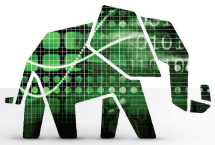
Hortonworks

Developing Solutions Using Apache Hadoop

A Course for Developers

Hortonworks UNIVERSITY

Course Introduction



Introductions

- Your name
- Job responsibilities
- Previous Hadoop experience (if any)
- What brought you here

Hortonworks

Course Outline

- Course Introduction
- Days 1 & 2
  - Lesson 1: Introducing Hadoop
  - Lesson 2: Working with HDFS
  - Lesson 3: Programming a MapReduce Job
  - Lesson 4: Implementing MapReduce
  - Lesson 5: Debugging MapReduce
- Days 3 & 4
  - Lesson 6: Pig
  - Lesson 7: Hive
  - Lesson 8: HCatalog
  - Lesson 9: HBase
  - Lesson 10: Hadoop Web Services

Hortonworks

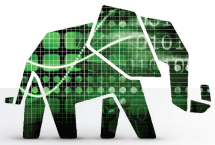
Class Logistics

- Schedule
  - 9 a.m. – 5 p.m.
- Restrooms
- Computers
  - Need external access to Amazon Cluster
  - You need some program to ssh into the cluster

Hortonworks

Hortonworks UNIVERSITY

Who is Hortonworks?



Who is Developing Apache Hadoop?

Hortonworks Largest PMC and Committer Base from any Single Organization

Hortonworks	Apache Hadoop
Co-Founder Hortonworks	Arun Murthy (VP Chair Apache Hadoop)
12 of 30 work at Hortonworks	Project Management Committee (Elected)
17 of 57 work at Hortonworks	Committers (Elected)
	Contributers (Volunteers)

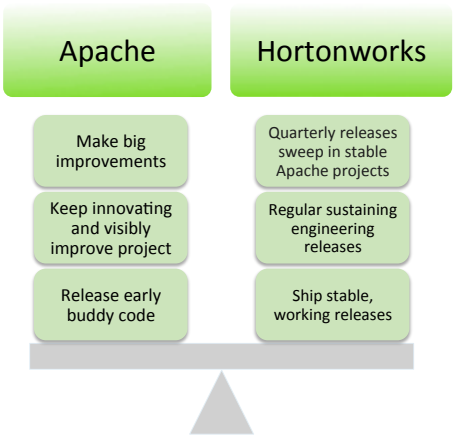
Apache Bylaws
http://hadoop.apache.org/who.html
As of 5/2012

Hortonworks

Balancing Innovation & Stability

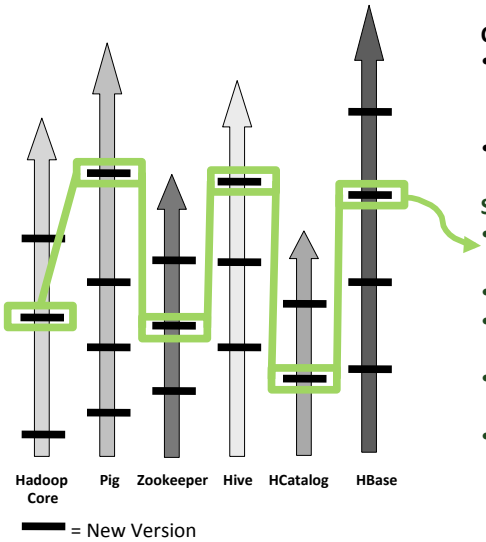
Be aggressive—Ship early and often

Be predictable—ship when stable



Hortonworks Data Platform

Fully Supported Integrated Platform



Challenge:

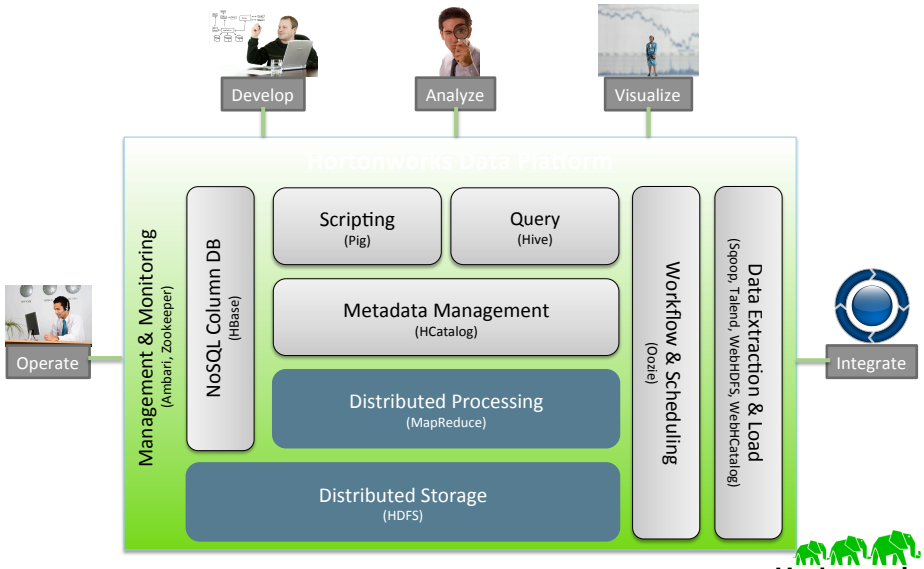
- Integrate, manage, and support changes across a wide range of open source Hadoop
- Time intensive, complex, expensive

Solution: Hortonworks Data Platform

- Integrated certified platform distributions
- Extensive Q/A process
- Industry-leading Support with clear service levels for updates and patches
- Multi-year Support and Maintenance Policy
- Technical guidance support for Universe and Multiverse components

Hortonworks

Hortonworks Data Platform



Hortonworks

Hortonworks UNIVERSITY

Lesson 1: Introducing Hadoop

Lesson Outline

Topic	Objective
The Problem Scope	Understand the issues leading to Big Data, and possible solutions
The Hadoop Approach	Describe the Hadoop approach for addressing the issues

Hortonworks

Hortonworks UNIVERSITY

Problem Scope

What is Big Data?

Hortonworks

Big Data Facts

1. In what timeframe do we now create the same amount of information that we created from the dawn of civilization until 2003?

   2 days

2. 90% of the world's data was created in the last (how many years)?

   2 years

3. What is 1024 petabytes also known as?

   An exabyte

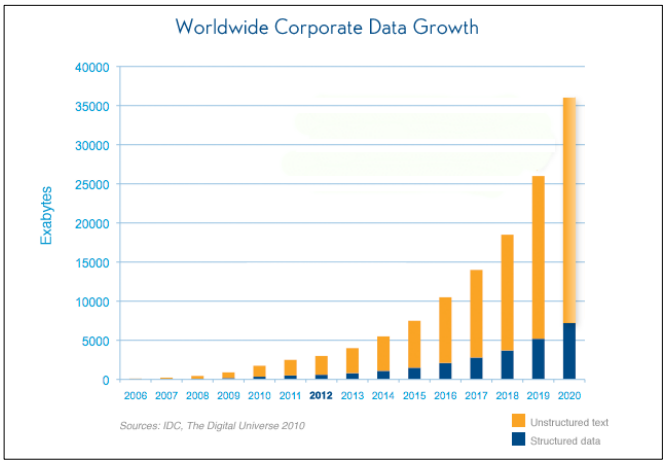
4. What is the anticipated shortage in the U.S. of skilled workers with deep analytical skills by 2018?

   140,000 to 190,000

Sources:
http://www.itbusinessedge.com/cm/blogs/lawson/just-the-stats-big-numbers-about-big-data/?cs=48051
http://techcrunch.com/2010/08/04/schmidt-data/

Hortonworks

What is Big Data?



Hortonworks

Drivers for Big Data

- Google and other web 2.0 companies, initially…

Hortonworks

Drivers for Big Data (Cont.)

- Now established companies are using the technology
- Much more than "count the clicks" that originally drove the solution

Hortonworks

Big Data Includes All Types of Data

Structured

- Pre-defined schema
- Highly
- Example: relational database system

Semi-structured

- Inconsistent structure
- Cannot be stored in rows and tables in a typical database
- Examples: logs, tweets, sensor feeds

Unstructured

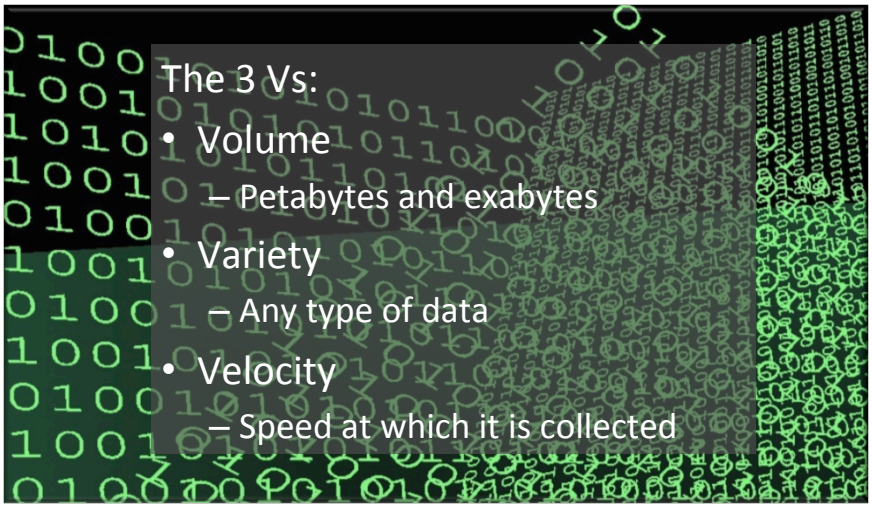
- Lacks structure or..
- Parts of it lack structure
- Examples: free-form text, reports, customer feedback forms

Time-based

Immutable

Hortonworks

Characteristics of Big Data

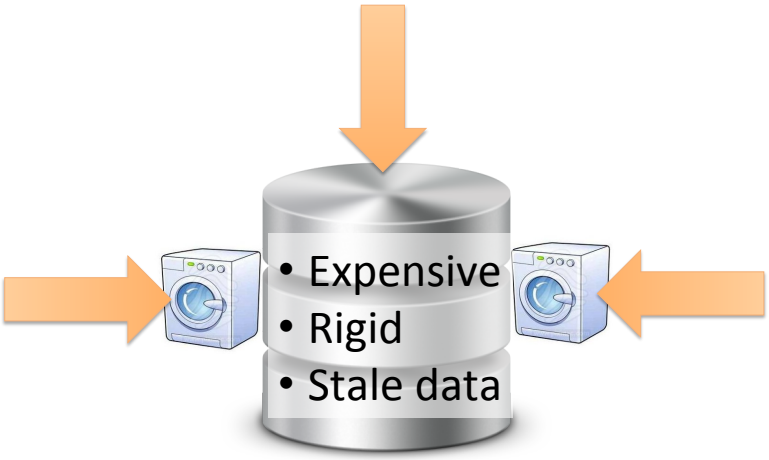


Hortonworks

The Legacy Solution

Hortonworks

The Legacy Solution



Hortonworks

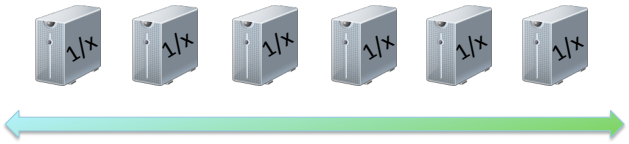
The Legacy Solution

Result:
Scale Up

Hortonworks

An Alternative Solution?

Scale out



Hortonworks

An Alternative Solution?

What issues exist between nodes?

Hortonworks

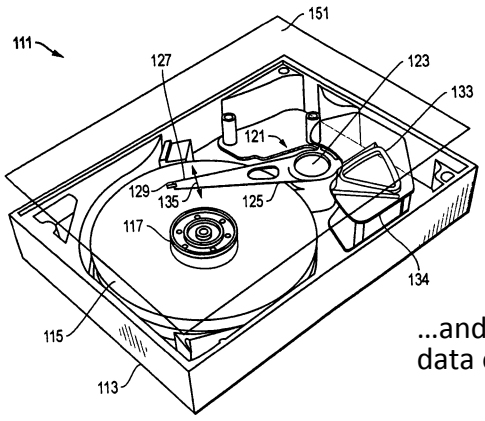
An Alternative Solution?

node x

What issues exist within each node?

Hortonworks

An Alternative Solution?



…and with actual storage of data on a hard drive?

Hortonworks

The Problem with Disk Seek

- 100 Billion records x 10 bytes = 1 TB
- You want to update 10% of the records – update 10 billion records
- Traditional approach – most classical systems:
  - Seek, read, write
  - The seek part with an average cost of 10ms is the slowest part (physically moving head of disk)
  - A seek for every record would take 2,700 hours
  - A scan read would take about 10,000 seconds (2.7 hours)
  - If you seek **more** the number goes up

Hortonworks

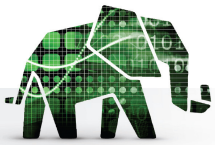
Original Assumptions

- Hardware can be reliable
- Machines have an identity
- Data sets must be centralized

Hortonworks

Hortonworks UNIVERSITY

The Hadoop Approach



A New Paradigm

- Process data locally
- Reduce dependence on bandwidth
- Expect failure
- Handle failover elegantly
- Duplicate finite blocks of data to small groups of nodes (rather than entire database)
- Reduce elapsed seek time

Hortonworks

A New Solution

= HDFS + Map/Reduce

- HDFS provides storage
- MapReduce provides analysis

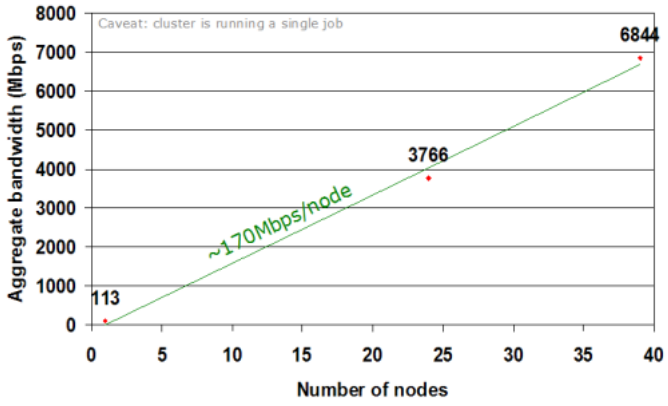
Hortonworks

The Hadoop Approach

- Distribute large amounts of data across thousands of commodity hardware nodes
  - Process data in parallel
  - Replicate data across cluster for reliability
- Analysis moved to data
  - Avoids data copy
- Scanning of data
  - Avoids random seeks
  - Easiest way to process

Hortonworks

Hadoop Scalability



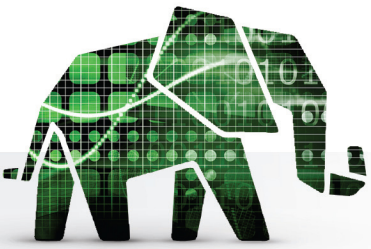
http://www.bitquill.net/blog/?tag=hadoop

Hortonworks

Lab 1.1: Login to Amazon EC2 Cluster

- Goal: Log in to Amazon server and verify that you have the correct directories
- Procedure:
  - Open a terminal window
  - Establish an ssh connection to the Amazon cluster and log in
  - Change to your student directory and verify that you have the correct directories
- Additional instructions:
  - All the labs will be done with this Amazon cluster
  - Record your EC2 instance in your lab guide
  - Logins and passwords are in lab guide

Hortonworks



Hortonworks UNIVERSITY

Lesson 2: Working with HDFS

Lesson Outline

Topic	Objective
Introducing Hadoop Distributed File System	Describe the key characteristics of HDFS
HDFS Architecture	Describe the basic design of the Hadoop Distributed architecture
HDFS Commands	Use HDFS file system commands
Programming HDFS	Use Java APIs to update HDFS
HDFS Monitoring and Permissions	Monitor HDFS, and explore various configurations and variables

Hortonworks

Hortonworks UNIVERSITY

Introducing HDFS

What is HDFS?

- Stands for Hadoop Distributed File System
- Primary storage system for Hadoop
- Fast and reliable
- Deployed only on Linux (as of May 2012)
  - Active work around Hadoop on Windows

HDFS Characteristics

- Persistent
- Replicated
- Linearly scalable
- Applications sequentially stream reads
  - Often from very large files
- Optimized for read performance
  - Avoids random disk seeks

HDFS Characteristics (Cont.)

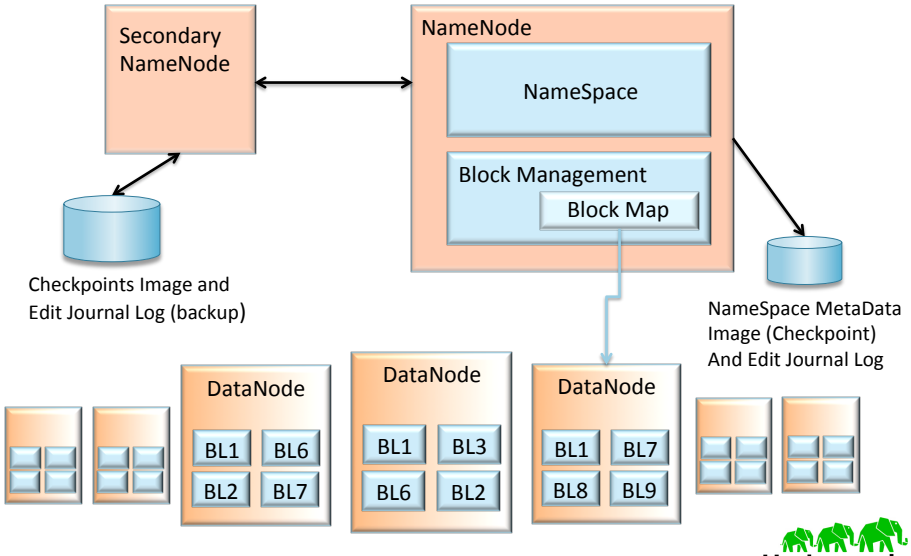
- Write once and read many times
- Files only append
- Data stored in blocks
  - Distributed over many nodes
  - Block sizes often range from 128MB to 1GB

Hortonworks

Hortonworks UNIVERSITY

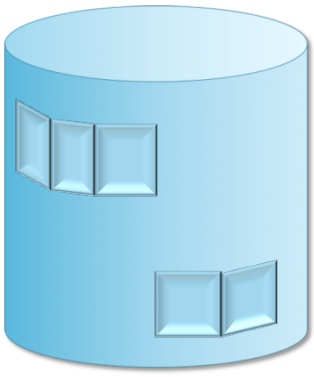
HDFS Architecture

HDFS Architecture



Hortonworks

Logical File System



- On Windows, Linux, and Mac:
- File's disk blocks are not physically contiguous
- They are logically contiguous

Hortonworks

Data Organization

- Metadata
  - organized into files and directories
  - linux-like permissions prevent accidental deletions
- Files
  - divided into uniform sized blocks
  - default 64 MB
  - distributed across clusters
- Rack-aware
- Keeps sizing checksums
  - for corruption detection

HDFS Cluster

- HDFS runs in Hadoop distributed mode
  - on a cluster
- 3 main components:
  - NameNode
    - Manages DataNodes
    - Keeps metadata for all nodes & blocks
  - DataNodes
    - Hold data blocks
    - Live on racks
  - Client
    - Talks directly to NameNode then DataNodes

NameNode

- Server running the *namenode* daemon
  - Responsible for coordinating datanodes
- The Master of the DataNodes
- Problems
  - Lots of overhead to being the Master
  - Should be special server for performance
  - Single point of failure

Secondary NameNode

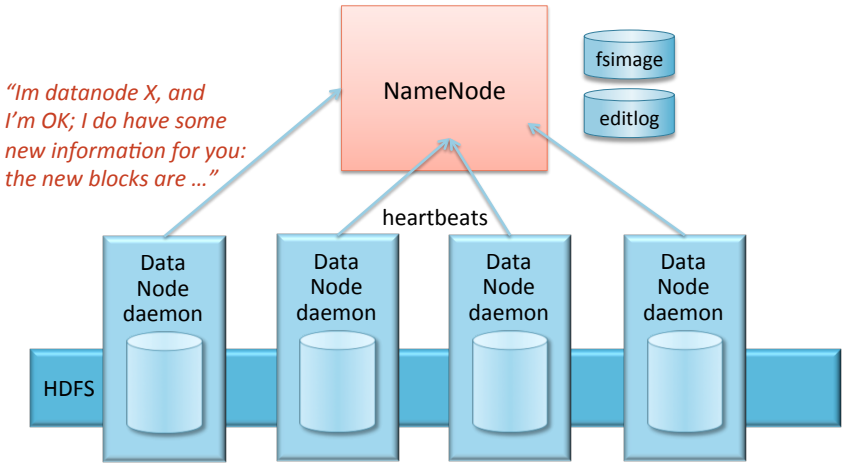
- Poorly Named!
- NOT auto failover
- Must be configured
- Merges and manages two files
  - Filesystem image (*fsimage*)
  - Edit log
- Sends resulting metadata back to NameNode
- Should run on a separate Node

DataNode

- A common node running a *datanode* daemon
- Slave
- Manages block reads/writes for HDFS
- Manages block replication
- Pings NameNode and gets instructions back
- If heartbeat fails
  - NameNode removes from cluster
  - Replicated blocks take over

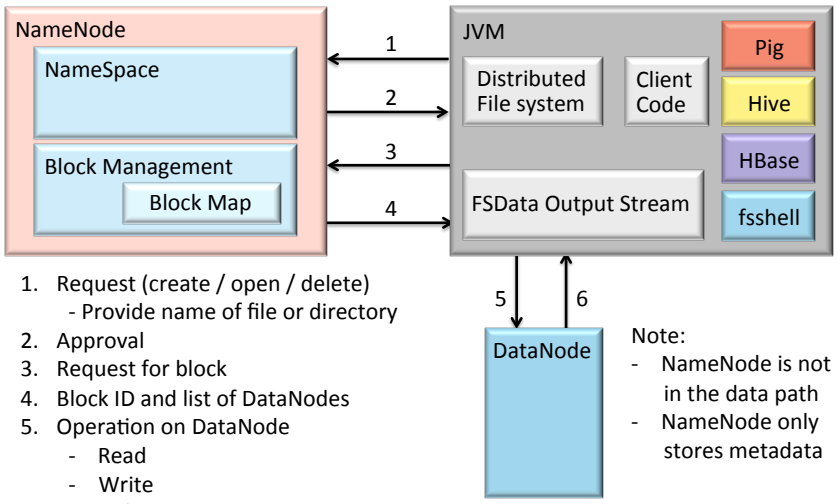
Hortonworks

HDFS Heartbeats



Hortonworks

File Access - RPC



1. Request (create / open / delete)
   - Provide name of file or directory
2. Approval
3. Request for block
4. Block ID and list of DataNodes
5. Operation on DataNode
   - Read
   - Write
   - Delete
6. Return

Note:
- NameNode is not in the data path
- NameNode only stores metadata

Hortonworks

Hortonworks UNIVERSITY

HDFS Commands

Basic HDFS File System Commands

```
$ hadoop fs –command <args>
```

Here are a few (of the almost 30) HDFS commands:

-cat: just like Unix cat – display file content (uncompressed)

-text: just like cat – but works on compressed files

-chgrp,-chmod,-chown: just like the Unix command, changes permissions

-put,-get,-copyFromLocal,-copyToLocal: copies files from the local file system to the HDFS and vice-versa. Two versions.

-ls, -lsr: just like Unix `ls`, list files/directories

-mv,-moveFromLocal,-moveToLocal: moves files

-stat: statistical info for any given file (block size, number of blocks, file type, etc.)

Hortonworks

Commands Example

```
$ hadoop fs –ls /user/brian/
$ hadoop fs -lsr
$ hadoop fs –mkdir notes
$ hadoop fs –put ~/training/commands.txt notes
$ hadoop fs –chmod 777 notes/commands.txt
$ hadoop fs –cat notes/commands.txt | more
$ hadoop fs –rm notes/*.txt
```


Hortonworks

Uploading Files into HDFS

```
$ hadoop fs –put filenameSrc filenameDest
$ hadoop fs –put filename dirName/fileName
```

```
$ hadoop fs –put foo bar
$ hadoop fs –put foo dirName/fileName
$ hadoop fs –lsr dirName
```


Hortonworks

Retrieving Files

```
$ hadoop fs –cat foo
$ hadoop fs –get foo LocalFoo
$ hadoop fs –rmr directory|file
```

Note: Another name for the –get command is -copyToLocal

Hortonworks

Lab 2.1: Use HDFS File System Commands

- Goal: Practice using fundamental HDFS file system commands

Hortonworks

Hortonworks UNIVERSITY

Programming HDFS

HDFS APIs

- APIs for access to HDFS
  - Abstract file system API
  - Easy access and storage on HDFS
  - Low-level details abstracted through API
- Package **`org.apache.hadoop.fs`**
  - Five interfaces
  - Over 20 classes, enums, exceptions such as:
    - FileSystem
    - Path
    - BlockLocation
    - FileChecksum

.fs Package (core types)

- **`Path`**
  - wraps a directory or a file
  - uses "/" as path separator
- **`FileSystem`**
  - either distributed Hadoop (HDFS) or Local Linux OS file system depending on which factory method used
- **`FSDataInputStream and FSDataOutputStream`**
  - File I/O
- **`FileStatus`**
  - directory or file metadata
  - group, owner, is a directory, etc.
- Others

class Path

- Represents an HDFS file or directory
- Used by `FileSystem` object at runtime
- 7 Constructors and over a dozen methods
- To create an HDFS directory:

```
FileSystem hdfs = FileSystem.get(new Configuration());
Path test = new Path(
              hdfs.getWorkingDirectory(), "/test");
hdfs.mkdirs(test); //default permissions
boolean wasDeteled = hdfs.deleteOnExit(test);
```


class FileSystem

- Represents the HDFS or the local file system
- No public constructor, dozens of methods
- Public static factory methods to get instance:
  - `LocalFileSystem getLocal(Configuration conf)`
  - `FileSystem get(Configuration conf)`
  - `FileSystem get(URI uri, Configuration conf)`
  - `FileSystem get(URI uri, Configuration conf, String user)`

```
FileSystem hdfs = FileSystem.get(new Configuration());
```

class FileStatus

- Represents metadata for HDFS file or directory:
  - **`public Path getPath()`**
    - Once you know **`Path`** you can read file
  - Other data: length, modification time, permissions, etc.
- Returned by `FileSystem` methods

```
public FileStatus[] getFilesByDir(Path hdfsDir,
FileSystem hdfs){

   FileStatus[] allFiles = hdfs.listStatus(hdfsDir);
   return allFiles;
}
```


Hortonworks

Using HDFS API

```
public void showHDFS(Path inPath, Path outPath) throws
IOException {

    Configuration config = new Configuration();

    //Static factory methods. For the HDFS use:
    FileSystem hdfs = FileSystem.get(config);

    //For the local Linux file system use:
    LocalFileSystem local = FileSystem.getLocal(config);

    //Read data from local using input stream:
    FSDataInputStream inStream = local.open(inPath);
    FSDataOutputStream outStream = hdfs.create(outPath);
```


Hortonworks

HDFS API

```
    //We need to hold the file data in memory:
    byte[] fromFile = new byte[1000];


    //read the input file:
    int data = 0;
    while((data = inStream.read(fromFile)) > 0){
        System.out.println("\nReading input file: " +
                           inPath);
        //Write the same data to HDFS:
        outStream.write(fromFile);
    }

    inStream.close();
    outStream.close();
}
```


Hortonworks

Hadoop Configuration Files

- Earlier versions of Hadoop had just one configuration file to modify: **`hadoop-site.xml`**
- After v0.20 there are three:
  - **`core-site.xml`** for core settings
    - Ex: hostname for the NameNode daemon
  - **`hdfs-site.xml`** for HDFS issues
    - Ex: how often a data block replicates to DataNodes
  - **`mapred-site.xml`** for MapReduce issues
    - Ex: hostname for the `JobTracker` daemon
- For the environment modify: **`hadoop-env.sh`**
  - Ex: JAVA_HOME

Hortonworks

Lab 2.2: Program HDFS

- Goal: Use Java to accomplish HDFS tasks
- Procedure:
  - Read a file from local Linux filesystem
  - Write a file to the HDFS

Hortonworks

Hortonworks UNIVERSITY

HDFS Monitoring and Permissions

HDFS Web Interface

- NameNode can be accessed from port 50070

  `http://namenode:50070`

  - Provides basic monitoring and file browsing operations
- DataNode can be accessed by using port 50075

  `http://datanode:50075`

HDFS Log Files

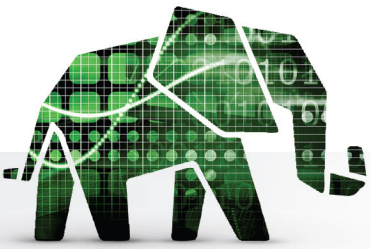
- Hadoop in distributed modes uses log files
- The following daemons create logs:
  - namenode
  - secondarynamenode
  - datanode
  - jobtracker
  - tasktracker

HDFS File Permissions

- Simple permission system based on POSIX model
- Does not provide strong security for HDFS
- Designed to prevent accidental corruption of data or casual misuse of information
- Users and Groups defined
- Each file or directory has 3 permissions
  - Read, write, execute
- Superuser supersedes all permission settings

Lab 2.3: Monitor HDFS

- Goal: Practice using basic monitoring tasks in HDFS, and explore various configurations and variables
- Procedure:
  - Explore NameNode and DataNode configuration
  - Explore JobTracker and TaskTracker configuration
  - Use the HDP web-based GUI to browse web UIs
  - Examine Hadoop primary daemons and environmental variables
  - Start and stop Hadoop services



Hortonworks UNIVERSITY

Lesson 3: Programming a MapReduce Job

Lesson Outline

Topic	Objective
How MapReduce Works	Describe the problems pertinent to Hadoop
The Original Use Case	Describe the Hadoop approach for addressing these problems
MapReduce Jobs	The lifecycle of a MapReduce job

Hortonworks

Hortonworks UNIVERSITY

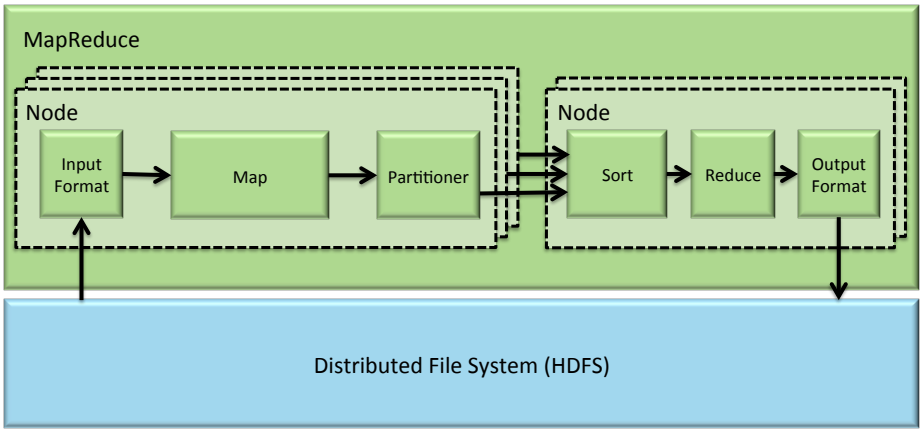
How MapReduce Works

What is MapReduce?

- A framework
- Big Data analytics and processing
  - Large datasets
- Node-local computation
- Parallel processes
- Handles node fail-over
- Java
  - Other languages supported

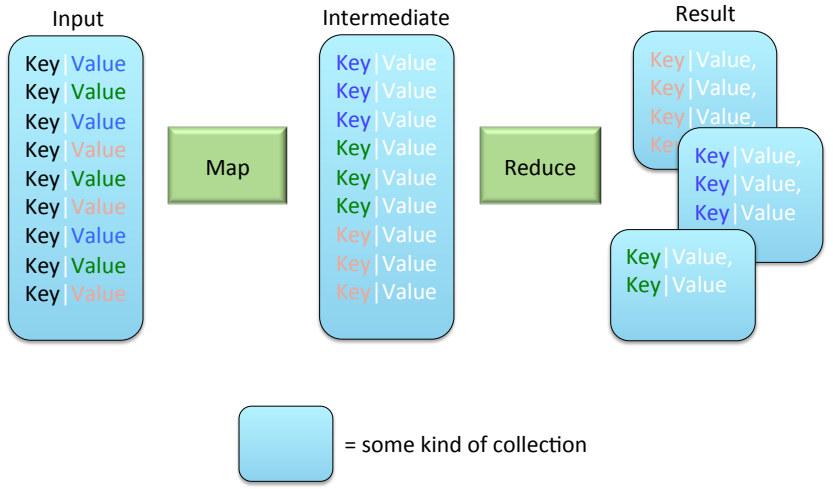
Hortonworks

Basic MapReduce Architecture



Hortonworks

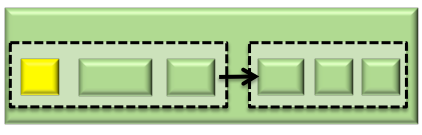
Simple MapReduce



Hortonworks

InputFormat

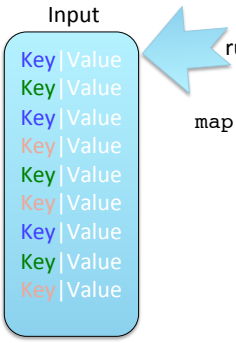
- Determines how the data is split up
- Creates `InputSplit[]` arrays
  - Each is individual map
  - Associated with a list of destination nodes
- `RecordReader`
  - Makes key,value pairs
  - Converts data types



Hortonworks

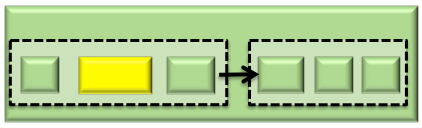
Mapping

- Operates on every data set individually
- Often used for ETL



runs algorithm on each data row sequentially

```
map(K1 key, V1 val, OutputCollector<K2, V2> output,
       Reporter reporter)
```

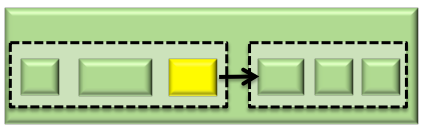


Hortonworks

Partitioner

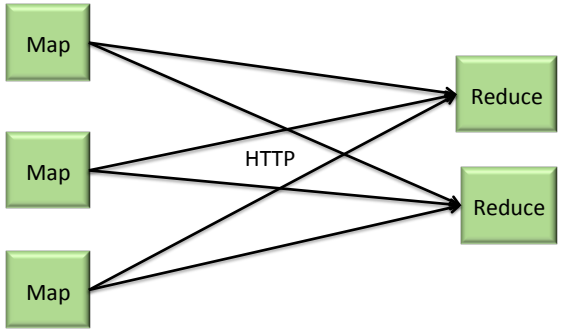
- Distributes key,value pairs
- Decides the target Reducer
  - Uses the key to determine
  - Uses Hash function by default
  - Can be custom

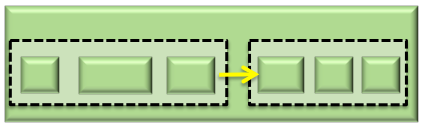
```
getPartition(K2 key, V2 value, int numPartitions)
```



Hortonworks

The Shuffle

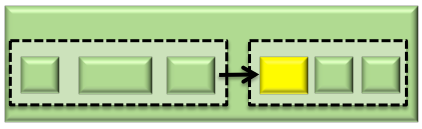




Hortonworks

Sort

- Guarantees sorted inputs to Reducers
- Final step of Shuffle
- Helps to merge Reducer inputs

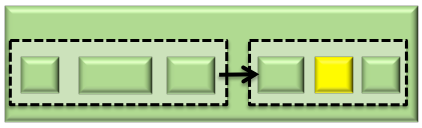


Hortonworks

Reduce

- Receives output from many Mappers
- Consolidates values for common intermediate keys
- Groups values by key

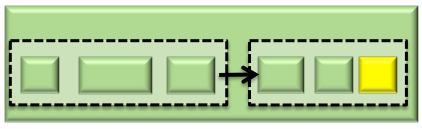
```
reduce(K2 key, Iterator<V2> values,
              OutputCollector<K3,V3> output,
                    Reporter reporter)
```



Hortonworks

OutputFormat

- Validator
  - For output specs
- Sets up a `RecordWriter`
  - Which writes out to HDFS
  - Organizes output into `part-0000x` files



Hortonworks

Hadoop Properties

- Hundreds of properties
  - Some are related to HDFS
    - `dfs.block.size=134217728` for 128MB blocks
  - Some are related to MapReduce
    - `mapred.compress.map.output=true` for compressed map() output
  - Some are related to the system as a whole
    - `fs.default.name=hdfs://localhost:8020` identifies NameNode
- Many are undocumented and found in the source tree
- Your MapReduce jobs use some properties
- Create your own properties
  - `String propertyName, String propertyValue`

Hortonworks

Hortonworks UNIVERSITY

The Original Use Case

In The Beginning

It all started when Google needed a way to:

- Determine which web sites to provide for searches
- Do page ranking

Hortonworks

A Basic MapReduce Job

Nested Class with Generics Structure

```
public class WordCount {

    public static class Map extends MapReduceBase implements Mapper<> {
       public void map(LongWritable key, Text value, OutputCollector<…>
output, Reporter reporter) throws IOException {}
    }

    public static class Reduce extends MapReduceBase implements Reducer<>{
       public void reduce(Text key, Iterator<…> values, OutputCollector<>
output, Reporter reporter) throws IOException {}
    }

   public static void main(String[] args) throws Exception {}

(continued)
```


Hortonworks

A Basic MapReduce Job

map() Method

```
public void map(LongWritable key,
    Text value,
    OutputCollector<Text, IntWritable> output,
    Reporter reporter)
    throws IOException {}
```


Hortonworks

A Basic MapReduce Job

map() implemented

```
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> output, Reporter reporter)
throws IOException {
          String line = value.toString();
          StringTokenizer tokenizer = new
StringTokenizer(line);
          while (tokenizer.hasMoreTokens()) {
              word.set(tokenizer.nextToken());
              output.collect(word, one);
          }
}
```


Hortonworks

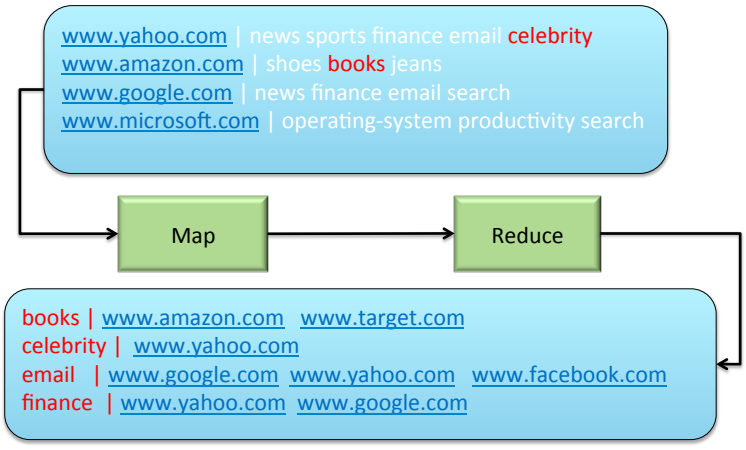
A Basic MapReduce Job

reduce() implemented

```
private final IntWritable totalCount = new IntWritable();

public void reduce(Text key,
    Iterator<IntWritable> values, OutputCollector<Text,
    IntWritable> output, Reporter reporter) throws
IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        totalCount.set(sum);
        output.collect(key, totalCount);
}
```


Hortonworks

Another Use Case – Inverted Index



Hortonworks

Lab 3.1: Run an Existing MapReduce Job

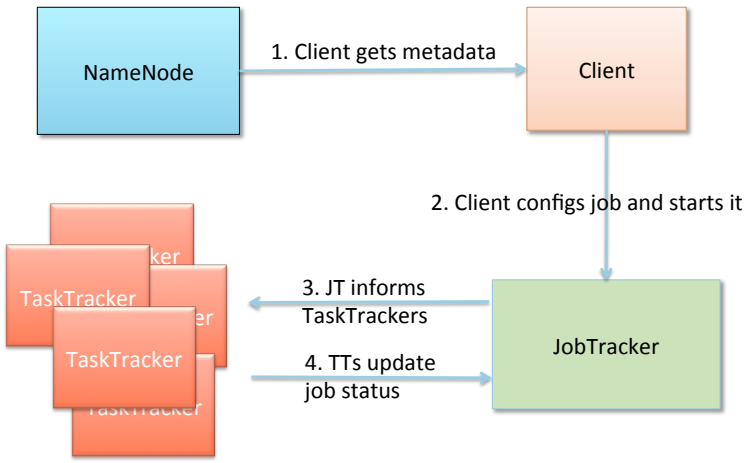
- Goal: Run a simple MapReduce job
- Procedure:
  - Use Hadoop built-in map() and Hadoop built-in reduce() code
  - Review the code for Job setup and configuration
    - Configuration() class, JobConf() class, and JobClient() class
  - Run the job

Hortonworks

Hortonworks UNIVERSITY

MapReduce Jobs

The MapReduce Lifecycle



Hortonworks

JobTracker

- A "master" daemon of MapReduce
  - along with NameNodes
- Provides resource management
  - Manages TaskTrackers at each node
  - Schedules tasks to available resources
- Manages Job lifecycle
  - Job submission
  - Task execution
  - Task error recovery
  - Run job's tasks to completion

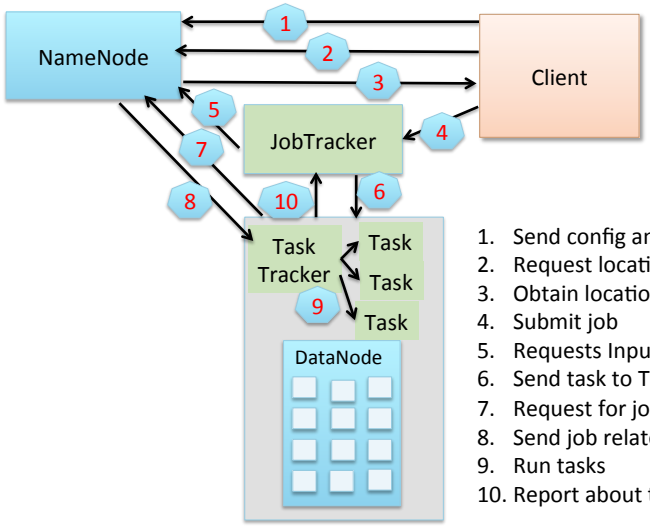
Hortonworks

TaskTracker

- Node-level daemon
- Manages tasks at the node
- Talks with `JobTracker`
  - Gets new tasks
  - Provides updates
- Fetches executables from HDFS (typically)
- Has *slots*
  - One task per slot

Hortonworks

Job Submission and Running



1. Send config and jars
2. Request location of data
3. Obtain location of data
4. Submit job
5. Requests InputSplits
6. Send task to TaskTracker
7. Request for job related info
8. Send job related info
9. Run tasks
10. Report about tasks

Hortonworks

Hadoop Job Submission

```
$ hadoop jar myMR.jar com.mycomp.hadoop.MyJob
inputFile outputDirectory
```

```
-Dproperty=value
-conf  path & file
-fs some uri
-files
```


Hortonworks

Invocation Options

```
$ hadoop jar myMR.jar MyJob arg1 arg2
```

```
// config job in main
public static void main(String[] args)
```

OR

```
// call run() from main()
public static void main(String[] args)
    ToolRunner.run (MyClass, args)
```


Hortonworks

ToolRunner

- Handles command line arguments
- Job config is in `run() method`
- `main()` now calls ToolRunner
  - Invokes your run() method
  - Needs to know the class holding your run method, and the command line arguments
  - Passes parameters to run() via a Configuration object

Tool Interface

- When Tool is implemented you also get `GenericOptionsParser`
  - a class that interprets Hadoop command-line options
    - Sets them in a Configuration object
- `run()` method gets Configuration using `getConf()`

```
public interface Tool extends Configurable {
      int run(String [] args) throws Exception;
}
```

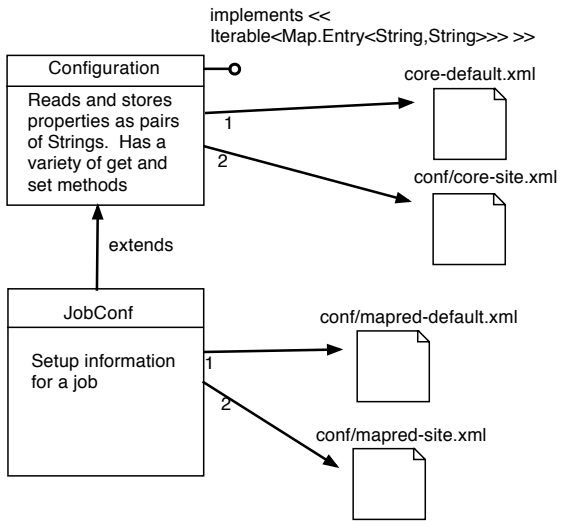
Job Configuration

`JobConf:`

- Defines Hadoop job properties
  - Reads a Configuration() instance to get some of those properties
- Assigns job name
- Specifies key/value data types for map & reduce
- Assigns map & reduce runtime classes
- Specifies HDFS input and output directories

Hortonworks

JobConf() Properties



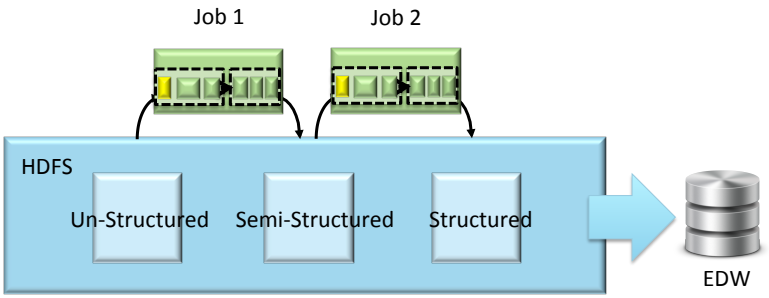
Hortonworks

Some JobConf() Methods

```
job.setJobName("MyJob");
job.setMapperClass(Mapper.class);
job.setReducerClass(Reducer.class);
job.setOutputKeyClass(Text.class);  // to HDFS from reduce()
job.setOutputValueClass(Text.class);  // to HDFS from reduce()
job.setCombinerClass(Reducer.class);
job.setJarByClass(PageStat.class);
job.setMapOutputKeyClass(Text.class);   // output from map()
job.setMapOutputValueClass(IntWritable.class); // from map()
job.setNumReduceTasks(1);
job.set(String name, String value);  //arbitrary property
String jobName = job.get(String name);
```


Hortonworks

Chaining Jobs



Many problems can be solved this way

Hortonworks

Fault Tolerance

- TaskTracker sends heartbeats to the JobTracker
- If the TaskTracker fails during Mapping
  - Other TaskTrackers will be assigned
  - Re-execute all tasks
- If the TaskTracker fails during Reduce
  - Other TaskTrackers will be assigned
  - Re-execute only incomplete tasks

Output Files

- Number of reducers is configurable
  - # output files = # reducers
  - One-one reducer to output file
  - So, ten reducers implies ten output files
  - Code you can specify the number of reducers
- A map-only job
  - Specify zero reducers
  - # output files = # mappers = # input blocks
  - Override InputFormat.getSplits() to control the number of map tasks

Killing A Job

- If you see a runaway job, use the "job –kill" command to kill it:

```
$ hadoop job -list
1 jobs currently running
JobId   State   StartTime       UserName
job_201204111901_001   1       1218506470390   horton

$ hadoop job -kill job_201204111901_001
```


Hortonworks

Compiling A Job

```
$ javac -cp /usr/lib/hadoop/hadoop-core.jar  mymr.java
```

```
core-default.xml
hdfs-default.xml
mapred-default.xml

org/apache/hadoop/*
```

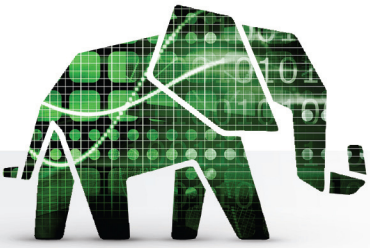
← Default parameters you can jar -xvf <filename>

Hortonworks

Lab 3.2: Write a Simple MapReduce Program

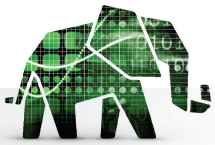
- Goal: Write your first MapReduce program to count the occurrences of distinct words in a file
- Procedure:
  - Create an input file and copy it to HDFS
  - Create a Java file that adds the Java and Hadoop imports, the MapReduce class and subclasses, map and reduce methods, and specify other aspects of the job
  - Compile and run the program

Hortonworks



Hortonworks UNIVERSITY

Lesson 4: Implementing MapReduce

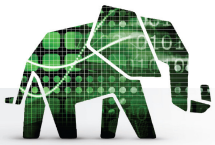


Lesson Outline

Topic	Objective
Hadoop Data Types	Work with Hadoop built-in and custom data types
InputFormats	Work with common MapReduce input formats, input splits, and the RecordReader
OutputFormats	Work with common MapReduce output format
Combiner	Implement a MapReduce combiner to reduce the amount of data in the shuffle
Partitioner	Implement a custom partitioner to define the number of reducers and specify reducers
Streaming	Implement streaming as an alternative to writing Java MapReduce
Distributed Cache	Efficiently distributing shared files

Hortonworks UNIVERSITY

Hadoop Data Types

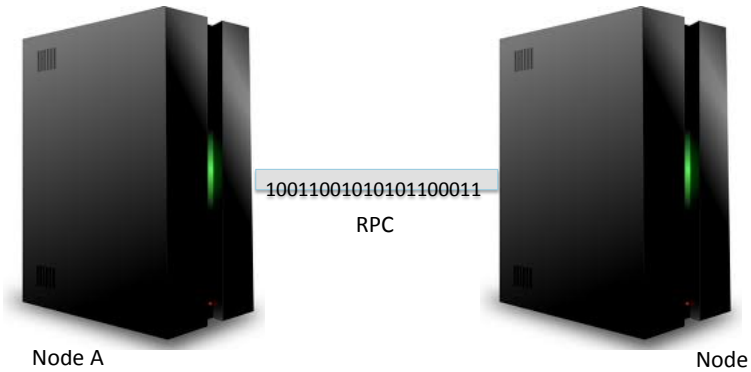


Hadoop Built-in Classes

- Simple implementations of map() and reduce()
  - simple – e.g. reverse K,V in mapper
- map() "pass through"
  - copies K,V input as K,V output
  - still groups by key
  - sends the grouped data to a reducer
- reduce() "pass through"
  - pulls out each V from the Iterator and creates a KV pair
  - KV pairs are sent as output to HDFS

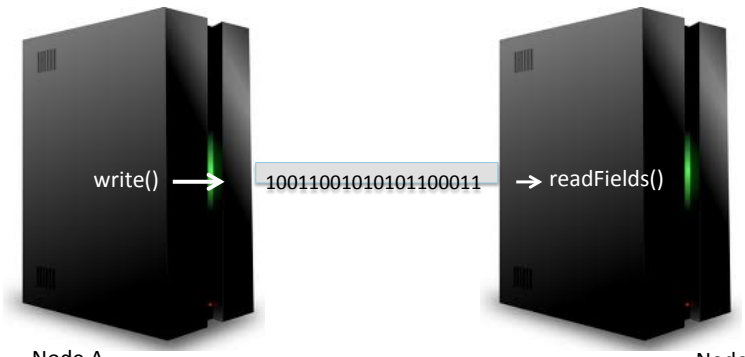
Hortonworks

Serialization in Hadoop



Hortonworks

Serialization Solution



Solution: `Writable` Interface

Hortonworks

A Writable Datatype

```
public class MyWritable implements Writable {
    private int myInt;
    private long myLong;

    public MyWritable() { set(0,0L); }
    public void set(int i, long l) { myInt=i; myLong=l };

    public void write(DataOutput out) throws IOException {
        out.writeInt(myInt);
        out.writeLong(myLong);
    }

    public void readFields(DataInput in) throws IOException {
        this.myInt = in.readInt();
        this.myLong = in.readLong();
    }
```


Hortonworks

Writable

- Methods
  - `write(DataOutput out)`
  - `readFields(DataInput in)`
- `DataInput` and `DataOutput` are streaming datatypes
  - methods that can be used on those streams
    - e.g. `readInt()` , `writeInt()`, `readLong()`, `writeLong()`
  - I/O on those streams is Hadoop-specific

Hortonworks

WritableComparable

- Interface for objects that are both Writable and Comparable
  - Same methods as `Writable`
  - plus the `compareTo` method
- Make your keys implement this interface

RawComparator

- Compares keys by bytes during the sorting phase of MapReduce
- Avoids the overhead of keys being deserialized
- Improves performance greatly!
- Write your own class that implements `RawComparator` (or extends `WritableComparator`)

WritableComparator

- Utility class to simplify implementing `RawComparator`

```
public class MyComparator extends WritableComparator {
    protected MyComparator() {
        super(MyKey.class);
    }

    @Override
    public int compare(byte[] b1, int s1, int l1, byte[] b2,
                       int s2, int l2) {

        //Return 0 if b1 and b2 are equal
        //Return a negative value if b1 < b2
        //Return a positive value if b1 > b2
    }
}
```

- `conf.setOutputKeyComparatorClass(MyComparator.class);`

Basic Writable Datatypes

- Generally support `.get()`, `.set()`, and `.toString()`
  - `ByteWritable`
  - `IntWritable`
  - `LongWritable`
  - `FloatWritable`
  - `DoubleWritable`
  - `BooleanWritable`
  - `NullWritable` (no support for `.set()`)
  - `IntPairWritable`
  - `Text`

Writable Collections

- `ArrayWritable` for a Java Array
  - `myAW.set(Writable[])`
  - `myAW.get() returns the Writable[]`
  - `myAW.toArray() returns a Java Object`
- `MapWritable` for a Java Hash Map
  - `myMapW.put(K,V)`
  - `myMapW.get(K)`
- `BytesWritable` for a Java array of bytes
  - `myBW.set(byte[], offset, len)`
  - `myBW.get()  returns byte[]`

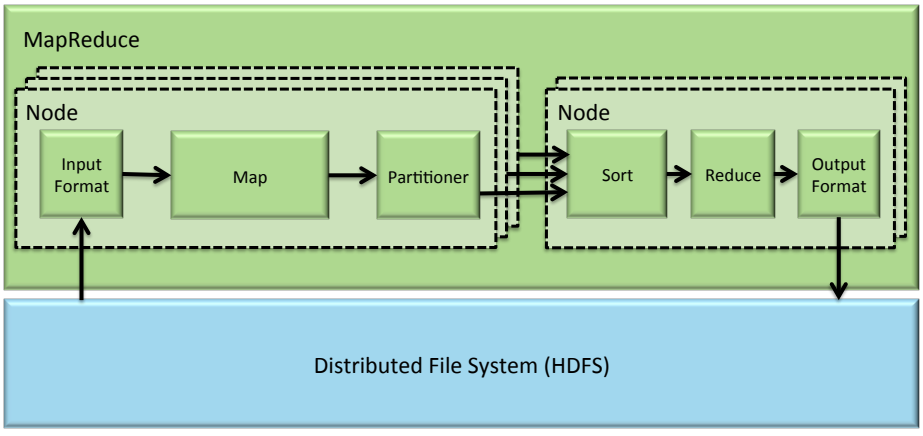
Other Serialization Options

- Apache Thrift
- Avro
- Google Protocol Buffers

Lab 4.1: Create a Custom Writable Type

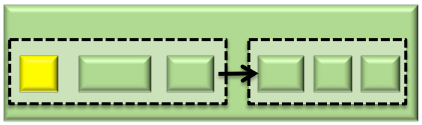
- Goal: Create a new Hadoop datatype that can be passed as a value between your map() and reduce() code. This datatype can be serialized to disk or transmitted across the network.

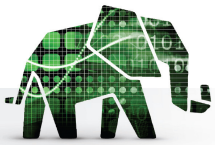
MapReduce Architecture



Hortonworks UNIVERSITY

InputFormat





InputFormat

- Identifies input type
- Can be anything
  - Text
  - Numbers
  - Complex types
  - DB rows
- Defines the way input files are chopped up
  - `getSplits()`
- Individual records come from splits
  - `getRecordReader()`
    - Parses input split into a set of key/value pairs

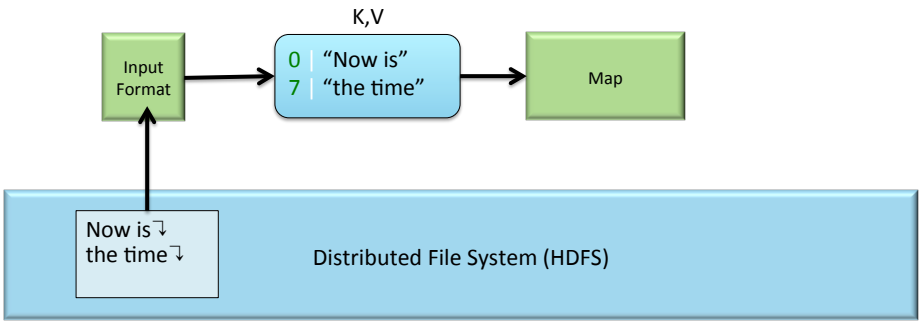
Hortonworks

Common Input Formats

- `TextInputFormat` (Default)
  - Processes one line of text at a time
  - Key is LongWritable, Value is Text
- `KeyValueInputFormat`
  - One line of text at a time
  - Assumes a separator between Key and Value
  - Key & Value are Text
  - Separator may be specified in a parameter
- `SequenceFileInputFormat`
  - Hadoop specific binary format for storing keys and values
  - Supports compression
  - Is splitable (some compression types are not splitable into blocks)
- `DBInputFormat`

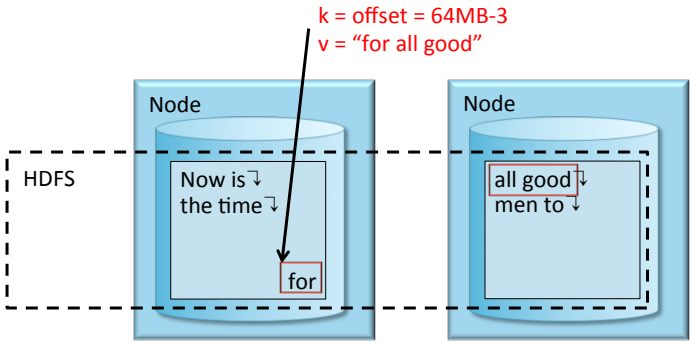
Hortonworks

TextInputFormat



Hortonworks

TextInputFormat



Hortonworks

FileInputFormat

- For large files
  - Larger than an HDFS block
- Split size
  - Controlled by Hadoop properties
  - Application can override
- Hadoop is optimized with a small number of large files

Hortonworks

InputSplits and HDFS

- Split is a portion of the data to be processed by a single map process in parallel
- Split = single block of data
  - records may straddle blocks
  - tasks often need to access a small portion of a record from a block on another node
- Use larger blocks for a split to:
  - Increase performance due to more data for a mapper
  - A mapper has setup/teardown time that is constant
  - A small split means less work for the constant setup time
- `JobClient` computes input splits

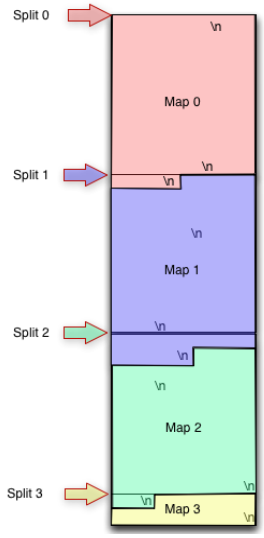
Hortonworks

RecordReader

```
public interface RecordReader<K, V> {

  /** Read the next key/value pair from the input for
processing.  */
  boolean next(K key, V value) throws IOException;

  // Create an object to be used as a key
  K createKey();

  // Create an object to be used as a value
  V createValue();

  // Returns the current position in the input
  long getPos() throws IOException;

  public void close() throws IOException;
  float getProgress() throws IOException;
}
```


Hortonworks

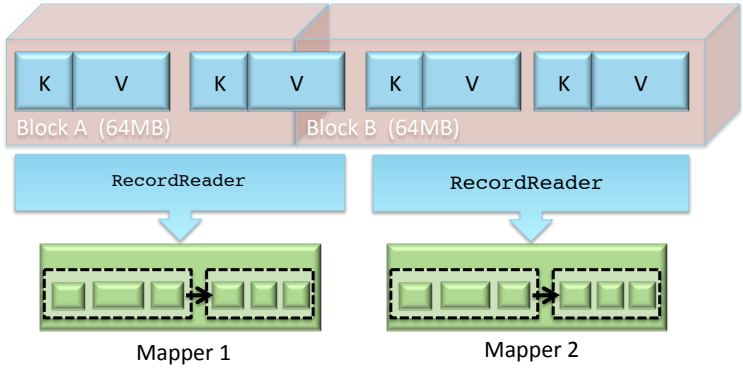
InputSplit



- Usually crosses boundaries
  - Of Mappers
  - Of HDFS blocks

Hortonworks

RecordReader –split vs. record



Hortonworks

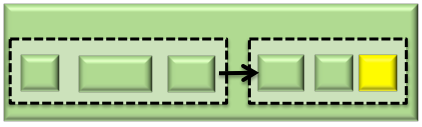
Binary Input

- `SequenceFileInputformat`
  - Stores binary keys & values
  - Split-able
  - Compressable
  - Many types
- Variants
  - `SequenceFileAsTextInputFormat`
  - `SequenceFileAsBinaryInputFormat`

Hortonworks

Hortonworks UNIVERSITY

OutputFormat



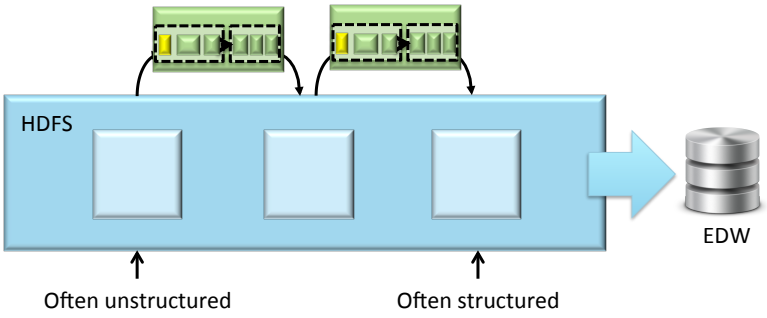
OutputFormat

- Output files are all placed in the same directory for a job
- By default they are named `part-` with a five digit number
  - Numbering starts with part-00000
  - The number corresponds to the reducer number

```
$ hadoop fs -lsr
drwxrwxrwx   - train hdfs          0 2012-05-30
17:02 /user/train/tout
-rw-------   1 train hdfs       2222 2012-05-30
17:02 /user/train/tout/part-00000
```


Hortonworks

Output Format and Input Format



Hortonworks

Lab 4.2: Create a Custom `InputFormat`

- Goal: Write an `InputFormat` that can read input files that are not lines of strings

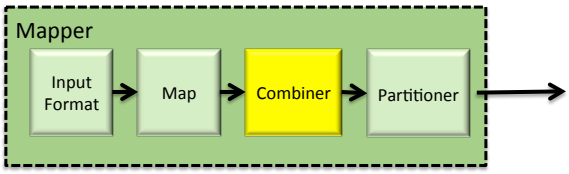
Hortonworks

Hortonworks UNIVERSITY

Combiner

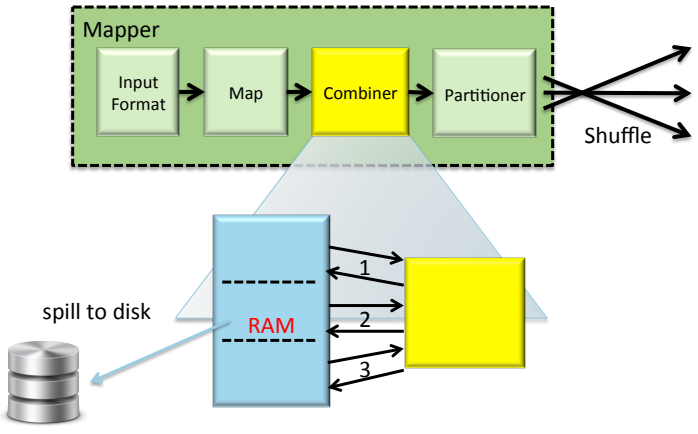
Combiner Goals

- Optional
- A map-side "mini-reducer"
  - Operates only on one map's data
- Reduces the amount of data to be shuffled
  - Across the wire
  - Should reduce the number of records



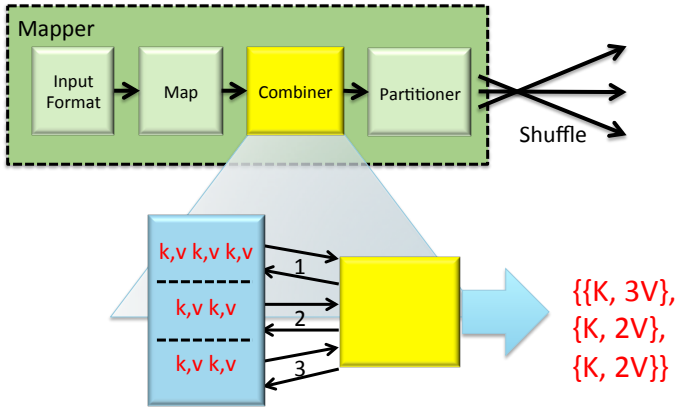
Hortonworks

Map-Side Reducer



Hortonworks

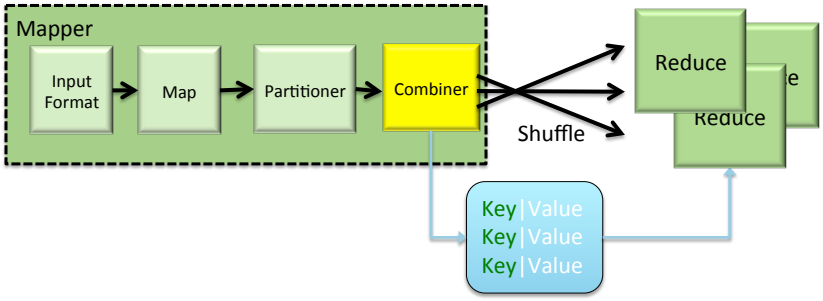
Map-Side Reducer



Hortonworks

Combiner Contract

- Implements the Reducer interface
- Reducer's input pairs must be the same type as Map output
- Can run 0 to multiple iterations
- Cannot assume that incoming list is fully sorted data
- Cannot change the values of keys



Hortonworks

Combiner in Action

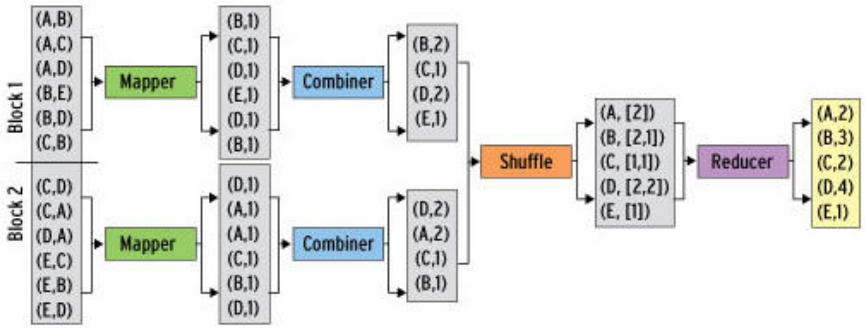


Image: HPC Admin Magazine 2012

Hortonworks

Combiner Cautions

- Should show repeatable results
  - No matter how many times it's run
- Combiners are slow
  - Compare `MAP_OUTPUT_RECORDS` without the Combiner to `COMBINER_OUTPUT_RECORDS` with the Combiner.
    - `MAP_OUTPUT_RECORDS` should be far greater than the `COMBINER_OUTPUT_RECORDS`
- Make sure the combiner does not cause any needed data to be dropped.

Hortonworks

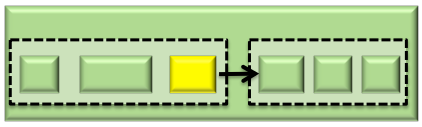
Lab 4.3: Use Combiner with MapReduce

- Goal: Use the combiner to enhance performance of the job

Hortonworks

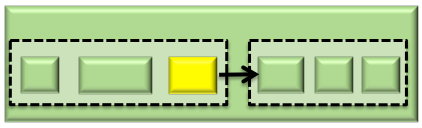
Hortonworks UNIVERSITY

Partitioner



Partitioner Contract

- Allocate data across the number of partitions that you specify
- Key hash value determines the reducers
  - `HashPartitioner.getPartition()`
- Customize to ensure specific reducer target



Hortonworks

Default Partitioner

- Key's `hashcode()` is used
  - Converted to a positive value
  - Returns a number from 0 to (# of Reducers minus 1)
- Works for most keys
- May unevenly distribute
  - If keys hash to similar values
  - Customize to load-balance

Hortonworks

HashPartitioner Class

```
package org.apache.hadoop.mapred.lib;

//   Partition keys by their {@link Object#hashCode()}
public class HashPartitioner<K2, V2> implements Partitioner<K2, V2>
{
   public void configure(JobConf job) {  }

   public int getPartition(K2 key, V2 value,  int numReduceTasks) {
    return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
  }
}
```


Hortonworks

Partitioner Example

Key	Assigned Reducer
A	0
B	1
C	2
D	3
E	4
F	5
G	6
H	7
I	8
J	9
K	10
…	

- The developer needs to know the key space.
- Sampling the data can help to create that understanding.

This could be divided up separately. For example 'S' might be broken up to go to several reducers.

Sa to Sh	20
Si to Sm	21
Sn to Sz	22

Hortonworks

Lab 4.4: Create a Custom Partitioner

- Goal: Write a MapReduce Java program that uses a custom partitioner
- Procedure:
  - Create AlphabeticalPartitioner.class
  - Configure Hadoop to use your custom partitioner and to have 26 reducers
  - Compile and run the code

Hortonworks

Hortonworks UNIVERSITY

Streaming

Streaming

- Create & run MapReduce jobs with any executable or script
- Typically a scripting language is used
  - Supports Unix commands as mapper or reducer
    - awk, sed, grep
  - Supports scripting languages such as:
    - Python
    - Perl
  - Supports other languages such as:
    - C#

Hortonworks

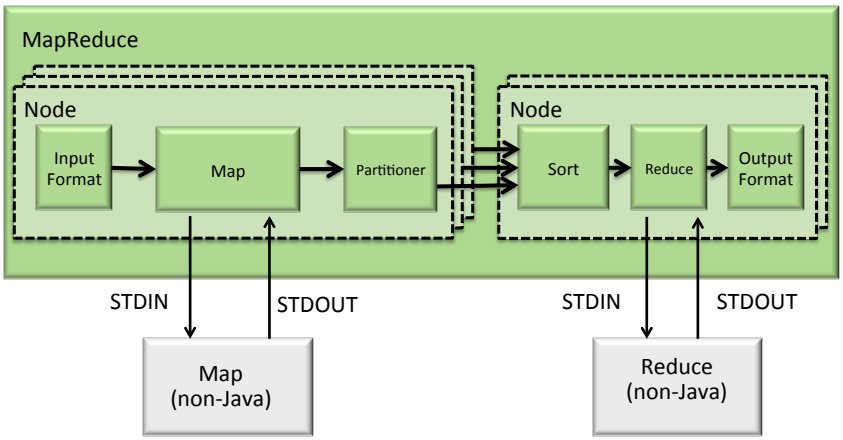
Uses of Streaming

Useful for:

- Short MapReduce programs
  - more easily written in other languages
- Quick answers
  - small quickly created data analysis needs
- Data is text based
  - Each line read as a record
- Integration with legacy languages

Hortonworks

Streaming



Hortonworks

Streaming Syntax

- Invoke the streaming jar in the contrib directory

```
$ hadoop jar contrib/streaming/hadoop-x.x-
streaming.jar
    -input inputfile
    -output outputfile
    -mapper myScript.py
    -file myScript.py
    -reducer /bin/somecmd
```

- Using built-in Unix commands to process data
  - Examples: wc, cat

Hortonworks

Streaming & Key Value Pairs

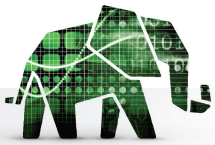
- Streaming can operate on key/value pairs
- Uses the tab character as a separator
  - Default to separate the key from the value in a record
  - First value is considered the key
  - Remainder (minus the tab) is the value
  - Without a delimiter the the entire record is read in as a key

Lab 4.5: Demo MapReduce Streaming using Python

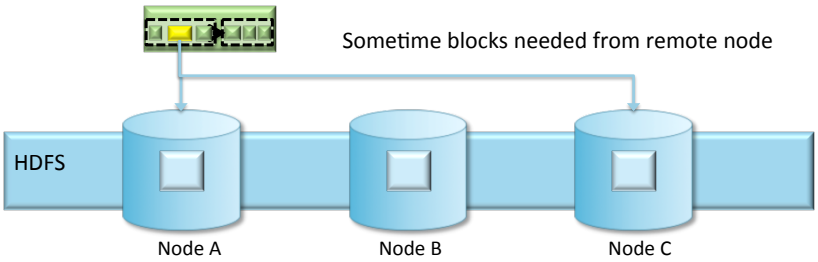
- Goal: Demo MapReduce using other languages such as Python language instead of Java
- Procedure:
  - Examine the Python map application
  - Ensure you have input data
  - Run the streaming job
  - Display the streaming job

Hortonworks UNIVERSITY

Distributed Cache



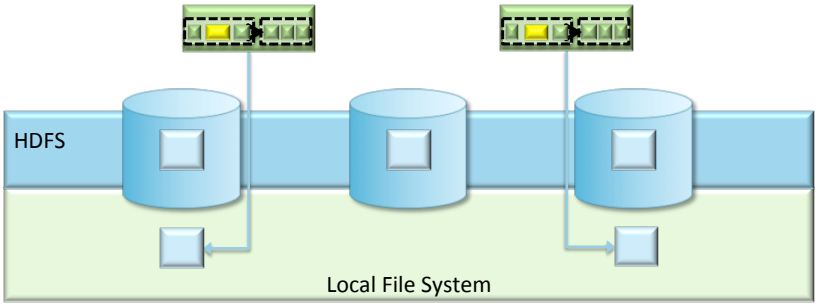
Shared File Access



Problem: Network I/O !

Hortonworks

Cache Solution



Hortonworks

Distributed Cache

- Copies files to the task nodes
  - Before the tasks are launched
  - Files are read-only
- Good for supplemental data
  - Used *locally* by mappers and reducers
- Often stored on node's Linux file system
  - Optionally upload from the local file system of the client
- Efficient
  - Downloaded once per node not once per task
  - Many tasks from the same job may run on a single node

Hortonworks

Distributed Cache Uses

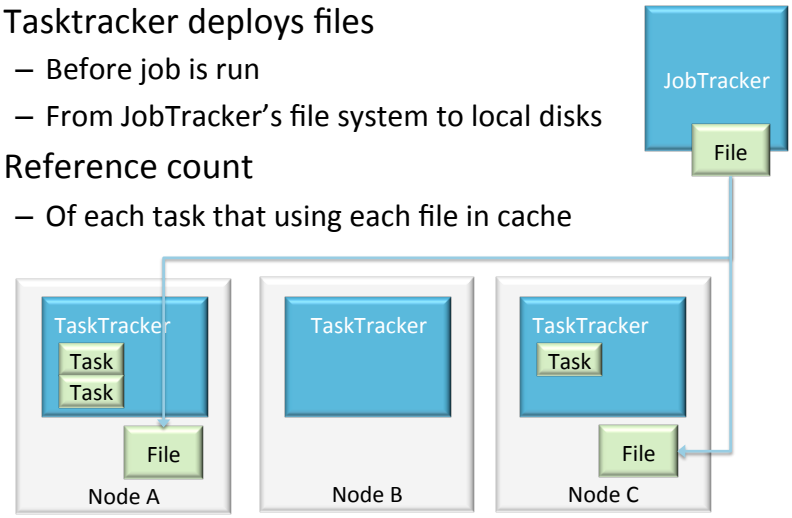
- Software distribution mechanism
  - For example distribute shared objects `(.so files)`
- `-files` option
  - comma separated list of paths
- `-archive` option
  - Distribute JAR, ZIP, tar, or tar.gz files
  - Decompression of archives is automatic
- `–libjars` option
  - modifies classpath of the mapper and reducer tasks
- Default cache size
  - is 10GB
  - can be set with the config `local.cache.size`

Distributed Cache Uses

- Classification
  - Key words for documents
  - City names for zip codes
- Distributed data
  - for Map side Joins

Cache Deployment

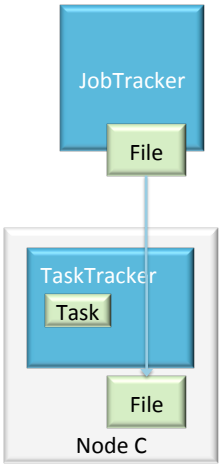
- Tasktracker deploys files
  - Before job is run
  - From JobTracker's file system to local disks
- Reference count
  - Of each task that using each file in cache



Hortonworks

Using DistributedCache

1. Place file on HDFS
2. Call the static method
   `DistributedCache.addCacheFile()` or `DistributedCache.addCacheArchive()` to specify the files
3. Mappers on each individual TaskTracker
   Call method `getLocalCacheFiles()`
   to get an array of local file Paths where the local copy is located
4. The mapper uses Java file I/O techniques to read the local copy



Hortonworks

Configure Method

- Use to load the cache data into memory
  - when the mapper is first initialized
  - overwritten each time the mapper is initialized
- Provides an array of file paths
  - to the local copy of the files pushed by DistributedCache
- Retrieves cache files
  - using its original name
  - relative to the working directory of the task

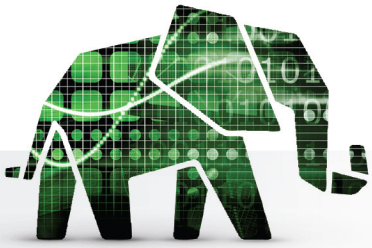
Configure Method

```
public static class MapClass extends MapReduceBase
                        implements Mapper<K, V, K, V> {

    private Path[] localArchives;
    private Path[] localFiles;

    public void configure(JobConf job) {
        // Get the cached archives/files
        localArchives = DistributedCache.getLocalCacheArchives(job);
        localFiles = DistributedCache.getLocalCacheFiles(job);
    }

    public void map(K key, V value, OutputCollector<K, V> output,
                Reporter reporter) throws IOException {
        // Use data from the cached archives/files here
    }
}
```

Lab 4.6: Using MapReduce with Distributed Cache

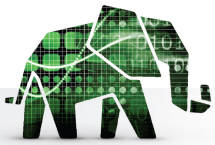
- Goal: Write a MapReduce program that uses the Distributed Cache
- Procedure:
  - Configure your job to get the local cache file for your mapper
  - In your Map Class open the File from the Distributed Cache
  - Tell conf to add the cache file
  - Compile and run your code

Hortonworks



Hortonworks UNIVERSITY

Lesson 5: Debugging MapReduce



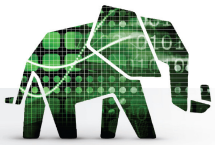
Lesson Outline

Topic	Objective
Monitoring MapReduce with JobTracker	Monitor MapReduce jobs using the JobTracker GUI
Debugging with Logging	Debug jobs using stdout, sterr, and syslogs

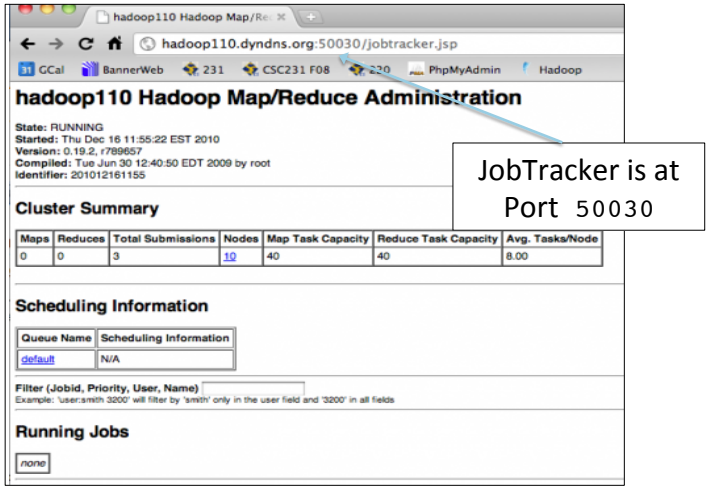
Hortonworks

Hortonworks UNIVERSITY

Monitoring MapReduce with JobTracker



Monitoring a MapReduce Job



Hortonworks

TaskTracker

Drill down from JobTracker GUI

Hortonworks

Lab 5.1: Monitoring a Job With JobTracker

- Goal: Run a MapReduce job and analyze statistics in JobTracker
- Procedure:
  - Run a Java MapReduce program
  - Use JobTracker GUI to analyze the job

Hortonworks

Hortonworks UNIVERSITY

Debugging MapReduce

Debugging Jobs

- Use an IDE
- Find the log files
  - Typically *hadoop-username-service-hostname.log*
  - Service = jobtracker, namenode, tasktracker, etc
  - For programs, tasktracker most relevant
- TaskTracker logs to `$(HADOOP_LOG_DIR)/userlogs`
  - Start at JobTracker GUI and drill down

Hortonworks

Logging

- All Hadoop deamons and jobs use log4j
  - `conf/log4j.properties` configures cluster-wide!
  - So you should use log4j, but with a different configuration file, or programmatic configuration for logging level.
- Hadoop logs output from `System.out.println()` and `System.err.println()` and log4j from your code
  - Stored in files named stdout, stderr, and syslog
  - Spread around the cluster
    - Colocated on nodes with TaskTracker daemons
    - `$(HADOOP_LOG_DIR)/userlogs/<subdirs>`

Hortonworks

Where are the logs?

Two methods

1. Use the JobTraker GUI and follow the links to see your stdout, stderr, and syslog (ie. log4j)
2. Somebody writes a script that collects logs from the cluster.
   - Links to logs are stored at `$HADOOP_LOG_DIR/userlogs`
     - There are other userlogs directories kept by each TaskTracker
     - This can make it confusing, when you are in directory userlogs and you only see a subset of your logs

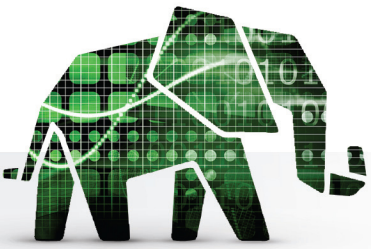
Hortonworks

Lab 5.2: Debug MapReduce Code

- Goal: Write a simple MapReduce program that uses log4j and standard error output (stderr) to debug code

Lab 5.3: MapReduce Use Case

- Baseball Stats



Hortonworks UNIVERSITY

Lesson 6: Pig

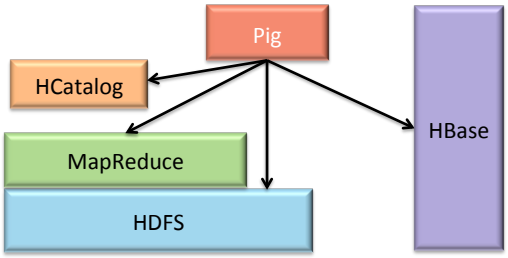
Lesson Outline

Topic	Objective
Pig Overview	Understand Pig
Grunt	Run commands from the Grunt CLI
Fundamentals of Pig	Write and understand simple Pig scripts
Pig Latin	Understand the Pig Latin scripting language

What is Pig?

- Pig is an extension of Hadoop that simplifies the ability to query large HDFS datasets
- Pig is made up of two main components:
  - A SQL-like data processing language called **Pig Latin**
  - A compiler that compiles and runs Pig Latin scripts
- Pig was created at Yahoo! to make it easier to analyze the data in your HDFS without the complexities of writing a traditional MapReduce program
- With Pig, you can develop MapReduce jobs with a few lines of Pig Latin

Pig In The EcoSystem



- Pig runs on Hadoop utilizing both HDFS and MapReduce
- By default, Pig reads and writes files from HDFS
- Pig stores intermediate data among MapReduce jobs

Hortonworks

Running Pig

A Pig Latin script executes in three modes:

1. **MapReduce**: the code executes as a MapReduce application on a Hadoop cluster (the default mode)

```
$ pig myscript.pig
```

2. **Local**: the code executes locally in a single JVM using a local text file (for development purposes)

```
$ pig -x local myscript.pig
```

3. **Interactive**: Pig commands are entered manually at a command prompt known as the Grunt shell

```
$ pig
grunt>
```


Hortonworks

Understanding Pig Execution

- Pig Latin is a **data flow language**
- During execution each statement is processed by the Pig interpreter
- If a statement is valid, it gets added to a **logical plan** built by the interpreter
- The steps in the logical plan do not actually execute until a DUMP or STORE command

A Pig Example

```
logevents = LOAD 'input/my.log' AS (date, level, code,
message);

severe = FILTER logevents BY (level == 'severe'
                              AND  code >= 500);

grouped = GROUP severe BY code;

STORE grouped INTO 'output/severeevents';
```

- The first three commands are built into a logical plan
- The STORE command triggers the logical plan to be built into a physical plan
- The physical plan will be executed as one or more MapReduce jobs

Grunt

An Interactive Pig Session

```
$ pig
grunt>  A = LOAD 'myfile';
grunt>  DUMP A;

(output appears here)
```

- Command line history and editing
- Tab will complete commands (but not filenames)
- To exit enter `quit`

Pig Command Options

- To see a full listing enter:
  pig –h or help
- Execute
  -e or –execute
  -f *scriptname* or -filename *scriptname*
- Specify a parameter setting
  -p or –parameter
  example: -p key1=value1 –p key2=value2
- List the properties that Pig will use if they are set by the user
  -h properties
- Display the version
  -version

Hortonworks

Grunt – HDFS Commands

- Grunt acts as a shell to access HDFS
- Commands include:

```
fs -ls
fs -cat filename
fs -copyFromLocal localfile hdfsfile
fs -copyToLocal hdfsfile localfile
fs -rm filename
fs -mkdir dirname
fs -mv fromLocation/filename toLocation/filename
```


Hortonworks

Lab 6.1: Introduction to Pig

- Run Pig commands using Grunt

The Fundamentals of Pig

Pig's Data Model

- 6 Scalar Types
  - int, long, float, double, chararray, bytearray
- 3 Complex types
  - Tuple: ordered set of values
    - (F, 66, 41000, 95103)
  - Bag: unordered collection of tuples
    - { (F, 66, 41000, 95103), (M, 40, 14000, 95102) }
  - Map: collection of key value pairs
    - [name#Bob, age#34]

Hortonworks

Relations

- Pig Latin statements work with **relations**
- A bag of tuples
- Similar to a table in a relational database, where the tuples in the bag correspond to rows in a table
- Unlike rows in a table
  - the tuples in a Pig relation do not have to contain the same number of fields
  - nor do the fields have to be the same data type
- Relation schemas are optional

Hortonworks

Defining Relations

- The `LOAD` command loads data from a file into a relation. The syntax looks like:

```
alias = LOAD 'data';
```

where `'data'` is either a filename or directory

- Use the `AS` option to define a schema for the relation:

```
alias = LOAD 'data' AS (name1:type,name2:type,...);
```

- TIP: Use the `DESCRIBE` command to view the schema of a relation

```
DESCRIBE alias;
```


Hortonworks

A Relation with a Schema

- Suppose we have the following data in HDFS:

Tom,21,94085,62000

John,45,95014,25000

Joe,21,94085,50000

Larry,45,95014,36000

Hans,21,94085,80000

- The data above represents the name, age, ZIP code and salary of employees

Hortonworks

A Relation with a Schema

- The following LOAD command defines a relation named employees with a schema:

```
employees = LOAD 'pig/input/File1'
       USING PigStorage(',')
       AS (name:chararray, age:int,
           zip:int, salary:double);
```

- The DESCRIBE command for employees outputs the following:

```
describe employees;
employees: {name: chararray,age: int,zip:
int,salary: double}
```


Hortonworks

Default Schema Datatype

- If not specified, the data type of a field in a relation defaults to bytearray
- What will the data type be for each field in the following relation?

```
employees = LOAD 'pig/input/File1'
      USING PigStorage(',')
      AS (name:chararray,
          age,
          zip:int,
          salary);
```


Hortonworks

Using a Schema

- Defining a schema allows you to refer to the values of a relation by the name of the field in the schema

```
employees = LOAD 'pig/input/File1'
    USING PigStorage(',')
    AS (name:chararray,age:int,
        zip:int,salary:double);
newhires = FILTER employees BY age <= 21;
```

- Because we defined a schema for the employees relation, the FILTER command can refer to the second field in the relation by the name "age"

Relations without a Schema

- If a relation does not define a schema, then Pig will simply load the data anyway (because "pigs eat anything")

```
employees = LOAD 'pig/input/File1'
    USING PigStorage(',');
DESCRIBE employees;
```

- The output of the above DESCRIBE command is:

```
Schema for employees unknown.
```

Relations without a Schema

- Without a schema, a field is referenced by its position within the relation
- $0 is the first field, $1 is the second field, and so on

```
employees = LOAD 'pig/input/File1'
    USING PigStorage(',');
newhires = FILTER employess BY $1 <= 21;
DUMP newhires;
```

- The output of the above commands is:

```
(Tom,21,94085,5000.0)
(Joe,21,94085,50000.0)
(Hans,21,94085,80000.0)
```


Hortonworks

Nulls in Pig

- Data elements may be null
  - Null means that the data element value is "undefined"
- In a LOAD command, null is automatically inserted for missing or invalid fields
- Example:

```
LOAD 'a.txt' AS (a1:int, a2:int) using
PigStorage(',');
```

This data:	Is loaded as:
1,2,3	(1,2)
5	(5, null)
6,bye	(6, null)

Hortonworks

The GROUP Operator

- The GROUP operator groups together tuples based on a specified key
- The usage for GROUP is:

  x = GROUP *alias* BY *expression*

  – *alias* = the name of the existing relation that you want to group together

  – *expression* = a tuple expression that is the key you want to group by
- The result of the GROUP command is a new relation

Hortonworks

A GROUP Example

```
employees = LOAD 'pig/input/File1'
            USING PigStorage(',')
            AS (name:chararray,age:int,
                zip:int,salary:double);
a = GROUP employees BY salary;
DESCRIBE a;
```

- The output of DESCRIBE is:

```
a: {group: double, employees:
        {(name: chararray,
          age:int,
          zip:int,
          salary: double)
         }
    }
```


Hortonworks

More GROUP Examples

```
daily = LOAD 'NYSE_daily' AS (exchange, stock,
                              date, dividends);
grpd = GROUP daily BY (exchange, stock);
DESCRIBE grpd;

grpd: {group: (exchange:bytearray,
stock:bytearray), daily: {{exchange:bytearray,
stock:bytearray, date:bytearray, dividends:
bytearray)}}
```

```
daily = LOAD 'NYSE_daily' AS (exchange, stock);
grpd = GROUP daily BY stock;
cnt = FOREACH grpd GENERATE group, COUNT (daily);
```


Hortonworks

The JOIN Operator

- The JOIN operator performs an inner join on two or more relations based on common field values
- The syntax for JOIN is:

x = JOIN *alias* BY *expression*, *alias* BY *expression*,…

  - *alias* = an existing relation
  - *expression* = a field of the relation
- The result of JOIN is a flat set of tuples

Hortonworks

A JOIN Example

- Suppose we add another set of data that contains the employee's name and a phone number:

Tom,4085551211

Tom,6505550123

John,4085554332

Joe,4085559898

Joe,4085557777

Hortonworks

A JOIN Example

```
e1 = LOAD 'pig/input/File1' USING PigStorage(',')
         AS (name:chararray,age:int,
             zip:int,salary:double);
e2 = LOAD 'pig/input/File2' USING PigStorage(',')
         AS (name:chararray,phone:chararray);
e3 = JOIN e1 BY name, e2 BY name;
DESCRIBE e3;
```

- The output of the DESCRIBE above is:

```
e3: {e1::name:chararray, e1::age:int,
     e1::zip:int,e1::salary:double,
     e2::name:chararray,e2::phone:chararray}
```


Hortonworks

A JOIN Example

- The JOIN output looks like:

```
grunt> DUMP e3;

(Joe,21,94085,50000.0,Joe,4085559898)
(Joe,21,94085,50000.0,Joe,4085557777)
(Tom,21,94085,5000.0,Tom,4085551211)
(Tom,21,94085,5000.0,Tom,6505550123)
(John,45,95014,25000.0,John,4085554332)
```


Hortonworks

The FOREACH Operator

- The FOREACH operator transforms data into a new relation based on the columns of the data
- The syntax looks like:

x = FOREACH *alias* GENERATE *expression*

  - *alias* = an existing relation
  - *expression* = an expression that determines the output

Hortonworks

A FOREACH Example

```
e1 = LOAD 'pig/input/File1' USING PigStorage(',')
        AS (name:chararray,age:int,
            zip:int,salary:double);
f = FOREACH e1 GENERATE age,salary;
DESCRIBE f;
DUMP f;
```

- The output of this example is a bag:

```
f: {age:int, salary:double}
(21,5000.0)
(45,25000.0)
(21,50000.0)
(45,36000.0)
(21,80000.0)
```


Hortonworks

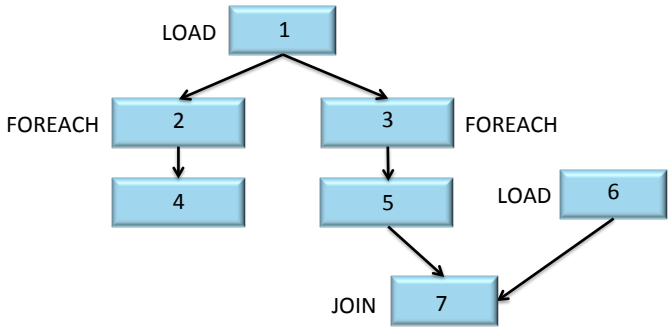
Using FOREACH on Groups

```
e1 = LOAD 'pig/input/File1' USING PigStorage(',')
        AS (name:chararray,age:int,
            zip:int,salary:double);
g = GROUP e1 BY age;
DESCRIBE g;
g: {group: int,e1: {(name:chararray,
           age:int, zip:int, salary:double)}}
f = FOREACH g GENERATE group, SUM(e1.salary);
DESCRIBE f;
f: {group: int,double}
DUMP f;

(21,135000.0)
(45,61000.0)
```


Hortonworks

Pig Latin Structured Processing Flow

- Pig Latin script describes a directed acyclic graph (DAG)
  - The edges are data flows and the nodes are operators that process the data



Hortonworks

Lab 6.2: Pig Data Operations

- In this lab, you will become familiar with relations, schemas, and some common Pig operations like JOIN, FILTER, FOREACH, EXPLAIN and ILLUSTRATE

Hortonworks

Pig Latin

An Example of a Pig Script

Find the top 10 URLs

```
Users = LOAD 'users' AS (name, age);
FilteredUsers = FILTER Users BY age >= 18 and age <=25;
Pages = LOAD 'pages' AS (user, url);
JoinResult = JOIN FilteredUsers BY name, Pages by user;
Grouped = GROUP JoinResult BY url;
Summed = FOREACH Grouped GENERATE group,
COUNT(JoinResult) as clicks;
Sorted = ORDER Summed BY clicks desc;
Top10 = LIMIT Sorted 10;
STORE Top10 INTO 'top10sites';
```


Case Sensitivity

- The names (aliases) of relations and fields are case sensitive
- The names of Pig Latin functions are case sensitive
- The names (aliases) of relations A, B, and C are case sensitive
- The names (aliases) of fields f1, f2, and f3 are case sensitive
- Function names PigStorage and COUNT are case sensitive
- The names of parameters and all other Pig Latin keywords are case insensitive
  - Keywords LOAD, USING, AS, GROUP, BY, FOREACH, GENERATE, and DUMP are case insensitive
  - They can also be written as load, using, as, group, by, etc.

The STORE Command

```
$ pig
grunt> A = LOAD 'inFile' USING PigStorage(',');
grunt> B = FILTER A BY ($1 == 'Male');
grunt> STORE B INTO 'outDir' using PigStorage(',');
```

- Executes the MapReduce job
- Puts data in HDFS
- Tab-delimited by default
- Specify different separator if needed

The FILTER Operator

```
$ pig
grunt> A = LOAD 'inFile' USING PigStorage(',');
grunt> B = FILTER A BY ($1 == 'Male');
grunt> STORE B INTO 'outDir' using PigStorage(',');
```

- Selects records to retain in the data pipeline
- Use with the keyword BY
- Contains a predicate evaluated to be true or false

Hortonworks

The DUMP Command

- Use to display data on your screen
- Each record displayed as a tuple surrounded by ( )
- Each bag is surrounded by { }
- Useful for:
  - Debugging
  - Prototyping
  - Looking at results of quick ad-hoc jobs
- Use LIMIT to limit the number of tuples (rows) returned to the display

Hortonworks

The ORDER BY Operator

- Sort a relation based on one or more fields
- Indicate the key or set of keys by which you wish to order your data

```
employees = LOAD 'pig/input/File1'
            USING PigStorage(',')
            AS (name:chararray,age:int,
                zip:int,salary:double);
sorted = ORDER employees BY salary;
DUMP sorted;

(Tom,21,94085,5000.0)
(John,45,95014,25000.0)
(Larry,45,95014,36000.0)
(Joe,21,94085,50000.0)
(Hans,21,94085,80000.0)
```


The LIMIT Operator

- Limits the number of output tuples
- Syntax is:

result = LIMIT *alias n*

- where *n* is the number of tuples to output

```
employees = LOAD 'pig/input/File1'
            USING PigStorage(',')
            AS (name:chararray,age:int,
                zip:int,salary:double);
g = GROUP employees BY age;
h = LIMIT g 1;
DUMP h;

(21,{(Tom,21,94085,5000.0),(Joe,21,94085,50000.0),
     (Hans,21,94085,80000.0)})
```

PARALLEL

- Allows you to specify the number of reducers
- Attach to any relational operator in Pig Latin
- Controls only reduce-side parallelism
- Works with the following operators:
  - group, order, distinct, join, limit, cogroup
  - To set the number of reducers to 10:
- Can be written as a script wide value
  - `set default_parallel 10;`

```
daily = LOAD 'NYSE_daily' AS (exchange, symbol,
          data, open, high, low, close,
          volume, adj_close);
bysymbl = GROUP daily BY symbol PARALLEL 10;
```


Hortonworks

Parameter Substitution

- Parameterize a script in order to specify information at runtime and vary execution
- Parameters are noted with "$" in a Pig script:

```
data = LOAD '$input' AS (name, age);
limitedData = LIMIT data $size;
DUMP limitedData;
```

- When run specify the parameters in this format:

```
-param input=excite-small.log -param size=1
```

- When used with executing a Pig job:

```
grunt> exec -param input=excite-small.log -param
size=4 myscript.pig
```


Hortonworks

The Example Pig Script Revisited

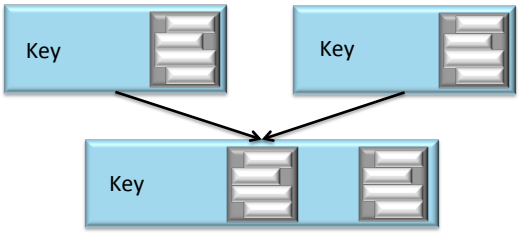
Find the top 10 URLs

```
Users = LOAD 'users' AS (name, age);
FilteredUsers = FILTER Users BY age >= 18 and age <=25;
Pages = LOAD 'pages' AS (user, url);
JoinResult = JOIN FilteredUsers BY name, Pages by user;
Grouped = GROUP JoinResult BY url;
Summed = FOREACH Grouped GENERATE group,
COUNT(JoinResult) as clicks;
Sorted = ORDER Summed BY clicks desc;
Top10 = LIMIT Sorted 10;
STORE Top10 INTO 'top10sites';
```


Hortonworks

The COGROUP Operator

- Collects records based on a key
- Pig brings together bags associated with a key
- Requires a reduce phase because it collects records with like keys together (partitioner's hash() imitates *group by*)
- Results in a record with a key and a bag for each input



Hortonworks

A COGROUP Example

```
e1 = LOAD 'pig/input/File1' USING PigStorage(',')
         AS (name:chararray,age:int,
             zip:int,salary:double);
e2 = LOAD 'pig/input/File2' USING PigStorage(',')
         AS (name:chararray,phone:chararray);
e3 = COGROUP e1 BY name, e2 BY name;
DESCRIBE e3;
```

- The output of the DESCRIBE above is:

```
e3: {group:chararray,
     e1:{(name:chararray, age:int, zip:int,
       salary:double)},
     e2:{(name:chararray, phone:chararray)}
}
```


Hortonworks

A COGROUP Example

- The output of the DESCRIBE above is:

```
grunt> DUMP e3;
(Joe,
    {(Joe,21,94085,50000.0)},
    {(Joe,4085559898),(Joe,4085557777)})
(Tom,
    {(Tom,21,94085,5000.0)},
    {(Tom,4085551211),(Tom,6505550123)})
(Hans,{(Hans,21,94085,80000.0)},{})
(John,
    {(John,45,95014,25000.0)},
    {(John,4085554332)})
(Larry,{(Larry,45,95014,36000.0)},{})
```


Hortonworks

The FLATTEN Operator

- Successive operations often produce bags of bags
- FLATTEN removes one level of nesting

```
players = LOAD 'baseball' AS (name:chararray,
     team:chararray, position:bag{t:(p:chararray)},
     bat:map[]);
pos = FOREACH players GENERATE name, FLATTEN(position) AS
position;
bypos = GROUP pos BY position;
```

```
Jorge Posada,New York Yankees,{(Catcher),(Designated_hitter)},...
```

Becomes:

```
Jorge Posada,Catcher
Jorge Posada,Designated_hitter
```


Hortonworks

The SAMPLE Operator

- Simple way to get a sample of your data
- Reads through all of the data and returns a percentage of the rows
- Expressed as a double value between 0 and 1
  - For example, 0.2 = 20%

```
employees = LOAD 'pig/input/File1' USING PigStorage(',')
     AS (name:chararray,age:int,zip:int,salary:double);
s = SAMPLE employees 0.2;
```


Hortonworks

Debugging Tips

- Use `illustrate, explain, and describe`
- Use local mode to test your script before running it in the cluster
  - Slow – but there is no waiting for a slot
  - Logs for your operations appear on your screen rather than on a remote task node
  - Local mode runs all in your local process – a debugger can be attached to the process.

EXPLAIN

- Shows how Pig compiles a script into MapReduce jobs
- Logical and physical execution details
  - Produces a logical plan
  - Logical operators that Pig will use to execute the script
  - Flow of the chart displayed is bottom to top
- For example:

```
bin/pig –x local –e 'EXPLAIN –script
explain.pig'
```

- Can also be displayed in graph (DAG) format:
  - `bin/pig –x local –e 'EXPLAIN –script input.pig –dot –out output.dat'`

ILLUSTRATE

- Use to help debug your Pig Latin script
- Takes a sample of your data and runs it though a script
  - It ensures that for every operator there is data which will pass through it and some which will not
  - If needed it will manufacture data that looks like the data entered and changes what it needs
- Displays the step-by-step transformations of the data
- Only works when a schema is defined for a relation

Hortonworks

User Defined Functions

- Users can combine Pig operators with their own User Defined Functions
- Collection of user-contributed UDFs in a Piggybank
  - Packaged and released with Pig
- UDF are written in Java and implemented as Java classes in jar files
  - Simply let Pig know about your JAR file

```
REGISTER 'your_path_to_piggybank/piggybank.jar';
divs = LOAD 'NYSE_dividends' AS (exchange:chararray,
     symbol:chararray, date:chararray, dividends:float);
backwards = FOREACH divs GENERATE
org.apache.pig.piggybank.evaluation.string.Reverse(symbol);
```

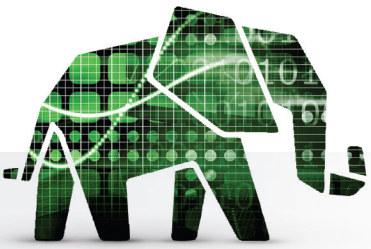

Hortonworks

Lab 6.3: Pig Latin

- In this lab, you will learn how to use Pig Latin commands like COGROUP, FOREACH and FLATTEN

Lab 6.4: Pig Baseball Challenge

- Goal: Attempt to solve a problem on your own with Pig



Hortonworks UNIVERSITY

Lesson 7: Hive

Lesson Outline

Topic	Objective
Hive basics	Understand Hive
Hive Shell	Understand the Hive CLI
HiveQL	Understand the Hive Query Language
Hive Data	Understand the Hive Data Types
Operators & Functions	Use some of the operators

What is Hive?

- Hive is a subproject of the Apache Hadoop project that provides a data warehousing layer built on top of Hadoop
- Hive allows you to define a structure for your unstructured big data, simplifying the process of performing analysis and queries by introducing a familiar, SQL-like language called HiveQL
- Hive is for data analysts familiar with SQL who need to do ad-hoc queries, summarization and data analysis on their HDFS data

Hive is not…

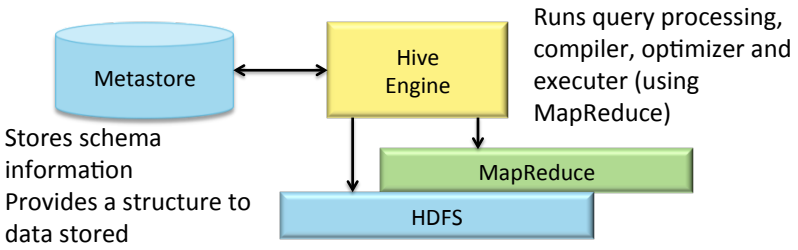
- Hive is not a relational database
- Hive uses a database to store metadata, but the data that Hive processes is stored in HDFS
- Hive is not designed for on-line transaction processing and does not offer real-time queries and row level updates

Pig vs. Hive

- Pig and Hive work well together
- Hive is a good choice:
  - when you want to query the data
  - when you need an answer to a specific questions
  - if you are familiar with SQL
- Pig is a good choice:
  - for ETL (Extract -> Transform -> Load)
  - preparing your data so that it is easier to analyze
  - when you have a long series of steps to perform
- Many businesses use both Pig and Hive together

Hive Basics

- Data Warehousing package built on top of Hadoop
- System for querying and managing structured data
  - Uses MapReduce for execution
  - Uses HDFS (or HBase) for storage



Hortonworks

What is a Hive Table?

- A Hive table consists of:
  - Data: typically a file or group of files in HDFS
  - Schema: in the form of metadata stored in a relational database
- Schema and data are separate.
  - A schema can be defined for existing data
  - Data can be added or removed independently
  - Hive can be "pointed" at existing data
- You have to define a schema if you have existing data in HDFS that you want to use in Hive

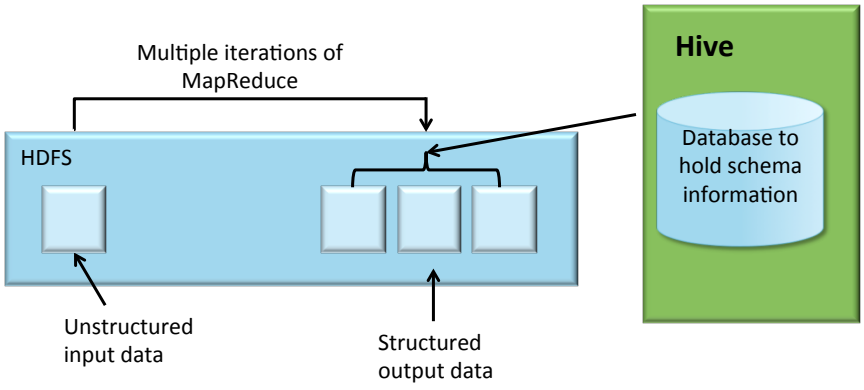
Hortonworks

HiveQL

- Hive's SQL like language, HiveQL, uses familiar relational database concepts such as tables, rows, columns and schema
- Designed to work with structured data
- Converts SQL queries to into MapReduce jobs
- Supports uses such as:
  - Ad-hoc queries
  - Summarization
  - Data Analysis

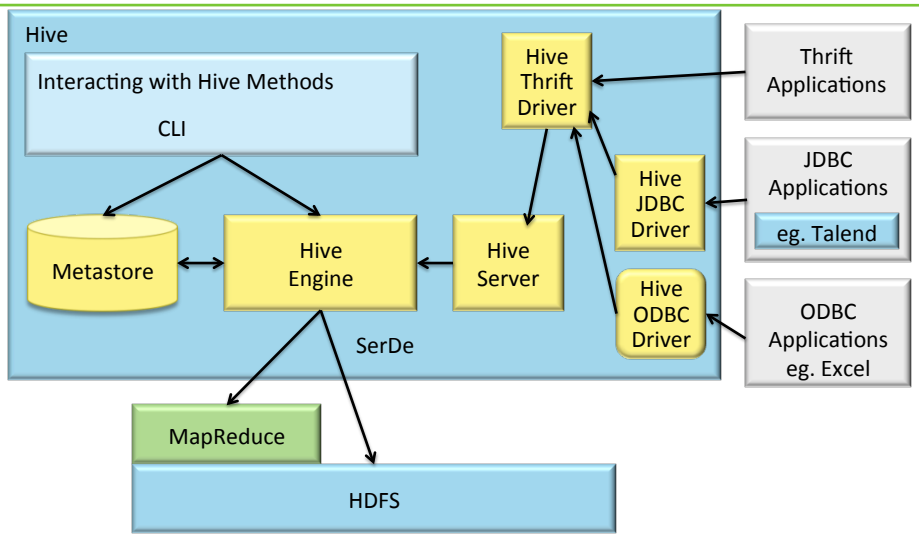
Hortonworks

The Typical Hive Use Case



Hortonworks

Hive Architecture



Hortonworks

Hortonworks

Hive Shell

Running Jobs with the Hive Shell

- Primary way people use to interact with Hive

```
$ hive
hive>
```

- Can run in the shell in a non-interactive way

```
$ hive -f myhive.q
```

  - Use `-S` option to have only the results show

Hive Shell - Information

- At terminal enter:
  - `$ hive`
- List all properties and values:
  - `hive> set -v`
- List and describe tables
  - `hive> show tables;`
  - `hive> describe <tablename>;`
  - `hive> describe extended <tablename>;`
- List and describe functions
  - `hive> show functions;`
  - `hive> describe function <functionname>;`

Hive Shell – Querying Data

- Selecting Data
  - `hive> SELECT * FROM students;`
  - `hive> SELECT * FROM students`
    `WHERE gpa > 3.6 SORT BY gpa ASC;`

HiveQL

HiveQL

- HiveQL is similar to other SQLs
- User does not need to know Map/Reduce
- HiveQL is based on the SQL-92 specification
- Supports multi-table inserts via your code

Table Operations

- Defining a table:

```
hive> CREATE TABLE mytable (name chararray,
age int)
        ROW FORMAT DELIMITED
        FIELDS TERMINATED BY ','
        STORED AS TEXTFILE;
```

- ROW FORMAT is a Hive-unique command that indicate that each row is comma delimited text
- HiveQL statements are terminated with a semicolon `';'`
- Other table operations:
  - `SHOW TABLES`
  - `CREATE TABLE`
  - `ALTER TABLE`
  - `DROP TABLE`

Managing Tables

- See current tables:
  - `hive> SHOW TABLES;`
- Check the schema:
  - `hive> DESCRIBE mytable;`
- Change the table name:
  - `hive> ALTER TABLE mytable RENAME to mt;`
- Add a column
  - `hive> ALTER TABLE mytable ADD COLUMNS (mycol STRING);`
- Drop a partition
  - `hive> ALTER TABLE mytable DROP PARTITION (age=17)`

Loading Data

- Use LOAD DATA to import data into a Hive table
- To load data indicate the path of the data
  - `LOAD DATA LOCAL INPATH 'input/mydata/data.txt' INTO TABLE myTable;`
- The files are not modified by Hive – they are loaded in as is
- Hive warehouse default location is:
  - `/user/hive/warehouse`
- Hive can read all of the files in a particular directory
- Use the word OVERWRITE to write over a file of the same name
- The schema is checked when the data is queried. If a row does not match the schema, it will be read as null

INSERT

- Use INSERT statement to populate data into a table from another Hive table
- Since query results are usually large it is best to use an INSERT clause to tell Hive where to store your query
- Creating a table and inserting into it

```
hive> CREATE TABLE age_count  (name chararray, age int);
hive> INSERT OVERWRITE TABLE mytable
        SELECT age, COUNT(age)
        FROM mytable;
```


Hortonworks

INSERT OVERWRITE

- OVERWRITE is used to replace the data in the table, otherwise the data is appended to the table.
  - Append happens by adding files to the directory holding the table data
  - `INSERT OVERWRITE 'mytable' SELECT * FROM mytable;`
- Can write to a directory in HDFS
  - `INSERT OVERWRITE DIRECTORY '/hdfs/myresult' SELECT * FROM mytable;`
- Can write to a local directory (Linux typically)
  - `INSERT OVERWRITE LOCAL DIRECTORY...`

Hortonworks

SELECT

- SELECT
  - Simple query example:
    - `SELECT * FROM mytable;`
- Supports the following:
  - WHERE clause
  - ALL and DISTINCT
  - GROUP BY and HAVING
  - LIMIT clause
    - Rows returned are chosen at random
  - Can use REGEX Column Specification
    - Example:
      - `SELECT ' (ds|hr)?+.+' FROM sales;`

Subqueries

- Hive supports subqueries only in the FROM clause
- For example:

```
SELECT total FROM
    (SELECT c1 + c2 AS total FROM mytable) my_query;
```

- The subquery has to be given a name (my_query in this example)
- The columns in the subquery SELECT list are available in the outer query

JOIN - Inner Joins

- Inner joins are implemented with ease:

```
SELECT * FROM students;
   Steve    2.8
   Raman    3.2
   Mary     3.9
SELECT * FROM grades;
   2.8      B
   3.2      B+
   3.9      A
 SELECT students.*, grades.*
   FROM students JOIN grades ON (students.grade =
grades.grade)
      Steve     2.8  2.8  B
      Raman     3.2  3.2  B+
       Mary     3.9  3.9  A
```


JOIN – Outer Joins

- Allows for the finding of rows with non-matches in the tables being joined
- Outer Joins can be of three types
  - LEFT OUTER JOIN
    - Returns a row for every row in the first table entered
  - RIGHT OUTER JOIN
    - Returns a row for every row in the second table entered
  - FULL OUTER JOIN
    - Returns a row for every row from both tables

Sorting

- ORDER BY
  - Sorts but sets the number of reducers to 1
- SORT BY
  - Multiple reducers with a sorted file from each

Hive Summary

- Not suitable for complex machine learning algorithms
- All jobs have a minimum overhead and will take time just for set up
  - Still Hadoop MapReduce on a cluster
- Good for batch jobs on large amounts of append only data
  - Immutable filesystem
  - Does not support row level updates (except through file deletion or creation)

Lab 7.1: Intro to Hive

- Load datafile
- Create another table
- Load new table from existing datafile

Hive Data

Hive Data Model

- Tables
  - Schema is associated with files
- Partitions
  - Partition Columns into physical directories
  - Often organized by date to increase performance for date based queries

Tables

- A Hive table consists of the data stored and the metadata description of the data
  - The metadata is stored in a relational database
- Tables can be Hive-managed or external
  - Hive manages and can delete Hive-managed data
  - Hive can drop an external table, but it does not delete the data
  - Use external tables if other processes besides Hive will be accessing the data

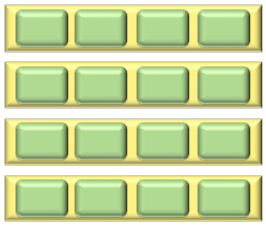
Partitions

- A way to divide a table into parts based on the value of a partition column
  - Often date is used
- Queries that select a partitioned value are more efficient
- Subpartitions can be created
- Dynamic partitions are supported

Hortonworks

Hash Buckets

- Tables or partitions can be further divided into buckets
- Bucketing divides data into a specified number of files based on the hash of the bucket column
- Each partition will have the specified number of files
- Buckets are useful to:
  - Make more efficient queries
  - Make sampling more efficient



Data partitioned by date, with each Date consisting of four buckets (arbitrary or by value)

Hortonworks

Sampling Data

- Buckets – means to provide efficient queries on a random sample of data
- If there are 20 buckets then a Hive query can be run against only 1/20th of the data
- Specify the number of buckets in CREATE TABLE:

```
hive> CREATE TABLE mytable (USERID BIGINT, age
INT, gpa DOUBLE)
   PARTITION BY (age)
   CLUSTERED BY (userid) INTO 20 BUCKETS;
```


Hive Data Types - Scalar

- TINYINT – 1 byte integer
- BOOLEAN
- SMALLINT – 2 byte integer
- INT – 4 byte integer
- BIGINT – 8 byte integer
- DOUBLE – Double precision
- STRING – Sequence of characters

Hive Data Types – Complex

- STRUCT
  - Collection of named fields
  - Fields can be of different types
- MAP
  - Unordered collection of key-value pairs. Keys must be primitives; values may be any type
  - For a particular map the keys must be the same type and the values must be the same type
- ARRAY
  - An ordered collection of fields of the same type

Lab 7.2: Hive External Tables & Partitions

- Load data into HDFS which will be used for an external table
- Use dynamic partitions in Hive

Hive Operators and Functions

Operators

- Typical SQL operators supported
  - = for equality
  - IS NULL for testing whether null
  - LIKE for pattern matching
  - Arithmetic operators (+, -, etc.)
  - Logical operators
    - || is logical OR (syntax as in MySQL, not SQL-92)
- Built-in Operators include:
  - Relational operators, arithmetic operators, logical operators, complex type constructors and operators on complex types

Some Hive Built-in Functions

- Mathematical Functions (`floor, rand, log`, etc.)
- Collection Functions (`size`)
  - Size of a map or an array
- Type Conversion Functions (`cast`)
- Date Functions (`unix_timestamp, day`, etc.)
- Conditional Functions (`if, case … when … then`, etc.)
- String Functions (`concat, substr, rpad`, etc.)
- Misc. Functions (`xpath`, etc.)
- Note: To see all functions in Hive run a **`SHOW FUNCTIONS`**
- Note: to get a description of a function enter:
  - **`DESCRIBE FUNCTION [function name]`**

Hortonworks

Hive User Defined Functions

- Must be written in Java
  - UDFs must implement an evaluate() method
- Three types of User-Defined Functions:
  - User Defined Function (UDF)
    - Operates on a single row at a time
  - User Defined Aggregate Function (UDAF)
    - Works on multiple input rows and produces a single row output
  - User Defined Table Function (UDTF)
    - Operates on a single row and produces multiple rows output (table)
- Provides for Pre and Post Hooks
  - Can be used for statement validation, auditing, replication, and authorization

Hortonworks

UDAF and UDTF

- Hive Built-in Aggregate Functions (UDAF) includes:
  - `Count`
  - `Sum`
  - `Avg`
  - `Percentile`
- Hive Built-in Table-Generating Functions
  - `Explode (expands an array to rows)`
  - `Json_tuple (extract strings from JSON string)`

Lab 7.3: Hive User Defined Functions

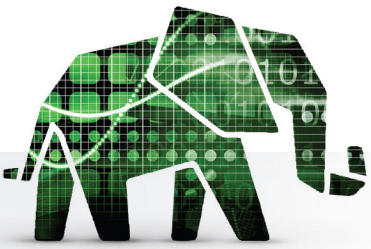
- Use a Java jar as a User Defined Function in Hive

Lab 7.4: Using Hive to Read Pig Data

- Load data with Pig
- Reading that data using Hive

Lab 7.5: Hive Use Case

- Challenge: Baseball Stats



Hortonworks UNIVERSITY

Lesson 8: HCatalog

Lesson Outline

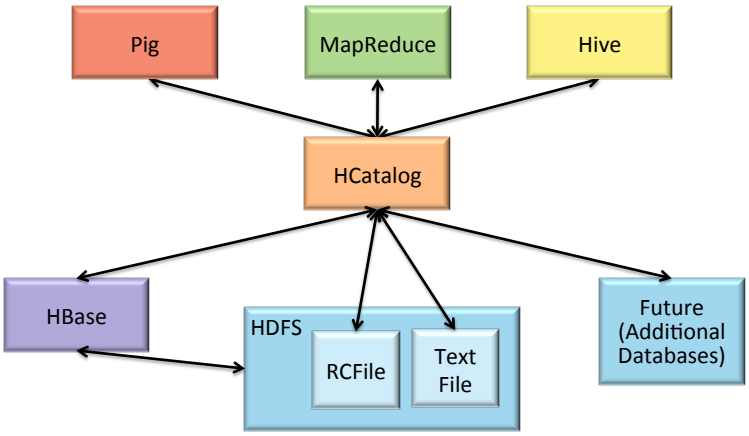
Topic	Objective
HCatalog Overview	Understand HCatalog architecture, data types and uses
HCatalog API	Use the HCatalog interfaces
HCatalog CLI	Use the HCatalog command line interface

What is HCatalog?

- Table management and storage management layer for Hadoop
- HCatalog provides a shared schema and data type mechanism
  - Enables users with different data processing tools – Pig, MapReduce and Hive to have data interoperability
  - HCatalog provides read and write interfaces for Pig, MapReduce and Hive to HDFS (or other data sources)
  - HCatalog's data abstraction presents users with a relational view of data
- Command line interface for data manipulation
- Designed to be accessed though other programs such as Pig, Hive, MapReduce and HBase
- HCatalog installs on top of Hive

HCatalog in the EcoSystem

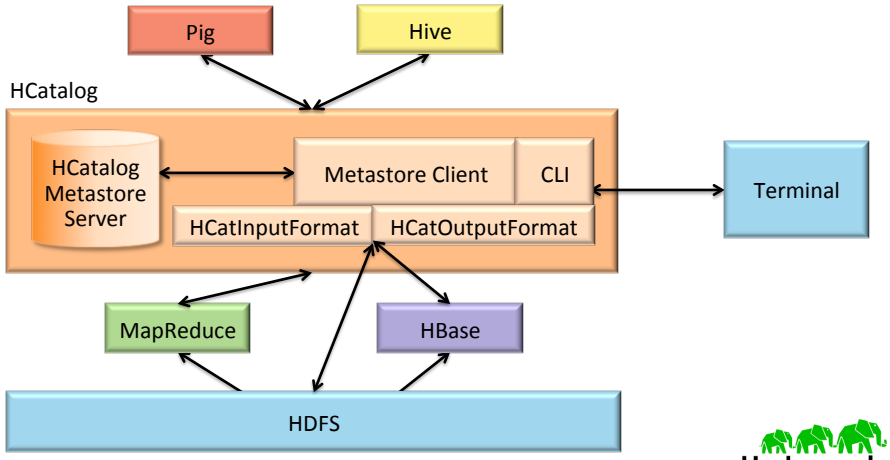
- Provides an abstraction layer for data storage
  - Access data through HCatalog rather than underlying software



Hortonworks

HCatalog Architecture

- HCatalog provides read and write interfaces for MapReduce, Pig and Hive.
- Command line interface for data definitions.



Hortonworks

HCatalog Components

- HCatInputFormat interface
- HCatOutputFormat interface
- Metastore Client
- HCatalog Metastore
  - Stores metadata in a standard relational database
- CLI

HCatalog Data Storage

- Data is stored in tables and these tables can be placed in databases
- Tables can be partitioned on one or more keys
  - For a given key value one partition contains all rows with that value

Invoking HCatalog CLI "shell"

- **`-g mygroup`** ....
  - Indicates to HCatalog that table that needs to be created must have group as "mygroup"
- **`-p rwxr-xr-x`** ....
  - Indicates to HCatalog that table that needs to be created must have permissions as "rwxr-xr-x"
- **`-f myscript.hcatalog`** ....
  - Indicates to hcatalog that myscript.hcatalog is a file which contains DDL commands it needs to execute.
- **`-e 'create table mytable(a int);'`** ....
  - Indicates to HCatalog to treat the following string as DDL command and execute it.

HCatalog DDL

- CREATE/ALTER/DROP Table
- SHOW TABLES
- SHOW FUNCTIONS
- DESCRIBE
- Many of the commands in Hive are supported
  - Any command which is not supported throws an exception and returns the message "Operation Not Supported".

Accessing HCatalog Metastore through CLI

- Using HCatalog client
  - Execute a script file

```
hcat –f "myscript.hcatalog"
```

  - Execute DDL

```
hcat –e 'create table mytable(a int);'
```


Hortonworks

Define a New Schema

- A schema is defined as an HCatalog table:

```
create table mytable (
  id                int,
  firstname         string,
  lastname          string
)
comment 'An example of an HCatalog table'
partitioned by (birthday string)
stored as sequencefile;
```


Hortonworks

Define a New Schema

- Suppose the script on the previous slide is saved in a file named create_table.hcatalog
- To execute the script, enter the command:

```
$ hcat -f create_table.hcatalog
$ hcat -e 'describe mytable'
OK
id int
firstname   string
lastname string
birthday string
Time taken: 0.859 seconds
```


Pig Specific HCatStorer Interface

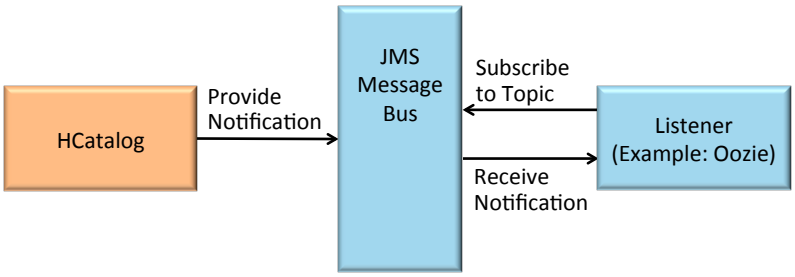
- Used with Pig scripts to write data to HCatalog managed tables.
- Accepts a table to write to and optionally a specification of partition keys to create a new partition
- HCatStorer is implemented on top of HCatOutputFormat
  - HCatStorer is accessed via a Pig store statement.
  - Storing into table partitioned on month, date, hour

…

```
STORE my_processed_data INTO 'dbname.tablenmame'
        USING
org.apache.hcatalog.pig.HCatStorer('month=12,
date=25, hour=0300', 'a:int,b:chararray,c:map[]');
```

Partition Notifications

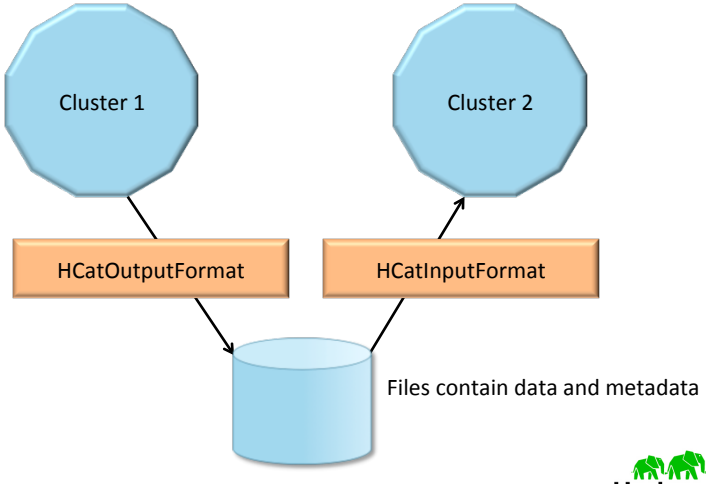
- Provide notification of a new partition or a new set of partitions



Hortonworks

Importing and Exporting Between Clusters

- Use IMPORT and EXPORT commands



Hortonworks

HCatalog IMPORT and EXPORT

- Hcatalog IMPORT and EXPORT commands enable you to:
  - Extract the data and the metadata associated with a table in HCatalog
  - Create the data and metadata associated with a table
  - Import the data and the metadata into an existing HCatalog instance
  - Use the exported package as input to both pig and MapReduce jobs

Hortonworks

Data Storage Structure

- An _metadata file that contains the metadata of the table, and if the table is partitioned, for all the exported partitions
- A subdirectory hierarchy for each exported partition (or just one "data" subdirectory, in case of a non-partitioned table) that contains the data files of the table/partitions

Hortonworks

EXPORT COMMAND

- Syntax
  - `EXPORT TABLE tablename [PARTITION (partcol1=val1, partcol2=val2, …)] TO 'filepath'`
- Filepath can be:
  - Relative path (`'project/data1'`)
  - Absolute path (`'/user/hcat/project/data1'`)
  - Full URI (*`'hsfs://namenode:9000/user/hcat/project/data1'`*)

Hortonworks

IMPORT Command

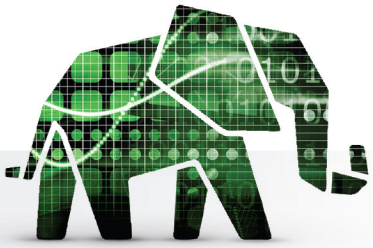
- Syntax:
  - `IMPORT [[EXTERNAL] TABLE tablename [PARTITION (partcol1=val1, partcol2=val2, …)]] FROM 'filepath' [LOCATION 'tablepath']`
- EXTERNAL indicates that the imported table is an external table
- LOCATION is the target loaction the table will be copied to
  - For managed tables, the default location of the table within the warehouse/database structure is used
  - For external tables, the data is imported in-place, that is, no copying takes place

Hortonworks

Lab 8.1: HCatalog with Pig and Hive

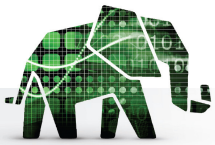
- Use HCatalog to define data
- ETL with Pig
- Query with Hive

Hortonworks



Hortonworks UNIVERSITY

Lesson 9: HBase



Lesson Outline

Topic	Objective
HBase Overview	Understand HBase and its use cases
Architecture	Understand HBase's architecture
Methods	Use some of HBase methods

Hortonworks

What is a NoSQL Database?

- NoSQL database systems are developed to manage large volumes of data that do not necessarily follow a fixed schema
- Means "not only SQL", meaning you can access the database with or without SQL
- Designed for distributed datastore for very large-scale data needs
- Example NoSQL databases include MarkLogic, HBase, Cassandra, MongoDB, and Accumulo

What is HBase?

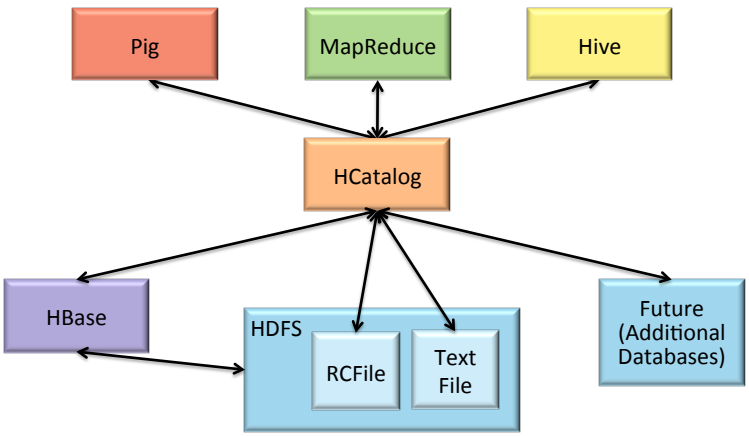
- HBase is a NoSQL database that stores its data in HDFS
- Inherits the characteristics of HDFS:
  - Distributed
  - Linearly scalable
  - Reliable
  - Big Data!
- Column-oriented
- Use HBase when you need random, realtime (as opposed to batch) read/write access to your Big Data

HBase is not…

- Not a relational database
- Not a standalone solution – it relies on HDFS
- Not a replacement for a traditional RDBMS
- Not optimized for classic, traditional applications
- Not ACID compliant

Hortonworks

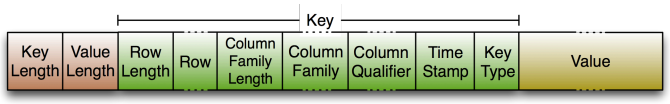
HBase in the EcoSystem



Hortonworks

Key/Value Storage in HBase

- Coordinates of the cell include:
  - Row key
  - Name of column family
  - Column qualifier
  - Timestamp
- Key/Value Format



Hortonworks

Column Families

- Columns families are composed of columns
  - Column families can consist of millions of columns
- Column families need to be defined when the table is being created
- Typically a column name is made up of its column family prefix and a qualifier

Hortonworks

Column Family Diagram

- All columns in a column family will be stored together.
  - This allows for quicker access for items that are accessed together

Column Family: user

Name	Age	Gender	City	State
Edna	34	f	Chicago	Illinois
Ram	44	m	Atlanta	Georgia
Eric	54	m	Flagstaff	Arizona

Hortonworks

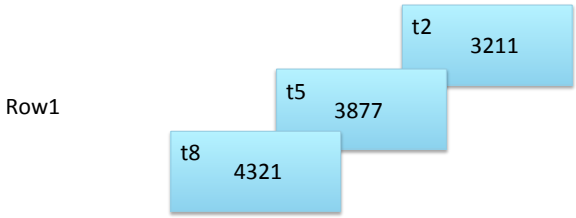
Column Family Members

- Row columns are grouped into column families
  - All column family members have a common prefix with different qualifiers
  - Examples:
    - members of the “car” column family:
      - car:make
      - car:model
    - Members of the “user” column family:
      - user:name
      - user:age
      - user:gender
      - user:city
      - user:state
  - The column family prefix must be composed of printable characters
  - The column family qualifier can be made of any arbitrary bytes

Hortonworks

Cells

- Value is in a cell
- Cells can have the same row but a different time stamp
- A cell is located by its:
  - Table Name => Column Family => Row ID => Timestamp



Hortonworks

Data Storage – Relational vs. HBase

Relational Data Base

	Column1	Column2	Column3	Column4
Row1	f - t5 a – t1	null	null	d – t4
Row2	null	b – t1	null	null
Row3	null	null	null	e – t4
Row4	c – t3	null	g – t5	null

HBase

Data is located by cell coordinates consisting of row key, column family name, column qualifier and timestamp

HFile

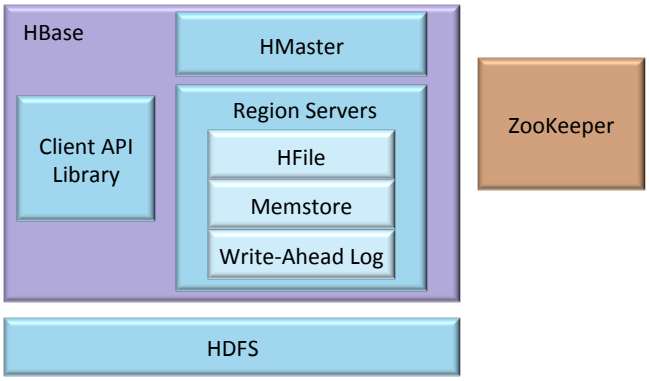
Column1	Column2	Column3	Column4
f – t5	B – t1	g – t5	d – t4
a – t1			e – t4
C – t3			

Hortonworks

Hortonworks

HBase Architecture

HBase Architecture



Hortonworks

HBase API

The HTable Interface

- Primary client interface to HBase is `org.apache.hadoop.hbase.client.HTable`
  - Used for storing data, retrieving data and deleting obsolete values and more.
- Retrieve values within a row by row key

```
Configuration config = HBaseConfiguration.create();
HTable htable = new HTable(config, "users");
```


Put

- Stores data in HBase
- Supply a row in order to create a Put instance

```
String userName = "user" + i;
String email = "user" + i + "@foo.com";
String phone = "555-1234";
byte [] key = Bytes.toBytes(userName);
Put put = new Put (key);
```


Hortonworks

Add Method

- After creating a Put instance can use “add” to add data to it
- Each time add calls a single column

```
byte [] family = Bytes.toBytes("info");
put.add(family, Bytes.toBytes("email"),
                        Bytes.toBytes(email));
put.add(family, Bytes.toBytes("phone"),
                        Bytes.toBytes(phone));
htable.put(put);
```


Hortonworks

Retrieving Data

```
Get get = new Get(key);
Result result = htable.get(get);
if (result.isEmpty()) {
    System.out.println(userName + " : not found");
} else {
    byte[] family = Bytes.toBytes("info");
    byte[] emailCol = Bytes.toBytes("email");
    KeyValue kv = result.getColumnLatest(family,
emailCol);
    if (kv == null)
      System.out.println("column 'email' not found");
    } else {
      byte[] value = kv.getValue();
      String email = new String(value);
      System.out.println("email=" + email);
    }
}
```


Hortonworks

The ResultScanner Class

```
Configuration config = HBaseConfiguration.create();
HTable htable = new HTable(config, "users");
byte [] family = Bytes.toBytes("info");
byte [] emailColumn = Bytes.toBytes("email");
Scan scan = new Scan();
ResultScanner scanner = htable.getScanner(scan);
try {
   for (Result rr : scanner) {
      KeyValue kv = rr.getColumnLatest(family,
emailColumn);
      if (kv != null)  {
         String user = new String(kv.getRow());
         String email = new String(kv.getValue());
         System.out.println (user + "=" + email);
      }
   }
} catch (Exception ex) {}
```


Hortonworks

HBase CLI

- Can get a status of HBase (can be summary, simple or detailed)

```
hbase> status 'simple'
```

- Create a table

```
hbase> create 'mytable', {NAME => 'myColFamily'}
```

- Describe a table

```
hbase> describe table 'mytable'
```

- List all tables

```
hbase> list
```


Hortonworks

HBase - UserScan.java

```
public class UserScan {

    public static void main(String[] args) throws Exception {
        Configuration config = HBaseConfiguration.create();
        HTable htable = new HTable(config, "users");
        byte [] family = Bytes.toBytes("info");
        byte [] emailColumn = Bytes.toBytes("email");
        Scan scan = new Scan();
        ResultScanner scanner = htable.getScanner(scan);
        try {
            for (Result rr : scanner) {
                KeyValue kv = rr.getColumnLatest(family, emailColumn);
                if (kv != null)  {
                    String user = new String(kv.getRow());
                    String email = new String(kv.getValue());
                    System.out.println (user + "=" + email);
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally { scanner.close(); }
        htable.close();
    }
}
```


Hortonworks

HBase - UserPut.java

```
public class UserPut {

    public static void main(String[] args) throws Exception {
        Configuration config = HBaseConfiguration.create();
        // create table
        HTableDescriptor tableDescriptor = new
HTableDescriptor("users");
        HBaseAdmin admin = new HBaseAdmin(config);
        if (!admin.tableExists("users")) {
            tableDescriptor.addFamily(new HColumnDescriptor("info"));
            admin.createTable(tableDescriptor);
            System.out.println ("created table");
        } else {
            System.out.println ("table exists");
        }
```


Hortonworks

HBase - UserPut.java

```
         HTable htable = new HTable(config, "users");
        long t1 = System.currentTimeMillis();
        int total = 10;
        byte [] family = Bytes.toBytes("info");
        for (int i=1; i <= total ; i++) {
            String userName = "user" + i;
            String email = "user" + i + "@foo.com";
            String phone = "555-1234";
            byte [] key = Bytes.toBytes(userName);
            Put put = new Put (key);
            // add email
            byte [] qualifier = Bytes.toBytes("email");
            byte [] value = Bytes.toBytes(email);
            put.add(family, qualifier, value);
            // and phone number
            put.add(family, Bytes.toBytes("phone"), Bytes.toBytes(phone));
            htable.put(put);
            System.out.println ("added user : " + userName);
        }
        long t2 = System.currentTimeMillis();
        System.out.println ("inserted " + total + " users  in " + (t2-t1)
+ " ms");
        htable.close();
    }
}
```


Hortonworks

HBase - UserGet.java

```
public class UserGet {
    public static void main(String[] args) throws Exception {
        Configuration config = HBaseConfiguration.create();
        HTable htable = new HTable(config, "users");
        findUser("user1", htable);
        findUser("user2", htable);
        findUser("userXXX", htable);
    }
```


Hortonworks

HBase - UserGet.java (2 of 2)

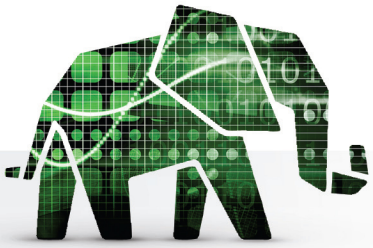
```
public static void findUser(String userName, HTable htable)
    throws Exception {
        System.out.println("querying for : " + userName);
        byte[] key = Bytes.toBytes(userName);

        Get get = new Get(key);
        Result result = htable.get(get);
        if (result.isEmpty()) {
            // first check, this row may not exist
            System.out.println("     " + userName + " : not found");
        } else {
            byte[] family = Bytes.toBytes("info");
            byte[] emailCol = Bytes.toBytes("email");
            KeyValue kv = result.getColumnLatest(family, emailCol);
            if (kv == null) {   // if row exist, col may not exist
                System.out.println("     column 'email' not found");
            } else { // found
                byte[] value = kv.getValue();
                String email = new String(value);
                System.out.println("     email=" + email);
            }
```


Hortonworks

Lab 9.1: HBase Basics

- Creating a table via CLI
- Inserting into a table
- Creating a table via Java
- Using Java to query a database
- HBase counters

Hortonworks



Hortonworks UNIVERSITY

Lesson 10: Hadoop Web Services

What is WebHDFS?

- HTTP REST API
  - Permits clients to access HDFS from multiple languages without installing Hadoop
  - Read and write calls are redirected to the corresponding datanodes
  - Built-in component of HDFS
- Runs as part of NameNode and DataNodes
  - Can use all of HDFS functionality
  - Does not require an additional server to run
- Kerberos support for security
- Reads & writes data between HDFS versions

Hortonworks

Using WebHDFS

- The URL to access the REST API of WebHDFS is:
  http://*hostname:port*/webhdfs/v1
- To read a file named input/mydata:

```
curl -i -L "http://host:50070/webhdfs/v1/input/mydata?op=OPEN"
```

- To list the contents of a directory named “foo”:

```
curl -i "http://host:50070/webhdfs/v1/foo/?op=LISTSTATUS"
```

- To make a directory named “myoutput”:

```
curl -i -X PUT "http://host:50070/webhdfs/v1/myoutput?op=MKDIRS"
```


What is Templeton?

- Templeton provides a REST API to the following Hadoop technologies:
  - HCatalog
  - Hadoop MapReduce
  - Pig
  - Hive

Using Templeton

- The URL to access the REST API of Templeton is:
  - http://*hostname:port*/templeton/v1/
- Here is an example of running a MapReduce job:

```
% hadoop fs -put wordcount.jar .
% hadoop fs -put transform.jar .
% curl -s -d user.name=hadoop_user \
       -d jar=wordcount.jar \
       -d class=com.hortonworks.WordCount \
       -d libjars=transform.jar \
       -d arg=wordcount/input \
       -d arg=wordcount/output \
       'http://host:50111/templeton/v1/mapreduce/jar'
```


Hortonworks