HDFS

HDFS goals and features

HDFS goals and features

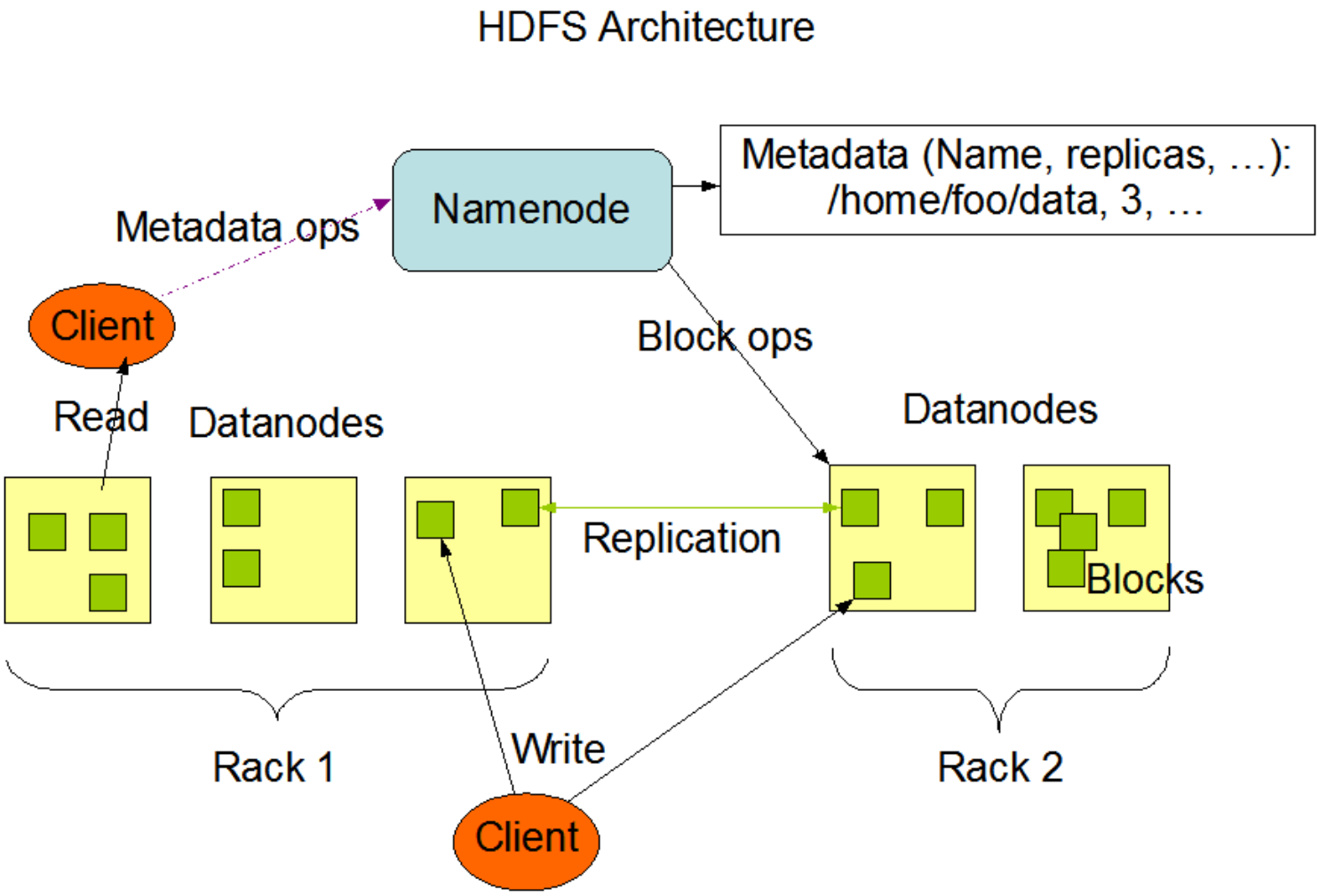
ZZEN9313_HDFS_Transcript.pdf

A brief introduction to HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.



The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

The File System Namespace

HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories. The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file. HDFS supports user quotas and access permissions. HDFS does not support hard links or soft links. However, the HDFS architecture does not preclude implementing these features.

While HDFS follows naming convention of the FileSystem, some paths and names (e.g. /.reserved and .snapshot) are reserved. Features such as transparent encryption and snapshot use reserved paths.

The NameNode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the NameNode. An application can specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. This information is stored by the NameNode.

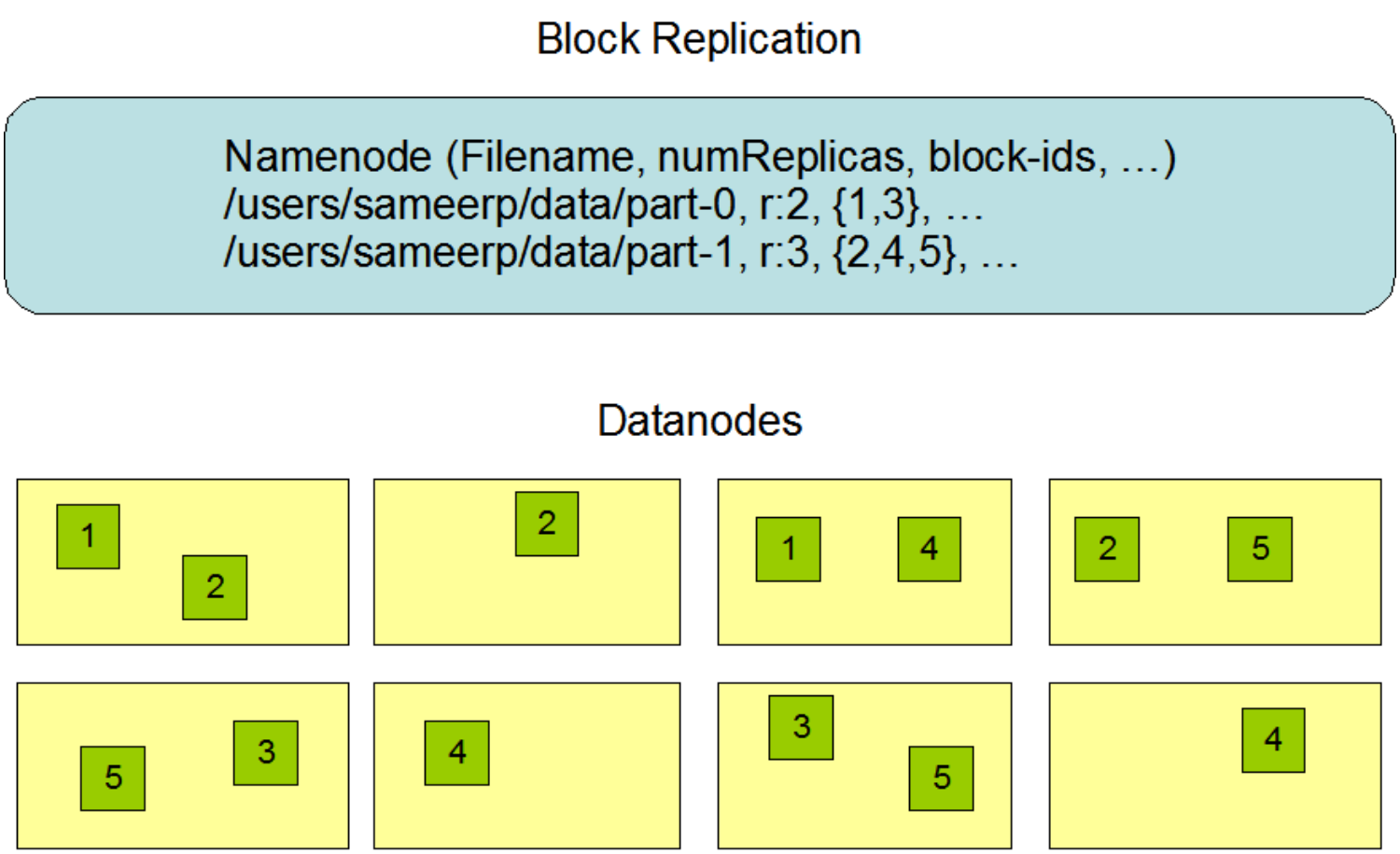
Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file.

All blocks in a file except the last block are the same size, while users can start a new block without filling out the last block to the configured block size after the support for variable length block was added to append and hsync.

An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once (except for appends and truncates) and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

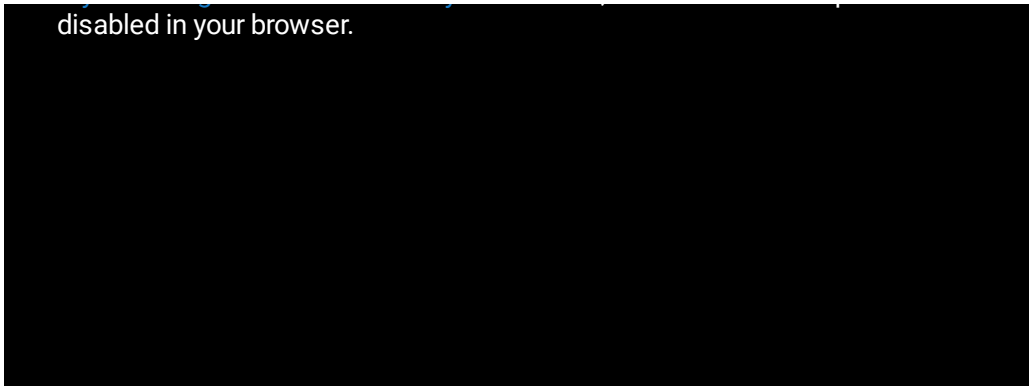


Data Disk Failure, Heartbeats and Re-Replication

Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

The time-out to mark DataNodes dead is conservatively long (over 10 minutes by default) in order to avoid replication storm caused by state flapping of DataNodes. Users can set shorter interval to mark DataNodes as stale and avoid stale nodes on reading and/or writing by configuration for performance sensitive workloads.

The below video is very interesting which explains HDFS using Lego. It might provide you a better understanding of HDFS.


disabled in your browser.

HDFS Architecture (Optional Reading)

Using HDFS

Now let's get some hands-on experience using Hadoop to manage files that are distributed across a cluster of nodes.

When you click the panel on the right you'll get a terminal connection to a server that has Hadoop installed. It's installed in standalone mode, which means that it's running on just that one server, rather than on a bigger cluster of computers. But it behaves as if it's running in a bigger cluster. This is a good way to learn about Hadoop.

Starting HDFS

Click on the panel to activate the terminal.

The first thing you need to do is start HDFS (the Hadoop file management system). Run the following command at the prompt in the terminal window (i.e. type or copy it at the prompt and then hit enter):

```
$ start-dfs.sh
```

You have just run a **Linux** command. Linux is the operating system on the server you are connected to. Linux is an alternative to the operating systems Windows and MacOS, and is very commonly used as the operating system for servers. To work with Hadoop and HDFS, and the other pieces of software that are running on the server, you will have to use a few Linux commands. Don't worry - they're pretty simple, and will be explained.

Linux uses a dollar sign ($) as a prompt. Throughout these slides, when you see $ followed by a command you know its a Linux command.

To check that HDFS has started successfully, run the following command:

```
$ jps
```

The command "jps" is short for "Java process status" - it lists all Java processes that are running. Since Hadoop nodes run as Java processes, you should see the following items on the list:

```
DataNode
SecondaryNameNode
NameNode
```

Your default directory in HDFS

In HDFS, your default directory is `/user/{your username}`. On Ed, your username is `user`. Thus, your default directory is `/user/user`. To see the contents of this directory, use the following command ("ls" is short for "list"):

```
$ hdfs dfs -ls
```

Initially you'll get an error message saying there's no such file or directory. This is because your default directory does not yet exist. You can create it using the `mkdir` command (short for "make directory").

First create the directory `/user`, then create directory `/user/user`:

```
$ hdfs dfs -mkdir /user
$ hdfs dfs -mkdir /user/user
```

Alternatively, you can create both directories in one step by adding the `-p` option, which tells Hadoop to automatically create parent directories as required ("p" is for "parents"):

```
$ hdfs dfs -mkdir -p /user/user
```

Now try listing the contents again:

```
$ hdfs dfs -ls
```

You will get an empty result set, because there is nothing in this directory yet (but you won't get an error this time).

Rather than re-typing a command that you have already run, you click the up arrow - Linux will scroll through the previous commands, and when you get to the one you want you can just hit enter.

Adding a file

In addition to HDFS the server also has its own regular file system, called its **local file system** (note: "local" here means local *to the server* you are connected to - it does mean local to your personal computer, the one you are now using). There is file called "text.txt" in your home directory in this local file system. You can check the contents of your home directory by running the following command (note that there is no dash "-" in front of "ls" when you are working with the local file system):

```
$ ls
```

You should see the file "text.txt" listed.

You can copy this file into your default directory in HDFS using the `-put` command, followed by the name of the local file that you want to copy to HDFS:

```
$ hdfs dfs -put text.txt
```

You can check that the operation was successful by listing the contents of your default directory again:

```
$ hdfs dfs -ls
```

You should see text.txt in the results. There is a lot of information about the file listed - the name of the file is at the right-hand end.

Creating sub-directories

You can create sub-directories in HDFS using the `-mkdir` command. For example, to create a subdirectory of your default directory called "files" use the following command:

```
$ hdfs dfs -mkdir files
```

You can check that the operation was successful by listing the contents of your default directory:

```
$ hdfs dfs -ls
```

You should now see the file "text.txt", which you added above, and the directory "files", which you just created.

Moving files

You move files in HDFS using the `-mv` command. Let's move the file "text.txt" from your default directory into the `files` subdirectory. Use the following command:

```
$ hdfs dfs -mv text.txt files
```

After `-mv` you specify the path of the file you want to move, and then the path of where you want to move it to.

To check that this worked, you can list the contents of your default directory as you have been doing - it should contain just the sub-directory `files:`

```
$ hdfs dfs -ls
```

You can list the content of `files` using the following command:

```
$ hdfs dfs -ls files
```

You should see the file "text.txt" listed in the `files` directory.

Copying files

You can copy files in HDFS using the `-cp` command. Let's make a copy of text.txt in `files`, keeping it in the same directory, and calling it "text2.txt":

```
$ hdfs dfs -cp files/text.txt files/text2.txt
```

To check that this worked, try listing the contents of the `files` directory:

```
$ hdfs dfs -ls files
```

Renaming files

To rename a file, just move it to the same directory but with a different name. Let's rename "text2.txt" to "moreText.txt":

```
$ hdfs dfs -mv files/text2.txt files/moretext.txt
```

To check that this worked, try listing the contents of the `files` directory:

```
$ hdfs dfs -ls files
```

Removing files

You can remove a file using the `-rm` command. Let's remove the file "moreText.txt" from the `files` directory:

```
$ hdfs dfs -rm files/moretext.txt
```

To check that this worked, try listing the contents of the `files` directory:

```
$ hdfs dfs -ls files
```

Viewing files

To see the contents of a file you can use the `-cat` command ("cat" is short for "concatenate" - you can use `-cat` to concatenate the contents of files):

```
$ hdfs dfs -cat files/text.txt
```

Getting files

To copy a file from HDFS to the local file system on the server you can use the `-get` command. The following command will copy the file "files/text.txt" in HDFS to "text2.txt" in your home directory in the local file system:

```
$ hdfs dfs -get files/text.txt text2.txt
```

To check that this worked, try listing the contents of your home directory in the local file system:

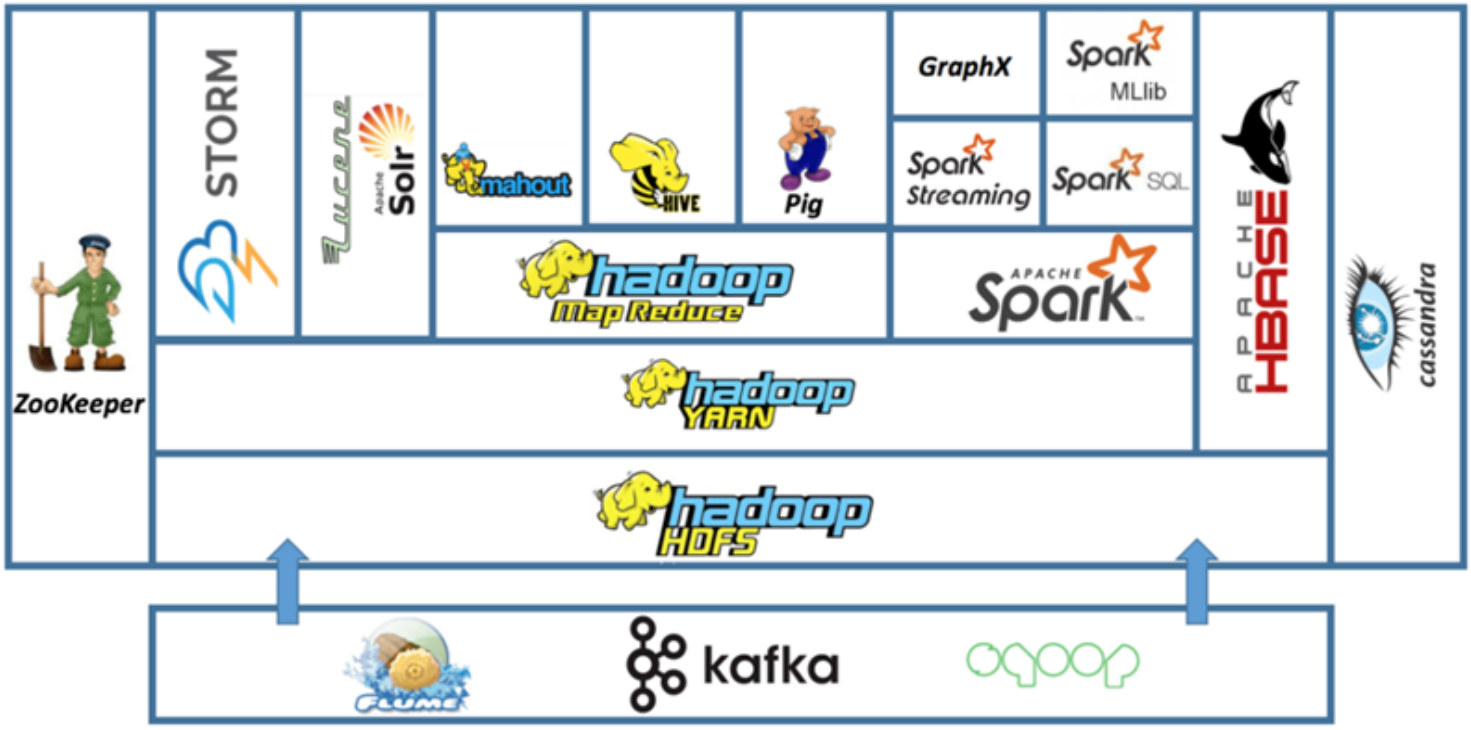
```
$ ls
```

More commands

You can find a complete list of HDFS commands at the Hadoop website:

https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/FileSystemShell.html

The Hadoop Ecosystem



(The above figure is from: https://www.softwaretestingclass.com/introduction-to-hadoop-architecture-and-components/)

The core of Hadoop is HDFS (for storing data), MapReduce (for processing data), and YARN (for scheduling tasks). These are powerful and popular, but they can be difficult to work with. So quite a few pieces of software have been developed and added to what's called the **Hadoop ecosystem**.

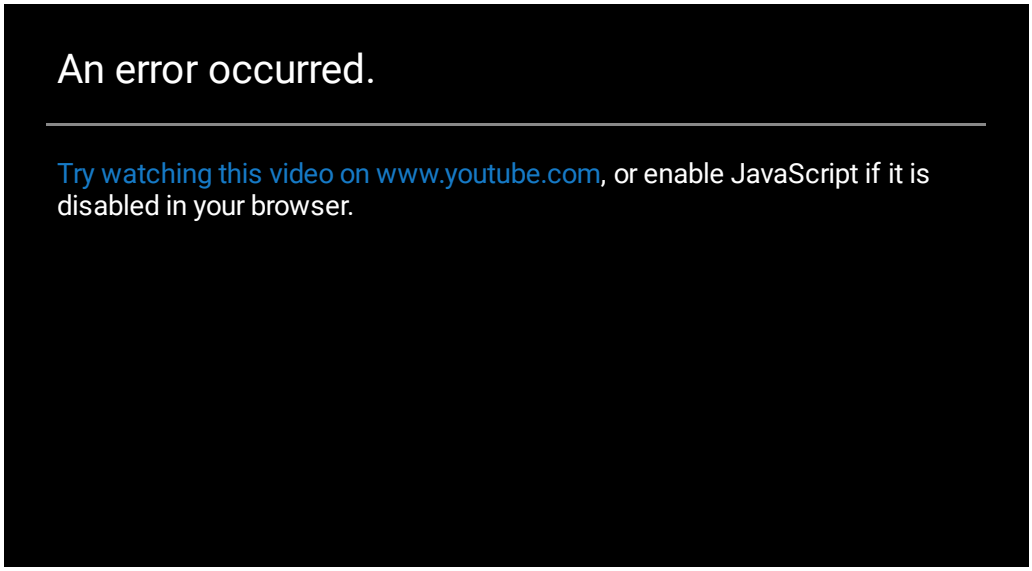
Mesos is an alternative to YARN. **Oozie** provides a way to schedule Hadoop operations. **Sqoop**, **Flume**, and **Kafka** provide ways of getting data from an external source into your Hadoop cluster. **MRJob** is a Python library that allows you to write and run mapper and reducer functions more easily in Python. **Hue** provides an alternative way to program MapReduce jobs. So do **Pig** and **Hive**, but they allow you to write SQL queries for your data which are automatically turned into MapReduce jobs. **Storm** allows you to process streaming data. Mahout allows you to do machine learning. **HBase** allows you to manage a (non-relational) database on your Hadoop cluster.

Next week you will learn how to use two of the most popular of these - MRJob and Hive. These will draw upon your Python and SQL skills.

If you are interested to know some more details about these tools, You may find the following blog post helpful:

EduPristine blog post about the Hadoop ecosystem

The following video may also help you better understand the Hadoop Ecosystem.


An error occurred.
Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

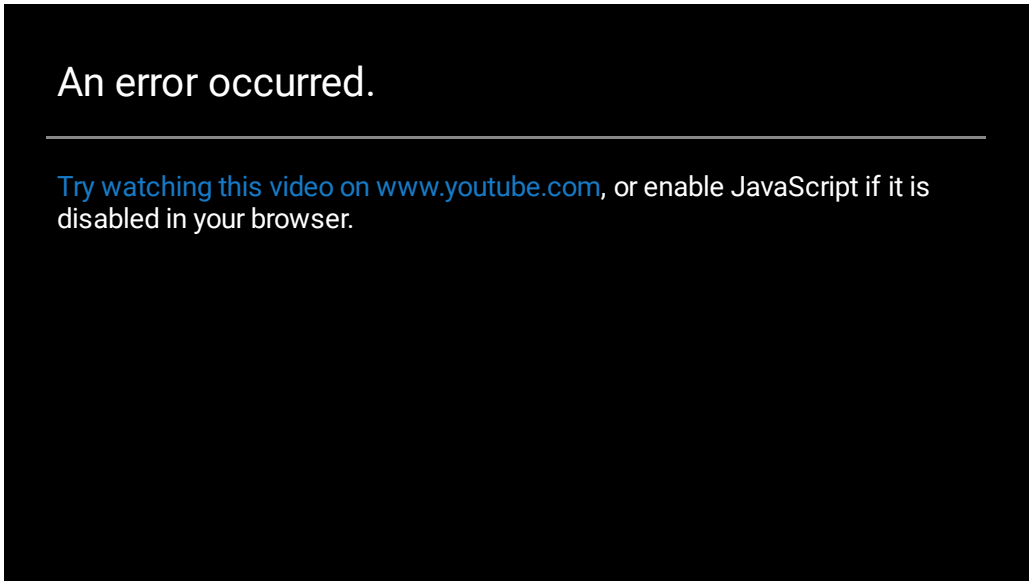
Comparing RDBMS and Hadoop

00:00 / 00:00

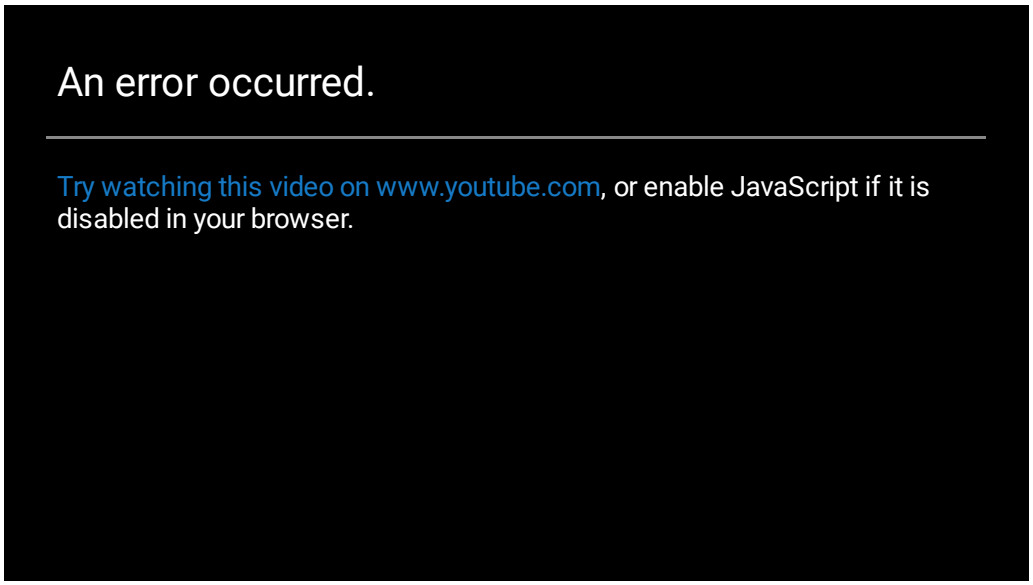
ZZEN9313_RDBMS_Transcript.pdf

More videos that can help you understand HDFS

1. Hadoop Tutorial - HDFS Commands



2. Hadoop Tutorial - HDFS Features



3. Hadoop Tutorial - Architecture