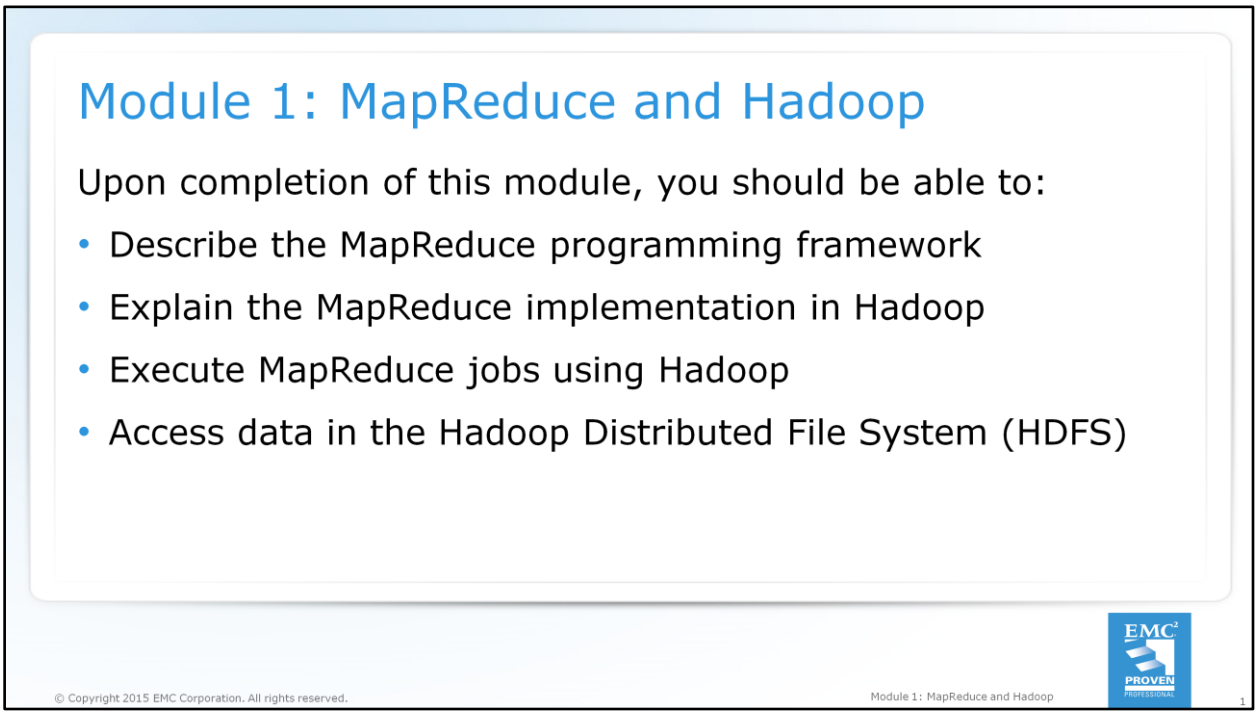
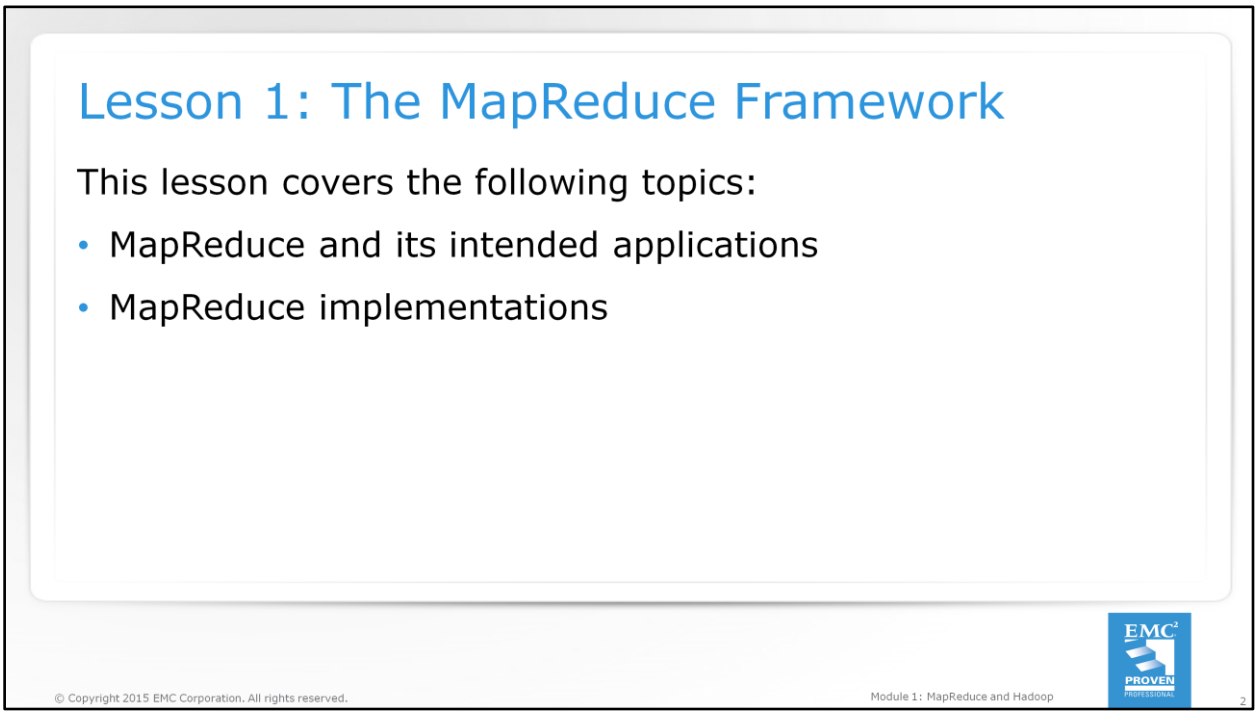



This module covers the MapReduce framework and its implementation in Apache Hadoop.




This lesson explains the basis for the MapReduce framework.

# Understanding MapReduce

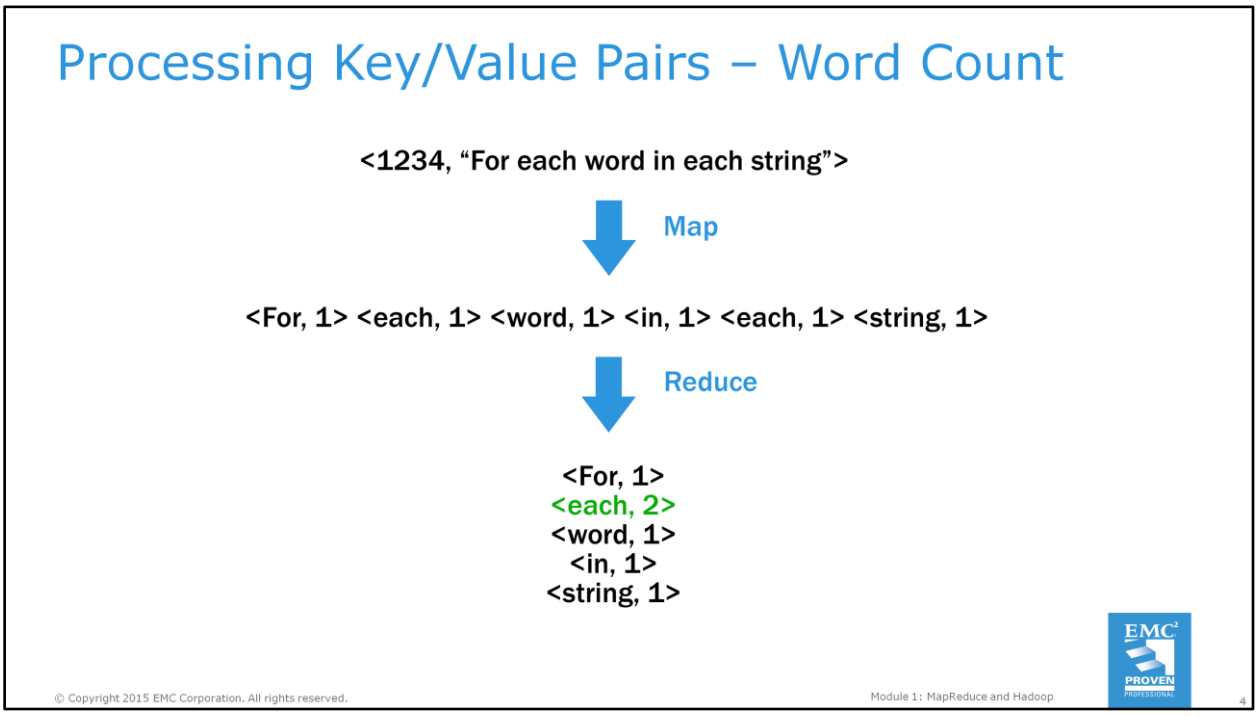
- Map Step
  - Applies an operation to a piece of data
  - Provides some intermediate output
- Reduce Step
  - Consolidates the intermediate outputs from the map steps
  - Provides the final output
- Each step uses key/value pairs for input and output
  - Denoted <key, value>




The MapReduce paradigm provides the means to break a large task into smaller tasks, to run the tasks in parallel, and to consolidate the outputs of the individual tasks into the final output. As its name implies, MapReduce consists of two basic parts, a map step and a reduce step.

Each step uses key/value pairs, denoted as <key, value>, as input and output. It is useful to think of the key/value pairs as a simple ordered pair; however, the pairs can take fairly complex forms. For example, the key could be a file name and the value could be the entire contents of the file.

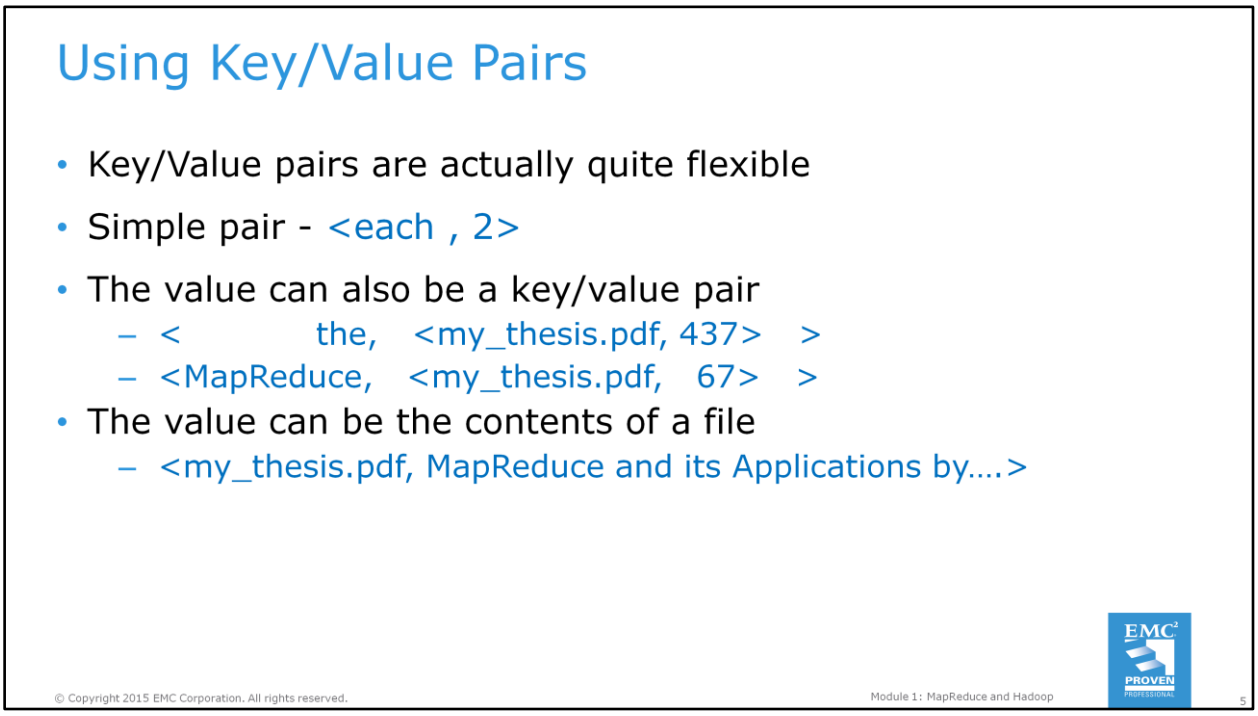
The simplest illustration of MapReduce is a word count example where the task is to simply count the number of times each word appears in a collection of documents. In practice, the objective of such an exercise is to establish a list of words and their frequency for purposes of search or establishing the relative importance of certain words.




In this example, the code for the Map step will parse the text string into words and emit a set of key/value pairs of the form <word, 1>. For each unique word, the code for the Reduce step sums up the "1" values and outputs the <word, count> key/value pairs.

This simple example illustrates the MapReduce processing for just one input record, in this case a text string. In practice, the Map step runs over millions of strings, then the Reduce step summarizes the billions of key/value pairs generated by the Map steps.

Note: In this example, the original key, "1234", is ignored in the processing.




The key value can be constructed in such a way that it also represents an n-tuple (e.g. concatenate text values, separated by a delimiter).

Although a key can be a meaningless entity, a key is typically the item of interest on which some additional work will be executed. Examples are:

- a word (as seen in the word count example)
- IP address
- User ID

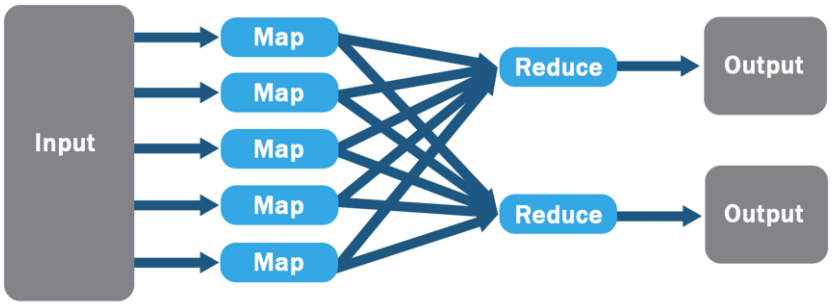
The value in the key/value pairs can be:

- a numeric
- a file name
- the actual file content (text, jpeg, binary, etc.)

After the MapReduce job completes, the final key will often be the item of interest that the user or an application will want to examine.

# Why is MapReduce Useful?

- Breaks one large task into many smaller tasks
- The smaller tasks can be processed in parallel







Rear admiral Grace Hopper (1906-1992), who was a pioneer in the field of computers, said that in the pioneer days oxen were used for heavy pulling, but when one ox couldn't budge a log, the pioneers didn't try to raise a larger ox, they added additional oxen. Her point was that as computational problems grow we should not be trying to build a bigger computer, but we should attempt to build systems of computers.

Thus, in the MapReduce context, a large task will be spread-out, or distributed, among many computers.

# Applying MapReduce Intended Applications

- Intended applications
  - Anything that involves processing all of the data
  - Breaks the effort into smaller pieces (parallel processing)
  - Problems that scale linearly by adding more processing power
- Applications to avoid
  - Real time processing requirements
  - Large number of small files
  - Dependencies between the dataset records



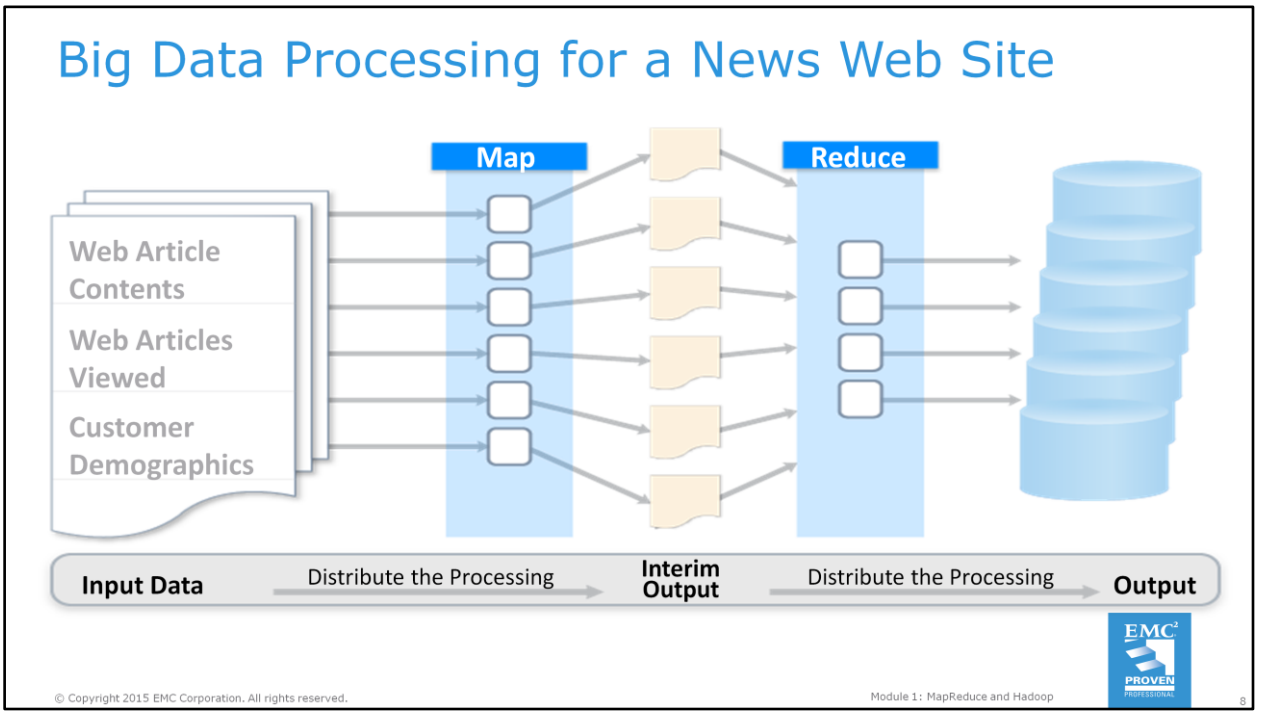


MapReduce lends itself to distributed processing. So, if there are petabytes of data to be processed and the processing can be easily broken into smaller, identical tasks, the MapReduce framework can be used to efficiently process the data.

If the task is to only process a small set of data, say ten log files, for a very specific purpose, using MapReduce may not be the best choice; however, if the need exists to routinely access the contents of the log files, MapReduce could be used to process a set of daily log files to make the contents available for further analysis.

Another example where MapReduce may not be a suitable tool is when the processing criteria depends on additional data that is not part of the inputted key/value pair.

Less than ideal applications include a use case that only needs a very small subset of the data; however, MapReduce can help to make that part easier. Also, MapReduce is not useful for those case where it is difficult to break into independent pieces. In other words, when the processing of one record depends on another record's contents.



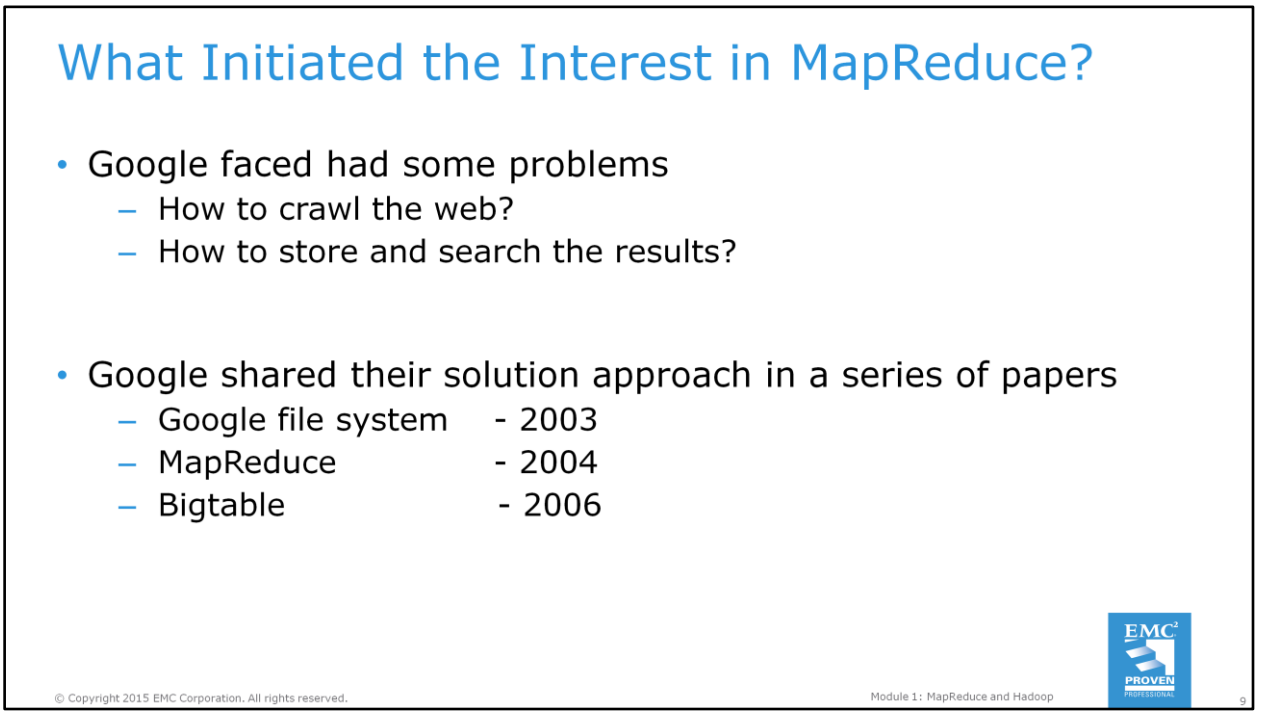

Next, we will look at a particular use case where MapReduce may be applied.

In general, MapReduce:

- Splits input data into distributable chunks
- Defines the steps to process those chunks
- Runs that process in parallel on the chunks

The Map step processes each chunk. The Reduce step combines the results of the Map steps.

In this News Web Site example, Web article contents are broken down and stored as the input for a search engine. The viewing history and frequency of each article is stored to provide input into a recommender system. The recommender system can be tailored to a particular user, based on customer demographics and preferences.




Although the concept of MapReduce has existed for decades, Google has led the resurgence in its interest and adoption starting with Google's 2004 paper, "MapReduce: Simplified Data Processing on Large Clusters," which described Google's approach for crawling the Web and building its search engine. As the paper describes, MapReduce has been used for decades in functional programming languages such as Lisp. Lisp is a programming language that obtained its name from being able to easily process lists.

The paper, "Bigtable: A Distributed Storage System for Structured Data," describes Google's development of a data store to access the results of various MapReduce jobs. Bigtable was the basis for the data store, HBase, which is covered in Module 2.

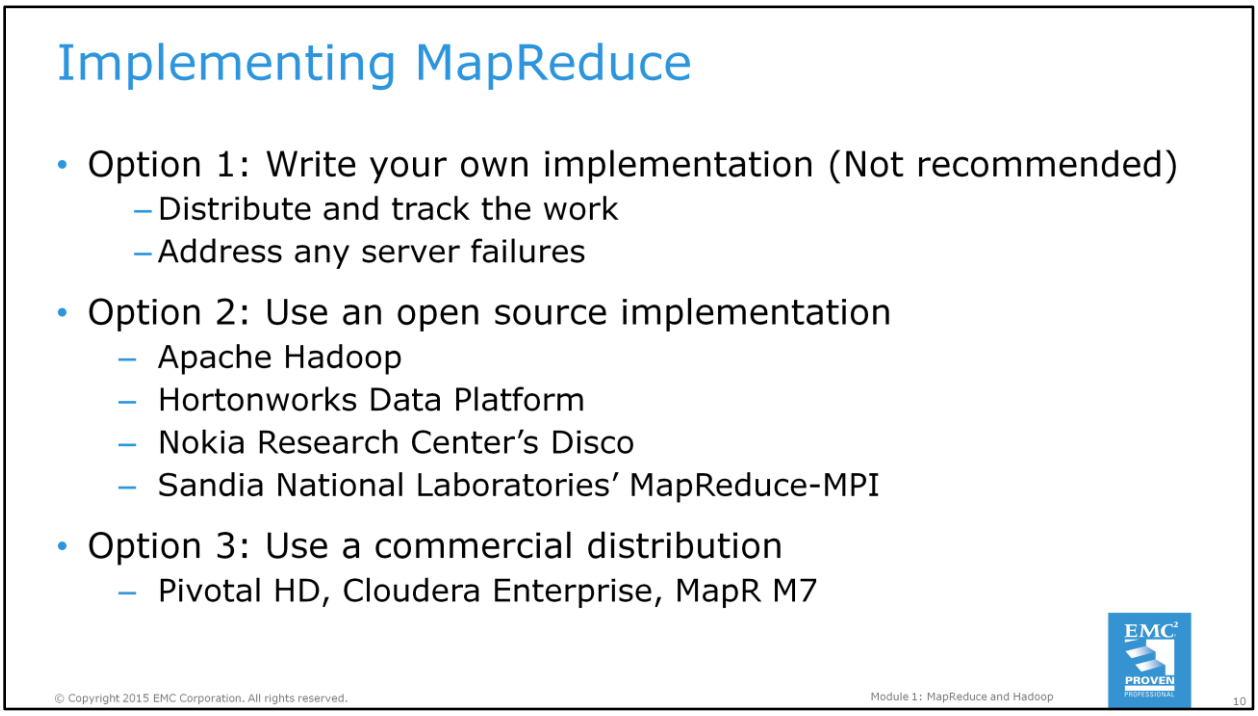
References:
Dean, Ghemawat , et. Al (2006). *Bigtable: A Distributed Storage System for Structured Data*
http://research.google.com/archive/mapreduce.html
http://research.google.com/archive/bigtable.html

Additional Resources:

Google has pledged "…the free use of certain of its patents in connection with Free or Open Source Software" on 10 of its MapReduce patents.

http://www.google.com/patents/opnpledge/pledge/
http://www.google.com/patents/opnpledge/patents/

Another key publication on the Google File System: http://research.google.com/archive/gfs.html



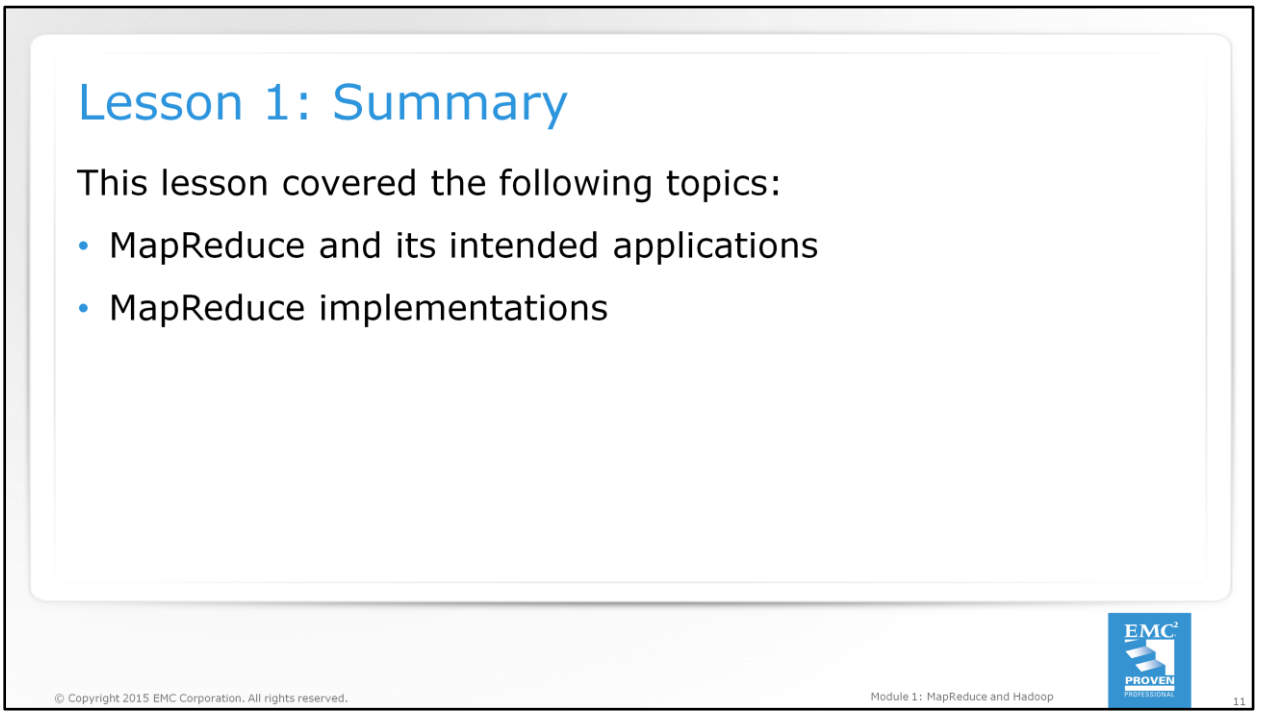

While MapReduce is a fairly simple paradigm to understand, it is not as easy to implement, especially in a distributed system. The implementation needs to break a MapReduce job into smaller pieces for execution across the cluster, monitor the running tasks, restart any failed tasks, and then coordinate the reduce steps to consolidate the map tasks. So, a better option is to use an open source implementation.

Links to the Open Source Implementations:
http://hadoop.apache.org
http://hortonworks.com/products/hortonworksdataplatform
http://discoproject.org
http://mapreduce.sandia.gov

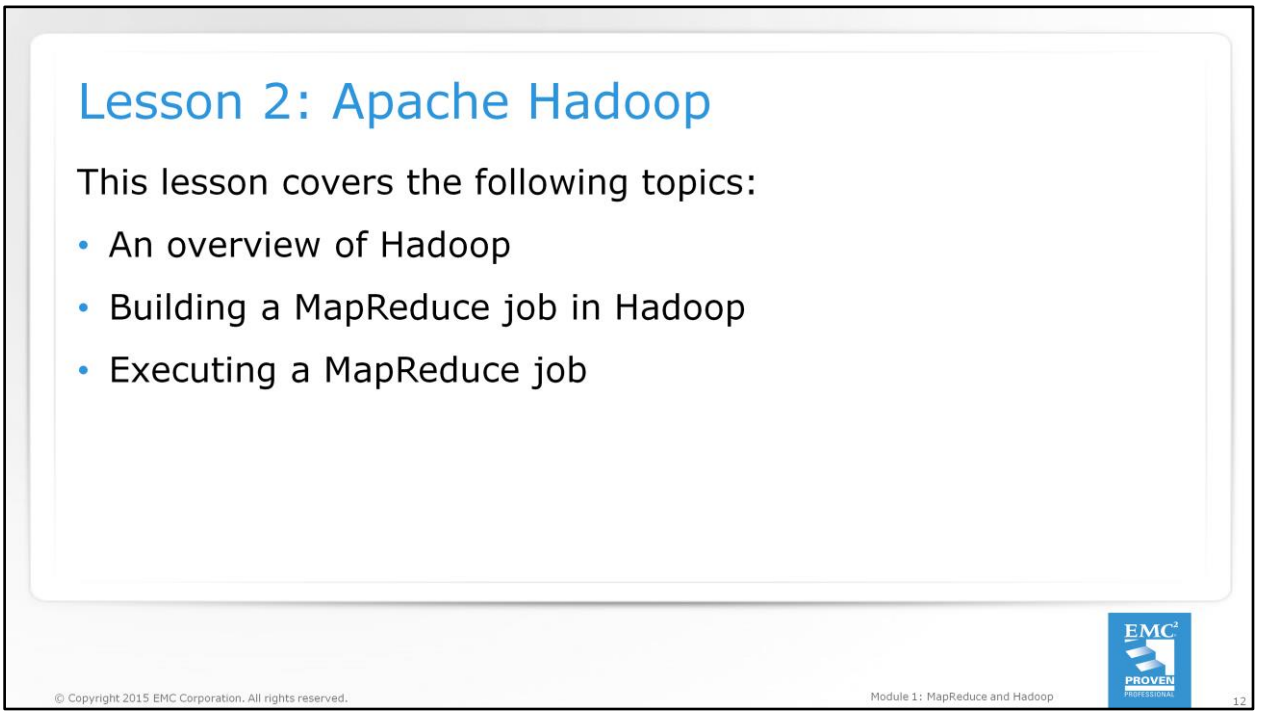
The listed commercial distributions are based on Apache Hadoop and often include other related Apache projects, such as Pig and Hive, in the Hadoop ecosystem. Commercial distributions often provide additional management software as well as service and support.

Links to the Commercial Distributions
http://www.pivotal.io/big-data/pivotal-hd
http://www.cloudera.com/content/cloudera/en/products-and-services/cloudera-enterprise.html
http://www.mapr.com/products/m7

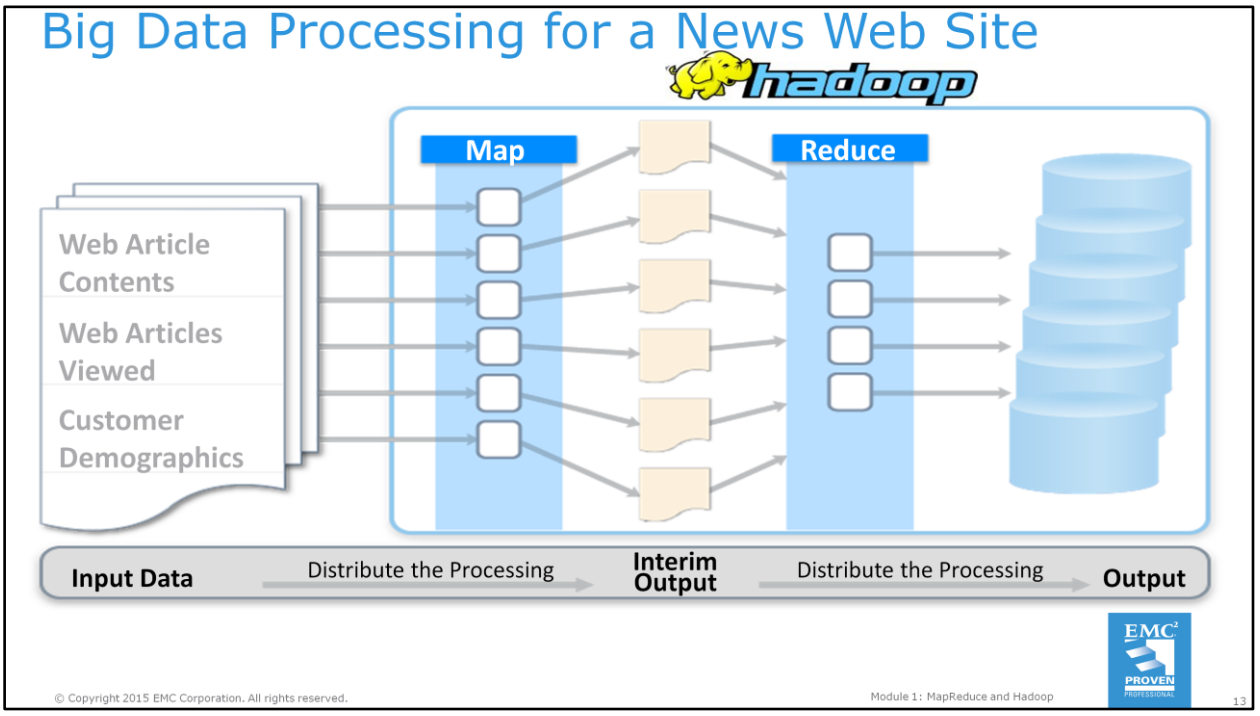
Note: MPI – Message Passing Interface




With the use of the word count example, this lesson provided an overview of MapReduce and its use of key/value pairs. MapReduce is intended for those applications where a large data set can be broken into smaller pieces such that the pieces can be processed in parallel. Applying MapReduce in a distributed cluster of computers enables the solution to scale as the data grows by adding more processing to the cluster. Apache Hadoop is a popular implementation of MapReduce, which is addressed in the next lesson.




This lesson focuses on how MapReduce is implemented in Hadoop and how a MapReduce job is run in Hadoop.




In this previous example, Apache Hadoop addressed the map and reduce steps, as well as the storage of the results.

Apache Hadoop also allows computational-intensive applications to be run across hundreds and thousands of computers. As the data volume increases, adding more compute and storage nodes enables the processing to scale linearly.

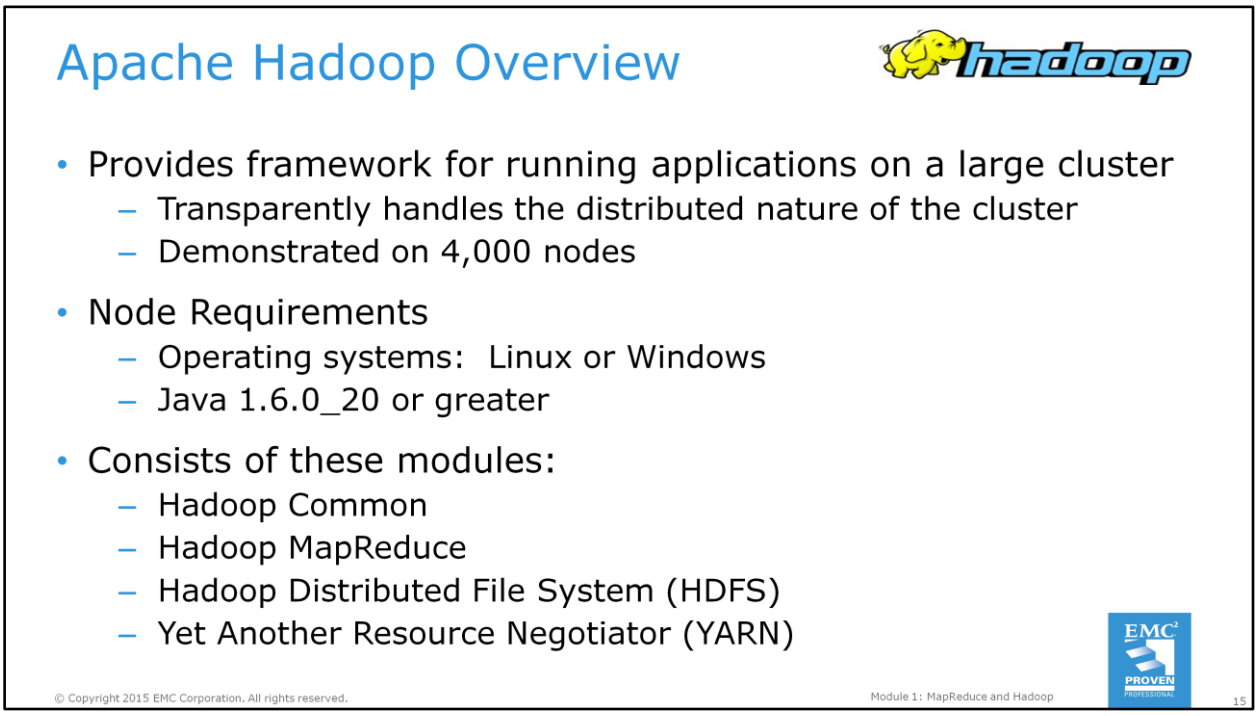

Some Hadoop use examples include:

- Collecting facial recognition attributes from a repository of pictures
- Compiling Twitter feeds
- Building the knowledge base of IBM's Watson from various inputs
- Processing click stream data to determine optimal web advertisement placement
- Enabling applications like LinkedIn's People You May Know

Many more uses cases can be found at: http://wiki.apache.org/hadoop/PoweredBy

Sources:

- Facial recognition: www.l1id.com/pages/116-face
- Twitter: https://twitter.com/logo
- Jennings/Watson: www.slate.com/articles/arts/culturebox/2011/02/my_puny_human_brain.html
- Web ad placements: www.elementsfactory.com/website-enhancements.php
- LinkedIn: www.linkedin.com/




Hadoop downloads and documentation are available at: http://hadoop.apache.org/

Node hardware considerations can be found at:
http://wiki.apache.org/hadoop/MachineScaling

The latest Java Development Kit (JDK) testing results can be found at:
http://wiki.apache.org/hadoop/HadoopJavaVersions

The Hadoop project includes these modules written in Java:

- Hadoop Common: The common utilities that support the other Hadoop modules.
- Hadoop MapReduce: A system for parallel processing of large data sets.
- Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput access to application data.
- Hadoop YARN: A framework for job scheduling and cluster resource management.

After a brief overview of HDFS, this lesson will focus on running a MapReduce job in Hadoop. The remaining lessons in this module cover HDFS and YARN in more detail.

# Hadoop Distributed File System (HDFS)

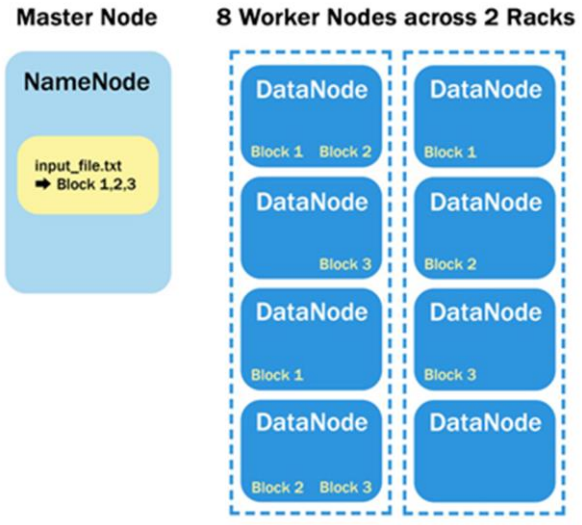
- Each file is broken up into 128 MB blocks (default)
- Each block has three copies (default)
  - Data availability
  - Flexibility in analysis
- A file system, not a database
  - Load the raw data, as is, regardless of the structure
  - Use MapReduce jobs to prepare the data for future use
- HDFS stores any file
  - not only files of key/value pairs




The map tasks process each block. For example, if a file is stored as 10 blocks in the cluster, then to process that file in Hadoop, 10 map tasks will be run on the 10 data blocks. The advantage, especially when dealing with unstructured data is that the data is easily loaded into HDFS as it is. Of course, the work will be how to extract the useful data out of the files, but now the data is distributed across the cluster and can be processed in parallel using MapReduce jobs.

More details on HDFS and its architecture can be found at:
http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

# Storing a File in HDFS

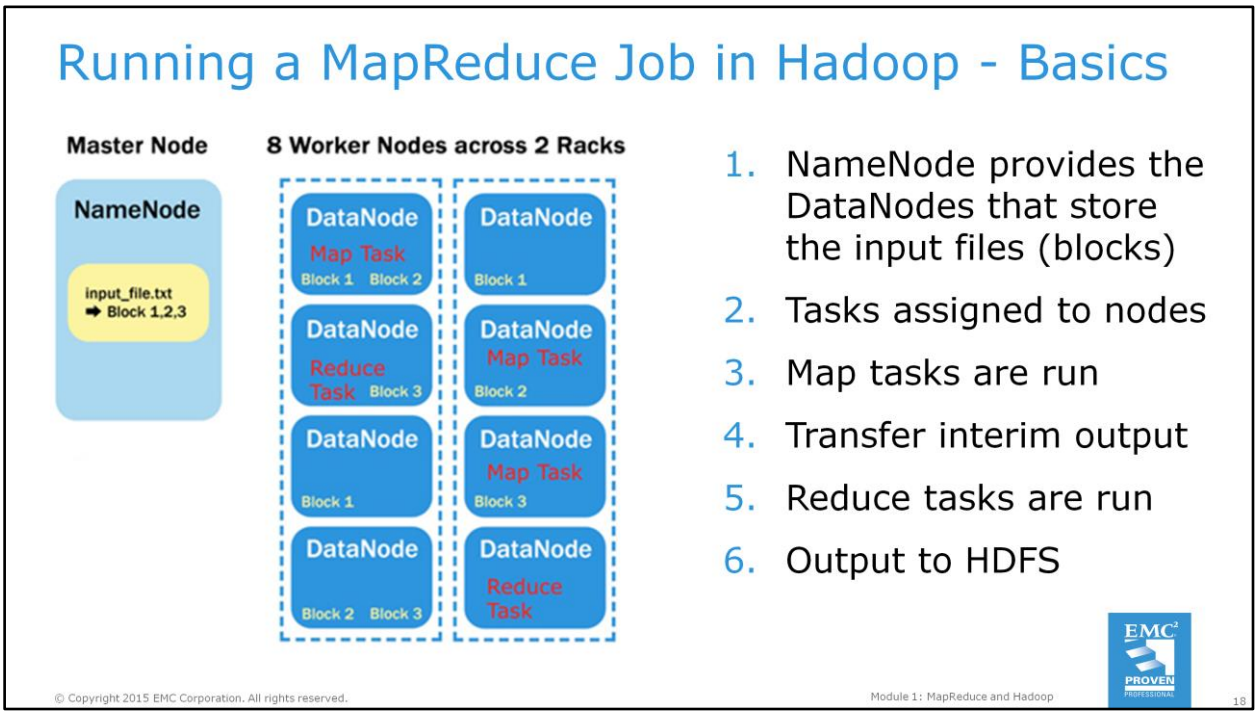



- Example: input_file.txt
  - Stored in 3 blocks
  - Replicated 3 times
  - Across the racks




In this example, input_file.txt consists of three blocks in HDFS. Each block is replicated three times across the worker nodes running the DataNode daemons. Running in memory, the NameNode keeps track of where each block is stored for each file stored in HDFS.

Replication provides: 1) redundancy for data protection and 2) scheduling flexibility for which nodes perform the map tasks.




The following describes the typical execution of a MapReduce job.

Typically, the input files are loaded into HDFS on the cluster. By default, a data file is broken up into 128 MB blocks and distributed among the cluster. Also, by default, each block is replicated for a total of three copies. These redundant copies result in data availability and in performance gains in terms of being able to better distribute the processing workload. The NameNode keeps track of which data blocks correspond to which file names.

The MapReduce job submission includes: mapper and reducer functionality, the input file locations, any changes to default configurations, and the output destination. The map and reduce tasks are assigned to the appropriate nodes.

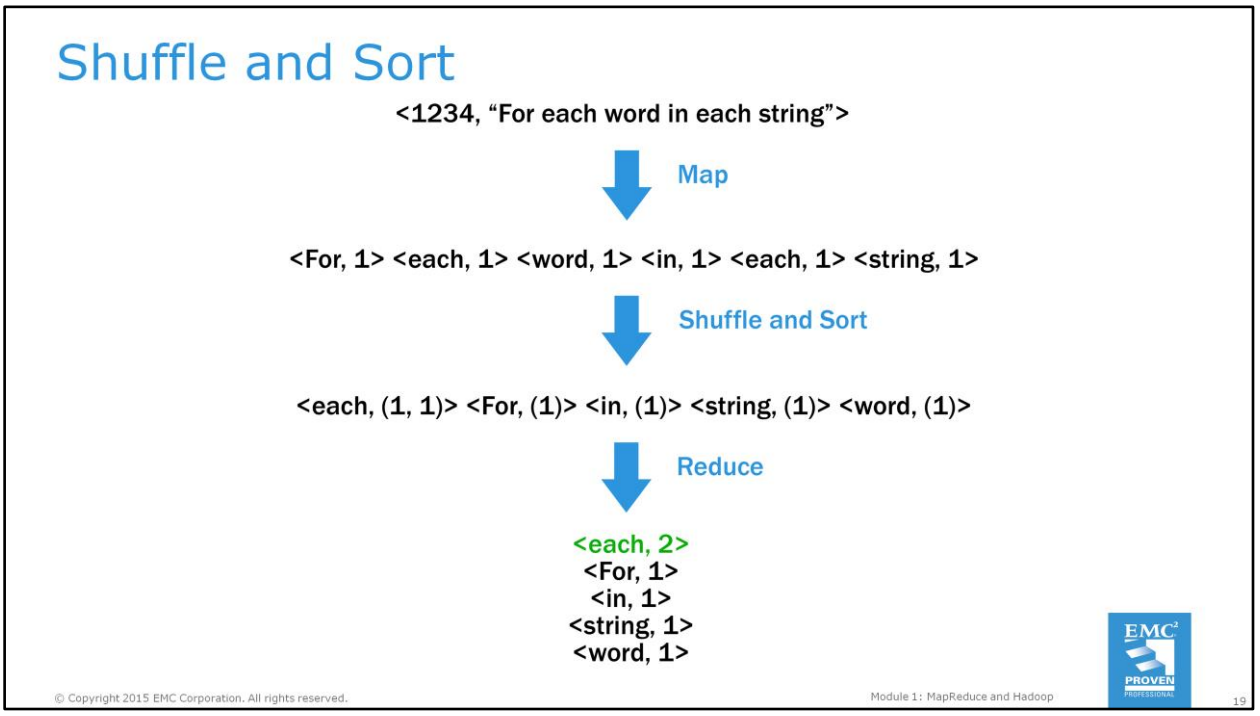
The map tasks are assigned to the data nodes based on where the data blocks reside and cluster performance considerations. The Mapper output is stored in a 100 MB buffer and dumped to disk if too large.

As each Map task completes, the map task output is arranged by the outputted key and moved to the assigned reducer node. As the data arrives at the reducer node, the reducer task begins to sort the results by the key. This processing is known as the shuffle and sort.

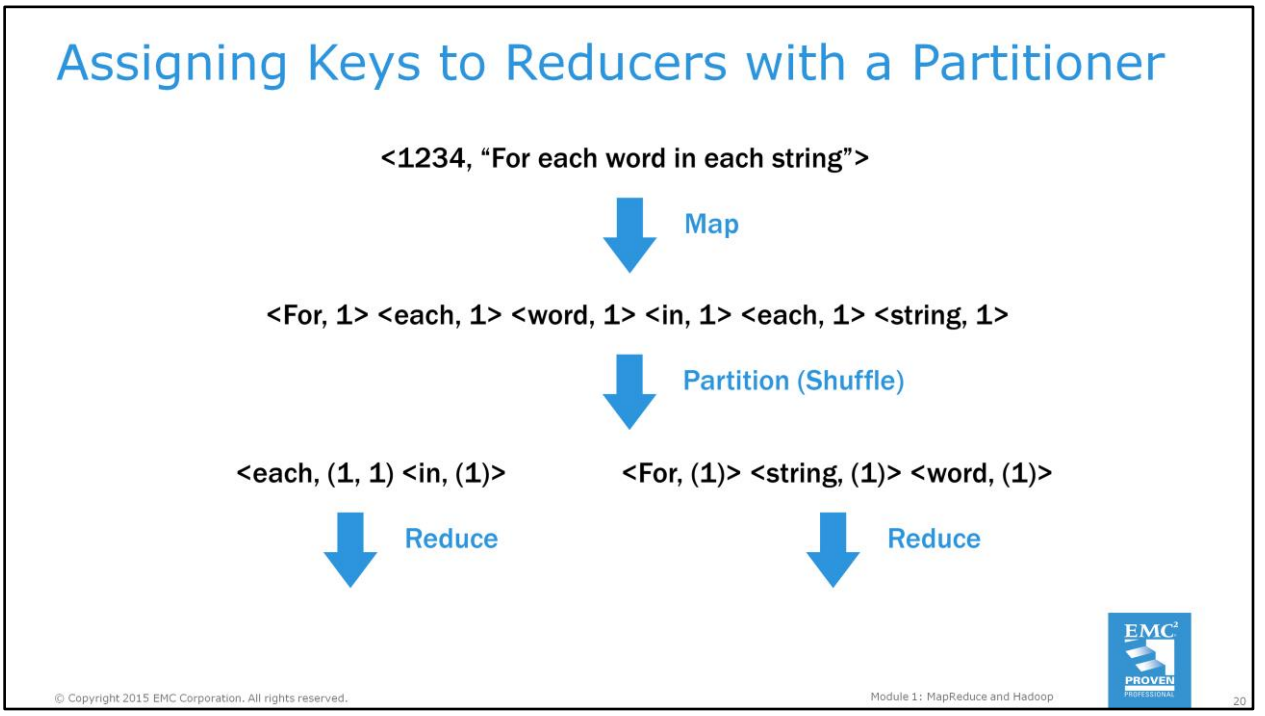
Once every Map task completes and their outputs are shuffled and sorted, the reduce tasks begin.

The reducer output files are written to HDFS, and a separate file is created for each reducer. If two reducers are run, two output files are created and stored in HDFS.

More details can be found at the following MapReduce tutorial:
http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

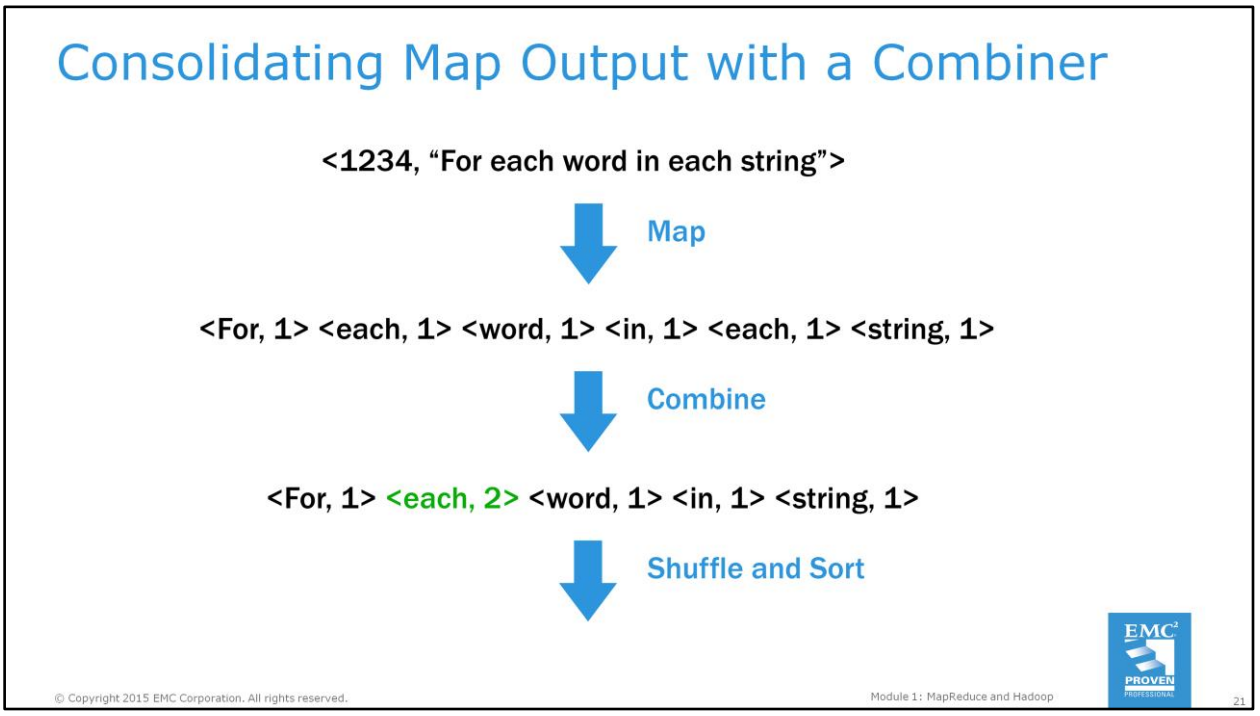



The transfer of the interim output from the map tasks is known as the shuffle and sort. For each key, the shuffle and sort provides a list of associated values. For example, since the word "each" appears twice, the reduce step provides a key/value pair <each, (1,1)> where (1,1) denotes a list of the two values from the map step.

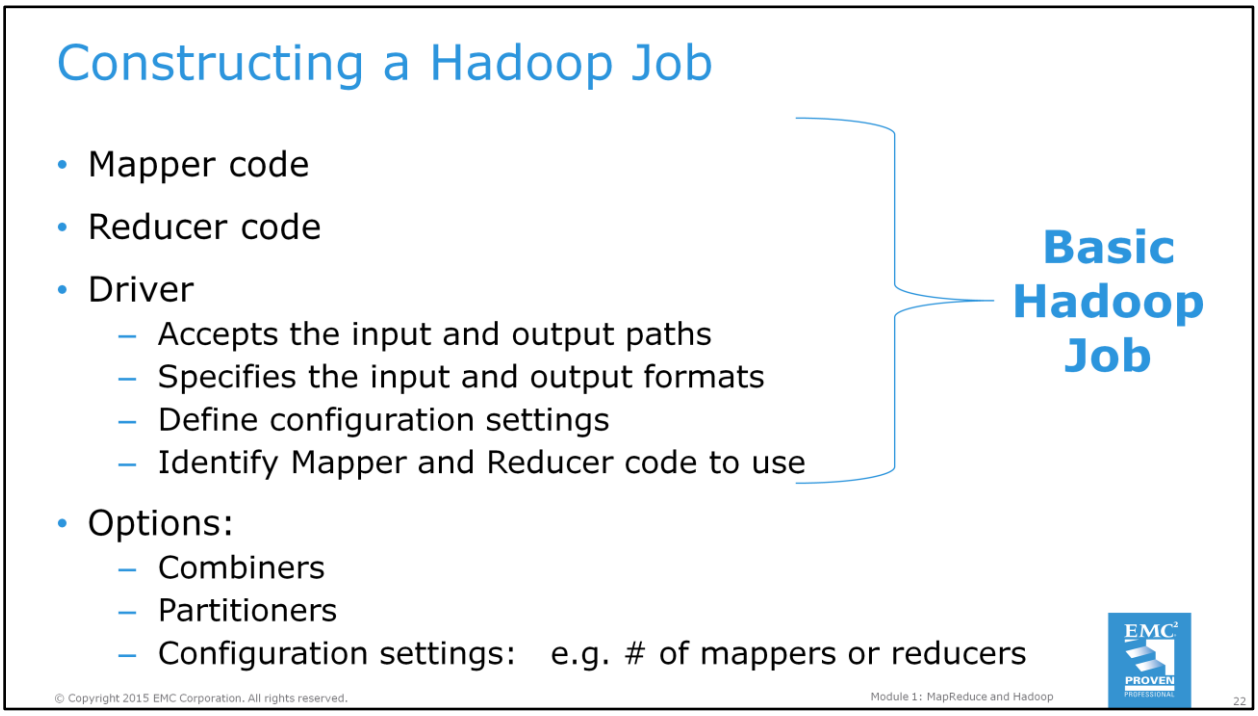



A partitioner is used to identify which keys are sent to which reducer. In this example, the partitioner sends the words beginning with a vowel to one reducer and the words beginning with a consonant to the other reducer. In practice, partitioners are used to evenly distribute the work load across the reducers. If one reducer is processing 90% of the intermediate values, the completion of the Hadoop job will likely run until that one reducer completes.

A second reason to define a partitioner is that the results need to be grouped in some manner. For example, if sales data need to be summarized for each of the last 4 quarters, a partitioner can be defined to direct each quarter's sales data to the appropriate reducer.



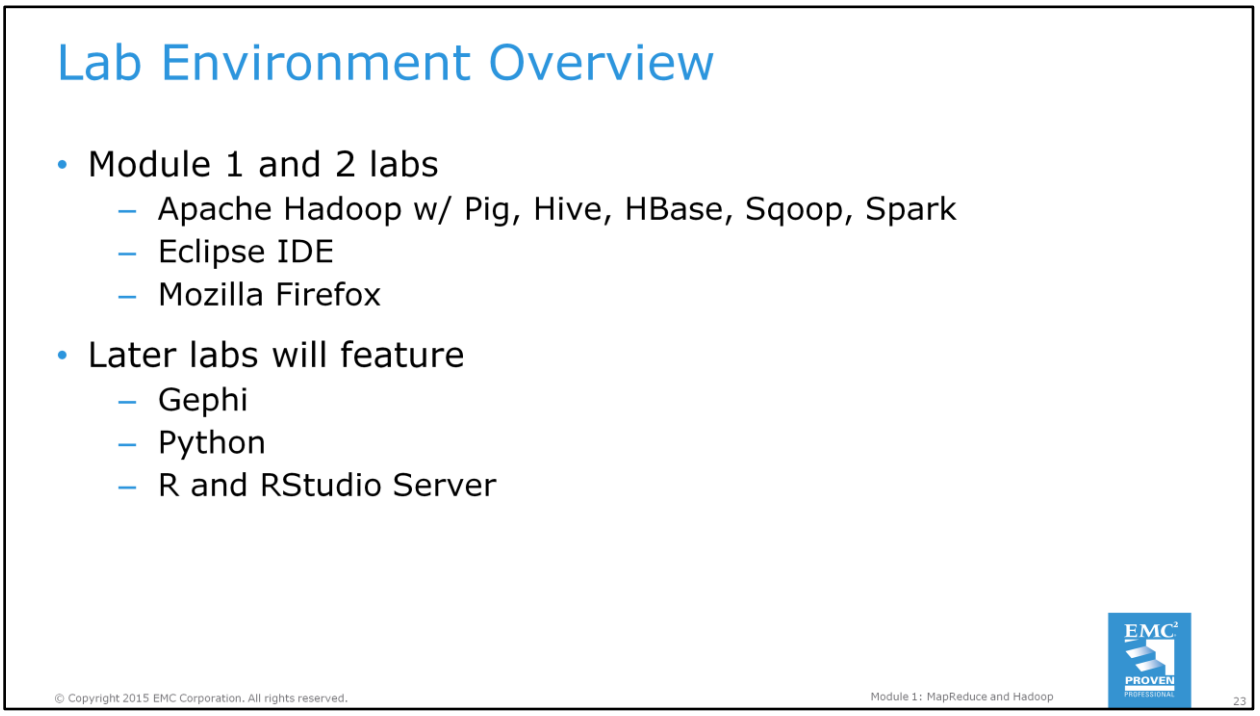

A combiner is optional functionality that can be added to a MapReduce job. A combiner allows some preprocessing of each map task prior to the shuffle and sort. In the word count example, a combiner would perform the same reduce operation by summing the occurrences of each word. It is very common for the combiner functionality to be identical to the reducer functionality. The use of a combiner is illustrated in the forthcoming lab exercise.




There are three pieces to a building a Hadoop Job: the Mapper code, the Reducer code, and the Driver which wraps everything together. Using Java, the three pieces of code are assembled into a JAR (Java Archive) file.

For users not comfortable coding in Java, Hadoop streaming can be used to run Hadoop jobs where the mapper and reducer are written in another language. A key requirement is that the language accepts input and output from stdin and stdout.

More details on Hadoop streaming is provided at: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/HadoopStreaming.html




The instructor will provide the necessary details on accessing the lab environment. Details on the software installed in the environment is provided in the README file.

For the Module 1 and Module 2 labs, the key software packages are:

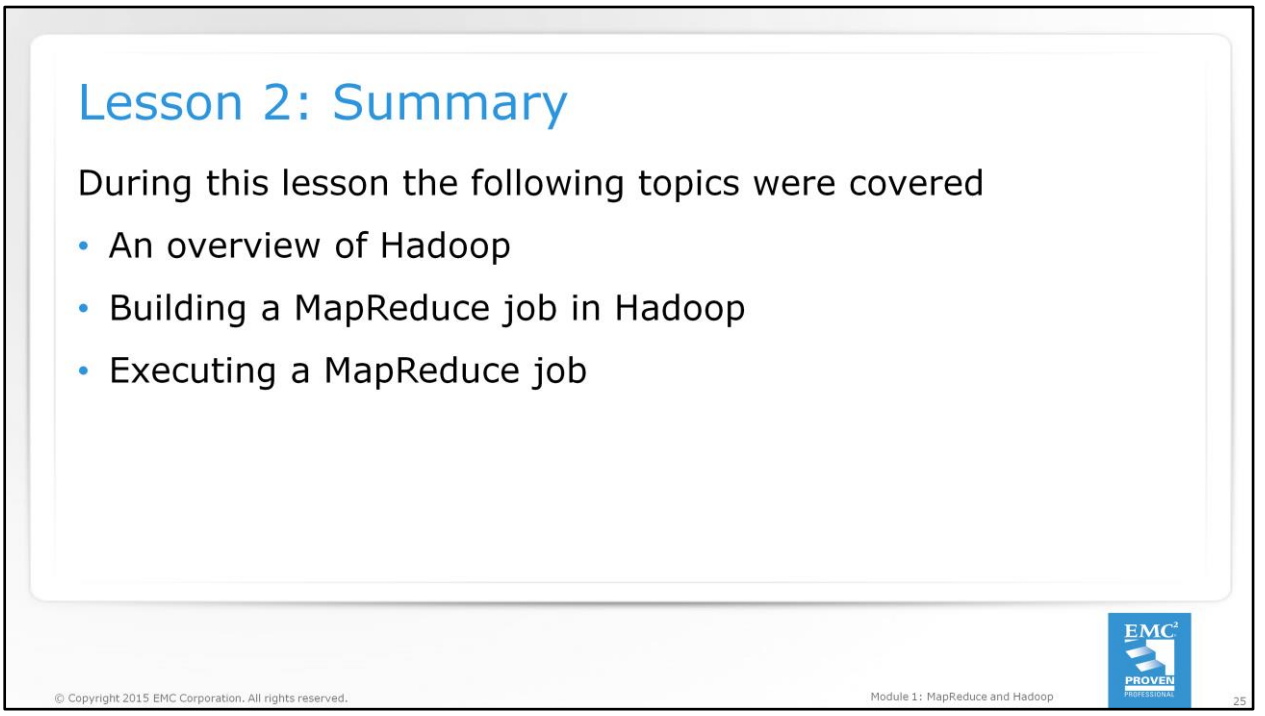
- Apache Hadoop Pig, Hive, Zookeeper, HBase, Sqoop, Mahout, Spark
- Eclipse – a Integrated Development Environment (IDE) for Java
- Mozilla Firefox

# Lab 1: Executing a MapReduce Job

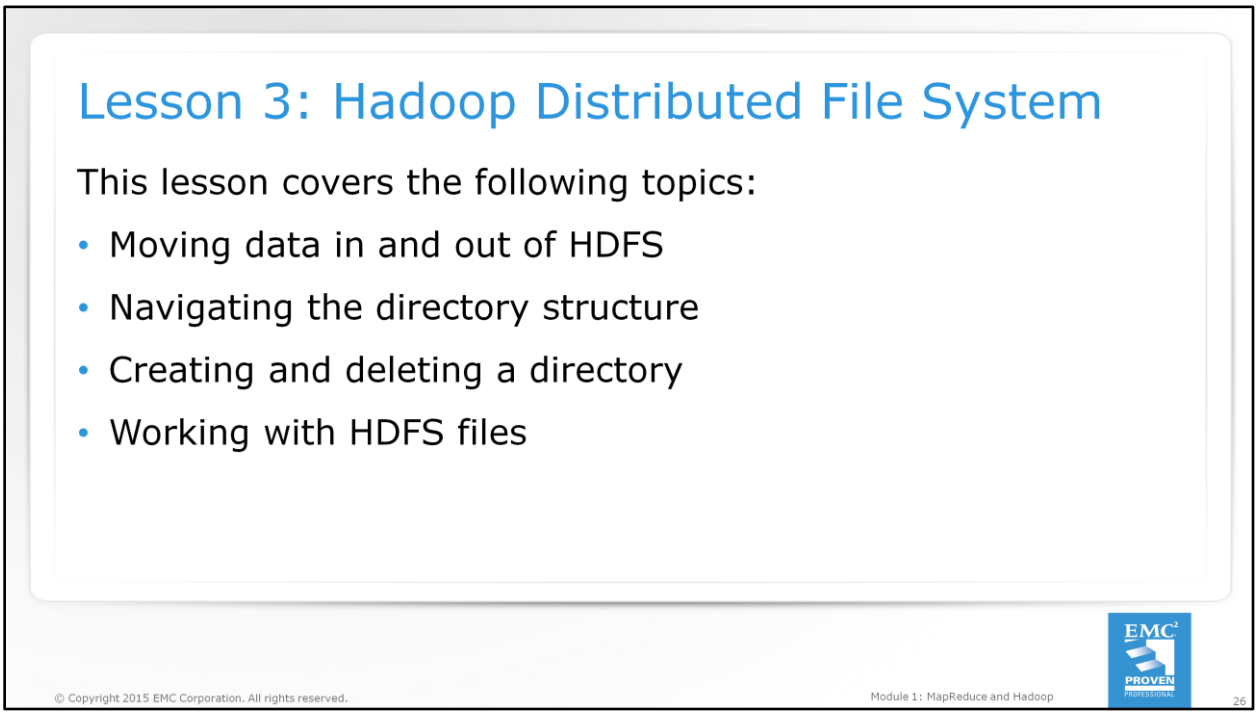
- Build a JAR file to perform word count
- Submit the MapReduce job
- Review the output
- Modify the existing word count code
  - Ignore certain words
  - Add a combiner
  - Explicitly define the number of reducers
- Build a new JAR file
- Submit the revised job




This lab exercise will guide the student through the steps of building and running a MapReduce job. Although it is not required to write any Java in these exercises, the student is encouraged to review the provided Java code to identify the basic mechanisms with the mapper, reducer, and driver code.




This lesson provided the basic steps of building and executing a MapReduce job in Apache Hadoop.




The previous lesson provided a brief overview of how files are stored in HDFS by breaking them into replicated blocks across the cluster. This lesson focuses on how files are moved in and out of HDFS as well as how HDFS files can be accessed.

# Hadoop Distributed File System (HDFS)

- Stores any file
- Distributed storage
- Key characteristics
  - Exists on top of a low level file system (e.g. xfs4 for Linux)
  - Scalable
  - Files are written, but not modified
  - Fault tolerant storage
- MapReduce is not required to move files into or out of HDFS




HDFS is a system for storing and accessing files. HDFS is designed to take advantage of a distributed cluster of nodes in performing parallel processing, such as MapReduce. HDFS is also designed to prevent data loss by storing multiple copies of a file; the default is 3 copies. As described earlier, a file is stored as 128 MB blocks. Thus, a MapReduce job can process the individual blocks of a very large file, in parallel.

HDFS is ideal for WORM files that are write once, read many times. HDFS is not intended for files that need updates or modifications. HBase is an option for storing data that may need to be updated frequently and in a random-access manner. HBase is discussed further in Module 2.

# HDFS Features

- Namenode federation
  - Multiple, independent NameNodes
  - Enables isolated pools of storage under each NameNode
  - Provides scalability and isolates the NameNode workloads
- Snapshots
  - Enables backup copies of an HDFS directory's contents
    - Stores only the list of the blocks and the associated files sizes
    - Data blocks are not copied
  - Protects against user-errors or other disasters
- User-specific permission control
- Encryption




As more files are added to a Hadoop cluster, the NameNode needs to keep track of more and more files and the associated blocks distributed across the data nodes. To allow the cluster to better scale, the NameNode federation creates multiple NameNodes that are independent of each other. Thus, certain HDFS directories can be managed by specific NameNodes under specific pools of storage. This configuration helps to ensure that the demand placed on each NameNode can be more predictable as new workloads are added to the Hadoop cluster.
http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Federation.html

To protect against unintentional loss of data, HDFS provides snapshot capabilities which track the file blocks and their locations in order to recover a file or directory's contents at a particular point in time.
http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsSnapshots.html

Also, HDFS has built-in features to control users' access and permissions as well the ability to encrypt the data.
http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/TransparentEncryption.html

http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsPermissionsGuide.html

The next several slides will cover several key HDFS shell commands to move files between a local file system and HDFS.

# Loading Local Files into HDFS (put)

```
hdfs dfs -put <local_source>  <HDFS_destination>
```

Examples:

```
$ hdfs dfs -put local_file.txt new_file.txt
$ hdfs dfs -put local_file.txt
$ hdfs dfs -put local_file.txt tgt
$ hdfs dfs -put local_file.txt tgt/new_file.txt
```




The put command is used to load a file from a local directory into HDFS. A file can be copied with the same file name or an alternate file name into HDFS. In earlier versions of Hadoop, `$ hadoop fs …` was used to access HDFS files and directories. This approach has been deprecated in favor of `$ hdfs dfs …`.

More details on the put command and other HDFS shell commands can be found by typing `hdfs dfs –help` at the command prompt or by visiting http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html.

# Extracting Files from HDFS (get)

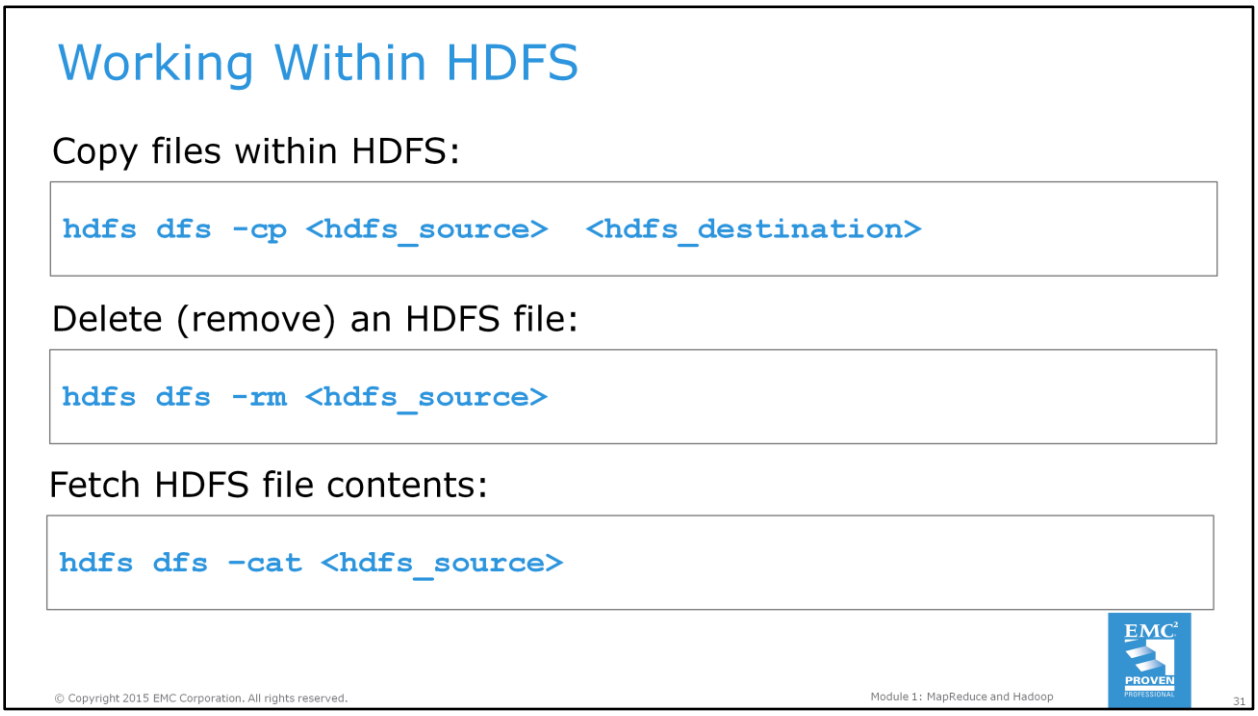
```
hdfs dfs -get <hdfs_source>  <local_destination>
```

Example:

```
$ hdfs dfs -get output/WCoutput/part-r-00000 MR_output.txt
```

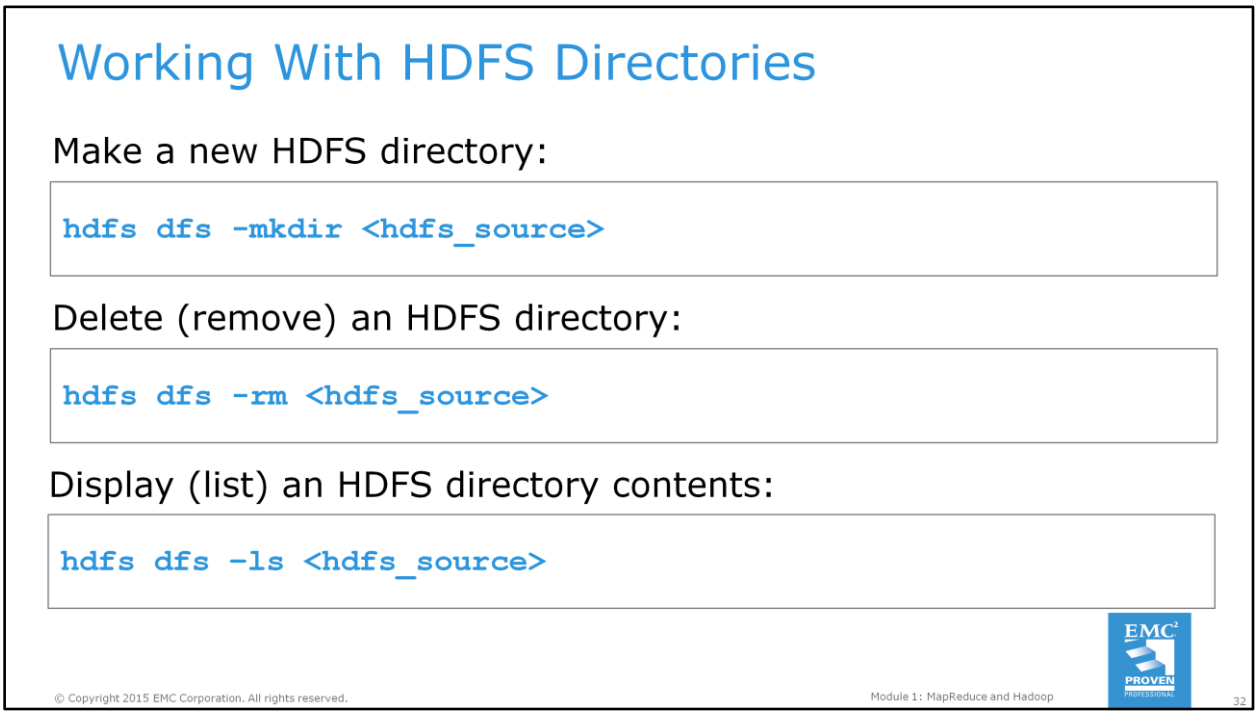



The get command moves an HDFS file into a local file system. In this example, the MapReduce output file, part-R-00000, is copied from HDFS as MR_output.txt and into the current local directory.




For large scale movement of data within HDFS, Hadoop includes a useful tool, DistCp, for performing distributed copies within or between clusters. The source directory structures and files are the input to a MapReduce job that will copy the files.

More details on DistCp is available at: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/DistCp.html




When working with the HDFS directory structure,

- `mkdir` is used to create (make) a new directory
- `rm` is used to delete (remove) a directory
- `ls` is used to list the contents of a directory

# Working with HDFS Files

- Hadoop and its ecosystem
  - MapReduce
  - Sqoop, Pig, Hive, Mahout, and Spark
  - Hadoop file system (FS) java class
  - webHDFS
- SQL on Hadoop
  - Pivotal HAWQ
  - Cloudera Impala
- 3rd party applications
  - Often dependent on ODBC/JDBC drivers and Hive tables
  - RHadoop




In addition to MapReduce, Hadoop ecosystem members provide ways to access and process HDFS files. This course will address such tools as Sqoop, Pig, Hive, Mahout, and Spark. At a more basic level, the Hadoop Java class, org.apache.hadoop.fs.FileSystem, provides the mechanisms for Java developers to read and write HDFS files. A simple HDFS example using this Java class is provided at: http://wiki.apache.org/hadoop/HadoopDfsReadWriteExample. Alternatively, the FileSystem class can be utilized with webHDFS REST API: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/WebHDFS.html.

In addition to Hive, there are several options for using SQL tools against HDFS datasets. Pivotal HD w/HAWQ is a parallel SQL query engine. HAWQ reads data from and writes data to HDFS natively. Ref: http://www.pivotal.io/sites/default/files/Hawq_WP_042313_FINAL.pdf

As an extension to Hive functionality, Cloudera Impala provides faster querying, particularly when the source data is formatted as a Parquet file. Ref: http://www.cloudera.com/content/cloudera-content/cloudera-docs/Impala/latest/Installing-and-Using-Impala/ciiu_parquet.html

Many 3rd party applications such as Tableau, SAS, and Excel can access data files stored in HDFS. As is the case with many such tools, Tableau uses Hive and its catalog of tables to import data. Ref: http://www.tableausoftware.com/solutions/hadoop-analysis. Also utilizing Hive, SAS provides its own connector SAS/ACCESS® to obtain HDFS data files. Utilizing its Power Query add-in, Microsoft Excel can import HDFS into a workbook.

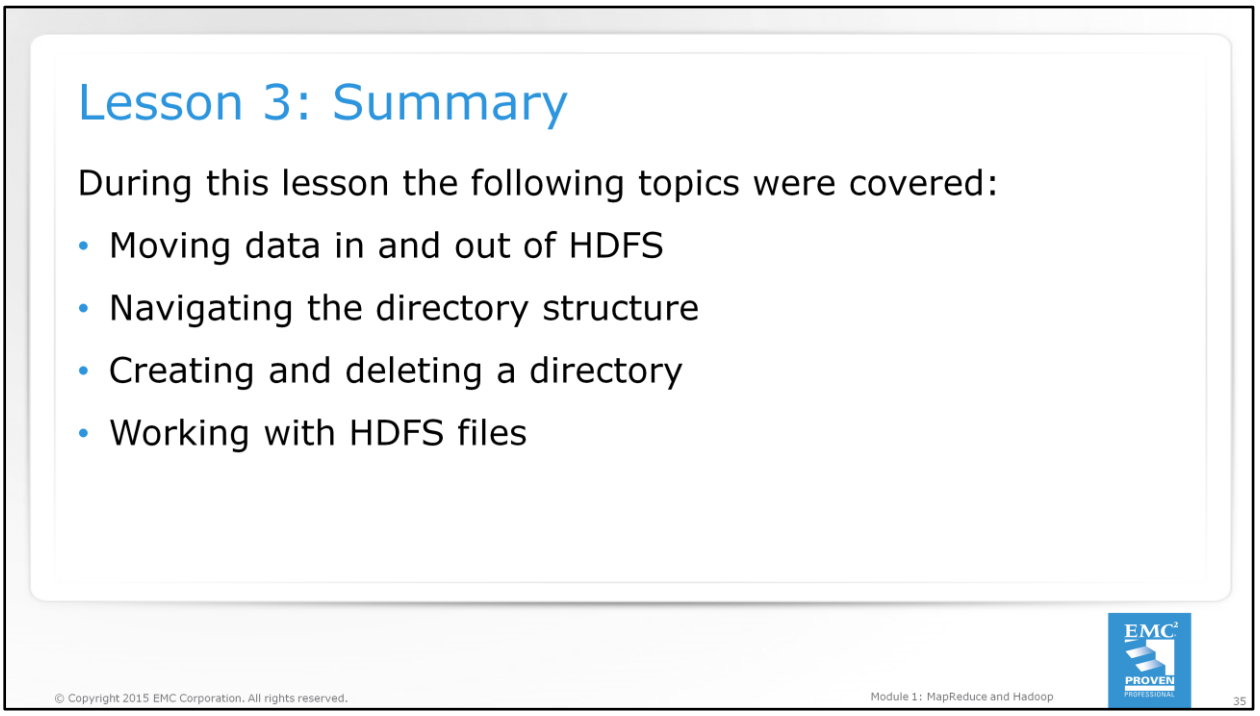
RHadoop consists of five R packages that enable users to run MapReduce jobs from R using the streaming API, as well as to work with HDFS and HBase files. Ref: https://github.com/RevolutionAnalytics/RHadoop/wiki

# Lab 2: Working with HDFS files

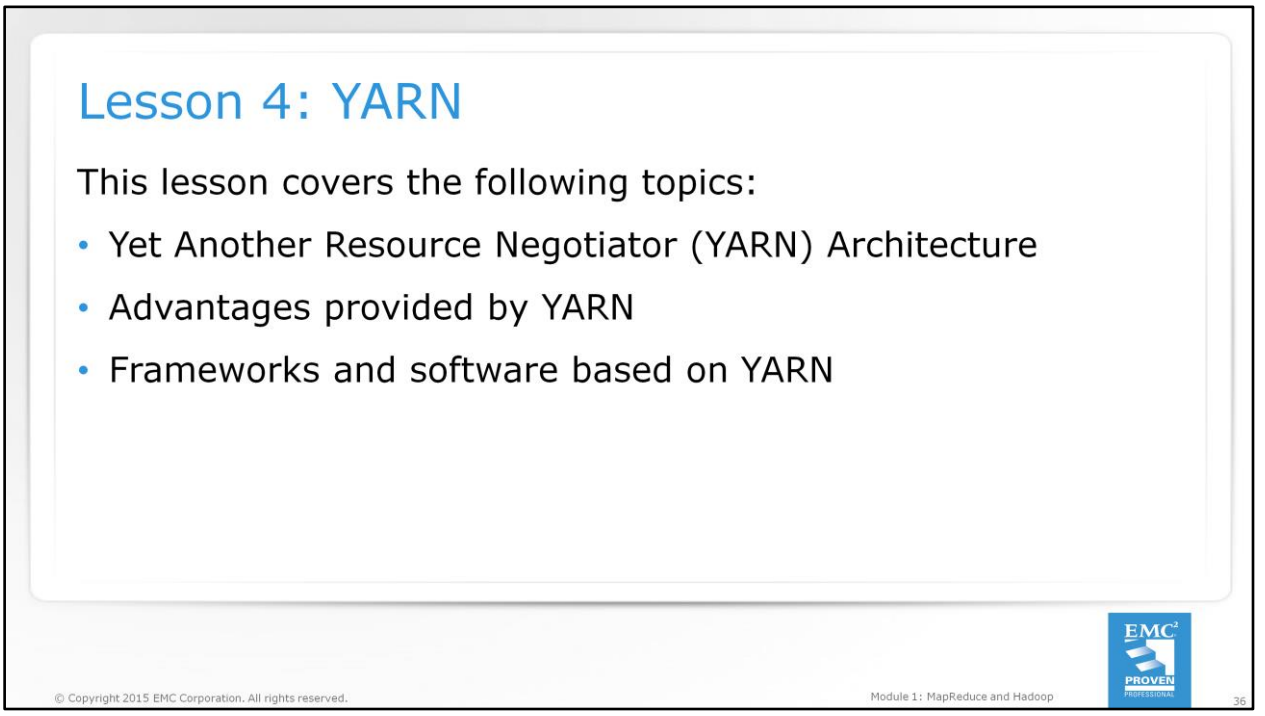
- Load a file into HDFS with the –put command
- Copy a file into the local file system with the –get command
- Use Sqoop to import data into HDFS from a RDBMS
- Using MapReduce
  - Merge (column-wise join) two HDFS files
  - Select a subset of joined records
  - Export subset to a CSV file in the local file system




Apache Sqoop enables the bulk transfer of data between HDFS and relational databases. In this lab, Sqoop is used to import Greenplum tables into HDFS. These tables will be joined using a MapReduce job. Finally, the results are exported to a CSV file for further downstream processing.




This lesson covered the HDFS shell commands to move data in and out of HDFS with the put and get command, respectively. This lesson also covered the basics of navigating the HDFS directory structure, as well as changing the directory structure. Finally, several options were provided on how HDFS files can be accessed beyond the basic shell commands.




This lesson describes YARN and the additional frameworks beyond MapReduce that YARN makes possible.

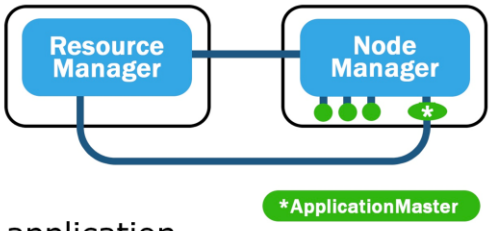
# YARN Overview

- Allocates and manages a Hadoop cluster's resources
  - Schedules the start of a new application
  - Responds to resource requests from each application
- Comparison to earlier versions of Hadoop
  - Cluster management is now separate from MapReduce
  - MapReduce API remains the same
- Advantages of YARN
  - Enables larger Hadoop clusters
  - Enables additional frameworks beyond MapReduce




In earlier versions of Hadoop, cluster management was tightly integrated into MapReduce. YARN provides the resource management and scheduling separated from MapReduce. This separation allows the cluster to scale better and enables frameworks beyond MapReduce to be executed on the Hadoop cluster.

# YARN Components




- ResourceManager
  - Receives job submissions
  - Allocates cluster resources
  - Starts ApplicationMaster for a new application
- ApplicationMaster
  - Requests resources for each task
  - Establishes the tasks to be run
- NodeManager
  - Provides containers per the ResourceManager
  - Executes and tracks the tasks running in its containers
  - Monitors general node health




The ResourceManager consists of two key parts: the Scheduler and the ApplicationsManager. The Scheduler is responsible for allocating cluster resources to run the submitted jobs. The ApplicationsManager starts, monitors, and when necessary, restarts each ApplicationMaster which oversees the execution of a job.

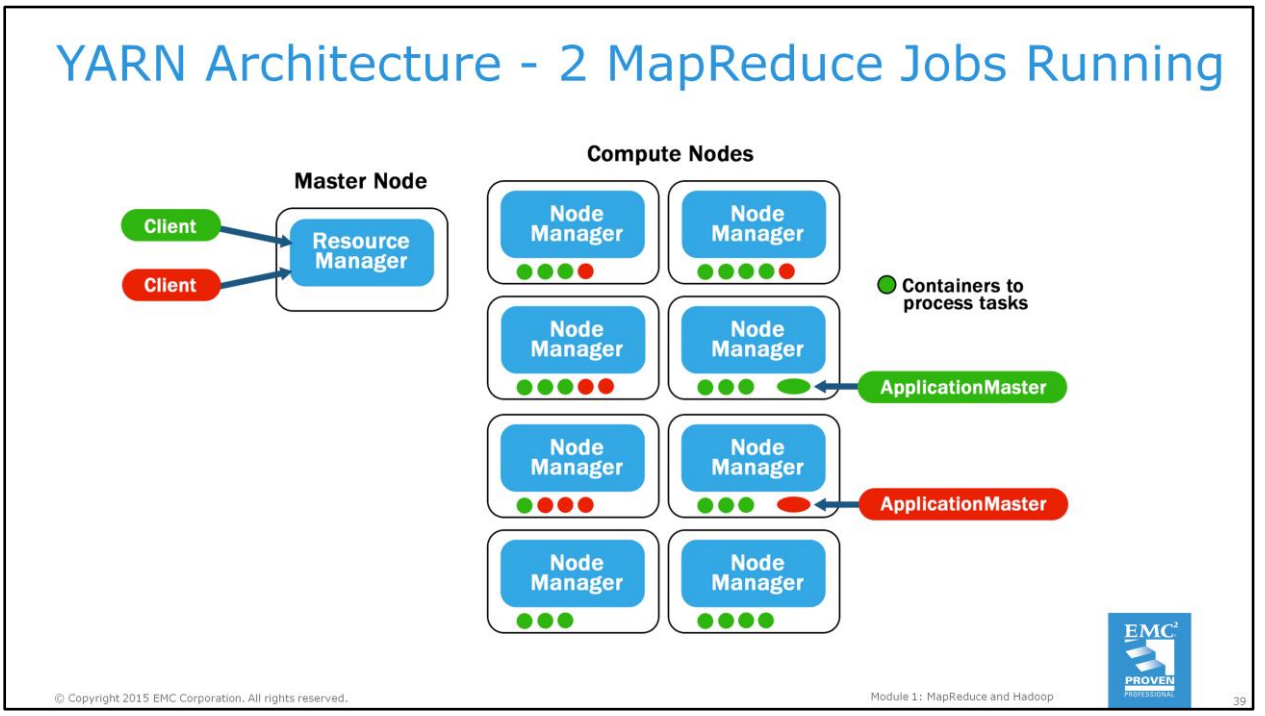
The ApplicationMaster builds an execution plan, requests resources, adjusts the execution plan based on the provided resources, and works with the appropriate NodeManagers to start and run the job tasks. An ApplicationMaster is framework specific. Hadoop provides an ApplicationMaster for MapReduce. Other frameworks to be run on the cluster will require their own ApplicationMaster. As Murthy, et al. stated, from YARN's perspective, an ApplicationMaster is nothing more than user code. Thus, an ApplicationMaster needs to be limited in its ability to affect the cluster's operations.

A NodeManager runs on each worker node. The NodeManager provides the negotiated resources to execute the tasks. These resources are provided as containers. A container is a collection of system resources such as CPUs, memory, and disk space.

The running tasks are monitored by the specific NodeManager. The statuses of the tasks are reported to the ApplicationMaster. A NodeManager also monitors the status and health of the ApplicationMaster which is run in its own container on one of the worker nodes.

Reference:

A. Murthy, J. Markham, V. Vavilapalli, & D. Eadline (2014). *Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2*. New York: Addison-Wesley.



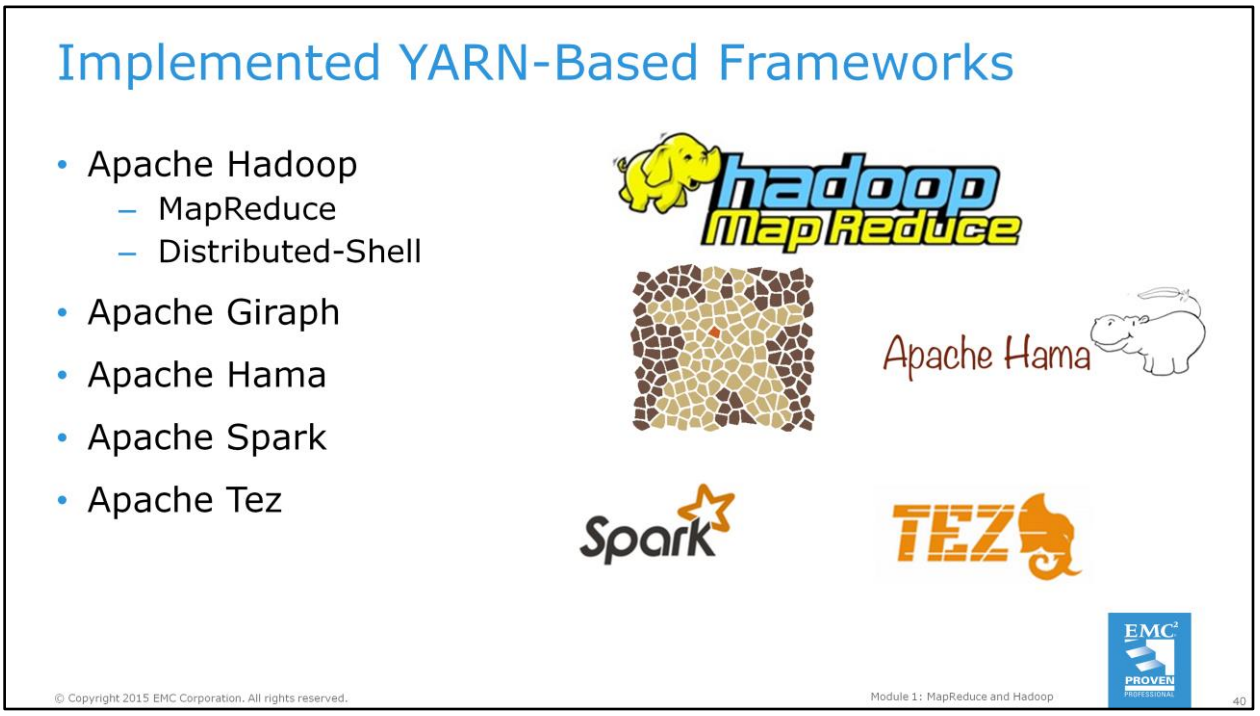

This diagram illustrates the case of two MapReduce jobs running on a Hadoop cluster.

Key Steps in a MapReduce Job Execution:

1. Client submits a job request to the ResourceManager
2. ResourceManager:
   a. Scheduler allocates container to run ApplicationMaster
   b. ApplicationManager starts ApplicationMaster
3. ApplicationMaster request resources from ResourceManager
4. ResourceManager allocates containers to run map and reduce tasks
5. NodeManagers establish the negotiated containers
6. ApplicationMaster assigns a map or reduce task to a container
7. NodeManager monitors the running tasks
8. Job Completes
9. Containers are deallocated

Reference:

"Apache Hadoop NextGen MapReduce (YARN)." *The Apache Software Foundation.* (2014). http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html




In addition to the MapReduce framework, Apache Hadoop includes a the Distributed-Shell application for running shell commands on the Hadoop cluster.

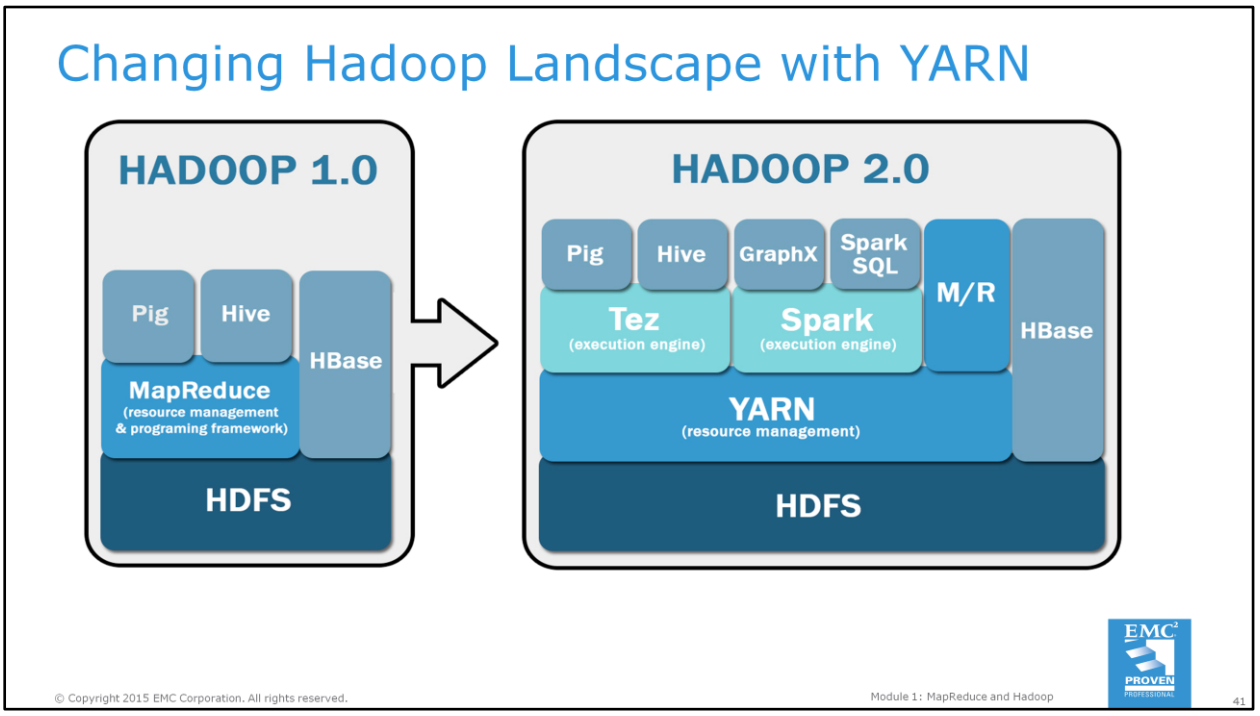
Apache Giraph enables the processing of graphs using the bulk synchronous parallel (BSP) model of distributed computing. Ref: http://giraph.apache.org

Apache Hama provides a general BSP framework to perform analytics. Ref: http://hama.apache.org

Apache Spark provides over 80 high-level operators for building parallel applications written in Java, Scala, or Python. Such Spark-based applications include Spark SQL and GraphX. Ref: http://spark.apache.org. A separate Spark lesson is provided in Module 2.

Apache Tez enables the execution of a directed-acyclic graph (DAG) of tasks. One Tez job can replace the processing that was previously accomplished with multiple MapReduce jobs. Ref: http://tez.apache.org

For additional frameworks, visit http://wiki.apache.org/hadoop/PoweredByYarn



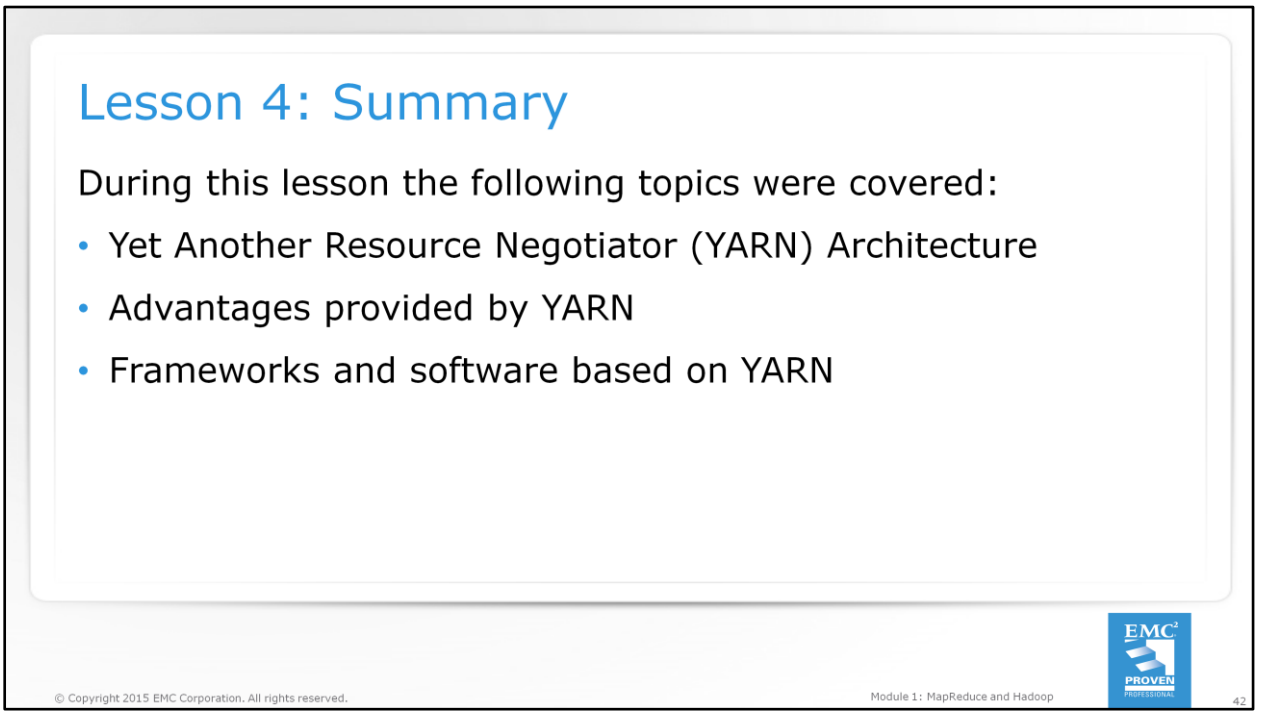

Before YARN, MapReduce was responsible for managing the cluster resources of the Hadoop cluster. Most analytical processing was limited to MapReduce jobs possibly based on Pig Latin or HiveQL commands. With Yarn, additional tools to implement additional frameworks are being built. These tools include:

Apache Tez – more efficiently executes Pig and Hive code

Apache Spark – a DAG engine that enables other tools such as:

- GraphX
- SparkSQL

Additionally, YARN enables production HBase environments to run on the same cluster that is running MapReduce jobs. YARN can dedicate enough resources to applications like HBase to prevent contention issues with any concurrent MapReduce jobs.



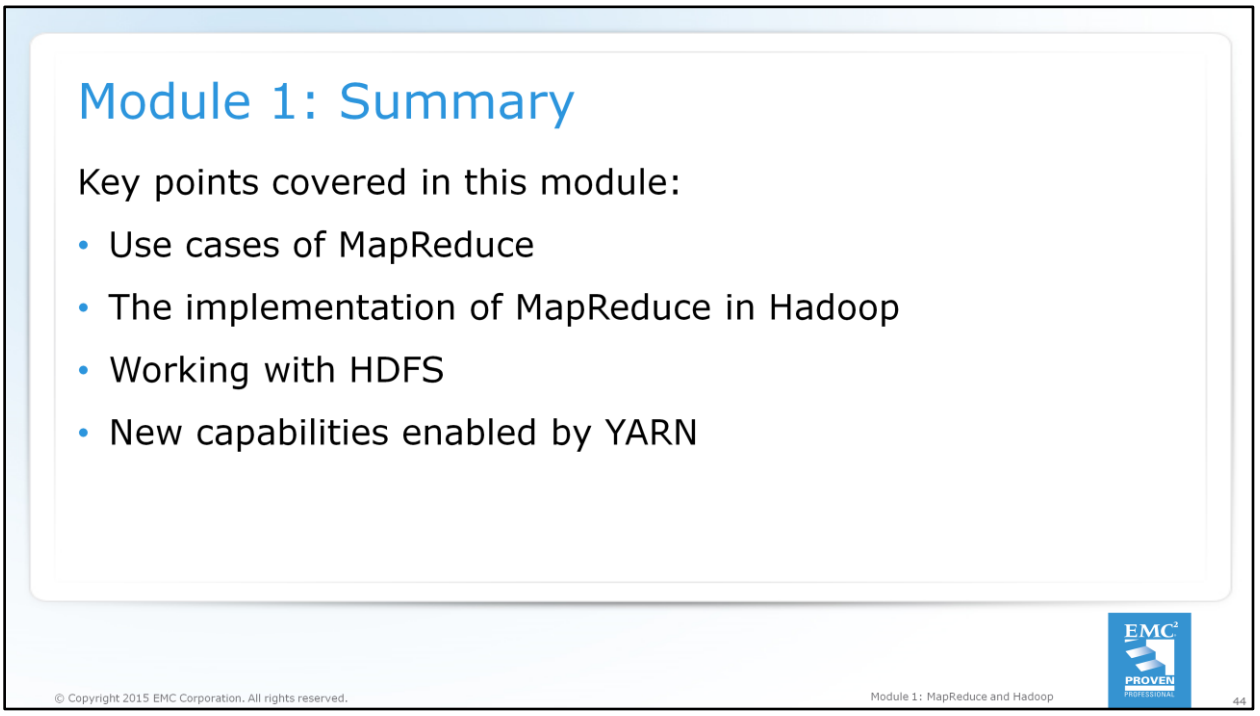

This lesson provided an overview of YARN, the separation of MapReduce from the services that manage and coordinate the applications running on the cluster, and the new frameworks beyond MapReduce that are being developed.

# Check Your Knowledge

1. Name four modules of Hadoop and their functions.
2. What type of analytic problem does MapReduce best address?
3. What tool transfers data between Hadoop and relational databases?
4. Describe two advantages provided by YARN.




Write the answers here.




This module presented how Apache Hadoop implements MapReduce, as well as stores data using HDFS. YARN enables additional frameworks beyond MapReduce to be executed on a Hadoop cluster. Providing DAG execution under YARN, Apache Spark will be examined in Module 2.