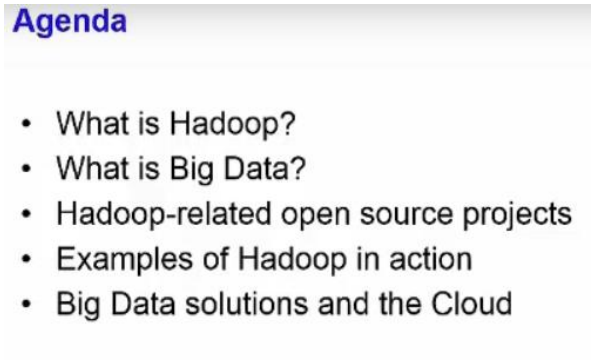
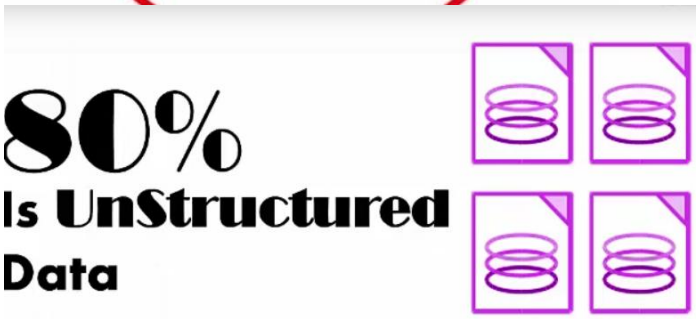
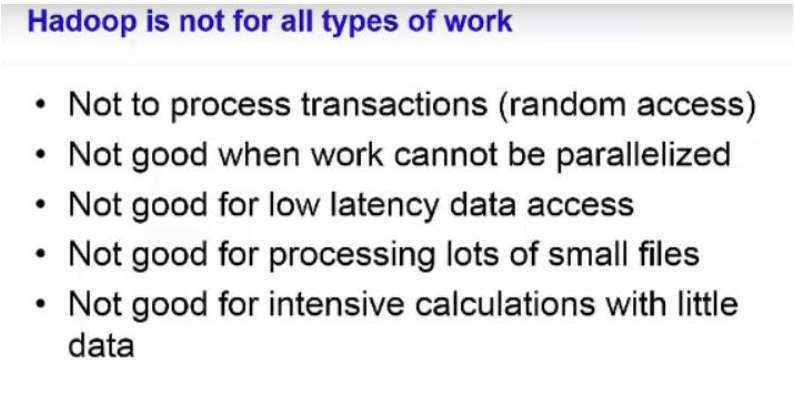
MODULE 1 – INTRODUCTION TO HADOOP

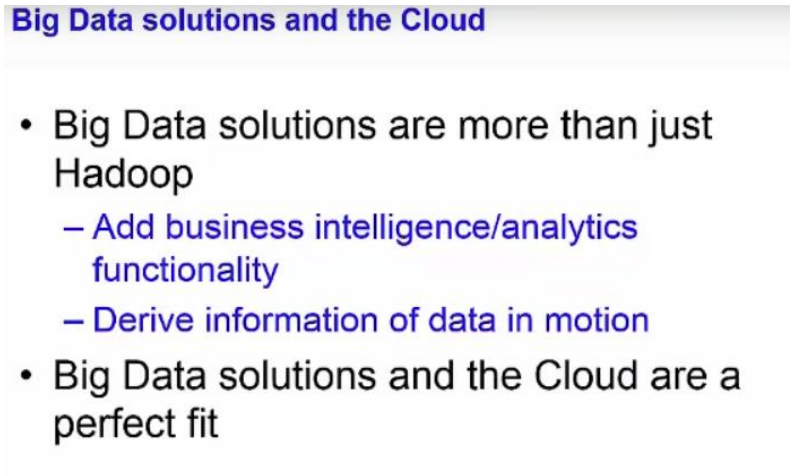


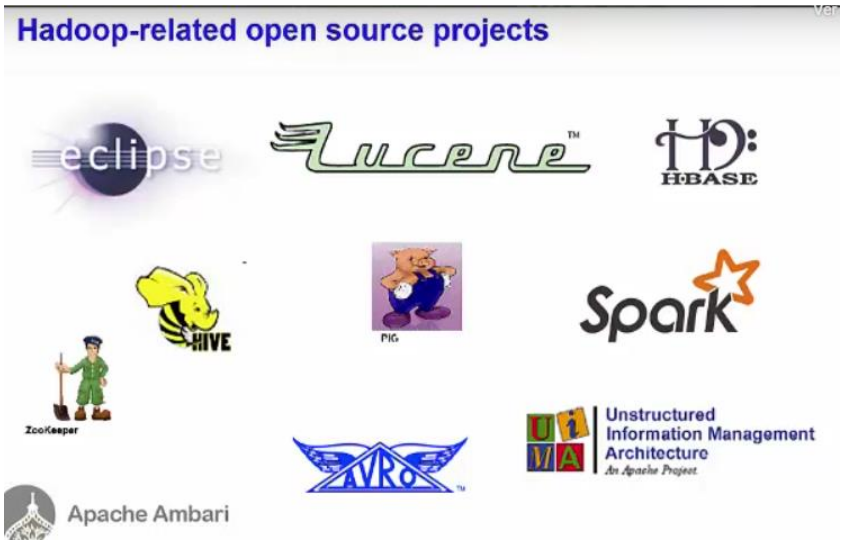


Hello everyone and welcome to Hadoop Fundamentals What is Hadoop. My name is Asma Desai and I will be covering this topic.
In this video we will explain what is Hadoop and what is Big Data. We will define some Hadoop-related open source projects and give some examples of Hadoop in action. Finally we will end off with some Big Data solutions and the Cloud.
Imagine this scenario: You have 1GB of data that you need to process.
The data is stored in a relational database in your desktop computer which has no problem handling the load
Then your company starts growing very quickly, and that data grows to 10GB,
then 100GB,
And you start to reach the limits of your current desktop computer.
so what do you do? you scale up by investing in a larger computer, and you are then OK for a few more months
When your data grows from 1 TB to 10TB, and then 100TB,
you are again quickly approaching the limits of that computer.
Moreover, you are now asked to feed your application with unstructured data coming from sources like Facebook, Twitter, RFID readers, sensors, and so on.
Your management wants to derive information from both the relational data and the unstructured data and wants this information as soon as possible

What should you do? Hadoop may be the answer
What is Hadoop? Hadoop is an open source project of the Apache
Foundation. It is a framework written in Java originally
developed by Doug Cutting who named it after his son's toy elephant.
Hadoop uses Google's MapReduce technology as its foundation.
It is optimized to handle massive quantities of data which could be structured, unstructured
or semi-structured, using commodity hardware, that is, relatively inexpensive computers.
This massive parallel processing is done with great performance. However, it is a batch
operation handling massive amounts of data, so the response time is not immediate. Currently,
in place updates are not possible in Hadoop, but appends to existing data is supported.
Now, what's the value of a system if the information it stores or retrieves is not
consistent? Hadoop replicates its data across different
computers, so that if one goes down, the data is processed on one of the replicated computers.
Hadoop is not suitable for OnLine Transaction Processing workloads where data is randomly
accessed on structured data like a relational database.
Also, Hadoop is not suitable for OnLine Analytical Processing or Decision Support System workloads
where data is sequentially accessed on structured data like a relational database, to generate
reports that provide business intelligence. As of Hadoop version 2.6, updates are not
possible, but appends are possible. Hadoop is used for Big Data. It complements
OnLine Transaction Processing and OnLine Analytical Processing.
It is NOT a replacement for a relational database system.
So, what is Big Data? With all the devices available today to collect
data, such as RFID readers, microphones, cameras, sensors, and so on, we are seeing an explosion
in data being collected worldwide. Big Data is a term used to describe large
collections of data (also known as datasets) that may be unstructured, and grow so large
and quickly that it is difficult to manage with a regular database or statistical tools.
In terms of numbers, what are we looking at? How BIG is "big data"?
Well there are more than 3.2 billion internet users, and
Active cell phones have surpassed 7.6 billion. There are now more in-use cell phones than
there are people on the planet (7.4 billion).
Twitter processes 7TB of data ever day,
and 600TB of data is processed by Facebook every day.
Interestingly, about 80% of this data is unstructured.
With this massive amount of data, businesses need fast, reliable, deeper data insight.
Therefore, Big Data solutions based on Hadoop and other analytics software are becoming
more and more relevant
This lesson continues in the next video







This is a list of some other open source project related to Hadoop:
- Eclipse is a popular IDE donated by IBM to the open-source community
- Lucene is a text search engine library written in Java
- Hbase is a Hadoop database - Hive provides data warehousing tools to extract, transform and load (ETL) data, and query this data stored in Hadoop files
- Pig is a high level language that generates MapReduce code to analyze large data sets.
- Spark is a cluster computing framework -ZooKeeper is a centralized configuration service and naming registry for large distributed systems
-Ambari manages and monitors Hadoop clusters through an intuitive web UI
- Avro is a data serialization system - UIMA is the architecture for the development, discovery, composition and deployment for the analysis of unstructured data
- Yarn is a large-scale operating system for big data applications
- Mapreduce is a software framework for easily writing applications which processes vast amounts of data

Let's now talk about examples of Hadoop in action.

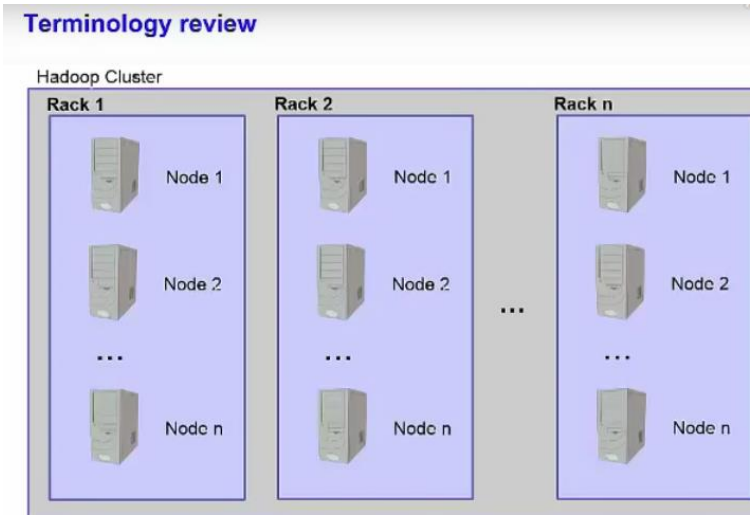
Early in 2011, Watson, a super computer developed by IBM competed in the popular Question and Answer show Jeopardy!. In that contest, Watson was successful in beating the two most winning Jeopardy players. Approximately 200 million pages of text were input using Hadoop to distribute the workload for loading this information into memory. Once this information was loaded, Watson used other technologies for advanced search and analysis.

In the telecommunication industry we have China Mobile, a company that built a Hadoop cluster to perform data mining on Call Data Records. China Mobile was producing 5-8 TB of these records daily. By using a Hadoop-based system they were able to process 10 times as much data as when using their old system, and at one fifth the cost.

In the media we have the New York Times which wanted to host on their website all public domain articles from 1851 to 1922. They converted articles from 11 million image files (4TB) to 1.5TB of PDF documents. This was implemented by one employee who ran a job in 24 hours on a 100-instance Amazon EC2 Hadoop cluster at a very low cost.
In the technology field we again have IBM with IBM ES2, and enterprise search technology based on Hadoop, Nutch, Lucene and Jaql. ES2 is designed to address unique challenges of enterprise search such as: - The Use of enterprise-specific vocabulary, abbreviations and acronyms ES2 can perform mining tasks to build Acronym libraries, Regular expression patterns, and Geo-classification rules.
There are also many internet or social network companies using Hadoop such as: Yahoo, Facebook, Amazon, eBay, Twitter, StumbleUpon, Rackspace, Ning, AOL, etc.
Yahoo of course is the largest production user with an application running a Hadoop cluster consisting of about 10,000 Linux machines. Yahoo is also the largest contributor to the Hadoop open source project.
Now, Hadoop is not a magic bullet that solves all kinds of problems.
Hadoop is not good to process transactions due to its lack random access.
It is not good when the work cannot be parallelized or when there are dependencies within the data, that is, record one must be processed before record two.
It is not good for low latency data access. Not good for processing lots of small files although there is work being done in this area, for example, IBM's Adaptive MapReduce.
And it is not good for intensive calculations with little data.
Now let's move on, and talk about Big Data solutions.
Big Data solutions are more than just Hadoop. They can integrate analytic solutions to the mix to derive valuable information that can combine structured legacy data with new unstructured data.
Big data solutions may also be used to derive information from data in motion, for example, IBM has a product called InfoSphere Streams that can be used to quickly determine customer sentiment for a new product based on Facebook or Twitter comments.
Finally we would like to end this presentation with one final thought: Cloud computing has gained a tremendous track in the past few years, and it is a perfect fit for Big Data solutions. Using the cloud, a Hadoop cluster can be setup in minutes, on demand, and it can run for as long as needed without having to pay for more than what is used.
This is the end of this video. Thank you for watching. Please continue with the other units in this course.
Here is a list of trademarks that may have been used in this presentation.

MODULE - 2 HADOOP ARCHITECTURE & HDFS

HADOOP ARCHITECTURE



Pre Hadoop 2.2 architecture

- Two main components
  - Distributed File System
    - Hadoop Distributed File System (HDFS)
    - IBM Spectrum Scale
  - MapReduce Engine
    - Framework for performing calculations on the data in the file system
    - Has a built-in resource manager and scheduler

Hadoop Distributed File System (HDFS)

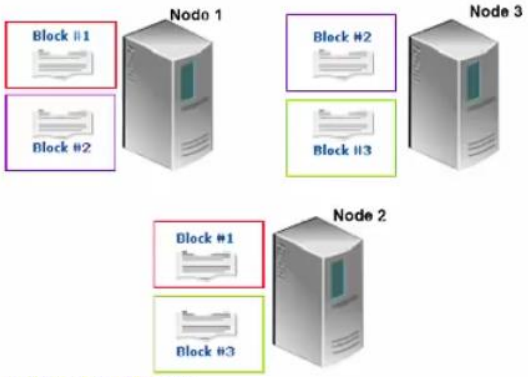
- HDFS runs on top of the existing file system
  - Not POSIX compliant
  - Designed to tolerate high component failure rate
    - Reliability is through replication
- Designed to handle very large files
  - Large streaming data access patterns
    - No random access
- Uses blocks to store a file or parts of a file

HDFS file blocks

- Not the same as the operating system's file blocks
  - –HDFS book made up of multiple operating system blocks
- Default for Hadoop is 64MB
  - –Recommended is 128MB (this is the BigInsights default)
- Size of a file can be larger than any single disk in the cluster
  - –Blocks for a single file are spread across multiple nodes in the cluster
- If a chunk of the file is smaller than the HDFS block size
  - –Only the needed space is used
- Blocks work well with replication

HDFS - Replication

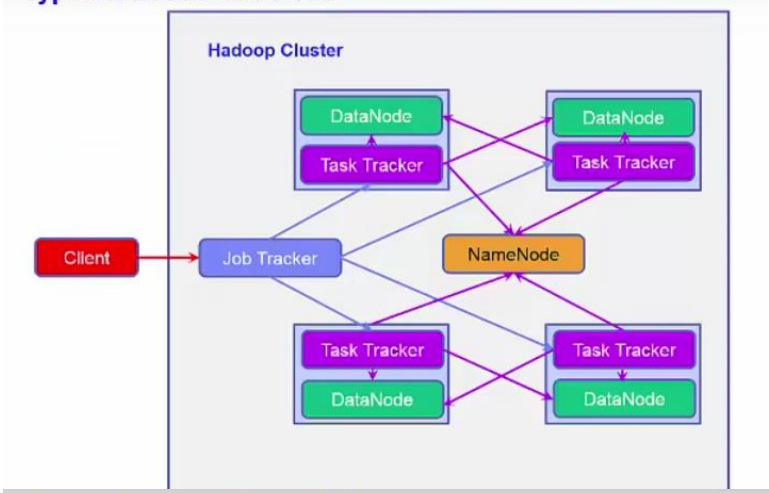
- Blocks with data are replicated to multiple nodes
- Allows for node failure without data loss



MapReduce framework

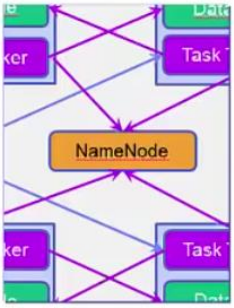
- Based on technology from Google
- Processes huge datasets for certain kinds of distributable using a large number of nodes
- A MapReduce program consists of map and reduce func
- Allows for distributed processing of the map and reduce
  - Tasks run in parallel

Types of nodes - overview



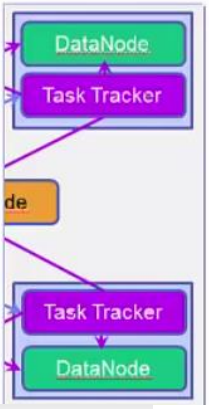
Types of nodes - NameNode

- Only one per Hadoop cluster
- Manages the file system namespace and metadata
  - Data does not go through the NameNode
  - Data is not stored on the NameNode
- Single point of failure
  - Good idea to mirror the NameNode
  - Do not use inexpensive, commodity hardware



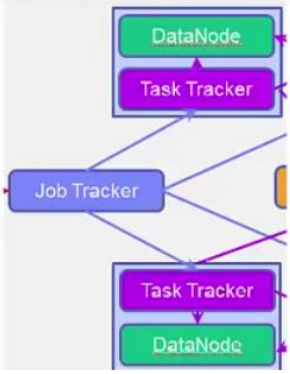
Types of nodes - DataNode

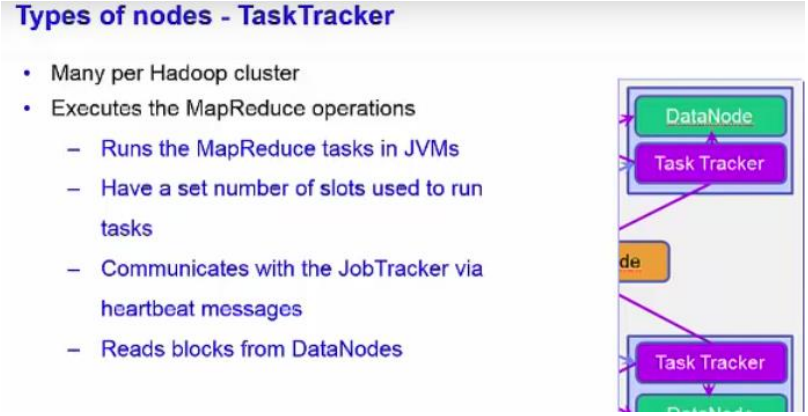
- Many per Hadoop cluster
- Blocks from different files can be stored on the same DataNode
- Manages blocks with data and serves them to clients
- Periodically reports to NameNode the list of blocks it stores
- Suitable for inexpensive, commodity hardware



Types of nodes - JobTracker

- Manages the MapReduce jobs in the cluster
- One per Hadoop cluster
- Receives job requests submitted by the client
- Schedules and monitors MapReduce jobs on TaskTrackers
  - Attempts to direct a task to the TaskTracker where the data





Welcome to the unit of Hadoop Fundamentals on Hadoop architecture.
I will begin with a terminology review and then cover the major components of Hadoop. We will see what types of nodes can exist in a Hadoop cluster and talk about how Hadoop uses replication to lessen data loss. Finally I will explain an important feature of Hadoop called "rack awareness" or "network topology awareness".
Before we examine Hadoop components and architecture, let's review some of the terms that are used in this discussion. A node is simply a computer. This is typically non-enterprise, commodity hardware for nodes that contain data. So in this example, we have Node 1. Then we can add more nodes, such as Node 2, Node 3, and so on.
This would be called a rack. A rack is a collection of 30 or 40 nodes that are physically stored close together and are all connected to the same network switch.
Network bandwidth between any two nodes in the same rack is greater than bandwidth between two nodes on different racks. You will see later how Hadoop takes advantage of this fact. A Hadoop Cluster (or just cluster from now on) is a collection of racks
Let us now examine the pre-Hadoop 2.2 architecture. Hadoop has two major components:
- the distributed filesystem component, the main example of which is the Hadoop Distributed File System, though other file systems, such as IBM Spectrum Scale, are supported.
- the MapReduce component, which is a framework for performing calculations on the data in the distributed file system. Pre-Hadoop 2.2 MapReduce is referred to as MapReduce V1 and has its own built-in resource manager and schedule. This unit covers the Hadoop Distributed File System and MapReduce is covered separately.
Let's now examine the Hadoop distributed file system - HDFS
HDFS runs on top of the existing file systems on each node in a Hadoop cluster. It is not POSIX compliant. It is designed to tolerate high component failure rate through replication of the data. Hadoop works best with very large files. The larger the file, the less time Hadoop spends seeking for the next data location on disk, the more time Hadoop runs at the limit of the bandwidth of your disks.
Seeks are generally expensive operations that are useful when they only need to analyze a small subset of your dataset. Since Hadoop is designed to run over your entire dataset, it is best to minimize seeks by using large files. Hadoop is designed for streaming or sequential data access rather than random access. Sequential data access means fewer seeks, since Hadoop only seeks to the beginning of each block and begins reading sequentially from there. Hadoop uses blocks to store a file or parts of a file. This is shown in the figure.
Let us now examine file blocks in more detail. A Hadoop block is a file on the underlying filesystem. Since the underlying filesystem stores files as blocks, one Hadoop block may consist of many blocks in the underlying file system. Blocks are large. They default to 64 megabytes each and most systems run with block sizes of 128 megabytes or larger. Blocks have several advantages: Firstly, they are fixed in size. This makes it easy to calculate how many can fit on a disk.
Secondly, by being made up of blocks that can be spread over multiple nodes, a file can be larger than any single disk in the cluster. HDFS blocks also don't waste space.

If a file is not an even multiple of the block size, the block containing the remainder does not occupy the space of an entire block. As shown in the figure, a 450 megabyte file with a 128 megabyte block size consumes four blocks, but the fourth block does not consume a full 128 megabytes. Finally, blocks fit well with replication,
which allows HDFS to be fault tolerant and available on commodity hardware.
As shown in the figure: Each block is replicated to multiple nodes. For example, block 1 is stored on node 1 and node 2. Block 2 is stored on node 1 and node 3. And block 3 is stored on node 2 and node 3. This allows for node failure without data loss. If node 1 crashes, node 2 still runs and has block 1's data. In this example, we are only replicating data across two nodes, but you can set replication to be across many more nodes by changing Hadoop's configuration or even setting the replication factor for each individual file.
The second major component of Hadoop, described in detail in another lecture, is the MapReduce component. HDFS was based on a paper Google published about their Google File System, Hadoop's MapReduce is inspired by a paper Google published on the MapReduce technology. It is designed to process huge datasets for certain kinds of distributable problems using a large number of nodes. A MapReduce program consists of two types of transformations that can be applied to data any number of times - a map transformation and a reduce transformation. A MapReduce job is an executing MapReduce program that is divided into map tasks that run in parallel with each other and reduce tasks that run in parallel with each other.
Let us examine the main types of nodes in pre-Hadoop 2.2. They are classified as HDFS or MapReduce V1 nodes. For HDFS nodes we have the NameNode, and the DataNodes. For MapReduce V1 nodes we have the JobTracker and the TaskTracker nodes. Each of these is discussed in more detail later in this presentation. There are other HDFS nodes such as the Secondary NameNode, Checkpoint node, and Backup node that are not discussed in this course. This diagram shows some of the communication paths between the different types of nodes on the system. A client is shown as communicating with a JobTracker. It can also communicate with the NameNode and with any DataNode.
There is only one NameNode in the cluster. While the data that makes up a file is stored in blocks at the data nodes, the metadata for a file is stored at the NameNode. The NameNode is also responsible for the filesystem namespace. To compensate for the fact that there is only one NameNode, one should configure the NameNode to write a copy of its state information to multiple locations, such as a local disk and an NFS mount. If there is one node in the cluster to spend money on the best enterprise hardware for maximum reliability, it is the NameNode. The NameNode should also have as much RAM as possible because it keeps the entire filesystem metadata in memory.
A typical HDFS cluster has many DataNodes. DataNodes store the blocks of data and blocks from different files can be stored on the same DataNode. When a client requests a file, the client finds out from the NameNode which DataNodes stored the blocks that make up that file and the client directly reads the blocks from the individual DataNodes. Each DataNode also reports to the NameNode periodically with the list of blocks it stores. DataNodes do not require expensive enterprise hardware or replication at the hardware layer. The DataNodes are designed to run on commodity hardware and replication is provided at the software layer.
A JobTracker node manages MapReduce V1 jobs. There is only one of these on the cluster. It receives jobs submitted by clients. It schedules the Map tasks and Reduce tasks on the appropriate TaskTrackers, that is where the data resides, in a rack-aware manner and it monitors for any failing tasks that need to be rescheduled on a different TaskTracker. To achieve the parallelism for your map and reduce tasks, there are many TaskTrackers in a Hadoop cluster. Each TaskTracker spawns Java Virtual Machines to run your map or reduce task. It communicates with the JobTracker and reads blocks from DataNodes.
This lesson continues in the next video.

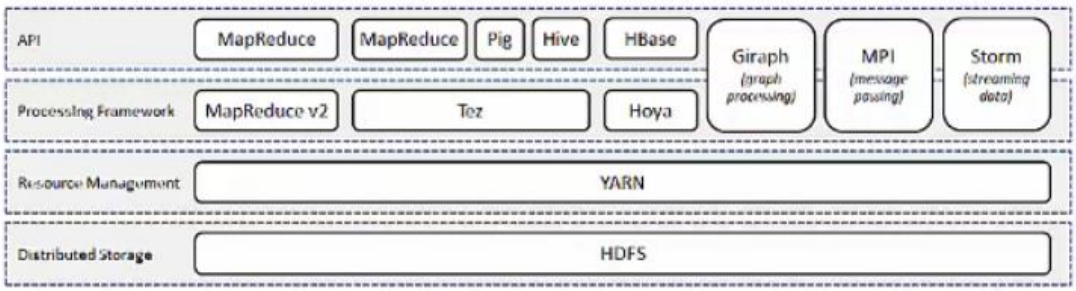
Hadoop 2.2 architecture

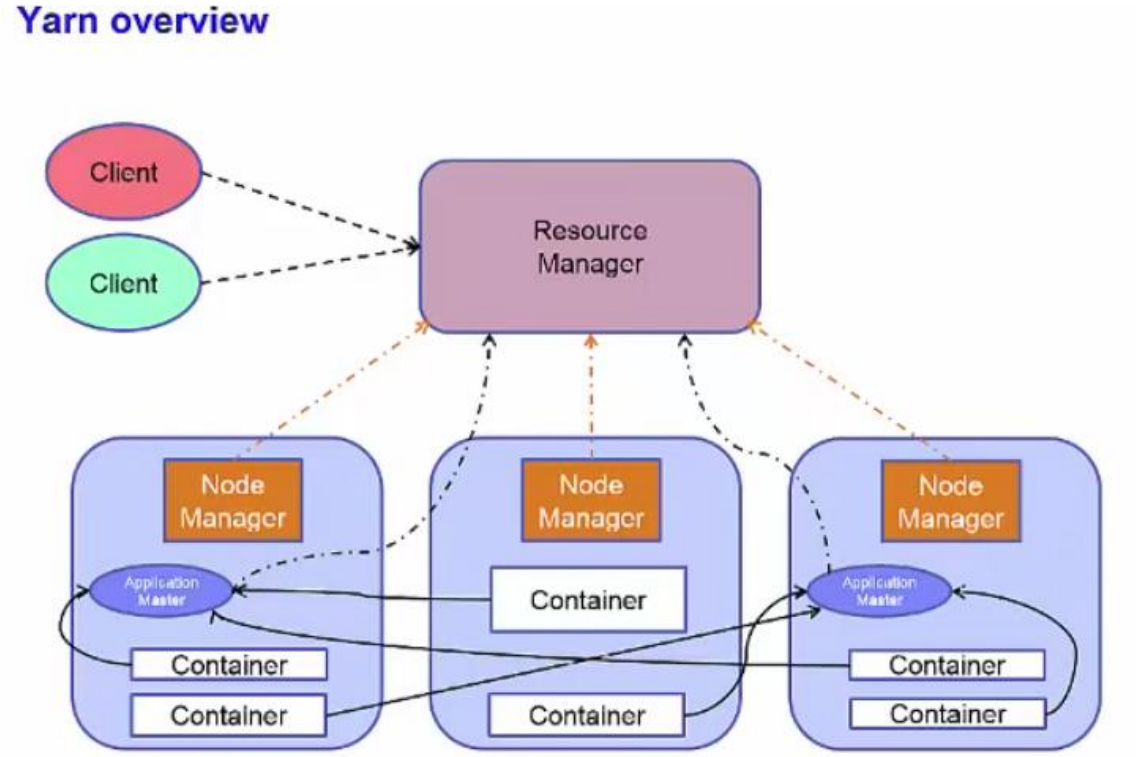
- Provides YARN
  - Referred to as MapReduce V2
  - Resource manager and scheduler external to any framework
  - DataNodes still exist
  - JobTracker and TaskTrackers no longer exist

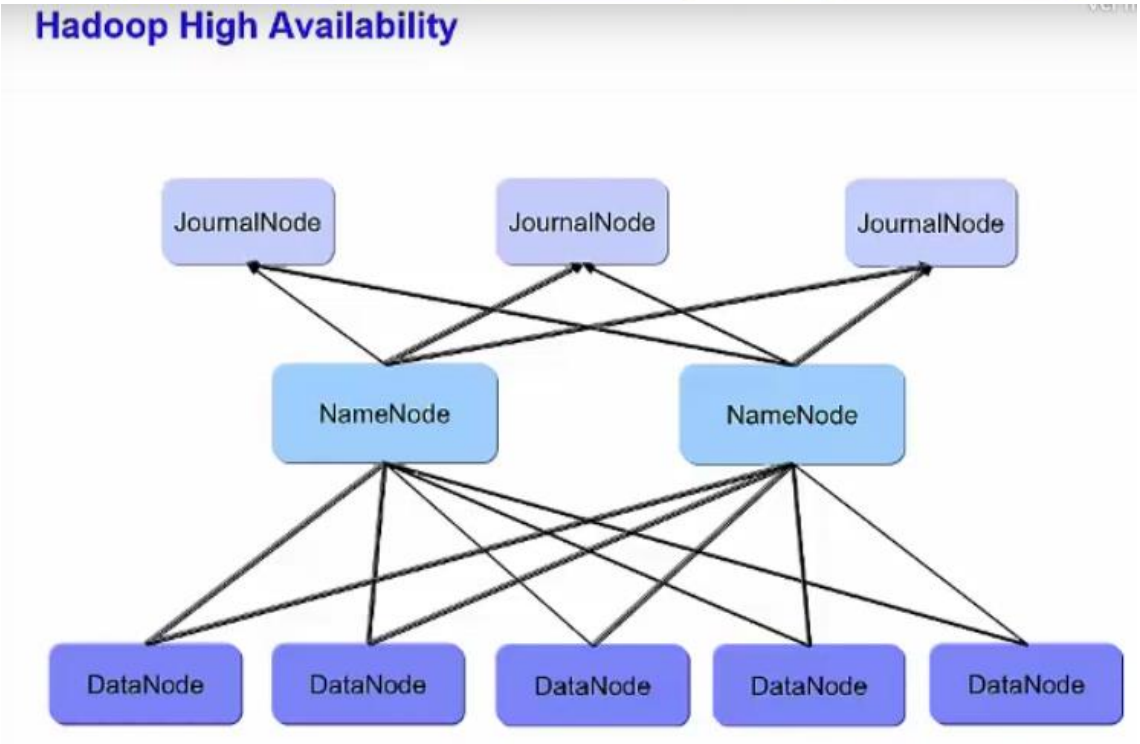
- Not a requirement to run YARN with Hadoop 2.2
  - Still supports MapReduce V1

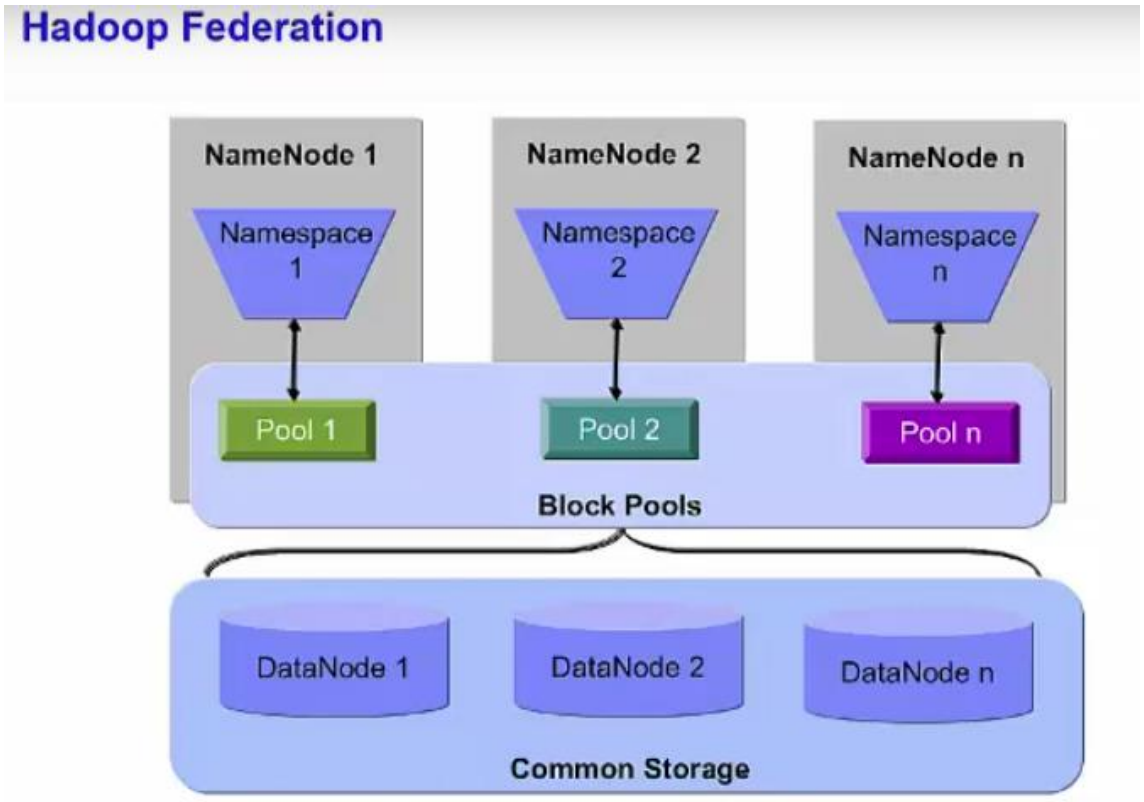
YARN

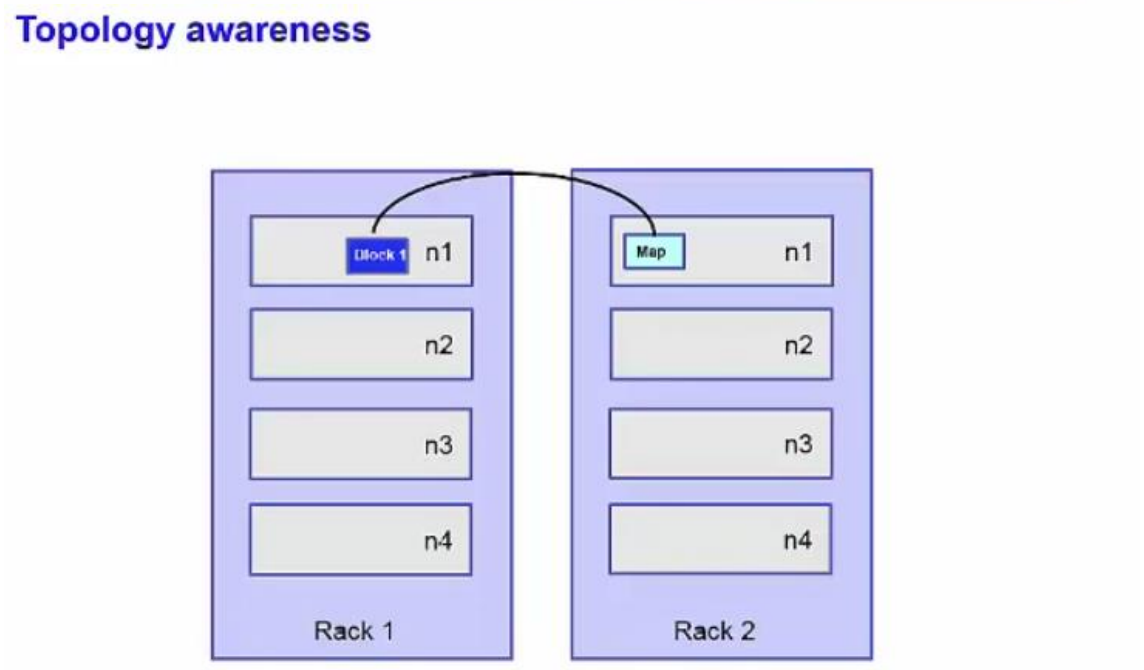
- **Two main ideas**
  - Provide generic scheduling and resource management
    - Support more than just MapReduce
    - Support more than just batch processing
  - More efficient scheduling and workload management
    - No more balancing between map slots and reduce slots!



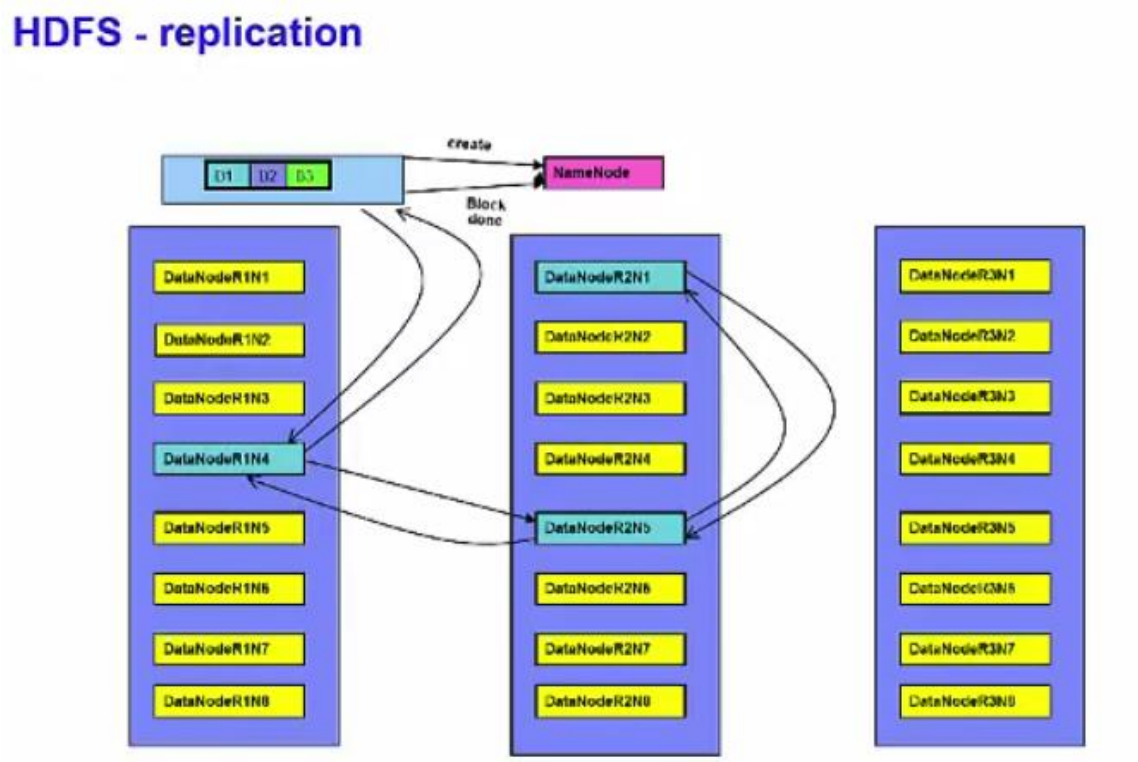

Yarn overview
Client
Client
Resource Manager
Node Manager
Node Manager
Node Manager
Application Master
Application Master
Container
Container
Container
Container
Container
Container


Hadoop High Availability
JournalNode
JournalNode
JournalNode
NameNode
NameNode
DataNode
DataNode
DataNode
DataNode
DataNode





ıe administrator defines the topology using the *topology.script.file.name* property in *core-site.xml*



Hadoop 2.2 brought about architectural changes to MapReduce. As Hadoop has matured, people have found that it can be used for more than running MapReduce jobs. But to keep each new framework from having its own resource manager and scheduler, that would compete with the other framework resource managers and schedulers, it was decided to have the recourse manager and schedulers to be external to any framework. This new architecture is called YARN. (Yet Another Resource Negotiator) . You still have DataNodes but there are no longer TaskTrackers and the JobTracker. You are not required to run YARN with Hadoop 2.2. as MapReduce V1 is still supported. There are two main ideas with YARN.

Provide generic scheduling and resource management. This way Hadoop can support more than just MapReduce. The other is to try to provide a more efficient

scheduling and workload management. With MapReduce V1, the administrator had to define how many

map slots and how many reduce slots there were on each node. Since the hardware capabilities for each node in a Hadoop cluster can vary, for performance reasons, you might want to limit the number of tasks on certain nodes. With YARN, this is no longer required.

With YARN, the resource manager is aware of the capabilities of each node via communication with the NodeManager running on each node. When an application gets invoked , an Application Master gets started. The Application Master is then responsible for negotiating resources from the ResourceManager. These resources are assigned to Containers on each slave-node and you can think that tasks then run in Containers. With this architecture, you are no longer forced into a one size fits all. The NameNode is a single point of failure.

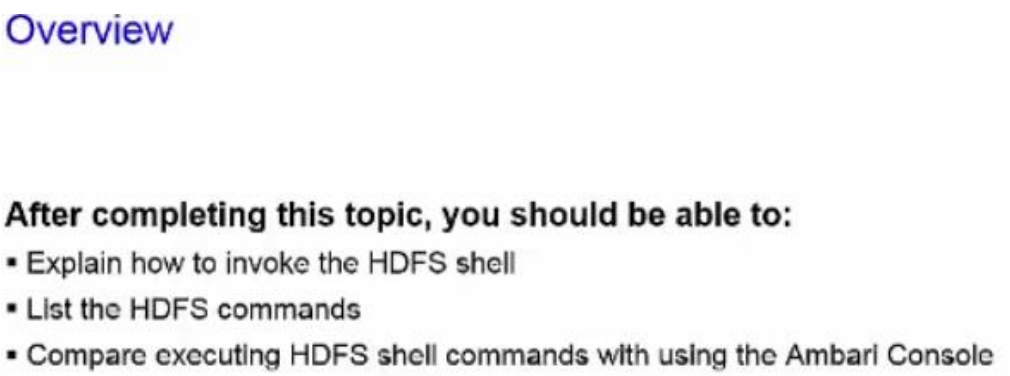
Is there anything that can be done about that? Hadoop now supports high availability. In this setup, there are now two NameNodes, one active and one standby.

Also, now there are JournalNodes. There must be at least three and there must be an odd number. Only one of the NameNodes can be active at a time. It is the JournalNodes, working together , that decide which of the NameNodes is to be the active one and if the active NameNode has been lost and whether the backup NameNode should take over.

The NameNode loads the metadata for the file system into memory. This is the reason that we said that NameNodes needed large amounts of RAM. But you are going to be limited at some point when you use this vertical growth model. Hadoop Federation allows you to grow your system horizontally. This setup also utilizes multiple NameNodes. But they act independently. However, they do all share all of the DataNodes. Each NameNode has its

own namespace and therefore has control over its own set of files. For example, one file that has blocks on DataNode 1 and DataNode 2 might be owned by NameNode 1. NameNode 2 might own a file that has blocks on DataNode 2 and DataNode 3. And NameNode 3 might have a file with blocks on all three DataNodes.
Hadoop has awareness of the topology of the network. This allows it to optimize where it sends the computations to be applied to the data. Placing the work as close as possible to the data it operates on maximizes the bandwidth available for reading the data. In the diagram, the data we wish to apply processing to is block B1, the dark blue rectangle on node n1 on rack 1. When deciding which TaskTracker should receive a MapTask that reads data from B1, the best option is to choose the TaskTracker that runs on the same node as the data. If we can't place the computation on the same node, our next best option is to place it on a node in the same rack as the data. The worst case that Hadoop currently supports is when the computation must be processed from a node in a different rack than the data. When rack-awareness is configured for your cluster, Hadoop will always try to run the task on the TaskTracker node with the highest bandwidth access to the data.
Let us walk through an example of how a file gets written to HDFS. First, the client submits a "create" request to the NameNode. The NameNode checks that the file does not already exist and the client has permission to write the file. If that succeeds, the NameNode determines the DataNode to where the first block is to be written. If the client is running on a DataNode, it will try to place it there. Otherwise, it chooses DataNode at random. By default, data is replicated to two other places in the cluster. A pipeline is built between the three DataNodes that make up the pipeline. The second DataNode is a randomly chosen node on a rack other than that of the first replica of the block. This is to increase redundancy. The final replica is placed on a random node within the same rack as the second replica. The data is piped from the second DataNode to the third. To ensure the write was successful before continuing, acknowledgment packets are sent from the third DataNode to the second, From the second DataNode to the first And from the first DataNode to the client This process occurs for each of the blocks that makes up the file Notice that, for every block, by default, there is a replica on at least two racks. When the client is done writing to the DataNode pipeline and has received acknowledgements, it tells the NameNode that the write has completed. The NameNode then checks that the blocks are at least minimally replicated before responding.
This lesson continues with the next video in this unit.

HDFS



HDFS file command interface

- The FileSystem (FS) shell is invoked by

```
hdfs dfs <args>
```

- Example to list the current directory in HDFS

```
hdfs dfs -ls .
```

HDFS file command interface

- All FS shell commands take path URIs as arguments

```
scheme://authority/path
```

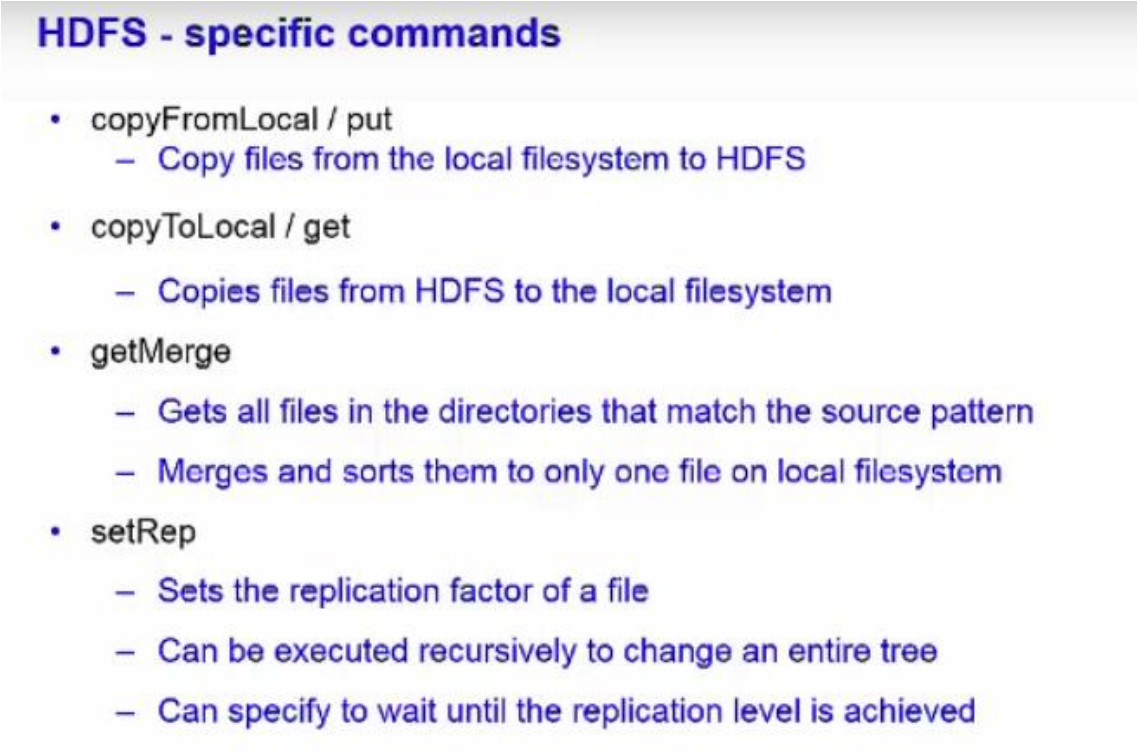
- Scheme
  - Scheme for HDFS is *hdfs*
  - Scheme for the local filesystem is *file*

```
hdfs dfs -cp
        file:///sampleData/spark/myfile.txt
        hdfs://rvm.svl.ibm.com:8020/user/spark/test/myfile.txt
```

- Scheme and authority are optional
  - Defaults are taken from the *core-site.xml* configuration file
- Most of the FS shell commands behave like corresponding UNIX commands

HDFS file commands

- A number of POSIX-like commands
  - cat, chgrp, chmod, chown, cp, du, ls, mkdir, mv, rm, stat, tail
- Some HDFS-specific commands
  - copyFromLocal, copyToLocal, get, getmerge, put, setrep



Welcome to HDFS command line interface.
In this presentation, I will cover the general usage of the HDFS command line interface and commands specific to HDFS. Other commands should be familiar to anyone with UNIX experience and will not be covered.
The HDFS can be manipulated through a Java API or through a command line interface. All commands for manipulating HDFS through Hadoop's command line interface begin with "hdfs", a space, and "dfs". This is the file system shell. This is followed by the command name as an argument to "hdfs dfs". These commands start with a dash. For example, the "ls" command for listing a directory is a common UNIX command and is preceded with a dash.
As on UNIX systems, ls can take a path as an argument. In this example, the path is the current directory, represented by a single dot.
As we saw for the "ls" command, the file system shell commands can take paths as arguments. These paths can be expressed in the form of uniform resource identifiers or URIs. The URI format consists of a scheme, an authority, and path. There are multiple schemes supported. The local file system has a scheme of "file". HDFS has a scheme called "hdfs". For example, let us say you wish to copy a file called "myfile.txt" from your local filesystem to an HDFS file system on the localhost. You can do this by issuing the command shown. The cp command takes a URI for the source and a URI for the destination. The scheme and the authority do not always need to be specified. Instead you may rely on their default values. These defaults can be overridden by specifying them in a file named core-site.xml in the conf directory of your Hadoop installation.
HDFS is not a fully POSIX compliant file system, but it supports many of the commands. The HDFS commands are mostly easily-recognized UNIX commands like cat and chmod. There are also a few commands that are specific to HDFS such as copyFromLocal. We'll examine a few of these.
copyFromLocal and put are two HDFS-specific commands that do the same thing - copy files from the local filesystem to a location on another filesystem. Their opposite is the copyToLocal command which can also be referred to as get. This command copies files out of the filesystem you specify and into the local filesystem.
getMerge is an enhanced form of get that can merge the files from multiple locations into a single local file. setRep lets you override the default level

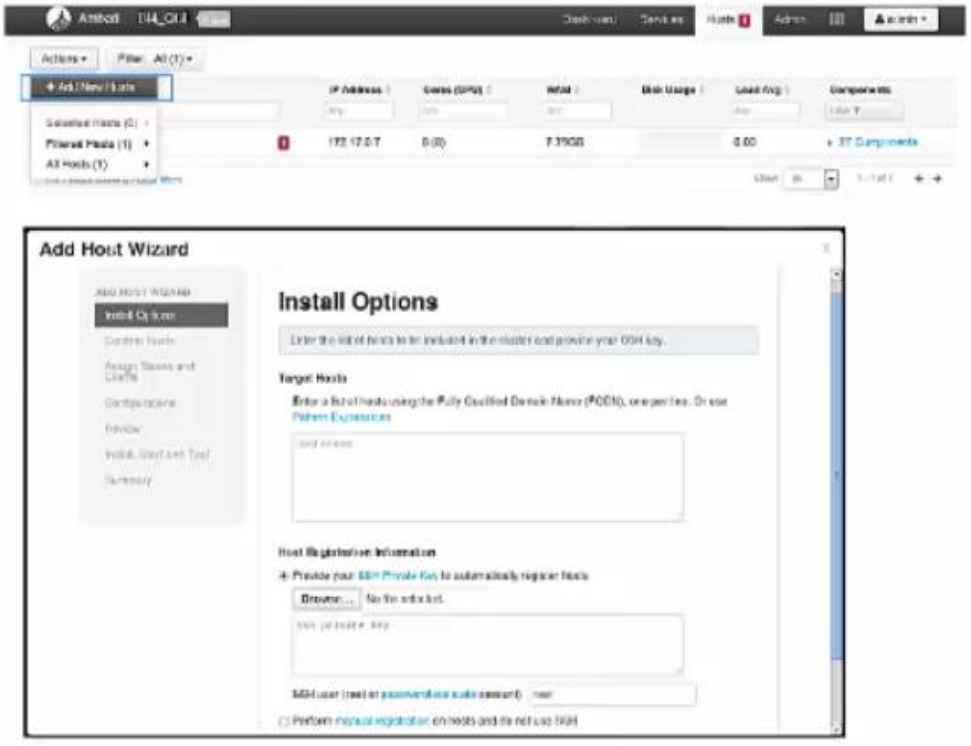
of replication. You can do this for one file or, with the -R option, to an entire tree.
This command returns immediately after requesting the new replication level. If you want the command to block until the job is done, pass the -w option.
IBM, with BigInsights, provides the Ambari Console as graphical way to work with HDFS.
The services tab provides a simple way to view the status of the Hadoop components.
Create a file view to browse and work with directories and files
This concludes the presentation. Thank you for watching.
Here is a list of trademarks that might have been referenced in this presenetation.

MODULE 3 – HADOOP ADMINISTRATION

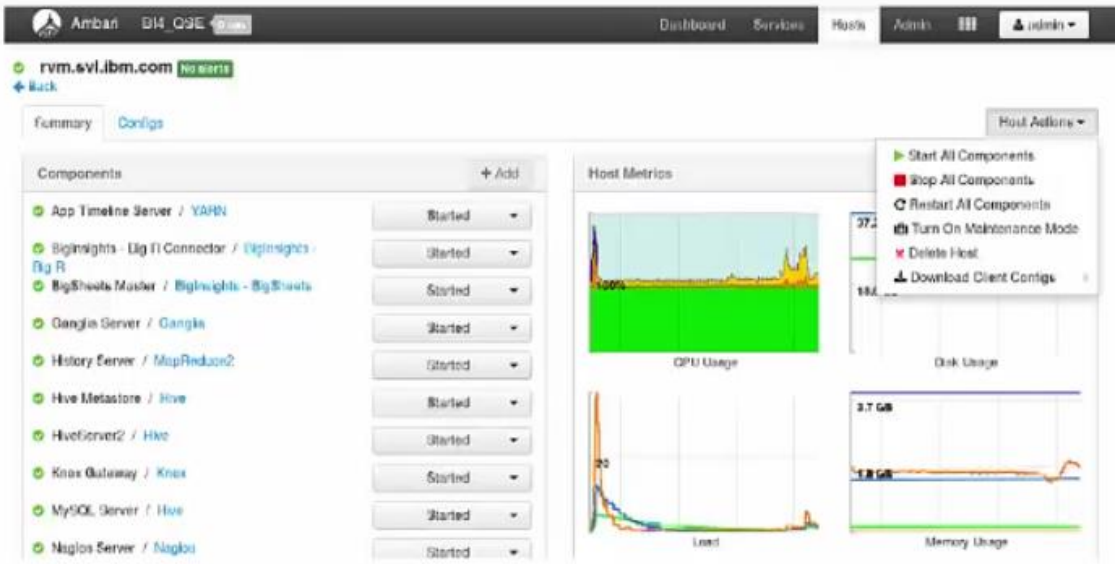
- Can be performed from Ambari Web Console
  - Need IP address or hostname of node to add
  - Node must be reachable

    Eg: `ssh 192.168.44.15`
  - BigInsights on node to add must NOT be installed
  - /etc/hosts on both master and child nodes should be updated prior to adding child nodes

Using Web Console to add nodes



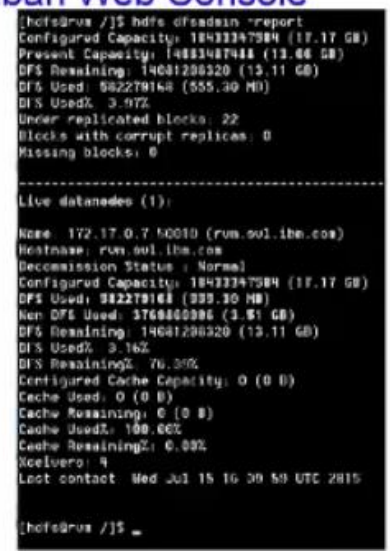
Using Web Console to remove nodes



Verifying cluster health – Disk space

- Performed DFS Disk Check by running DFS Report
  - Helps determine if there is low disk storage
  - Can be viewed from the Ambari Web Console
- Run DFS Report using:

```
hdfs dfsadmin -report
```



Starting / stopping components

- Not all components may need to be running
  - Stopping some can save resources

Configuration files

- hadoop-env.sh Environment variables that are used in the scripts to run Hadoop.
- core-site.xml Configuration settings for Hadoop Core, such as I/O settings that are common to HDFS and MapReduce
- hdfs-site.xml Configuration settings for HDFS daemons: the name node, secondary name node, and the data nodes.
- mapred-site.xml Configuration settings for MapReduce daemons and jobtracker, and tasktrackers.
- masters A list of machines (one per line) that each run secondary NameNode
- slaves A list of machines (one per line) that each run data node and tasktracker
- hadoop-metrics.properties Properties for controlling how metrics are published in Hadoop.
- log4j.properties Properties for system logfiles, the NameNode audit log, and the task log for the tasktracker child process

BigInsights Configuration Directory: /usr/iop/current/hadoop-client/conf

hadoop-env.sh settings

- Most variables by default and not set
- Only export JAVA_HOME is required and should be set to java JDK
- HADOOP_HOME – Contains code & config files /usr/iop/current/hadoop-client
- HADOOP_LOG_DIR – Keeps logs /var/log/Hadoop/$USER
- HADOOP_HEAPSIZE – heap size used by JVM of each daemon
  - Can be overwritten for each daemon:
    - NameNode - HADOOP_NAMENODE_OPTS
    - DataNode - HADOOP_DATANODE_OPTS
    - Secondary NameNode - HADOOP_SECONDARYNAMENODE_OPTS
    - JobTracker - HADOOP_JOBTRACKER_OPTS
    - TaskTracker - HADOOP_TASKTRACKER_OPTS
- Other environment variables: HADOOP_CLASSPATH, HADOOP_PID_DIR

core-site.xml settings

fs.defaultFS	The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation. The uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem. Default: file:///
hadoop.tmp.dir	A base for other temporary directories. Default: /tmp/hadoop-${user.name}
fs.trash.interval	Number of minutes between trash checkpoints. If zero, the trash feature is disabled (default). When greater than zero erased files will be inserted in .trash in user's home directory.
io.file.buffer.size	The size of buffer for use in sequence files. The size of this buffer should be a multiple of hardware page size (4096 on Intel x86), and it determines how much data is buffered during read and write operations.

hdfs-site.xml settings

dfs.datanode.data.dir	Determines where on the local filesystem a DFS data node should store its blocks.
dfs.namenode.name.dir	Determines where on the local filesystem the DFS namenode should store the name table.
dfs.blocksize	HDFS block size. Default is 64MB. **Recommendation:** Set block size to 128MB or as appropriate for your data.

mapred-site.xml configuration (continue)

mapreduce.job.reduces	The default number of reduce tasks per job. Typically set to 99% of the cluster's reduce capacity, so that if a node fails the reduces can still be executed in a single wave. Ignored when mapred.job.tracker is "local". Default: 1. Recommendation: set it to 90%
mapreduce.map.speculative	If true, then multiple instances of some map tasks may be executed in parallel. Default: true.
mapreduce.reduce.speculative	If true, then multiple instances of some reduce tasks may be executed in parallel. Default: true. Recommended: false.
mapreduce.tasktracker.map.tasks.maximum	The maximum number of map tasks that will be run simultaneously by a task tracker. Default: 2. Recommendations: set relevant to number of CPUs and amount of memory on each data node.
mapreduce.tasktracker.reduce.tasks.maximum	The maximum number of reduce tasks that will be run simultaneously by a task tracker. Default: 2. Recommendations: set relevant to number of CPUs and amount of memory on each data node.

mapred-site configuration (continue)

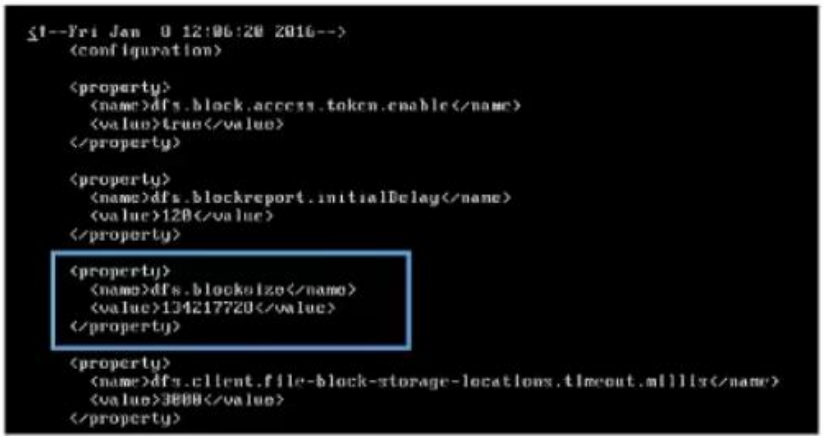
mapreduce.jobtracker.taskscheduler	The class responsible for scheduling the tasks. Default points to FIFO scheduler. Recommendation: Use Job Queue Task - org.apache.hadoop.mapred.JobQueueTaskScheduler
mapreduce.jobtracker.restart.recover	Recover failed job when JobTracker restarts. For production clusters recommended to be set to TRUE
mapreduce.cluster.local.dir	The local directory where MapReduce stores intermediate data files. May be a comma-separated list of directories on different devices in order to spread disk i/o. Directories that do not exist are ignored. Default: ${hadoop.tmp.dir}/mapred/local

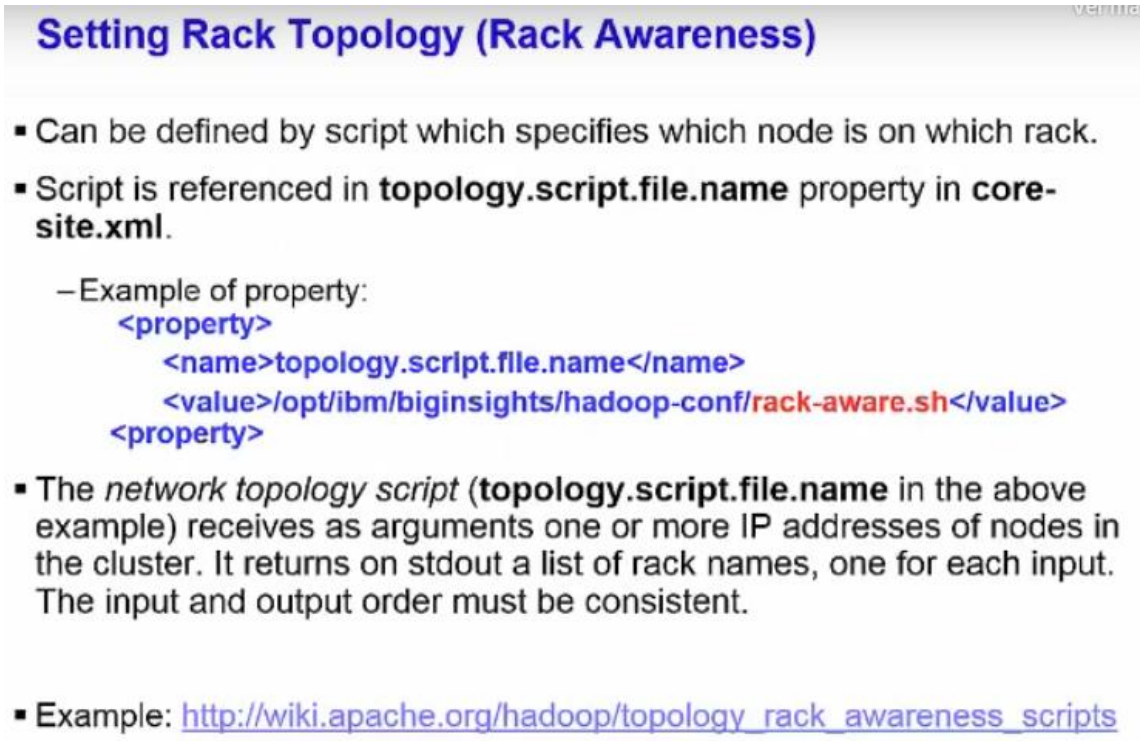
Configuring Hadoop - Example

- Stop appropriate services before making the change
- Change to the conf directory, look for hdfs-site.xml:

```
cd /usr/iop/current/hadoop-client/conf

vi hdfs-site.xml
```





Welcome to the unit on Hadoop Administration. The agenda covers adding nodes to a cluster, verifying the health of a cluster, and stopping / starting components. Then the unit covers configuring Hadoop and setting up rack topology.
Let's begin with adding and removing nodes from a cluster.
Adding nodes can be performed from the Ambari Console. To do so requires either the ip address or hostname of the node to be added. The node to be added must also be reachable. And as a matter of fact, it works both ways. The master and child nodes must all be able to communicate with each other. In this case, a child node refers to the node that is being added. It may not have BigInsights already installed on it. When the node is added to a cluster, the BigInsights code is transferred to the new node and installed.
From the Ambari Console you navigate to the Hosts tab and, on the left side, under Actions select Add New Hosts. You are then presented with a dialog that allows you to specify one or more nodes to be added. You may type in ip address, hostnames or any combination thereof. You can even specify an ipaddress range or a regular expression with your hostname.
Once the nodes have been added, you define which services are to be hosted on those nodes. You can select multiple services for one or more nodes.
Services can also be removed from a node.
As a matter of fact, if you are using the Ambari Console to remove a node, you must first remove all services from that node. Depending on which services are running on a node, you select which are to be removed. When there are no services running on a node, it can be removed using the Ambari Console.
Next let us discuss verifying the heath of your cluster.
You are able to view all of the nodes in the cluster, see the status of each node and which services are running on each node.
From the command line you can run the DFS Disk Check report . This lets you see how much space is still available on each DataNode.
Next let us look at start and stopping components and services.
In order to save some resources, you may only want to start a subset of the Hadoop components . Using Ambari, navigate to the Services tab and choose a service from the left that you would like to stop or start. When the services main page appears on the right side under Service Actions you can start or stop that service.
All services can be stopped or started from the main Dashboard
Now let us look at how to configure Hadoop.
Hadoop is configured using a number of XML files. And each file controls a number of

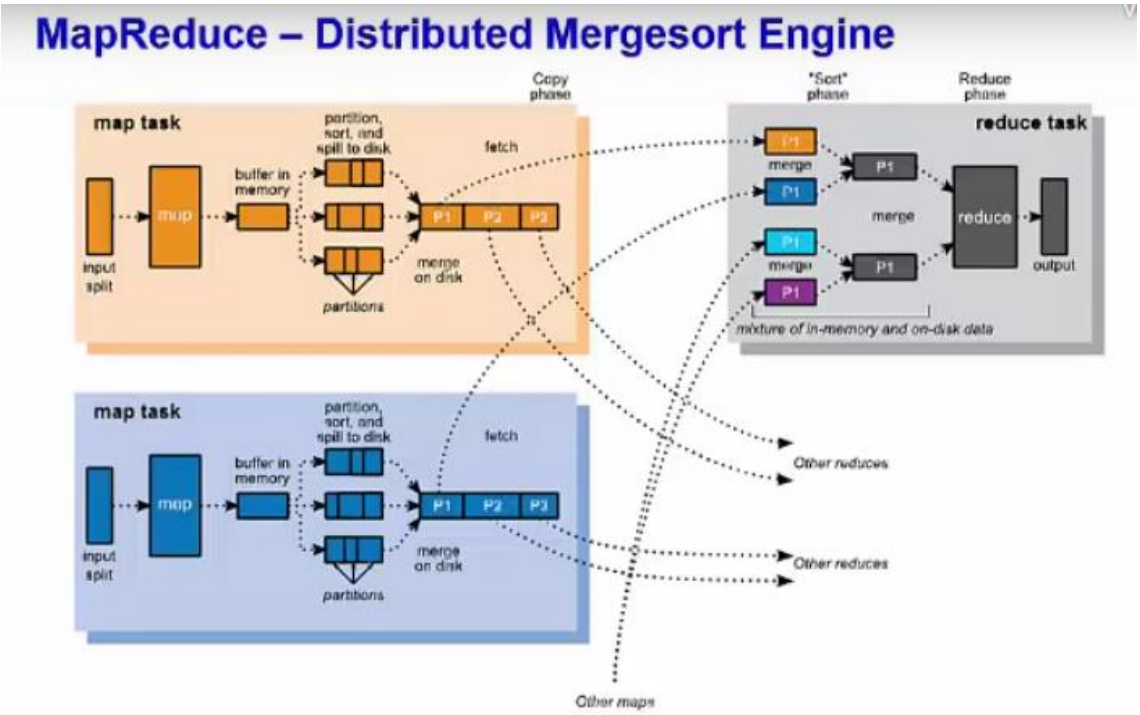
parameters. There are three main configuration files with which you will work.
core-site.xml is used to configure the parameters that are common to both HDFS and MapReduce. hdfs-site.xml contains parameters that
are for the HDFS daemons, like the NameNode and DataNodes.
mapred-site.xml controls the settings for MapReduce daemons, JobTracker and TaskTrackers.
We are not going to spend the time covering all of the configuration files. That would just take too much time. However, you do have the option of pausing this video if you would like to read the descriptions of the other configuration files.
The hadoop-env.sh is a script that sets a number of environment variables. Normally, with Hadoop, these variables are not set but with BigInsights, they are. There is one that must always be set and that is the JAVA_HOME environment variable.
Here are some of the settings found in core-site.xml. We are not going to spend time on these nor those on this page as well. If you want to pause the video to read their description, feel free to do so.
Next we have some setting in hdfs-site.xml. If you want to set a different value for the default block size, then you would modify dfs.block.size. Likewise, if you want to change the default replication factor, then you would modify dfs.replication. Once again, I am not going to cover all the parameters.
To change MapReduce settings, you modify mapred-site.xml. You can control which nodes can connect to
the JobTracker. Mapreduce.job.reduces lets you set the number
of reduce tasks per job. mapreduce.map.speculative. execution allows the JobTracker, when having determined that there might be a problem with one map task, to start another map task running in parallel. Both map tasks process the same data and, upon successful completion of one of the tasks, the other is terminated. mapreduce.tasktracker.map.tasks.maximum and mapreduce.tasktracker.reduce.tasks.maximum
lets you define the number of slots on a TaskTracker that can run map and reduce task.
mapreduce.jobtracker.taskScheduler points to the scheduler that is to be used for MapReduce jobs.
So how do you set these parameters? First of all, you must stop the appropriate service or services before making the change. You are making changes to value element for the appropriate property element. The configuration files are in the hadoop-client/conf directory.
The changes must be made to the configuration files on all nodes in the cluster.
Let me spend a few minutes and focus on BigInsights. With BigInsights the hadoop-conf directory is under $BIGINSIGHTS_HOME. But, and this is very important, you do not make changes to the configuration files in that directory. BigInsights has a staging directory which is $BIGINSIGHTS_HOME/hdm/hadoop-conf-staging that has copies of the configuration files.
You make changes to the files in this staging directory and then execute a script that distributes the changes to all nodes in the cluster.
Finally, let us talk about setting up rack topology.
To make Hadoop aware of the clusters topology, you code a script that receives as arguments, one or more ip addresses of nodes in the cluster. The script returns on stdout, a list of rack names, one for each input value. Then you update core-site.xml and modify the topology.script.file.name
property to point to your script. The good news is that there are examples available for you to review. This ends this unit. Thank you for attending.
And here are some trademarks that may have been referenced in this presentation.rones

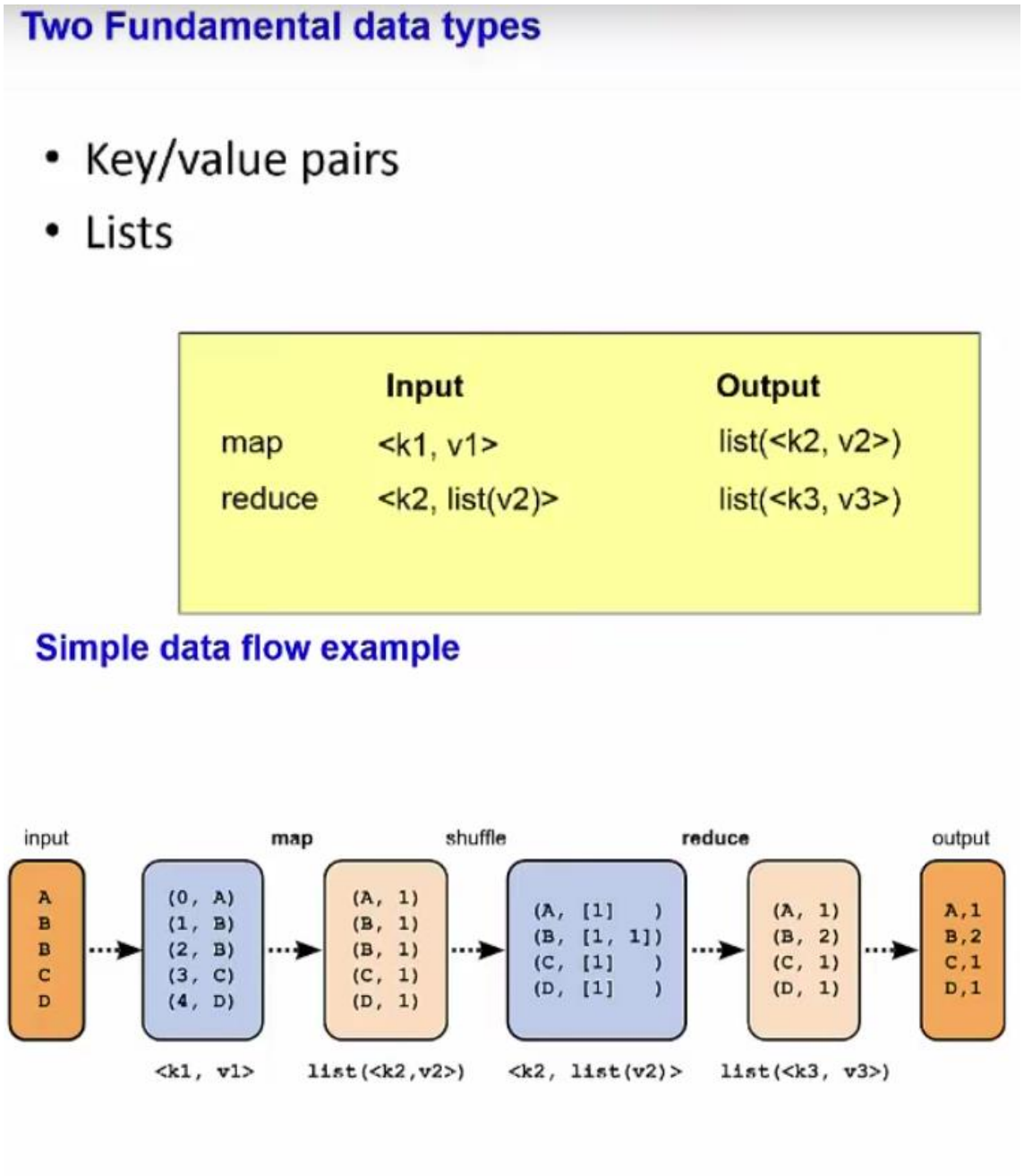
MODULE 4 – HADOOP COMPONENTS

MAP REDUCE

MapReduce

- Processes huge datasets for certain kinds of distributable problems using a large number of nodes
- Map
  - Master node partitions the input into smaller sub-problems
  - Distributes the sub-problems to the worker nodes
- Reduce
  - Master node then takes the answers to all the sub-problems
  - Combines them in some way to get the output
- Allows for distributed processing of the map and reduce operations





Welcome to Hadoop Fundamentals, Hadoop Components. In this unit I will discuss the MapReduce Philosophy, describe the usage of Pig and Hive, talk about how to get data into Hadoop through the use of Flume and Sqoop and finally describe how to schedule and control job execution using Oozie.
First I need to set the boundaries for this unit. The components presented in this unit are done so at a very high level. The Hadoop environment is littered with a number of open source components with funny sounding names. And to some people, it is difficult to understand their usage. This unit is merely an attempt to provide you with descriptions of some of these components. If you are interested in getting more detail about each of the components covered in this unit, then I would direct you to the other Big Data University courses that are specific to these components. Let us take a look at MapReduce. MapReduce is designed to process very large datasets for certain types of distributable problems. It attempts to spread the work across a large number of nodes and allows those nodes to process the data in parallel. You cannot have dependencies within the data, meaning that you cannot have a requirement that one record in a dataset must be processed before another. Results from the initial parallel processing are sent to additional nodes where the data

is combined to allow for further reductions of the data.
Now let's take a look at how the map and reduce operations working sequence on your data to produce the final output. In this case, we will have a job with a single map step and a single reduce step. The first step is the map step. It takes a subset of the full dataset called an input split and applies to each row in the input split an operation that you have written, such as parsing each character string.
The output data is buffered in memory and spills to disk.
It is sorted and partitioned by key using the default partitioner.
A merge sort sorts each partition. There may be multiple map operations running in parallel with each other, each one processing a different input split.
The partitions are shuffled among the reducers. For example, partition 1 goes to reducer 1.
The second map task also sends its partition 1 to reducer 1.
Partition 2 goes to another reducer.
Additional map tasks would act in the same way.
Each reducer does its own merge steps and executes the code of your reduce task.
For example, it could do a sum on the number of occurrences of a particular character string.
This produces sorted output at each reducer. The data that flows into and out of the mappers and reducers takes a specific form. Data enters Hadoop in unstructured form but before it gets to the first mapper, Hadoop has changed it into key-value pairs with Hadoop supplying its own key. The mapper produces a list of key value pairs.
Both the key and the value may change from the k1 and v1 that it came in as to a k2 and v2.
There can now be duplicate keys coming out of the mappers. The shuffle step will take care of grouping them together. The output of the shuffle is the input to the reducer step. Now, we still have a list of the v2's that came out of the mapper step, but they are grouped by their keys and there is no longer more than one record with the same key. Finally, coming out of the reducer is, potentially, an entirely new key and value, k3 and v3. For example, if your reducer summed the values associated with each k2, your k3 would be equal to k2 and your v3 would be the sum of the list of v2s.
Let us look at an example of a simple data flow. Say we want to transform the input on the left to the output on the right. On the left, we just have letters. On the right, we have counts of the number of occurrences of each letter in the input.
Hadoop does the first step for us. It turns the input data into key-value pairs and supplies its own key: a increasing sequence number. The function we write for the mapper needs to take these key-value pairs and produce something that the reduce step can use to count occurrences. The simplest solution is make each letter a key and make every value a 1. The shuffle groups records having the same key together, so we see B now has two values, both 1, associated with it.
The reduce is simple: it just sums the values it is given to produce a sum for each key.
This lesson is continued in the next video.

PIG AND HIVE

- **Pig and Hive**
  - Similarities
    - All translate high-level languages to MapReduce jobs
    - All offer significant reductions in program size over Java
    - All provide points of extension to cover gaps in functionality
    - All provide interoperability with other languages
    - None support random reads/writes or low-latency queries

Languages - Hive

- Developed at Facebook
- Declarative language (SQL dialect)
- Schema non-optional but data can have many schemas
- Relationally complete
- Turing complete when extended with Java UDFs

Languages - Pig

- Developed at Yahoo!
- Data flow language
- Can operate on complex, nested data structures
- Schema optional
- Relationally complete
- Turing complete when extended with Java UDFs

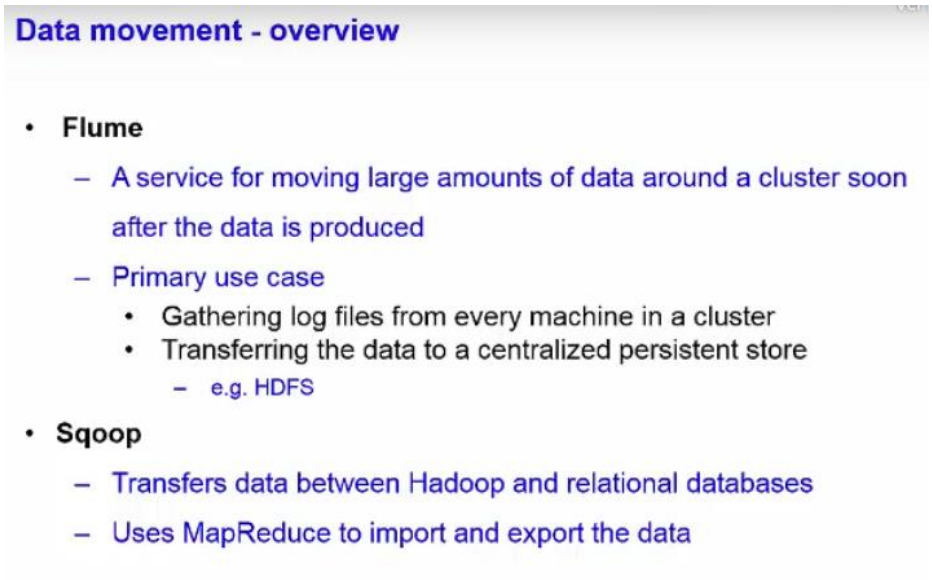
Pig

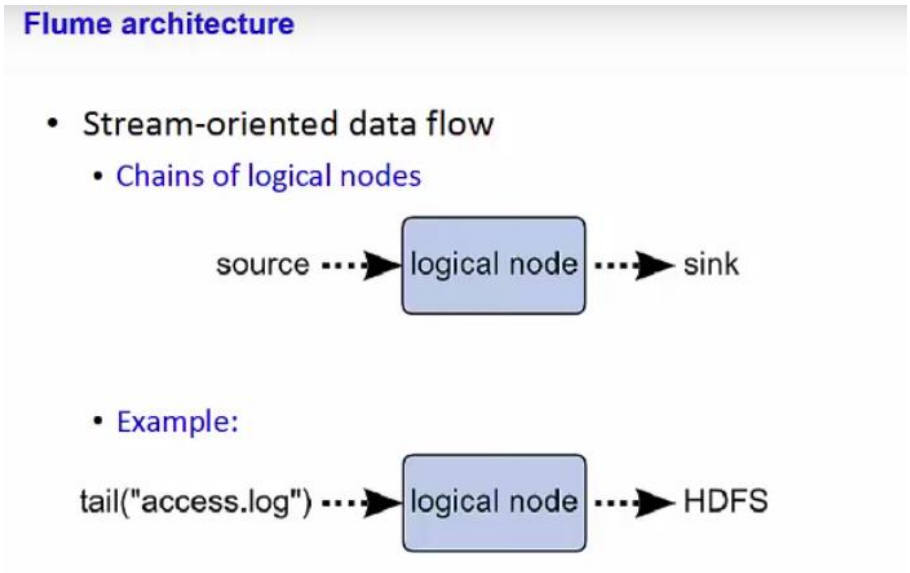
- **Running Pig**
  - Script
    - pig scriptfile.pig
  - Grunt
    - pig (to launch command line tool)
  - Embedded
    - Call in to Pig from Java
  - Execution environments
    - Local
    - Distributed

Pig and Hive have much in common. They all translate high-level languages into MapReduce jobs so that the programmer can work at a higher level than he or she would when writing MapReduce jobs in Java or other lower-level languages supported by Hadoop using Hadoop streaming. The high level languages offered by Pig and Hive let you write programs that are much smaller than the equivalent Java code. When you find that you need to work at a lower level to accomplish something these high-level languages do not support themselves, you have the option to extend these languages, often by writing user-defined functions in Java. Interoperability can work both ways since programs written in these high-level languages can be imbedded inside other languages as well. Finally, since all these technologies run on top of Hadoop, when they do so, they have the same limitations with respect to random reads and writes and low-latency queries as Hadoop does.
Now, let us examine what is unique about each technology, starting with Pig. Pig was developed at Yahoo Research around 2006 and moved into the Apache Software Foundation in 2007. Pig's language, called PigLatin, is a data flow language - this is the kind of language in which you program by connecting things together. Pig can operate on complex data structures, even those that can have levels of nesting. Unlike SQL, Pig does not require that the data have a schema, so it is well suited to processing unstructured data. However, Pig can still leverage the value of a schema if you choose to supply one. Like SQL, PigLatin is relationally complete, which means it is at least as powerful as relational algebra. Turing completeness requires looping constructs, an infinite memory model, and conditional constructs. PigLatin is not Turing complete on its own, but is Turing complete when extended with User-Defined Functions. Hive is a technology developed at Facebook that turns Hadoop into a data warehouse complete with a dialect of SQL for querying. Being an SQL dialect, HiveQL is a declarative language. Unlike in PigLatin, you do not specify the data flow, but instead describe the result you want and Hive figures out how to build a data flow to achieve it. Also unlike Pig, a schema is required, but you are not limited to one schema. Like PigLatin and SQL, HiveQL on its own is a relationally complete language but not a Turing complete language. It can be extended through UDFs just like Pig to be Turing complete. Let us examine Pig in detail. Pig consists of a language and an execution environment. The language is called PigLatin. There are two choices of execution environment: a local environment and distributed environment. A local environment is good for testing when you do not have a full distributed Hadoop environment deployed. You tell Pig to run in the local environment when you start Pig's command line interpreter by passing it the -x local option. You tell Pig to run in a distributed environment by passing -x mapreduce instead. Alternatively, you can start the Pig command line interpreter without any arguments and it will start it in the distributed environment. There are three different ways to run Pig. You can run your PigLatin code as a script, just by passing the name of your script file to the pig command. You can run it interactively through the grunt command line launched using pig with no script argument. Finally, you can call into Pig from within Java using Pig's embedded form. As mentioned in the overview, Hive is a technology for turning Hadoop into a data warehouse, complete with an SQL dialect for querying

it. There are three ways to run Hive. You can run it interactively by launching the hive shell using the hive command with no arguments. You can run a Hive script by passing the -f option to the hive command along with the path to your script file. Finally, you can execute a Hive program as one command by passing the -e option to the hive command followed by your Hive program in quotes.
This lesson is continued in the next video.

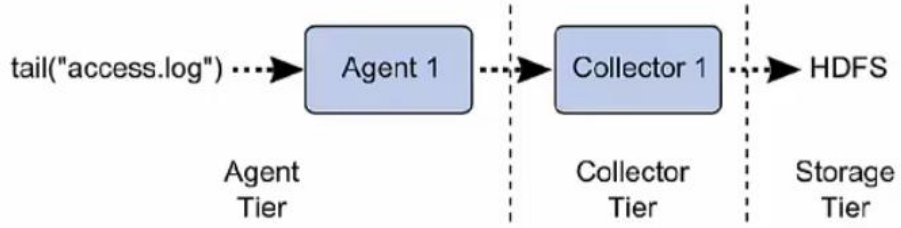
FLUME, SQOOP AND OOZIE



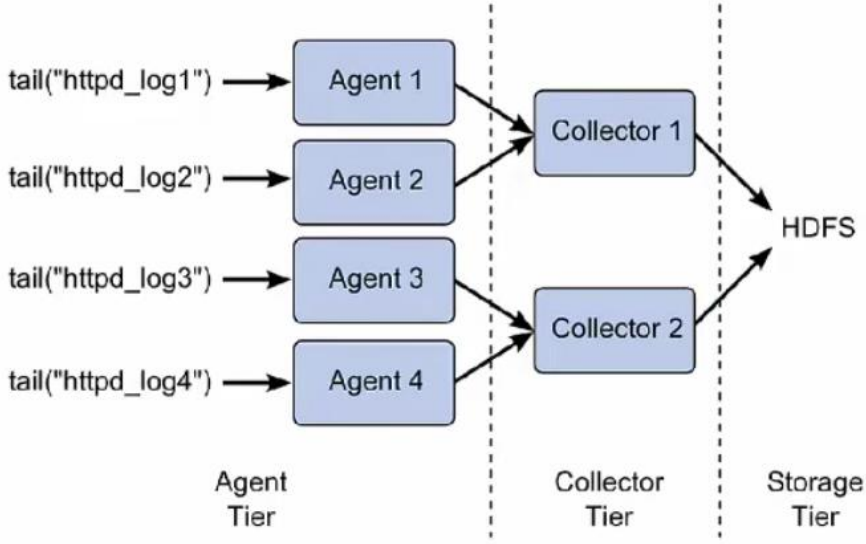


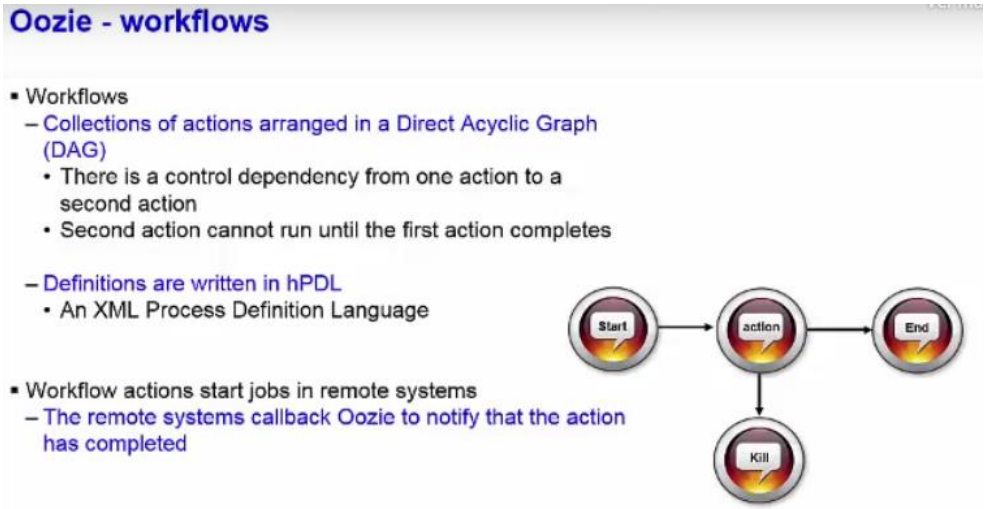
Flume architecture

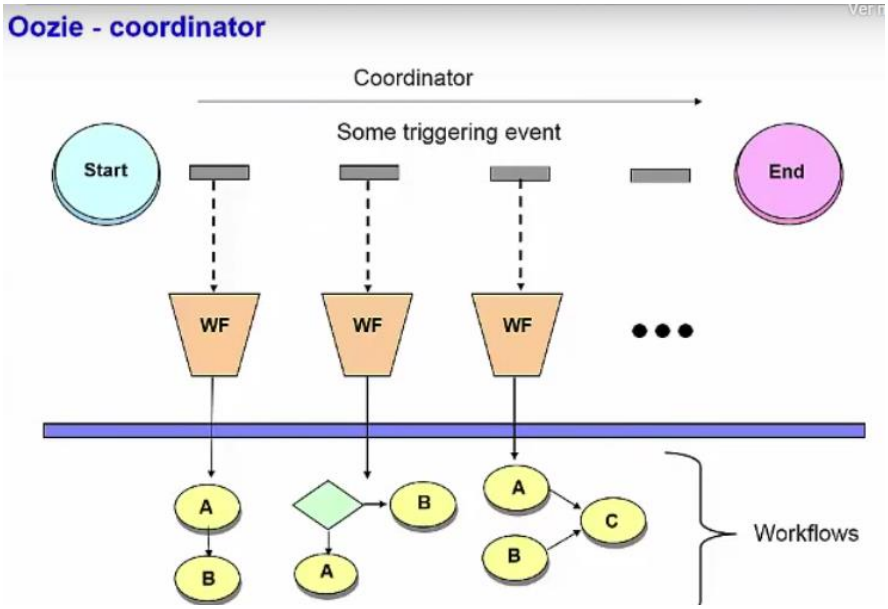
- Tiers
  - Agent Tier
  - Collector Tier
  - Storage Tier



Flume architecture







Now let us look at moving data into Hadoop. We will begin by looking at Flume's architecture, then examining the three modes it can run in followed by a look at the event data model.

Flume is an open source software program developed by Cloudera that acts as a service for aggregating and moving large amounts of data around a Hadoop cluster as the data is produced or shortly thereafter. Its primary use case is the gathering of log files from all the machines in a cluster to persist them in a centralized store such as HDFS.

This topic is not intended to cover all aspects of Sqoop but to give you an idea of the capabilities of Sqoop. Sqoop is an open source product designed to transfer data between relational database systems and Hadoop. It uses JDBC to access the relational systems. Sqoop accesses the database in order to understand the schema of the data. It then generates a MapReduce application to import or export the data.

When you use Sqoop to import data into Hadoop, Sqoop generates a Java class that encapsulates one row of the imported table. You have access to the source code for the generate class. This can allow you to quickly develop any other MapReduce applications that use the records that Sqoop stored into HDFS. In Flume, you create data flows by building up chains of logical nodes and connecting them to sources and sinks. For example, say you wish to move data from an Apache access log into HDFS. You create a source by tailing access.log and use a logical node to route this to an HDFS sink.

Most production Flume deployments have a three tier design. The agent tier consists of Flume agents co-located with the source of the data that is to be moved. The collector tier consists of perhaps multiple collectors each of which collects data coming in from multiple agents and forwards it on to the storage tier which may consist of a file system such as HDFS. Here is an example of such a design. Say we have four http server nodes producing log files labelled httpd_logx where x is a number between 1 and 4. Each of these http server nodes has a Flume agent process running on it. There are two collector nodes. Collector1 is configured to take data from Agent1 and Agent2 and route it to HDFS. Collector2 is configured to take data from Agent3 and Agent4 and route it to HDFS. Having multiple collector processes allows one to increase the parallelism in which data flows into HDFS from the web servers.
Oozie is an open source job control component used to manage Hadoop jobs.
Oozie workflows are collections of actions arranged in a Direct Acyclic Graph. There is a control dependency between actions in that a second action cannot run until the proceeding action completes. For example, you have the capability of having one job execute only when a previous job completed successfully. You can specify that several jobs are permitted to execute in parallel but a final job must wait to begin executing until all parallel jobs complete. Workflows are written in hPDL an XML process definition language, and are stored in a file called workflow.xml.
Each workflow action starts jobs in some remote system and that remote system performs a callback to Oozie to notify that the action completed. The coordinator component can invoke workflows based upon a time interval, that is for example, once every 15 minutes, or based upon the availability of specific data. It might also be necessary to connect workflow jobs that run regularly but at different time intervals. For example, you may want the output of the last four jobs that run every 15 minutes to be the input to a job that runs every hour.
A single workflow can invoke a single task or multiple tasks, either in sequence or based upon some control logic. The is the end of this unit and the course.
As I said earlier, the components discussed in this unit have Big Data University courses that cover them in more detail. Thank you for attending.
Here is a list of trademarks that may have been referenced in this unit.