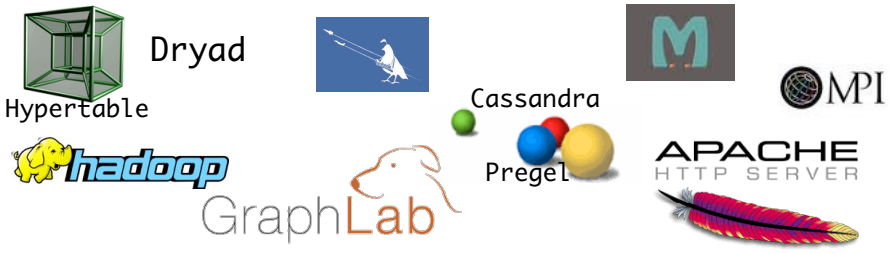
Project 3: Resource Scheduling with Apache YARN

15-719

Greg Ganger
Garth Gibson
Majd Sakr

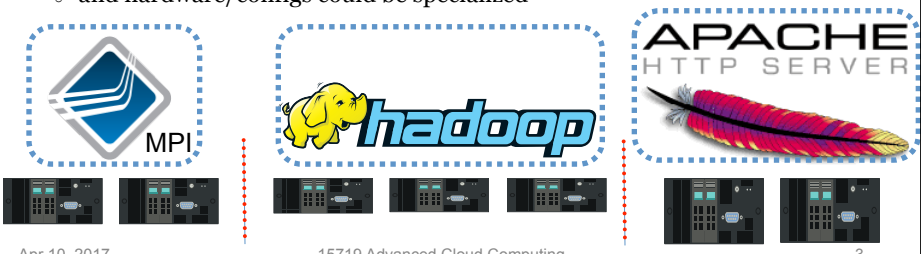
Context: many execution frameworks

- There are many cluster resource consumers
  - Big Data frameworks, elastic services, VMs, …
  - Number going up, not down: GraphLab, Spark, …



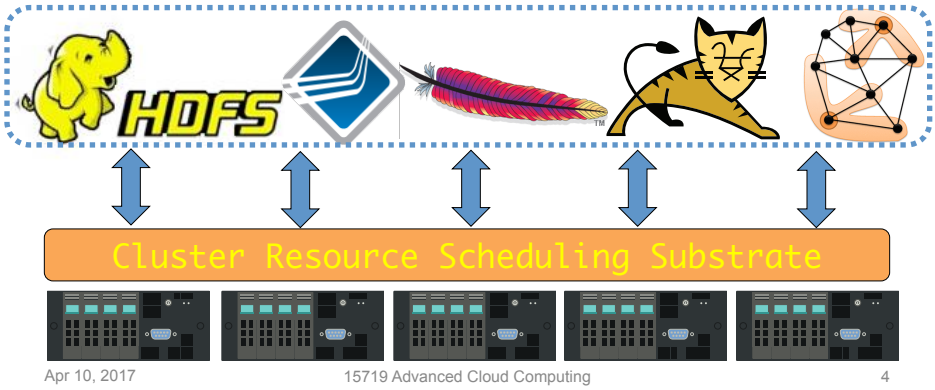
Traditional: separate clusters

- There are many cluster resource consumers
  - Big Data frameworks, elastic services, VMs, …
  - Number going up, not down: GraphLab, Spark, …
- Historically, each would get its own cluster
  - and use its own cluster scheduler
  - and hardware/configs could be specialized



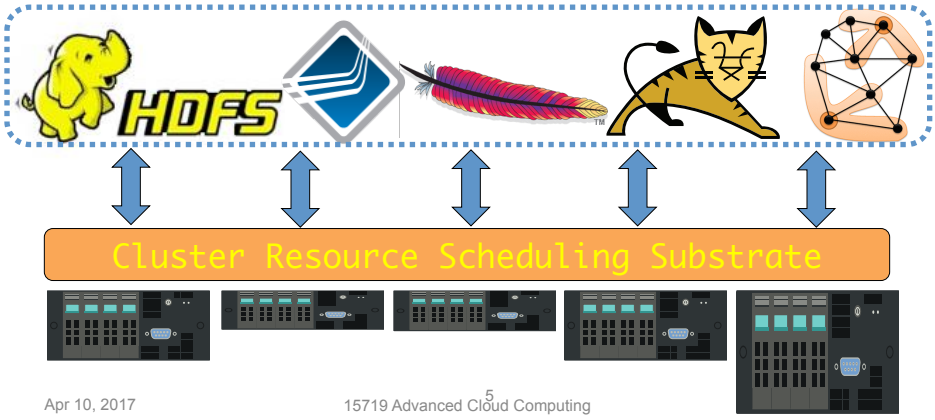
Preferred: dynamic sharing of cluster

- Heterogeneous mix of activity types
  - Some long-lived HA services; others short-lived batch jobs w/ lots of tasks
- Each grabbing/releasing resources dynamically
  - Why? all the standard cloud efficiency story-lines



And, INTRA-cluster heterogeneity

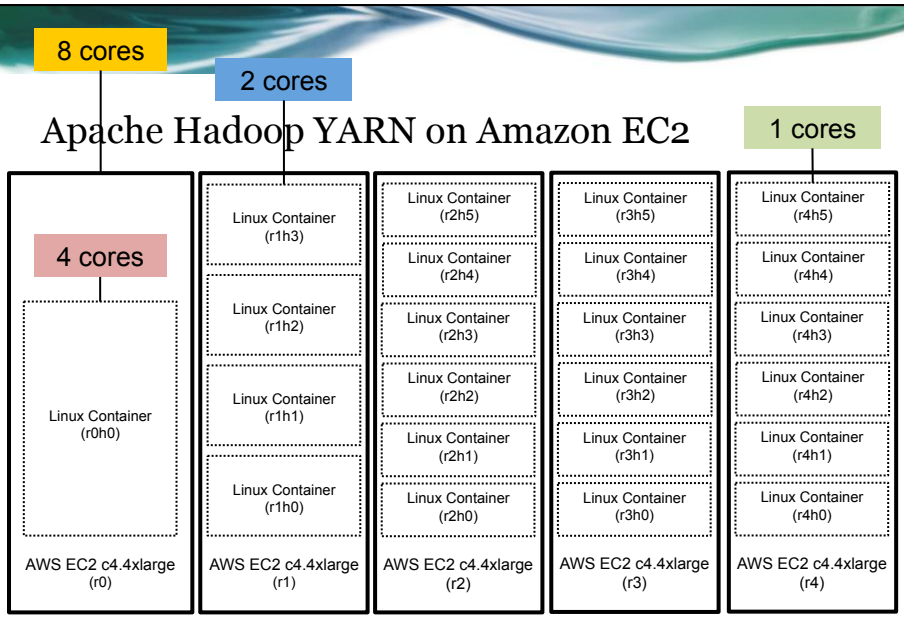
- Have a mix of platform types, purposefully
  - Providing a mix of capabilities and features
  - Then, match work to platform during scheduling



Project 3 Overview

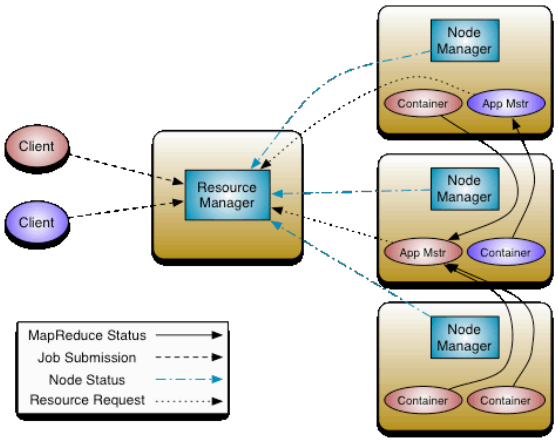
- Deploy a container-based heterogenous YARN cluster on cloud
- Implement a scheduling policy server paired with YARN
- Schedule a set of "MPI" and "GPU" jobs on your YARN cluster
- Evaluate and compare different scheduling policies (FIFO, SJF…)
- Consider and try to schedule jobs to their preferred resources

Apache Hadoop YARN on Amazon EC2



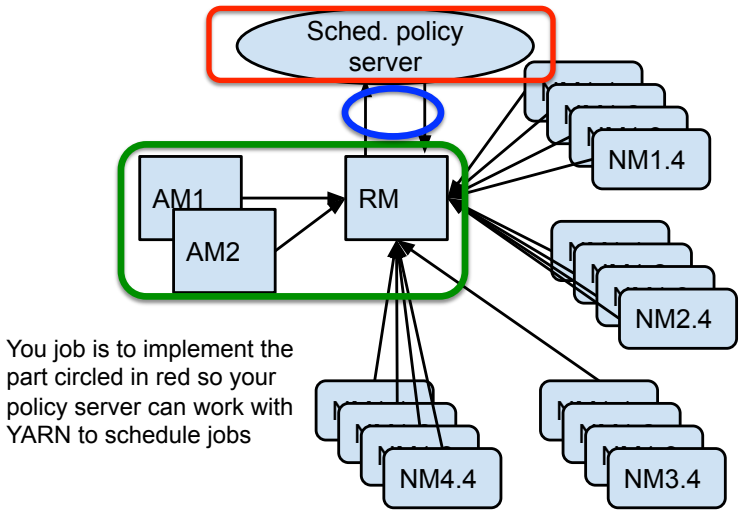
5 “racks”, 1x 4-core “machine”, 4x 2-core “machines”, 18x 1-core “machines”
(Each “Linux Container” is being treated as a “machine”)

Hadoop YARN Architecture



Do not confuse YARN container with Linux containers created to serve as a YARN node

Modified YARN Architecture: Logical Topology



You job is to implement the part circled in red so your policy server can work with YARN to schedule jobs

RPC Interface

Interface your policy server should implement

```
service TetrischedService {
    void AddJob(1:JobID jobId, 2:job_t jobType, 3:i32 k, 4:i32 priority,
    5:double duration, 6:double slowDuration),
    void FreeResources(1:set<i32> machines),
}
service YARNTetrischedService {
    void AllocResources(1:JobID jobId, 2:set<i32> machines),
}
```

Callback your policy server can call to finish schedule

Example Scheduling Policies

- First Come First Served (FCFS)
  - with or without heterogeneity awareness
- Shortest Job First (SJF)
  - using job runtime estimate, schedule the shortest job first
- Earliest Deadline First
  - if the deadline is set, pick the most urgent job from the queue
- Deferred allocation decisions (as a modification to other algos)
  - concept: don't immediately schedule a pending job when nodes are freed
    - in case a new job that needs them is submitted soon
  - deciding how long to wait is key

Evaluation metric for project 3:
Mean job completion time

Project Overview

- Part 1 – implement the 3 scheduling policies we specify
  - 2 FIFO policies, 1 SJF policy
  - Get familiar with the framework and the code
  - 10 days
- Part 2 – design and implement your own policy
  - Must be able to consider job's preferred resources as soft/hard constraints
  - 12 days

Project Logistics/Notes

- Individual project, no groups
- We provide a tool that can setup YARN on EC2
- We provide a sample policy server to get you started
- Write your code in C++
- Not okay to install arbitrary open-source C++ libraries
  - Ask us if you think you need something; we'll ok or (likely) not
- Do not push code to public source code control system (github.com)
- Do not leave your EC2 instances unattended

References

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

https://github.com/apache/hadoop-common/tree/branch-2.2.0/hadoop-yarn-project/hadoop-yarn