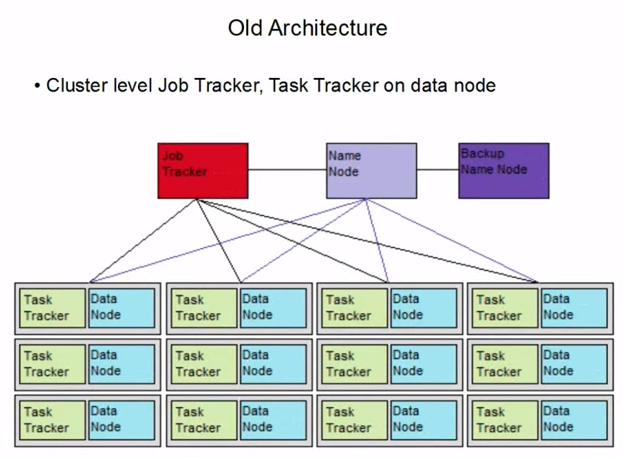
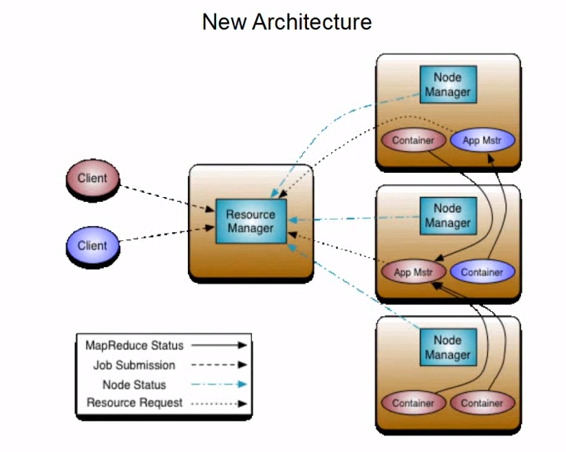
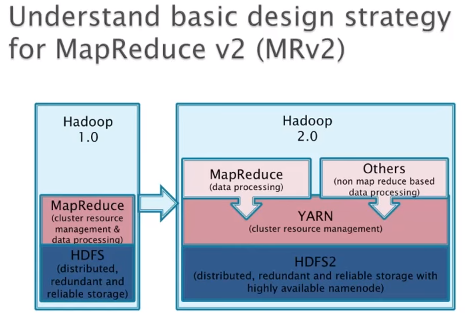
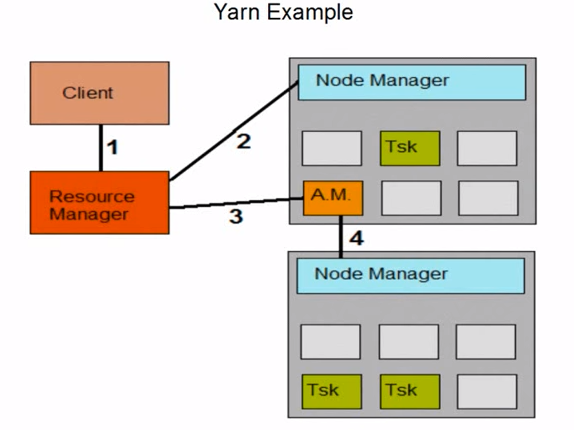
**YARN - Yet Another Resource Negotiator (YARN)**









|  |  |  |
| --- | --- | --- |
| **S.No** | **Topic** | **Details** |
|  | **Web Reference** |  |
|  | About Yarn | <https://www.youtube.com/watch?v=jLDyLnr9YuE&index=15&list=PLf0swTFhTI8pOZ4VBSGerKUmF9USWL6vd> |
|  | About Yarn | <http://www.devx.com/opensource/intro-to-apache-mapreduce-2-yarn.html> |
|  | Property configuration  *<Details in last page>* | <http://www.cloudera.com/documentation/cdh/5-1-x/CDH5-Installation-Guide/cdh5ig_mapreduce_to_yarn_migrate.html> |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | **General Information** |  |
| 1 | Yarn Version | # No specific version for yarn. Yarn is available after Hadoop 2.x  # so use Hadoop version  MRv1 – is classic  MRv2 – is local |
| 2 | Purpose of YARN | 1. Yarn is a platform for Resource management, application scheduling and execution. 2. A generic job management tool which can share resources between “map reduce” based distributed processing tools (Hive, Pig, Sqoop, Ooze, Flume) and “non-map reduce” based distributed processing tools (Hbase, Impala, Spark) 3. It is capable to run more than one application simultaneously. We can run 60k to 70k applications per day, still the capacity is growing |
| 2.1 | Hadoop 1.x Architecture   1. Resource management is part of MapReduce | 1. MapReduce handles Data processing, Resource Management + Job Processing (Job Tracker, Task tracker ) 2. Name node uses Job Tracker and Data Node uses Task tracker   *Refer Quick ref -MapReduce for more details* |
| 2.2 | Hadoop 2.x Architecture   1. Yarn is introduced to handle not only MapReduce, it will also handle other applications | 1. New component has been created called YARN to handle Resource management + JOB processing 2. So MapReduce handles only data processing and it is sitting on top of YARN, like PIG, HIVE, Streaming, etc |
| 3 | **Parameter files** | # Parameter files used in Yarn   1. Mapred-site.xml 2. Yarn-site.xml |
| 3.1 | mapred-site.xml | # mapred-site.xml should have framework name as YARN  <property>  <name> Mapreduce.framework.name</name>  <value>yarn</value> </property>  # Job history server Web UI IP address and port no. For all the completed jobs  <name> Mapreduce.jobhistory.webapp.address</name>  <value>ipaddress:port</value> </property>  # Parameter related to application master  <name> yarn.app.mapreduce.am.\*</name>  <value>? </value> </property>  # JVM Heap size for child task of map container  <name> Mapreduce.map.java.opts</name>  <value>? </value> </property>  # JVM Heap size for child task of reduce container  <name> Mapreduce.reduce.java.opts</name>  <value>? </value> </property>  # Memory Size of container for map  <name> Mapreduce.map.memory.mb</name>  <value>? </value> </property>  # Memory Size of container for reduce  <name> Mapreduce.reduce.memory.mb</name>  <value>? </value> </property>  What is virual core or vcore?  # Number of virtual cores required to run each map tasks  <name> Mapreduce.map.cpu.vcore</name>  <value>1</value> </property>  # Number of virtual cores required to run each reduce tasks  <name> Mapreduce.reduce.cpu.vcore</name>  <value>1</value> </property> |
| 3.2 | Yarn-site.xml | # yarn-site.xml. some of the important parameters  # Resource manager IP and port  <name>yarn.resourcemanager.address</name>  <value>node name where resource manager configured:8088</value>  # Resource manager Web UI IP and port  <name>yarn.resourcemanager.webapp.address</name>  <value>node name where resource manager configured:8088</value>  #Classes which determines schedulers – Fair or capacity. Note: Capacity is not supported by cloudera  <name>yarn.resourcemanager.scheduler.class</name>  <value>?</value>  #Minimum total memory for containers on each nodes  <name>yarn.scheduler.minimum-allocation-mb</name>  <value>1024</value>  #Maximum total memory for containers on each nodes  <name>yarn.scheduler.maximum-allocation-mb</name>  <value>4096</value>  #Minimum number of virtual cores on each of the nodes  <name>yarn.scheduler.minimum-allocation-vcores</name>  <value>1</value>  #Maximum number of virtual cores on each of the nodes  <name>yarn.scheduler.maximum-allocation-vcores</name>  <value>4</value> |
| 4 | **Apache Web Interfaces** | 1. Resource Manager WUI 2. <http://node> name where resource manager configured: 8088 3. Application Master WUI 4. Job History Server |
| 5 | **Daemon Process** |  |
| 5.0 | Daemon process in MR1 | # Before Hadoop 2  Web Interface: Job Tracker WUI  **Daemon Process in MRV1:**   1. Job Tracker -> Master (Code is going to data. It does two tasks. # HA is not available in JT) 2. Submit jobs 3. Keep track of resources: Jobs will be tracked in master (Centralized tracking). 4. Task Tracker -> Slave Node (Tasktracker and Data node are located in same nodes. TT used to submit jobs in slave nodes) |
| 5.1 | Daemon process in MR2 and YARN | # After Hadoop 2  Web Interface: Resource Manager WUI  **Daemon Process in MRV2:**   1. Job Tracker -> Master (Code is going to data? It does ~~two~~ tasks) 2. Submit jobs 3. ~~Keep track of resources~~ 4. Task Tracker -> Slave Node (Tasktracker and Data node are located in same nodes. TT used to submit jobs in slave nodes) 5. History server is related to MapReduce in YARN. Not sure this is part of daemon   **Daemon Process in YARN:**   1. **Resource Manager (RM Master):** RM will de-centralize the job tracking. As soon as RM submit jobs, it will create AM in all the slave nodes and track jobs in slave. Ex: if you submit 100 jobs, instead of tracking all jobs from master, RM will allocate AM to track them in slave node and get report from AM. The advantage here is, RM has less work load compare to Job tracker, so the failure chances are very less for RM compare to JT. So MRV2 will be stable compare to MRV1. Also we can enable high availability on RM. It ensures the safety of MRV2. Where as in Job Tracker, HA is not possible. 2. **Per job Application Master (AM) :** Keep track of resources for the currently running jobs. Once job completes, it will be moved to history server (we can set history server node in configuration) 3. **Node Manager (NM Slave)**: 4. **Container**: |
| 6 | 5 Major Components of YARN | Components  a. Resource Manager (RM)  b. Node Manager (NM)  c. Application Master (AM)  d. Container  e. Client |
| 6.1 | Resource Manager (RM)   1. Cluster Level RM 2. Long Life | This is responsible for allocating and managing resources available within the cluster. There is only one RM per cluster  RM has two components:   1. Scheduler - Responsible for allocating resources to the various applications 2. Application manager - responsible for handling application submissions made by clients   RM has three interfaces (Two public and one private/internal):   1. A public interface for clients to submit jobs (Application-client protocol) 2. A public interface for AMs to request for resoures (Application-Master Protocol) 3. An internal interface for NM interations |
| 6.2 | Node Manager(NM)   1. One per Data Server 2. Monitors resources on node | Responsible for local resource management   1. Local container management - authentication to resource monitoring 2. Report to the RM using heartbeats 3. Container Launch Context (CLC) record is used to specify container 'metadata' such as dependencies, data file paths and environment variables 4. Monitoring of local physical resources such as CPU, memory and disk health and NM provides services such as log aggregation to the application |
| 6.3 | Application Master (AM)   1. One per application 2. Short life 3. Manages task/scheduling | Encapsulation of all application-specific logic and libraries. This is per application based. Ex: MapReduce, Pig, Hive are application sitting on top of YARN. AM manages those applications   1. Applications Master negotiates a container with the Scheduler 2. The container is used to bootstrap the AM for this particular application |
| 6.4 | Container | This is an abstract representation of a resource that is given to a particular application. It manages entire application life cycle. Ex: Specification of resources in each container is 2 CPS and 4GB RAM |
| 6.5 | Client | This is an entity in the cluster that can submit applications to the RM and specify the type of AM that needs to be spawned to execute the application   1. YARN Client is responsible for submitting an appropriate CLC for AM 2. AM itself runs in a container whose resources are negotiated by the RM. Registering the AM is also the responsibility of the clients 3. The client is free to provide other services to its consumers...meaning Submit a new CLC to AM |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | **Scheduling Jobs** |  |
| 1 | FIFO Scheduler  *<<Not recommended>>*  *<<Low priority>>* | You have a cluster running with a FIFO scheduler enabled.You submit a large job A to the cluster, which you expect to run for one hour.Then, you submit job B to the cluster, which you expect to run a couple of minutes only.You submit both jobs with the same priority. Describe how FIFO Scheduler arbitrates the cluster resources for job and its tasks?  Tasks are scheduled on the order of their job submission. Given job A and submitted in that order, all tasks from job A are guaranteed to finish before all tasks from job B |
| 2 | Fair scheduler  *<< Recommended>>* | Fair scheduling is a method of assigning resources to applications such that all apps get, on average, an equal share of resources over time. Hadoop NextGen is capable of scheduling multiple resource types. By default, the Fair Scheduler bases scheduling fairness decisions only on memory  When there is a single app running, that app uses the entire cluster. When other apps are submitted, resources that free up are assigned to the new apps, so that each app eventually on gets roughly the same amount of resources. Unlike the default Hadoop scheduler, which forms a queue of apps, this lets short apps finish in reasonable time while not starving long-lived apps. (cluster allows short jobs to finish within a reasonable time without starting long-running jobs)  Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time. When there is a single job running, that job uses the entire cluster. When other jobs are submitted, tasks slots that free up are assigned to the new jobs, so that each job gets roughly the same amount of CPU time. Unlike the default Hadoop scheduler, which forms a queue of jobs, this lets short jobs finish in reasonable time while not starving long jobs. It is also a reasonable way to share a cluster between a number of users. Finally, fair sharing can also work with job priorities - the priorities are used as weights to determine the fraction of total compute time that each job should get  <https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/FairScheduler.html> |
| 2.1 | yarn.resourcemanager. scheduler.class | <property>  <name>yarn.resourcemanager.scheduler.class</name>  <value>org.apache.hadoop.yarn.server. resourcemanager.scheduler.fair.FairScheduler</value>  </property> |
| 3 | Capacity Scheduler  *<<Recommended>>*  *<<****Note:*** *Cloudera does not support the Capacity Scheduler in YARN>>*  ***General Concept***  *There is a concept called pooling and each pool has different priority. By default the job will be submitted in the default pool which has medium/less priority. Whenever we need run important jobs, we can assign that job in dedicated pool with high priority.*  *Pool will have additional xml/parameter file* | The CapacityScheduler is designed to allow sharing a large cluster while giving each organization capacity guarantees. The central idea is that the available resources in the Hadoop cluster are shared among multiple organizations who collectively fund the cluster based on their computing needs. There is an added benefit that an organization can access any excess capacity not being used by others. This provides elasticity for the organizations in a cost-effective manner  **Ex:** Consider there are 3 different teams (Team A, B and C). CapacityScheduler will equally distribute the resources to each team. If there is a situation that Team A and C are running their jobs and Team B doesn’t have any job then Team A and C will also use free resources from Team B  <https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html> |
| 3.1 | yarn.resourcemanager. scheduler.class | <property>  <name>yarn.resourcemanager.scheduler.class</name>  <value>org.apache.hadoop.yarn.server. resourcemanager.scheduler.capacity.CapacityScheduler </value>  </property> |
| 4 | pluggable scheduler | The introduction of the pluggable scheduler was yet another evolution in cluster computing with Hadoop. The pluggable scheduler permits the use (and development) of schedulers optimized for the particular workload and application. The new schedulers have also made it possible to create multi-user data warehouses with Hadoop, given the ability to share the overall Hadoop infrastructure with multiple users and organizations |
|  |  |  |
|  |  |  |
|  | **Additional info** |  |
| 1 | Late binding | One good thing in YARN is, the resource model follows the concept of late binding which means the resource is not been used at the time of allocation, only when it is assigned then only it uses resource |
| 2 | developing YARN applications | There are many more in development. Now Storm and Spark.  YARN is very powerful tool coz it brings other application platform paradigm to the platform (Ex: Storm and Spark) |
| 3 | Advantages | 1. Simplified API 2. Reduced development effort |
| 4 | Need to set the Gateway for Yarn after the configuration | CM Home -> Yarn -> Instance (Menu) -> Add roles (button) -> Gateway (select all nodes) -> Stale configuration -> Deploy Client Configuration   1. Refer youtube video: Hadoop Administration - 07 Set up core components 08 Map Reduce YARN 2. Channel : itversity |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | **Yarn commands** | <https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/YarnCommands.html> |
| 1 | Run a job  >hadoop jar <jar> [mainClass] args... | # The syntax is same for both MRV1 and YARN (MRV2) to run the JOB  # To run jar file  **Step 1:** In case if you are not aware what classes that jar contains then run the below command, it will list all the classes available in this jar. Ex: it returns two classes wordcount and randomwriter  >Hadoop jar jar\_filepath\_name  **Step 2:** Run Jar file  Syntax > Hadoop jar jar\_filepath\_name wordcount <input file path> <output file path><unique\_output\_file\_name>  Ex: >Hadoop jar /usr/lib/Hadoop-mapreduce/Hadoop-mapreduce-examples-2.5.0-cdh5.2.1.jar wordcount /user/input /user/output/wrdcount\_output\_unq\_Name  Note: wrdcount\_output\_unq\_Name is required otherwise if more than one person run the same job then job will fail |
| 1.1 | Link to track jobs | # As soon as you submit job using >Hadoop jar, a job id will be allocated. you can track the job in two ways   1. <http://node> name where resource manager configured: 8088 2. Per job ‘Application Master’ (AM) link will be there. 3. Yarn/MRV2 will show the link immediately you submit the job using >haddop jar. Go to that link 4. In MRV1, it shows only job id, not link. Where as in MRV2, it will show both job id and link to track |
| 2 | >yarn application –list | It will return the job status as application\_id, name, user, status (finished, killed jobs), progress and host |
| 3 | >yarn application -kill $ApplicationId | To kill Hadoop yarn jobs  <http://stackoverflow.com/questions/11458519/how-to-kill-hadoop-jobs> |
|  |  |  |
|  |  |  |
| 1 | **Examples** |  |
| 1.1 | Yarn above Example explained here | 1. Client -> RM (Submit App Master) 2. RM -> NM (Start App Master) 3. Application Master -> RM (Request and release Container) 4. RM -> NM (Start task in container) |
| 1.2 | Example 2: Consider you have a server with 100 NameNodes. How many RM you will have ? and explain the functionality of each Yarn components? | 1. Each NameNode will have a unique cluster ID. And each Unique cluster ID will have a unique RM. So I will have 100 RM 2. Consider each NameNode assigned to 10 DataNode then I will have 100\*10 = 1000 DM 3. Consider I have 5 applications (MapReduce, Hive, Hbase, Storm, Spark) and each application requesting RM to run their jobs. Also same application can give more than one job at a time then Application Master will manage it 4. Container? |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | **Pending** |  |
| 1 | Describe the YARN architecture | Done |
| 2 | Describe the YARN application lifecycle | Done |
| 3 | Write a YARN client application | TBD |
| 4 | Run a YARN application on a Hadoop cluster | TBD |
| 5 | Monitor the status of a running YARN application | TBD |
| 6. | View the aggregated logs of a YARN application | TBD |
| 7 | Configure a ContainerLaunchContext | TBD |
| 8 | Use a LocalResource to share application files across a cluster | TBD |
| 9 | Write a YARN ApplicationMaster | TBD |
| 10 | Describe the differences between synchronous and asynchronous ApplicationMasters | TBD |
| 11 | Allocate Containers in a cluster | TBD |
| 12 | Launch Containers on NodeManagers | TBD |
| 13 | Write a custom Container to perform specific business logic | TBD |
| 14 | Explain the job schedulers of the ResourceManager | TBD |
| 15 | Define queues for the Capacity Scheduler | TBD |
|  |  |  |
|  |  |  |
|  | **Acronym** |  |
| 1 | CLC | Container Launch Context |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  | **Ports** | **#** The following is a list of default ports used by MRv2 and YARN, as well as the configuration properties used to configure them. |
| **Port** | **Use** | **Property** |
| 8032 | ResourceManager Client RPC | yarn.resourcemanager.address |
| 8030 | ResourceManager Scheduler RPC (for ApplicationMasters) | yarn.resourcemanager.scheduler.address |
| 8033 | ResourceManager Admin RPC | yarn.resourcemanager.admin.address |
| 8088 | ResourceManager Web UI and REST APIs | yarn.resourcemanager.webapp.address |
| 8031 | ResourceManager Resource Tracker RPC (for NodeManagers) | yarn.resourcemanager.resource-tracker.address |
| 8040 | NodeManager Localizer RPC | yarn.nodemanager.localizer.address |
| 8042 | NodeManager Web UI and REST APIs | yarn.nodemanager.webapp.address |
| 10020 | Job History RPC | mapreduce.jobhistory.address |
| 19888 | Job History Web UI and REST APIs | mapreduce.jobhistory.webapp.address |
| 13562 | Shuffle HTTP | mapreduce.shuffle.port |