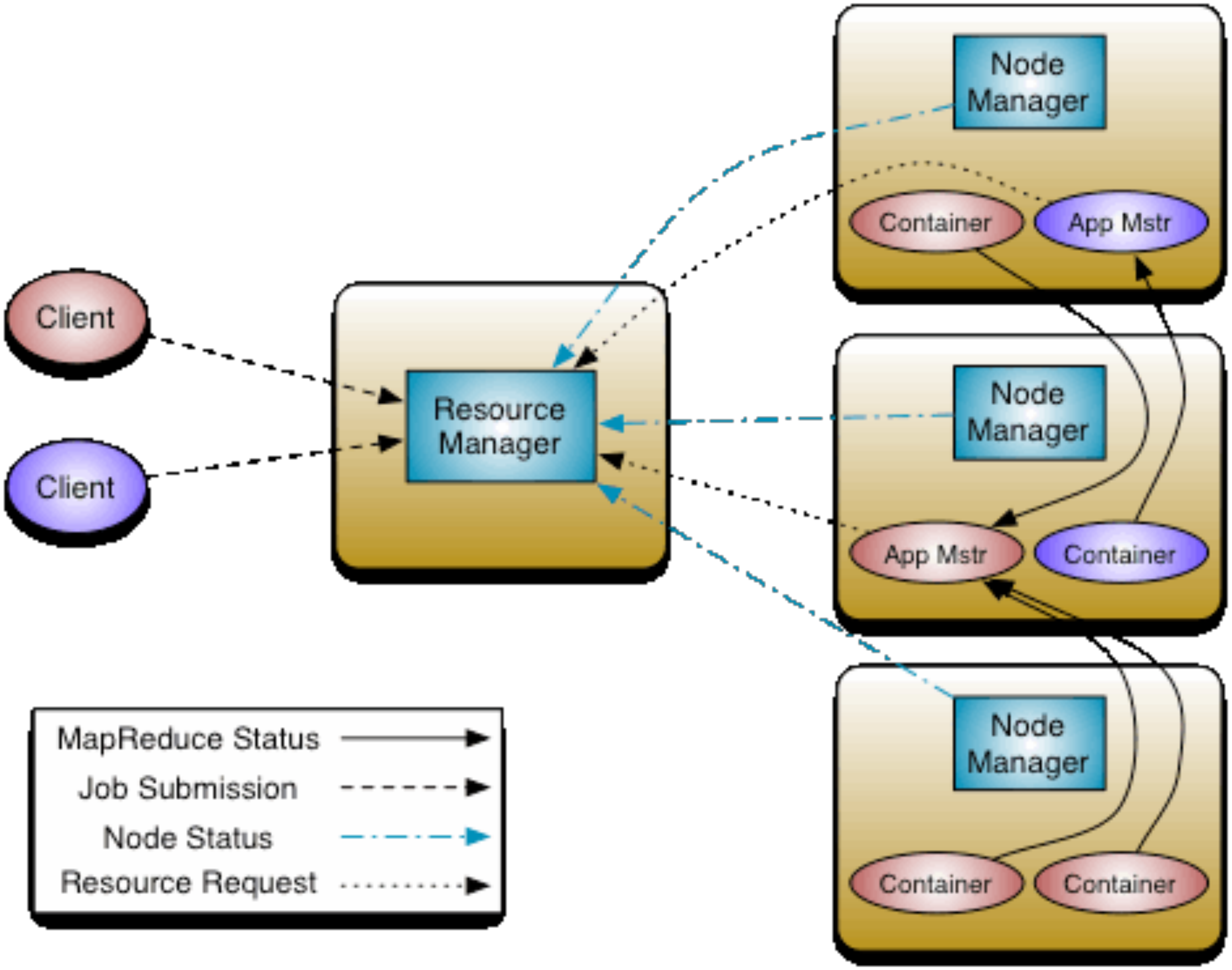


Apache YARN

- Originally designed for Hadoop by Yahoo!
    - Now used for others, e.g., Storm, Spark
- Single level scheduling, request-based
- Distinguishes application masters from generic app components
- No security or FT

Overview

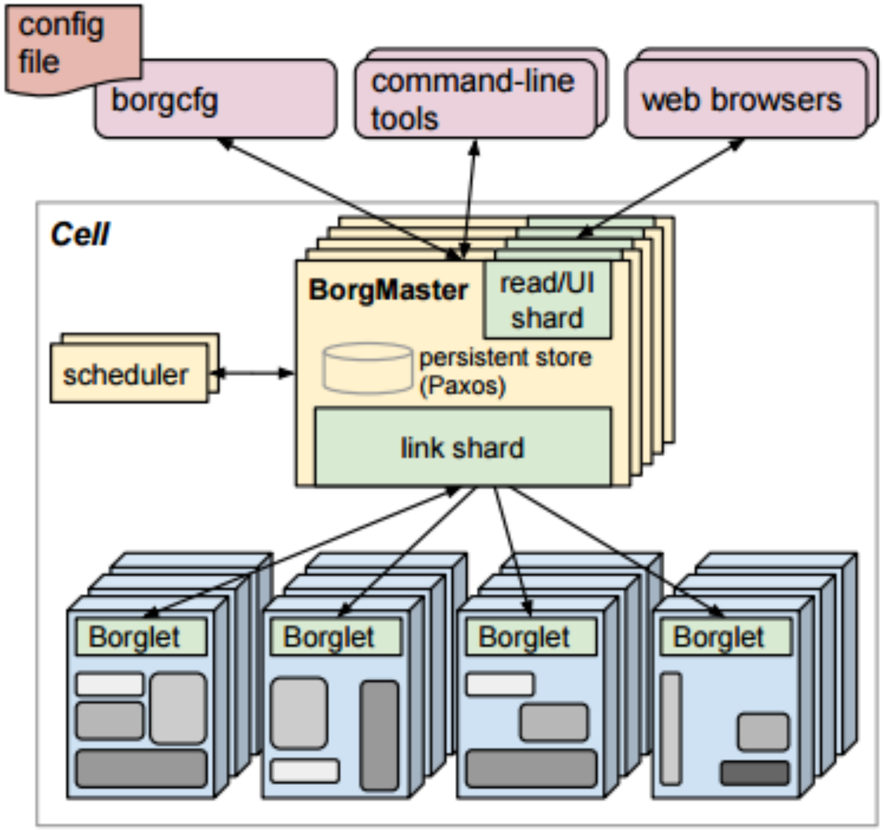
http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html



Borg

http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.pdf

- Programs submit jobs which have multiple tasks
- Task are fine-grained (no VMs)
- Borg master replicated using Paxos
- Security via `chroot` and `borgssh`

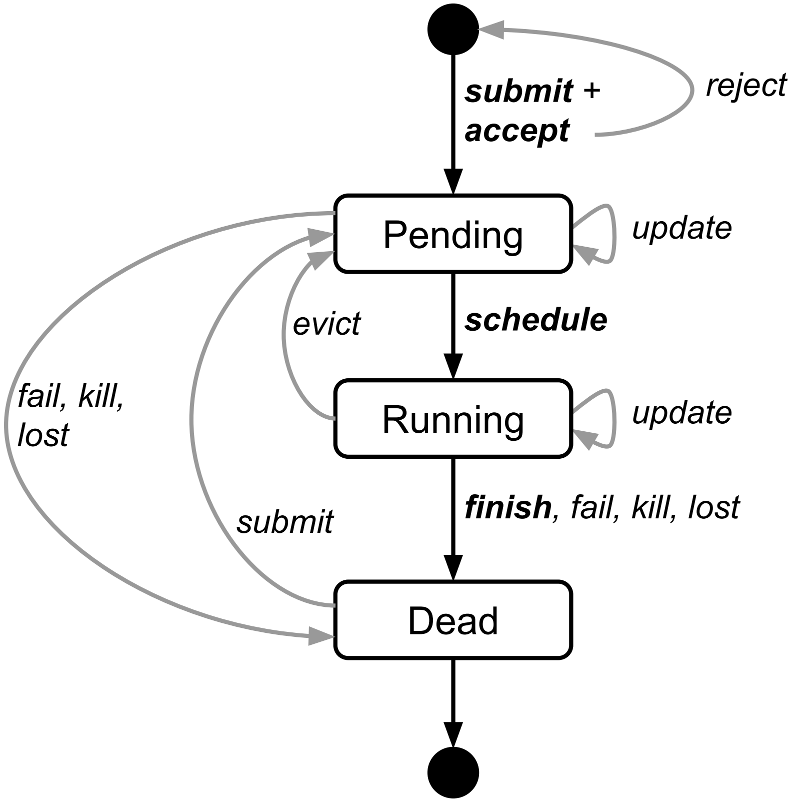


More than just an RMS

- Sophisticated management of job and task *execution*

- Chubby used to persist state of tasks, interaction info

- Apps interact with tasks via RPC



Conclusions

- RMS represents core abstraction layer in clouds
- Still much work needed, e.g.,
  - FT & security support
  - Network provisioning in addition to local resources
    - E.g. via traffic engineering in software defined networking (SDN)