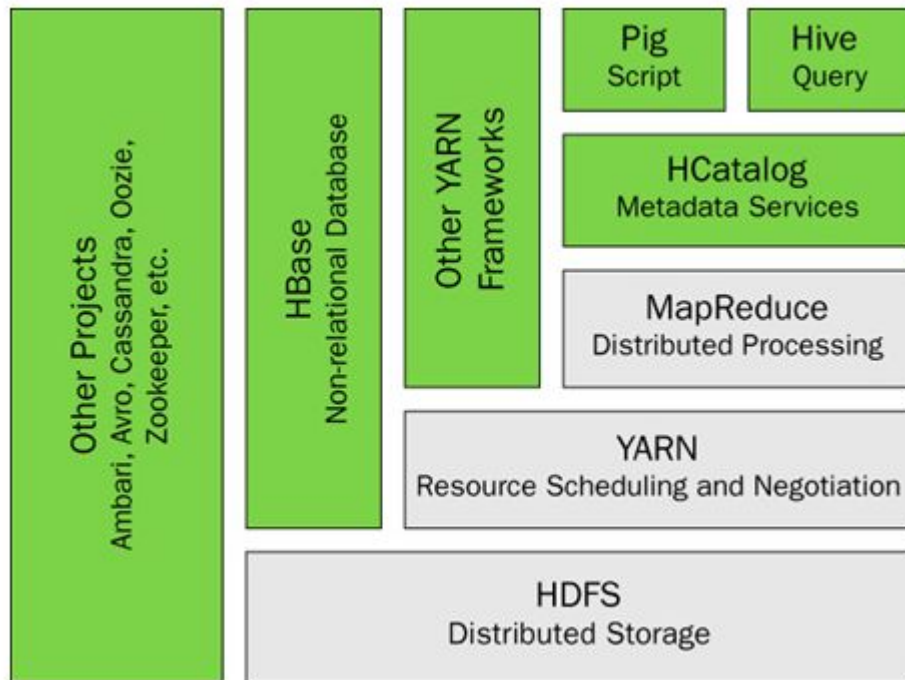


Gemini Hadoop Workshop

01-01-2016

Hadoop Eco System



HDFS

- Appears as single disk and runs on native file system
- Based on Google's file system GFS
- Fault Tolerant
 - Can handle disk crashes, machine crashes etc
- Storing large files
 - Terabytes, Millions rather than billions of files
 - 100MB or more per file
- Streaming Data
 - Write once and read-many times patterns
 - Optimized for streaming read rather than random reads
- Not good for Low Latency reads
 - High throughput rather than low latency of small chunks of data

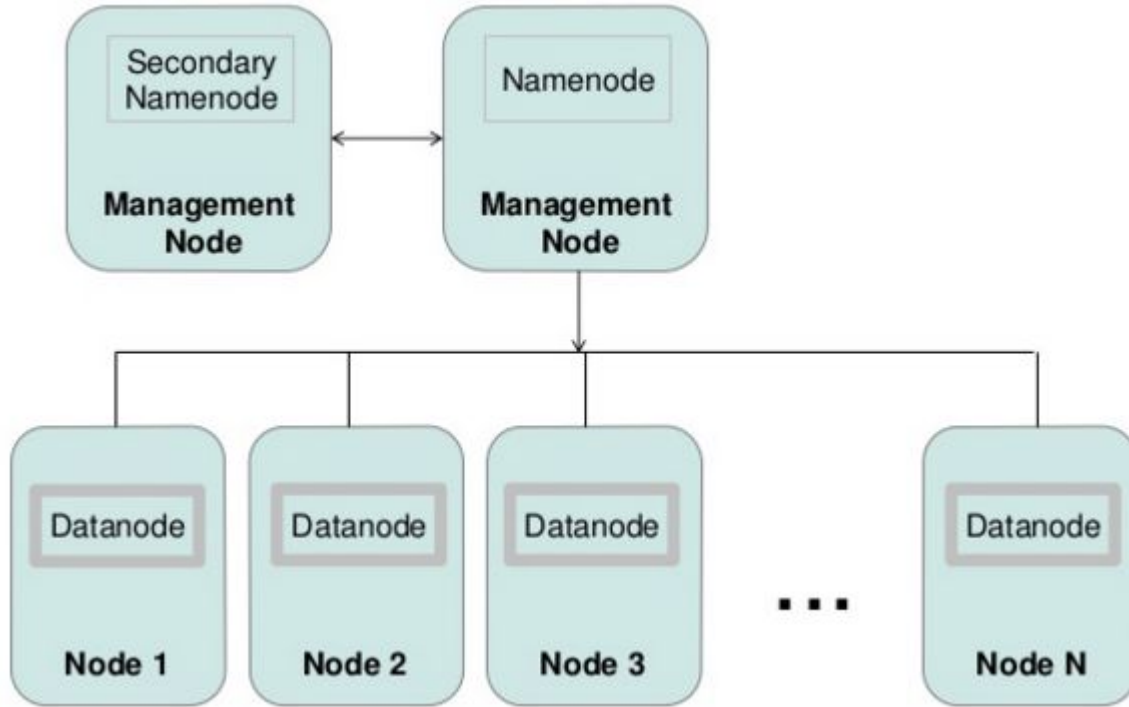
HDFS Abstractions

- Data is organized into files and directories
- Files are divided into uniform sized blocks (default 128MB)
 - A file can be larger than a single disk
- Blocks are replicated (default 3 replicas)
 - Distributed to handle hardware failure
 - Replication for performance and fault tolerance (Rack-Aware placement)
- HDFS exposes block placement so that computation can be migrated to data
- Checksum for detecting corruption
 - DataNode stores checksums in a metadata file separate from the block's data file
- HDFS FileSystem Check - % **hdfs fsck / -files -blocks -locations -racks**

HDFS Daemons

- Operates in master-workers pattern
- NameNode (Master)
 - Manages filesystem's namespace/meta-data/file-blocks
 - knows the datanodes on which all the blocks for a given file are located
 - Runs on 1 machine to several machines
- DataNode (Worker Nodes)
 - Stores and retrieves data blocks
 - Reports to NameNode and runs on many machines
- Secondary NameNode
 - HDFS can be configured in High Availability Mode
 - Secondary NameNode will be in StandBy Mode (receiving all updates)
 - Standby NameNode ready to take over if the active node of NameNode fails

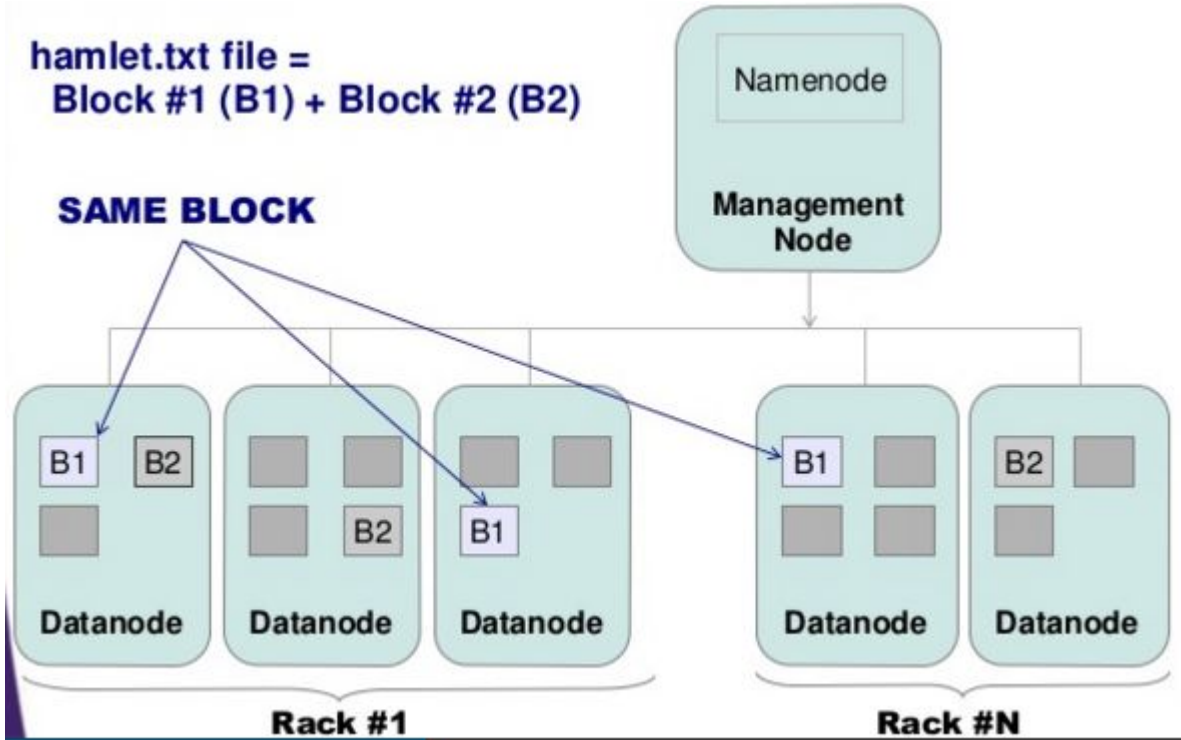
HDFS Daemons



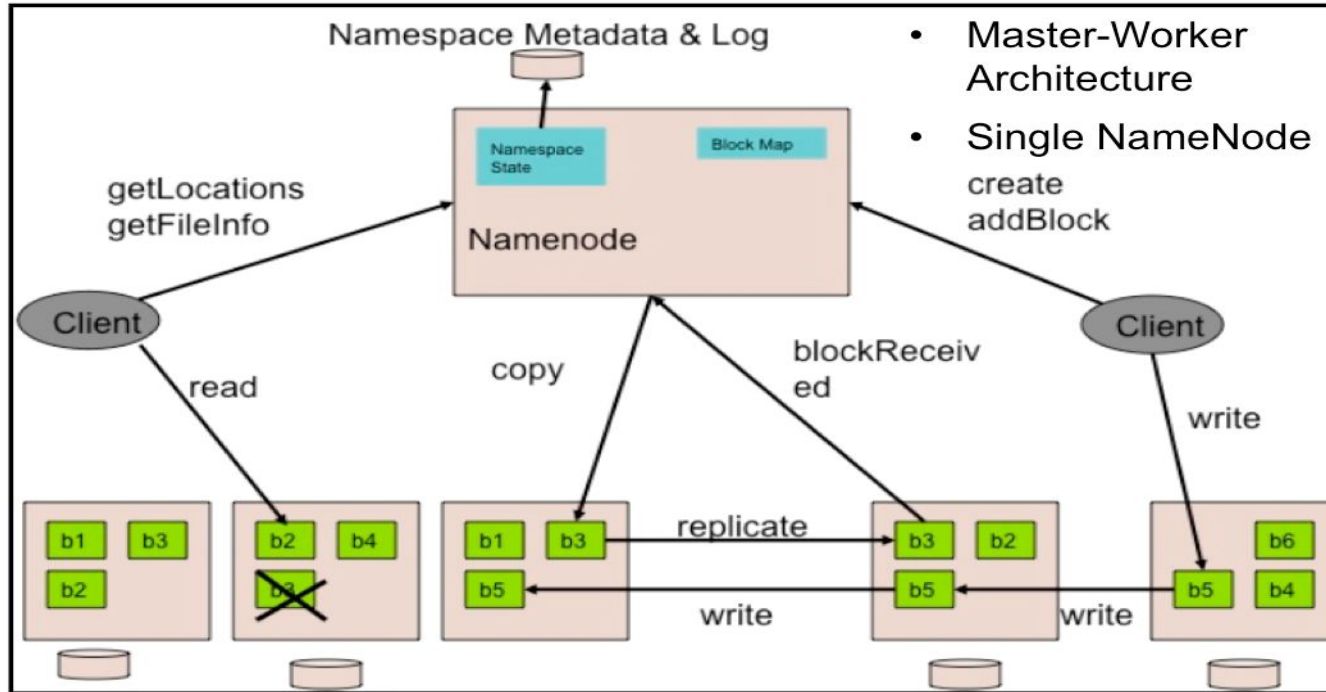
HDFS Files And Blocks

hamlet.txt file =
Block #1 (B1) + Block #2 (B2)

SAME BLOCK



HDFS Architecture



NameNode

- Manages NameSpace
- Maintains all meta-data (inode information) in memory
 - Mapping inode to a list of blocks and locations
 - Schedule replication for under-replicated blocks
- Block Replication
 - NameNode determines replica placement
 - Replica placements are rack aware
 - Attempts to improve reliability by putting replicas on multiple racks
 - Default replication is 3.
 - First replica on local rack
 - Second replica on local rack but different machine
 - 3rd replica on different rack

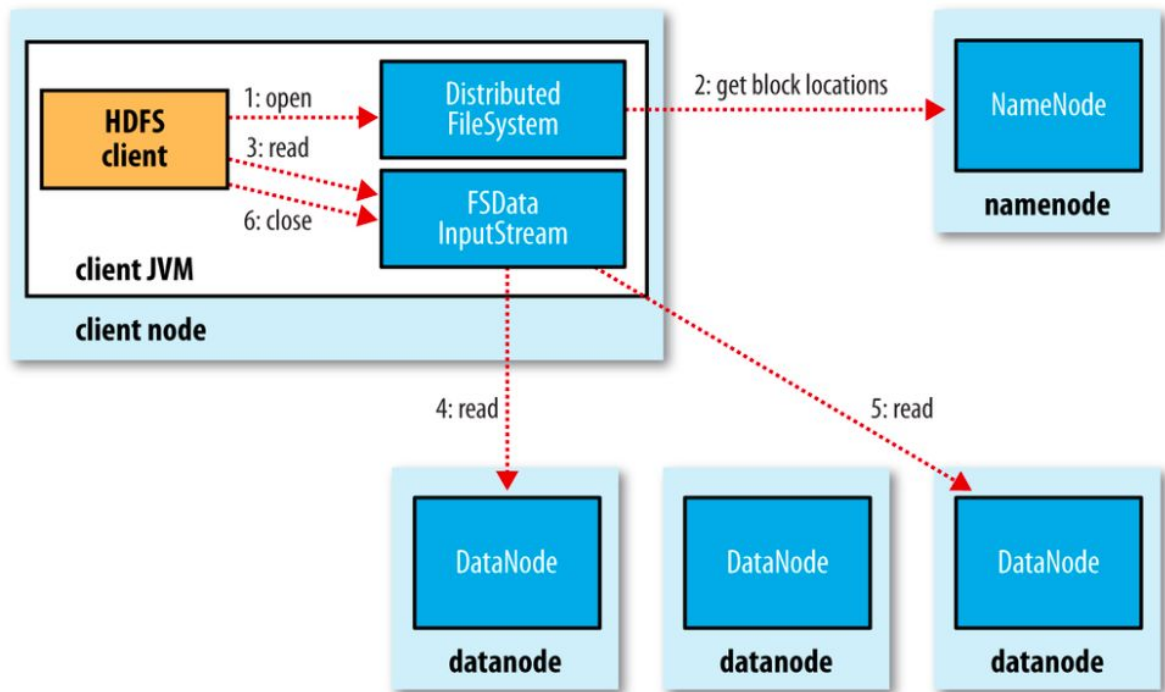
HDFS Client

- NameNode does not directly read or write data
 - Provides HDFS scalability
- Client interacts with namenode to update NameNode's HDFS namespace and retrieve block locations for reading and writing
- Client directly interacts with data node to read and write data.
- Simple command line interface
 - `hadoop dfs -ls / -lsr / -mkdir / -cat`
 - `hadoop dfs -get / -copyToLocal`
 - `hadoop dfs -put / -copyFromLocal`
 - `hadoop dfs -rm/-rmr`
 - `hadoop dfs -chmod / -chgrp /`

HDFS Java API

- Create FileSystem Instance (FileSystem)
 - Creates DistributedFileSystem (based on fs.default.name property - hdfs://localhost:8020)
- Open input stream to path on FileSystem instance
 - Returns FSDataInputStream
- Copy Bytes from InputStream to OutputStream
 - IOUtils.copyBytes(inputstream, outputstream, ...)
- Finally close stream

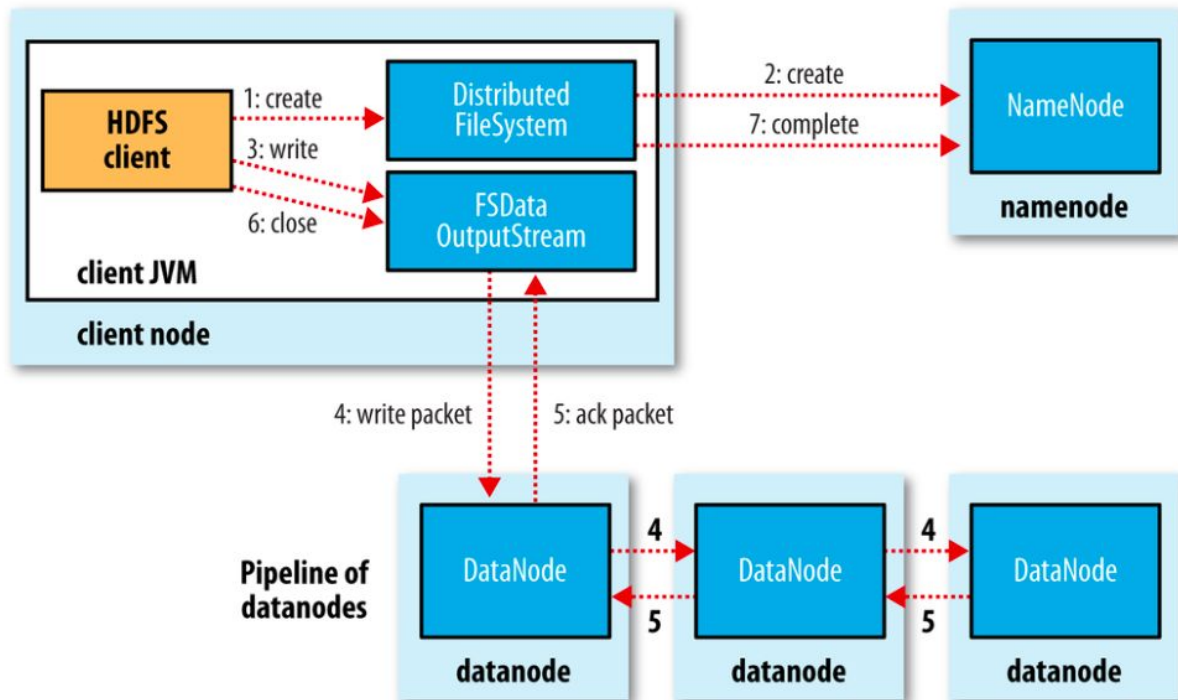
Anatomy of file read



Anatomy of File Read

- HDFS Client initiates Open on DistributedFileSystem (dfs) object
- DFS connects to NameNode (RPC call)
 - NameNode identifies the block locations of file to be read and sends the list of data nodes in order of nearest datanodes from client
- Client initiates read on FSDataInputStream (returned from Open call)
 - Underlying input stream is DFSInputStream
- Connect to the DataNodes selected and gets the block
- When the end of block is reached, DFSInputStream closes the connection and identifies the best datanode for the next block

Anatomy of File Write



HDFS Access Patterns

- Direct
 - Communicate with HDFS directly through native client (Java, C++)
 - Clients retrieve metadata (block locations of a given file) from NameNode
 - Client directly access datanodes
 - JAVA API.
- Proxy Based Access
 - Communicate through a proxy
 - Client ==> Proxy Server ==> [NameNode, DataNode]

NameNode Memory Concerns

- Changing block size will affect how much space a cluster can host
- For example, changing block size from 128MB to 256MB will reduce the number of blocks and significantly increase how much space the name node will be able to support
- NameNode daemon process must be running all time
- NameNode is a single point of failure
 - Active standby is always running and takes over in case main namenode

HDFS Configurations

- Hadoop environment variables
 - `$HADOOP_CONF_DIR/hadoop-env.sh`
- Configurations for NameNode, DataNode, Secondary NameNode
 - `$HADOOP_CONF_DIR/hdfs-site.xml`
 - Check property “dfs.namenode.https-address”
 - `phazonblue-nn1.blue.ygrid.yahoo.com:50470`
 - `$HADOOP_CONF_DIR/core-site.xml` - Configuration settings for Hadoop Core, such as I/O settings that are common to HDFS, MapReduce, and YARN

NameNode UI interface

- Check NameNode in PhazonBlue
 - <http://phazonblue-nn1.blue.ygrid.yahoo.com:50070/dfshealth.html#tab-overview>
- Few interesting Stats
 - 79,402,879 files and directories, 98,929,313 blocks = 178332192 total filesystem object(s).
 - Heap Memory used 42.95 GB of 118.4 GB Heap Memory. Max Heap Memory is 118.4 GB
 - Configured: 25.55 PB, Used: 73.84%
 - Live data nodes: 823 in this cluster
- NameNode Metadata
 - Metadata is present in one file called fsimage
 - Also contains edit file that has a log of all the changes made to the file system.
 - <http://hortonworks.com/blog/hdfs-metadata-directories-explained/>

Understanding HDFS Quotas

- `$ hadoop fsck /path/to/directory`
- Two types of Quotas
 - name quotas - limits number of file and (sub) directory names
 - space quotas - limits the HDFS “disk” space of the directory
 - `$ hadoop dfsadmin -setQuota <max_number> <directory>`
 - `$ hadoop dfsadmin -setSpaceQuota <max_size> <directory>`
- Checking quotas at directory level
 - `$hadoop fs -count -q /path/to/directory`
 - `<QUOTA, REMAINING_QUOTA, SPACE_QUOTA, REMAINING_SPACE_QUOTA, DIR_COUNT, FILE_COUNT, CONTENT_SIZE, FILE_NAME>`
- Changing replication
 - `$ hadoop fs -D dfs.replication=1 -copyFromLocal <file> <some-location-on-hdfs>`

MapReduce Paradigm

- Model for processing large amounts of data in parallel
 - Lots of nodes

MapReduce Model

- Imposes key-value input/output
- Defines map and reduce functions
 - map: $(K1, V1) \Rightarrow \text{list}(K2, V2)$
 - reduce: $(K2, \text{list}(V2)) \Rightarrow \text{list}(K3, V3)$
- Map function is applied to every input-key value pair
- Map function generates intermediate key-value pairs
- Intermediate key-values are sorted and grouped by key
- Reduce is applied to sorted and grouped intermediate key-value pairs
- Reduce emits result key-values

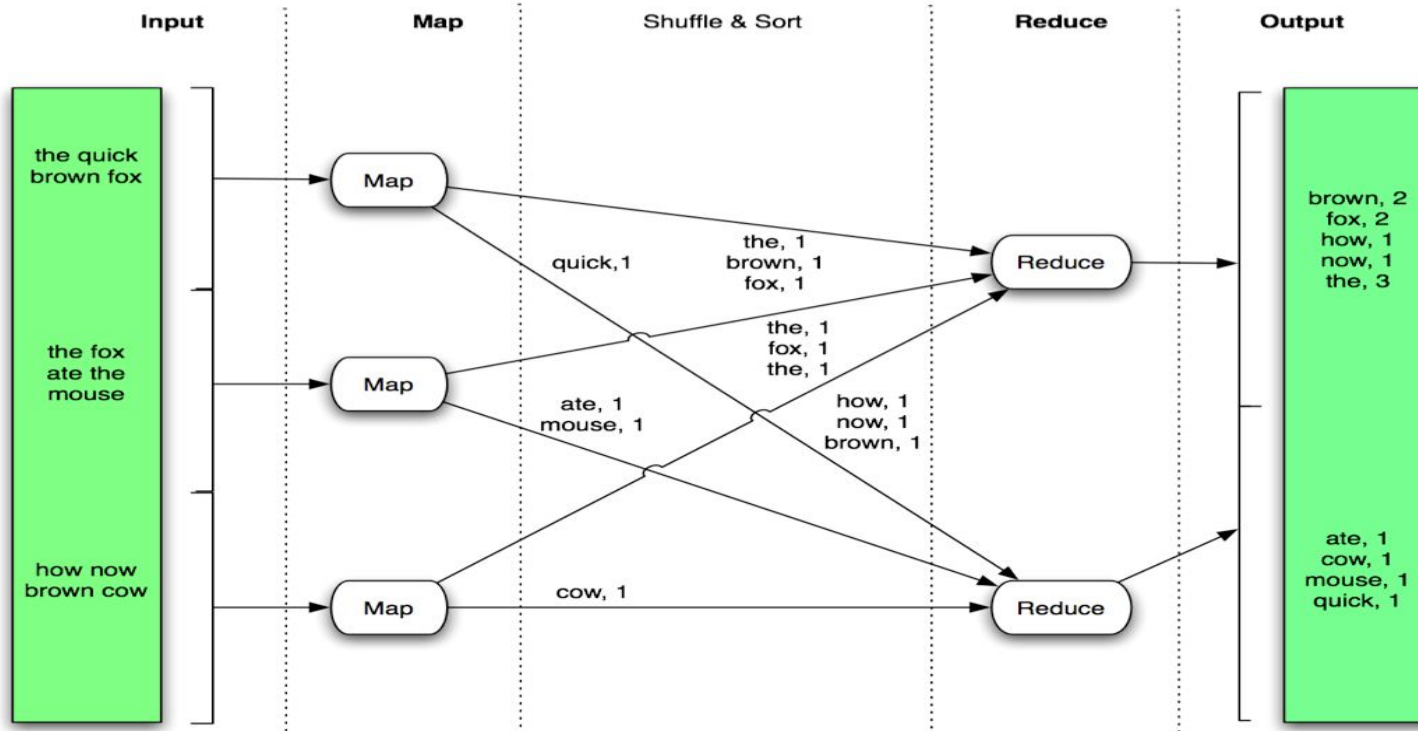
MapReduce Framework

- Takes care of distributed processing and coordination
- Scheduling
 - Jobs are broken down into smaller chunks of tasks
 - These tasks are scheduled
- Task localization with Data
 - Framework strives to place tasks on the nodes that host segment of data to be processed by specific task
 - Code is moved to where the data is
- Error Handling
 - Failure are an expected behaviour so tasks are automatically re-tried on other machines
- Data Synchronization
 - Shuffle and sort barrier re-arranges and moves data between machines

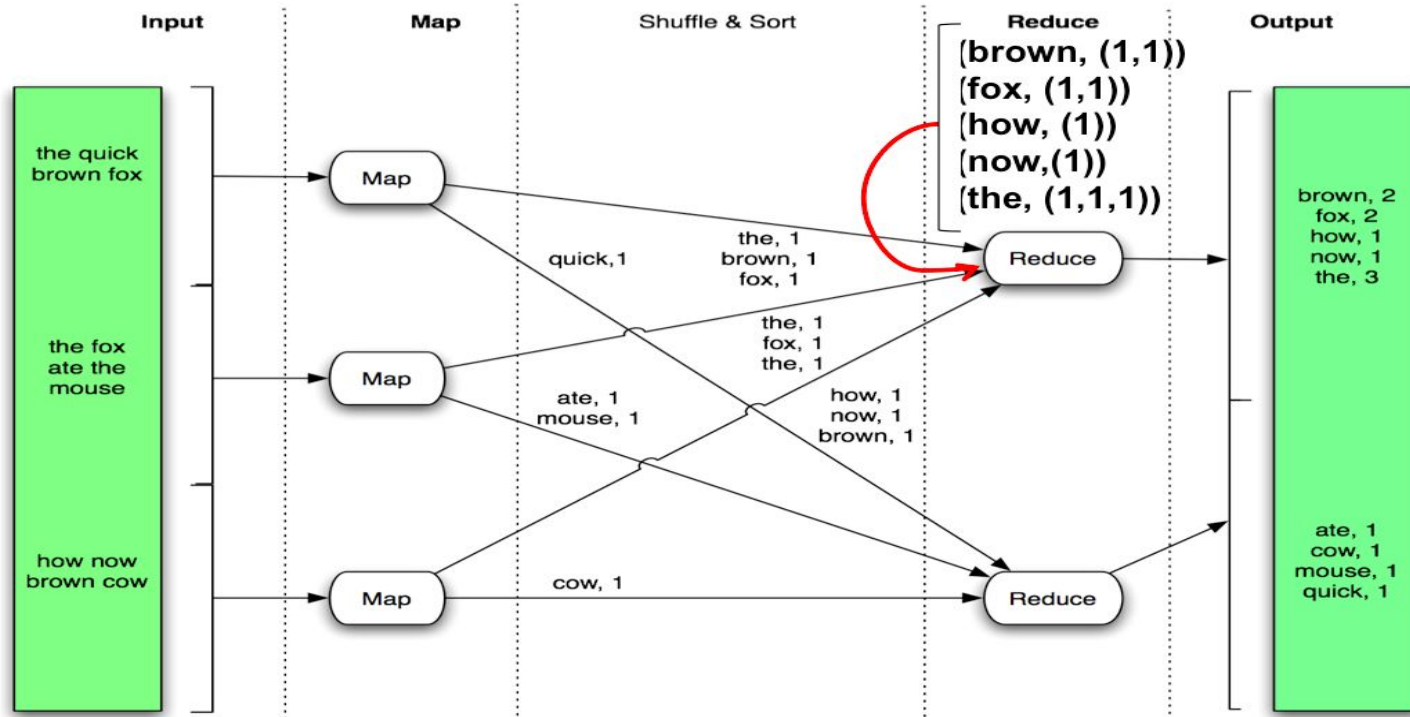
MapReduce - WordCount Example

- Plain Text File
 - `cat sample.txt | sed -e 's/ \n/g' | grep . | sort | uniq -c`
 - Returns word frequencies count
- Mapper - `mapper(filename, file-contents)`
 - for each word in file-contents
 - `emit(word, 1)`
- Shuffle/Sort
- Reducer - `reducer(word, values)`
 - `Sum = 0`
 - for each value in values:
 - `sum = sum + value`
 - Finally `emit(word, sum)`

Word Count Example



Word Count Example

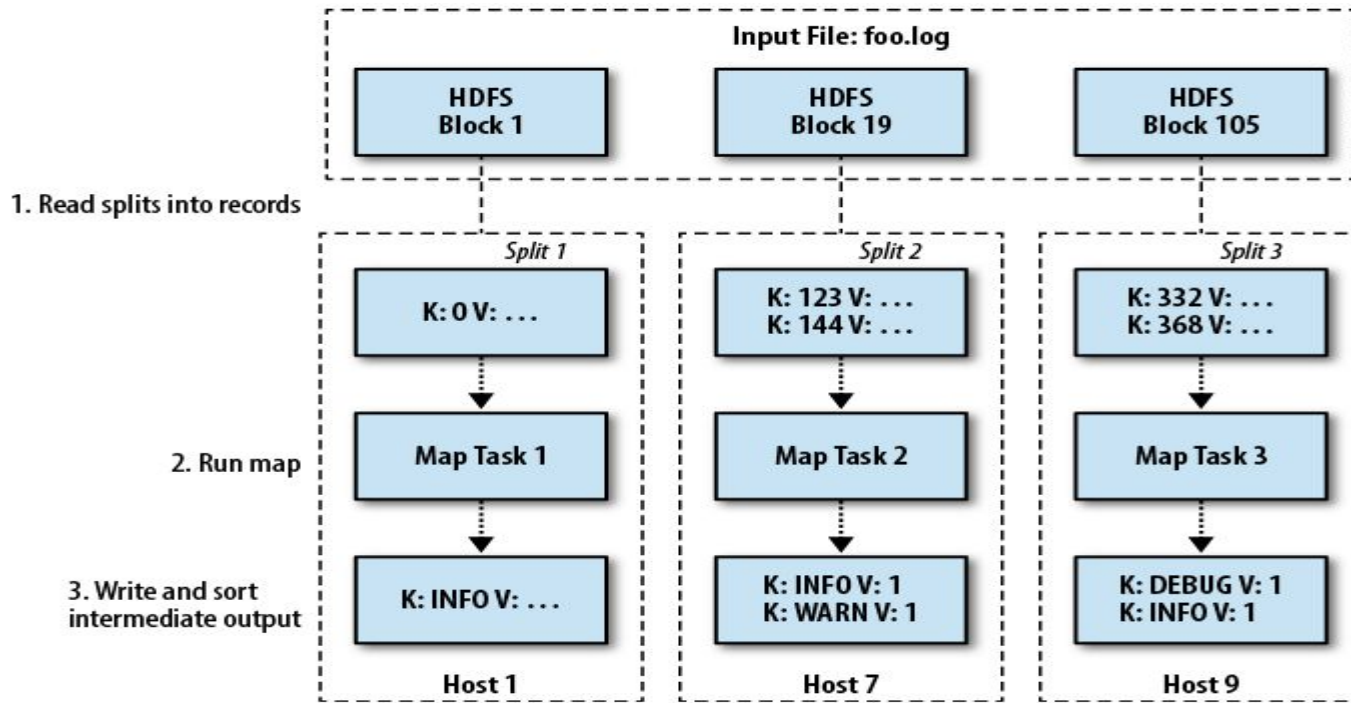


Another Example

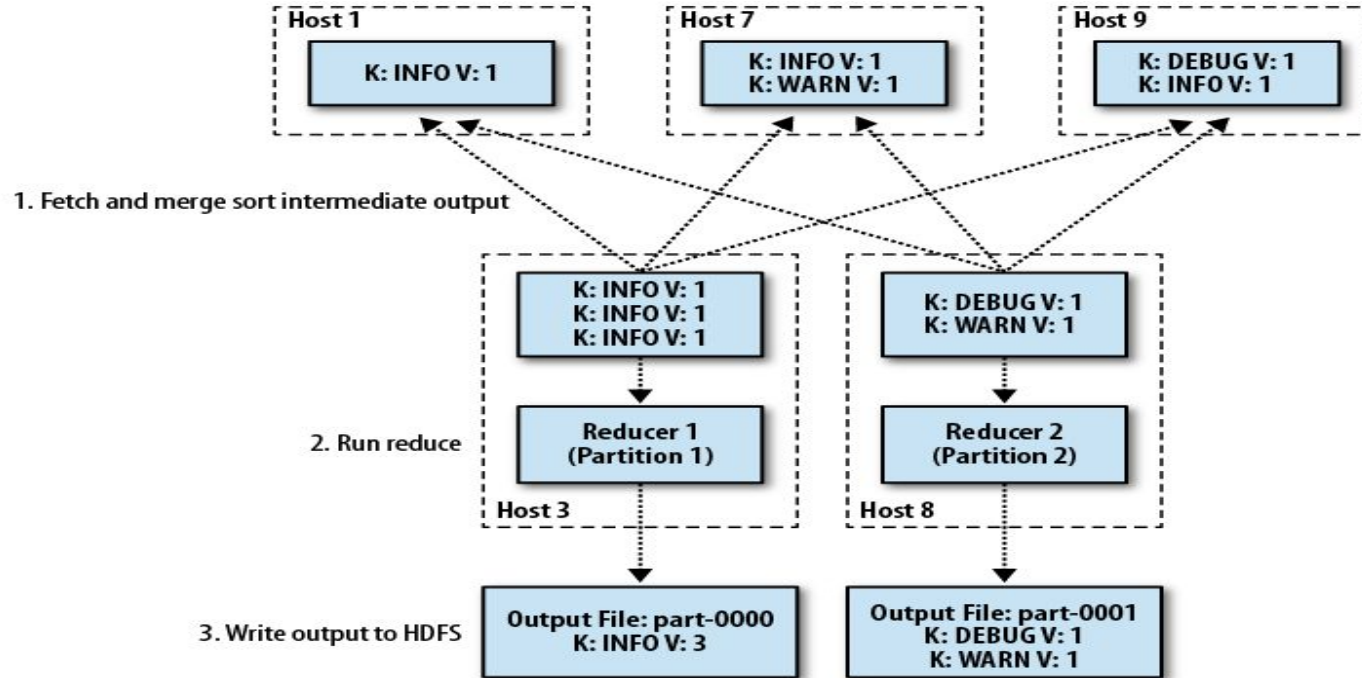
- Count lines by severity in log files

```
SELECT SEVERITY,COUNT(*)  
  
FROM logs GROUP BY SEVERITY  
  
WHERE EVENT_DATE = '2012-02-13'  
  
GROUP BY SEVERITY  
  
ORDER BY SEVERITY
```

Map Execution Phase



Shuffle/Sort/Reduce Phase



MapReduce Data Flow

- Divided into two phases - Map and Reduce phases
- Both phases use key-value pairs as input and output
 - Implementer provides map and reduce functions
- MapReduce framework
 - orchestrates input splitting
 - distributing map and reduce phases
 - Shuffling and sorting
- Job - execution of map and reduce functions to accomplish a task
- Task - A single mapper or reducer

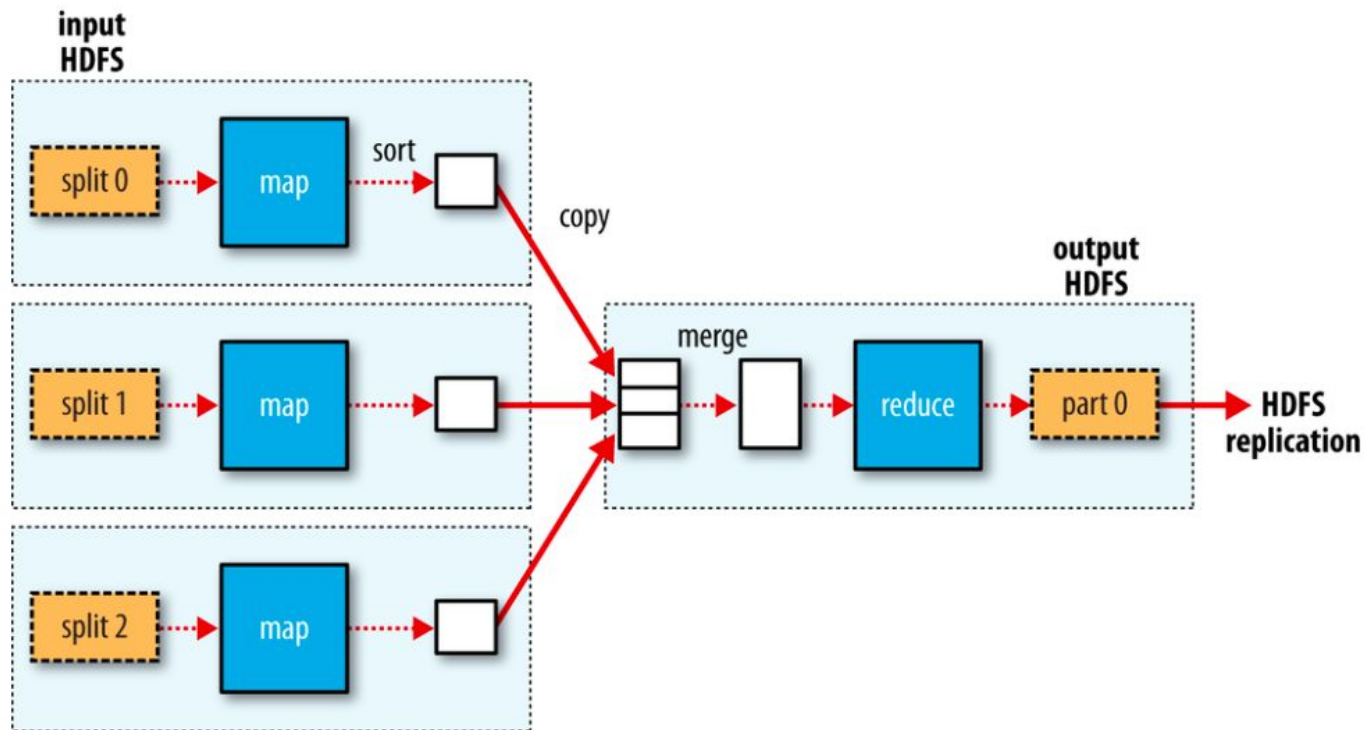
A typical Java Map/Reduce Program

- **Configure a Job**
 - Specify input, output, mapper, reducer, combine and partition functions
- **Implement mapper - WordCount Example**
 - Input is text file
 - Tokenize the text and emit each token with count of 1
- **Implement reducer**
 - Sum up counts for each token
 - write out results to HDFS
- **Finally run the job**
- **Source:** <https://git.corp.yahoo.com/hadoop/Hadoop/blob/branch-2.6/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples/WordCount.java>

Job Execution

- Tasks are scheduled using YARN and run on nodes in the cluster
- Hadoop divides input to Job into fixed-size pieces - called input splits
- Hadoop creates one map task for each split which runs mapper function for each record in the split
- Hadoop tries to run maptask where input data resides in HDFS (data locality optimization)
- Map tasks write their output to local disks not to HDFS
- Maps output is processed by reduce tasks

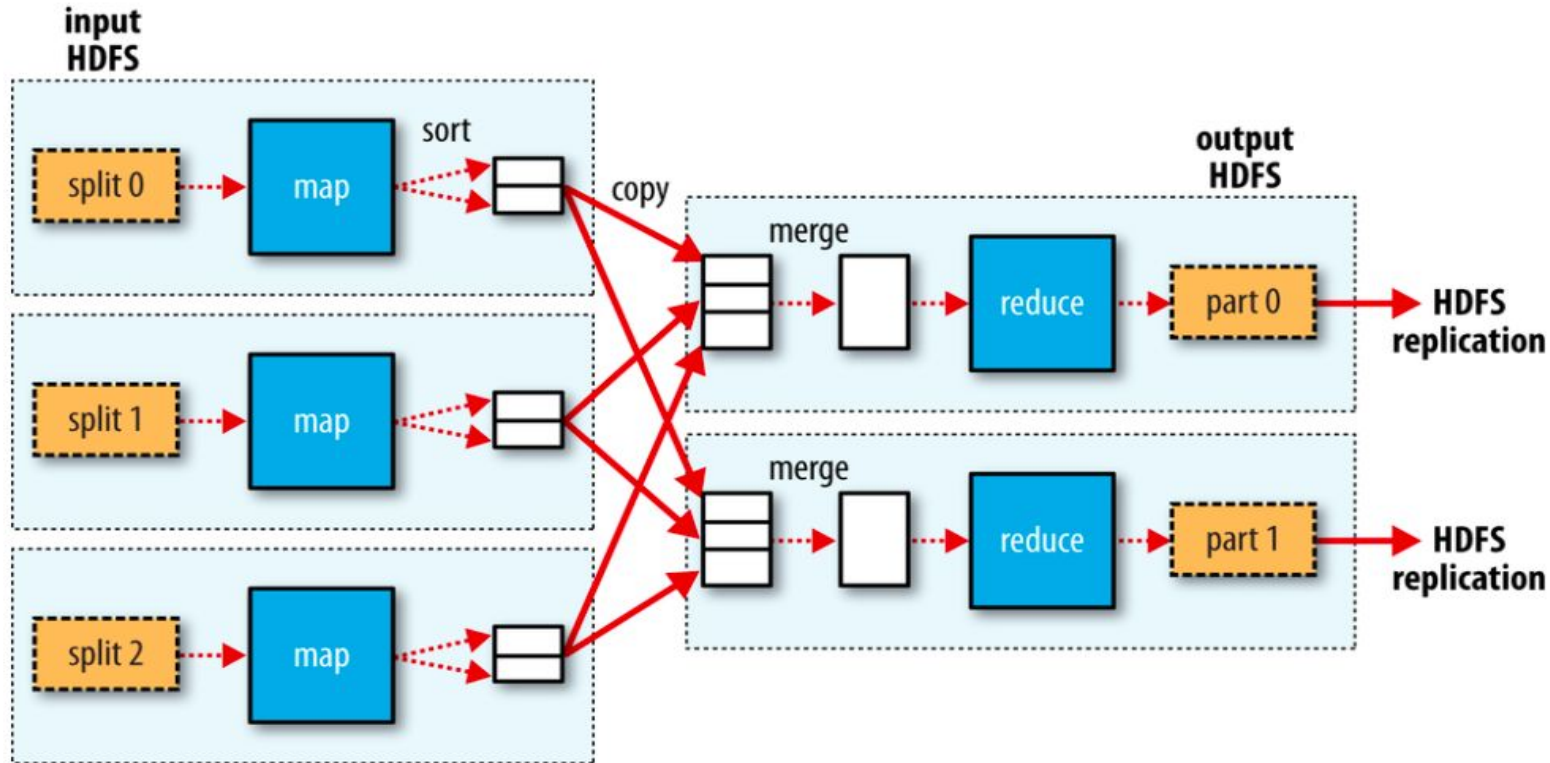
Map-Reduce data flow with single reducer



Map/Reduce Data Flow - 2

- Sorted map outputs are pulled by reducer tasks across network
- Results are merged (aka merge sort by key)
- Now (key, list(value)) are passed to reduce function
- Output of reduce is stored on HDFS
- When there are multiple reducers, map task partitions their output, each creating one partition for each reduce task

Map/Reduce flow with multiple reducers

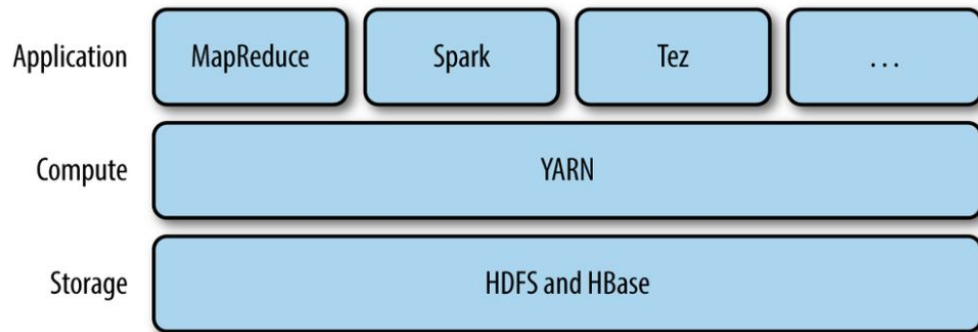


Combiners

- Minimize the data transfer between map and reduce tasks
- Map-Side Pre Aggregation
- Use can specify the combiner function to be run on mapper output
- Combiner won't replace reduce function, but it can apply reduce functionality on mapper output for the same set of keys
- Combiner - WordCount Example
 - For example, if an input split has 10,000 occurrences of word "the", the mapper will generate 10,000 (the,1) pairs, while the combiner will generate one (the,10,000) thus reducing the amount of data transferred to the reduce task.
 - Combiner only works with commutative and associative functions - Same idea for computing mean will not work

YARN

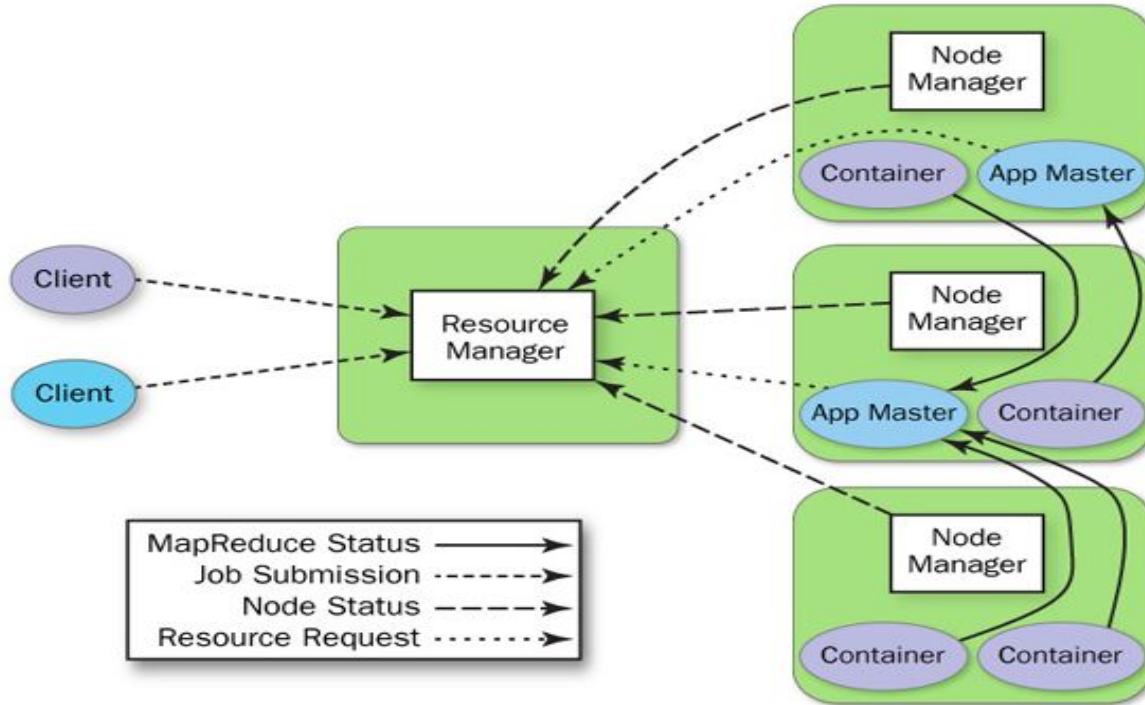
- Hadoop cluster resource management
- Provides APIs for requesting and working with cluster resources
- Various distributed computing frameworks were built using YARN API
 - Map/Reduce, Spark, Tez etc



YARN Daemons

- Node Manager
 - Manages resources of a single node
 - There is one instance per node in the cluster
- Resource Manager
 - Manages Resources for the entire cluster
 - Instructs node manager to allocate resources
 - Application negotiates for resources with resource manager
 - There is only one instance of resource manager in the cluster
- MapReduce Specific Daemon
 - MapReduce History Server
 - Archives jobs metrics and meta-data

YARN Architecture



MapReduce on YARN Components

- Client - Submits map/reduce job
- ResourceManager - Controls the usage of resources across cluster
- NodeManager
 - Runs on each node in cluster
 - Creates execution container and monitors containers usage
- Map/Reduce Application Master
 - Co-ordinates and manages map/reduce jobs
 - Negotiates with resource manager to schedule map/reduce tasks
 - Tasks are started by NodeManager

YARN - Recap

ResourceManager

- A per-cluster service
- Authority that arbitrates resources among all the applications in the system.
- Has a pluggable scheduler for cluster resource optimization & an ApplicationsManager

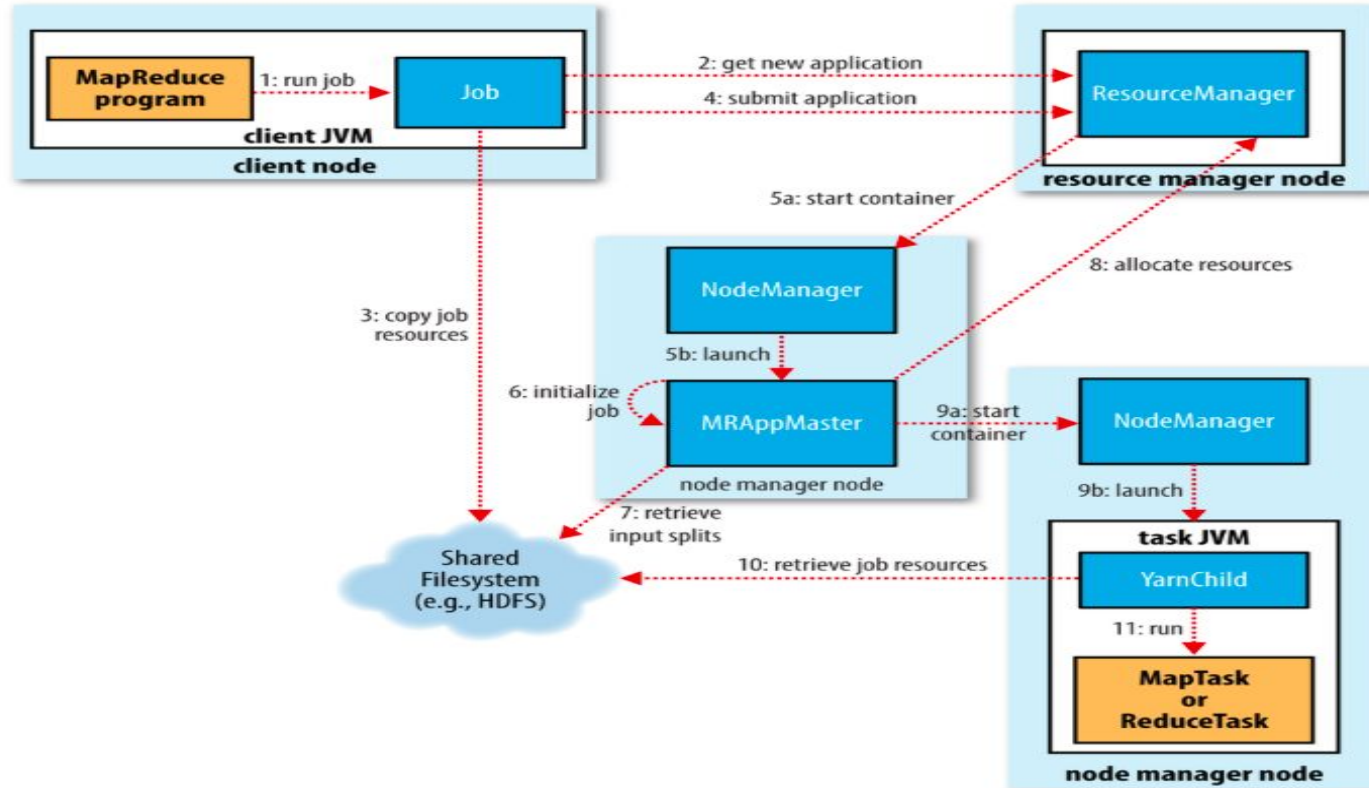
NodeManager

- A per-machine framework agent / the "worker" daemon in YARN
- Creates applications' execution container, monitors their resource usage and reports to the ResourceManager

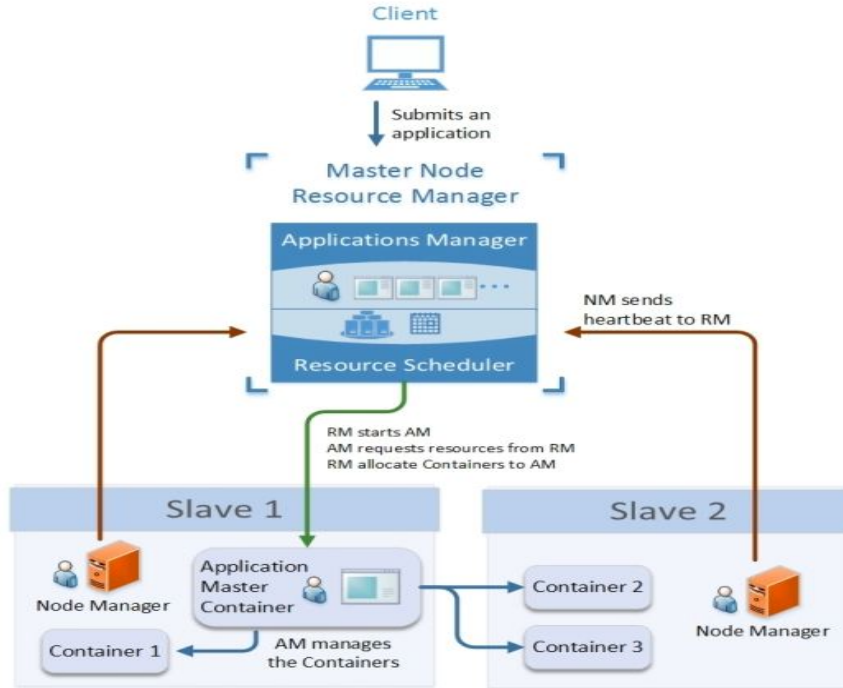
ApplicationMaster

- A per-application framework specific library
- Negotiates resources from the ResourceManager
- Works with the NodeManager(s) to execute and monitor the tasks

Map/Reduce Job Execution on YARN



Recap -- YARN Architecture - Simplified



Application Execution Flow

- Application Initialization and Submission
 - First get application id from RM and then prepare Application Submission Context - (Application, Queue, How to start ApplicationMaster (command) etc and submit
 - Launch Context contains information related to AM (like memory required etc)
- Allocate memory and start ApplicationMaster on a NodeManager
- ApplicationMaster Registration and resource Allocation
 - Registers with RM and it also sends tracking URL for application
 - Requests resources (aka Containers) - include desired capabilities of containers
- Launch and monitor containers
- Application progress report
- Application completion

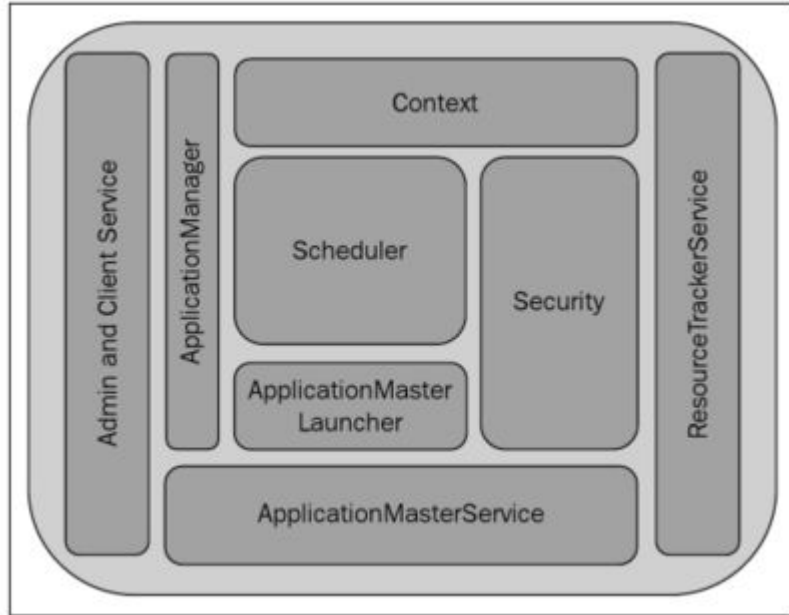
Map/Reduce Job Submission Steps

- Application Id is retrieved from Resource Manager
- Job Client verifies output specification of the job
 - Like output directory already exists checks ...
- Computes input splits
- Copy Job resources to HDFS
 - Jar files, configurations and input split specifications to HDFS
- Defines an ApplicationMaster and a command to start the AM on a node.
- Finally submit the job

Job Initialization Steps

- ResourceManager receives request for a new application
- ResourceManager delegates to its internal component - Scheduler
- ApplicationsManager requests a container for an application master process
 - MapReduce application master is MRAppMaster
- MRAppMaster initializes its internal objects and executes a job by negotiating with ResourceManager

ResourceManager - in depth



Resource Manager

- Client Service - Application Submission/Termination etc
- Administration Service - Adding Queues/Adding Nodes/User-Group-Mappings
- Applications Manager - Maintains housekeeping of all submitted applications
- Application Master Launcher - Launches ApplicationMaster by communicating to Node Managers
- Yarn Scheduler - Responsible for allocating resources - Queue Management
- Application Master Service - Responds to requests from ApplicationMasters - Container Allocation
- AMLivenessMonitor - Monitors ApplicationMaster through heartbeats
- Resource Tracker Service - Registration of NewNodes, Monitoring new nodes
 - Node List Manager - Maintains the list of resource nodes (aka computing nodes)
 - NodeManager Liveness Monitor - Responds to heartbeats from resource nodes in the system
- Yarn Scheduler - Responsible for allocating resources - Queue Management

Resource Manager Entry Points Summary

Port	Use	Property	
8030	ResourceManager Client RPC	yarn.resourcemanager.address	
8032	ResourceManager Scheduler RPC (ApplicationMaster)	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	

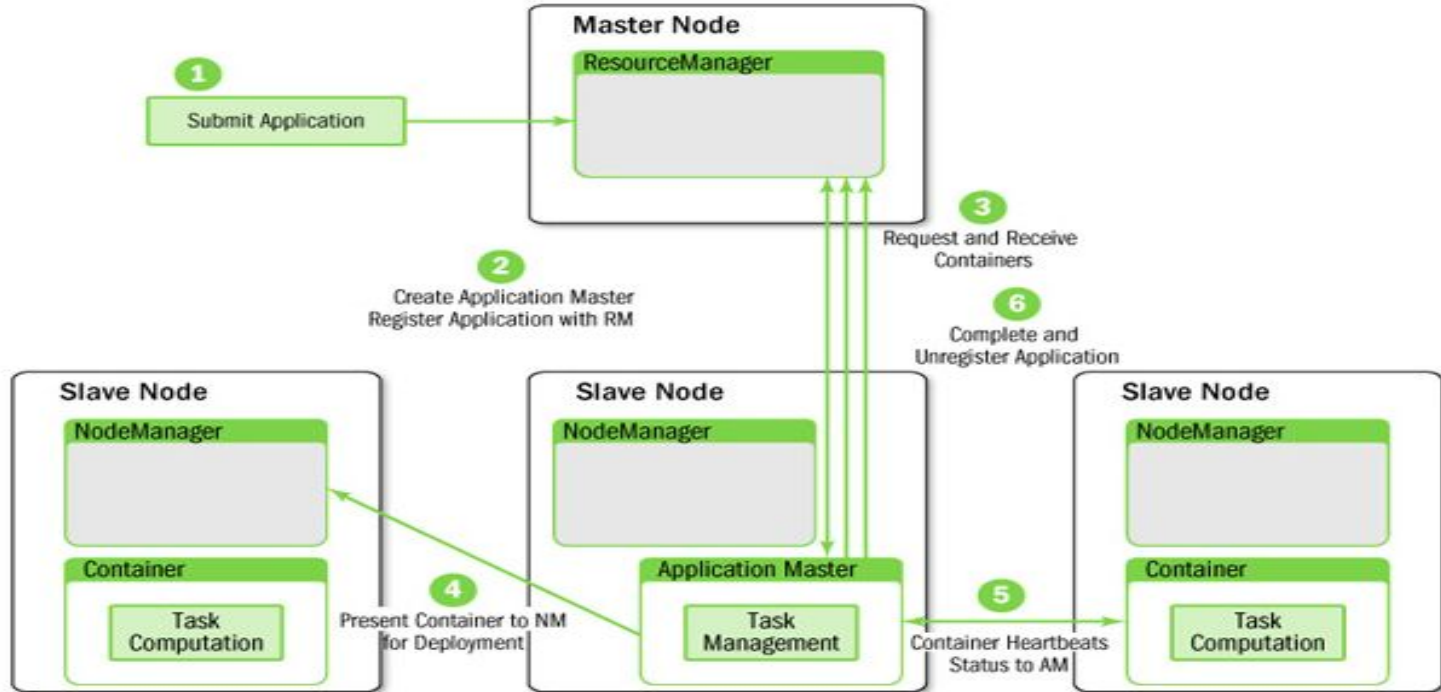
Resources in YARN

- Defines Two resources
 - virtual cores (vcore) and memory
 - Each node manager tracks its own local resources and report to RM
 - RM has global view of available resources in the cluster
 - vcore - Usage share of CPU core
- Containers
 - A request to hold resources in the cluster
 - Currently it contains of vcore and memory
 - Application Master requests containers from RM to run Map/Reduce tasks
- YARN Configuration - yarn-site.xml

MRAppMaster Initialization Steps

- Creates internal bookkeeping objects to monitor progress
- Retrieves input splits
 - These were created by JobClient and copied onto HDFS
- Creates Tasks
 - Map tasks per split
 - Reduce tasks based on mapreduce.job.reduces property
- Decides how to run the tasks
 - Negotiates containers with resource manager
 - Container is the basic unit of hardware allocation, for example a container that has 4 GB of RAM and one CPU
 - Executes tasks on NodeManager

Application Master Interaction in YARN



Task Assignment

- MRAppMaster negotiates container for map and reduce tasks from RM
- Request carry
 - Data locality information (hosts & racks) computed by InputFormat and stored inside input splits information
 - Memory requirements for a task
- Scheduler on RM utilizes the provided information to make a decision on where to place these tasks
 - Attempts data locality: Ideally placing tasks on the same nodes where the data to process resides. Plan B is to place within the same rack

YARN Containers

- Represents unit of work in application
- Runs on a node managed by NodeManager
- Container Environment - User libraries, Jars, Input/Output paths etc
- Communication with Application Master - Containers send progress to AM

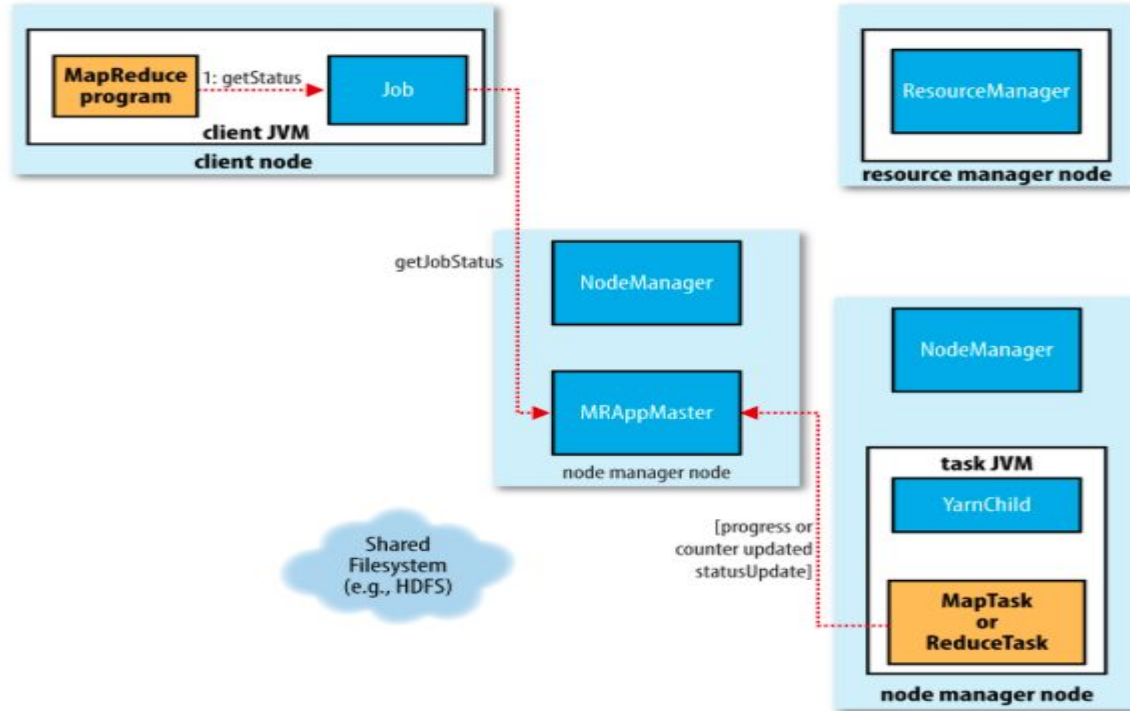
Task Execution

- MRAppMaster requests Node Manager to start containers
- For each task NodeManager start container
 - A java process with YarnChild as the main class
 - YarnChild is executed in the dedicated JVM
 - Separates user code from long running daemons
- YarnChild copies required resources locally
 - Configuration, Jars etc
- YarnChild executes map or reduce tasks

Status Updates

- Tasks report status to MRAppMaster
 - Poll every 3 seconds
- MRAppMaster accumulates and aggregates the information to assemble current status of job
 - Determines if job is done
- Client (Job Object) polls MRAppMaster for status updates
- ResourceManager UI displays all the running YARN applications where each one is a link to web UI of Application Master

Status Updates Flow



Failures

- Failures can occur in
 - Tasks
 - Tasks exceptions and JVM crashes are propagated to MRAppMaster
 - Attempt is marked as 'failed'
 - Task is considered to be failed after 4 attempts
 - `mapreduce.[map,reduce].maxattempts` property controls number of attempts
 - Application Master -- MRAppMaster
 - RM receives heartbeats from MRAppMaster can restart it in case of failure
 - Node Manager
 - Failed NodeManager will not send heartbeats to RM
 - RM will blacklist a Node Manager that hasn't reported within 10 minutes
 - Tasks on failed NM are recovered and placed on healthy nodes
 - Resource Manager - Jobs or Tasks can not be launched - very serious downtime

Job Scheduling and Completion

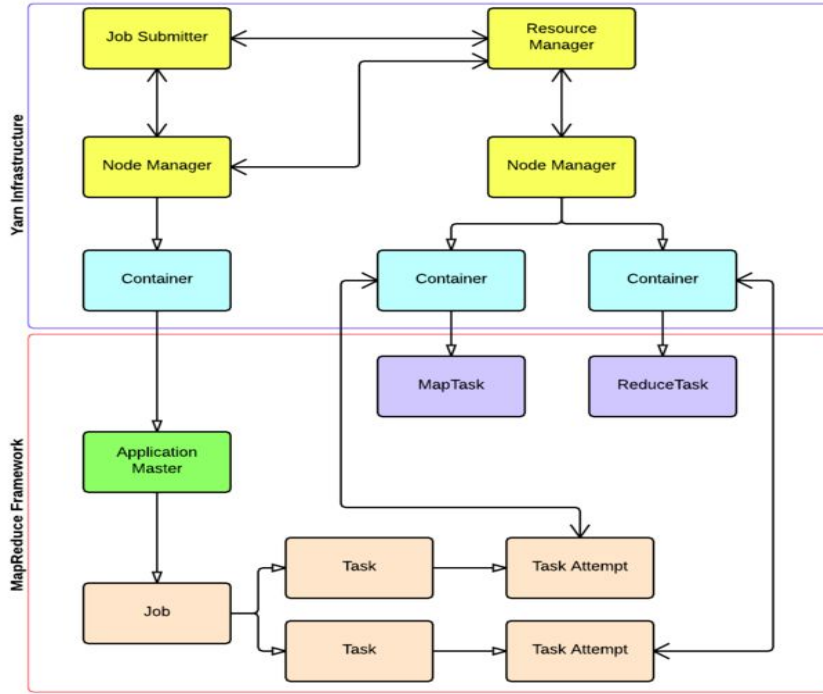
- By default FIFO scheduler is used
 - Capacity and Fair schedulers are also available
- Management and metrics information is sent from MRAppMaster to MapReduce History Server
- Capacity Scheduler
 - Allows sharing of cluster resources among groups (like mx3, search, native etc)
 - Each group is setup with dedicated queue and configured to use certain capacity of cluster
 - Queues may be further divided in hierarchical fashion
 - If there is more than one job in the queue and there are idle resources available, then the Capacity Scheduler may allocate the spare resources to jobs in the queue, even if that causes the queue's capacity to be exceeded - Queue elasticity

MapReduce Examples

- Running MapReduce Examples

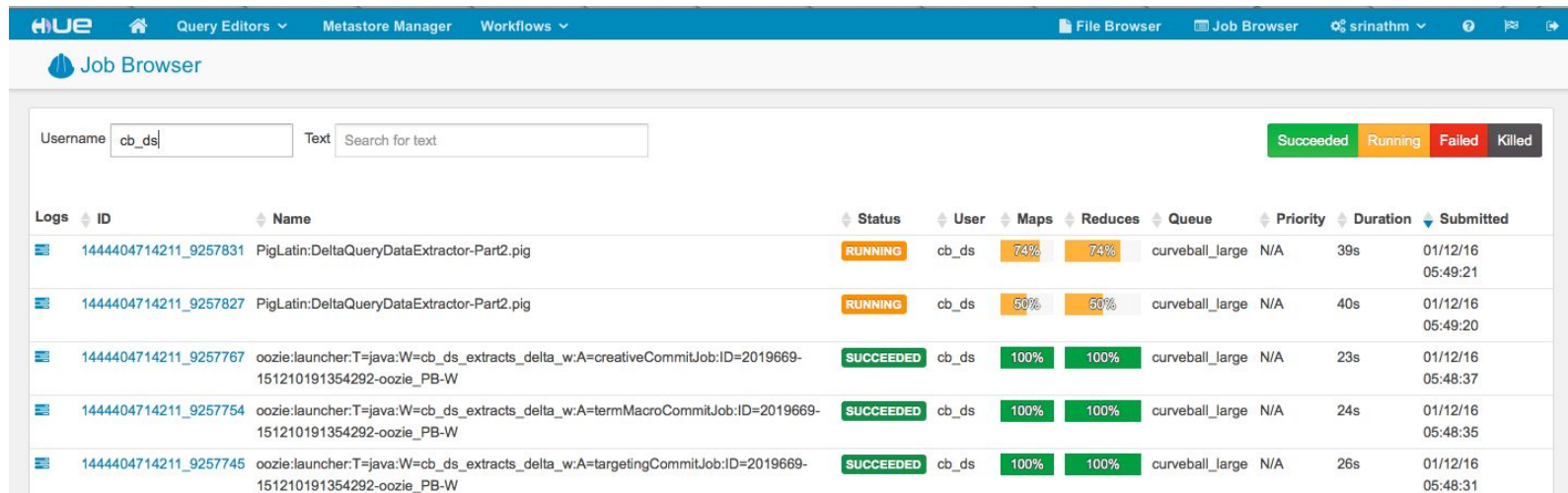
- `yarn jar $HADOOP_PREFIX/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.0.24.1509041917.jar <program name> <parameters>`
- Source Code
 - <https://git.corp.yahoo.com/hadoop/Hadoop/blob/branch-2.6/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples/>
- Example Programs
 - WordCount, WordMean, Sort, SecondarySort, Join, Grep ...
- WordCount example
 - `yarn jar $HADOOP_PREFIX/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.0.24.1509041917.jar wordcount /projects/cb_nctr/marketplace/gpa_queries/ /user/srinathm/hw`

Yarn And MapReduce Frameworks Intereaction



Job Browser UI

- Hue - <http://yo/hue.pb> (phazon blue cluster)
- Explore Job Browser
 - Explore jobs by user id (example headless users: cb_gpa, cb_nctr, cb_ds etc)

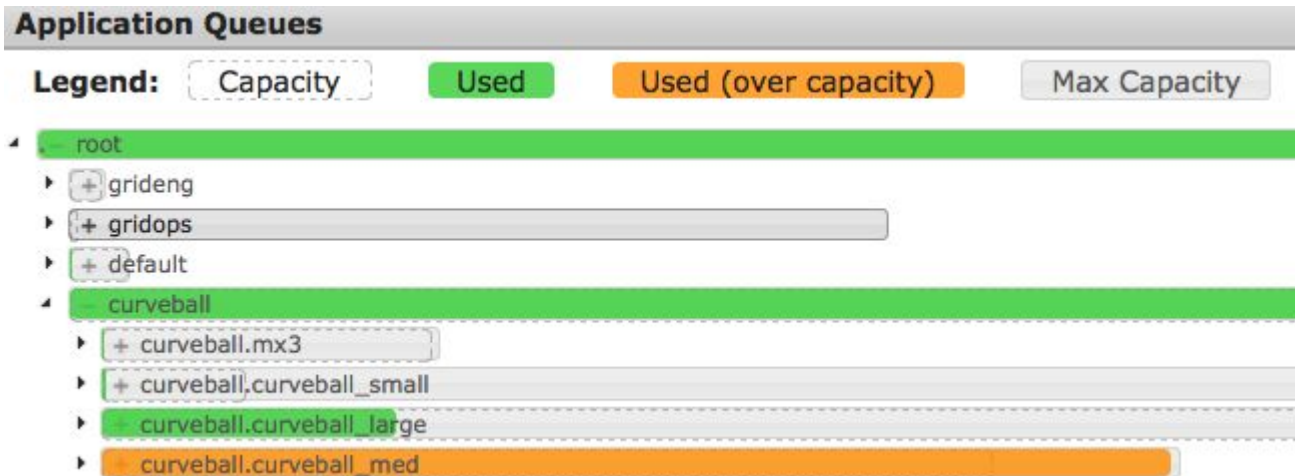


The screenshot shows the Hue Job Browser interface. At the top, there's a navigation bar with 'Hue' logo and links to 'Query Editors', 'Metastore Manager', 'Workflows', 'File Browser', 'Job Browser', and a user profile 'srinathm'. Below the navigation bar, the 'Job Browser' title is displayed. A search bar is present with 'Username' set to 'cb_ds' and a 'Text' search field. To the right of the search bar are four status filters: 'Succeeded' (green), 'Running' (orange), 'Failed' (red), and 'Killed' (dark grey). The main content area is a table of jobs.

Logs	ID	Name	Status	User	Maps	Reduces	Queue	Priority	Duration	Submitted
	1444404714211_9257831	PigLatin:DeltaQueryDataExtractor-Part2.pig	RUNNING	cb_ds	74%	74%	curveball_large	N/A	39s	01/12/16 05:49:21
	1444404714211_9257827	PigLatin:DeltaQueryDataExtractor-Part2.pig	RUNNING	cb_ds	50%	50%	curveball_large	N/A	40s	01/12/16 05:49:20
	1444404714211_9257767	oozie:launcher:T=java:W=cb_ds_extracts_delta_w:A=creativeCommitJob:ID=2019669-151210191354292-oozie_PB-W	SUCCEEDED	cb_ds	100%	100%	curveball_large	N/A	23s	01/12/16 05:48:37
	1444404714211_9257754	oozie:launcher:T=java:W=cb_ds_extracts_delta_w:A=termMacroCommitJob:ID=2019669-151210191354292-oozie_PB-W	SUCCEEDED	cb_ds	100%	100%	curveball_large	N/A	24s	01/12/16 05:48:35
	1444404714211_9257745	oozie:launcher:T=java:W=cb_ds_extracts_delta_w:A=targetingCommitJob:ID=2019669-151210191354292-oozie_PB-W	SUCCEEDED	cb_ds	100%	100%	curveball_large	N/A	26s	01/12/16 05:48:31

Hadoop UI

- Cluster Summary
 - <http://phazonblue-jt1.blue.ygrid.yahoo.com:8088/cluster> (Phazon Blue)
 - Applications - Submitted/Pending/Running (queue/user/start time/tracking UI)
 - Nodes - Metrics about each computing node - Number of containers running, Memory used
- Scheduler

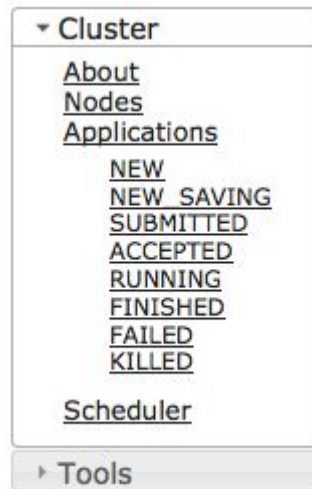
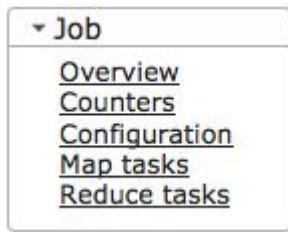


Capacity Scheduler

- Configure guaranteed capacities across organizational groups - Example
 - curveball_mx3 (12%),
 - curveball_small (5.3% but can go upto 50%)
 - curveball_med (48.7% but can go upto 80%)
 - curveball_large (33.8% but can go upto 40%)
- Configure resource limits to ensure against a single large application from monopolizing the cluster resource
 - Enforcing capacity limits - min/max capacity %
- Configure resource and access control limits for various users within an organization group
 - Who can submit applications to which queues?
 - What is is max % of queue capacity a single user can use?

Hadoop UI - 2

- Quick view of configuration
 - <http://phazonblue-jt1.blue.ygrid.yahoo.com:8088/cluster> (Tools/Configuration)
- Click on Application Master Tracking UI for a given application
 - Contains all active jobs associated
 - Current Status, Maps/Reducers progress
- Explore Job



YARN - UI

- Resource Manager Web-UI
 - Cluster Resource Usage, Job Scheduling and running Jobs
 - <http://phazonblue-jt1.blue.ygrid.yahoo.com:8088/cluster>
- Application Proxy Web-UI
 - Proxy between the application master and client
 - Application Master(AM) has the responsibility to provide a web UI, sends link to RM
 - http://phazonblue-jt1.blue.ygrid.yahoo.com:8088/proxy/application_1452874451211_25674/
- Node Manager Web-UI
 - Node information and containers being executed
 - <http://gsbl584n07.blue.ygrid.yahoo.com:8042/node> (Example)
- MapReduce History Server Web-UI
 - Provides the details of already executed jobs
 - http://phazonblue-jt1.blue.ygrid.yahoo.com:19888/jobhistory/job/job_1452874451211_245577
(Example)

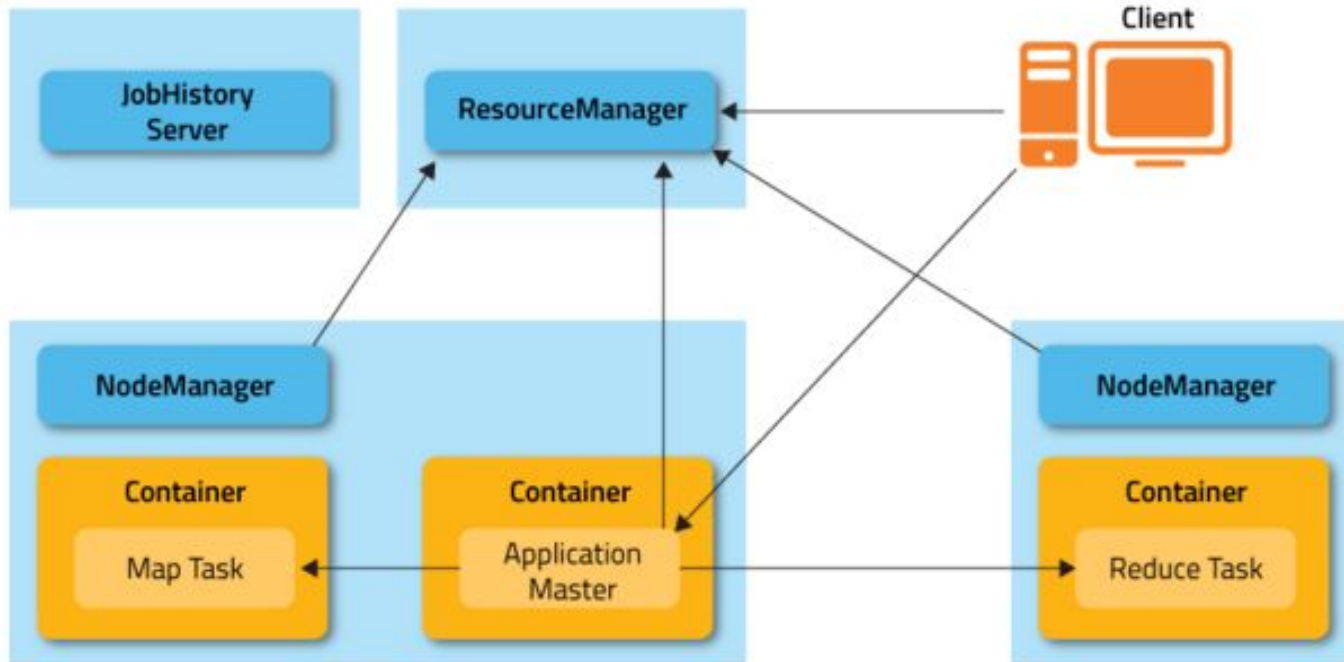
Job Counters

- File System Counters
- Job Counters
- Map/Reduce framework counters
- File input format counters
- File Output Format counters
- Application specific counters

Job Counters	<u>Launched map tasks</u>
	<u>NUM_UBER_SUBMAPS</u>
	<u>Other local map tasks</u>
	<u>Total megabyte-seconds taken by all map tasks</u>
	<u>Total time spent by all map tasks (ms)</u>
	<u>Total time spent by all maps in occupied slots (ms)</u>
	<u>Total vcore-seconds taken by all map tasks</u>
	<u>TOTAL_LAUNCHED_UBERTASKS</u>

Map-Reduce Framework	<u>CPU time spent (ms)</u>
	<u>Failed Shuffles</u>
	<u>GC time elapsed (ms)</u>
	<u>Input split bytes</u>
	<u>Map input records</u>
	<u>Map output records</u>
	<u>Merged Map outputs</u>
	<u>Physical memory (bytes) snapshot</u>
	<u>Spilled Records</u>
	<u>Total committed heap usage (bytes)</u>
	<u>Virtual memory (bytes) snapshot</u>

JobHistory Server



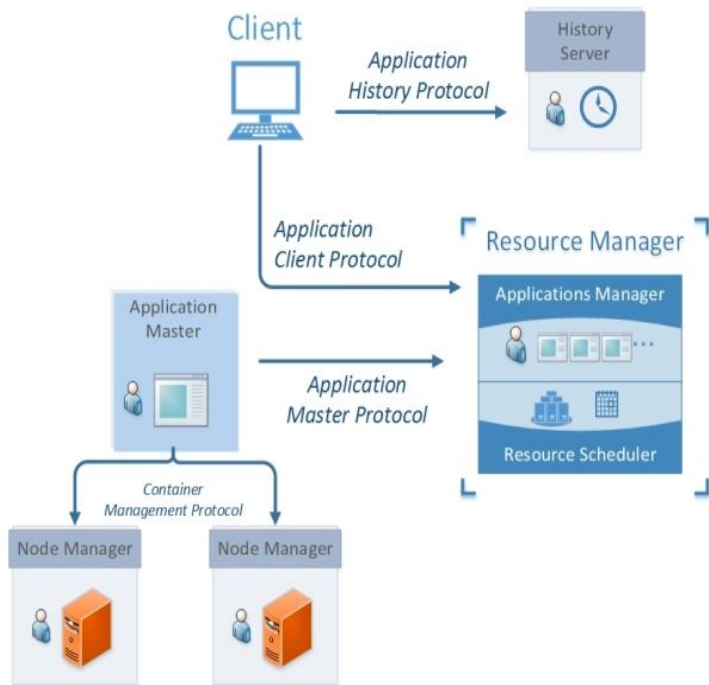
YARN User Commands

- The jar command is used to run a custom, user-built JAR file
 - `yarn jar <jar file path> [main class name] [arguments...]`
- The application command is used to manipulate a running application in YARN
 - `yarn application -list [-appStates <state identifiers> | - appTypes <type identifiers>] | -status <application id> | -kill <application id>`
- The node command is used to report the status on the nodes in a cluster
 - `yarn node -list [-all | -states <state identifiers> | -status <node id>`
- The logs command is used to dump logs of completed applications
 - `yarn logs --applicationId <application id> -appOwner <appOwner> | (-nodeAddress <node address> & -containerId <container id>)`
- Job acls can be accessed using
 - `hadoop queue -showacls`

YARN more commands

- List running applications for a given user id
 - `yarn application -list -appStates RUNNING | awk 'if($4 == "cb_ds") print $1, $2'`
- Application Status
 - `yarn application -status application_1452874451211_221763`
- Application Attempt
 - `yarn applicationattempt -list application_1452874451211_221763`
 - `yarn applicationattempt -status appattempt_1452874451211_221763_000001`
- Container
 - `yarn container -list appattempt_1452874451211_221763_000001`
 - `yarn container -status container_e03_1452874451211_221763_01_000001`
- Logs
 - `yarn logs -applicationId application_1452874451211_221763` (includes all container logs)
 - `yarn logs -applicationId application_1452874451211_221763 -container container_e03_1452874451211_221763_01_000613` (Particular container log)

YARN Protocols Summary

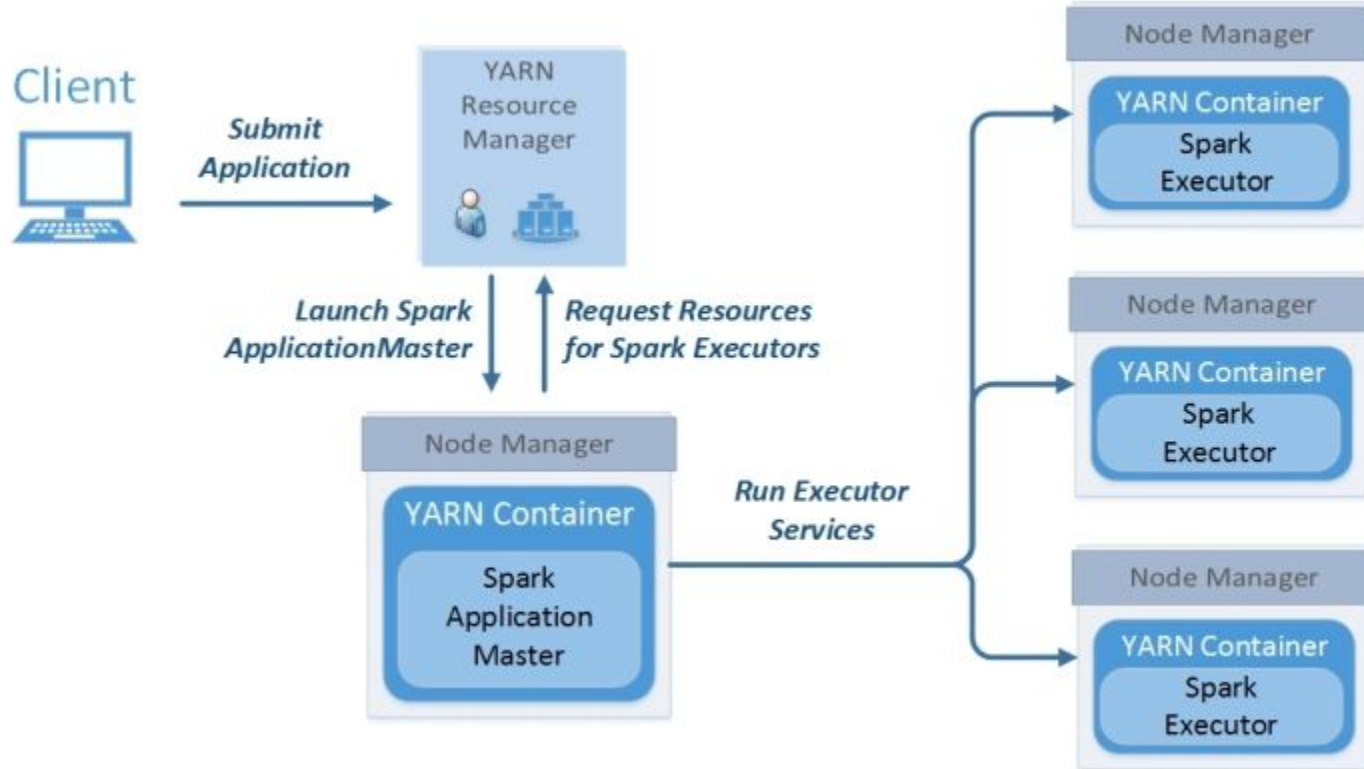


- ApplicationClientProtocol
 - YarnClient
- ApplicationMasterProtocol
 - AMRMClient, AMRMClientAsync
- ContainerManagementProtocol
 - NMClient, NMClientSync
- ApplicationHistoryProtocol
 - AHSCClient
 - Once applications are completed, client can fetch report from History Server

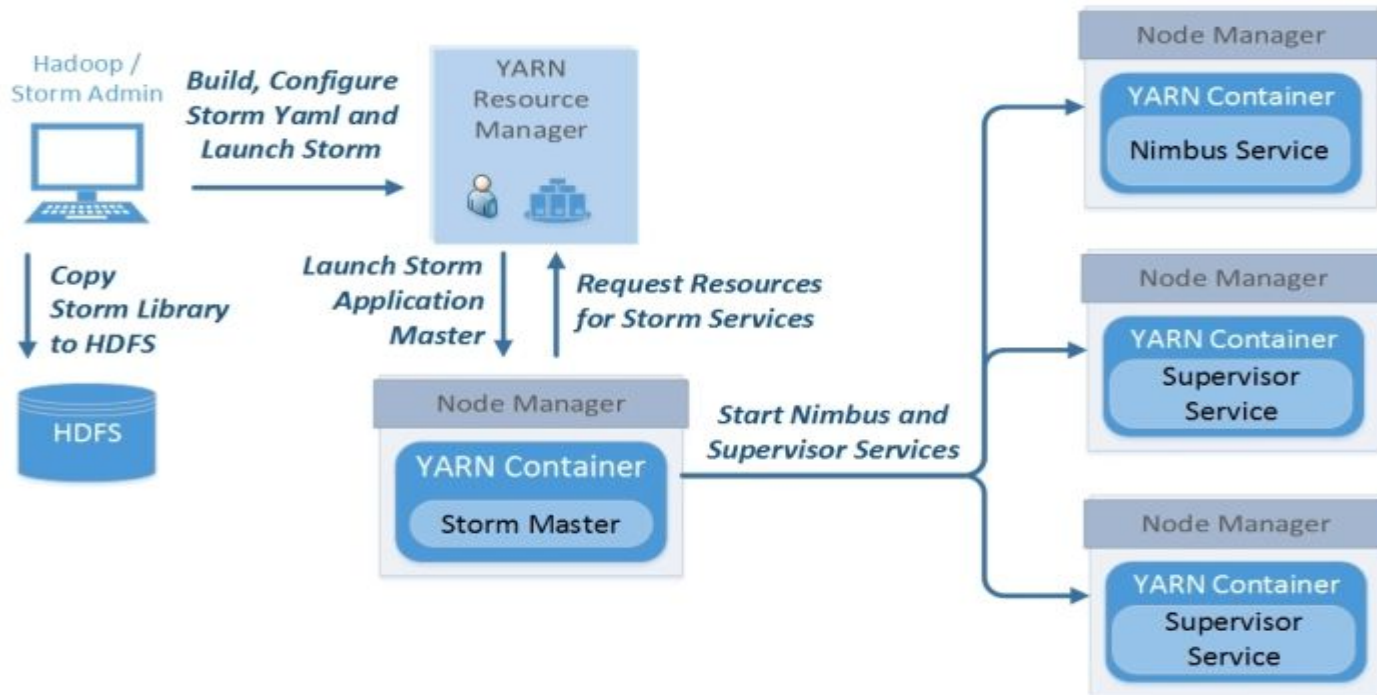
YARN Applications

- Applications powered by YARN
 - Apache Giraph - Graph processing
 - Apache Hadoop Map/Reduce - Batch Processing
 - Apache Tez - Speedy version of batch processing
 - Apache Storm - Stream Processing
 - Apache Spark - Real Time Iterative Processing
 - Many more ...

Another YARN Example: Spark on YARN



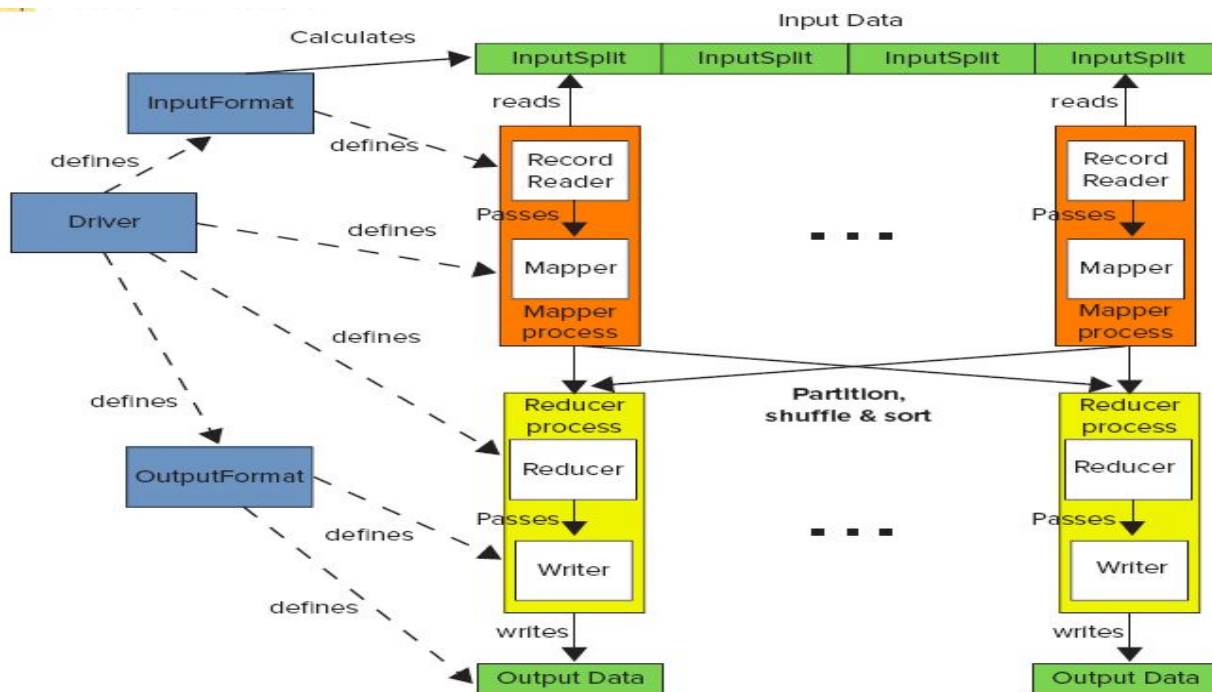
Storm on YARN



Map/Reduce Deep Dive Topics

- Input/Output Formats
 - Compressions, Sequence File Formats
- Computing Input Splits
 - Controlling the number of mappers
- Performance tuning of Mapper (spill phase)
- Application Specific Counters Framework
- How joins work
 - Reducer side join, Replicated join, Skewed join, Merge Join

InputFormat



InputFormat classes

- TextInputFormat is used to read text files line by line
- SequenceFileInputFormat is used to read binary file formats
- CombineFileInputFormat is the abstract subclass of the FileInputFormat class that can be used to combine multiple files into a single split
- DBInputFormat subclass is a class that can be used to read data from a SQL database

InputFormat Interfaces

- Driver invokes `getSplits()` on `InputFormat` to compute `InputSplits`
 - `InputSplits` information are persisted to HDFS
 - `InputSplit` info contains File path, start and end positions and Location of file
 - Application Master retrieves this file from HDFS to create resource requests based on the location of input split
- `RecordReader` - `getRecordReader()`
 - Invoked by the mapper task in the container execution on `InputFormat`
 - Returns `RecordReader` instance for the input split processed by Mapper

Computing Input Splits

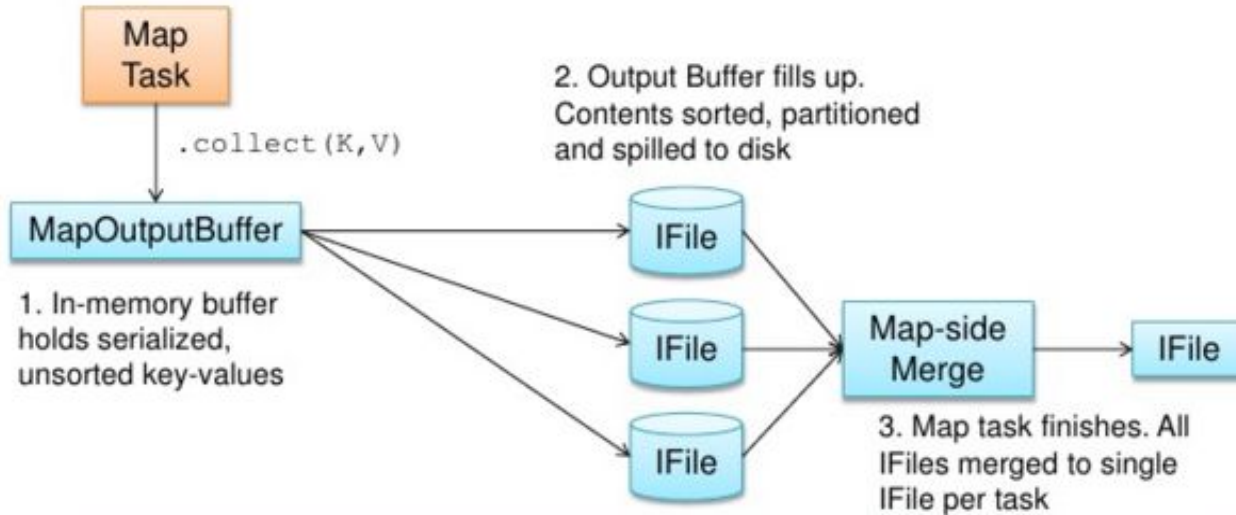
- **FileInputFormat computes input splits**
 - Split size is normally 1 DFS block size
 - Applications can impose minimum size on split block size
 - `mapreduce.input.fileinputformat.split.maxsize`
 - Setting min input split size > DFS block size will impact the data locality
- **Small files and CombinedFileFormat**
 - `CombineFileInputFormat` packs many files into each split -Each mapper has more to process
- **Preventing splitting**
 - A couple of options, set `maxsize` to MAX value
 - Customize `TextInputFormat` by overriding “issplittable” to false
- **Each mapper has access to its input split - `getInputSplit()`**
 - Contains file path, start, end byte positions and length

SequenceFileFormat

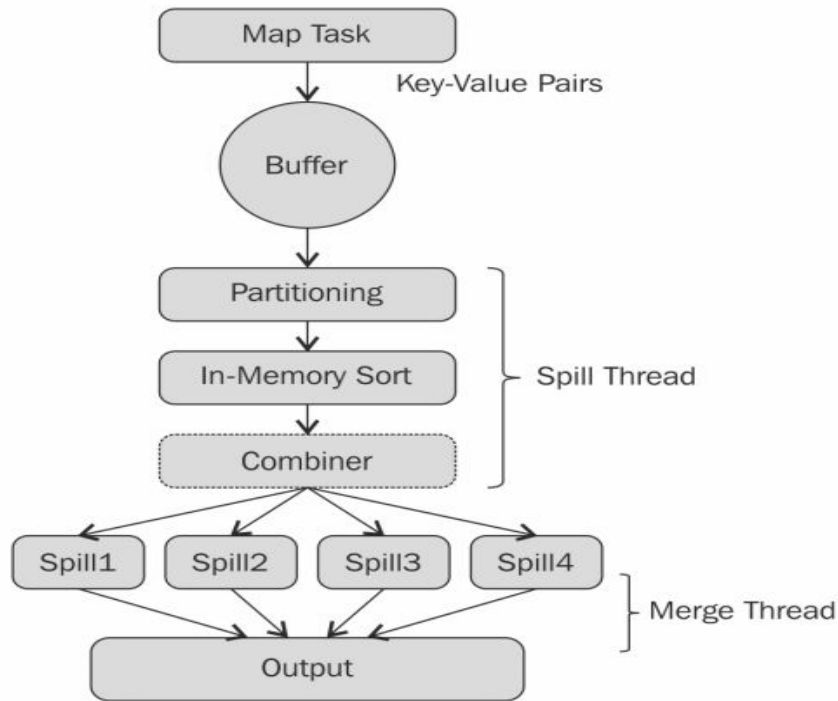
- SequenceFile is a binary format that stores binary key and value pairs
 - SequenceFileInputFormat, SequenceFileOutputFormat
- SequenceFile has header, contains the following information
 - Key and value class names
 - A Boolean value indicating whether the key and value instances are compressed.
 - A Boolean value indicating whether BLOCK compression is turned on.
 - Sync marker to denote the end of the header
 - A SequenceFile has a sync marker every few records, which enables a SequenceFile to be splittable.
- Record format
 - [record length, key length, key, value] and sync marker for every block of records
- Record Compression/Block Compression

Map Side - Sort/Spill overview

- **Goal:** when complete, map task outputs one *sorted* file
- What happens when you call `OutputCollector.collect()` ?

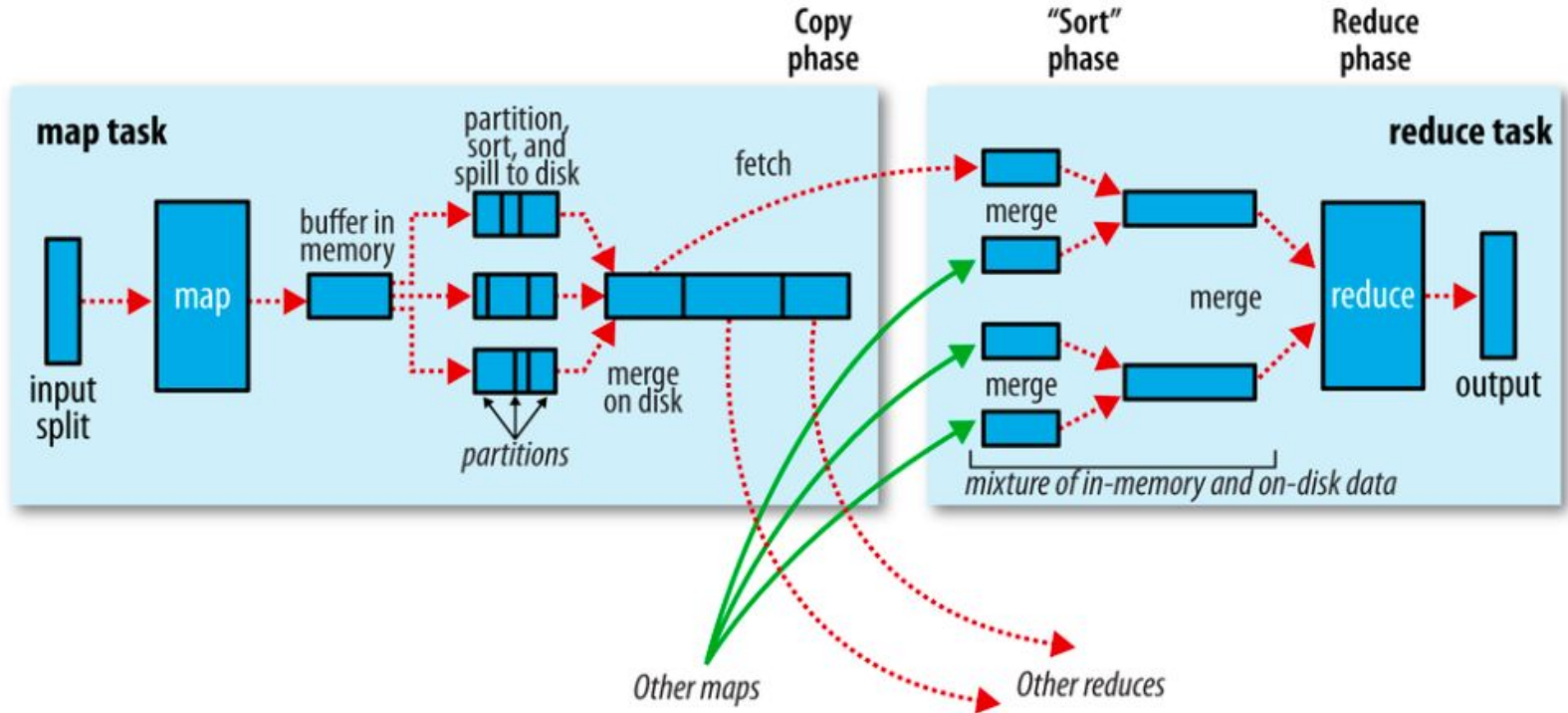


Map side task execution flow



- `mapreduce.task.io.sort.mb`
- `mapreduce.task.io.sort.spill.percent`
- `map.sort.class`
- `mapreduce.partitioner.class`
- `mapreduce.cluster.local.dir`
- `mapreduce.task.io.sort.factor` (k-way merge)
- `mapreduce.map.output.compress`
- `mapreduce.map.output.compress.codec`

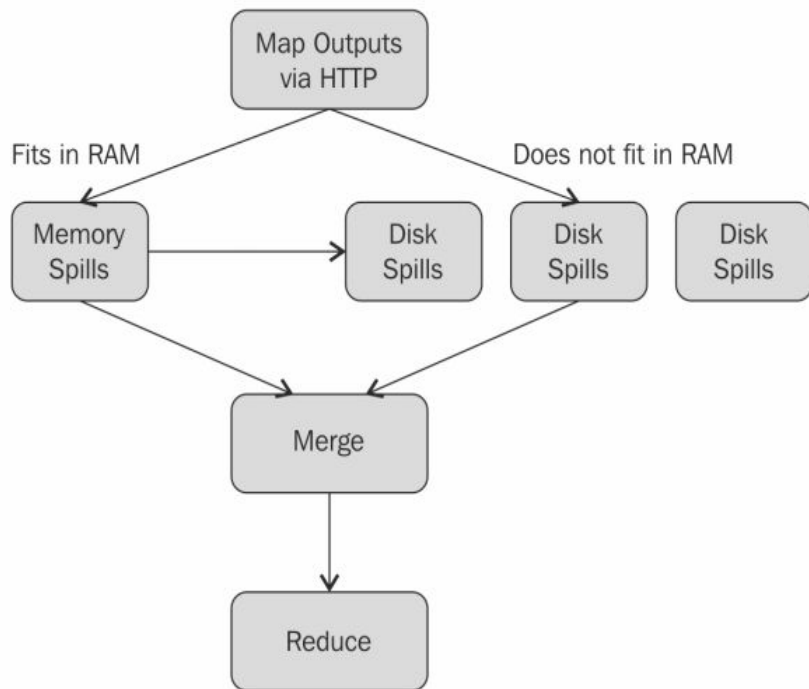
Shuffle/Sort in Map/Reduce



Mapper Spill

- Each map task has circular buffer to write mapper output
 - `mapreduce.task.io.sort.mb` - property controls buffer size
- When buffer is reached certain threshold (`mapreduce.map.sort.spill.percent`), contents are partitioned & sorted & flushed to disk
- Can also run combiner to reduce mapper output
- Each time buffer reaches spill threshold, one file is created
- Before the task is finished, the spill files are merged into a single partitioned and sorted output file
- Try to compress map output to reduce amount of data to be written to disk
(`MAPreduce.map.output.compress TO TRUE`)

Reducer Task workflow



- `mapreduce.job.reduces`
- `mapreduce.shuffle.reduce.parallelcopies`
- `mapred.reduce.copy.backoff`
- `mapreduce.reduce.shuffle.input.buffer.percent`
- `mapreduce.reduce.shuffle.merge.percent`
- `mapreduce.reduce.merge.inmem.threshold`
- `mapreduce.task.io.sort.factor`

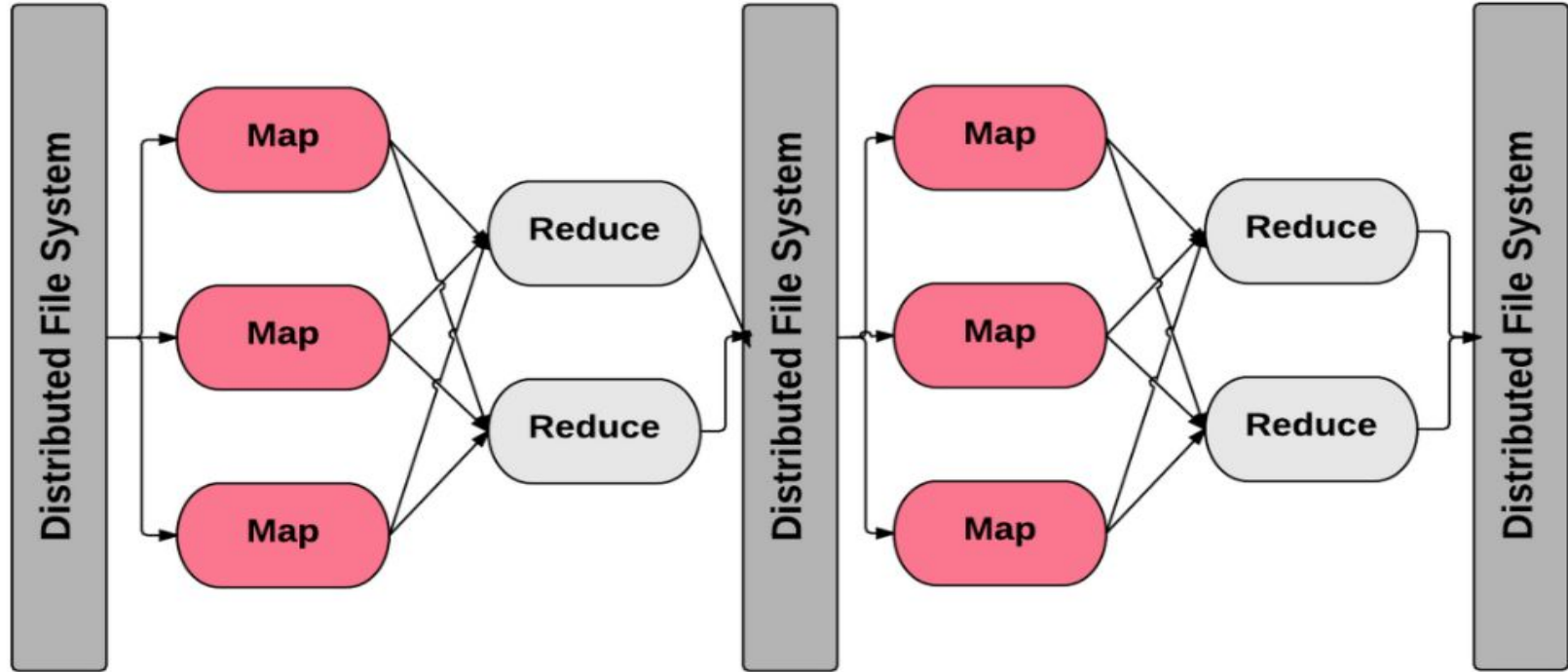
Reducer Side - Fetch/Merge

- Map Output file (partitioned & sorted) are sitting on the mapper node local disk executed mapper task
- Reduce task needs the map output for its particular partition from several map tasks across the cluster
- Multiple copier threads are configured to fetch these outputs from mappers
- Another thread starts merging map output files once they were downloaded
- Each merge round may sort k (`mapreduce.task.io.sort.factor`) map output files.

Distributed Cache

- Copies files from local FS to HDFS at job submission
- When task is launched, task copies cached resources from HDFS to local FS
 - followed by cache
 - Known as distributed cache
- Cached resources are available to reducer and mapper tasks
- Can be accessed using symbolic links in mapper and reducer code
- Can be shared across multiple instances of task on the same node
- Modify cache size using `yarn.nodemanager.localizer.cache.target-size-mb`.

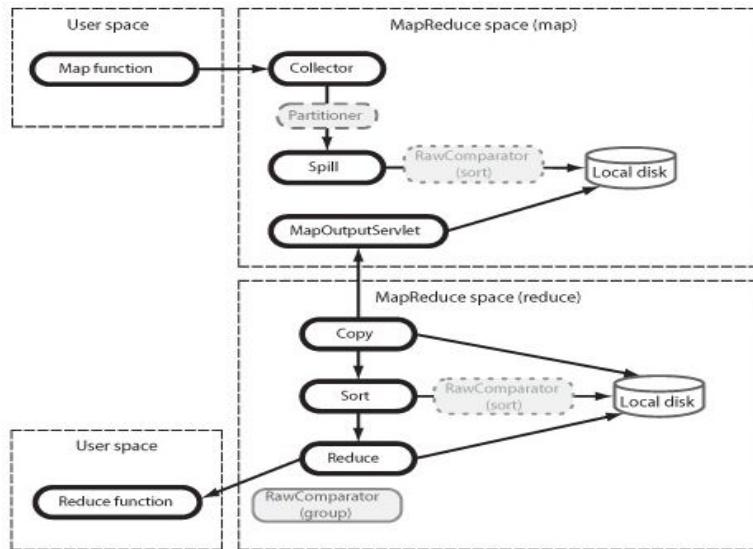
Job Chaining (aka Data Flows)



Hadoop Streaming

- Just Recap
 - MR framework is written in JAVA and exposes JAVA API to framework functions
 - A typical application composed of Job, Mapper, Reducer, Input/OutputFormat class instances and interaction among them
 - User code is compiled, jared and copied to HDFS and then a application submission request is sent to ResourceManager to launch the job via MRAppMaster.
- Hadoop comes with hadoop streaming utility which allows any executable can be used as either mapper or reducer - (stdin/stdout)
 - Mapper/reducers can also be written in scripting languages (Ex: python, perl, awk ...)
- Example: mapper and reducers are written in python
 - `$ hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar -input <input path> -output <output path> -mapper mapper.py -reducer reducer.py -file mapper.py -file reducer.py`

Sorting, Partitioning and Regrouping



The three aspects that affect sorting and grouping:

Partitioner

Determine what reducer will receive the record

RawComparator (sort)

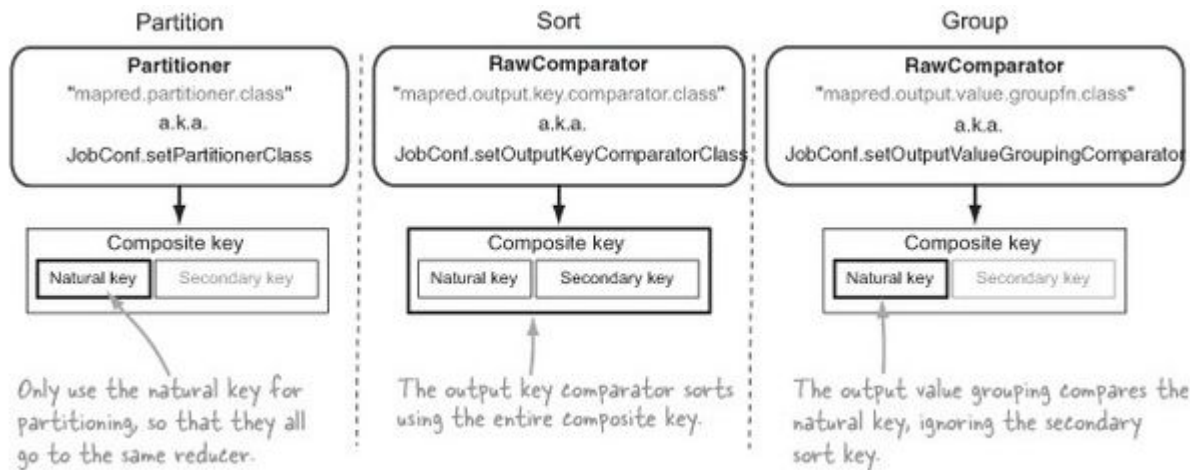
Determines how records are sorted

RawComparator (group)

Determines how sorted records are logically grouped together for a single reducer function call

Secondary Sort

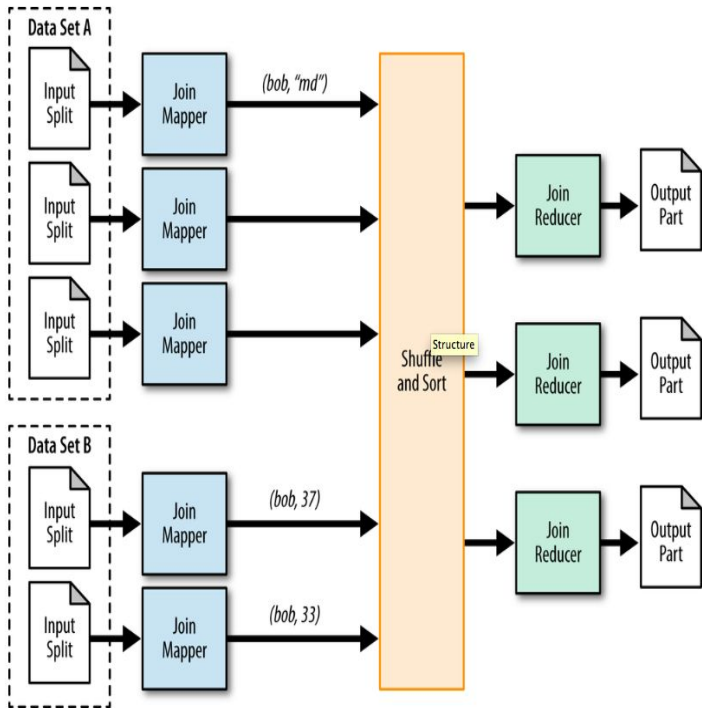
- Apply secondary sort for sorting values for a given key
 - MapReduce framework sorts the records by key, but values are not sorted
 - Make the key a composite of the natural key and the natural value.



Reduce Side Join

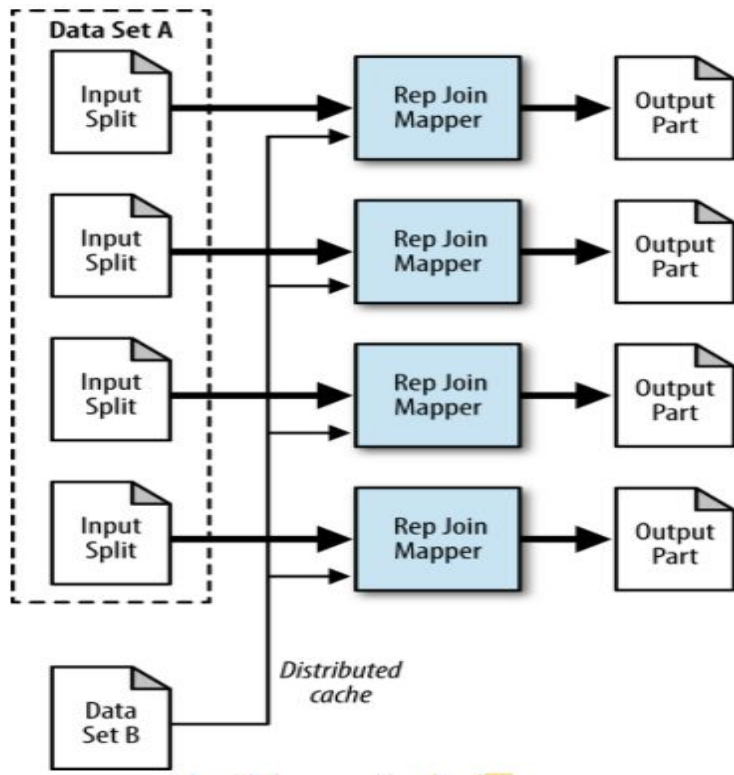
- Join large multiple data sets together by some foreign key.
 - Requires a large amount of network bandwidth
 - Bulk of data sent to reducer
- Mappers are operating on multiple data sets
 - Extract join key from each each record
 - Tag value with data source tag for that data set
 - such as A or B if two data sets are used
- Reducer performs actual join operation
 - Collect the values of each input group for a given key
 - All records flagged with A are stored in the 'A' list
 - All records flagged with B are stored in the 'B' list
 - Iterate lists A and B - Join records from both sets
 - Optimization: For a given key, Keep one tag data set in memory and stream other data set

Reduce Side Join



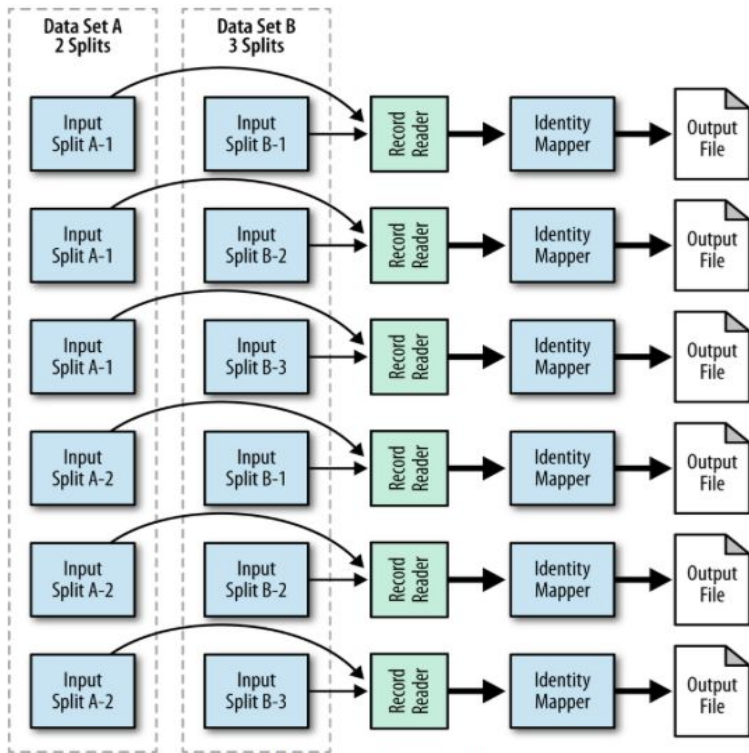
- Inner Join
 - Output record if all lists are not empty
- Outer join (Left, Right and Full)
 - Empty lists are still joined with non empty lists
- Anti Join
 - Full outer join minus the inner join.
- Use MultipleInputs to specify multiple inputs
 - `MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, DataSetAMapper.class);`
 - `MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, DataSetAMapper.class);`

Replicated Join



- Join operation between one large and many small data sets
- Can be performed on the map-side
- No need to shuffle data to the reduce phase.
- Small data set read into memory on mapper
- Restriction on the size of smaller data set
- Each mapper reads small data sets from distributed cache and stores them in memory as lookup tables during setup phase
 - `Path[] files = DistributedCache.getLocalCacheFiles(context.getConfiguration());`

Cross Product

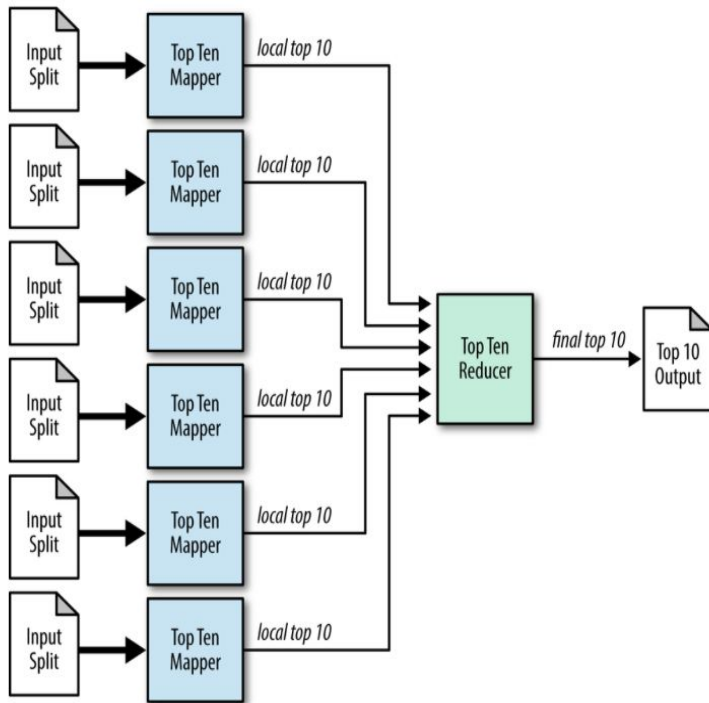


- Pair every record of multiple inputs with every other record
 - `SELECT * FROM tablea, tableb`
- The cross product of the input splits is determined during job setup and configuration.
- Each record reader is responsible for generating the cross product from both of the splits it is given.
- The record reader gives a pair of records to a mapper class, which simply writes them both out to the file system.
- No reducer phase

Counting with Counters

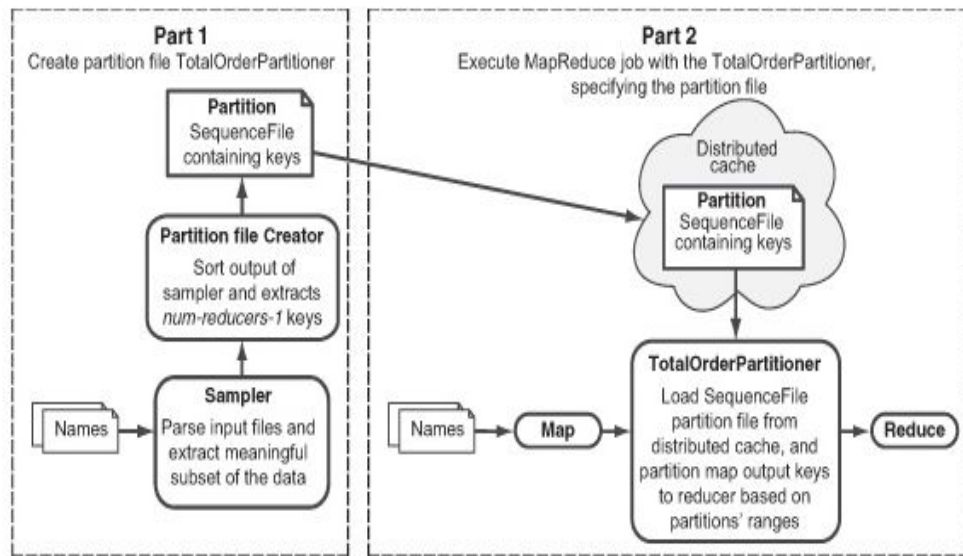
- Gather counts or summations over large data sets
 - Ex: How many records got rejected, deleted ... etc
 - Mapper updates the counter based on criteria while processing each record
 - Ex: `context.getCounter(RECORD_STATUS, status).increment(1);`
 - Counters are aggregated at Map task level and sent to MRAppMaster
 - Ex: rejected_records = 100, deleted_records = 234 etc
 - MRAppMaster aggregates counters across all mappers and reports at the completion of job
- Should be small number of counters
- A final output of counters are available in Job UI
 - After completion of job, retrieve counters
 - Ex: `job.getCounters().getGroup(RECORD_STATUS)`

Top k Records



- Retrieve top K records, according to some ranking scheme
 - `SELECT * FROM table ORDER BY col4 DESC LIMIT 10;`
- Mappers will find their local top K
 - Emit local top k records in cleanup stage
- Reducer expects $K * M$ (number of mappers) records at most
- Finally reducer will emit top k records

Total Order Sorting



- Sorting map output keys across all reducers
- Each individual reducer will sort its data by key, but unfortunately, this sorting is not global across all reducers
- Two phases
 - Create Partition File
 - Order Phase
- TotalOrderPartitioner distributes keys to specific reducers based on a partition file
- InputSampler samples records from input file

What is PIG ?

- PIG is an abstraction on top of Hadoop
- SQL like language (PIG Latin) for processing large semi-structured data sets using Hadoop map/reduce
- PIG is data flow language
 - Lets users to specify explicit sequence of steps for data processing
 - Chain together multiple map/reduce jobs
 - Supports primitive relational operators like FILTER, JOIN, GROUP, ORDER etc
- PIG can be treated as higher level programming language
 - Increases programmer productivity - Ease of programming
 - Requires less code compared to native java map/reduce jobs

PIG Components

- Pig Latin
 - Command based language
 - Designed specifically for data transformation and flow expression
- Execution environment
 - Local mode - Executes in single JVM, interacts with local file system
 - Hadoop Mode
 - Renders Pig Latin into map/reduce jobs and executes them on Hadoop Cluster
- Pig compiler
 - Converts Pig Latin commands into Map/Reduce jobs
 - Compiler strives to optimize execution

Running PIG

- Script
 - Execute commands in a file
 - `pig <script_file.pig>`
 - `pig -x local <script_file.pig> --` in local mode
- Grunt
 - Interactive shell for executing PIG commands
 - Started when script file is not provided
 - Can execute scripts from Grunt
- Embedded
 - Execute Pig commands using PigServer Class

PIG Data Types

- Pig Latin consists of the following datatypes
- Data Atom
 - Simple atomic data value (e.g. alice or 1.0)
- Tuple
 - Tuple is a row with one or more fields of any data type
 - Consists of a sequence of “fields” e.g. (alice, lakers) or (alice, kings)
- Data Bag
 - Set of Tuples equivalent to a “table” in SQL terminology
 - Can have tuples with different number of fields, even fields can have different data types - {(alice,lakers), (alice,kings)} - unlike regular relational table
- Data Map
 - A set of key value pairs [likes#(lakers,kings),age#22]

PIG Example - WordCount

```
input_lines = LOAD 'sample.txt' AS (line:chararray);

words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

filtered_words = FILTER words BY word MATCHES '\\w+';

word_groups = GROUP filtered_words BY word;

word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;

ordered_word_count = ORDER word_count BY count DESC;

STORE ordered_word_count INTO 'sample.out';
```

DUMP And STORE Statements

- No action is taken until Dump or Store statements are encountered
 - PIG will parse, validate and analyze them but not execute them
- DUMP - Displays results to the screen
- STORE - Saves results to a file

LOAD Command

- LOAD 'data' [USING function] [AS schema]
- data - name of the directory or file
- USING - specifies the load function to use
 - By default uses PigStorage which parses each line into fields using delimiter (“\t”)
 - Delimiter can be customized using regular expressions
- AS - Assign a schema to incoming data
 - Assign names to fields
 - Declares types to fields
- Example:

Schema Data Types

- Primitive Data Types
 - int, long, float, double, chararray (utf-8), bytearray (blob)
- Complex Data Types
 - Tuple - ordered set of fields - (19, "adv_1", 100, 20, \$12)
 - Bag - collection of tuples - {(19, "adv_2", 100, 20, \$12), (20, "adv_2", 120, 30, \$14)}
 - Map - collection of tuples - [open#apache]

PIG Diagnostic Tools

- Display the structure of Relation (aka Bag)
 - `grunt> DESCRIBE <relation_name>`
- Display Execution Plan
 - Produces various plans
 - Logical, physical and map/reduce plans
 - `grunt> EXPLAIN <relation_name>`
- Illustrates how PIG engine transforms the data
 - `grunt> ILLUSTRATE <relation_name>`

FOREACH Operator

- FOREACH takes a set of expressions and applies them to every record
 - PIG's projection operator
 - User can select fields to be projected for the next set of actions in pipeline
- Fields can be either referenced by field name or position
 - Avoid referring fields by position (like \$0, \$1, \$2 etc)
 - Can also reference all fields by *
 - Can also refer to ranges of fields using .. (two periods)
 - Tuple elements are referred by <tuple name>.<field name>
 - Map elements are referred by <map name>#<field name>
- UDFs (User defined functions) can be invoked in FOREACH statement

FLATTEN Operator

- Un-nests tuples and bags
- For tuples - flatten substitutes the fields of a tuple in place of the tuple
 - Example: Consider relation (a, (b, c))
 - GENERATE \$0, FLATTEN(\$1) will result in (a, b, c)
- For bags - un-nesting creates more tuples
 - Consider relation {(b,c), (d,e)}
 - GENERATE FLATTEN(\$0) will result in (b,c), (d,e)
 - Consider another relation (a, {(b,c), (d, e)})
 - GENERATE \$0, FLATTEN(\$1) will result in (a, b, c) and (a, d, e)
 - Causes a cross product to happen
 - If the bag is empty, FLATTEN will produce empty records - Surprise sideeffect

FILTER Operator

- FILTER contains a predicate. If that predicate evaluates to true for a given record, that record will be passed down the pipeline. Otherwise, it will not.
- Predicate can contain any boolean expression among fields
- Predicates can also contain UDFs
- Can test fields against nulls (like is not null)

GROUP operator

- Collects together records with the same key into a bag
 - `word_groups = GROUP filtered_words BY word;`
- Output records contains 2 fields
 - the key and the bag of collected records
 - the key field is named `group`
 - bag is named for the alias that grouped (`filtered_words`), has the same schema as alias
 - `{group: chararray, filtered_words: {word: chararray}}`
- Requires reduce phase
 - Number of reducers can be controlled by using `PARALLEL <number of reducers>`
 - `word_groups = GROUP filtered_words BY word PARALLEL 10;`
- Aggregate functions can be applied on BAG columns

ORDER BY, DISTINCT

- Sorts data by key (ASC or DESC)
- Produces total order for the data
 - Data is sorted in each partition
 - All records in partition n are also less than all records in partition $n-1$ for all n
 - Uses custom partition file using key range
 - Pig does sampling of input records to get key distribution
 - Pig distributes to key in such a way that it reduces skewness at a given reducer
- Distinct removes duplicate records
 - Forces reducer phase
 - Apply combiner in map phase to reduce any duplicate records

JOIN operator

- Join selects records from one input to put together with records from another input - Just like any SQL statement
- Supports LEFT, RIGHT Outer Joins and FULL joins
- Can specify number of reducers using parallel construct
- Different types of join
 - Reduce side join
 - Replicated join (using 'replicated') - Joining large data set small data set
 - Skewed join (using 'skewed') - Joining skewed data
 - merge join (using 'merge') - Joining already sorted data
- More than 2 data sets can be joined once

```
A = load 'input1' as (x, y);  
B = load 'input2' as (u, v);  
C = load 'input3' as (e, f);  
alpha = join A by x, B by u, C by e;
```


COGROUP

- Groups rows based on a column, and creates bags for each group.
 - Exactly same as GROUP
- COGROUP can be applied on two tables (or more) on a join column
 - Result contains group key and one bag for each table records marching key
 - {group: key, {(records from data set A)}, {(records from data set B)}}
 - Join is a special case of COGROUP
 - JOIN is cross product of records from data set A and records from data set B

NESTED FOREACH

- Order, limit, and transform data within a GROUP BY expression
- Each statement inside the FOREACH is generated separately for each group

```
my_data = FOREACH (group data BY field) {  
    ordered = ORDER data by field2 ASC;  
    limited = LIMIT ordered 4;  
    GENERATE flatten(limited);  
}
```

In-Line GROUP BY and JOIN

- No need to write two separate statements
 - GROUP followed by FOREACH
 - JOIN followed by FOREACH
- Let's write them in-line

```
grouped = FOREACH (group data BY field1)  
  GENERATE group, SUM(data.field2);
```

```
joined = FOREACH (JOIN alias1 BY field, alias2 by field)  
  GENERATE alias1::field, field2;
```

Cast Relation As A scalar

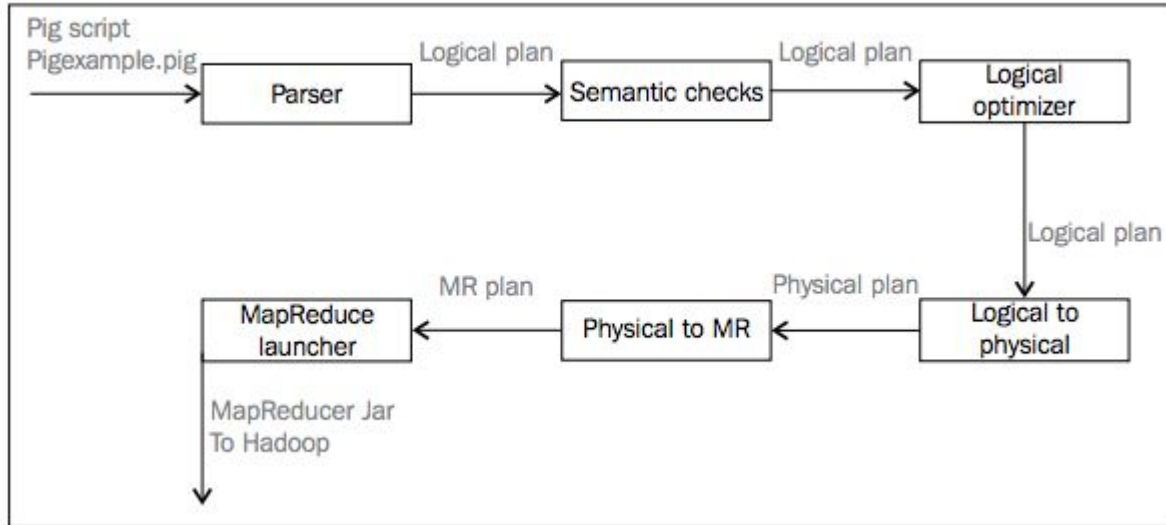
- Cast any field of a single-tuple relation as a scalar, and use it without joining to the relation
- Let's say computed hits on a given site
 - `total_hits = FOREACH (GROUP data ALL)`
`GENERATE SUM(data.clicks) as total;`
- Now compute each user's percentage of the traffic
 - Doing a relation-to-scalar cast avoids having to do a CROSS
 - `user_hits = FOREACH data`
`GENERATE user_id, clicks/total_hits.total;`

MACROS in PIG

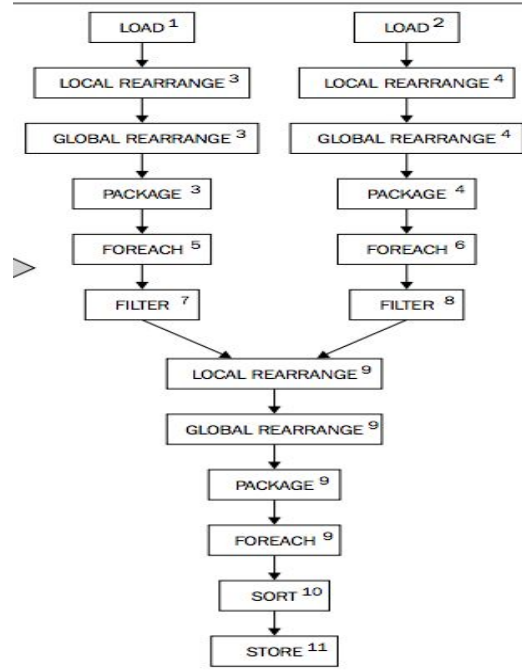
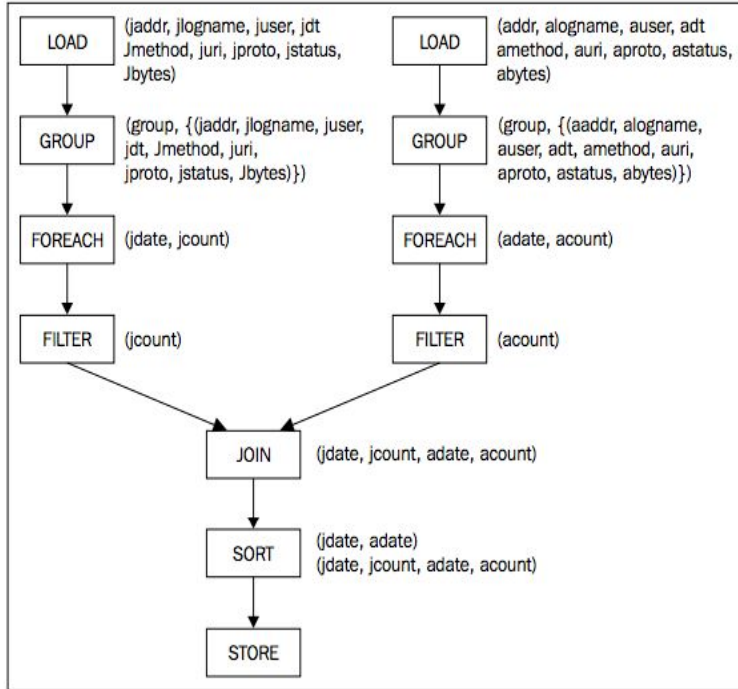
- Pig's mechanism of code reuse
 - Define a “function” by writing a macro
 - Reuse that macro either within the same script or in different scripts.
 - You can also put these macros into a separate file and include them
 - include 'CBCoreEventsMacros.pig';

```
define function_1(int_value, alias) returns new_alias {  
    $new_alias = foreach $alias generate field * int_value;  
};
```

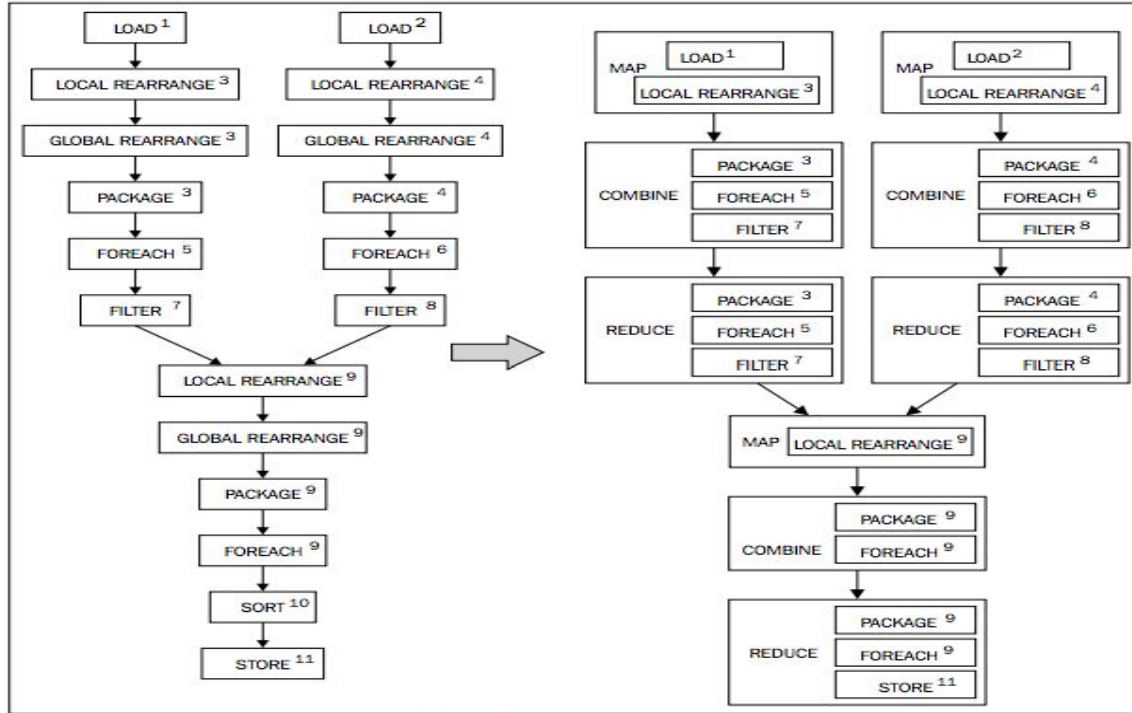
PIG Architecture



Logical Plan ==> Physical Plan



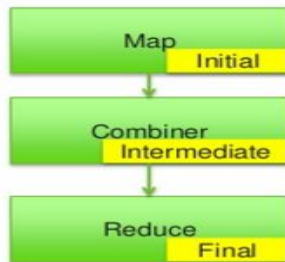
Physical Plan To MapReduce Plan



User Defined Functions in PIG

- EvalFunc<T>
 - public <T> exec(Tuple input)
- FilterFunc
 - public Boolean exec(Tuple input)
- Aggregate Functions

```
public interface Algebraic {  
    public string getInitial();  
  
    public string getIntermed();  
  
    public string getFinal();  
};
```



```
A = load 'input';  
B0 = group A by $0;  
C0 = foreach B0 generate group, SUM(A);  
Store C0 into 'output0';
```

- Load/Store Functions
 - public Tuple getNext()
 - public void putNext(Tuple input)

Accumulator

```
Visits = LOAD 'visits';  
UserVisits = GROUP Visits BY user;  
Sessions = FOREACH UserVisits {  
    OrderedVisits = ORDER UserVisits BY timestamp;  
    GENERATE user, FLATTEN(sessionize(OrderedVisits));  
}
```

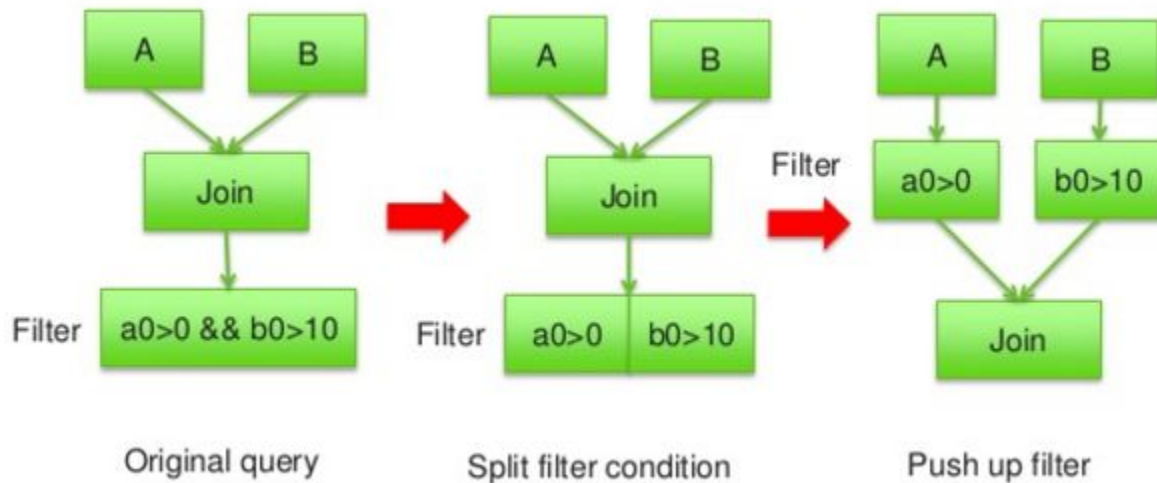
- All tuples belong to same key must be materialized into a databag
 - before passing databag to UDF
 - If key contains a large number of records - can cause memory problems
 - Many UDFs do not really need to see all records at one time
 - Pass records in batches - Examples: COUNT(), SUM()
 - Tuples can be passed to UDFs in accumulative manner
 - When all tuples are passed, the final method is called to retrieve value

Accumulator Interface

```
public interface Accumulator <T> {  
    /**  
     * Pass tuples to the UDF. You can retrieve DataBag by calling b.get(index).  
     * Each DataBag may contain 0 to many tuples for current key  
     */  
    public void accumulate(Tuple b) throws IOException;  
  
    /**  
     * Called when all tuples from current key have been passed to accumulate.  
     * @return the value for the UDF for this key.  
     */  
    public T getValue();  
  
    /**  
     * Called after getValue() to prepare processing for next key.  
     */  
    public void cleanup();  
  
}
```

Push up Filter

- Pig split the filter condition before push



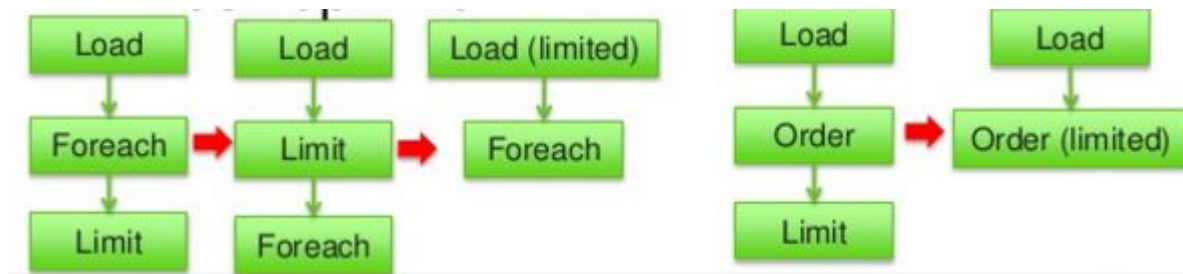
Push Up/Down Optimizations in PIG

- Push Down Flatten

```
A = load 'input' as (a0:bag, a1);  
B = foreach A generate  
  flatten(a0), a1;  
C = order B by a1;  
Store C into 'output';
```



- Push Up Limit



CUBE Operator

```
rawdata = load 'input' as (ptype, pstore, number);  
cubed = cube rawdata by rollup(ptype, pstore);  
result = foreach cubed generate flatten(group), SUM(cube.number);  
dump result;
```

Ptype	Pstore	number
Dog	Miami	12
Cat	Miami	18
Turtle	Tampa	4
Dog	Tampa	14
Cat	Naples	9
Dog	Naples	5
Turtle	Naples	1



Ptype	Pstore	Sum
Cat	Miami	18
Cat	Naples	9
Cat		27
Dog	Miami	12
Dog	Tampa	14
Dog	Naples	5
Dog		31
Turtle	Tampa	4
Turtle	Naples	1
Turtle		5
		63

Rank Operator

```
rawdata = load 'input' as (name, gpa:double);  
ranked = rank rawdata by gpa;  
dump ranked;
```

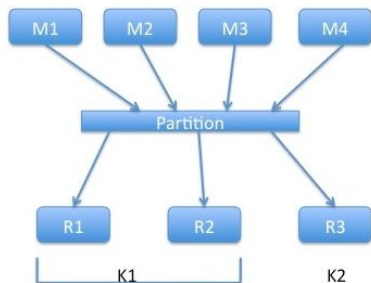
Name	Gpa
Katie	3.5
Fred	4.0
Holly	3.7
Luke	3.5
Nick	3.7



Rank	Name	Gpa
1	Katie	3.5
5	Fred	4.0
2	Holly	3.7
1	Luke	3.5
2	Nick	3.7

Skewed Join in PIG

- Control over the allocation of reducers to counteract the skew
 - `C = JOIN big BY b1, massive BY m1 USING "skewed";`
- Translates into two map/reduce jobs - Sample and join
 - Sample computes histogram of underlying key space (pig.keydist file)
 - Second job performs a join on predicate
 - One of the table is partitioned and the other is streamed to reducer
 - For streamed table, mapper task copies the data to each of reducer partitions using pig.keydist file



- For partitioned table, partitioner uses key distribution (pig.keydist) to send the data to reducer in a round robin fashion

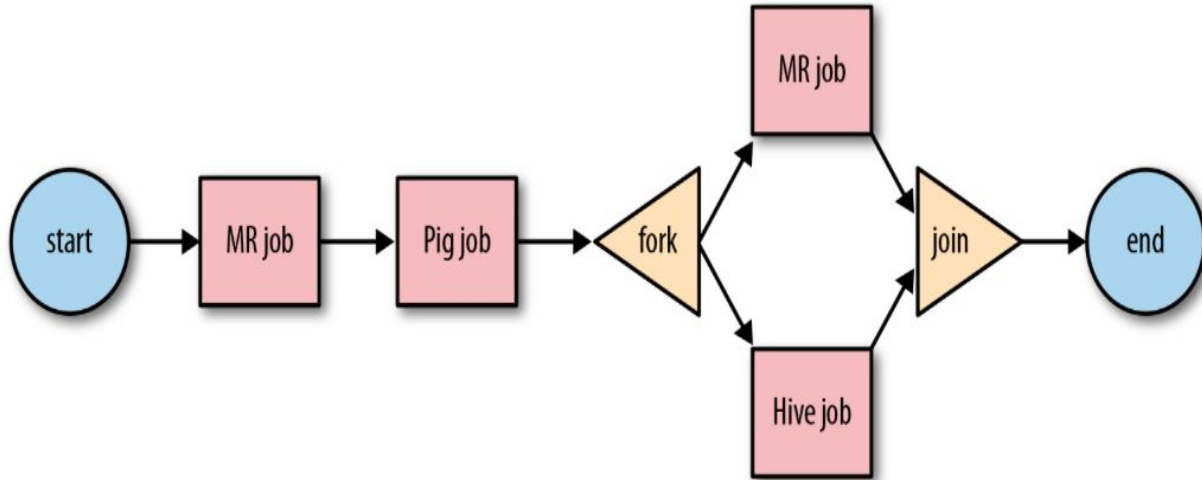
Oozie

- Workflow scheduler to run hadoop jobs
 - Workflow jobs are DAG (directed acyclic graph) of actions
 - The output of one action can be consumed by the next action to create a chain sequence
- Oozie concepts
 - Workflows
 - Collection of actions arranged in a required dependency graph
 - Actions can be various types: Pig action, MapReduce Action, Hive action, fs action etc
 - Coordinator
 - Coordinator tells Oozie when to do a task
 - Example: At specified time or when some input data set is available etc
- Workflows, coordinator files are specified in xml

Workflows

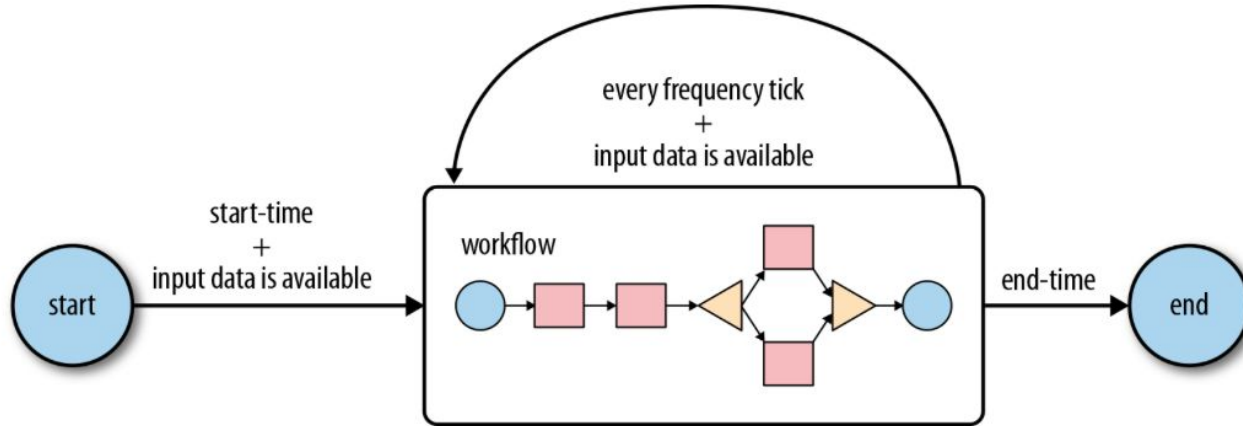
- Workflow is composed of nodes
 - Each node does some specified work
 - On success moves to another node (Ex: OK node), failure moves to another node (Kill node)
 - There are two types of nodes
- Control Flow Nodes
 - Start, End, Kill, Decision (switch, if else conditions), Fork/Join nodes
- Action Nodes
 - Action nodes represent the actual processing tasks that are executed when called
 - Examples: PIG Action, HIVE action, MAPREDUCE Action, FileSystem Action

Oozie Workflow example



Oozie Coordinator


- Schedules workflow executions based on a start-time and a frequency
- Starts the workflow when all the necessary input data becomes available



Oozie Web Console


PB: <http://phazonblue-oozie.blue.ygrid.yahoo.com:4080/oozie/>

PR: <http://phazonred-oozie.red.ygrid.yahoo.com:4080/oozie/>

 [Apache Documentation](#) [Yahoo Documentation](#)

Oozie Web Console (v2) [/oozie/]

[Workflow Jobs](#) **[Coordinator Jobs](#)** [Bundle Jobs](#) [SLA](#) [System Info](#) [Instrumentation](#) [Settings](#)

 [All Jobs](#) **[Active Jobs](#)** [Done Jobs](#) [Custom Filter](#) ▼

	Job Id	Name	Status	User ▲	Group	Frequency	Unit	Started	Next Materialization
1	2061163-151210191710682-oozie_PB-C	aggregator-coord	RUNNING	aishp	users	1440	MINUTE	Wed, 13 Jan 2016 00:00:00 G...	Mon, 25 Jan 2016 00:00:00 G...
2	2034148-151210191710682-oozie_PB-C	cb_ds_extracts_bootstrap_c	RUNNING	cb_ds	users	720	MINUTE	Wed, 13 Jan 2016 07:18:00 G...	Sun, 24 Jan 2016 19:18:00 GMT
3	2028792-151210191354292-oozie_PB-C	cb_ds_extracts_delta_c	RUNNING...	cb_ds	users	15	MINUTE	Wed, 13 Jan 2016 00:58:00 G...	Sun, 24 Jan 2016 18:13:00 GMT
4	2028800-151210191354292-oozie_PB-C	cb_ds_extracts_gpa_delta_c	RUNNING	cb_ds	users	180	MINUTE	Tue, 12 Jan 2016 00:00:00 GMT	Sun, 24 Jan 2016 18:00:00 GMT
5	2028796-151210191354292-oozie_PB-C	cb_ds_extracts_negkeywords_c	RUNNING	cb_ds	users	240	MINUTE	Wed, 13 Jan 2016 00:58:00 G...	Sun, 24 Jan 2016 20:58:00 GMT
6	2028798-151210191354292-oozie_PB-C	cb_ds_extracts_syndication_c	RUNNING	cb_ds	users	1440	MINUTE	Wed, 13 Jan 2016 00:58:00 G...	Mon, 25 Jan 2016 00:58:00 G...
7	2028802-151210191354292-oozie_PB-C	cb_ds_extracts_termscore_c	RUNNING	cb_ds	users	1	DAY	Tue, 12 Jan 2016 00:00:00 GMT	Mon, 25 Jan 2016 00:00:00 G...
8	2034152-151210191710682-oozie_PB-C	cb_ds_extracts_variant_bootstrap_c	RUNNING	cb_ds	users	720	MINUTE	Wed, 13 Jan 2016 14:18:00 G...	Mon, 25 Jan 2016 02:18:00 G...
9	2034144-151210191710682-oozie_PB-C	cb_ds_extracts_variant_delta_c	RUNNING	cb_ds	users	15	MINUTE	Wed, 13 Jan 2016 07:17:00 G...	Sun, 24 Jan 2016 18:32:00 GMT

Oozie Coordinator

Job (Name: cb_ds_extracts_bootstrap_c/coordJobId: 2034148-151210191710682-oozie_PB-C)

Coord Job Info

Coord Job Definition


Coord Job Configuration

Coord Job Log

Coord Error Log

Coord Audit Log

Coord Action Reruns



Job Id: 2034148-151210191710682-oozie_PB-C

Name: cb_ds_extracts_bootstrap_c

Status: RUNNING

User: cb_ds

Group: users

Frequency: 720

Unit: MINUTE

Parent Bundle:

Start Time: Wed, 13 Jan 2016 07:18:00 GMT

Next Matd: Sun, 24 Jan 2016 19:18:00 GMT

End Time: Sat, 08 Apr 2017 06:15:00 GMT

Pause Time:

Concurrency: 1

Actions

	Action Id	Status	Ext Id	Error Code	Created Time	Nominal Time	Las
1	2034148-151210191710682-oozie_PB-C@23	SUCCEEDED	2242936-151210191354292-oozie_PB-W		Sun, 24 Jan 2016 07:12:31 GMT	Sun, 24 Jan 2016 07:18:00 GMT	Sur
2	2034148-151210191710682-oozie_PB-C@22	SUCCEEDED	2233615-151210191354292-oozie_PB-W		Sat, 23 Jan 2016 19:12:24 GMT	Sat, 23 Jan 2016 19:18:00 GMT	Sur
3	2034148-151210191710682-oozie_PB-C@21	SUCCEEDED	2224262-151210191354292-oozie_PB-W		Sat, 23 Jan 2016 07:12:17 GMT	Sat, 23 Jan 2016 07:18:00 GMT	Sat
4	2034148-151210191710682-oozie_PB-C@20	SUCCEEDED	2214662-151210191354292-oozie_PB-W		Fri, 22 Jan 2016 19:12:04 GMT	Fri, 22 Jan 2016 19:18:00 GMT	Sat
5	2034148-151210191710682-oozie_PB-C@19	SUCCEEDED	2205174-151210191710682-oozie_PB-W		Fri, 22 Jan 2016 07:11:58 GMT	Fri, 22 Jan 2016 07:18:00 GMT	Fri,

Workflow Nodes

- Extracts (Bootstrap) - Workflow
 - Sequence of nodes (Control Nodes and Action Nodes connected together)

Actions							
	Action Id	Name	Type	Status	Transition	StartTime	EndTime
1	2242936-151210191354292-oozie_PB-W@:start:	:start:	:START:	OK	progress	Sun, 24 Jan 2016 07:18:00 GMT	Sun, 24 Jan 2016 07:18:00 GMT
2	2242936-151210191354292-oozie_PB-W@progress	progress	fs	OK	deltaPrep	Sun, 24 Jan 2016 07:18:00 GMT	Sun, 24 Jan 2016 07:18:00 GMT
3	2242936-151210191354292-oozie_PB-W@deltaPrep	deltaPrep	java	OK	checkUpdate	Sun, 24 Jan 2016 07:18:01 GMT	Sun, 24 Jan 2016 07:18:21 GMT
4	2242936-151210191354292-oozie_PB-W@checkUpdate	checkUpdate	switch	OK	bootstrapEx...	Sun, 24 Jan 2016 07:18:21 GMT	Sun, 24 Jan 2016 07:18:21 GMT
5	2242936-151210191354292-oozie_PB-W@bootstrapExt...	bootstrapExtractionFork	:FORK:	OK	*	Sun, 24 Jan 2016 07:18:21 GMT	Sun, 24 Jan 2016 07:18:21 GMT
6	2242936-151210191354292-oozie_PB-W@advertiserAn...	advertiserAndQueryJob	pig	OK	bootstrapEx...	Sun, 24 Jan 2016 07:18:21 GMT	Sun, 24 Jan 2016 12:07:09 GMT
7	2242936-151210191354292-oozie_PB-W@adExtension...	adExtensionJob	pig	OK	bootstrapEx...	Sun, 24 Jan 2016 07:18:24 GMT	Sun, 24 Jan 2016 07:33:20 GMT
8	2242936-151210191354292-oozie_PB-W@bootstrapExt...	bootstrapExtractionJoin	:JOIN:	OK	stats	Sun, 24 Jan 2016 12:07:09 GMT	Sun, 24 Jan 2016 12:07:09 GMT
9	2242936-151210191354292-oozie_PB-W@stats	stats	java	OK	validateBoo...	Sun, 24 Jan 2016 12:07:09 GMT	Sun, 24 Jan 2016 12:07:36 GMT
10	2242936-151210191354292-oozie_PB-W@validateBoot...	validateBootstrap	shell	OK	bootstrapC...	Sun, 24 Jan 2016 12:07:36 GMT	Sun, 24 Jan 2016 12:07:53 GMT
11	2242936-151210191354292-oozie_PB-W@bootstrapCo...	bootstrapCommitFork	:FORK:	OK	*	Sun, 24 Jan 2016 12:07:53 GMT	Sun, 24 Jan 2016 12:07:53 GMT
12	2242936-151210191354292-oozie_PB-W@forexCommit...	forexCommitJob	java	OK	bootstrapC...	Sun, 24 Jan 2016 12:07:53 GMT	Sun, 24 Jan 2016 12:08:17 GMT
13	2242936-151210191354292-oozie_PB-W@advertiserCo...	advertiserCommitJob	java	OK	bootstrapC...	Sun, 24 Jan 2016 12:07:56 GMT	Sun, 24 Jan 2016 12:08:36 GMT
14	2242936-151210191354292-oozie_PB-W@adGroupCo...	adGroupCommitJob	java	OK	bootstrapC...	Sun, 24 Jan 2016 12:07:59 GMT	Sun, 24 Jan 2016 12:09:03 GMT
15	2242936-151210191354292-oozie_PB-W@campaignCo...	campaignCommitJob	java	OK	bootstrapC...	Sun, 24 Jan 2016 12:08:01 GMT	Sun, 24 Jan 2016 12:08:46 GMT
16	2242936-151210191354292-oozie_PB-W@termCommit...	termCommitJob	java	OK	bootstrapC...	Sun, 24 Jan 2016 12:08:05 GMT	Sun, 24 Jan 2016 12:09:33 GMT

Oozie parameters and config files

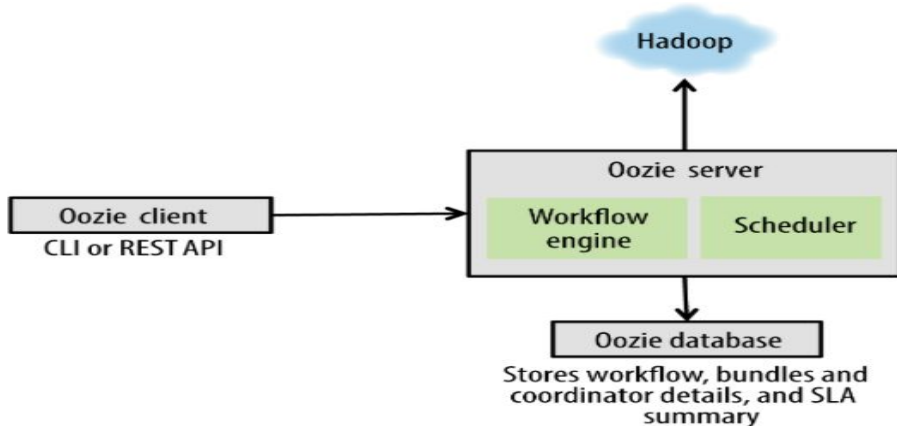
- Parameters are used across coordinators and workflows
- Parameters are specified in a text file (aka job.properties) and passed as input parameter to oozie job submission command
- Application deployment model
 - Workflow application consists of a workflow.xml file
 - Coordinator applications consist of a coordinator.xml file
 - Properties file
 - All actions - Pig scripts, HIVE query files, JAR files and any other dependency libs etc

Oozie Commands

- Getting Job Info
 - `oozie job -info <job id>`
- Running a job
 - `oozie job -run -config <job properties file>`
- Killing a job
 - `oozie job -kill <job id>`

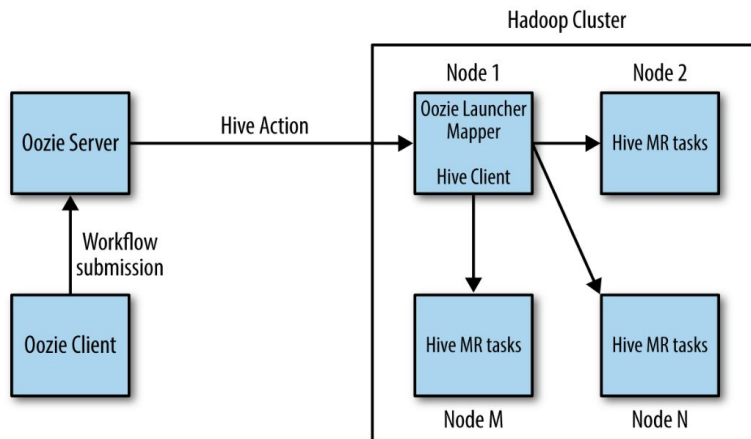
Oozie Architecture

- Oozie server is java application running in Java servlet container (Tomcat)
 - Clients interact with it through CLI, Java Client API, REST API
 - Oozie server is stateless application
 - All information about jobs is stored in mysql database (or some other DB)
 - Oozie needs all application files available on HDFS
 - submit oozie job to the server (through command line)



Execution Model in Oozie

- How does user run PIG/HIVE/MapReduce job?
 - Runs on a gateway node (outside of hadoop cluster but talks to hadoop cluster)
 - JobClient (on local node (gateway)) is actually submitting job to Resource Manager in cluster
- Oozie runs actual actions through a launcher job
 - Launcher job is MapReduce job runs on Hadoop cluster
 - The launcher is a map-only job that runs only one mapper



Sample PIG action in Oozie workflow

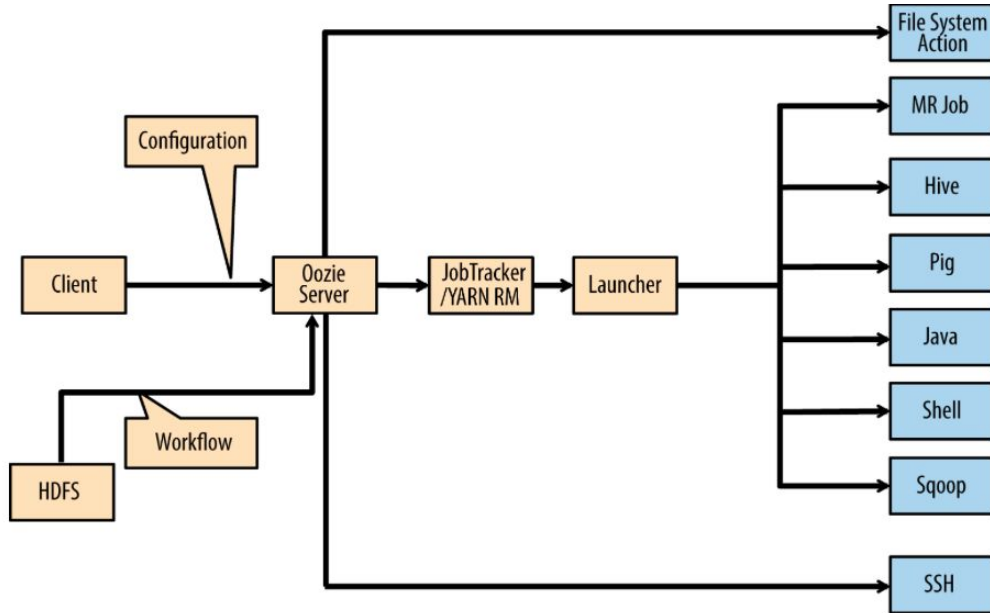
```
<action>
<action name="adExtensionJob" cred="hbase.cert">
  <pig>
    <prepare>
      <delete path="${outRoot}/aeds_tmp"/>
    </prepare>
    <configuration>
      <property>
        <name>mapred.child.java.opts</name>
        <value>-server -XX:+UseParallelGC -XX:ParallelGCThreads=4 -Djava.io.tmpdir=./tmp</value>
      </property>
    </configuration>
    <script>${wfAppPigPath}/BootstrapAdExtension.pig</script>
    <param>AdvertiserTable=${advertiser_table}</param>
    <param>AdExtensionTable=${adextension_table}</param>
    <param>AdditionalActiveAdvertisersPath=${advertiser_whitelist_input}</param>
    <param>MIN_HMM=${wf:actionData('deltaPrep')['minHMM']}</param>
    <param>MAX_HMM=${wf:actionData('deltaPrep')['maxHMM']}</param>
    <param>namespace=${outRoot}</param>
    <param>version=${wf:actionData('deltaPrep')['version']}d</param>
    <file>${wfHBaseConfPath}/hbase-site.xml#hbase-site.xml</file>
    <file>${wfAppPigPath}/CommonMacros.pig#CommonMacros.pig</file>
    <file>${wfAppPigPath}/CommonDefines.pig#CommonDefines.pig</file>
    <file>${wfAppPigPath}/AdvertiserDataMacros.pig#AdvertiserDataMacros.pig</file>
    <file>${wfAppPigPath}/AdvertiserDataMacrosExt.pig#AdvertiserDataMacrosExt.pig</file>
  </pig>
  <ok to="bootstrapExtractionJoin" />
  <error to="fail" />
</action>
```

Workflow Language

Flow-control node	XML element type	Description
Decision	workflow:DECISION	expressing "switch-case" logic
Fork	workflow:FORK	splits one path of execution into multiple concurrent paths
Join	workflow:JOIN	waits until every concurrent execution path of a previous fork node arrives to it
Kill	workflow:kill	forces a workflow job to kill (abort) itself

Action node	XML element type	Description
java	workflow:JAVA	invokes the main() method from the specified java class
fs	workflow:FS	manipulate files and directories in HDFS; supports commands: move, delete, mkdir
MapReduce	workflow:MAP-REDUCE	starts a Hadoop map/reduce job; that could be java MR job, streaming job or pipe job
Pig	workflow:pig	runs a Pig job
Sub workflow	workflow:SUB-WORKFLOW	runs a child workflow job
Hive *	workflow:HIVE	runs a Hive job
Shell *	workflow:SHELL	runs a Shell command
ssh *	workflow:SSH	starts a shell command on a remote machine as a remote secure shell
Sqoop *	workflow:SQOOP	runs a Sqoop job
Email *	workflow:EMAIL	sending emails from Oozie workflow application
Distcp ?		Under development (Yahoo)

Workflow execution



Coordinator

- Trigger based workflow executions
 - Time Trigger (Just like unix cron) - workflow is executed at fixed intervals or frequency
 - Start Time, End Time and Frequency
- Data Availability Trigger
- Coordinator application xml is template to capture triggers
 - Triggers
 - A reference to workflow to be launched
 - Workflow execution parameters
- Coordinator creates (materialize) a coordinator action for a specific time instance (nominal time)

Expressing Data Dependencies

- Dataset

- A dataset is a logical entity to represent a set of data produced by an application
- User can define a dataset either using its directory location or using metadata

```
<!-- datasets -->
<datasets>
  <dataset name="HDFS_COW_TARGETING_ATTRIBUTE" frequency="${coord:days(1)}" initial-instance="2013-01-01T00:00Z" timezone="${timezone}">
    <uri-template>${cow_location}/complete_snapshot/${YEAR}${MONTH}${DAY}0000/targetingAttribute</uri-template>
    <done-flag></done-flag>
  </dataset>
  <dataset name="HDFS_COW_ADGROUP" frequency="${coord:days(1)}" initial-instance="2013-01-01T00:00Z" timezone="${timezone}">
    <uri-template>${cow_location}/complete_snapshot/${YEAR}${MONTH}${DAY}0000/adGroup</uri-template>
    <done-flag></done-flag>
  </dataset>
</datasets>
```

- Input Events

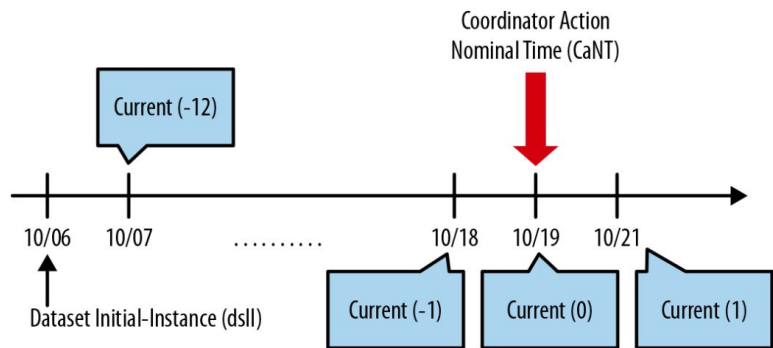
- Describe the actual instance(s) of dependent dataset for this coordinator
- workflow will not start until all the data instances defined in the input-events are available

```
<!-- input-events -->
<input-events>
  <data-in name="input_hdfs_nctr_dataset" dataset="HDFS_NCTR_DATASET">
    <instance>${coord:latest(0)}</instance>
  </data-in>
  <data-in name="input_hdfs_cow_targeting_attribute" dataset="HDFS_COW_TARGETING_ATTRIBUTE">
    <instance>${coord:latest(0)}</instance>
  </data-in>
</input-events>
```

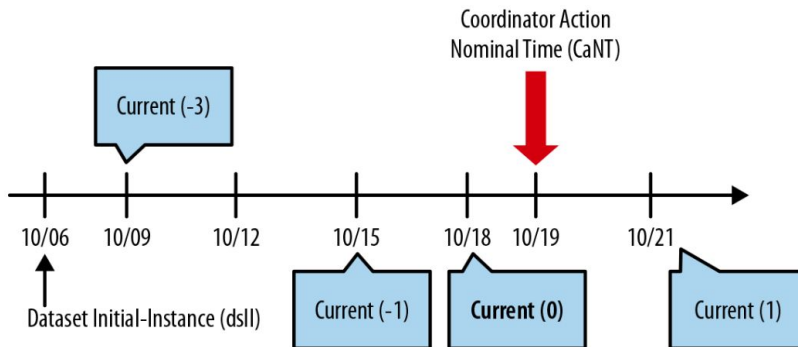

Parametrization of Data Instances

- Each coordinator action waits for data instances defined in <data-in>
- Each data instance needs the absolute timestamp to evaluate the exact directory provided in uri-template
- If the start time of a coordinator job is cS and the frequency is cF , the nominal time of the n th coordinator action $caNT \Rightarrow cS + n * cF$
- $current(n)$ - Returns the timestamp of the n th instance of a dataset relative to a specific coordinator action's nominal time ($caNT$)
 - $current(n) = dsII + dsF * (n + (caNT - dsII) / dsF)$
 - $dsII$ = data set initial instance, dsF = data set frequency, n = integer

Current(n) evaluation

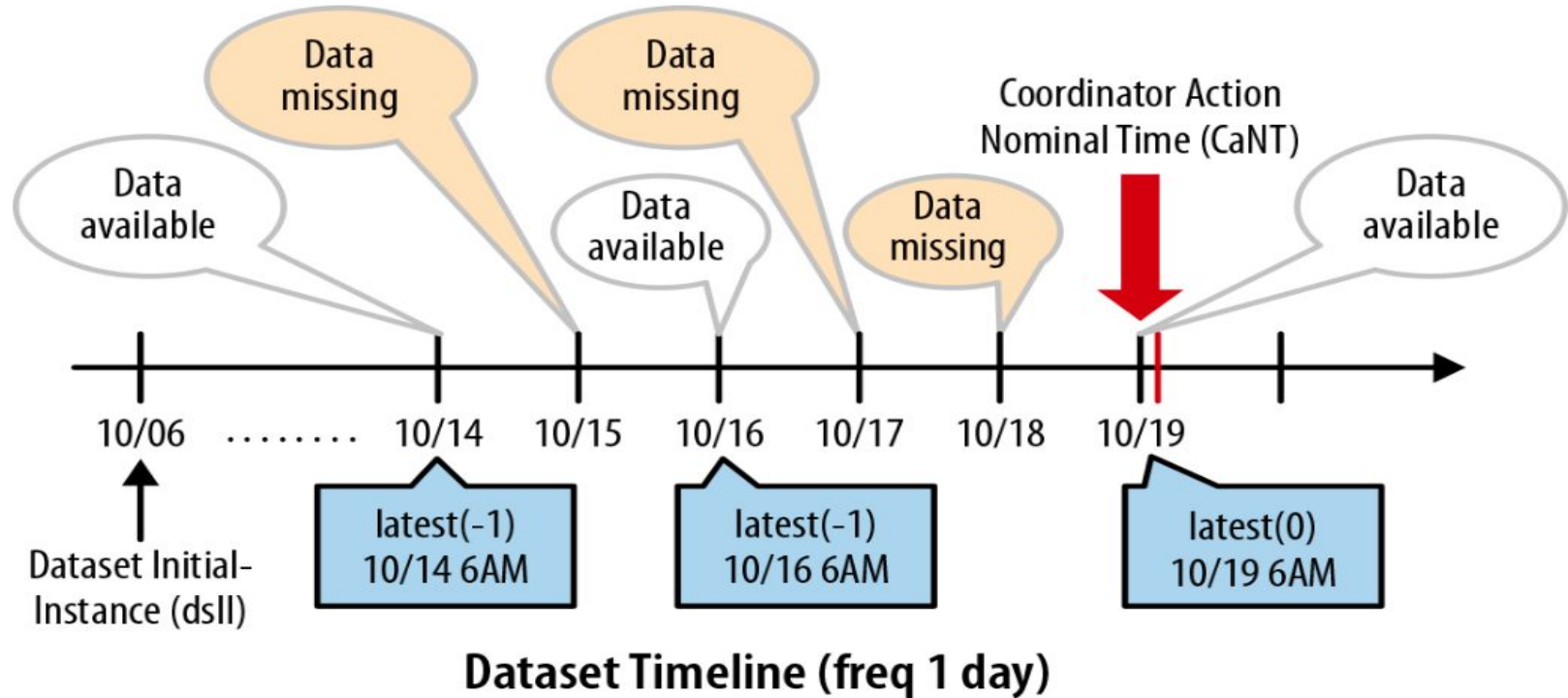


Dataset (*ds1*) Timeline (freq 1 day)



Dataset (*ds3*) Timeline (freq 3 days)

latest() evaluation



Sample Coordinator XML

```
<?xml version="1.0" encoding="UTF-8"?>
<coordinator-app xmlns="uri:oozie:coordinator:0.4" name="${coordinatorName}" frequency="${coord_freq}" start="${start}" end="${end}" timezone="${ti
  <datasets>
    <dataset name="dsReadyFlag" frequency="${ds_freq}" initial-instance="${start}" timezone="${timezone}">
      <uri-template>${nameNode}${outRoot}</uri-template>
      <done-flag>trigger.dat</done-flag>
    </dataset>
  </datasets>
  <action>
    <workflow>
      <app-path>${wfAppConfPath}</app-path>
      <configuration>
        <property>
          <name>jobTracker</name>
          <value>${jobTracker}</value>
        </property>
        <property>
          <name>nameNode</name>
          <value>${nameNode}</value>
        </property>
        <property>
          <name>queueName</name>
          <value>${queueName}</value>
        </property>
        <property>
          <name>oozie.libpath</name>
          <value>${wfAppLibPath}</value>
        </property>
      </configuration>
    </workflow>
  </action>
</coordinator-app>
```

Any Questions ?