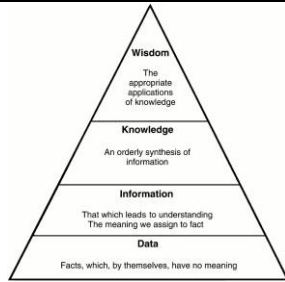


# CS286 Solving Big Data Problems – Exam #1 Study Guide

By: Zayd Hammoudeh

## Lecture #01 – Introduction to Big Data

Data Categories		Data – Raw values	
<b>Quantitative</b> <ul style="list-style-type: none"> <li>Observable and <b>measureable</b></li> <li>Structured and <b>objective</b></li> <li>Numerical</li> </ul> <b>Example:</b> Income, Height	<b>Qualitative</b> <ul style="list-style-type: none"> <li>Observable but <b>NOT measureable</b></li> <li>Unstructured and subjective</li> <li><b>Descriptive</b></li> </ul> <b>Example:</b> Favorite Color	<b>Information</b> – Set of data with meaning  <b>Knowledge</b> – Interpretation of the data with meaning.  <b>Wisdom</b> – Appropriate application of knowledge.	

### Storage Terminology

<b>Directly Attached Storage (DAS)</b> <ul style="list-style-type: none"> <li>Storage attached directly to the processing node.</li> <li>Lowest capacity</li> <li>Minimal data sharing</li> <li><b>Highest Speed.</b></li> </ul>	<b>Network Attached Storage (NAS)</b> <ul style="list-style-type: none"> <li>Storage accessible via a network connection.</li> <li><b>Capable of using NFS</b></li> </ul>	<b>Relational Database Management System (RDBMS)</b> <ul style="list-style-type: none"> <li>Traditional database providers.</li> <li><b>Examples:</b> Oracle, MySQL, IBM DB2</li> </ul>	<b>Storage Area Network (SAN)</b> <ul style="list-style-type: none"> <li>Storage accessible via a network connection.</li> <li>Uses different protocols than NAS.</li> </ul>	<b>Network File System (NFS)</b> Allows a computer to view and store data on remote disk as if that disk was directly attached to the local computer.  <b>Access Transparency</b> – Access data the same way whether it is remote or local.
--	---	---	--	--

Data Analysis Categories		<b>Four Steps in Traditional Data Mining</b> <ol style="list-style-type: none"> <li>Problem Definition</li> <li>Data gathering and preparation</li> <li>Model building and evaluation</li> <li>Knowledge Deployment</li> </ol> Process is <b>cyclical</b> and <b>may repeat multiple times.</b>	
<b>Descriptive</b> <ul style="list-style-type: none"> <li><b>Backward</b> looking.</li> <li>Hindsight</li> <li>Explain a previous phenomenon.</li> <li><b>Analysis</b></li> </ul>	<b>Predictive</b> <ul style="list-style-type: none"> <li><b>Forward</b> looking</li> <li>Foresight</li> <li>Investigate future trends.</li> <li><b>Mining</b></li> </ul>		

### Big Data

<b>Big Data</b> – Data whose scale, diversity, and complexity require new architecture, techniques, algorithms, and analytics to manage it and to extract value and hidden knowledge from it.	3 V's of Big Data		
	<b>Volume</b> – The amount of data is too large for traditional database software tools to cope with.  <b>Example:</b> Image server	<b>Velocity</b> – The data is being produced at a rate that is beyond the performance limits of traditional systems.  <b>Example:</b> Social media site	<b>Variety</b> – Data lacks the structure to make it suitable for storage and analysis in traditional databases and data warehouses.  <b>Example:</b> Data organization variety.

Data Organization			Scaling to Process Big Data		
<b>Structured</b> – Every piece of data and its format is known. Fits in a database.  <b>Example:</b> RDBMS	<b>Semi-structured</b> – For some fields, data may not exist and some fields can have different formats. Not in a typical database but has structure.  <b>Example:</b> XML, CSV, JSON	<b>Unstructured</b> – Does not fit into a database well. Most data is in this category.  <b>Examples:</b> Text document, multimedia content.	<b>Scale Up</b>  <b>Limitations:</b> <ul style="list-style-type: none"> <li>Large capital and operating expense.</li> <li>Lower availability and scalability.</li> </ul> <b>Example:</b> Monolithic Database	<b>Scale Out</b>  <b>Limitations:</b> <ul style="list-style-type: none"> <li>Synchronization overhead</li> <li>Programming Complexity</li> <li>Specialized hardware.</li> </ul> <b>Example:</b> Grid Cluster	<b>Sampling</b>  <b>Limitation:</b> <ul style="list-style-type: none"> <li>Lower accuracy and precision.</li> </ul> <b>Example:</b> Any approach

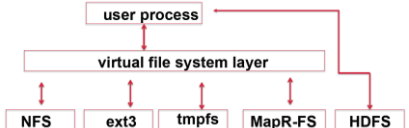
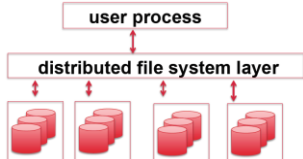
<b>Exploiting Locality of Reference</b> – In Big Data, accessing the data can be very time consuming. <b>Solution:</b> Keep the data and program close together.  <b>Distribute Data and Computation</b> – Map the data to multiple nodes and the program with it to decrease execution time.	Three Laws of Big Data		
	<b>Moore's Law</b> – Every two years, the number of transistors per chip doubles.  <b>Kryder's Law</b> – Every two years, storage capacity doubles. ( <b>Storage version of Moore's Law</b> )	<b>Amdahl's Law</b> – The extent to which a program's execution can be sped up is dependent on its level of parallelism.	<b>Murphy's Law</b> – What can go wrong will go wrong.  <b>Big data must be resistant to failures.</b>

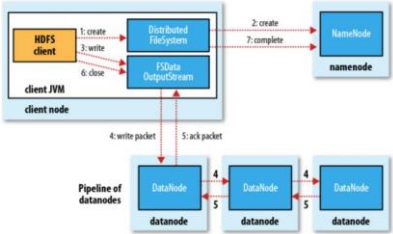
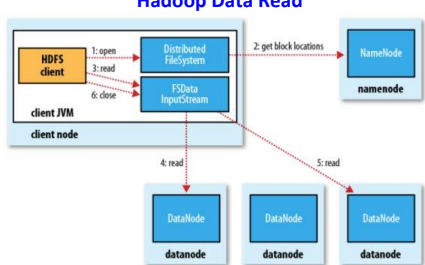
# Hadoop

Summary of the Hadoop Strategy			Core of Hadoop	Name Node	Job Tracker
<b>Distribute Data</b> Processing nodes share no data.	<b>Distribute Computation</b> Achieve <b>parallelism without synchronization</b> .	<b>Tolerate Failures</b> Eliminate <b>single points of failure</b> .	1. <b>Hadoop File System (HDFS)</b> – Storage level 2. <b>MapReduce</b> – Compute Level	Key component in HDFS that <b>stores the location of distributed data in the file system</b> .	Manages <b>computation tasks in the Hadoop system</b> .

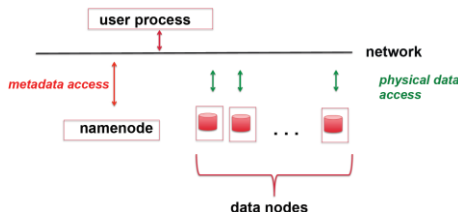
## Lecture #02 – Introduction to HDFS and MapR-FS

File System	Storage in a File System	Block Structure in an ext2 File System			
Like a database. <b>A system to store data so that the data can be accessed later.</b>  <b>Typical Structure:</b> A rooted tree.	<b>Data</b> – Actual file in the FS.  <b>Metadata</b> – Information about the data/file. <b>Example:</b> Size, location	<b>Hadoop Block Size:</b> 64MB	<b>Inode</b> – Data structure used to represent a file system object. This includes the location of the disk block location.	<b>Direct Block</b> – File block location <b>pointed to directly by the inode</b> .	<b>Indirect Block</b> – Block <b>pointed to by the inode through exactly one intermediary block</b> .  <b>Double Indirect Block</b> – Block <b>pointed to by the inode through exactly two intermediary blocks</b> .

Virtual File System	Distributed File System
<ul style="list-style-type: none"> <li><b>Transition layer</b> between a generic file system and a real file system.</li> <li>Virtualizes different file system types into a single common interface.</li> <li><b>Enables standard POSIX</b> file access.</li> <li><b>HDFS is not compatible with a virtual file system while MapR-FS is.</b></li> </ul> 	<ul style="list-style-type: none"> <li><b>Centrally stores metadata</b> (e.g. <b>name node</b>) and <b>distributes actual data</b> (e.g. <b>data node</b>)</li> <li>Overcomes <b>space, performance, and availability limitations of a single machine</b>.</li> <li><b>Location Transparency</b> – Abstracts data locality from client access.</li> </ul> 

Hadoop Data Write	Hadoop Data Read	Hadoop Write Pipeline
		<p><b>Hadoop Write Pipeline</b> – Before a write can be acknowledged to the client, it must be acknowledged by the name node.</p> <ul style="list-style-type: none"> <li>Each replicate write is sequential <b>through a pipeline where one data node writes to the next</b>.</li> </ul> <p><b>Sequential Block Reading</b> – Each file block is read <b>sequentially</b> even if the blocks reside on multiple data nodes and could theoretically be read in parallel.</p> <ul style="list-style-type: none"> <li><b>Block size:</b> 64MB</li> </ul>

## Hadoop Distributed File System (HDFS) Architecture

Architecture Diagram	User Process	Name Node – Master	Data Node – Slave
	<ul style="list-style-type: none"> <li>Connected to HDFS through the network.</li> <li><b>Communicates with the name node to know where to read and write data.</b></li> </ul>	<ul style="list-style-type: none"> <li><b>Manages file names and locations on disk. Provides metadata information</b></li> <li><b>All data is persisted in memory (RAM)</b></li> <li>May have a <b>secondary name node</b> used to offload processing (e.g. writing logs) off the primary. <b>Secondary is not for high availability.</b></li> <li>All writes must be acknowledged by the name node before they can be acknowledged to the user process.</li> </ul>	<ul style="list-style-type: none"> <li><b>Persistent storage disks for the data.</b></li> <li>Data is replicated across multiple data nodes if possible across multiple racks.</li> </ul>

## Limitations of HDFS

Mutability	Block Size	POSIX Semantics	Availability	Scalability	Performance
Data is write once, read many.	Single block size (e.g. <b>64MB</b> ) for disk I/O, replication and sharding	Must use the command <b>“hadoop fs”</b> to access the data. <b>Example POSIX Commands:</b> Open, close, read, write.	No snapshot or built-in mirroring capability.	Name node only scales to 100M files. This is <b>due to the single name node persisting all data in RAM</b> .	Written in <b>Java</b> and runs on a block device

## Overview of MapR File System (MapR-FS)

<b>Physical Disk</b> – A single hard drive.  <b>Storage Pool</b> – Three striped physical disks. Striping is used to increase write performance.	<b>Node</b> – A set of storage pools.  <b>Topology</b> – A set of nodes.	<b>Container</b> – <b>Unit of shared storage</b> . It is the size of replicated data. A <b>storage pool has multiple containers</b> . Each container belongs to only one volume.	<b>Volume</b> – A tree of files and directories grouped for the purpose of applying a policy or set of policies.
--	--	--	--

## MapR-FS Volume Features

<b>Topologies</b> Provide data placement policies.	<b>Compression</b> Compress data as it is written to disk.	<b>Mirroring</b> Copy data locally or remotely for protection <b>in real time for load balancing, backup, and disaster readiness.</b>	<b>Snapshots</b> Maintain point-in-time data and updates.	<b>Quotas</b> Restrict total capacity per-user or per-group.	<b>Permissions</b> Restrict access to users or groups.	<b>Replication</b> Replicate containers in a volume across the cluster
---	---	--	--	---	---	---

## Differences between MapR-FS and HDFS

<b>Block Size</b> MapR-FS supports different block sizes for sharding, replication, and performing I/O.	<b>Mutability</b> MapR-FS has full read write capability.	<b>Access</b> MapR-FS volumes can be NFS-mounted.	<b>POSIX Support</b> MapR-FS supports native OS commands to access data.	<b>Availability</b> MapR-FS supports snapshots and local/remote mirroring support.	<b>Scalability</b> No limit to the number of files.	<b>Performance</b> <b>MapR-FS is written in C and runs on a raw device (i.e. no filesystem overhead).</b>
--	--	--	---	---	--	--

<div>Block Size Comparison between HDFS and MapR-FS</div>			<div>Role of a Single Sharding Unit (e.g. Block/Chunk) – In Map Reduce, each mapper is assigned a single shard (e.g. block/chunk) to analyze.</div>	<div>Using the “hadoop fs” Command Line Interface (CLI)</div>
<div>Storage Unit</div>	<div>HDFS</div>	<div>MapR-FS</div>	<div>Relationship between Container and Volume – In MapR-FS, a container is assigned to a single volume and a volume is made up of one or more containers.</div>	<div>Format:</div>
<div>Unit of Sharding</div>	<div>Block=64MB</div>	<div>Chunk=256MB</div>	<div>Example Block/Chunk Count Calculation: If a Map Reduce file has 300MB of data, it will required 5 blocks in HDFS and 2 chunks in MapR-FS.</div>	<div>hadoop fs -&lt;command&gt; [args]</div>
<div>Unit of Replication</div>	<div>Block=64MB</div>	<div>Container = 16-32GB</div>		<div>Examples:</div>
<div>Unit of I/O</div>	<div>Block=64MB</div>	<div>Block=8KB</div>		<div>hadoop fs -mkdir newDirectory</div>
<div>MapR-FS allows for different storage unit sizes to optimize performance.</div>				<div>hadoop fs -rm my_file.txt</div>
				<div>Not Supported Command:</div>
				<div>hadoop fs -cd ...</div>
				<div>This command has no directory state so must use absolute path.</div>
				<div>hadoop mfs [command] [args]</div>
				<div>Performs MapR-FS operations similar to hadoop fs.</div>

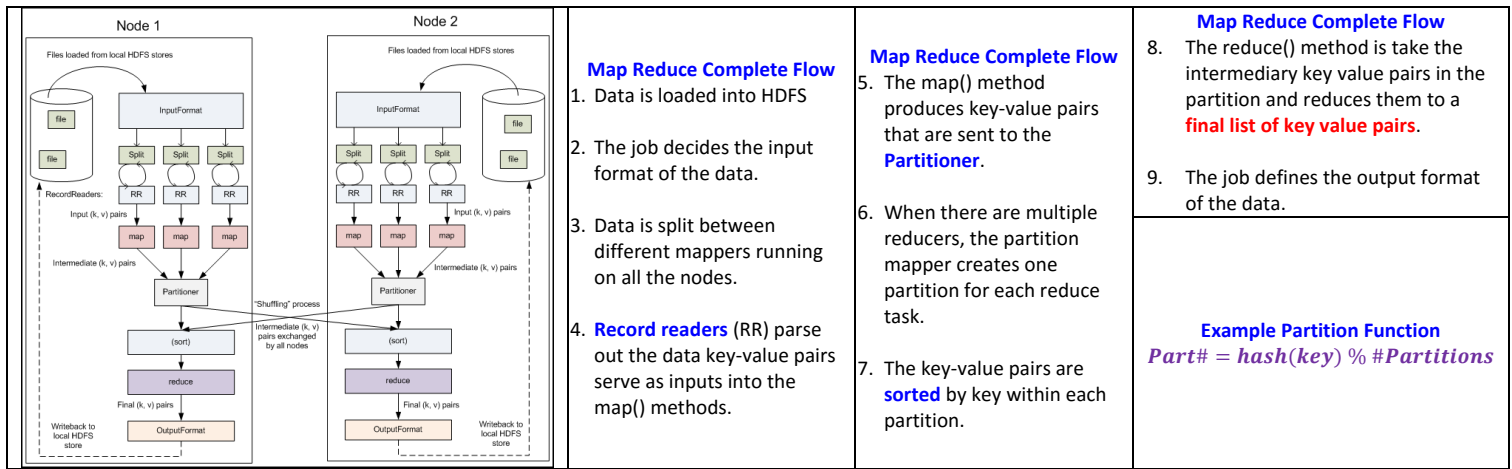
## Lecture #03 – Introduction to MapReduce

<b>Map Reduce Underlying Principle:</b> <b>Divide and Conquer</b>  <b>Derives from Lisp</b>	<b>map</b> (String key, <b>String</b> value): // key: document or shard name // value: document or shard contents <b>for each</b> word <b>w</b> <b>in</b> value: <b>EmitIntermediate</b> ((w,"1")); // key value pair	<b>reduce</b> (String key, <b>Iterator</b> values): // key: a word // values: a list of word counts <b>int</b> results = 0 <b>for each</b> v <b>in</b> values: results += <b>ParseInt</b> (v) <b>Emit</b> ( <b>AsString</b> (result))  <b>Reduce is called one on each key NOT each partition.</b>	<b>Key Methods</b>  <b>EmitIntermediate</b> – Output of the mapper function. Writes an intermediary key-value pair to be analyzed by a reducer.  <b>Emit</b> – Outputs the result of the reducer.
--	---	--	---

<b>Three Phases of Map Reduce</b> 1. Map 2. Sort/Shuffle/Merge 3. Reduce	<b>Map</b> <ul style="list-style-type: none"> <li>One mapper is assigned per input split. The <b>“map” function is called once for each key-value pair</b> (i.e. record).</li> <li>Each mapper processes a local data set and can <b>output a set of intermediary key-value pairs</b>.</li> <li><b>“Send the compute to where the data is.”</b></li> <li><b>Outputs zero or more key-value pairs.</b></li> </ul>	<b>Sort/Shuffle/Merge</b> <ul style="list-style-type: none"> <li>Transfer results from mappers to reducers.</li> <li>Creates <b><i>n</i></b> partitions where <b><i>n</i></b> is equal to the number of reducers.</li> <li>Divides intermediary key value pairs into the <b><i>n</i></b> partitions.</li> <li>May run a <b>“Combiner”</b> function to merge results from the Map stage to reduce the amount of data to transfer over the network.</li> <li>After keys are partitioned and merge, the keys in the partition are sorted.</li> <li>Partitions are sent over the network to the reducers. Hadoop uses HTTP while MapR-FS uses RPC.</li> </ul>	<b>Reduce</b> <ul style="list-style-type: none"> <li>One reducer per input partition. The <b>“reduce” method is called once per key</b>.</li> <li><b>Outputs zero or more key value pairs.</b></li> <li>Reads one list of values for each key.</li> <li><b>No data locality exploitation in reduce.</b></li> </ul>
---	--	---	--

<b>Responsibilities of the Map Reduce Framework</b> <ul style="list-style-type: none"> <li><b>Split the incoming input file and read the records.</b></li> <li><b>Schedules, runs, and reruns map/reduce tasks.</b></li> <li><b>Transfers map outputs to reduce inputs.</b></li> <li><b>Collects and writes status and results.</b></li> </ul>	<b>Map Reduce Block and Record Splitting</b> <ul style="list-style-type: none"> <li>The Map Reduce framework divides an input file to one or more <b>splits\block</b>.</li> <li>A split\block contains one or more (typically many) <b>records. Default record delimiter is “\n”.</b></li> <li><b>The map function is called once per record.</b></li> </ul> <b>Map Record Key-Value Format</b> <ul style="list-style-type: none"> <li><b>key</b> – <b>Byte offset</b> for start of record</li> <li><b>value</b> – Record data in the split.</li> </ul>	<b>Typical Map Reduce Workflow</b> <ol style="list-style-type: none"> <li><b>Load the data into the cluster.</b> <ul style="list-style-type: none"> <li><b>HDFS</b> – Uses WORM (write once read many). Preload only.</li> <li><b>MapR-FS</b> – POSIX + network file system (NFS) access. Preload or persistent storage.</li> </ul> </li> <li><b>Analyze the data</b></li> <li><b>Store the results in the cluster</b> (e.g. in HDFS/MapR-FS)</li> </ol> <p><b>Read the results from the cluster.</b></p>
--	---	---

## MapReduce Complete Flow



## Hadoop Classes

<p><b>InputFormat</b></p> <ul style="list-style-type: none"> <li>Checks if the input file exists.</li> <li>Splits the input file into one or more <b>InputSplit</b> objects.</li> <li>Instantiates <b>RecordReader</b> to partition splits into records which are turned into key-value pairs. <ul style="list-style-type: none"> <li><b>Key is byte offset</b> of the start of the record.</li> </ul> </li> </ul>	<p><b>Mapper</b></p> <ul style="list-style-type: none"> <li>Implements the map() method.</li> <li>One Mapper object is created for each input split.</li> <li>Processes keys and/or values.</li> <li>Updates status in reporter.</li> <li>Writes output.</li> </ul>	<p><b>Partitioner</b></p> <ul style="list-style-type: none"> <li>Takes the output(s) generated by the map() method and <b>creates partitions based on the hashed key</b>.</li> <li>Each partition is assigned to a single reducer.</li> <li><b>All records with the same key are assigned to the same partition.</b></li> </ul>	<p><b>Combiner (Optional)</b></p> <ul style="list-style-type: none"> <li>Has <b>no default behavior</b>.</li> <li><b>Motivation:</b> Reduce the intermediate values of the mappers before they are sent over the network.</li> <li><b>Often the reducer can be repurposed as a combiner.</b></li> </ul>	<p><b>Reducer</b></p> <ul style="list-style-type: none"> <li>Implements the reduce() method.</li> <li>Each Reducer object is assigned one partition.</li> <li>Executes the reduce method on each key in the partition.</li> <li>Updates status in reporter.</li> <li>Writes output.</li> </ul>
--	---	---	---	--

<b>Outputs of a MapReduce Job</b> <ul style="list-style-type: none"><li>• <b>_SUCCESS</b> – Empty file indicating the job was completed successfully.</li><li>• <b>part-m-00000</b> – First intermediate results output file from a map task.</li><li>• <b>part-r-00000</b> – First intermediate results output file from a single reducer.</li></ul>	<b>Hadoop Job Execution Framework</b>			<b>Hadoop Schedulers</b> <ul style="list-style-type: none"><li>• <b>Fair Scheduler (default)</b> – Resources shared evenly among pools.<ul style="list-style-type: none"><li>◦ Each user has a pool. Custom pools can be created. Supports Pre-emption.</li></ul></li><li>• <b>Capacity Scheduler</b> – Resources shared among queues. Admin creates hierarchical queues. <b>Supports soft and hard capacity limits to users within a queue.</b></li></ul>
	<b>JobClient</b> <ul style="list-style-type: none"><li>• Instantiated by the client. Submits job to the JobTracker. Runs inside a JVM.</li></ul>	<b>JobTracker</b> <ul style="list-style-type: none"><li>• Instantiates a Job object which gets sent to the TaskTracker(s). Runs inside a JVM.</li><li>• Reschedules tasks on failed TaskTrackers to other TaskTrackers.</li></ul>	<b>TaskTracker</b> <ul style="list-style-type: none"><li>• Launches a child process that runs a <b>MapTask</b> or a <b>ReduceTask</b>.</li><li>• <b>HeartBeat Messages</b> to JobTracker include:<ul style="list-style-type: none"><li>◦ <b>Task Status</b></li><li>◦ <b>Task Counter</b></li><li>◦ <b>Data read/write status</b></li></ul></li></ul>	

Hadoop Fair Scheduler		Hadoop Capacity Scheduler		MCS – MapR Control System  CLDB – Container Location Database.
<ul style="list-style-type: none"><li>• <b>Pool</b> – Set of jobs.</li><li>• <b>User configures priority of jobs within a pool.</b></li><li>• Default of one user per pool.</li><li>• “Over-using” users can be preempted.</li><li>• <b>Developed at Facebook.</b></li></ul>	<p><b>Scheduling Algorithm</b></p> <ul style="list-style-type: none"><li>• Divide each pool’s min maps and reduces among jobs.</li><li>• When a slot is free, allocate a job that is below its minimum share (i.e. most starved).</li><li>• Preempt long running jobs to meet minimum guarantees.</li></ul>	<ul style="list-style-type: none"><li>• <b>Queue</b> – Set of Jobs</li><li>• Queues may be <b>hierarchically organized</b> (i.e. <b>a queue is made of other queues</b>).</li><li>• <b>Shares assigned to queues as a percentage of total resources.</b></li><li>• Per-Queue and Per-User configurations.</li><li>• <b>Developed at Yahoo.</b></li></ul>	<p><b>Scheduling Algorithm</b></p> <ul style="list-style-type: none"><li>• Allocate slots to queues based on percentage of shares.</li><li>• <b>FIFO scheduling within each queue.</b></li></ul>	

## Limitations of the Hadoop Execution Framework

<p><b>Scalability</b></p> <p>Single JobTracker restricts job throughput.</p>	<p><b>Availability</b></p> <p>Only one JobTracker and one NameNode introduces single points of failure (SPOF).</p>	<p><b>Inflexibility</b></p> <p>Map and reduce jobs are not interchangeable.</p>	<p><b>Scheduler Optimization</b></p> <p>Framework does not optimize scheduling of jobs.</p>	<p><b>Program Support</b></p> <p>Framework is limited to Map and Reduce programs.</p>
--	--	---	---	---

**Inflexibility and program support are addressed in Map Reduce version 2** (also known as **YARN**)

## Lecture #04 – Installing MapR

Disk Provisioning	Network Configuration	Joining Data
<ul style="list-style-type: none"> <li><b>Dynamic</b> – Thin provisioning</li> <li><b>Fixed</b> – Thick provisioning</li> </ul>	<ul style="list-style-type: none"> <li><b>NAT</b> – The VM does not have a separate IP from the host. Rather a separate private network is setup on the host machine and the VM gets an address in that network. Network traffic looks as though it came from the host PC.</li> <li><b>Bridged</b> – Replicates another node on the physical network and the VM gets its own IP.</li> <li><b>Host-Only</b> – The nested VM's network is within the host computer only.</li> </ul>	<p>Join can be done in the map and reduce stages.</p>

## Lecture #05 – Writing a MapReduce Program

### Common Map Reduce Applications

Summarizing Data	Filtering Data	Organizing Data	Joining Data
			Join can be done in the map and reduce stages.

### MapReduce Program Imports

<b>org.apache.hadoop.mapreduce.*</b> Includes the definition of the “ <b>Mapper</b> ”, “ <b>Reducer</b> ”, “ <b>Job</b> ”, and “ <b>Context</b> ” classes.	<b>org.apache.hadoop.io.*</b> Includes the definition of the “ <b>Text</b> ”, “ <b>LongWritable</b> ”, and “ <b>IntWritable</b> ” classes.	<b>org.apache.hadoop.conf.*</b> Includes the definition of the “ <b>Configured</b> ” and “ <b>Configuration</b> ” classes.	<b>org.apache.hadoop.util.*</b> Includes the definition of the “ <b>Tool</b> ” interface and “ <b>ToolRunner</b> ” class.
<b>org.apache.hadoop.mapreduce.lib.input.*</b> Includes the definition of the “ <b>TextInputFormat</b> ” and “ <b>FileInputFormat</b> ” classes.	<b>org.apache.hadoop.fs.*</b> Includes the definition of the “ <b>Path</b> ” class.	<b>java.util.*</b> Includes the definition of the “ <b>StringTokenizer</b> ” class.	<b>java.io.*</b> Includes the definition of the “ <b>IOException</b> ” class.
<b>org.apache.hadoop.mapreduce.lib.output.*</b> Includes the definition of the “ <b>FileOutputFormat</b> ” class.			

### MapReduce Class Definitions

Mapper Class Definition	Reducer Class Definition	Driver Class Definition
<pre>import java.util.*; import java.io.*; import org.apache.hadoop.mapreduce.*; import org.apache.hadoop.io.*;  public class MyMapper     extends Mapper&lt;InputKeyClassName,                   InputValueClassName,                   OutputKeysClassName,                   OutputValuesClassName&gt; { }</pre> <p>Must override the “<b>map</b>” method.</p> <p>InputFormat – <b>TextInputFormat</b> Key Class – <b>LongWritable</b> Value Class – <b>Text</b></p>	<pre>import java.util.*; import java.io.*; import org.apache.hadoop.mapreduce.*; import org.apache.hadoop.io.*;  public class MyReducer     extends Reducer&lt;InputKeyClassName,                    InputValueSClassName,                    OutputKeysClassName,                    OutputValuesClassName&gt; { }</pre> <p>Must override the “<b>reduce</b>” method.</p> <p><b>The input key and value types for the Reducer must match the output key and value types for the associated Mapper.</b></p>	<pre>import org.apache.hadoop.conf.*; import org.apache.hadoop.io.*; import org.apache.hadoop.mapreduce.*; import org.apache.hadoop.mapreduce.lib.input.*; import org.apache.hadoop.util.*;  public class ReceiptsDriver extends Configured     implements Tool {     ...     public static void main(String[] args) throws Exception{         Configuration conf = new Configuration();         System.exit(<b>ToolRunner.run</b>(conf, new ReceiptsDriver(), args));     } }</pre> <ul style="list-style-type: none"> <li>Must implement the “<b>run</b>” method.</li> <li><b>Specifies whether the job is run synchronously or asynchronously</b> via the “<b>waitForCompletion</b>” command.</li> <li><b>Specifies class types for mapper and reducer.</b></li> <li>Verifies function input arguments.</li> </ul>

## MapReduce Class Method Definitions

<p><b>map Function Format</b></p> <pre>@Override public void map(LongWritable key, Text value,                Context context)     throws IOException, InterruptedException {      StringTokenizer strToken = new StringTokenizer(value,         splitCriteria);      // Iterate through all the tokens in the record     while(strToken.hasMoreTokens()){         String myStr = strToken.nextToken();         ...          // Emit any intermediate &lt;key, value&gt; pairs         // Optional to emit any pairs.         context.write(new OutputKeysClassName(...),             new OutputValuesClassName (...));     }      First two arguments in the map method are     the input key and record value.      Map is called once per input record.</pre>	<p><b>reduce Function Format</b></p> <pre>@Override public void reduce(Text key, Iterable&lt;Text&gt; value, Context context)     throws IOException, InterruptedException {      // Parse the Iterable object     for(Text value: values)      // Emit any intermediate &lt;key, value&gt; pairs     // Optional to emit any pairs.     context.write(new OutputKeysClassName(...),         new OutputValuesClassName (...));     }      Reduce is called once per intermediate key.</pre>	<p><b>run Function Format</b></p> <pre>@Override public int run(String[] args) throws Exception {     if(args.length != 2){         System.err.printf("usage: %s [general options] &lt;inputfile&gt; &lt;outputfile&gt;\n",             getClass().getSimpleName());         System.exit(1);     }      // Configure the job     Job job = new Job ( getConf(), "job name");      job.setJarByClass(MyDriver.class);     job.setMapperClass(MyMapper.class);     job.setReducerClass(MyReducer.class);      // Define input file's format (e.g. text file)     job.setInputFormatClass(TextInputFormat.class);      // Setup the mapper output classes.     // Mapper Input class are a LongWritable by default and Text     job.setMapOutputKeyClass(MapperOutputKeysClassName.class);     job.setMapOutputValueClass(MapperOutputValuesClassName.class);      // Set the reducer's output class.     job.setOutputKeyClass(ReduceOutputKeysClassName.class);     job.setOutputValueClass(ReduceOutputValuesClassName.class);      // Set the reducer's output class.     FileInputFormat.addInputFormat (job, new Path(&lt;inputfilepath&gt;);     FileOutputFormat.setOutputFormat (job, new Path(&lt;outputfolderpath&gt;);      // Wait for the job to finish.     return job.waitForCompletion(true) ? 0 : 1; }</pre>
--	---	---

## MapReduce Environment Variables

<p><b>HADOOP_HOME</b></p> <ul style="list-style-type: none"> <li>Path: /opt/mapr/hadoop/Hadoop-0.20.2</li> <li>Not required.</li> <li>Useful when defining other environment variables.</li> </ul>	<p><b>LD_LIBRARY_PATH</b></p> <ul style="list-style-type: none"> <li>Path: \$HADOOP_HOME/lib/native/Linux-amd64-64</li> <li>Not required.</li> <li>Enables the use of libraries specifically compiled for MapR.</li> </ul>	<p><b>PATH</b></p> <ul style="list-style-type: none"> <li>Path: \$HADOOP_HOME/bin:\$PATH</li> <li>Not required.</li> <li>Order in PATH variable is important as earlier items in the list take precedence.</li> <li>Provides path to Hadoop executables so user does not need to specify the absolute path.</li> </ul>
<p><b>CLASSPATH</b></p> <ul style="list-style-type: none"> <li>Path: \$HADOOP_HOME/*:\$HADOOP_HOME/lib/*</li> <li>Not required.</li> <li>Points to all jars in the Hadoop distribution required to run a program.</li> </ul>	<p><b>HADOOP_CLASSPATH</b></p> <ul style="list-style-type: none"> <li>Path: \$CLASSPATH</li> <li>Not required.</li> <li>Makes it easier to run MapReduce applications from the hadoop command.</li> </ul>	<p><b>export</b></p> <p>Bash command to add environment variables to the terminal.</p>

## Command Line Instructions

<p><b>javac</b></p> <ul style="list-style-type: none"> <li>Compiles a Java class from ASCII to byte code.</li> <li>Example:                     <pre>javac -d &lt;FolderName&gt; &lt;ClassName&gt;.java</pre> </li> <li>-d – Allows for a custom output directory to be used.</li> </ul> <p><b>hadoop jar</b></p> <ul style="list-style-type: none"> <li>Launches a Hadoop job.</li> <li>Example:                     <pre>hadoop jar &lt;JarNameAndPath&gt;.jar &lt;DriverClass&gt; file://&lt;InputPathAndFile&gt; &lt;outputDirectory&gt;</pre> </li> <li>Arguments in the call correspond to the args argument in the driver.</li> <li>-D – Used to specify properties of the Hadoop jar operation.</li> </ul> <p><b>hadoop fs</b></p> <ul style="list-style-type: none"> <li>Enables POSIX style commands on HDFS</li> <li>Example:                     <pre>hadoop fs -&lt;CommandName&gt; [args]</pre> </li> <li>Must precede POSIX command (e.g. ls, cat, rm, etc.) with a hyphen.</li> </ul>	<p><b>jar</b></p> <ul style="list-style-type: none"> <li>Combines the different class files into a single Java Archive (JAR) File.</li> <li>Example #1: Create a New JAR File                     <pre>jar -cvf &lt;jarname&gt;.jar -C &lt;classfolder&gt;/.                      -c – Create a new JAR file.                     -v – Generate a verbose output.                     -f – Specifies that the command includes the output JAR's file name.                     -C – Specifies the location of the source .class files.</pre> </li> <li>Example #1: Updating an Existing JAR                     <pre>jar -uvf &lt;jarname&gt;.jar -C &lt;classfolder&gt;/.                      -u – Update a JAR.</pre> </li> </ul>
---	--



# Lecture #06 – Using the mapreduce API

## Hadoop and MapR

- MapR 3.0.1 ships with version 0.20.2 of Hadoop.

## Setting HADOOP\_HOME PATH

HADOOP\_HOME = /opt/mapr/hadoop/hadoop-0.20.2

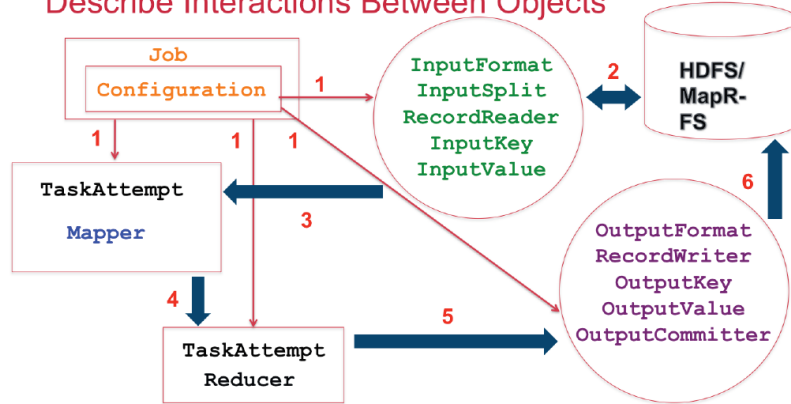
## Comparison of the mapreduce and mapred Libraries

	Supported on MapR	Deprecated	YARN-Compatible	Types	Objects
<b>mapred</b>	Yes	No	Yes	Interfaces	OutputCollector, Reporter, JobConf
<b>mapreduce</b>	Yes	No	Yes	Abstract Classes	Context

	Methods	Output Files	Reducer Input Values	Import Command
<b>mapred</b>	map(), reduce()	part-xxxx	java.lang.iterator	import org.apache.hadoop.mapred.*
<b>mapreduce</b>	map(), reduce(), cleanup(), setup(), run()	part-m-xxxx (Mapper) part-r-yyyy (Reducer)	Java.lang.Iterable	import org.apache.hadoop.mapreduce.*

## Describe Interactions Between Objects



## Writable Types

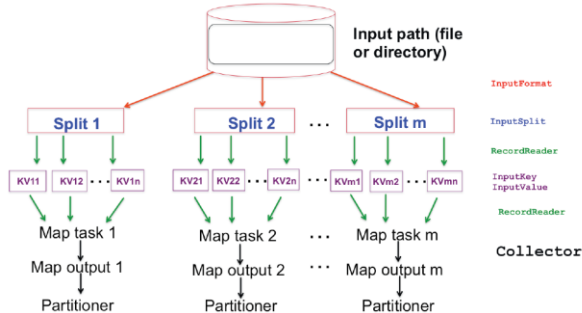
- All Key/Value types must implement the Writable Interface.
- Used to serialize keys/values before they are written to disk.
- All Java primitives must have a wrapper class to be able to return/pass from map/reduce calls.
- Do not support commands on equivalent Java primitives. **Example:** cannot use "+" to add to LongWritable's.

## Writable Interface Methods

- void write(DataOutput out) throws IOException
- void readFields(DataInput in) throws IOException

Java Primitive	Hadoop Writable Type
boolean	BooleanWritable
long	LongWritable new LongWritable(1)
double	DoubleWritable
string	Text (UTF-8 Format) new Text("my String")
N/A	BytesWritable (Writable Binary)

## Describe Mapper Input Flow



## WritableComparable

- All keys must implement the Writable and Comparable Interfaces.

## Comparable Interface

int compareTo(WriteComparable o)

- compareTo is used to provide a total ordering of keys in the Sort/Shuffle/Merge stage.
- Returns -1 if implicit parameter should be order first.
- Returns 0 if they are equal.
- Returns 1 if explicit parameter should be ordered first.

## InputFormat Class

- Valid input files/directories exists.
- Partitions the input file into splits.
- Instantiates RecordReader for parsing records in the splits.
- Throws IOException

## Methods

```
public abstract List<InputSplit>
getSplits(JobContext)
```

```
public abstract RecordReader<K,V>
createRecordReader(InputSplit split,
TaskAttemptContext context)
```

## Common Implementations

- TextInputFormat – Single Line Record Text Files. **Terminated by newline characters.**
- SequenceFileInputFormat – Binary Files

## InputSplit Class

- Object that encapsulates a single file split.
- Logical representation of a subset of the data.
- Split size is defined by:**

$\max(\minSplitSize, \max(\maxSplitSize, blockSize))$

## Methods

```
public abstract long getLength()
```

```
public abstract String[] getLocations() – Gets
a list of host names where the split is located.
```

## Common Implementations

- FileSplit

## Split Versus Block Size

- Split Size is configurable in Hadoop and MapR.
- A split may be smaller, larger, or the same size as a block as defined by equation on the left.

## Record Boundaries – Two Possibilities

- Last Record Boundary Falls On Split Boundary** – Read whole first record in the next split.
- Last Record Boundary Falls in the Next Split** – Record reader reads the next split until the end of the record (i.e. first delimiter).

## RecordReader Interface

- Breaks up the data in an input split into Key-Value pairs**
- Handles incomplete records**
  - Discards first record in a split after the first split
  - Reads ahead to first delimiter in the next split (except the last split).**

## Methods

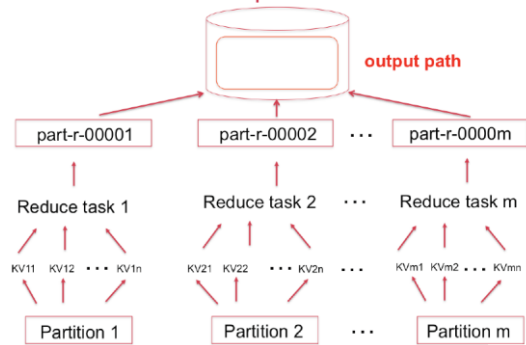
```
boolean next(K key, V value)
K createKey()
V createValue()
long getPos()
public void close()
float getProgress()
```

## Common Implementations

- LineRecordReader – Used for text files. Key is byte offset and text is the line.
- SequenceFileRecordReader – Binary input files

## Reducer Output Classes

### Describe Reducer Output Flow



### OutputFormat Class

- Valid output file specifications via method `checkOutputSpecs`.
- Provide `RecordWriter` to write output files.

#### Methods

```
public abstract RecordWriter<K,V>
getRecordWrite(TaskAttemptContext
context)
```

```
public abstract void
checkOutputSpecs(JobContext context)
```

```
public abstract OutputCommitter
getOutputCommitter(TaskAttemptContext
context)
```

#### Common Implementations

- `FileOutputFormat` – Wrapper of `OutputFormat`.
- `TextOutputFormat` – Plain text file.
- `NullOutputFormat` – Send all outputs to `/dev/null`
- `SequenceFileOutputFormat` – Binary Files

### RecordWriter Class

- Writes the key value pairs to the output files.
- Can automatically compress the output streams as they are written to disk.

#### Methods

```
public abstract void write(K key, V
value)
```

```
public abstract void
close(TaskAttemptContext)
```

#### Common Implementations

- `TextOutputFormat.LineRecordWriter` – Writes Key-Value pairs to plain text files.

### OutputCommitter Class

- Initializes the Job at job start (in `setupJob()`)
- Cleans up the job upon job completion (in `cleanJob()`).
- Sets up the task temporary outputs (in `setupTask()`)
- Checks whether a tasks needs to be committed (in `needsTaskCommit()`)
- Commit of the task output (in `commitTask()`)
- Discard the task commit (in `abortTask()`)

#### Common Implementations

- `FileOutputCommitter` – Commits files to job output directory.

### Mapper Class

- Based off `Java Generics` since key and value types are generic.
- Primary method to override is `map`.
- `Context` object is used to output to intermediate files.
- `run` method calls `setup`, `map`, and `cleanup`.
- `setup` is called before `map` and `cleanup` is called after `map`.

#### Methods

```
protected void cleanup(Context context)
```

```
protected void map(KEYIN key, VALUEIN
value, Context context)
```

```
void run(Context context)
```

```
protected void setup(Context context)
```

### Mapper and Reducer run Method

```
public void run(Context context){
    try{
        setup()
        while(context.nextKey()){
            map(context.getCurrentKey(),
                context.getCurrentValue(),
                context);
        }
    }
    finally{
        cleanup()
    }
}
```

### Reducer Class

- Based off `Java Generics` since key and value types are generic.
- If no `Reducer` class is specified, then `Mapper` outputs are sent directly as final outputs **after sorting by key**.
- Primary method to override is `reduce`.
- `Context` object is used to output to final files.
- `run` method calls `setup`, `reduce`, and `cleanup`.
- `setup` is called before `reduce` and `cleanup` is called after `reduce`. (Similar to `Mapper`)

#### Methods

```
protected void cleanup(Context context)
```

```
protected void map(KEYIN key,
Iterable<VALUEIN> values, Context context)
```

```
void run(Context context)
```

- `protected void setup(Context context)`

### Job Class

#### Job Methods

`void failTask(TaskAttemptID taskID)` – Indicate task with specified ID failed.

`String getJar()` – Gets the Job's JAR file pathname.

`boolean isComplete()` – Gets whether the job has completed.

`boolean isSuccessful()` – Returns whether the job completed successfully.

`void killJob()` – Kills the job.

`void killTask(TaskAttemptID taskID)` – Kills the task with the specified ID failed.

`float mapProgress()` – Gets progress of the map tasks. Between 0 and 1.

`float reduceProgress()` – Gets progress of the reduce tasks. Between 0 and 1.

#### Constructors

```
Job()
Job(Configuration conf)
Job(Configuration conf, String jobName)
```

#### Example Usage #1

```
Configuration conf = new Configuration();
Job job1 = new Job(conf, "Job1");
```

#### Example Usage #2

```
Job job2 = new Job(getConf(), "Job2");
```



More Job Class Methods		
<b>void setJarByClass(Class cls)</b> – Specifies the driver class.  <b>void setInputFormatClass(Class cls)</b> – Sets the InputFormat type for the job.  <b>void setMapperClass(Class cls)</b> – Sets the class type for the Mapper.  <b>void setMapOutputKeyClass(Class cls)</b> – Sets the class type for the Mapper output key(s).  <b>void setMapOutputValueClass(Class cls)</b> – Sets the class type for the Mapper output value(s).	<b>void setOutputFormatClass(Class cls)</b> – Sets the OutputFormat type for the job.  <b>void setReducerClass(Class cls)</b> – Sets the class type for the Reducer.  <b>void setOutputKeyClass(Class cls)</b> – Sets the class type for the Reducer output key(s) and the Mapper if setMapOutputKeyClass is not called.  <b>void setOutputValueClass(Class cls)</b> – Sets the class type for the Reducer output value(s) and the Mapper if setMapOutputValueClass is not called.	<b>void submit()</b> – Submit the job to the cluster and return immediately.  <b>void waitForCompletion(boolean verbose)</b> – Submit the job to the cluster and wait for it to finish. <b>Often called within System.exit() with a ternary operator.</b> <ul style="list-style-type: none"> <li>○ Returns “true” if the job succeeded.</li> </ul> <p><b>System.exit(job.waitForCompletion(True) ? 0 : 1);</b></p> <ul style="list-style-type: none"> <li>• <b>When configuring the job, almost all method names end in “Class”.</b></li> </ul>

Implementing the Driver	Job Configuration Code Example
<pre> public class MyDriver extends Configured     implements Tool{     public static void main(){         Configuration conf = new Configuration();         System.exit(ToolRunner.run(conf,                                    new MyDriver(),                                    args);     }      public int run(String[] args) throws Exception{         Job job = new Job(getConf(), "My Job");         ...         return job.waitForCompletion(True) ? 0 : 1;     } } </pre> <p><b>Use ToolRunner to execute driver code.</b></p>	<pre> Job job = new Job(getConf(), "myJob");  job.setJarByClass(MyDriver.class); job.setMapperClass(MyMapper.class); job.setReducerClass(MyReducer.class);  job.setOutputKeyClass(Text.class); job.setOutputValueClass(LongWritable.class);  job.setInputFormatClass(TextInputFormat.class); job.setOutputFormatClass(TextOutputFormat.class);  FileInputFormat.addInputPath(job, new Path(args[0])); FileOutputFormat.addOutputPath(job, new Path(args[1]));  System.exit(job.waitForCompletion(True) ? 0 : 1); </pre> <p><b>Drawback: Cannot be dynamically configured.</b></p>

Levels of MapReduce Configuration Priority		
<b>Highest Priority</b> <ol style="list-style-type: none"> <li>1. Driver Code</li> <li>2. Command Line Parameters</li> <li>3. Local XML Files</li> <li>4. Global XML Files (i.e. within the Global Map Reduce folder)</li> <li>5. Hadoop Framework Modifications</li> </ol> <b>Lowest Priority</b>		

# Lecture #07 – Managing, Monitoring, and Testing MapReduce Jobs

## MapReduce Counter Categories

<b>File System</b> – Total number of bytes written and read during a Hadoop job.	<b>Job</b> – Summary of task cardinality and CPU time.	<b>Framework</b> – Granular summaries of CPU and memory consumption, records read & written, and bytes read & written in each phase of MapReduce	<b>Custom</b> – Completely specific to the application.
--	--	--	---

## File System Counters

<b>FILE_BYTES_WRITTEN</b> – Total number of bytes written to the local file system. May occur during map, shuffle, or reduce phases.	<b>MAPRFS_BYTES_READ</b> – Total number of bytes read from MapR-FS.	<b>MAPRFS_BYTES_WRITTEN</b> – Total number of bytes written to MapR-FS.
--	---	---

## Job Counters

<b>DATA_LOCAL_MAPS</b> – Total number of map tasks executed on local data.	<b>FALLOW_SLOTS_MILLIS_MAPS</b> – Total time map tasks spend waiting after slots are reserved ( <b>pre-emption</b> )	<b>FALLOW_SLOTS_MILLIS_REDUCES</b> – Total time reduce tasks spend waiting after slots are reserved ( <b>pre-emption</b> )	<b>SLOTS_MILLI_MAPS</b> – Total time map tasks spent executing
<b>SLOTS_MILLI_REDUCES</b> – Total time reduce tasks spent executing	<b>TOTAL_LAUNCHED_MAPS</b> – Total number of map tasks launched, <b>including failed tasks.</b>	<b>TOTAL_LAUNCHED_REDUCES</b> – Total number of reduce tasks launched, <b>including failed tasks.</b>	

## Framework Counters

<b>COMBINE_INPUT_RECORDS</b> – Number of records <b>read</b> during the combine phase ( <b>if used – otherwise 0</b> )	<b>COMBINE_OUTPUT_RECORDS</b> – Number of records <b>written</b> during the combine phase ( <b>if used – otherwise 0</b> )	<b>CPU_MILLISECONDS</b> – Total CPU time spent on all tasks.	<b>GC_MILLISECONDS</b> – Total CPU time spent on <b>garbage collection.</b>
<b>MAP_INPUT_RECORDS</b> – Total number of records read in the Map phase.	<b>MAP_OUTPUT_RECORDS</b> – Total number of records written in the Map phase.	<b>PHYSICAL_MEMORY_BYTES</b> – Total physical memory consumed by all tasks.	<b>REDUCE_INPUT_GROUPS</b> – Total number of keys read in during reduce phase.
<b>REDUCE_INPUT_RECORDS</b> – Total number of records (i.e. <b>values</b> ) read in during reduce phase.	<b>REDUCE_OUTPUT_RECORDS</b> – Total number of records written during the reduce phase.	<b>REDUCE_SHUFFLE_BYTES</b> – Total number of bytes of output from map tasks copied during shuffle to reducers.	<b>SPILED_RECORDS</b> – Total number of records spilled to disk by all map and reduce tasks.
<b>SPLIT_RAW_BYTES</b> – Total number of bytes consumed for metadata ( <b>offset and length</b> ) during splits.	<b>VIRTUAL_MEMORY_BYTES</b> – Total number of virtual memory bytes consumed by tasks (RAM + swap)		

## Custom Counters

<b>Example Uses of Custom Counters</b> <ul style="list-style-type: none"> <li>Counting specific (e.g. bad) records</li> <li>Keeping track of outliers</li> <li>Summations</li> </ul>	<b>Two Ways to Define Custom Counters</b> <ol style="list-style-type: none"> <li><b>Enum</b> – <b>Compile time binding.</b></li> <li><b>Context Variable</b> – <b>Run time binding.</b></li> </ol> <p><b>Context based custom counters can be stored in groups.</b></p>	<b>Framework</b> – The JobTracker (MRV1)/Resource Manager (MRV2) <b>store custom counters in memory.</b> Recommended to keep the number of custom counters below 100.	<b>Example Syntax</b> <pre>context.getCounter("CounterGroupName", "CounterName").increment(1);</pre> <p>Counters do not need to be declared or initialized. They are made the first time it is incremented.</p>
--	---	---	---

## Selecting MapReduce Version when MRV1 and MRV2 are on the Same System

	Command Line	Environment Variables	Client	Cluster Wide
<b>MRV1</b>	hadoop1, hadoop -classic	MAPR_MAPREDUCE_MODE=classic	default_mode=classic	maprccli cluster mapreduce set –mode classic
<b>MRV2</b>	hadoop2, mapred, yarn	MAPR_MAPREDUCE_MODE=yarn	default_mode=yarn	maprccli cluster mapreduce set –mode yarn

## Selecting MapReduce Version when MRV1 and MRV2 are on the Same System

	Command Line	Environment Variables	Command Line Interface
<b>MRV1</b>	JobTracker, TaskTracker ( <b>Separate Web UIs</b> )	<b>Uses</b> MapR Metrics Database	hadoop job ... <b>Used to monitor job status</b>
<b>MRV2</b>	HistoryServer, ResourceManager, NodeManager ( <b>Only HistoryServer runs a Web UI</b> )	<b>No</b> MapR Metrics Database	mapred job ... yarn job ... <b>Used to monitor job status</b>

## Job Information

<b>Job ID</b> – Every job has an ID in the format <code>job_yyyymmddhhMM_cccc</code> which align to the date/time when the job started and a counter for that minute.	<b>Information Stored About a Job</b>		<b>Get List of Currently Running Jobs</b> hadoop job -list
	<ul style="list-style-type: none"> <li>Submit Time</li> <li>Start Time</li> <li>End Time</li> <li>Queue</li> </ul>	<ul style="list-style-type: none"> <li>User</li> <li>Counter Information</li> <li>Configuration Settings</li> </ul>	<b>Get Individual Job Status</b> hadoop job -status job_yyyymmddhhMM_cccc
			<b>Kill a Currently Running Job</b> hadoop job -kill job_yyyymmddhhMM_cccc <b>Only this should be used to kill MapReduce jobs.</b> <b>If OS level kill commands are used, TaskTracker will relaunch them.</b>

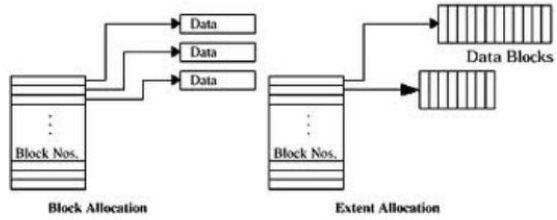
## Modifying Job Priority in MRV1

<p><b>Using the Job API</b></p> <pre>Configuration conf = new Configuration(); conf.set("mapred.job.priority", "VERY_HIGH"); Job job = new Job(conf, "JobName");</pre>	<p><b>XML Configuration</b></p> <pre>&lt;configuration&gt;   &lt;property&gt;     &lt;name&gt;mapred.job.priority&lt;/name&gt;     &lt;value&gt;HIGH&lt;/value&gt;   &lt;/property&gt; &lt;/configuration&gt;</pre>	<p><b>Command Line at Job Start</b></p> <pre>hadoop jar -D mapred.job.priority=VERY_LOW</pre> <p><b>Command Line While Job Is Running</b></p> <pre>hadoop job -set-priority job_yyyymmddhhMM_cccc VERY_LOW</pre>
--	---	--

<p><b>Label Based Scheduling</b></p> <ul style="list-style-type: none"> <li>Only available in MapR.</li> <li>Used to override default scheduling.</li> <li><b>Nodes in the cluster are associated with labels.</b></li> <li><b>When jobs are submitted with a label, the job is only executed on those nodes associated with that label.</b></li> </ul> <p><b>Starting a Job with a Label</b></p> <pre>hadoop jar -D mapred.job.label=LabelName ...</pre> <p><b>"LabelName" must exist or the job will hang.</b></p> <p><b>Showing All Job Labels</b></p> <pre>hadoop job -showlabels</pre>	<p><b>Apache Commons Logging (JCL)</b></p> <ul style="list-style-type: none"> <li><b>JCL</b> – Pluggable logging interface for Apache Programs written in Java.</li> <li><b>Examples:</b> Log4j, Avalon LogKit</li> <li>Logging is at multiple levels so users can specify what level of logging they want to use.</li> </ul> <p><b>Example:</b></p> <pre>private static Log log = LogFactory.getLog(MyClass.class); ... public void map(Key key, Value value){   ...   log.debug("Debug level logging");   ...   log.error("Error level logging");   ... }</pre>
---	---

<p><b>Getting Job History</b></p> <ul style="list-style-type: none"> <li>Job history is stored in a folder.</li> <li>This history can be viewed from the command line:</li> </ul> <p><b>Job History Command Line Syntax:</b></p> <pre>hadoop job -history   FolderName/</pre>	<p><b>MRUnit</b></p> <ul style="list-style-type: none"> <li>Developed at <b>Cloudera</b>.</li> <li>Based off <b>JUnit</b>.</li> <li>Uses <b>LocalJobRunner</b> to execute code.</li> <li>Used to perform unit testing.</li> <li>Code to be tested does not need to be modified for testing.</li> <li>If program output matches expected output, unit test exits silently. Otherwise, it flags an error.</li> </ul>	<pre>public class MapReduceTestClass{   private static Driver&lt;KEY1, VALUE1, KEY2, VALUE2&gt; driver;    @Before // Tells MRUnit to run this before executing test.   private static setUp(){     Mapper mapper = new Mapper();     driver = Driver.newMapDriver(mapper);   }    @Test // Indicates method to run when performing a test.   private static void TestMapper() throws IOException, InterruptedException{     KEY1 inKey = ...;     VALUE1 inValue = ...;     KEY2 outKey = ...;     VALUE2 outValue = ...;     driver.withInput(inKey, inValue).withOutput(outKey, outValue).runTest();   }    public static void main(){     setUp();     try{ TestMapper() };     catch(Exception e){ System.err.println("exception: " + e.toString()); }     return;   } }</pre>
---	--	---

## Miscellaneous



***block vs extent-based allocation***