COSC 6397
Big Data Analytics

Hadoop 2 and YARN

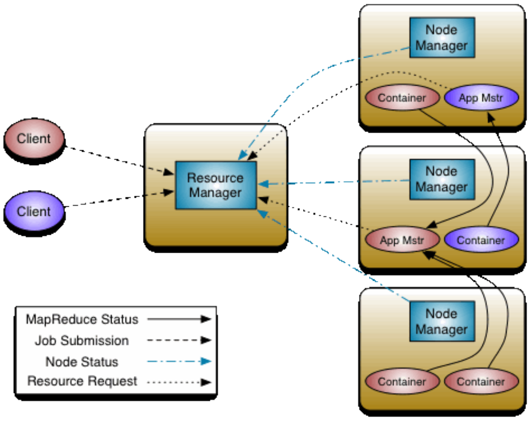
Edgar Gabriel
Spring 2015

Hadoop 2

- Major differences between MapReduce (v1) before and after hadoop-0.23 => MapReduce 2.0 (MRv2) or YARN.
- Split up the two major functionalities of JobTracker:
  - resource management and
  - job scheduling/monitoring
- Use a global ResourceManager (*RM*) and per-application ApplicationMaster (*AM*)
- An application can be DAG of jobs.
- ResourceManager is the ultimate authority to arbitrate resources among all the applications in the system
- Per-application ApplicationMaster is a framework specific library to negotiate resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

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

YARN architecture



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

YARN Platform Benefits

- Deployment
  - YARN provides a seamless vehicle to deploy your software to an enterprise Hadoop cluster
- Fault Tolerance
  - YARN ‘handles’ (detects, notifies, and provides default actions) for HW, OS, JVM failure tolerance
  - YARN provides plugins for the app to define failure behavior
- Scheduling (incorporating Data Locality)
  - YARN utilizes HDFS to schedule app processing where the data lives
  - YARN ensures that your apps finish in the SLA expected by your customers

Slide based on a lecture by Arun Murthy and Bob Page:
“Developing YARN Native applications”

Application Categories

Type	Definition	Examples
Framework / Engine	Provides platform capabilities to enable data services and applications	Twill, Reef, Tez, MapReduce, Spark
Service	An application that runs continuously	Storm, HBase, Memcached, etc
YARN App	A *temporal job* submitted to YARN	MapReduce job, spark job etc.

Slide based on a lecture by Arun Murthy and Bob Page:
"Developing YARN Native applications"

Yarn Concepts: Container

- Basic unit of allocation
  - Based on capability: e.g, memory, CPU etc
  - Replaces the fixed map/reduce slots from Hadoop 1
- Allows for fine grained resources allocation
  - Capability, Host, Rack, Priority etc.
- `ContainerLaunchContext`: object describing resources necessary to launch the Container
  - Local Resources: Resources needed to execute container application
  - Environment variables: e.g. CLASSPATH
  - Command to execute

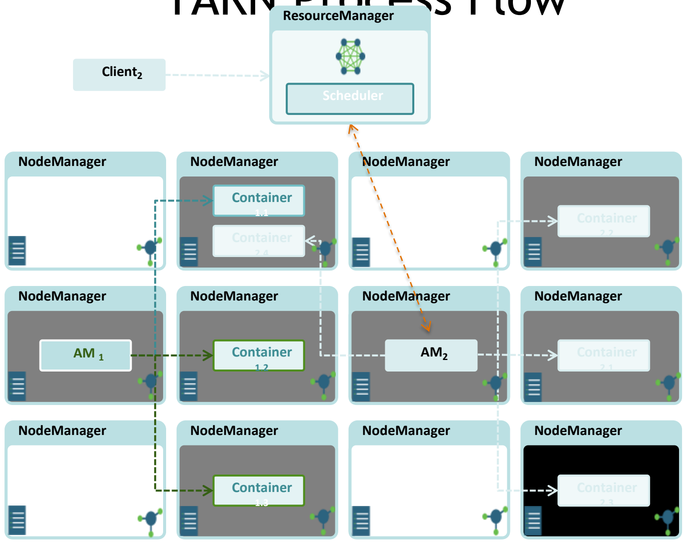
Slide based on a lecture by Arun Murthy and Bob Page:
"Developing YARN Native applications"

YARN Terminology

- Resource Manager
  - Central agent
  - Allocates & manages cluster resources
- Node Manager
  - Manages, monitors and enforces node resource allocation
  - Manages life cycle of containers
- User application
  - Client: submits application
  - Application Master (AM): manages application lifecycle and task scheduling
  - Container: executes application logic

Slide based on a lecture by Arun Murthy and Bob Page: "Developing YARN Native applications"

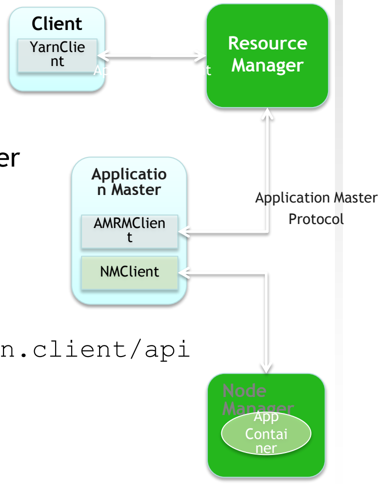
YARN Process Flow



Slide based on a lecture by Arun Murthy and Bob Page: "Developing YARN Native applications"

YARN APIs

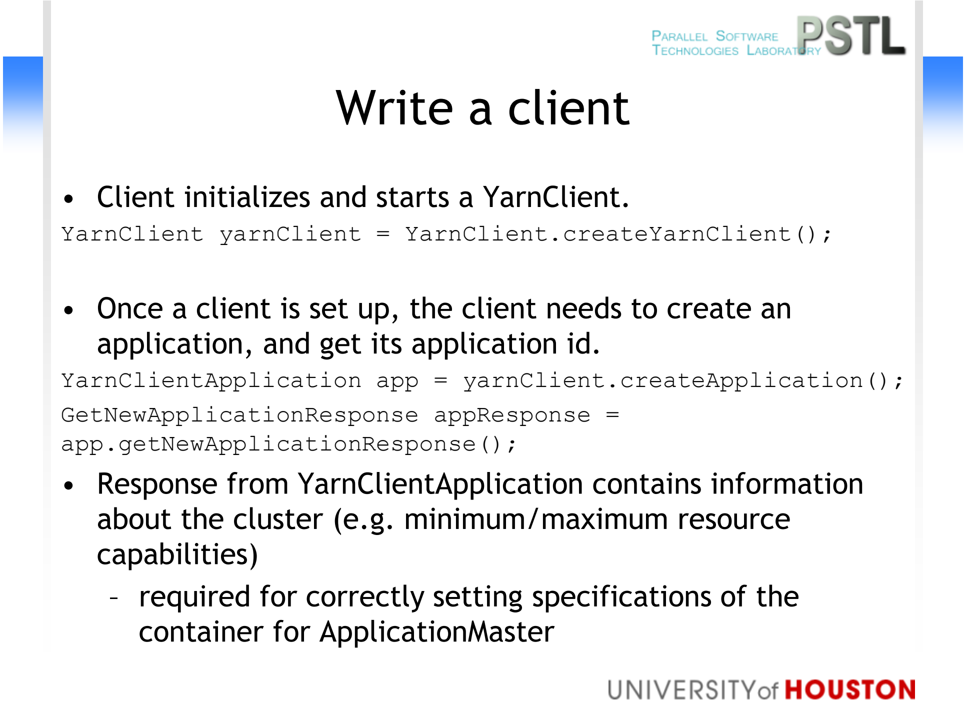
- Three protocols
  - Client to ResourceManager
    - Application submission
  - ApplicationMaster to ResourceManager
    - Container allocation
  - ApplicationMaster to NodeManager
    - Container launch
- Use client libraries
  - Package `org.apache.hadoop.yarn.client/api`



YARN Implementation Outline

1. Write a client
   1. Submit the application
   2. Monitor application
2. Write an ApplicationMaster
3. Get containers and run application

Slide based on a lecture by Arun Murthy and Bob Page:
“Developing YARN Native applications”



Write a client

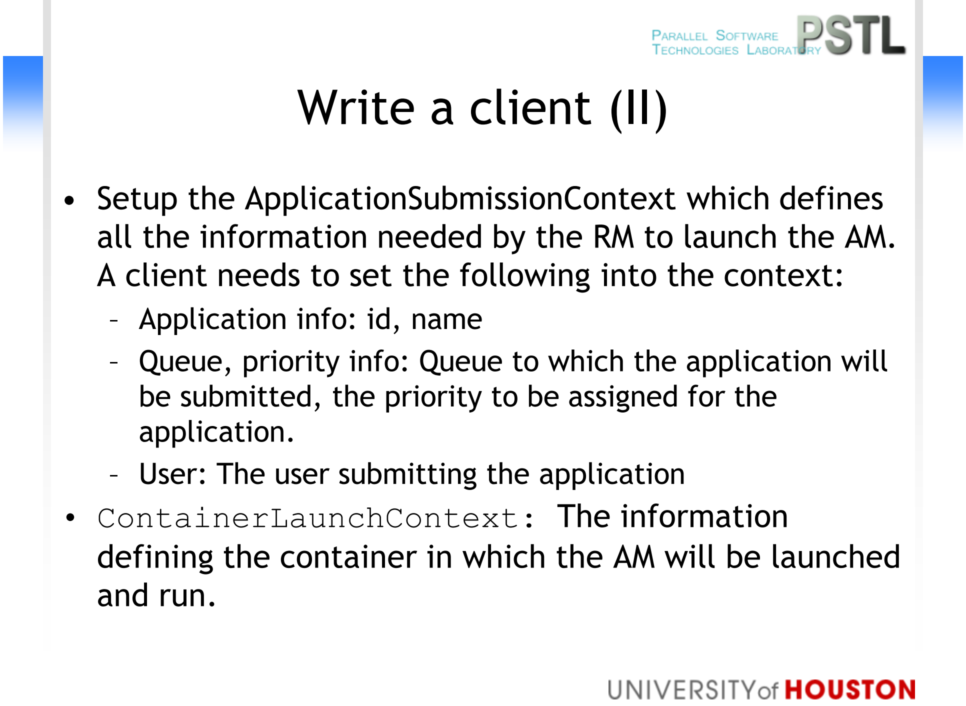
- Client initializes and starts a YarnClient.

```
YarnClient yarnClient = YarnClient.createYarnClient();
```

- Once a client is set up, the client needs to create an application, and get its application id.

```
YarnClientApplication app = yarnClient.createApplication();
GetNewApplicationResponse appResponse =
app.getNewApplicationResponse();
```

- Response from YarnClientApplication contains information about the cluster (e.g. minimum/maximum resource capabilities)
  - required for correctly setting specifications of the container for ApplicationMaster



Write a client (II)

- Setup the ApplicationSubmissionContext which defines all the information needed by the RM to launch the AM. A client needs to set the following into the context:
  - Application info: id, name
  - Queue, priority info: Queue to which the application will be submitted, the priority to be assigned for the application.
  - User: The user submitting the application
- `ContainerLaunchContext:` The information defining the container in which the AM will be launched and run.

```
// set the application submission context
appContext = app.getApplicationSubmissionContext();
appContext.setKeepContainersAcrossApplicationAttempts(keepContainers);
appContext.setApplicationName(appName);

// Set up resource type requirements
Resource capability = Resource.newInstance(amMemory, amVCores);
appContext.setResource(capability);

// Set up the container launch context for the application master
ContainerLaunchContext amContainer =
ContainerLaunchContext.newInstance( localResources, env, commands,
null, null, null);
appContext.setAMContainerSpec(amContainer);

Priority pri = Priority.newInstance(amPriority);
appContext.setPriority(pri);
appContext.setQueue(amQueue);

// Submit the application to the applications manager
yarnClient.submitApplication(appContext);
```

Client monitoring

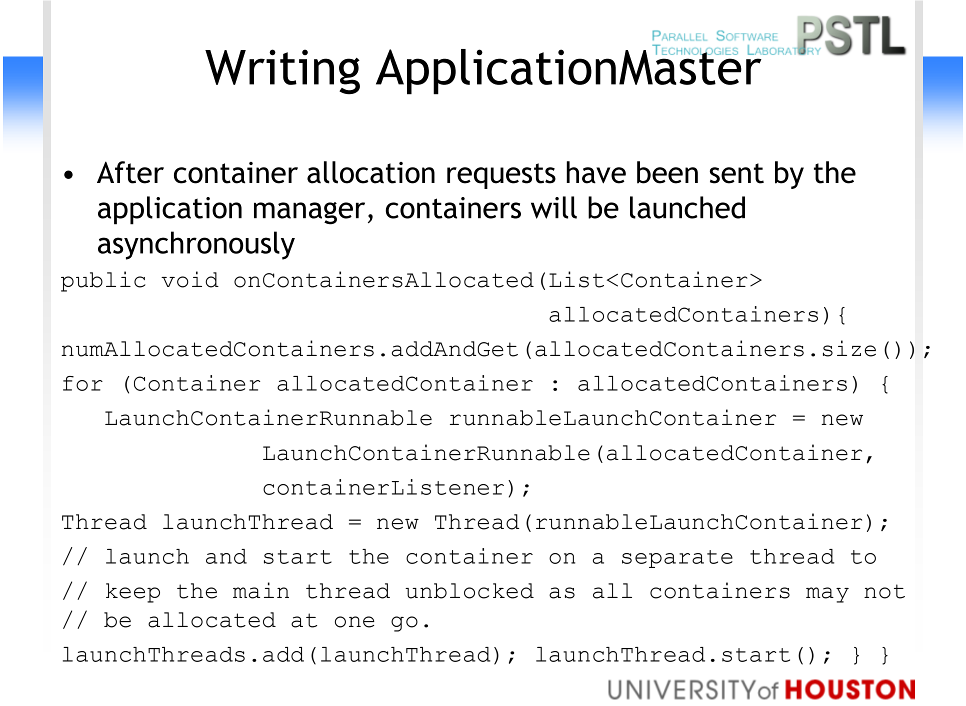
- Client communicates with RM and requests a report of the application via the `getApplicationReport()` method of YarnClient:
  - *General application information*: Application id, queue to which the application was submitted, user who submitted the application and the start time for the application.
  - *ApplicationMaster details*: the host on which the AM is running, the rpc port (if any) on which it is listening for requests from clients and a token that the client needs to communicate with the AM.
  - *Application tracking information*: tracking url
  - *Application status*: The state of the application as seen by the ResourceManager

Writing ApplicationMaster

- The AM is the actual owner of the job
- Launched by the ResourceManager
- Client provides information about resources required to oversee and complete the job
- AM is launched within a container that may (likely will) be sharing a physical host with other containers
- Several parameters are made available to AM via the environment, e.g.
  - ContainerId for the AM container,
  - application submission time
  - details about the NodeManager host running the AM
- All interactions with the RM require an ApplicationAttemptId

Writing ApplicationMaster

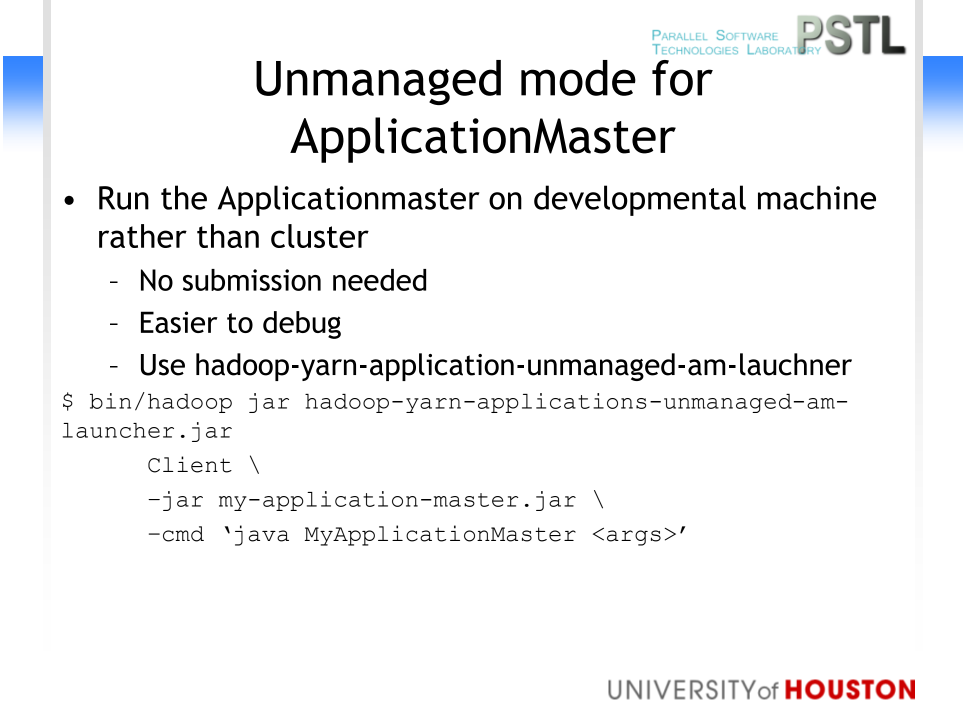
- AM starts two clients:
  - Communication with ResourceManager,
  - Handling NodeManager
- AM emits regular heartbeat signal to RM
- AM calculates number of containers needed based task requirements and requests the container from RM
- AM has to establish the following information for job submission(using `setupContainerAskForRM()`):
  - *Resource capability:* value is defined in MB and has to less than the max capability of the cluster and an exact multiple of the min capability.
  - *Priority:* different priorities can be requested for different sets (e.g. for mapper and reducer)



Writing ApplicationMaster

- After container allocation requests have been sent by the application manager, containers will be launched asynchronously

```
public void onContainersAllocated(List<Container>
                                   allocatedContainers){
numAllocatedContainers.addAndGet(allocatedContainers.size());
for (Container allocatedContainer : allocatedContainers) {
   LaunchContainerRunnable runnableLaunchContainer = new
              LaunchContainerRunnable(allocatedContainer,
              containerListener);
Thread launchThread = new Thread(runnableLaunchContainer);
// launch and start the container on a separate thread to
// keep the main thread unblocked as all containers may not
// be allocated at one go.
launchThreads.add(launchThread); launchThread.start(); } }
```



Unmanaged mode for ApplicationMaster

- Run the Applicationmaster on developmental machine rather than cluster
  - No submission needed
  - Easier to debug
  - Use hadoop-yarn-application-unmanaged-am-lauchner

```
$ bin/hadoop jar hadoop-yarn-applications-unmanaged-am-
launcher.jar
      Client \
      -jar my-application-master.jar \
      -cmd ‘java MyApplicationMaster <args>’
```

Non MapReduce Applications

- Distributed Shell
- Impala
- Apache Giraph
- Spark
- …

MapReduce on Yarn

- Hadoop includes a MapReduce Application master to manage MR jobs
- Each Mapreduce job is an instance of a YARN application
- Shuffle is an auxiliary service that runs in the NodeManager JVM

Fault Tolerance of MR jobs

- Task/Container failure
  - MRAppMaster will reexecute tasks that execute with an exception or stop responding (4 times by default)
  - Applications with too many failed tasks considered failed
- Application Master failure
  - If AM stops sending heartbeats, ResourceManager will attempt to start a new master (2 times by default)
- NodeManager
  - If NM stops sending heartbeats to RM, node will be removed from list of available nodes
  - Tasks on that node be treated as failed ab AM
- Resource Manager: single point of failure if not configured in High Availability Mode