

# **Mastering Hadoop, Spark & HBase for Big Data Solution**

Will Du

# Introducing Apache Hadoop

# Introduction to Apache Hadoop

Wayne Shen

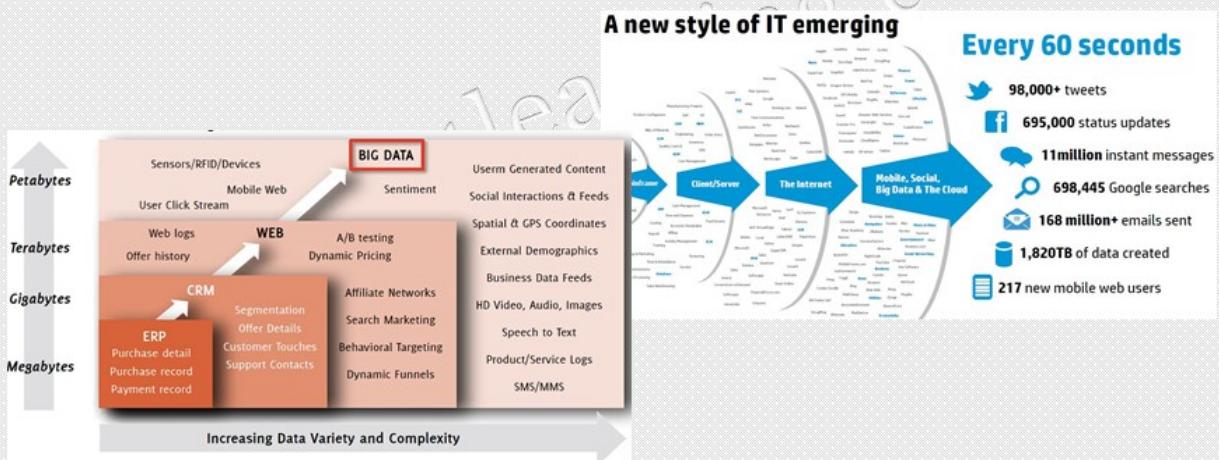


<http://www.it21learning.com>

## What is Big Data?

# What is Big Data?

- Big Data is a term that describes **large volumes** of high **velocity, complex** and **variable** data that require advanced techniques and technologies to enable the capture, storage, distribution, management, and analysis of the information.



# Characteristics of Big Data

- **Volume** – 90% created in the last 2ys. 50% in Hadoop after 5ys
- **Velocity** – The speed of data generated, analyzed, moved
- **Variety** – Multiple data formats

### Structured Data

- Tables & well defined schemas (RDB)
- Regular structures

### Unstructured Data

- No specific data model (free text, emails, logs)
- Heterogeneous data (audio, video)

### Semi Structured Data

- Irregular structures (xml, json)
- Schemas are not mandatory

- **Value** – The knowledge gained by exploring data

## Two Inherent Characteristics



<http://www.it21learning.com>

# Two Inherent Characteristics

- Time-based
  - A piece of data is something known at a certain moment in time, and that time is an important element.
- Immutable
  - Because of its connection to a point in time, the truthfulness of the data does not change. We look at changes in big data as new entries, not updates of existing entries.

# Distributed Computing

- Traditional Distributed Computing:
  - Computation has been CPU bound;
    - Relatively small amounts of data
    - Significant amount of complex processing performed on the data
  - For decades, the primary push was to increase the computing power of a single machine.
  - Typically at compute time, data is copied over to the compute nodes

# New Distributed Computing

## - Hadoop

- Distribute Data Storage
  - Big Data not copied for processing
- Move Computation to the Data
  - Processing Big Data becomes possible:
- Big (hardware) cluster of commodity machines at lower cost;
- Algorithms (software) to allow parallel computing

## What is Hadoop?



<http://www.it21learning.com>

# What is Hadoop?

- Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs.
  - Framework for solving data-intensive processes
  - Designed to scale massively
  - Very fast for big jobs
  - Variety of processing engines, such as Tez, Spark and Storm
  - Designed for hardware and software failures

# Hadoop History

- Hadoop is based on work done by Google in the early 2000s
  - Specifically, on papers describing the Google File System (DFS) published in 2003, and MapReduce published in 2004.
- This work takes a radical new approach to the problem of Distributed computing
  - Meets all the requirements we have for reliability, scalability etc.
- Core concept: distribute the data as it is initially stored in the system.
  - Individual nodes can work on data local to those nodes
    - No data transfer over the network is required for initial processing

## Why Hadoop?

# Why Hadoop?

- Why to create and use Hadoop?
  - ❖ Cost Effective: Brings massively parallel computing to commodity servers.
    - ❖ “Shared Nothing” Architecture
  - ❖ Scalable: New nodes can be added as needed without requiring changes on the existing data distribution, nor any changes on the jobs and applications.
  - ❖ Fault Tolerant: When you lose a node, the system redirects work to another location of the data and continues processing without disruption.
  - ❖ Flexible: Hadoop is schema-less, and can absorb any type of data.

## Hadoop vs. RDBMS

# Hadoop vs. RDBMS

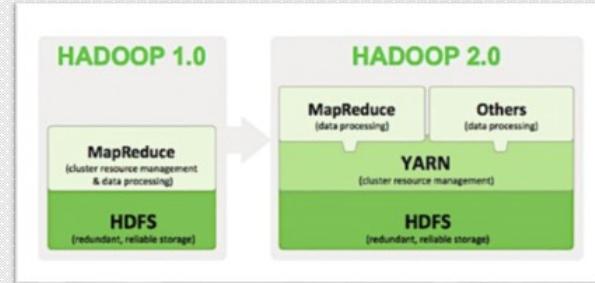
	Relational	Hadoop
Schema	Required on Write	Required on Read
Speed	Reads are fast	Writes are fast
Governance	Standards and structured	Loosely structured
Processing	Limited, no data processing	Processing coupled with data
Data Types	Structured	<ul style="list-style-type: none"><li>• Structured</li><li>• Semi-Structured</li><li>• Un-structured</li></ul>
Best Fit Use	<ul style="list-style-type: none"><li>• Interactive OLAP Analytics</li><li>• Complex ACID Transactions</li><li>• Operational Data Store</li></ul>	<ul style="list-style-type: none"><li>• Data Discovery</li><li>• Processing un-structured data</li><li>• Massive storage/processing</li></ul>

**Important:** Hadoop is not meant to replace relational database. Hadoop is for storing big data, which is often the type of data that you would otherwise not store in a database due to size or cost constraints.

Data governance (DG) refers to the overall management of the availability, usability, integrity, and security of the data employed in an enterprise.

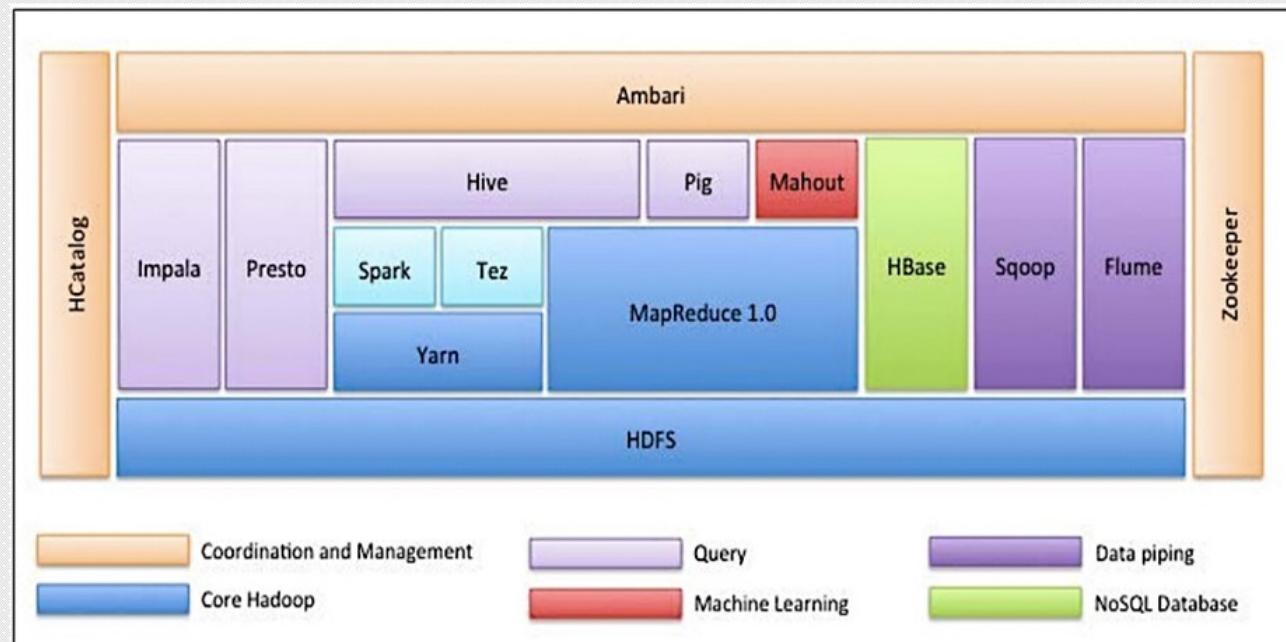
# Hadoop Architecture

- **Hadoop Common:**
  - Common Utilities that supports all other modules.
- **Hadoop Distributed File System (HDFS):**
  - File system that spans all the nodes in a Hadoop cluster for data storage.
  - Links the file systems on local nodes to make them into one big file system.
- **Hadoop MapReduce:**
  - Core computing framework available since Hadoop 1.x
- **Hadoop YARN:**
  - New distributed processing framework on Hadoop 2.x.
  - Addresses multiple limitations of MR 1.0
- **Hadoop Ecosystem:**
  - Open-source Apache projects



## Hadoop Ecosystem

# Hadoop Ecosystem



# Hadoop Ecosystem (Con't)

ZooKeeper

- Resources management



Mahout

- Algorithms for Machine Learning



Flume

- Streaming of Data (pull real time data from HDFS)



Sqoop

- Pull/Push data from/to RDBMS



Avro

- Data in JSON format



Pig

- MR abstraction via functional programming



Hive

- MR abstraction via SQL like data support

MapReduce

- Distributed computing

HBase and HDFS

- Distributed Database and File System

# Hadoop (HDFS) Roles

- **NameNode (NN)**

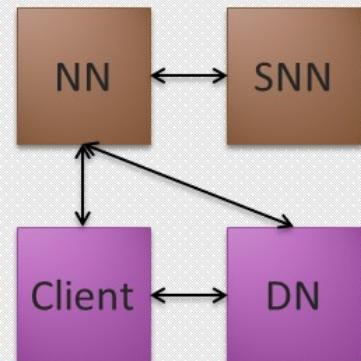
- Manages the File System's namespace/meta-data/file blocks
- Runs on 1 machine to several machines

- **Secondary NameNode (SNN)**

- Performs house keeping work so NameNode doesn't
- Requires similar hardware as NameNode machine
- Not used for high-availability – not a backup for NameNode

- **DataNode (DN)**

- Stores and retrieves data blocks
- Reports to NameNode
- Runs on many machines



# Hadoop (HDFS) Components

- **Client**

- User/App interface to interact with cluster, DN

- **Namespace**

- Files/Directories - Same to the regular file systems split into blocks

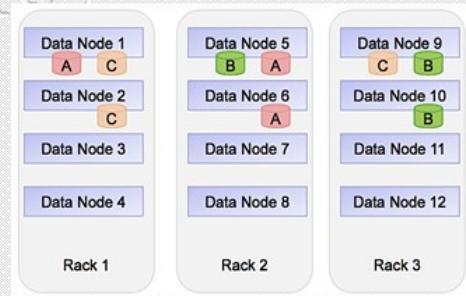
- **Blocks**

- Default: 64M (v1); 128M (v2)
- Blocks meta data kept in NN
  - Small files issue

- **Block Storage:**

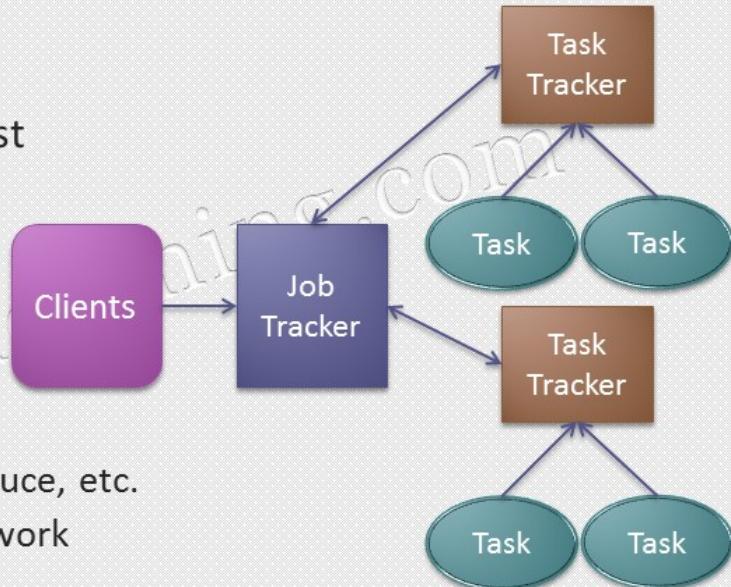
- **Replications**

- Default 3 and rebalanced for new added nodes
- 1st replica on the local, 2nd on the local but different node, 3rd on the different rack



# Hadoop v1 (MapReduce) Engine

- **Job Tracker - JT**
  - Run in name node
  - Accept client job request
  - Submit work to TT
  - Peer of NN
- **Task Tracker - TT**
  - Accept tasks from JT
  - Perform map, shuffle, reduce, etc.
  - Seek for available slot to work
  - Send heartbeats to JT
  - Peer of DN



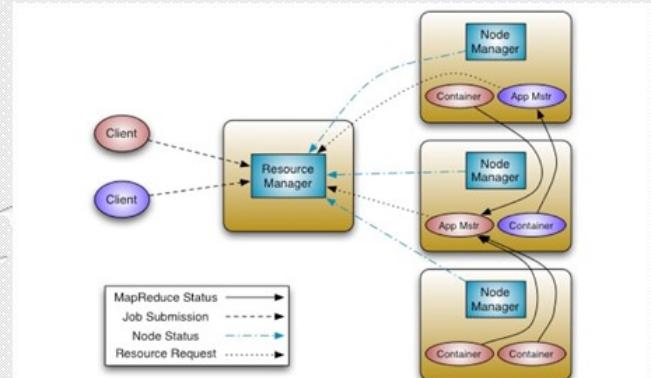
# Hadoop v2.0 YARN

- **Limitation of MapReduce**

- Single computing engine
  - MapReduce
- Lack of sense of system resource
- Too much overhead on JT

- **The changes of YARN**

- Support more computing engine
- Compatible with MapReduce
- Better resource management
- Split JT's resource management to ResourceManager
- Split JT's job scheduling and monitoring to ApplicationMaster
- NodeManager as alternatives of TT to launch and monitor containers

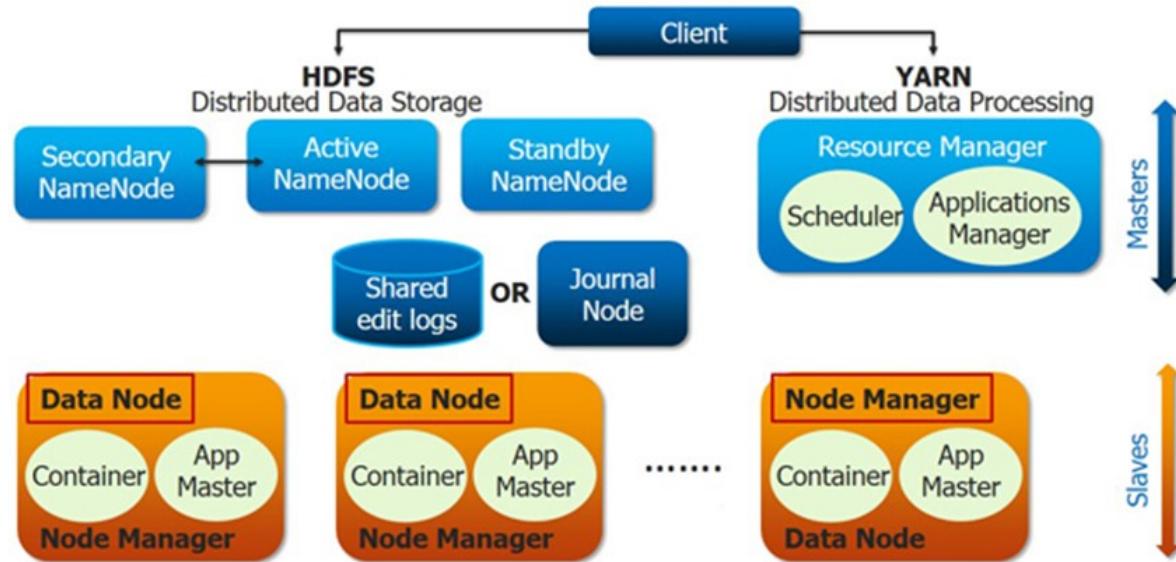


YARN stands for Yet Another Resource Negotiator. Let's walk through an application execution sequence (steps are illustrated in the diagram): A client program submits the application, including the necessary specifications to launch the application-specific ApplicationMaster itself. The ResourceManager assumes the responsibility to negotiate a specified container in which to start the ApplicationMaster and then launches the ApplicationMaster. The ApplicationMaster, on boot-up, registers with the ResourceManager – the registration allows the client program to query the ResourceManager for details, which allow it to directly communicate with its own ApplicationMaster. During normal operation the ApplicationMaster negotiates appropriate resource containers via the resource-request protocol. On successful container allocations, the ApplicationMaster launches the container by providing the container launch specification to the NodeManager. The launch specification, typically, includes the necessary information to allow the container to communicate with the ApplicationMaster itself. The application code executing within the container then provides necessary information (progress, status etc.) to its ApplicationMaster via an application-specific protocol. During the application execution, the client that submitted the program communicates directly with the ApplicationMaster to get status, progress updates etc. via an application-specific protocol. Once the application is complete, and all necessary work has been finished, the ApplicationMaster deregisters with the ResourceManager and shuts down, allowing its own container to be repurposed.

## Hadoop v2.0 YARN Architecture

# Hadoop v2.0 YARN Architecture

## Apache Hadoop 2.0 and YARN



As the data Operating System for Hadoop 2.0, YARN allows multiple applications to co-exist in the same shared cluster. YARN also takes into consideration of Scalability, High utilization, support multiple programming models, flexible resource model, Security, Reliability, and compatibility. YARN separates the JobTracker (Hadoop 1.x) into a cluster level ResourceManager (RM) and application specific ApplicationMaster (AM). YARN architecture follows master-slave architectural model in which ResourceManager as a master and node specific slave NodeManager (NM). NodeManager replaces the TaskTracker in Hadoop 1.x. A Client submits the application to the ResourceManager. In above diagram client 1 submit a MapReduce Request and client 2 submit Shell Script Request. ResourceManager allocates a container to start up the ApplicationMaster for each application request submitted by client – one ApplicationMaster for Shell Script and one for MapReduce application. After starting a ApplicationMaster, it registers application with ResourceManager. After startup of ApplicationMaster, it negotiates with ResourceManager for appropriate resources as per the application requirement. After resource allocation from ResourceManager, ApplicationMaster requests NodeManager to launch the containers allocated by ResourceManager. On successful launching of containers, application code executes within the container. ApplicationMaster reports back to ResourceManager with execution status of application. During execution of the application, client can request ApplicationMaster or directly with ResourceManager for application status, progress updates etc. On application execution process completion, ApplicationMaster request ResourceManager to unregister and shut down its own container process.

# How is Hadoop High Availability

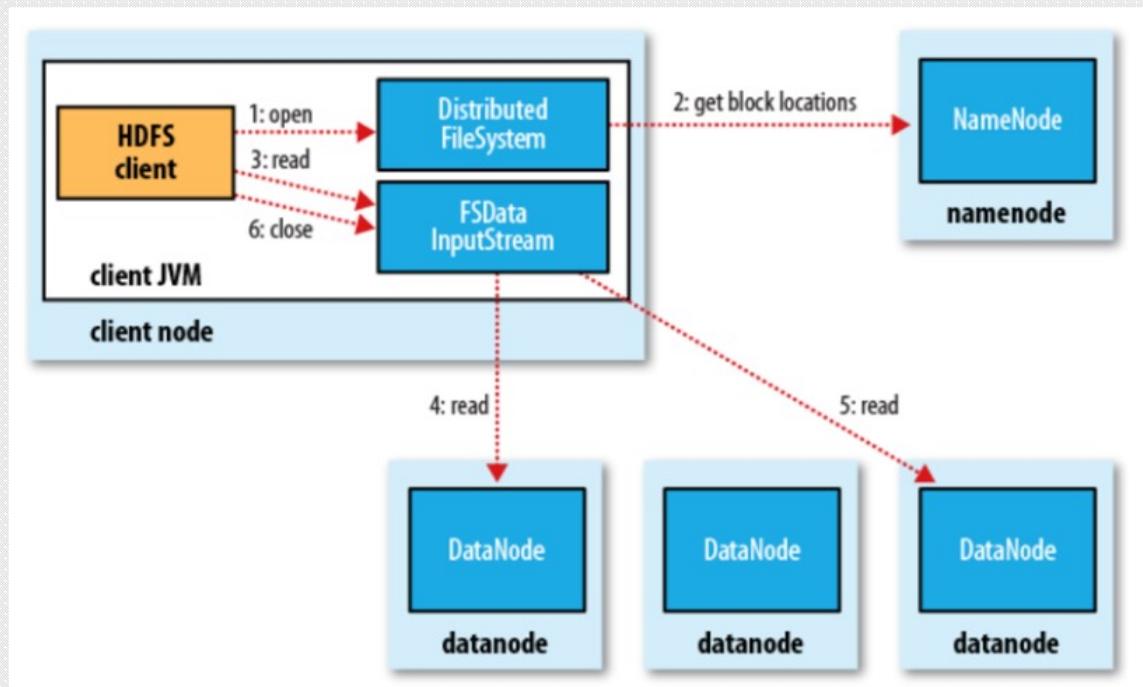
In Hadoop v1, NN has single point of failure (SPOF) issues

- **What are the solutions?**

1. HDFS Federation by Partitioning the file system namespace over multiple separated NameNodes
2. Active and Standby NNs share the storage for edit logs;
3. DNs are registered with all NNs, and send block location information and heartbeats to all NNs.

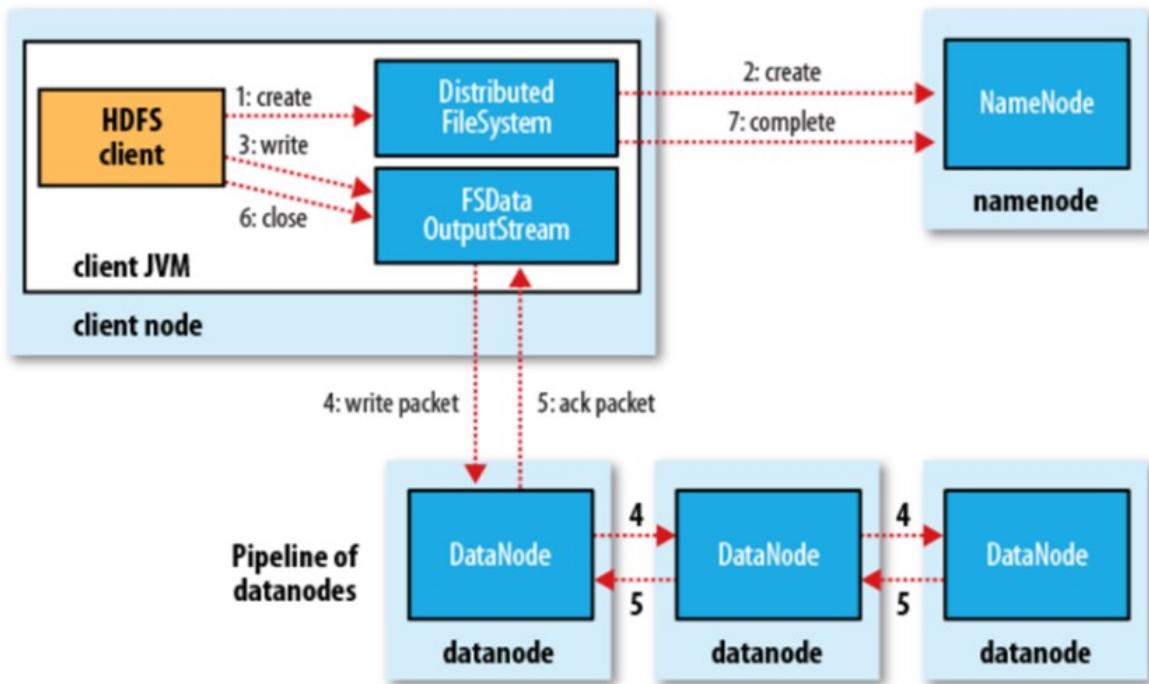
## HDFS – Read a File

# HDFS - Read a File



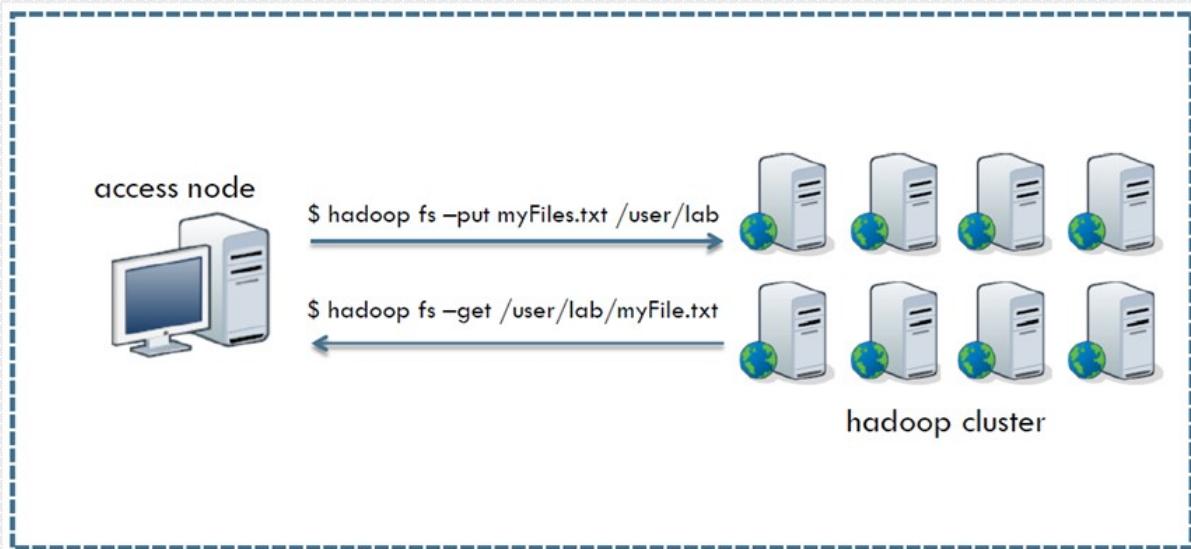
## HDFS – Write a File

# HDFS - Write a File



# HDFS - CLI (Command Line)

- Users typically access HDFS via **hdfs dfs** or **hadoop fs** command
  - *hadoopfs is deprecated.*



## hdfs dfs Command Examples



<http://www.it21learning.com>

# hdfs dfs Command Examples

Command	Comments
hdfs dfs –mkdir /user/data	Create a directory on hdfs
hdfs dfs –ls /user/data	List all files in the directory
hdfs dfs –put log.txt /user/data	Put a file from local file system to hdfs
hdfs dfs –cat /user/data/log.txt   head	Display the content of a file
hdfs dfs –get /user/data/log.txt	Get a file from hdfs to local file system
hdfs dfs –rmdir /user/data/log.txt	Remove a file from a hdfs directory
hdfs dfs –copyFromLocal t.txt hdfs://localhost/user/data	
hdfs dfs –copyToLocal /user/data/t.txt tt.txt	

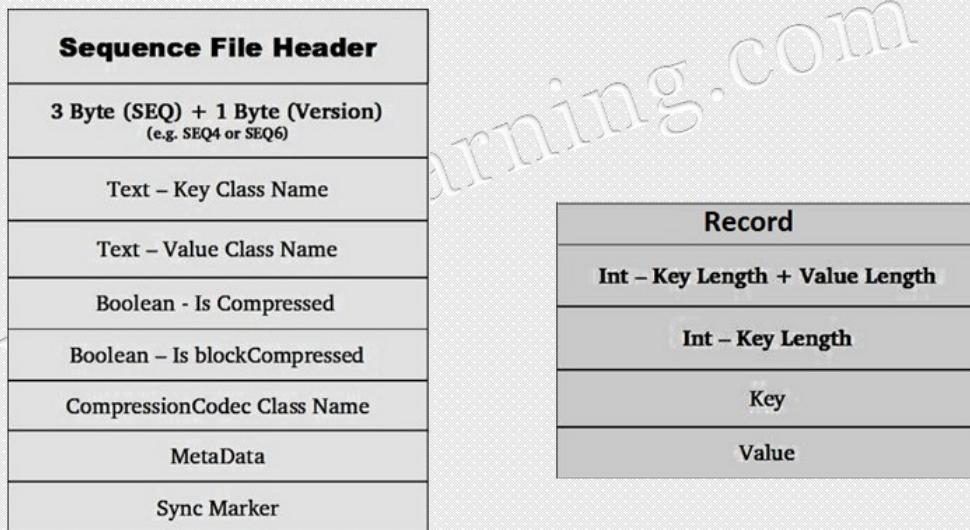
## Setup Environment, and DEMO

# HDFS File Types

- HDFS supports storing all types of files in different formats:
  - Text & Binary
  - Un-compressed & Compressed
  - However, files need to be splittable in order to support the best of Map-Reduce processing.
  - *Splittable doesn't mean a file has to be cut into blocks at the boundary of records, but the file can be read by records.*

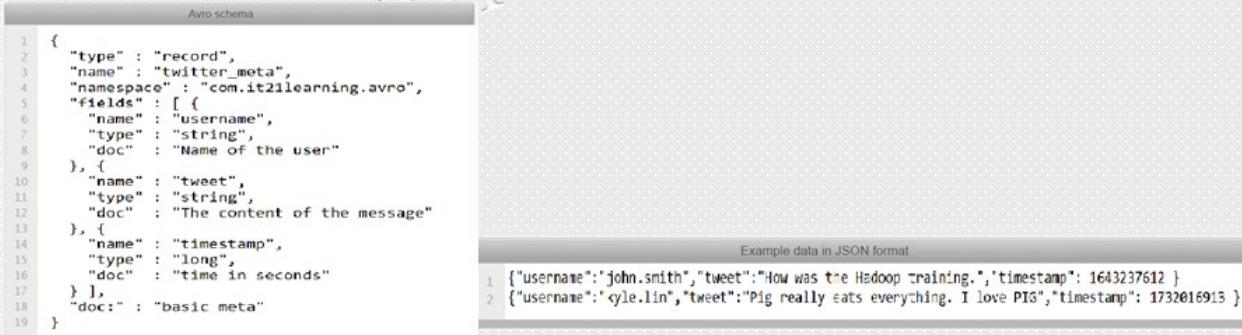
# SequenceFile

- A SequenceFile is just a flat file consisting of binary key/value pairs



# Avro File

- Avro stores both the data definition and the data together in one message
  - The data definition is stored in JSON format
  - The data is stored in binary format
  - The Avro markers are used to splitting large data sets into subsets suitable for MapReduce processing.



The screenshot shows a window titled "Avro schema". On the left, there is a code editor containing the following JSON schema:

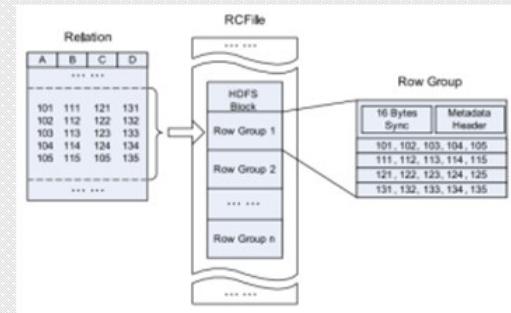
```
1 {
2   "type" : "record",
3   "name" : "twitter_meta",
4   "namespace" : "com.it21learning.avro",
5   "fields" : [ {
6     "name" : "username",
7     "type" : "string",
8     "doc" : "Name of the user"
9   }, {
10    "name" : "tweet",
11    "type" : "string",
12    "doc" : "The content of the message"
13  }, {
14    "name" : "timestamp",
15    "type" : "long",
16    "doc" : "time in seconds"
17  },
18  "doc:" : "basic meta"
19 } 
```

On the right, under the heading "Example data in JSON format.", there is a code editor showing two JSON objects:

```
1 {"username": "john.smith", "tweet": "How was the Hadoop training.", "timestamp": 1643237612 }
2 {"username": "kyle.lin", "tweet": "Pig really eats everything. I love PIG", "timestamp": 1732016913 } 
```

# RCFile - Record Columnar File

- RCFile has been adopted by the Hive and Pig projects as the core format for table like data storage.
  - The RCFile splits data horizontally into row groups.
    - Rows 1 to 100 are stored in one group and rows 101 to 200 in the next and so on.
  - The RCFile saves each row group data in a columnar format.
    - Instead of storing row one then row two, it stores column one across all rows then column two across all rows and so on.
- Benefit
  - Each processing node reads only the columns relevant to a query from a file and skips irrelevant ones



# ORC File & Parquet File

- ORC File – Optimized Row Columnar File stores collections of rows in one file and within the collection the row data is stored in a columnar format.
  - Use specific encoders for different column data types to improve storage;
  - Introduce a lightweight indexing that enables skipping of complete blocks of rows that do not match a query.
- Parquet is a columnar storage format for Hadoop that provides efficient encoding and compression schemes.

## Read a File from HDFS

```
public class Cat
{
    public static void main (String [] args) throws Exception
    {
        try
        {
            Path pt = new Path("hdfs://localhost:1109/user/data/t.txt");
            FileSystem fs = FileSystem.get(new Configuration());
            BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(pt)));
            String line=br.readLine();
            while (line != null)
            {
                System.out.println(line);
                line=br.readLine();
            }
        }
        catch(Exception e){ }
    }
}
```

## Write a File to HDFS

# Write a File to HDFS

- ```
Path newFolderPath= new Path("/user/data");
FileSystem hdfs =FileSystem.get(new Configuration());
if(hdfs.exists(newFolderPath))
{
    hdfs.delete(newFolderPath, true);
}
hdfs.mkdirs(newFolderPath);
Path newPath=new Path("/user/data/newFile.txt");
StringBuilder sb=new StringBuilder();
for(int i=1;i<=5;i++)
{
    sb.append("Data");
    sb.append(i);
    sb.append("\n");
}
byte[] byt=sb.toString().getBytes();
FSDataOutputStream fsOutStream = hdfs.create(newFilePath);
fsOutStream.write(byt);
fsOutStream.close();
```

## Any Question?

# Any Question?



## Programming MapReduce

# Programming MapReduce

Wayne Shen



<http://www.it21learning.com>

# What is MapReduce?

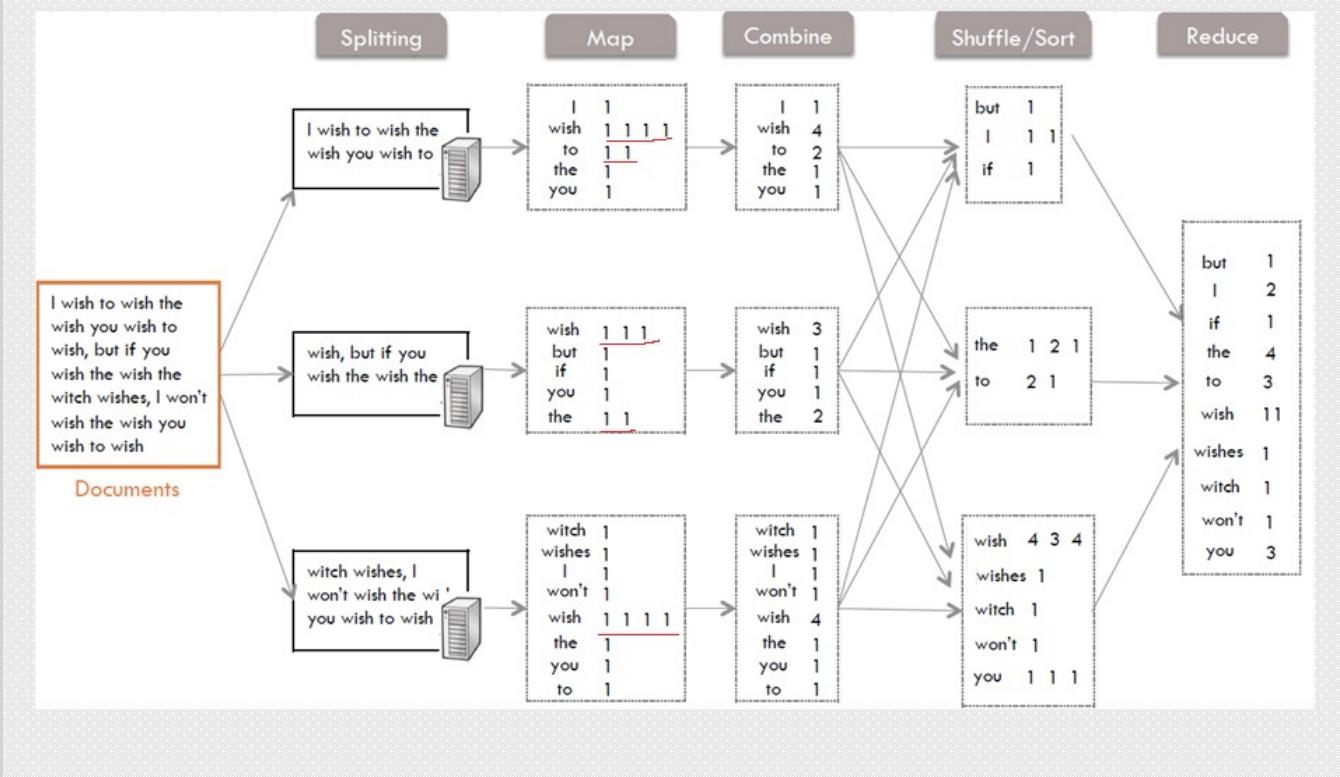
- Origin from Google, [OSDI'04]
- MapReduce is a computing model that decomposes large data manipulation jobs into individual tasks that can be executed in parallel across a cluster of servers.
- For large-scale data processing
  - Each node processes data stored on that node
- Each MapReduce job consists of two phases
  - Map
  - Reduce

# Automatic MapReduce Computing

- MapReduce Computing is paralleled and distributed automatically;
- Developers only need to focus on implementing the map and reduce functions;
- M/R is written in Java, but also supports streaming with Python.

## MapReduce – Word Count

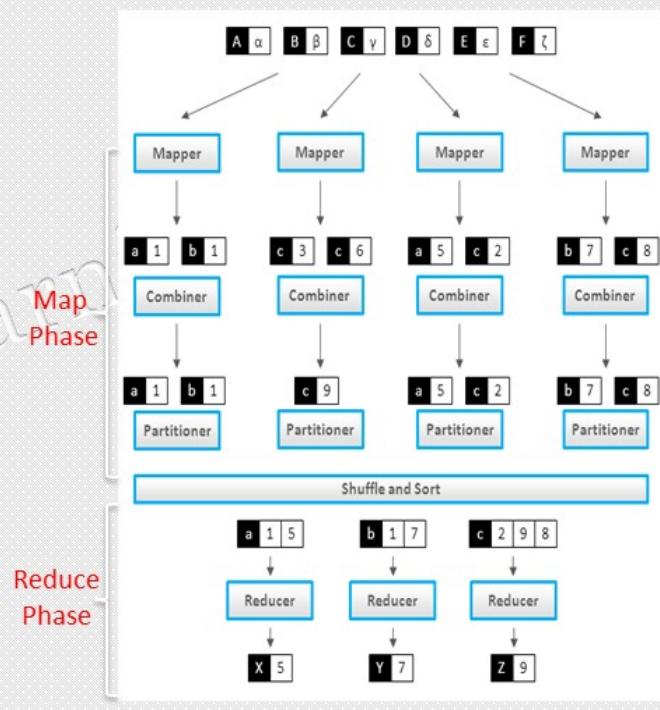
# MapReduce - Word Count



# MapReduce Framework

*Definition:*

- map:  $(K_1, V_1) \rightarrow \text{list } (K_2, V_2)$
- reduce:  $(K_2, \text{list}(V_2)) \rightarrow \text{list } (K_3, V_3)$
- Mapper (map)
- Combiner (optional)  
Can only be used when  
commutative( $a.b = b.a$ ) and associative  
 $\{a.(b.c) = (a.b).c\}$
- Partitioner  
Default is the HashPartitioner that  
performs a modulo against the  
numOfPartitions to return the partition  
number
- Shuffle and Sort
- Reducer (reduce)

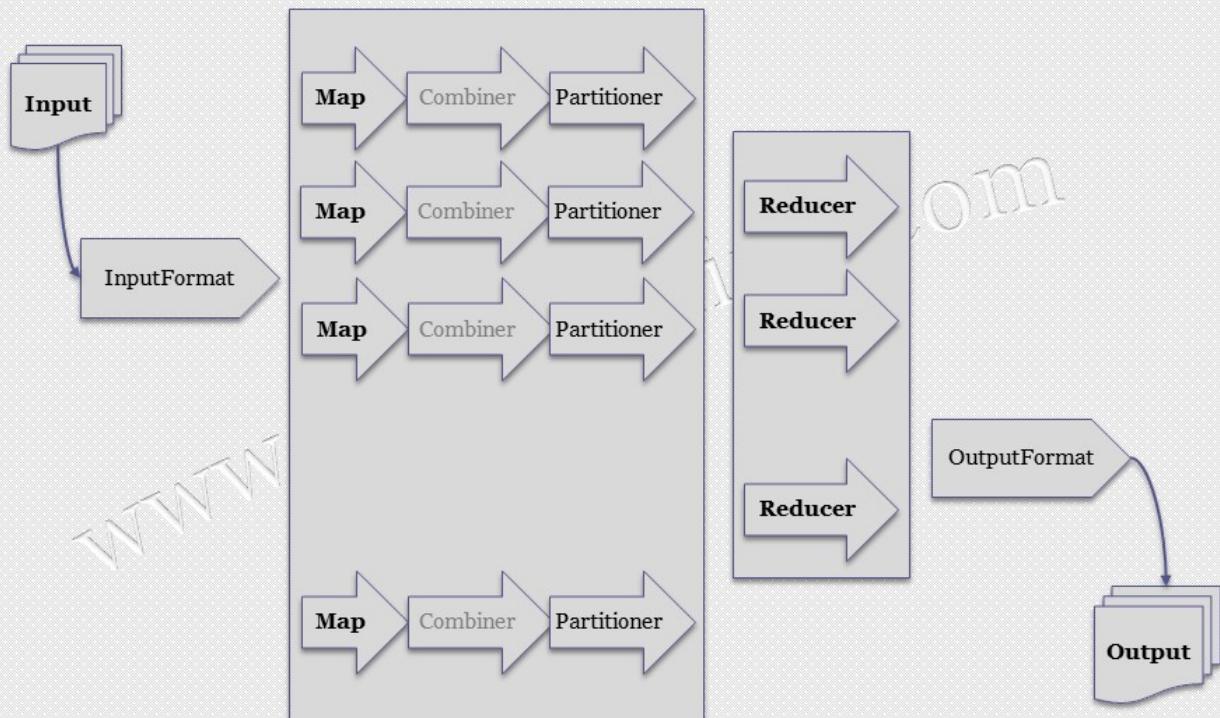


# What is InputSplit?

- InputSplit represents the data to be processed by an individual Mapper.
  - Block is the physical representation of data. Split is the logical representation of data present in Block.
- InputSplit respects logical record boundary.
  - During MapReduce execution Hadoop scans through the blocks and create InputSplits and each InputSplit will be assigned to individual mappers for processing.

## MapReduce Programming Model

# MapReduce Programming Model



# Key & Value Types

- Key & Value Types must be serializable
  - Utilize Hadoop's Writable-based serialization mechanism for writing data in and out of network;
  - Optimized for network serialization;
  - Has the ability to reduce the object-creation overhead by reusing the Writeable objects;
  - A set of basic types is provided
    - IntWritable, LongWritable, FloatWritable, DoubleWritable, BooleanWritable, Text, NullWritable, etc
    - Easy to create custom Key-Value types.

# Key & Value Types (Con't)

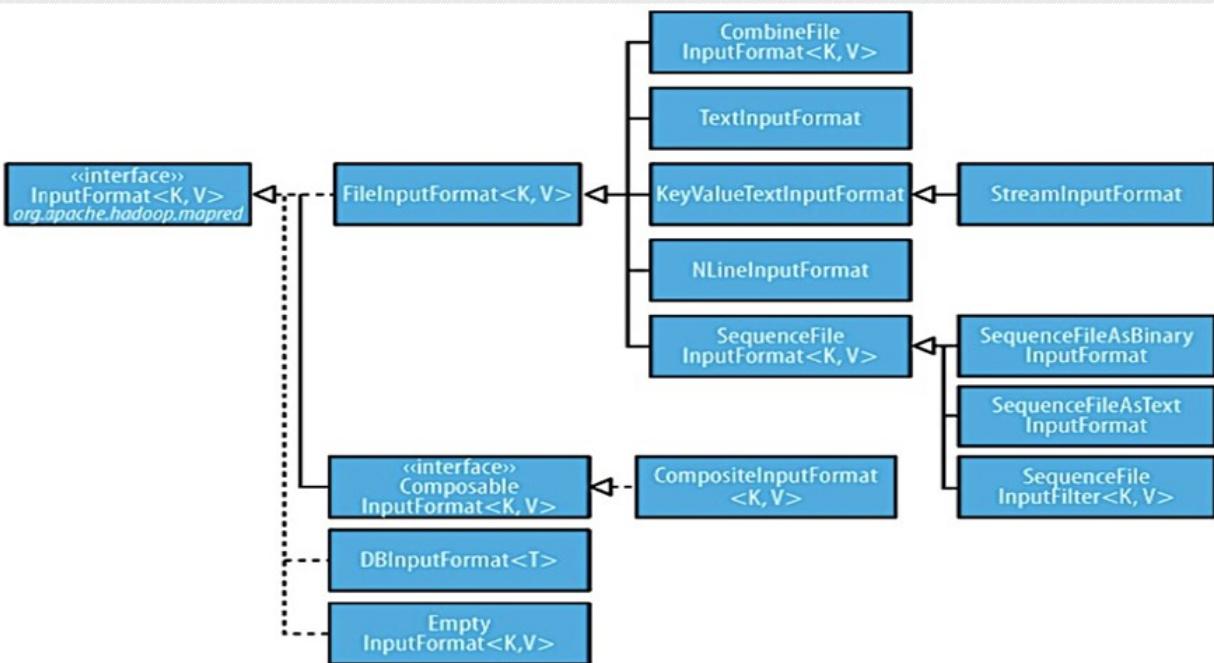
- Values must extend Writable interface;
  - `void readFields(DataInput in)` - deserialize the fields
  - `void write(DataOutput out)` – serialize the fields
- Keys must implement WritableComparable interface
  - Keys are sorted prior reduce phase;
  - `void readFields(DataInput in)` - deserialize the fields;
  - `void write(DataOutput out)` – serialize the fields;
  - `int compareTo( T t)` – compares this object with the specified object;
  - `int hashCode()` – used by partitioner to decide which reducer to send the key-value pairs.

# InputFormat Interface

- The *InputFormat* defines how to read data from an input into Mapper instances.
  - *InputSplit[] getSplits(JobConf job, int numSplits) throws IOException*
    - Each InputSplit is then assigned to an individual Mapper for processing.
  - *RecordReader<K,V> getRecordReader(InputSplit split, JobConf job, Reporter reporter) throws IOException*
    - Get the RecordReader for the given InputSplit.
    - It is the responsibility of the RecordReader to respect record boundaries while processing the logical split to present a record-oriented view to the individual task.

## Input Formats

# Input Formats



# Mapper Class

- `protected void setup(org.apache.hadoop.mapreduce.Mapper.Context context)`  
throws IOException, InterruptedException
  - `Called once at the beginning of the task.`
- `protected void map(KEY key, VALUE value, Context context)`  
throws IOException, InterruptedException
  - `Called once for each key/value pair in the input split. Most applications should override this, but the default is the identity function.`
- `protected void cleanup(Context context)` throws IOException,  
InterruptedException
  - `Called once at the end of the task.`
- `public void run(.Context context)`  
throws IOException,  
InterruptedException
  - `Expert users can override this method for more complete control over the execution of the Mapper.`

```
public class WCMapper extends Mapper<LongWritable, Text, Text, IntWritable>
{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context ctx)
        throws IOException, InterruptedException
    {
        StringTokenizer itr = new StringTokenizer( value.toString() );
        while ( itr.hasMoreTokens() )
        {
            word.set( itr.nextToken() );
            ctx.write( word, one );
        }
    }
}
```

# Combiner - Optional

1. Combiners are an optimization in MapReduce that allow for local aggregation before the shuffle and sort phase.
  2. Combiners can only be used on the functions that are
    - Commutative -  $a.b = b.a$
    - Associative -  $a.(b.c) = (a.b).c$
- Combiner has the same interface as Reducer, so in most case Reducers can be used as Combiners as long as #2 satisfied.
    - `job.setCombinerClass( WCReducer.class );`

# Partitioner Class

- When Mapper output is collected it is partitioned, which means that it will be written to the output specified by the Partitioner.
- Partitioners are responsible for dividing up the intermediate key space and assigning intermediate key-value pairs to reducers.
- The default Partitioner involves computing the hash value of the key and then taking the mod of that value with the number of reducers.

## Reducer Class

# Reducer Class

- `protected void setup(org.apache.hadoop.mapreduce.Mapper.Context context)`  
    throws IOException, InterruptedException
  - **Called once at the beginning of the task.**
- `protected void reduce(KEY key, Iterable<VALUE> values, Context context)`  
    throws IOException, InterruptedException
  - **Called once for each key. Most applications should override this, but the default is the identity function.**
- `protected void cleanup(Context context)` throws IOException,  
    InterruptedException
  - **Called once at the end of the task.**
- `public void run(.Context context)`  
    throws IOException,  
    InterruptedException
  - **Advanced application writers can use the run method to control how the reduce task works.**

```
public class WCReducer
    extends Reducer<Text, IntWritable, Text, IntWritable>
{
    private IntWritable result = new IntWritable();

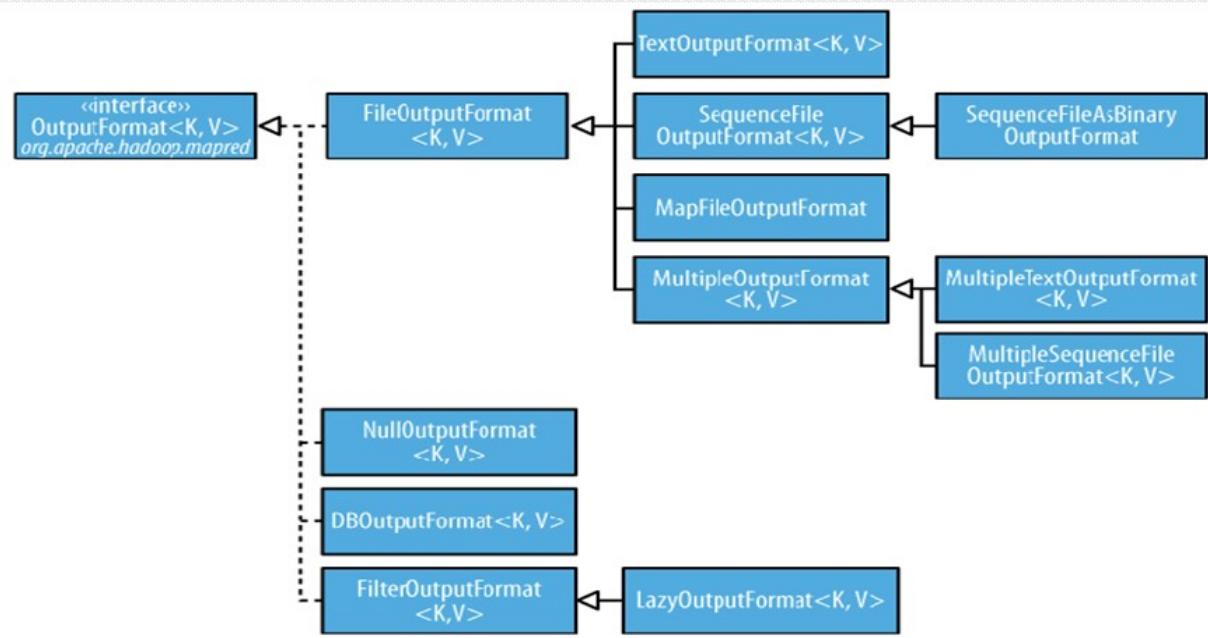
    public void reduce(Text key, Iterable<IntWritable> values, Context ctx)
        throws IOException, InterruptedException
    {
        int sum = 0;
        for ( IntWritable value : values )
        {
            sum += value.get();
        }
        result.set( sum );
        ctx.write( key, result );
    }
}
```

# OutputFormat Interface

- The *OutputFormat* defines how to write data to an output from Reducer instances.
  - `RecordWriter<K,V> getRecordWriter(FileSystem ignored, JobConf job, String name, Progressable progress)`  
`throws IOException`
    - Get the RecordWriter for the given job.
  - `void checkOutputSpecs(FileSystem ignored, JobConf job)`  
`throws IOException`
    - Check for validity of the output-specification for the job.
    - This is to validate the output specification for the job when it is a job is submitted. Typically checks that it does not already exist, throwing an exception when it already exists, so that output is not overwritten.

## Output Formats

# Output Formats



## Wire a M/R Job

# Wire a M/R Job

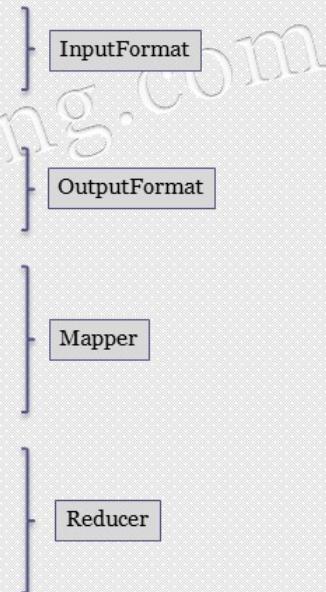
```
Job job = Job.getInstance(getConf(), "WordCountMR" );
job.setJarByClass( getClass() );

//input path
FileInputFormat.addInputPath(job, new Path(args[0]));
//input format
job.setInputFormatClass( TextInputFormat.class );

//output path
FileOutputFormat.setOutputPath(job, new Path(args[1]));
//output format
job.setOutputFormatClass( TextOutputFormat.class );

//mapper class
job.setMapperClass( WCMapper.class );
//set map output key
job.setMapOutputKeyClass( Text.class );
//map output value
job.setMapOutputValueClass( IntWritable.class );

//reducer class
job.setReducerClass( WCReducer.class );
//output key
job.setOutputKeyClass( Text.class );
//output value class
job.setOutputValueClass( IntWritable.class );
```



## Execute the M/R Job



<http://www.it21learning.com>

# Execute the M/R Job

- hadoop jar WCMR.jar /user/data /user/out
- Set number of mapper
  - java WCMR.jar /user/data /user/out -D mapred.map.tasks=5
- Set number of reducer
  - hadoop jar WCMR.jar /user/data /user/out -D mapred.reduce.tasks=2
  - Job.setNumReduceTasks( 3 );
- Debug M/R:
  - External JARs:
    - /usr/lib/hadoop/client-0.20 (all jars)
    - /usr/lib/hadoop (hadoop-annotations.jar, hadoop-auth.jar, hadoop-common.jar)
    - /usr/lib/hadoop/lib (commons-httpclient-3.1.jar)
    - Debug Arguments: data.txt /home/usr/output

## Pass Parameters

# Pass Parameters

```
//during start-up
Configuration cfg = new Configuration();
cfg.set("test", "123");

Job job = new Job(cfg);
```

```
//inside mapper and/or reducer
Configuration cfg = context.getConfiguration();
String value = cfg.get("test");
```

# Distributed Cache

- Distributed Cache helps in sending read-only files to task nodes in a Hadoop cluster when executing a M/R job.
  - The files could be look-up tables, text-files, JARs, etc;
  - The size of the files should be “small”;
  - The files are only copied once per job.

```
Job job = new Job();
//...
job.addCacheFile(new Path(filename).toUri());
job.addCacheFile(new URI("/user/data/customer_types.json#customer_type"));
```

```
URI[] files = context.getCacheFiles();
Path filename = new Path(files[0])
File customerType = new File("./customer_type");
```

- The **DistributedCache** class is deprecated.

# Hadoop Streaming

- The Hadoop streaming utility allows any script or executable to work as Mapper/Reducer provided they can work with stdin and stdout.

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar \
  -input myInputDirs \
  -output myOutputDir \
  -mapper /bin/cat \
  -reducer /bin/wc
```

- Both the mapper and the reducer are executables that read the input from stdin (line by line) and emit the output to stdout.

# Speculative Execution - SE

- Problem:
  - ✧ So when 99 of 100 map tasks are already complete, the system is still waiting for the final map task to check in, which takes much longer than all the other nodes.
- Resolution: Using SE
  - ✧ The same input can be processed multiple times in parallel, to exploit differences in machine capabilities.
  - ✧ Schedule redundant copies of the slow tasks across several nodes which do not have other work to perform.
  - ✧ Whichever copy of a task finishes first becomes the definitive copy.

## Any Question?

# Any Question?



# Mastering Apache Hive

# Mastering Apache Hive

Will Du



<http://www.it21learning.com>

# What do we cover today?

- Hive Overview and Highlight
- Hive Architecture
- Hive Interface
- Hive Data Type and Structure
- Hive database, tables, partitions, buckets, views
- Hive select, join, union
- Hive load, insert, import, export
- Hive order, sort

# Apache Hive Overview

- Data Warehousing Solution built on top of Hadoop
- Provides SQL-like query language named HiveQL - HQL, minimal learning curve
- Early Hive development work started at Facebook in 2007
- Today Hive is an top Apache project under Hadoop at [hive.apache.org](http://hive.apache.org)



## Milestone of Hive



<http://www.it21learning.com>

# Milestone of Hive

AUG 2007 – Journey Start

MAY 2013 – Hive 0.11.0 as Stinger Phase 1 - ORC, HiveServer2

OCT 2013 – Hive 0.12.0 as Stinger Phase 2 - ORC improvement

APR 2014 – Hive 0.13.0 as Stinger Phase 3 - vectorized query engine and Tez

NOV 2014 – Hive 0.14.0 as Stinger.next Phase 1- Cost-based optimizer

FEB 2015 – Hive 1.0.0

Most BI and ETL tools leverage Hive as connector

**The Stinger Initiative** - Making Apache Hive **100** Times Faster

## Highlight of Hive



<http://www.it21learning.com>

# Highlight of Hive

- Hive provides a simpler query model with less coding than MapReduce
- HQL and SQL have similar syntax
- Hive provides lots of functions that lead to easier analytics usage
- Hive supports running on different computing frameworks
- Hive supports ad hoc querying data on HDFS and HBase
- Hive supports user-defined functions, scripts, and customized I/O format to extend functionality
- Matured JDBC and ODBC drivers allow many applications to pull Hive data for seamless reporting
- Hive has a well-defined architecture for metadata management, authentication, optimizations
- There is a big community of practitioners and developers working on and using Hive

## Hive vs. MapReduce – Word Count Again

## Hive vs. MapReduce – Word Count Again

```
1 package org.apache.hadoop.examples;
2 import java.io.IOException;
3 import java.util.StringTokenizer;
4 import org.apache.hadoop.conf.Configuration;
5 import org.apache.hadoop.io.IntWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Job;
8 import org.apache.hadoop.mapreduce.Mapper;
9 import org.apache.hadoop.mapreduce.Reducer;
10 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.GenericOptionsParser;
13
14 public class WordCount {
15
16     public static class TokenizerMapper
17         extends Mapper<Object, Text, IntWritable> {
18
19         private final static IntWritable one = new IntWritable();
20         private Text word = new Text();
21
22         public void map(Object key, Text value, Context context
23                         ) throws IOException, InterruptedException {
24             StringTokenizer itr = new StringTokenizer(value.toString());
25             while (itr.hasMoreTokens()) {
26                 word.set(itr.nextToken());
27                 context.write(word, one);
28             }
29         }
30     }
31
32     public static class IntSumReducer
33         extends Reducer<Text, IntWritable, Text, IntWritable> {
34         private IntWritable result = new IntWritable();
35
36         public void reduce(Text key, Iterable<IntWritable> values,
37                            Context context
38                            ) throws IOException, InterruptedException {
39             int sum = 0;
40             for (IntWritable val : values) {
41                 sum += val.get();
42             }
43             result.set(sum);
44             context.write(key, result);
45         }
46     }
47
48     public static void main(String[] args) throws Exception {
49         Configuration conf = new Configuration();
50         String[] otherargs = new GenericOptionsParser(conf, args).getRemainingArgs();
51         if (otherargs.length > 2) {
52             System.err.println("Usage: wordcount <in> <out>");
53             System.exit(1);
54         }
55         Job job = new Job(conf, "word count");
56         job.setJarByClass(WordCount.class);
57         job.setMapperClass(TokenizerMapper.class);
58         job.setCombinerClass(IntSumReducer.class);
59         job.setReducerClass(IntSumReducer.class);
60         job.setOutputKeyClass(Text.class);
61         job.setOutputValueClass(IntWritable.class);
62         FileInputFormat.setInputPaths(job, new Path(otherargs[0]));
63         FileOutputFormat.setOutputPath(job, new Path(otherargs[1]));
64         System.exit(job.waitForCompletion(true) ? 0 : 1);
65     }
66 }
```

67 Lin

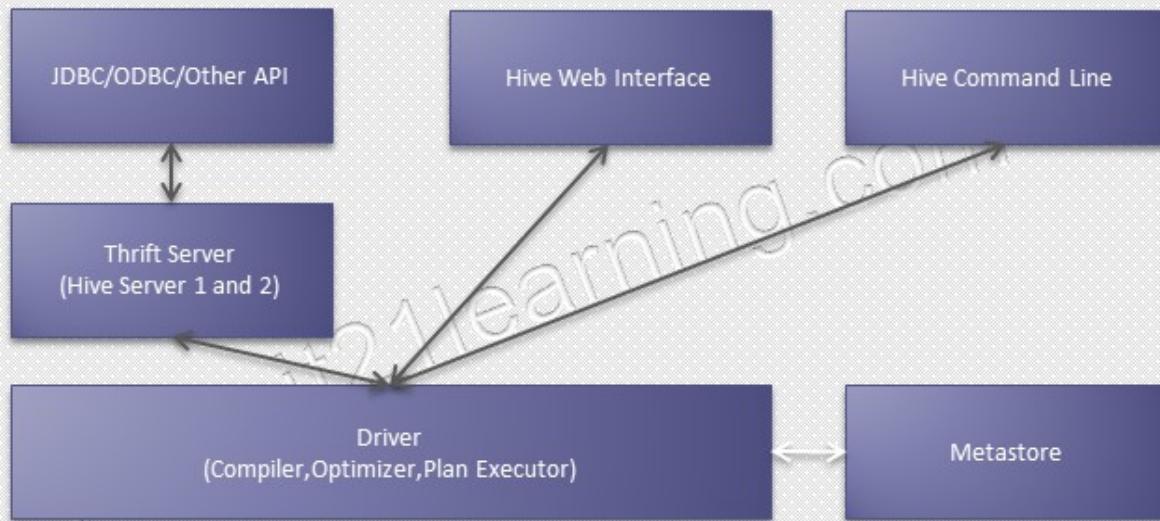
```
1 —Define metadata for the source
2 CREATE EXTERNAL TABLE lines(line STRING);
3 LOAD DATA INPATH 'book' OVERWRITE INTO TABLE lines;
4
5 — word count
6 SELECT word, count(*) as world_count
7 FROM lines
8 LATERAL VIEW explode(split(text, ' ')) t1 as word
9 GROUP BY word;
```

# Hive Metadata

- To support features like schema(s) and data partitioning Hive keeps its metadata in a Relational Database
- By default, Hive is Packaged with Derby, a lightweight embedded SQL DB
  - ❖ Default Derby based is good for evaluation and testing
  - ❖ Schema is not shared between users as each user has their own instance of embedded Derby
  - ❖ Stored in `metastore_db` directory which resides in the directory that hive was started from
- Can easily switch another SQL installation such as MySQL
- HCatalog as part of Hive exposes Hive metadata to other ecosystem

## Hive Architecture

# Hive Architecture



# Hive Interface – Command Line

- CML and Beeline
- Command Mode

| Purpose           | HiveServer2 Beeline                                                                                | HiveServer1 CLI                                                                              |
|-------------------|----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| Server connection | <code>beeline -u &lt;jdbcurl&gt; -n &lt;username&gt; -p &lt;password&gt;</code>                    | <code>hive -h &lt;hostname&gt; -p &lt;port&gt;</code>                                        |
| Help              | <code>beeline -h or beeline --help</code>                                                          | <code>hive -H</code>                                                                         |
| Run query         | <code>beeline -e &lt;query in quotes&gt;</code><br><code>beeline -f &lt;query file name&gt;</code> | <code>hive -e &lt;query in quotes&gt;</code><br><code>hive -f &lt;query file name&gt;</code> |
| Define variable   | <code>beeline --hivevar key=value.</code><br>This is available after Hive 0.13.0.                  | <code>hive --hivevar key=value</code>                                                        |

You need to use the special `hiveconf` for variable substitution. e.g. `hive> set CURRENT_DATE='2015-06-26'; hive> select * from foo where day >= '${hiveconf:CURRENT_DATE}'` similarly, you could pass on command line: `% hive -hiveconf CURRENT_DATE='2015-06-26' -f test.hql` Note that there are env and system variables as well, so you can reference  `${env:USER}` for example. To see all the available variables, from the command line, run `% hive -e 'set;'` or from the hive prompt, run `hive> set;`

# Hive Interface – Command Line Cont.

- Interactive Mode

| Purpose            | HiveServer2 Beeline                                            | HiveServer1 CLI                        |
|--------------------|----------------------------------------------------------------|----------------------------------------|
| Enter mode         | <code>beeline</code>                                           | <code>hive</code>                      |
| Connect            | <code>!connect &lt;jdbcurl&gt;</code>                          | n/a                                    |
| List tables        | <code>!table</code>                                            | <code>show tables;</code>              |
| List columns       | <code>!column &lt;table_name&gt;</code>                        | <code>desc &lt;table_name&gt;;</code>  |
| Run query          | <code>&lt;HQL query&gt;;</code>                                | <code>&lt;HQL query&gt;;</code>        |
| Save result set    | <code>!record &lt;file_name&gt;</code><br><code>!record</code> | N/A                                    |
| Run shell CMD      | <code>!sh ls</code><br>This is available since Hive 0.14.0.    | <code>!ls;</code>                      |
| Run dfs CMD        | <code>dfs -ls</code>                                           | <code>dfs -ls;</code>                  |
| Run file of SQL    | <code>!run &lt;file_name&gt;</code>                            | <code>source &lt;file_name&gt;;</code> |
| Check Hive version | <code>!dbinfo</code>                                           | <code>!hive --version;</code>          |
| Quit mode          | <code>!quit</code>                                             | <code>quit;</code>                     |

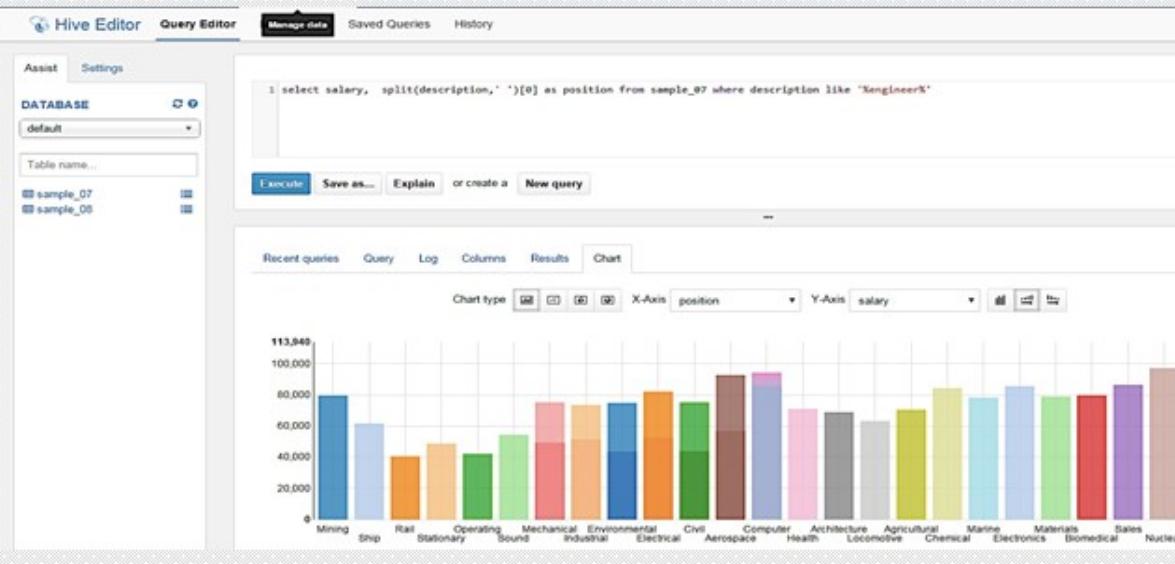
`jdbc:hive2://ubuntu:11000/db2;user=foo;password=barc`

## Hive Interface – Others

 http://www.it21learning.com

# Hive Interface – Others

Hive Web Interface | Hue | JDBC/ODBC | IDE



The screenshot shows the Hive Web Interface. At the top, there are tabs for 'Hive Editor' (selected), 'Query Editor', 'Message center', 'Saved Queries', and 'History'. Below this is a sidebar with 'Assist' and 'Settings' buttons, and a 'DATABASE' dropdown set to 'default'. Under 'Table name...', there are two tables listed: 'sample\_07' and 'sample\_06'. The main area contains a query editor with the following SQL code:

```
1 select salary, split(description,' ')[0] as position from sample_07 where description like '%engineer%'
```

Below the query editor are buttons for 'Execute', 'Save as...', 'Explain', 'or create a', and 'New query'. A navigation bar at the bottom includes 'Recent queries', 'Query', 'Log', 'Columns', 'Results' (selected), and 'Chart'. The 'Chart' section allows users to choose 'Chart type' (Bar, Line, Scatter, Bubble) and set 'X-Axis' to 'position' and 'Y-Axis' to 'salary'. The resulting bar chart displays salary values for various positions. The chart has a y-axis ranging from 0 to 100,000 and an x-axis with categories like Mining, Ship, Rail, Stationary, Operating, Sound, Mechanical, Industrial, Environmental, Electrical, Civil, Aerospace, Computer, Health, Architecture, Locomotive, Agricultural, Chemical, Marine, Electronics, Materials, Biomedical, Sales, and Nuclear.

| Position      | Salary  |
|---------------|---------|
| Mining        | 80,000  |
| Ship          | 60,000  |
| Rail          | 40,000  |
| Stationary    | 45,000  |
| Operating     | 50,000  |
| Sound         | 55,000  |
| Mechanical    | 60,000  |
| Industrial    | 60,000  |
| Environmental | 60,000  |
| Electrical    | 65,000  |
| Civil         | 65,000  |
| Aerospace     | 90,000  |
| Computer      | 95,000  |
| Health        | 65,000  |
| Architecture  | 65,000  |
| Locomotive    | 60,000  |
| Agricultural  | 70,000  |
| Chemical      | 80,000  |
| Marine        | 75,000  |
| Electronics   | 85,000  |
| Materials     | 75,000  |
| Biomedical    | 80,000  |
| Sales         | 85,000  |
| Nuclear       | 100,000 |

11

## Data Type – Primitive Type

# Data Type – Primitive Type

| Type    | Example        | Type      | Example                   |
|---------|----------------|-----------|---------------------------|
| TINYINT | 10Y            | SMALLINT  | 10S                       |
| INT     | 10             | BIGINT    | 100L                      |
| FLOAT   | 1.342          | DOUBLE    | 1.234                     |
| DECIMAL | 3.14           | BINARY    | 1010                      |
| BOOLEAN | TRUE           | STRING    | 'Book' or "Book"          |
| CHAR    | 'YES' or "YES" | VARCHAR   | 'Book' or "Book"          |
| DATE    | '2013-01-31'   | TIMESTAMP | '2013-01-31 00:13:00.345' |

## Data Type – Complex Type

# Data Type – Complex Type

| Type   | Example                    | Define                          | Example          |
|--------|----------------------------|---------------------------------|------------------|
| ARRAY  | ['Apple','Orange','Mongo'] | ARRAY<string>                   | a[0] = 'Apple'   |
| MAP    | {'A':'Apple','O':'Orange'} | MAP<string,string>              | b['A'] = 'Apple' |
| STRUCT | {'Apple':2}                | STRUCT<fruit:string,weight:int> | c.weight = 2     |

- ARRAY has same type for all the elements – equal to MAP
  - uses sequence from 0 as keys
- MAP has same type of key value pairs
- STRUCT likes table/records

## Hive Data Structure



<http://www.it21learning.com>

# Hive Data Structure

| Data Structure | Logical                      | Physical                         |
|----------------|------------------------------|----------------------------------|
| Database       | A collection of tables       | Folder with files                |
| Table          | A collection of rows of data | Folder with files                |
| Partition      | Columns to split data        | Folder                           |
| Buckets        | Columns to distribute data   | Files                            |
| Row            | Line of records              | Line in a file                   |
| Columns        | Slice of records             | Specified positions in each line |
| Views          | Shortcut of rows of data     | n/a                              |
| Index          | Statistics of data           | Folder with files                |

# Hive Database

The database in describes a collection of tables that are used for a similar purpose or belong to the same group. If the database is not specified (use database\_name), the default database is used. Hive creates a directory for each database at /user/hive/warehouse, which can be defined through hive.metastore.warehouse.dir property except default database.

- CREATE DATABASE IF NOT EXISTS myhivebook;
- SHOW DATABASES;
- DESCRIBE DATABASE default; //Provide more details than 'SHOW', such as comments, location
- DROP DATABASE IF EXISTS myhivebook CASCADE;
- ALTER DATABASE myhivebook SET OWNER user dayongd;

# Hive Tables - Concept

### **External Tables**

Data is kept in the HDFS path specified by LOCATION keywords. The data is not managed by Hive fully since drop the table (metadata) will not delete the data

### **Internal Tables/Managed Table**

Data is kept in the default path, such as /user/hive/warehouse/employee. The data is fully managed by Hive since drop the table (metadata) will also delete the data

## Hive Tables - DDL



<http://www.it21learning.com>

# Hive Tables - DDL

```
1 Michael|Montreal,Toronto|Male,30|DB:80|Product:Developer^DLead
2 Will|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
3 Shelley|New York|Female,27|Python:80|Test:Lead,COE:Architect
4 Lucy|Vancouver|Female,37|Sales:89,HR:94|Sales:Lead
```

```
CREATE EXTERNAL IF NOT EXISTS TABLE employee_external(
    namestring,
    work_place ARRAY<string>,
    sex_age STRUCT<sex:string,age:int>,
    skills_score MAP<string,int>,
    depart_title MAP<STRING,ARRAY<STRING>>
)
COMMENT 'This is an external table'
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'
    COLLECTION ITEMS TERMINATED BY ','
    MAP KEYS TERMINATED BY ':'
STORED AS TEXTFILE
LOCATION '/user/dayongd/employee';
```

IF NOT EXISTS is optional  
TEMPORARY table is also supported

List of columns with data type  
There is no constrain support

Table comment is optional

How to separate columns

How to separate MAP and Collections

File format

Data files path in HDFS (URI)

No sub-directory is allowed in the URI

# Side Notes - Delimiters

The default delimiters in Hive are as follows:

- Row delimiter: Can be used with *Ctrl + A* or <sup>A</sup> (Use \001 when creating the table)
- Collection item delimiter: Can be used with *Ctrl + B* or <sup>B</sup> (\002)
- Map key delimiter: Can be used with *Ctrl + C* or <sup>C</sup> (\003)

### Issues

If delimiter is overridden during the table creation, it only works in flat structure JIRA Hive-365. For nested types, the level of nesting determines the delimiter, for example

- ARRAY of ARRAY – Outer ARRAY is <sup>B</sup>, inner ARRAY is <sup>C</sup>
- MAP of ARRAY – Outer MAP is <sup>C</sup>, inner ARRAY is <sup>D</sup>

# Hive Table – DDL – Create Table

- CTAS – CREATE TABLE ctas\_employee as SELECT \* FROM employee
- CTAS cannot create a partition, external, or bucket table
- CTAS with Common Table Expression (CTE)

```
CREATE TABLE cte_employee AS  
WITH  
r1 AS (SELECT name FROM r2 WHERE name = 'Michael'),  
r2 AS (SELECT name FROM employee WHERE sex_age.sex= 'Male'),  
r3 AS (SELECT name FROM employee WHERE sex_age.sex= 'Female')  
SELECT * FROM r1 UNION ALL SELECT * FROM r3;
```

- Create table like other table (fast), such as CREATE TABLE employee\_like LIKE employee

# Hive Table – Drop/Truncate/Alter Table

- DROP TABLE IF EXISTS employee [PERGE] statement removes metadata completely and move data to .Trash folder in the user home directory in HDFS if configured. With PERGE option (since Hive 0.14.0), the data is removed completely. When to drop an external table, the data is not removed. Such as DROP TABLE IF EXISTS employee;
- TRUNCATE TABLE employee statement removes all rows of data from an internal table.
- ALTER TABLE employee RENAME TO new\_employee statement renames the table
- ALTER TABLE c\_employee SET TBLPROPERTIES ('comment'='New name, comments') statement sets table property
- ALTER TABLE employee\_internal SET SERDEPROPERTIES ('field.delim' = '\$') statement set SerDe properties
- ALTER TABLE c\_employee SET FILEFORMAT RCFILE statement sets file format.

```
DROP TABLE IF EXISTS empty_ctas_employee;
```

# Hive Table – Alter Table Columns

- ALTER TABLE employee\_internal CHANGE name employee\_name STRING [BEFORE|AFTER] sex\_age statement can be used to change column name, position or type
- ALTER TABLE c\_employee ADD COLUMNS (work string) statement add another column and type to the table at the end.
- ALTER TABLE c\_employee REPLACE COLUMNS (name string) statement replace all columns in the table by specified column and type. After ALTER in this example, there is only one column in the table.

**Note:**

The ALTER TABLE statement will only modify the metadata of Hive, not actually data. User should make sure the actual data conforms with the metadata definition.

# Hive Partitions - Overview

- To increase performance Hive has the capability to partition data
  - ◆ The values of partitioned column divide a table into segments (folders)
  - ◆ Entire partitions can be ignored at query time
  - ◆ Similar to relational databases' but not as advanced
- Partitions have to be properly created by users. When inserting data must specify a partition
- There is no difference in schema between "partition" columns and regular columns
- At query time, Hive will automatically filter out partitions for better performance

## Hive Partitions

# Hive Partitions

```
CREATE TABLE employee_partitioned(  
    name string,  
    work_place ARRAY<string>,  
    sex_age STRUCT<sex:string,age:int>,  
    skills_score MAP<string,int>,  
    depart_title MAP<STRING,  
        ARRAY<STRING>> )  
PARTITIONED BY (Year INT, Month INT)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '|'  
COLLECTION ITEMS TERMINATED BY ','  
MAP KEYS TERMINATED BY ':';
```

Partition is NOT enabled automatically. We have to add/delete partition by ALTER TABLE statement

```
1  --Add multiple partitions  
2  jdbc:hive2://> ALTER TABLE employee_partitioned ADD  
3  . . . . .> PARTITION (year=2014, month=11)  
4  . . . . .> PARTITION (year=2014, month=12);  
5  No rows affected (0.248 seconds)  
6  
7  jdbc:hive2://> SHOW PARTITIONS employee_partitioned;  
8  +-----+  
9  | partition |  
10 | +-----+  
11 | | year=2014/month=11 |  
12 | | year=2014/month=12 |  
13 | +-----+  
14 2 rows selected (0.108 seconds)  
15  
16  --Drop the partition  
17  jdbc:hive2://> ALTER TABLE employee_partitioned  
18  . . . . .> DROP IF EXISTS PARTITION (year=2014, month=11);  
19  jdbc:hive2://> SHOW PARTITIONS employee_partitioned;  
20  +-----+  
21  | partition |  
22  | +-----+  
23  | | year=2014/month=12 |  
24  | +-----+  
25  1 row selected (0.107 seconds)
```

# Hive Partitions – Dynamic Partitions

Hive also supports dynamically giving partition values. This is useful when data volume is large and we don't know what will be the partition values. To enable it, see line 1.

By default, the user must specify at least one static partition column. This is to avoid accidentally overwriting partitions. To disable this restriction, we can set the partition mode to nonstrict (see line 3) from the default strict mode.

```
1 SET hive.exec.dynamic.partition=true;
2
3 SET hive.exec.dynamic.partition.mode=nonstrict;
4
5 //example of dynamic partition
6 INSERT INTO TABLE employee_partitioned
7 PARTITION(year, month)
8 SELECT name, array('Toronto') as work_place,
9 named_struct("sex","Male","age",30) as sex_age,
10 map("Python",90) as skills_score,
11 map("R&D",array('Developer')) as depart_title,
12 year(start_date) as year,
13 month(start_date) as month
14 FROM employee_hr eh
15 WHERE eh.employee_id = 102;
```

# Hive Buckets - Overview

- The bucket corresponds to the segments of files in the HDFS
- Mechanism to query and examine random samples of data and speed JOIN
- Break data into a set of buckets based on a hash function of a "bucket column"
- Hive doesn't automatically enforce bucketing. User is required to specify the number of buckets by setting # of reducer or set the enforced bucketing.

```
SET mapred.reduce.tasks = 256;
```

```
SET hive.enforce.bucketing = true;
```

- To define number of buckets, we should avoid having too much or too little data in each buckets. A better choice somewhere near two blocks of data. Use  $2^N$  as the number of buckets

Data in the same buckets can join fast since they are likely have the same group of keys

# Hive Buckets - DDL

```
1 CREATE TABLE employee_id_buckets
2 (
3     name string,
4     employee_id int,
5     work_place ARRAY<string>,
6     sex_age STRUCT<sex:string,age:int>,
7     skills_score MAP<string,int>,
8     depart_title MAP<string,ARRAY<string >>
9 )
10 CLUSTERED BY (employee_id) INTO 2 BUCKETS
11 ROW FORMAT DELIMITED
12 FIELDS TERMINATED BY '|'
13 COLLECTION ITEMS TERMINATED BY ','
14 MAP KEYS TERMINATED BY ':';
15
16 INSERT OVERWRITE TABLE employee_id_buckets
17 SELECT * FROM employee_id;
18
19 --Verify the buckets in the HDFS
20 hdfs dfs -ls /user/hive/warehouse/employee_id_buckets
21
22 Found 2 items
23 /user/hive/warehouse/employee_id_buckets/000000_0
24 /user/hive/warehouse/employee_id_buckets/000001_0
25
```

- Use 'CLUSTERED BY' statements to define buckets (Line 10)
- To populate the data into the buckets, we have to use INSERT statement instead of LOAD statement (Line 16 – 17) since it does not verify the data against metadata definition (copy files to the folders only)
- Buckets are list of files segments (Line 23 and 24)

# Hive Views – Overview

- View is a logical structure to simplify query by hiding sub query, join, functions in a virtual table.
- Hive view does not store data or get materialized
- Once the view is created, its schema is frozen immediately. Subsequent changes to the underlying table schema (such as adding a column) will not be reflected.
- If the underlying table is dropped or changed, querying the view will be failed.

**CREATE VIEW view\_name AS SELECT statement;**

**ALTER VIEW view\_name SET TBLPROPERTIES ('comment' =  
'This is a view');**

**DROP VIEW view\_name;**

View is not likely optimized by optimizer

## Hive SELECT Statement



<http://www.it21learning.com>

# Hive SELECT Statement

- SELECT statement is used to project the rows meeting query conditions specified
- Hive SELECT statement is subset of database standard SQL

### Examples

- `SELECT *, SELECT column_name, SELECT DISTINCT`
- `SELECT * FROM employee LIMIT 5` //Limit rows returned
- `WITH t1 AS (SELECT ...) SELECT * FROM t1` //CTE – Common Table Expression
- `SELECT * FROM (SELECT * FROM employee) a;` //Nested query
- `SELECT name, sex_age.sex AS sex FROM employee a WHERE EXISTS  
(SELECT * FROM employee b WHERE a.sex_age.sex = b.sex_age.sex AND b.sex_age.sex =  
'male');`  
`//The subquery support EXIST, NOT EXSIT, IN, NOT IN.`

However, nested subquery is not supported. IN and NOT IN only support one column.

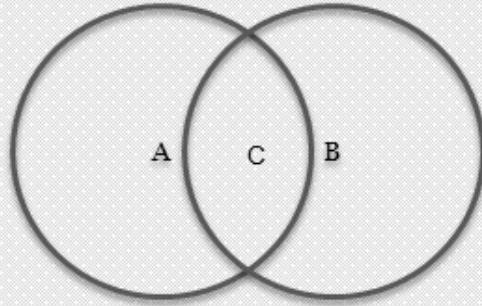
View is not likely optimized by optimizer

# Hive Join - Overview

- JOIN statement is used to combine the rows from two or more tables together
- Hive JOIN statement is similar to database JOIN. But Hive does not support unequal JOIN.
- INNER JOIN, OUTER JOIN (RIGHT OUTER JOIN, LEFT OUTER JOIN, FULL OUTER JOIN), and CROSS JOIN (Cartesian product/JOIN ON 1=1), Implicit JOIN (INNER JOIN without JOIN keywords but use , separate tables)

### Example

- C = C1 JOIN C2
- A = C1 LEFT OUTER JOIN C2
- B = C1 RIGHT OUTER JOIN C2
- AUBUC = C1 FULL OUTER JOIN C2



View is not likely optimized by optimizer

# Hive Join – MAPJOIN

- MAPJOIN statement means doing JOIN only by map without reduce job. The MAPJOIN statement reads all the data from the small table to memory and broadcast to all maps.
- When `hive.auto.convert.join` setting is set to true, Hive automatically converts the JOIN to MAPJOIN at runtime if possible instead of checking the MAPJOIN hint.

Example:

```
SELECT /*+ MAPJOIN(employee) */ emp.name, emph.sin_number  
FROM employee emp JOIN employee_hr emph ON emp.name = emph.name;
```

The MAPJOIN operator does not support the following:

- Use MAPJOIN after UNION ALL, LATERAL VIEW, GROUP BY/JOIN/SORT BY/CLUSTER BY/DISTRIBUTE BY
- Use MAPJOIN before by UNION, JOIN and other MAPJOIN

View is not likely optimized by optimizer

## Hive Set Ops – UNION



<http://www.it21learning.com>

# Hive Set Ops – UNION

- Hive support UNION ALL (keep duplications) and support UNION after v1.2.0
- Hive UNION ALL cannot use in the top level query until v0.13.0,  
such as SELECT A UNION ALL SELECT B
- Other set operator can be implemented using JOIN/OUTER JOIN to implement

Example:

```
//MINUS                                //INTERCEPT
jdbc:hive2://> SELECT a.name            jdbc:hive2://> SELECT a.name
.....> FROM employee a                 .....> FROM employee a
.....> LEFT JOIN employee_hr b        .....> JOIN employee_hr b
.....> ON a.name = b.name             .....> ON a.name = b.name;
.....> WHERE b.name IS NULL;
```

View is not likely optimized by optimizer

# Hive Data Movement – LOAD

- To move data in Hive, it uses the LOAD keyword. Move here means the original data is moved to the target table/partition and does not exist in the original place anymore.

```
1 LOAD DATA LOCAL INPATH '/home/dayongd/Downloads/employee_hr.txt'
2   OVERWRITE INTO TABLE employee_hr;
3
4
5 LOAD DATA LOCAL INPATH '/home/dayongd/Downloads/employee.txt'
6   OVERWRITE INTO TABLE employee_partitioned
7   PARTITION (year=2014, month=12);
8
9 LOAD DATA INPATH '/user/dayongd/employee/employee.txt'
10  OVERWRITE INTO TABLE employee;
11
12 LOAD DATA INPATH 'hdfs://[dfs_host]:8020/user/dayongd/employee/employee.txt'
13  OVERWRITE INTO TABLE employee;
```

LOCAL keywords specifies where the files are located in the host  
OVERWRITE is used to decide whether to append or replace the existing data

View is not likely optimized by optimizer

# Hive Data Exchange – INSERT

- To insert data into table/partition, Hive uses INSERT statement.
- Generally speaking, Hive INSERT is weak than what's in the DBMS.
- Hive INSERT supports OVERWRITE and INTO syntax
- Hive only support INSERT ... SELECT (INSERT VALUES start from v0.14.0 on special ACID tables only)
- Hive use INSERT OVERWRITE LOCAL DIRECTORY local\_directory SELECT \* FROM employee to insert data to the local files (opposite to LOAD statement). Only “INTO” keywords are supported. There is no support data append.
- Hive supports multiple INSERT from the same table

View is not likely optimized by optimizer

## Hive Data Exchange – INSERT Example



<http://www.it21learning.com>

# Hive Data Exchange – INSERT Example

```
1 //Insert from select
2 INSERT INTO TABLE employee SELECT * FROM ctas_employee;
3
4 //Multiple insert
5 FROM ctas_employee
6 INSERT OVERWRITE TABLE employee
7 SELECT *
8 INSERT OVERWRITE TABLE employee_internal
9 SELECT * ;
10
11 //Insert to file in the local directory
12 INSERT OVERWRITE LOCAL DIRECTORY '/tmp/output1'
13 SELECT * FROM employee;
14
15 //Insert to local file with specified row separators
16 INSERT OVERWRITE LOCAL DIRECTORY '/tmp/output2'
17 ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
18 SELECT * FROM employee;
```

View is not likely optimized by optimizer

# Hive Data Exchange – [EX|IM]PORT

- Hive uses IMPORT and EXPORT statement to deal with data migration
- All data and metadata are exported or imported
- The EXPORT statement export data in a subdirectory called data and metadata in a file called \_metadata. For example,
  - ✧ EXPORT TABLE employee TO '/user/dayongd/output3';
  - ✧ EXPORT TABLE employee\_partitioned partition  
(year=2014, month=11) TO '/user/dayongd/output5';
- After EXPORT, we can manually copy the exported files to the other HDFS. Then, import them with IMPORT statement.
  - ✧ IMPORT TABLE employee\_imported FROM '/user/dayongd/output3';
  - ✧ IMPORT TABLE employee\_partitioned\_imported FROM '/user/dayongd/output5';

View is not likely optimized by optimizer

# Hive Sorting Data – ORDER and SORT

- ORDER BY (ASC|DESC) is similar to standard SQL
- ORDER BY in Hive performs a global sort of data by using only one reducer. For example, `SELECT name FROM employee ORDER BY NAME DESC;`
- SORT BY (ASC|DESC) decides how to sort the data in each reducer.
- When number of reducer is set to 1, it is equal to ORDER BY. For example,

When there is more than 1 reducer,  
the data is not sort properly.

```
1 SET mapred.reduce.tasks = 2;
2 SELECT name FROM employee SORT BY NAME DESC;
3 +-----+
4 | name |
5 +-----+
6 | Shelley |
7 | Michael |
8 | Lucy |
9 | Will |
10 +-----+
```

View is not likely optimized by optimizer

# Hive Sorting Data – DISTRIBUTE

- DISTRIBUTE BY is similar to the GROUP BY statement in standard SQL
- It makes sure the rows with matching column value will be partitioned to the same reducer
- When to use with SORT BY, put DISTRIBUTE BY before SORT BY (since partition is ahead of reducer sort)
- The distributed column must appear in the SELECT column list.

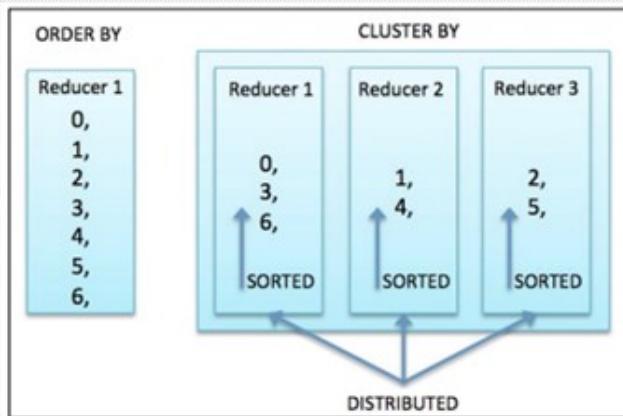
```
SELECT name, employee_id  
FROM employee_hr  
DISTRIBUTE BY employee_id SORT BY name;
```

View is not likely optimized by optimizer

# Hive Sorting Data – CLUSTER

- CLUSTER BY = DISTRIBUTE BY + SORT BY on the same column
- CLUSTER BY does not support DESC yet
- To fully utilize all reducer to perform global sort, we can use CLUSTER BY first, then ORDER BY after.
- An example,

```
SELECT name, employee_id
FROM employee_hr CLUSTER BY
name;
```



View is not likely optimized by optimizer

# Hive Other Topics – Not Covered

- Hive operators
- Hive UDFs
- Hive aggregations
- UDFs extension
- Streaming
- Hive securities
- Hue

View is not likely optimized by optimizer

## Q&A



<http://www.it21learning.com>

## Q&A

Thank You

# Mastering Apache Pig

# Programming Apache Pig

Wayne Shen



<http://www.it21learning.com>

# What is Apache Pig?

- Apache Pig is a platform for analyzing large data sets that consists of a high-level language (Pig Latin) for expressing data analysis programs.
  - **Pig is an engine that executes Pig Latin locally or on a Hadoop cluster.**
    - Pig generates and compiles a Map/Reduce program(s) on the fly.
  - **Pigs eat everything**
    - Pig can process any data, structured or un-structured.
  - **Pigs Live anywhere**
    - Pig can run on any parallel data processing framework, so Pig scripts do not have to run just on Hadoop.
  - **Pig history**
    - Pig was created at Yahoo! to make it easier to analyze the data in HDFS without the complexities of writing a traditional MapReduce program.

# Pig Latin

- Pig Latin is a high-level data flow scripting language. Pig Latin scripts can be executed in one of the three ways:
  - **Pig script**
    - Write a Pig Latin program in a text file and execute it using the pig executable;
  - **Grunt shell**
    - Enter Pig statements manually one at a time from a CLI tool
  - **Embedded in Java**
    - Use the PigServer class to execute a Pig query from within Java code

# Pig Execution

- Pig executes in a unique fashion – some commands build on previous commands, while certain commands trigger a MapReduce job.
  - 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 is used.

# Pig Latin Relation

- Pig Latin Relation
  - Each processing step of a Pig Latin results in a new data set - Relation.
  - The name of a relation is referred as alias
  - `log = load '/user/data/content.txt' AS (lang:chararray);`



## Pig Latin Field

# Pig Latin Field

- While a Relation defines the collection of records, a Field defines a data element of records.

▫ customers = LOAD 'customer.data'  
    USING PigStorage(',') AS (name, gender, age, income, address);

Alias

Fields / Schema

- Fields can be used in subsequent processing commands.
  - highIncome = FILTER customers BY income > 1000000;

# Pig Latin DEMO

- rs = LOAD '/user/data/full\_text.txt' USING PigStorage('\t') AS (id:chararray, ts:chararray, location:chararray, latitude:float, longitude:float, tweet:chararray);
- tokens = FOREACH rs GENERATE FLATTEN(TOKENIZE(tweet)) AS token;
- grps = GROUP tokens BY token;
- grpCnt = FOREACH grps GENERATE group AS token, COUNT(tokens) AS cnt;
- srt = ORDER grpCnt BY cnt DESC;
- lmt = LIMIT srt 50;
- DUMP lmt;
  - Pig doesn't execute until it sees keyword such as DUMP or STORE

# Pig Data Types

- int - A 32-bit signed integer
- long - A 64-bit signed integer
- float - A 32-bit floating-point number
- double - A 64-bit floating-point number
- chararray - Strings of Unicode characters
- bytearray - A blob or array of bytes
- boolean - Can be either true or false (case-sensitive)
- datetime - A date and time stored in the format 1970-01-01T00:00:00.000+00:00
- BigDecimal and BigInteger - For performing precision arithmetic

# Pig Complex Types

- Tuple - Ordered set of fields.
  - A tuple is analogous to a row in an SQL table, with the fields being SQL columns.
  - Example: (Yonge St., Toronto, ON, L3R 6G2)
- Bag - Unordered collection of tuples
  - Example: { (Yonge St., Toronto, ON, L3R 6G2), (Finch Ave, North York, ON, M2J 3X1), (19th Dr., Oakville, ON, J6N 8H1) }
- Map - Collection of key value pairs
  - Example: [state#ON, name#John Smith, zip#L9B 1M2]
- In real world, a Bag can be an element of a Tuple, which is a value element of a Map, ... etc.

## Don't Mix them up



<http://www.it21learning.com>

# Don't Mix them up

| Pig                                                               | Hive                                                            |
|-------------------------------------------------------------------|-----------------------------------------------------------------|
| <b>Tuple:</b><br>(John,18,4.0F), Yonge St., Toronto, ON, L3R 6G2) | <b>Array:</b><br>ARRAY<string>: ['Apple', 'Orange', 'Mongo']    |
| <b>Bag:</b><br>{(1, 2, 3) (4, 2, 1) (8, 3, 4) (4, 3, 3)}          | <b>Struct:</b><br>STRUCT<fruit:string, weight:int>:{'Apple': 2} |
| <b>Map:</b><br>[state#ON, name#John Smith, zip#L9B 1M2]           | <b>Map:</b><br>MAP<string, string>: {'A':'Apple', 'O':'Orange'} |

www.it21learning.com

## Defining a Schema



<http://www.it21learning.com>

# Defining a Schema

- When data is well-structured, the schema can be defined when loading the data;
  - `customers = LOAD 'customer_data'  
AS (name : chararray, street:chararray, city:chararray, province:chararray);`
- The schema can also specify complex types.

Fields / Schema

'John Smith', ('abc Yonge St.', 'Toronto', 'ON', { '416-xxx-2342', '905-123-xxxx' }), 98000  
'Mike Doug', ('xyz 19th Ave.', 'Markham', 'ON', { '416-789-xxxx', '905-xxx-1234' }), 79000

```
customers = LOAD 'customer_data' AS  
( name : chararray,  
  contact : tuple { street:chararray, city:chararray, province:chararray, bag: { phone : chararray } },  
  salary : int );
```

- Once Schema is defined and associated, Pig performs error-checking with it.*

## FILTER

# FILTER

- Selects tuples from a relation based on conditions.
  - alias = FILTER alias BY expression;
  - Usage:
    - Use the FILTER operator to work with tuples or rows of data
      - if it is to work with columns of data, use the FOREACH ...GENERATE operation.

```
rs = LOAD '/user/data/twitter/full_text.txt' USING PigStorage('\t')
      AS (id:chararray, dt:chararray, loc:chararray, lat:float, lon:float, tweet:chararray);

f = FILTER rs BY id eq 'USER_79321756';
DUMP f;
```

full\_text.pig

- Execute:
  - pig -f full\_text.pig

## FOREACH ... GENERATE

# FOREACH ... GENERATE

- Generates data transformations based on columns of data.
  - alias = FOREACH alias GENERATE expression [expression ....]
  - Usage:
    - Use the FOREACH ...GENERATE operation to work with columns of data
      - if it is to work with tuples or rows of data, use the FILTER operation.
    - FOREACH ...GENERATE works with relations (outer bags) as well as inner bags:

```
rs = LOAD '/user/data/twitter/full_text.txt' USING PigStorage('\t')
      AS (id:chararray, dt:chararray, loc:chararray, lat:float, lon:float, tweet:chararray);

c = FOREACH rs GENERATE id, lat, lon;

f = FILTER c BY id eq 'USER_79321756';
DUMP f;
```

## GROUP

# GROUP

- Groups the data in one or multiple relations.
  - alias = GROUP alias { ALL | BY expression} [USING 'collected'];
    - When ALL used, all tuples go to a single group.
    - "collected" → Allows for more efficient computation of a group if the loader guarantees that the data for the same key is continuous and is given to a single map. The efficiency is achieved by performing the group operation in map rather than reduce.
  - Usage:
    - The GROUP operator groups together tuples that have the same group key.
    - The result of a GROUP operation is a relation that includes one tuple per group. This tuple contains two fields:
      - The first field is named "group" and is the same type as the group key.
      - The second field takes the name of the original relation and is type bag.

```
rs = LOAD '/user/data/twitter/full_text.txt' USING PigStorage('\t')
        AS (id:chararray, dt:chararray, loc:chararray, lat:float, lon:float, tweet:chararray);
c = FOREACH rs GENERATE id, (lat, lon) AS pos, tweet;
g = GROUP c BY pos;
DUMP g;
n = FOREACH g GENERATE group AS pos, COUNT(c) AS cnt;
DUMP n;
```

# COGROUP

- COGROUP is the same as GROUP, but with multiple relations involved.
  - alias = COGROUP alias { ALL | BY expression} [, alias ALL | BY expression ...] [PARALLEL n];
    - PARALLEL → n is the number of reduce tasks. The default value for n is 1.

```
o = LOAD '/user/data/owner.txt' USING PigStorage('\t') AS (owner:chararray,pet:chararray);
f = LOAD '/user/data/friend.txt' USING PigStorage('\t') AS (friend1:chararray,friend2:chararray);
x = COGROUP o BY owner, f BY friend2;
describe x
x: {group: chararray, o: {owner: chararray,pet: chararray}, f: {firend1: chararray,friend2: chararray} }
```

## ORDER

# ORDER

- Sorts a relation based on one or more fields.
  - alias = ORDER alias BY { \* [ASC|DESC] | field\_alias [ASC|DESC] [, field\_alias [ASC|DESC] ...] } [PARALLEL n];

```
A = LOAD 'data' AS (a1:int,a2:int,a3:int);  
  
DUMP A;  
(1,2,3)  
(4,2,1)  
(8,3,4)  
(4,3,3)  
(7,2,5)  
(8,4,3)
```

```
X = ORDER A BY a3 DESC;  
  
DUMP X;  
(7,2,5)  
(8,3,4)  
(1,2,3)  
(4,3,3)  
(8,4,3)  
(4,2,1)
```

- If relation A is sorted to produce relation X (X = ORDER A BY \* DESC;) relations A and X still contain the same thing.
- If the contents of relation X (DUMP X;) is dumped, the order is guaranteed (descending).
- However, if relation X (Y = FILTER X BY \$o > 1;) is further processed, there is no guarantee that the contents will be processed in the order originally specified (descending).

## INNER JOIN

# INNER JOIN

- Performs inner, equijoin of two or more relations based on common field values. **Inner joins ignore null keys.**
  - alias = JOIN alias BY {expression} (, alias BY {expression} ...) [USING 'replicated' | 'skewed' | 'merge'] [PARALLEL n];

```
A = LOAD 'data1' AS (a1:int,a2:int,a3:int);  
  
DUMP A;  
(1,2,3)  
(4,2,1)  
(8,3,4)  
(4,3,3)  
(7,2,5)  
(8,4,3)
```

```
B = LOAD 'data2' AS (b1:int,b2:int);  
  
DUMP B;  
(2,4)  
(8,9)  
(1,3)  
(2,7)  
(2,9)  
(4,6)  
(4,9)
```

```
X = JOIN A BY a1, B BY b1;  
  
DUMP X;  
(1,2,3,1,3)  
(4,2,1,4,6)  
(4,3,3,4,6)  
(4,2,1,4,9)  
(4,3,3,4,9)  
(8,3,4,8,9)  
(8,4,3,8,9)
```

# REPLICATED JOINS

- In REPLICATED JOINS, the large relation is followed by one or more small relations. The small relations must be small enough to fit into main memory; if they don't, the process fails and an error is generated.
  - Map side JOIN

```
big = LOAD 'big_data' AS (b1,b2,b3);
tiny = LOAD 'tiny_data' AS (t1,t2,t3);
mini = LOAD 'mini_data' AS (m1,m2,m3);
c = JOIN big BY b1, tiny BY t1, mini BY m1 USING 'replicated';
```

## SKEWED JOINS

# SKEWED JOINS

- Skewed Joins split the left input on the join predicate and streaming the right input.

- Reduce side JOIN

```
big = LOAD 'big_data' AS (b1,b2,b3);  
massive = LOAD 'massive_data' AS (m1,m2,m3);  
c = JOIN big BY b1, massive BY m1 USING 'skewed';
```

- **LIMITATION:**

- Skewed JOINS work with only two-table inner joins.

# MERGED JOINS

- Merged JOINS require that both inputs are already sorted on the join key.

```
a= LOAD 'data_a' AS (b1,b2,b3);  
b = LOAD 'data_b' AS (m1,m2,m3);  
c = JOIN a BY b1, b BY m1 USING 'merged';
```

- **LIMITATION:**

- Both inputs are sorted in \*ascending\* order of join keys.
- The merge join only has two inputs

## OUTER JOIN

# OUTER JOIN

- Performs an outer join of two or more relations based on common field values.
  - alias = JOIN left-alias BY left-alias-column [LEFT|RIGHT|FULL] [OUTER], right-alias BY right-alias-column [USING 'replicated' | 'skewed'] [PARALLEL n];

```
A = LOAD 'a.txt' AS (n:chararray, a:int);
B = LOAD 'b.txt' AS (n:chararray, m:chararray);
C = JOIN A by $o LEFT, B BY $o;
```

```
A = LOAD 'a.txt' AS (n:chararray, a:int);
B = LOAD 'b.txt' AS (n:chararray, m:chararray);
C = JOIN A BY $o FULL, B BY $o;
```

```
A = LOAD 'studenttab' AS (name, age, gpa);
B = LOAD 'votertab' AS (name, age, registration, contribution);
C = JOIN A BY name, B BY name RIGHT USING 'skewed';
```

## CROSS JOIN

# CROSS JOIN

- Computes the cross product of two or more relations.
  - alias = CROSS alias, alias [, alias ...] [PARALLEL n];
  - Use the CROSS operator to compute the cross product (Cartesian product) of two or more relations.
  - **CROSS is an expensive operation and should be used sparingly.**

```
A = LOAD 'data1' AS (a1:int,a2:int,a3:int);
DUMP A;
(1,2,3)
(4,2,1)

B = LOAD 'data2' AS (b1:int,b2:int);
DUMP B;
(2,4)
(8,9)
(1,3)
```

```
X = CROSS A, B;
DUMP X;
(1,2,3,2,4)
(1,2,3,8,9)
(1,2,3,1,3)
(4,2,1,2,4)
(4,2,1,8,9)
(4,2,1,1,3)
```

## UNION

# UNION

- Computes the union of two or more relations.
  - alias = UNION alias, alias [, alias ...];
  - Does not preserve the order of tuples.
  - Does not ensure that all tuples adhere to the same schema or that they have the same number of fields.
  - Does not eliminate duplicate tuples.

```
rs = LOAD '/user/data/twitter/full_text.txt' using PigStorage('\t')
      AS (id:chararray, dt:chararray, loc:chararray, lat:float, lon:float, tweet:chararray);

toronto = FILTER rs BY (lan >= 43.5847 and lon >= -79.6393) and (lan <= 43.8554 and lon <= -79.1156);
newyork = FILTER rs BY (lan >= 40.4774 and lon >= -74.2589) and (lan <= 40.9176 and lon <= -73.7004);

t = FOREACH toronto GENERATE id, 'Toronto' AS region;
n = FOREACH newyork GENERATE id, 'NewYork' AS region;
all = UNION t, n;
g = GROUP all BY region;
result = FOREACH g GENERATE group AS region, COUNT_STAR(all) AS cnt;
DUMP result;
```

## DISTINCT

# DISTINCT

- Removes duplicate tuples in a relation.
  - alias = DISTINCT alias [PARALLEL n];

```
rs = LOAD '/user/data/twitter/full_text.txt' using PigStorage('\t')
      AS (id:chararray, dt:chararray, loc:chararray, lat:float, lon:float, tweet:chararray);

ids = FOREACH rs GENERATE id;
x = DISTINCT ids;
DUMP x;
```

# Function - COUNT & COUNT\_STAR

- COUNT → Computes the number of elements in a bag by ignoring NULL values.
- COUNT\_STAR → Computes the number of elements in a bag by including NULL values.

```
A = LOAD 'data' AS (f1:int,f2:int,f3:int);
B = GROUP A BY f1;
X = FOREACH B GENERATE group AS f1, COUNT(A) AS cnt;
```

# Function - IsEmpty

- Checks if a bag or map is empty.
  - `IsEmpty(expression)`
  - The `IsEmpty` function checks if a bag or map is empty (has no data).  
The function can be used to filter data.

```
SSN = load 'ssn.txt' using PigStorage() as (ssn:long);

SSN_NAME = load 'students.txt' using PigStorage() as (ssn:long, name:chararray);

-- do a left out join of SSN with SSN_Name
X = JOIN SSN by ssn LEFT, SSN_NAME by ssn;

-- only keep those ssn's for which there is no name
Y = FILTER X by IsEmpty(SSN_NAME);
```

# Function - TOKENIZE

- Splits a string and outputs a bag of words.
  - **TOKENIZE(expression)**
  - Use the TOKENIZE function to split a string of words (all words in a single tuple) into a bag of words (each word in a single tuple).

```
rs = LOAD '/user/data/twitter/full_text.txt' USING PigStorage('\t')
      AS (id:chararray, dt:chararray, loc:chararray, lat:float, lon:float, tweet:chararray);

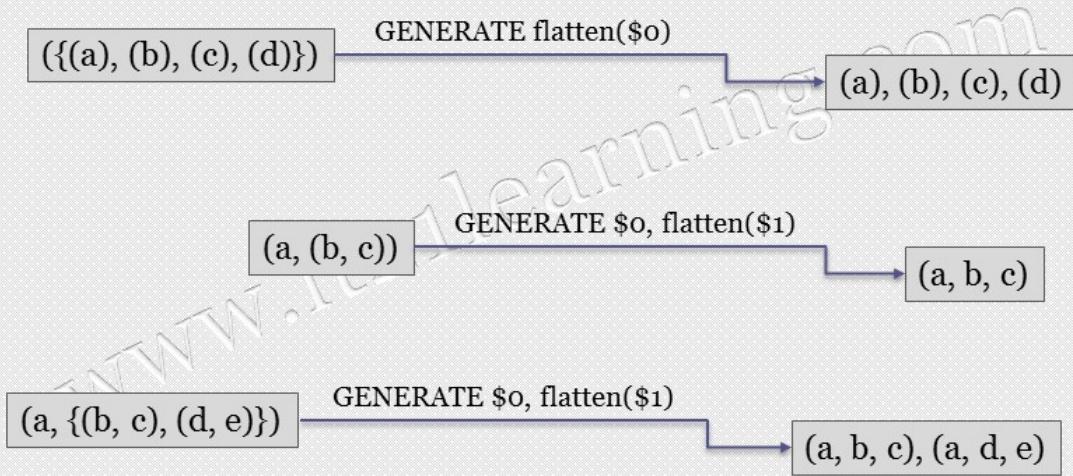
tws = FOREACH rs GENERATE tweet;
tw = LIMIT tws 1;

tokens = FOREACH tw GENERATE TOKENIZE($o);
DUMP tokens;
```

## Flatten Operator

# Flatten Operator

- Flatten un-nests tuples as well as bags, so it changes the structure of tuples and bags.



## CASTING/CONVERTING

# CASTING/CONVERTING

```
rs = LOAD '/user/data/twitter/full_text.txt' USING PigStorage('\t')
      AS (id:chararray, dt:chararray, loc:chararray, lat:float, lon:float, tweet:chararray);

x = FOREACH rs GENERATE (int)$3, (chararray)$4;
y = FOREACH rs GENERATE [id, {$1, ($3, $4)}];
z = FOREACH rs GENERATE TOMAP(id, TOBAG($1, TOTUPLE($3, $4))));
```

```
A = LOAD 'data' AS fld:bytearray;
DUMP A;
((1,2,3))
((4,2,1))
```

```
B = FOREACH A GENERATE (tuple(int,int,float))fld;
DUMP B;
((1,2,3))
((4,2,1))
```

```
A = LOAD 'data' AS fld:bytearray;
DUMP A;
({{4829090493980522200L}})
```

```
B = FOREACH A GENERATE (bag{tuple(long)})fld;
DUMP B;
({{(4829090493980522200L)}})
```

```
A = LOAD 'data' AS fld:bytearray;
DUMP A;
([open#apache])
([apache#hadoop])
([hadoop#pig])
([pig#grunt])
```

```
B = FOREACH A GENERATE ((map[])fld;
DUMP B;
([open#apache])
([apache#hadoop])
([hadoop#pig])
([pig#grunt]))
```

## STORE



<http://www.it21learning.com>

# STORE

- Stores or saves results to the file system.
  - STORE alias INTO 'directory' [USING function];

```
rs = LOAD '/user/data/twitter/full_text.txt' USING PigStorage('\t')
          AS (id:chararray, dt:chararray, loc:chararray, lat:float, lon:float, tweet:chararray);

toronto = FILTER rs BY ( lan>= 43.5847 and lon >= -79.6393) and (lan <= 43.8554 and lon <= -79.1156);
STORE toronto INTO '/user/data/twitter/toronto' USING PigStorage('\t');

newyork = FILTER rs BY (lan >= 40.4774 and lon >= -74.2589) and (lan <= 40.9176 and lon <= -73.7004);
STORE newyork INTO '/user/data/twitter/newyork' USING PigStorage('*');
```

***The default delimiter in Pig is a tab, same as in Hive.***

## Any Question?

# Any Question?



## Introducing HBase

## Apache HBase Overview

# Apache HBase Overview

Will Du

Dec. 19, 2015@GTA

## Outline

# Outline

- What is Apache HBase
- HBase History, Use Cases, and User Group
- HBase Architecture
- HBase Shell and Demo
- HBase API
- HBase SQL Access - Phoenix Demo
- HBase Advanced Topics
- Q&A



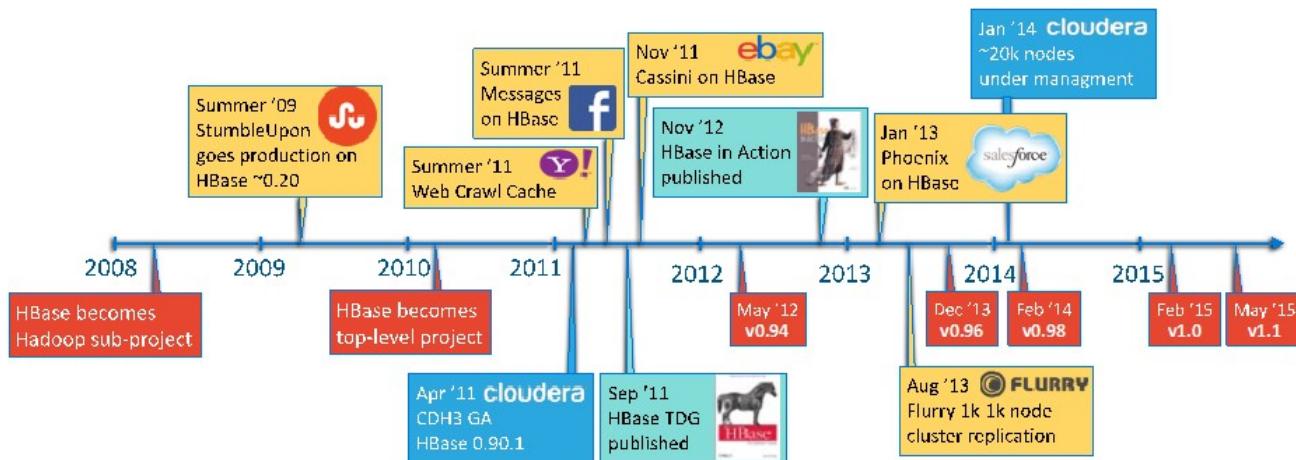
# What is Apache HBase

- HBase is a leading No-SQL database which stores data on HDFS.
- HBase is column-oriented database.
- HBase is based on the Google Big Table paper.
- HBase uses HDFS as storage and leverage its reliability.
- Data can be accessed quickly, ~2-20 millisecond response time.
- Great support random read and write 20k to 100k+ ops/s per node.
- Scales to 20,000+ nodes



## Apache HBase History

# Apache HBase History



| Year     | Event                                                         |
|----------|---------------------------------------------------------------|
| Nov 2006 | Google released the paper on BigTable.                        |
| Feb 2007 | Initial HBase prototype was created as a Hadoop contribution. |
| Oct 2007 | The first usable HBase along with Hadoop 0.15.0 was released. |

eBay presented a keynote at Hadoop World, describing the architecture of its completely rebuilt search engine, Cassini, slated to go live in 2012. It indexes all the content and user metadata to produce better rankings and refreshes indexes hourly. It is built using Apache Hadoop for hourly index updates and Apache HBase to provide random access to item information.

# Apache HBase Use Case

- Data access for medium- and high-scale services
  - ❖ Hundreds of enterprises and startups
  - ❖ Some of the largest Internet companies in the world
- Running major production workloads since 2011
- Use cases
  - ❖ messaging, security, collaboration, digital media
  - ❖ advertising, telecommunications, computing biology,
  - ❖ clinical informatics/healthcare, insurance
  - ❖ real-time access pattern
  - ❖ Time series of data

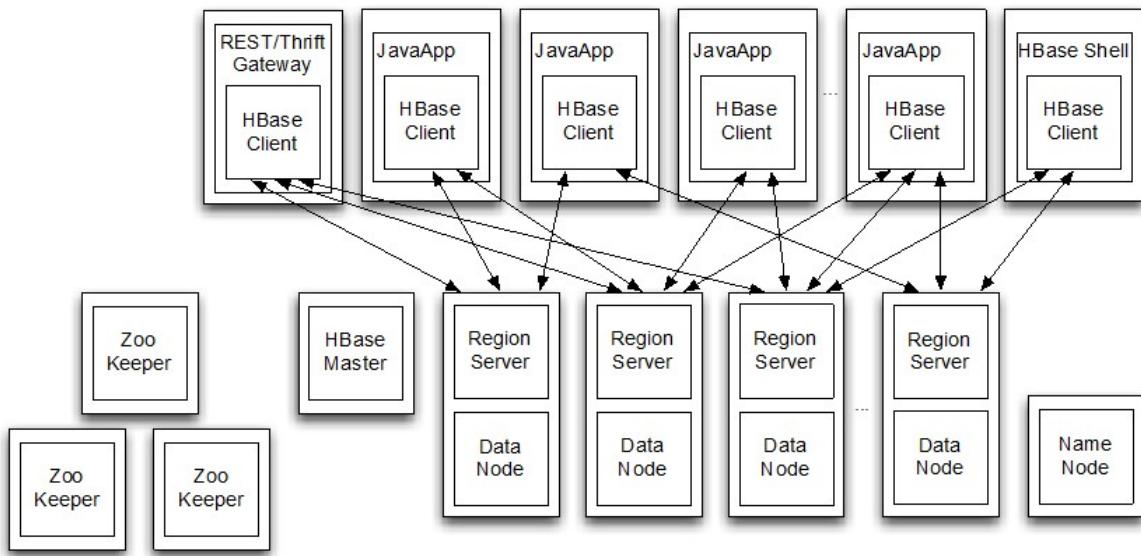
## Apache HBase User Groups

# Apache HBase User Groups



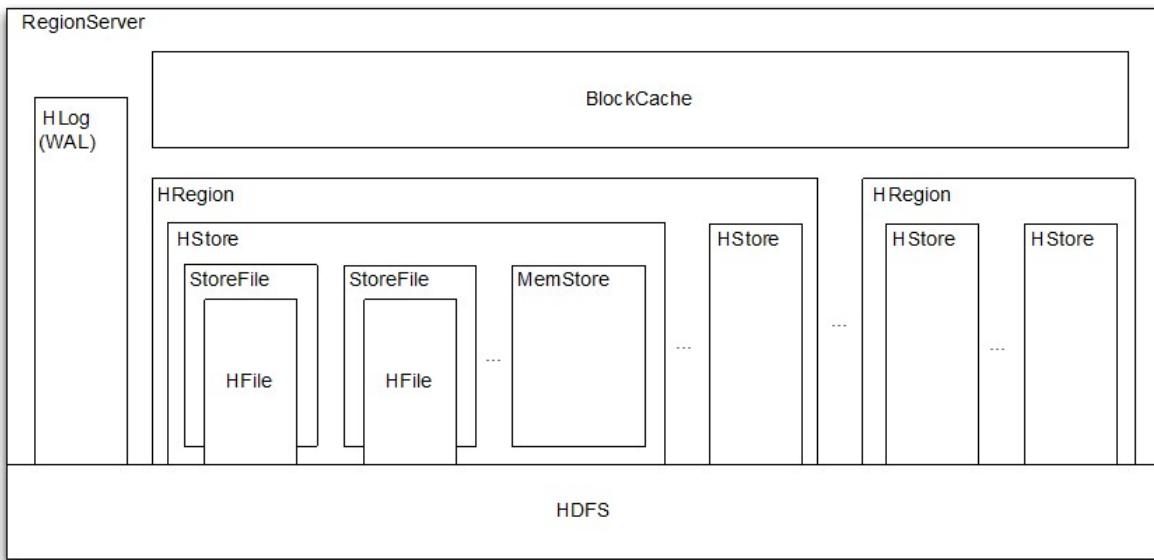
## HBase Overall Physical Architecture

# HBase Overall Physical Architecture



- An HBase RegionServer is collocated with an HDFS DataNode.
- HBase clients communicate directly with Region Servers for sending and receiving data.
- HMaster manages Region assignment and handles DDL operations.
- Online configuration state is maintained in ZooKeeper.
- HMaster and ZooKeeper are NOT involved in data path.

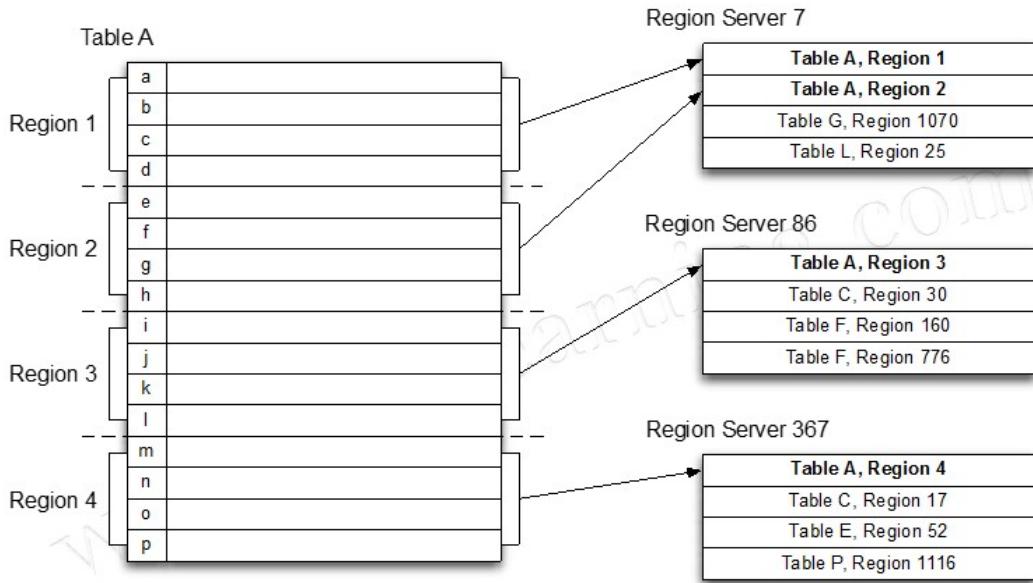
# Physical Architecture - Region Server



- A RegionServer contains a single WAL, single BlockCache (read cache), and multiple Regions. - A Region contains multiple Stores, one for each Column Family.
- A Store consists of multiple StoreFiles and a MemStore (write/edit cache).
- A StoreFile corresponds to a single HFile.
- HFiles and WAL are persisted on HDFS as sequence file.

When data is added to HBase, it is first written to the WAL (Write ahead log) called Hlog – A sequence file. Once the write is done, it is then written to an in memory called MemStore. Once the memory exceeds a certain threshold, it flushes to disk as an HFile. Over time HBase merges smaller HFiles into larger ones. This process is called compaction. WAL is used to recover.

# Logical Architecture - Region&Table



- A single table is partitioned into Regions of roughly equal size.
- Regions are assigned to the Region servers across the cluster.
- Region servers host roughly the same number of region.

## Logical Architecture – Rows

# Logical Architecture - Rows

| rowkey | column family | column qualifier | timestamp  | value                         |
|--------|---------------|------------------|------------|-------------------------------|
| a      | cf1           | "bar"            | 1368394583 | 7                             |
|        |               |                  | 1368394261 | "hello"                       |
|        |               | "foo"            | 1368394583 | 22                            |
|        | cf2           |                  | 1368394925 | 13.6                          |
|        |               |                  | 1368393847 | "world"                       |
|        |               | "2011-07-04"     | 1368396302 | "fourth of July"              |
| b      | cf2           | "number"         | 1368387684 | "almost the loneliest number" |
|        |               | "thumb"          | 1368387247 | [3.6 kb png data]             |

- Row key are unique and sorted.
- Schema can define when inserts the records
- Each Row can define its own columns, even if other rows do not use them.
- Related Columns grouped into Column Families (prefer<3) which are saved into different files.
- Multiple row versions maintained with unique timestamps.
- Value types can change between versions.
- HBase only knows bytes and clients must impart meaning.

Data is stored... in a big table • Sorted map datastore Tables consist of sorted rows, each of which has a primary row key  
Each row has a set of columns A column is specified as a column family and column qualifier pair A given cell (row, column family:column qualifier) can have different time- stamped values

# How to Access HBase - Shell

- hbase shell is an interactive mode of using hbase
- It support full set of hbase commands

| CMD Category | CMDs                                                                                                          |
|--------------|---------------------------------------------------------------------------------------------------------------|
| General      | version, status, whoami, help                                                                                 |
| DDL          | alter, create, describe, disable, drop, enable, exists, is_disabled, is_enabled, list                         |
| DML          | count, delete, deleteall, get, get_counter, incr, put, scan, truncate                                         |
| Tools        | assign, balance_switch, balancer, close_region, compact, flush, major_compact, move, split, unassign, zk_dump |
| Replication  | add_peer, disable_peer, enable_peer, remove_peer, start_replication, stop_replication                         |

## HBase Shell Demo

12

# HBase Shell Demo

Zeppelin Notebook



<http://www.it21learning.com>

12

# How to Access HBase - API

- Official API is Java
- It support full set of hbase commands
- External API [http://hbase.apache.org/book.html#external\\_apis](http://hbase.apache.org/book.html#external_apis)

| API Category | API Information                                                                                                                                                                                 |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Java         | <a href="https://hbase.apache.org/apidocs/">https://hbase.apache.org/apidocs/</a>                                                                                                               |
| Python       | <a href="https://happybase.readthedocs.org/en/latest/">https://happybase.readthedocs.org/en/latest/</a>                                                                                         |
| Scala        | Through Java/Spark                                                                                                                                                                              |
| Thrift       | <a href="https://wiki.apache.org/hadoop/Hbase/ThriftApi">https://wiki.apache.org/hadoop/Hbase/ThriftApi</a>                                                                                     |
| REST         | <a href="http://blog.cloudera.com/blog/2013/03/how-to-use-the-apache-hbase-rest-interface-part-1/">http://blog.cloudera.com/blog/2013/03/how-to-use-the-apache-hbase-rest-interface-part-1/</a> |

# How to Access HBase - PHOENIX

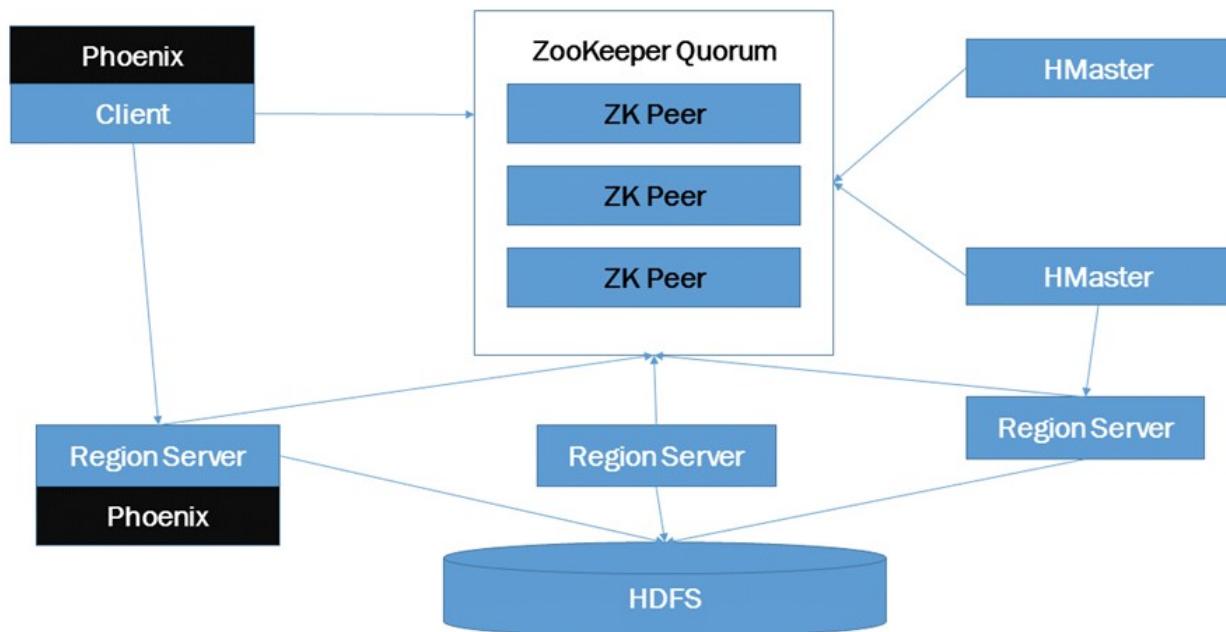
APACHE



- Phoenix Is:
  - ❖ A SQL Skin for HBase.
  - ❖ Provides a SQL interface for managing data in HBase.
  - ❖ Create tables, insert, update data and perform low-latency lookups through JDBC.
  - ❖ Phoenix JDBC driver easily embeddable in any app that supports JDBC.
- Phoenix Is NOT:
  - ❖ An replacement for the RDBMS.
  - ❖ Why? No transactions, lack of integrity constraints, many other areas still maturing.
- Phoenix Makes HBase Better:
  - ❖ Killer features like secondary indexes, joins, aggregation pushdowns.
  - ❖ Phoenix applies performance best-practices automatically and transparently.
  - ❖ If HBase is a good fit for your app, Phoenix makes it even better.

## Phoenix Architecture

# Phoenix Architecture



## Phoenix Provides SQL-like Syntax

# Phoenix Provides SQL-like Syntax

| API Code                                                                                                                                                                                                                                                                                                                                                                                                                                    | Phoenix DDL                                                                                                                                            |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>HBaseAdmin hbase = new HBaseAdmin(conf); HTableDescriptor desc = new HTableDescriptor("us_population"); HColumnDescriptor state = new HColumnDescriptor("state".getBytes()); HColumnDescriptor city = new HColumnDescriptor("city".getBytes()); HColumnDescriptor population = new HColumnDescriptor("population".getBytes()); desc.addFamily(state); desc.addFamily(city); desc.addFamily(population); hbase.createTable(desc);</pre> | <pre>CREATE TABLE us_population ( state CHAR(2) NOT NULL, city VARCHAR NOT NULL, population BIGINT CONSTRAINT my_pk PRIMARY KEY (state, city) );</pre> |
| <ul style="list-style-type: none"><li>• More flexible</li><li>• Integration</li></ul>                                                                                                                                                                                                                                                                                                                                                       | <ul style="list-style-type: none"><li>• Familiar SQL syntax.</li><li>• Provides additional constraint checking.</li></ul>                              |

## Phoenix Supported Keywords



<http://www.it21learning.com>

# Phoenix Supported Keywords

| Support Now                    | Support in Future        |
|--------------------------------|--------------------------|
| Standard SQL Data Types        | UNION                    |
| SELECT, UPSERT, DELETE         | More Windowing Functions |
| JOINS: Inner and Outer         | Transactions             |
| Subqueries                     | Cross Joins              |
| Secondary Indexes              | Authorization            |
| GROUP BY, ORDER BY, HAVING     | Replication Management   |
| AVG, COUNT, MIN, MAX, SUM      |                          |
| Primary Keys, Constraints      |                          |
| CASE, COALESCE                 |                          |
| VIEWS                          |                          |
| PERCENT_RANK, LAST FIRST VALUE |                          |

# Phoenix Use Case

- **Phoenix Is A Great Fit For:**

- ❖ Rapidly and easily building an application backed by HBase.
- ❖ SQL applications needing extreme scale, performance and concurrency.
- ❖ Re-using existing SQL skills while making the transition to Hadoop.
- ❖ BI Tools
- ❖ Ad hoc reporting

- **Consider Other Tools For:**

- ❖ Sophisticated SQL queries involving large joins or advanced SQL features.
- ❖ Full-Table Scans.
- ❖ ETL jobs (Kind of)
- ❖ Application drivers/interface

# Phoenix SQL Demo

Zeppelin Notebook



<http://www.it21learning.com>

# HBase Advanced Topics

Splits, Balance, Compaction, etc.



<http://www.it21learning.com>

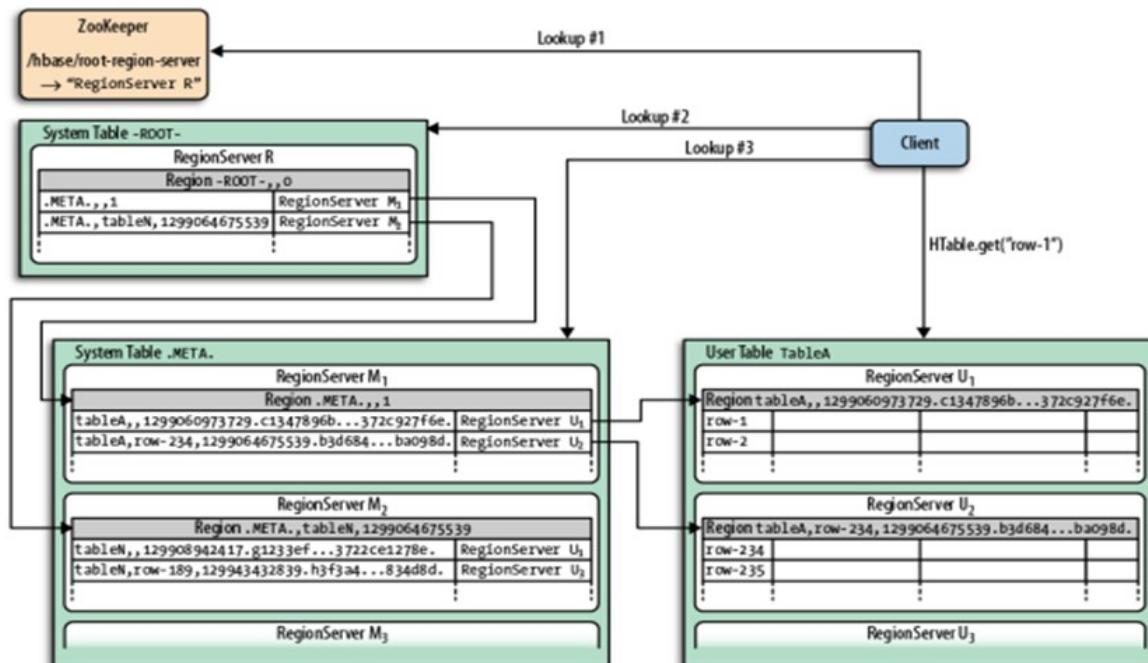
# Region Splits

- What is a Split
  - ❖ A “split” or “region split” is when a region is divided into 2 regions.
  - ❖ Usually because it gets too big.
  - ❖ The two splits will usually wind up on different servers.
- Region Split Strategies
  - ❖ Automatic (most common)
  - ❖ Manual (or Pre-Split by code)
  - ❖ Pluggable Split Policy
- Almost everyone uses “`ConstantSizeRegionSplitPolicy`”
- Splits happen when a storefile becomes larger than `hbase.hregion.maxfilesize`.
- Experts only: Other split policies exist and you can write your own.

## Row Identification

# Row Identification

ZooKeeper=>ROOT=>META=>User Table



# Compactions: Motivation

## Log-Structured Merge (LSM)

- Traditional databases are architected to update data in-place.
- Most modern databases use some sort of Log-Structured Merge (LSM).
- That means just write values to the end of a log and sort it out later.
- Pro: Inserts and updates are extremely fast.
- Con: Uses lots more space, so we need to do compaction.

Heather  
Hello my name is Bruce

Traditional Database

1. Update the value in-place.
2. Serve the value from disk.

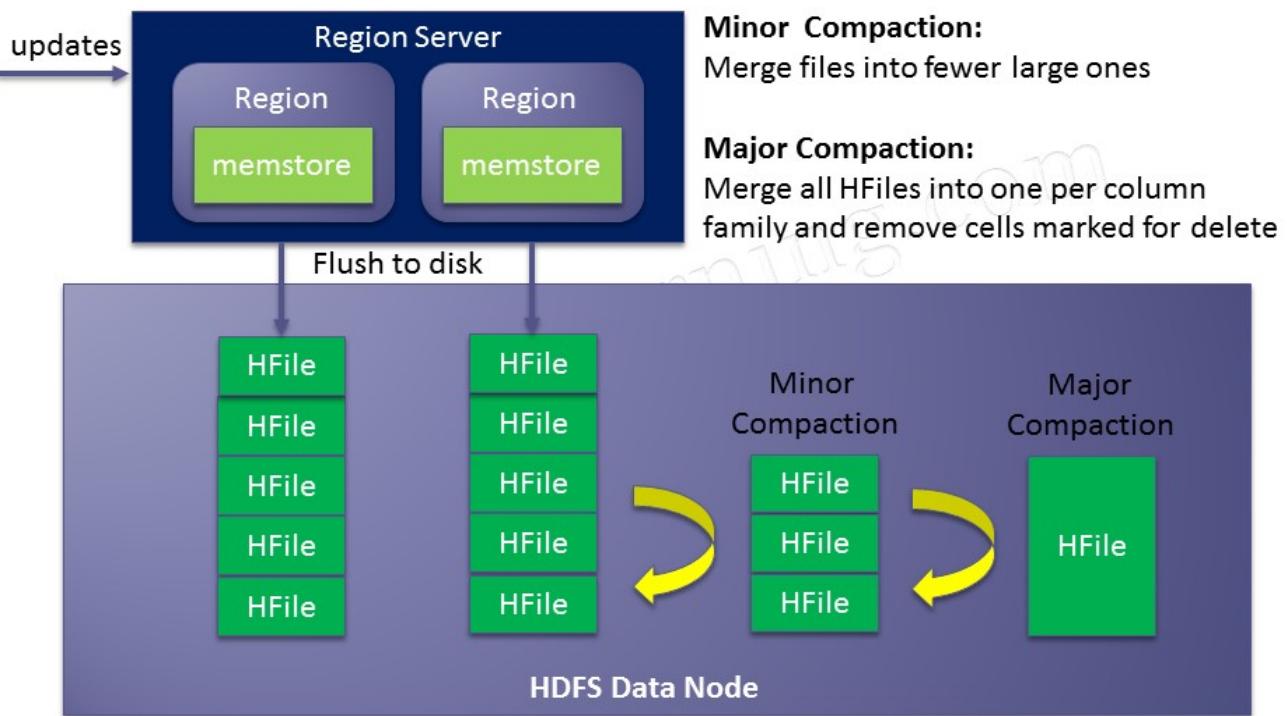
Hello my name is Bruce  
Hello my name is Heather

LSM System

1. Write both values into a log.
2. Merge them in memory at read time.
3. Serve the latest value.

## Compaction Implementation

# Compaction Implementation



# Compactions Control

- **Compactions:**

- ✓ Compaction: Re-write the log files and discard old values.
- ✓ Saves space, makes reads and recoveries faster.
- ✓ Compaction: Expensive, I/O intensive operation. Usually want this to happen off peak times.
- ✓ Some people schedule compactions externally. Rarely, compactions are completely disabled.

- **Flush -> Minor Compaction -> Major Compaction**

- ✓ Flush: Write the memstore out to a new store file. Event triggered.
- ✓ Minor Compaction: Combine recent store files into a larger store file. Event triggered.
- ✓ Major Compaction: Major rewrite of store data to minimize space utilization. Time triggered.

- **Relevant Controls:**

- ✓ Flush: `hbase.hregion.memstore.flush.size`
- ✓ Minor Compaction: `hbase.hstore.compaction.min/max`: Minimum / maximum # of store files (created by flushes) that must be present to trigger a minor compaction.
- ✓ Major Compaction: `hbase.hregion.majorcompaction`: Time interval for major compactions

**Thank You**

26



**Thank You**

Will Du  
[sparkera.ca](http://sparkera.ca)

26

## Introducing Apache Spark

## Apache Spark Overview

# Apache Spark Overview

Will Du

Dec. 08, 2015@GTA



<http://www.it21learning.com>

## Outline

# Outline

- Apache Spark History
- Spark Stack and Advantage
- Spark Architecture
- Resilient Distributed Dataset – RDD
- RDD Transformation and Action
- Short demo – wordcount
- Spark SQL and DataFrame
- Short demo – spark shell
- Q&A

# Apache Spark History



- Started in 2009 as a research project in UC Berkeley RAD lab which became AMP Lab.
- Spark researchers found that Hadoop MapReduce was inefficient for iterative and interactive computing.
- Spark was designed from the beginning to be fast for interactive, iterative with support for in-memory storage and fault-tolerance.
- Apart from UC Berkeley, Databricks, Yahoo! and Intel are major contributors.
- Spark was open sourced in March 2010 and transformed into Apache Foundation project in June 2013. One of most active project in apache.

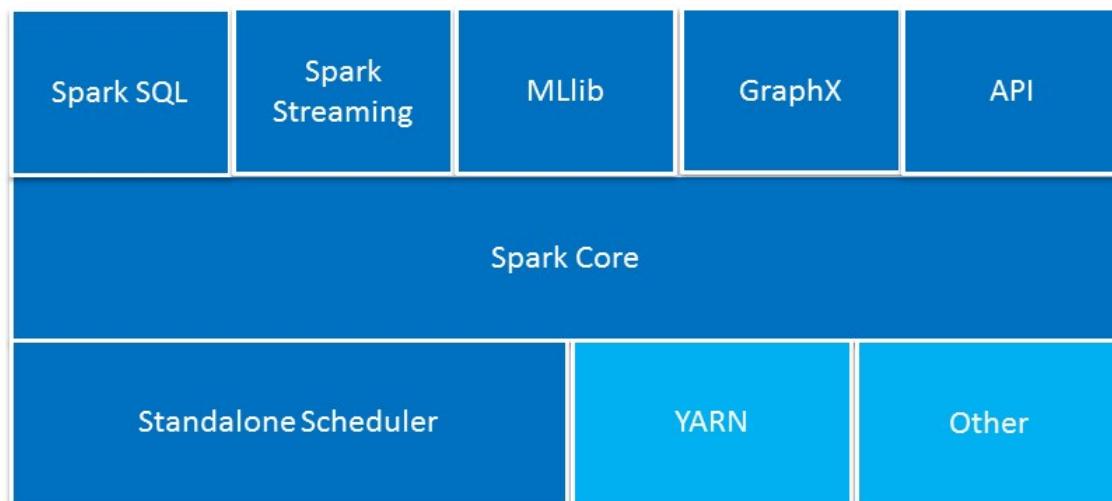
Resilient Distributed Dataset

## Apache Spark Stack



<http://www.it21learning.com>

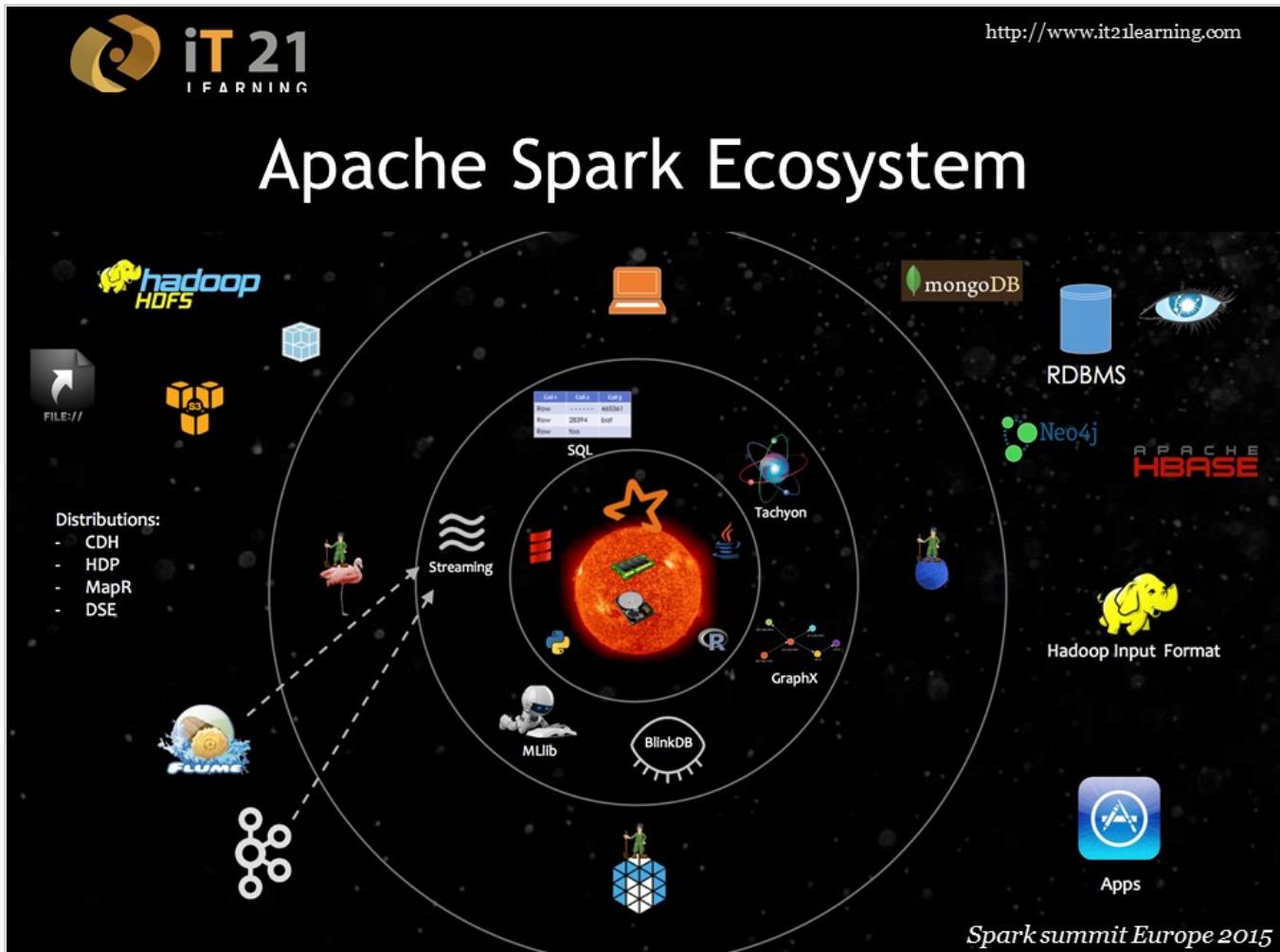
# Apache Spark Stack



*Full stack and great ecosystem.  
One stack to rule them all.*

Tachyon Mesos Blinkdb – 17T 100nodes 2sec 2-10 error

## Apache Spark Ecosystem



# Apache Spark Advantages

- Fast by leverage memory
- General purpose framework – ecosystem
- Good and wide API support - native integration with Java, Python, Scala
- Spark handles batch, interactive, and real-time within a single framework
- Leverage Resilient Distributed Dataset – RDD to process data in memory
  - ❖ Resilient – Data can be recreated if it is lost in the memory
  - ❖ Distributed – Stored in the memory across cluster
  - ❖ Dataset – Can be created from file or programmatically

Resilient Distributed Dataset

## Spark Hall of Fame



<http://www.it21learning.com>

### Spark Hall of Fame

#### ★ Largest cluster



#### ★ Largest single-day intake



#### ★ Longest-running job



#### ★ Largest shuffle job



LARGEST CLUSTER Tencent (8000+ nodes) LARGEST SINGLE-DAY INTAKE Tencent (1PB+ /day)  
LONGEST-RUNNING JOB Alibaba(1 week on 1PB+ data) LARGEST SHUFFLE Databricks PB Sort (1PB) MOST  
INTERESTING APP Jeremy Freeman Mapping the Brain at Scale (with lasers!)

# How to Work with Spark

- Scala IDE
  - Add hadoop dependency or use mvn
  - Compile to jar, build or download  
(<http://spark.apache.org/docs/latest/building-spark.html>)
  - Run with spark-submit
- spark-shell
  - Integrative command line application
- spark-sql
  - Where spark using hive metadata

## Demo – Word Count



http://www.it21learning.com

# Demo - Word Count

### MR code

```
1 package org.apache.hadoop.examples;
2 import org.apache.hadoop.conf.Configuration;
3 import java.util.StringTokenizer;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.IntWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Job;
8 import org.apache.hadoop.mapreduce.Mapper;
9 import org.apache.hadoop.mapreduce.Reducer;
10 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12 import org.apache.hadoop.util.GenericOptionsParser;
13
14 public class WordCount {
15
16     public static class TokenizerMapper
17         extends Mapper<Object, Text, Text, IntWritable>{
18
19         private final static IntWritable one = new IntWritable();
20         private Text word = new Text();
21
22         public void map(Object key, Text value, Context context
23                         throws IOException, InterruptedException {
24             StringTokenizer itr = new StringTokenizer(value.toString());
25             while (itr.hasMoreTokens()) {
26                 word.set(itr.nextToken());
27                 context.write(word, one);
28             }
29         }
30     }
31
32     public static class IntSumReducer
33         extends Reducer<Text,IntWritable,Text,IntWritable> {
34         private IntWritable result = new IntWritable();
35
36         public void reduce(Text key, Iterable<IntWritable> values,
37                           Context context
38                           ) throws IOException, InterruptedException {
39             int sum = 0;
40             for (IntWritable val : values) {
41                 sum += val.get();
42             }
43             result.set(sum);
44             context.write(key, result);
45         }
46     }
47
48     public static void main(String[] args) throws Exception {
49         Configuration conf = new Configuration();
50         String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
51         if (otherArgs.length < 2) {
52             System.err.println("Usage: wordcount <in> <out>");
53             System.exit(2);
54
55         Job job = new Job(conf, "word count");
56         job.setJarByClass(WordCount.class);
57         job.setMapperClass(TokenizerMapper.class);
58         job.setMapOutputKeyClass(Text.class);
59         job.setMapOutputValueClass(IntWritable.class);
60         job.setReducerClass(IntSumReducer.class);
61         job.setOutputKeyClass(Text.class);
62         job.setOutputValueClass(IntWritable.class);
63         FileInputFormat.setInputPaths(job, new Path(otherArgs[0]));
64         FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
65         System.exit(job.waitForCompletion(true) ? 0 : 1);
66     }
67 }
```

### Hive code

```
1 select word, count(*) as word_cnt
2 from wordcount
3 lateral view explode(split(line,' ')) t1 as word
4 group by word
5 order by word_cnt desc
6 limit 10
```

## How about spark code?

## Demo – Word Count Cont.

# Demo - Word Count Cont.

### MR code

```
1 package org.apache.hadoop.examples;
2 import org.apache.hadoop.conf.Configuration;
3 import java.util.StringTokenizer;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.IntWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Mapper;
8 import org.apache.hadoop.mapreduce.Job;
9 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
10 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11 import org.apache.hadoop.util.GenericOptionsParser;
12
13 public class WordCount {
14
15     public static class TokenizerMapper
16         extends Mapper<Object, Text, Text, IntWritable>{
17
18         private final static IntWritable one = new IntWritable();
19         private Text word = new Text();
20
21         public void map(Object key, Text value, Context context
22                         throws IOException, InterruptedException {
23             StringTokenizer itr = new StringTokenizer(value.toString());
24             while (itr.hasMoreTokens()) {
25                 word.set(itr.nextToken());
26                 context.write(word, one);
27             }
28         }
29
30     }
31
32     public static class IntSumReducer
33         extends Reducer<Text,IntWritable,Text,IntWritable> {
34         private IntWritable result = new IntWritable();
35
36         public void reduce(Text key, Iterable<IntWritable> values,
37                           Context context
38                           ) throws IOException, InterruptedException {
39             int sum = 0;
40             for (IntWritable val : values) {
41                 sum += val.get();
42             }
43             result.set(sum);
44             context.write(key, result);
45         }
46     }
47
48     public static void main(String[] args) throws Exception {
49         Configuration conf = new Configuration();
50         String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
51         if (otherArgs.length < 2) {
52             System.err.println("Usage: wordcount <in> <out>");
53             System.exit(2);
54
55         Job job = new Job(conf, "word count");
56         job.setJarByClass(WordCount.class);
57         job.setMapperClass(TokenizerMapper.class);
58         job.setMapOutputKeyClass(Text.class);
59         job.setMapOutputValueClass(IntWritable.class);
60         job.setReducerClass(IntSumReducer.class);
61         job.setOutputKeyClass(Text.class);
62         job.setOutputValueClass(IntWritable.class);
63         FileInputFormat.setInputPaths(job, new Path(otherArgs[0]));
64         FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
65         System.exit(job.waitForCompletion(true) ? 0 : 1);
66     }
67 }
```

### Hive code

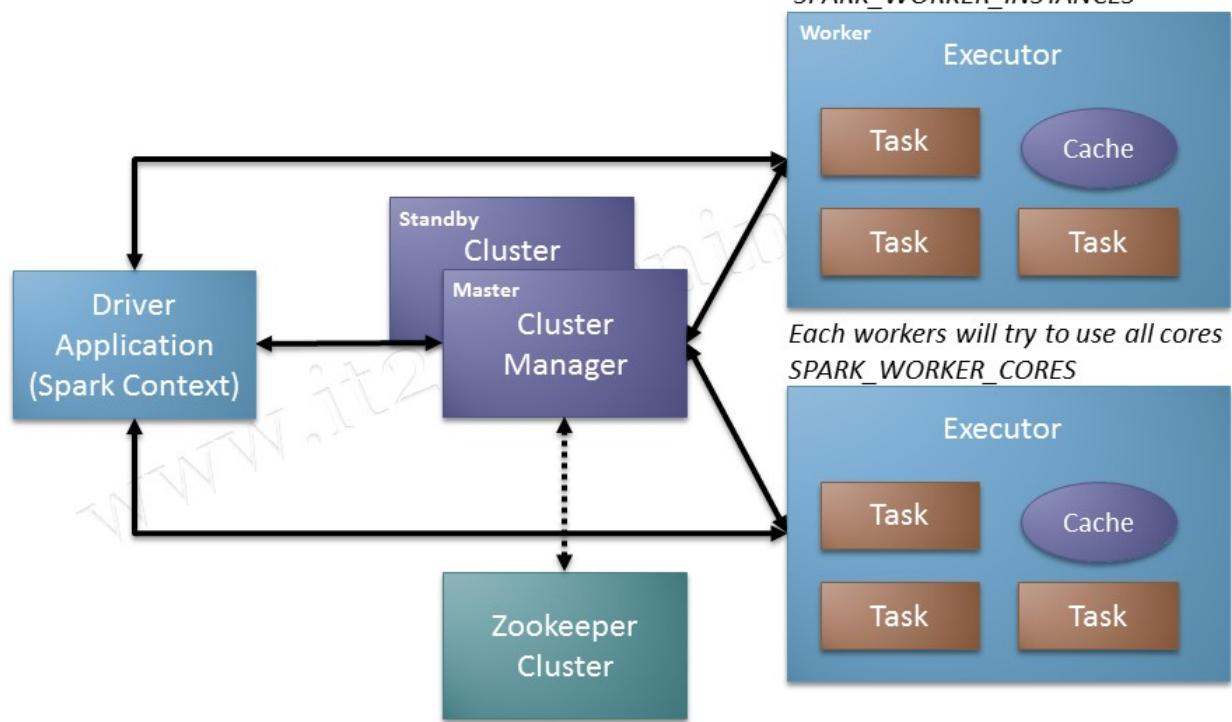
```
1 select word, count(*) as word_cnt
2 from wordcount
3 lateral view explode(split(line,' ')) t1 as word
4 group by word
5 order by word_cnt desc
6 limit 10
```

## How about spark code?

```
1 val wordCounts = sc.textFile("/data/data_words/wordcount.txt").
2 flatMap(_.split(" ")).filter(_.length>1).map(word=>(word,1)).
3 reduceByKey(_+_).map(item => item.swap).sortByKey(false);
4
5 val result = wordCounts.take(10).foreach(x=>println(x));
```

## Spark Architecture

# Spark Architecture

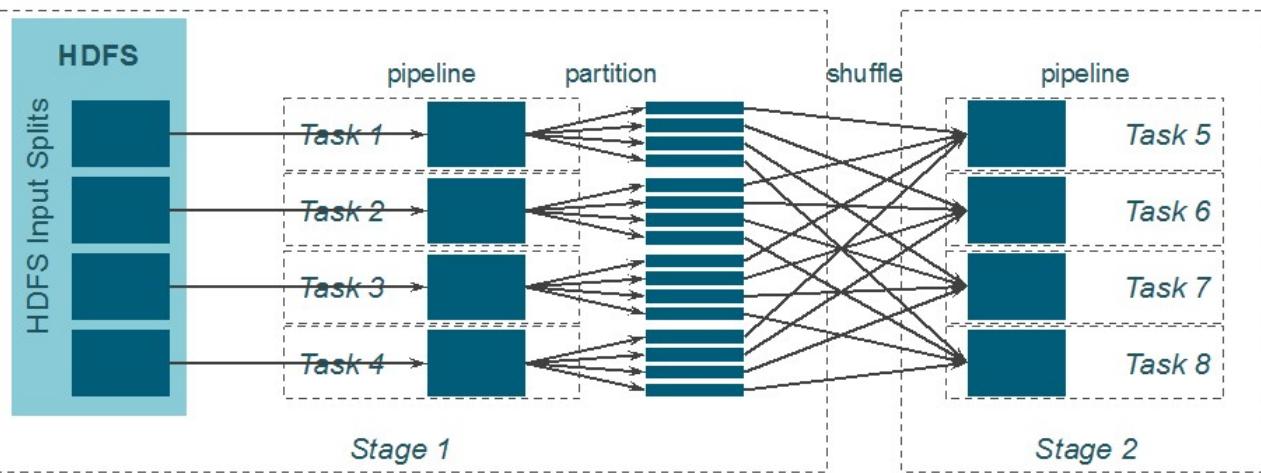


# Spark Core Components

|                        |                                                                                                                                                                                               |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Application</b>     | User program built on Spark. Consists of a driver program and executors on the cluster.                                                                                                       |
| <b>Driver program</b>  | The process running the main() function of the application and creating the SparkContext                                                                                                      |
| <b>Cluster manager</b> | An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN)                                                                                             |
| <b>Worker node</b>     | Any node that can run application code in the cluster                                                                                                                                         |
| <b>Executor</b>        | A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.                         |
| <b>Task</b>            | A unit of work that will be sent to one executor                                                                                                                                              |
| <b>Job</b>             | A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. save, collect); you'll see this term used in the driver's logs.                     |
| <b>Stage</b>           | Each job gets divided into smaller sets of tasks called stages that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs. |

## Example of Partial Word Count

# Example of Partial Word Count



- *Each core by default execute one task a time (spark.task.cpus)*
- *One task per partition, one partition per split*
- *No. of stages is coming from shuffle/wide transformations*

# Two Main Spark Abstractions

**RDD – Resilient Distributed Dataset**

**DAG – Direct Acyclic Graph**



<http://www.it21learning.com>

# RDD Concept

### Simple explanation

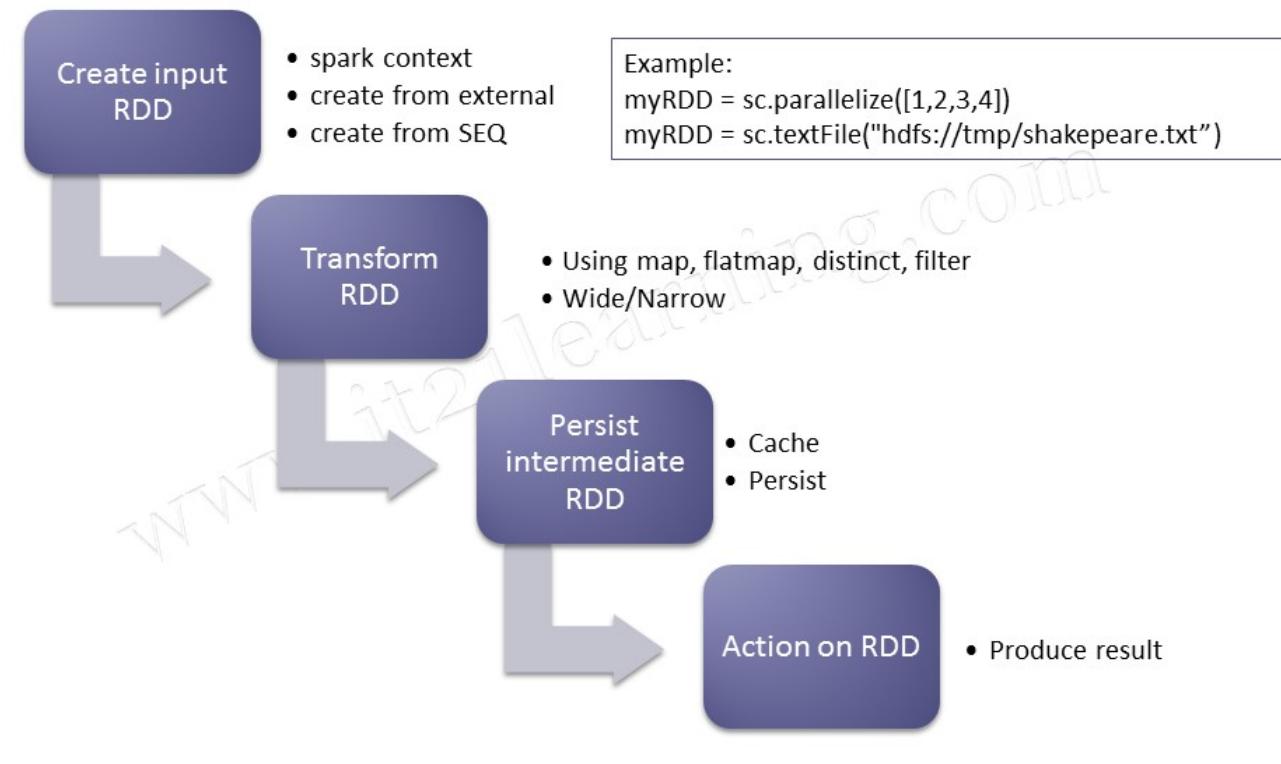
- RDD is collection of data items split into partitions, stored in memory on worker nodes of the cluster, and perform proper operations.

### Complex explanation

- RDD is an interface for data transformation
- RDD refers to the data stored either in persisted store (HDFS, Cassandra, HBase, etc.) or in cache (memory, memory+disks, disk only, etc.) or in another RDD
- Partitions are recomputed on failure or cache eviction

## Spark Program Flow by RDD

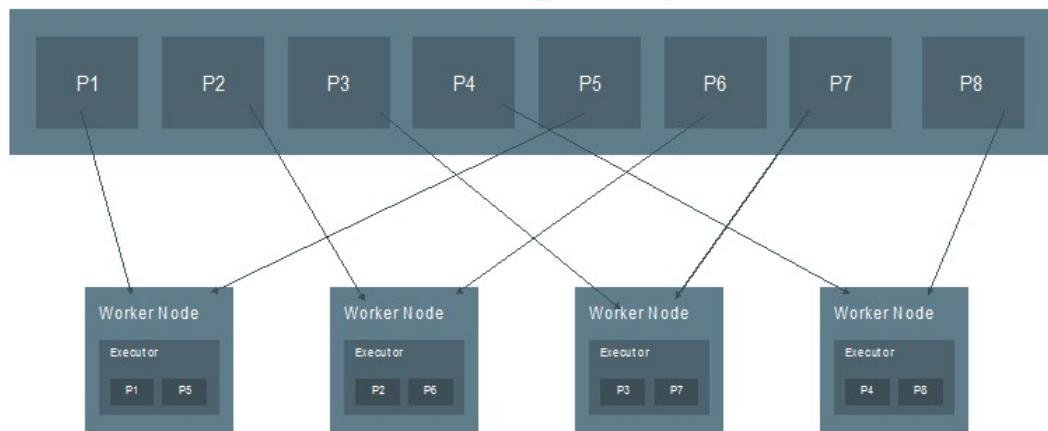
# Spark Program Flow by RDD



## RDD Partitions

# RDD Partitions

- A **Partition** is one of the different chunks that a RDD is splitted on and that is sent to a node
- The more partitions we have, the more **parallelism** we get
- Each partition is candidate to be spread out to different **worker nodes**

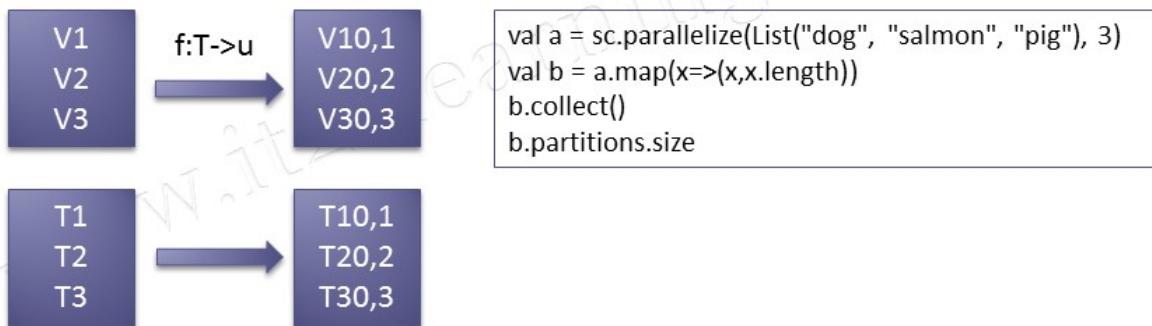


# RDD Operations

- RDD is the only data manipulation tool in Apache Spark
- Lazy and non-lazy operations
- There are two types of RDDs
  - ❖ Transformation – lazy – returns Array[T] by collect action
  - ❖ Actions – non-lazy – returns value
- Narrow and wide transformations

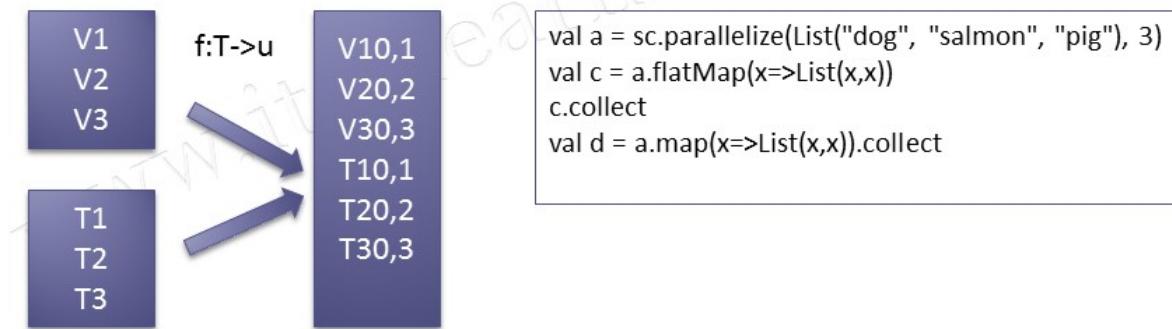
# RDD Transformations - map

- Applies a transformation function on each item of the RDD and returns the result as a new RDD.
- A narrow transformation



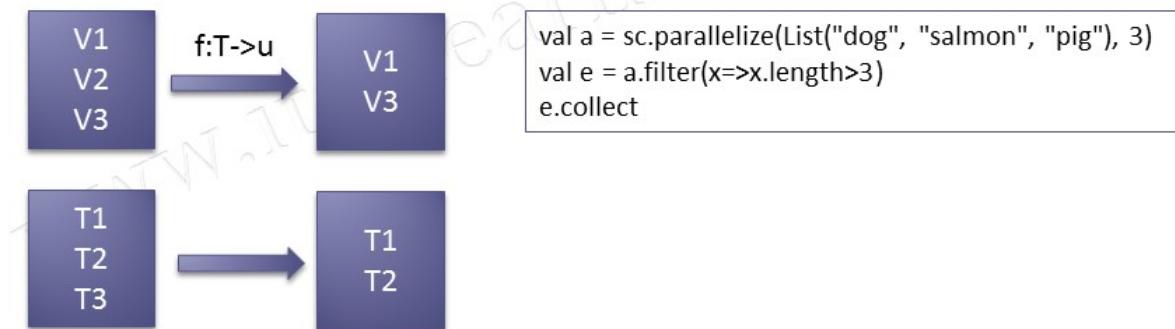
# RDD Transformations - flatMap

- Similar to map, but each input item can be mapped to 0 or more output items.
- A narrow transformation



# RDD Transformations - filter

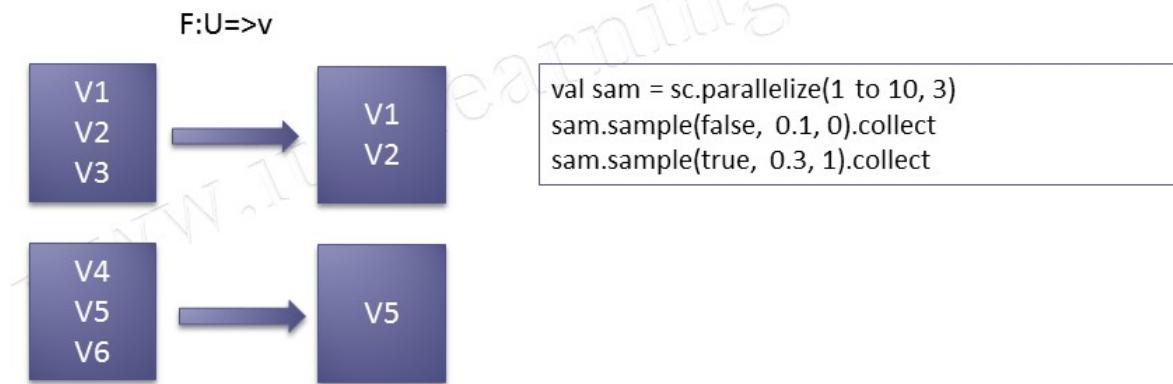
- Create a new RDD by passing in a function used to filter the results. It is similar to where clause in SQL language
- A narrow transformation



# RDD Transformations - sample

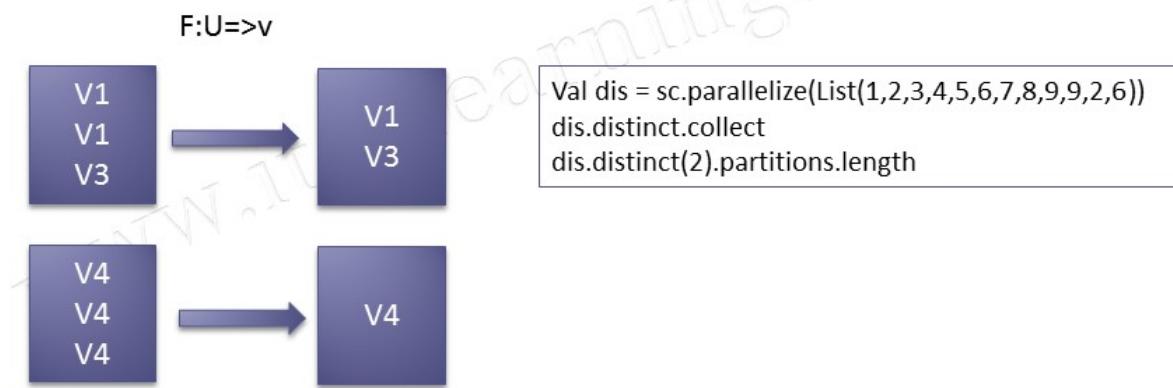
- Return a random sample subset RDD of the input RDD
- A narrow transformation

```
def sample(withReplacement: Boolean, fraction: Double, seed: Int): RDD[T]
```



# RDD Transformations - distinct

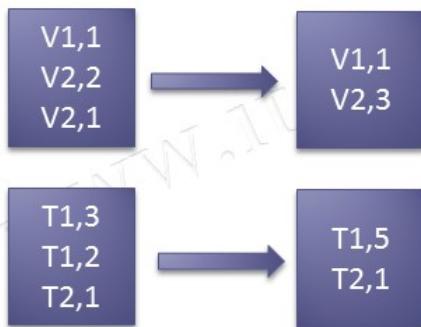
- Return a new RDD with distinct elements within a source RDD with specific partitions
- A narrow/wide transformation



# RDD Transformations - reduceByKey

- Operates on (K,V) pairs of course, but the func must be of type (V,V) => V
- A narrow/wide transformation

$$F(A,B)=(A+B)$$



```

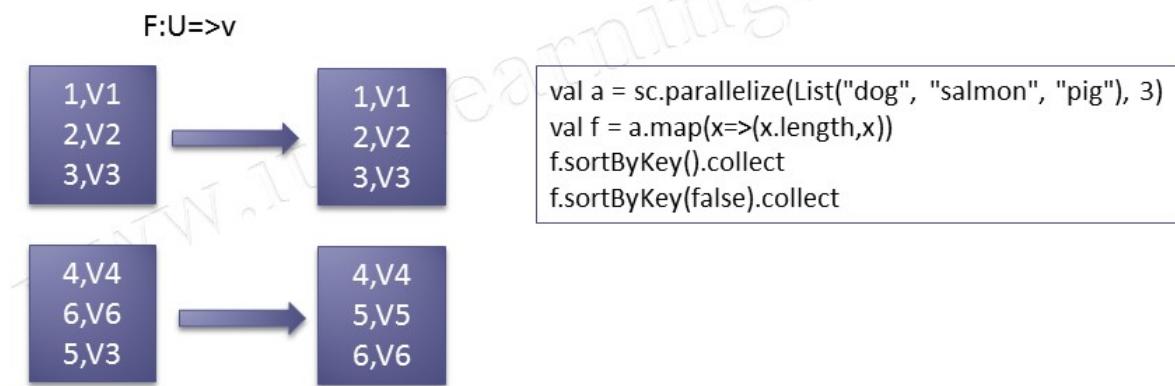
val a = sc.parallelize(List("dog", "salmon", "pig"), 3)
val f = a.map(x=>(x.length,x))
f.reduceByKey((a,b)=>(a+b)).collect
f.reduceByKey(_+_).collect
f.groupByKey.collect

```

*In terms of group, ReduceByKey is more efficient with combiner in the map phase.*

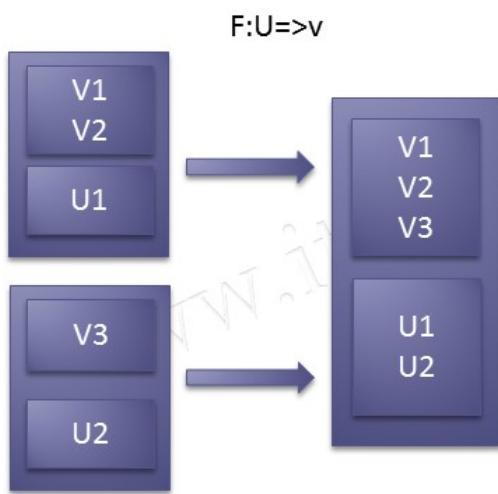
# RDD Transformations - sortByKey

- This simply sorts the (K,V) pair by K. Default is asc. False as desc.
- A narrow/wide transformation



# RDD Transformations - union

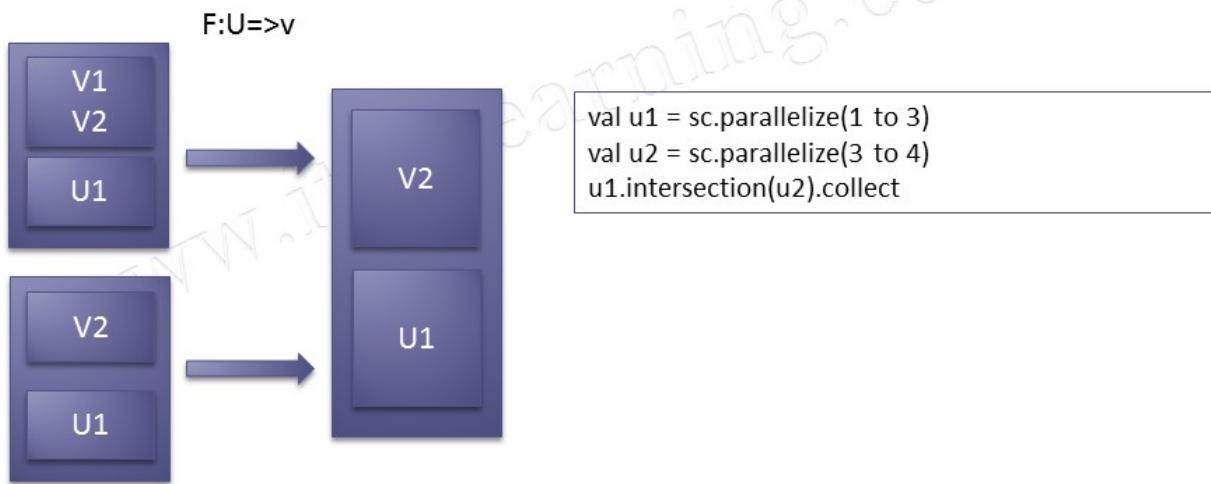
- Returns the union of two RDDs to a new RDD
- A narrow transformation



```
val u1 = sc.parallelize(1 to 3)
val u2 = sc.parallelize(3 to 4)
u1.union(u2).collect
(u1 ++ u2).collect
```

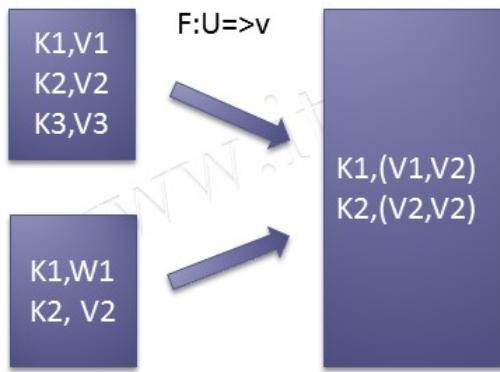
# RDD Transformations - intersection

- Returns a new RDD containing only common elements from source RDD and argument.
- A narrow/wide transformation



# RDD Transformations - join

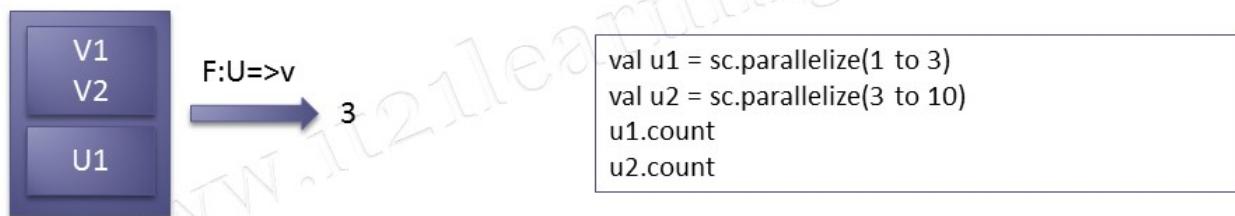
- When invoked on (K,V) and (K,W), this operation creates a new RDD of (K, (V,W)). Must apply on key-value pairs.
- A wide transformation



```
val j1 = sc.parallelize(List("abe", "abby",  
"apple")).map(a => (a, 1))  
val j2 = sc.parallelize(List("apple", "beatty",  
"beatrice")).map(a => (a, 1))  
j1.join(j2).collect  
j1.leftOuterJoin(j2).collect  
j1.rightOuterJoin(j2).collect
```

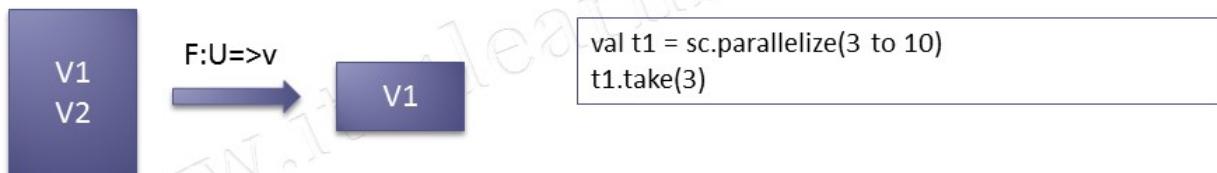
# RDD Actions - count

- Get the number of data elements in the RDD



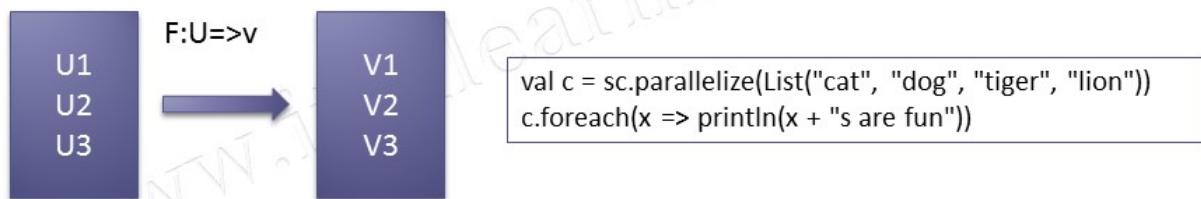
# RDD Actions - take

- Fetch number of data elements from the RDD



# RDD Actions - foreach

- Execute function for each data element in RDD.



# RDD Actions - saveAsTextFile

- Writes the content of RDD to a text file or a set of text files to local file system/HDFS. The default path is HDFS or hdfs://

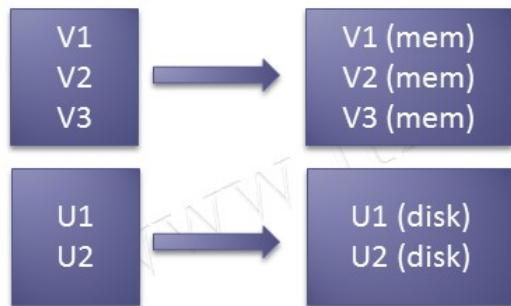
```
val a = sc.parallelize(1 to 10000, 3)
a.saveAsTextFile("file:/Users/will/Downloads/mydata")
```

```
[root@sandbox Downloads]# ls -al /Users/will/Downloads/mydata
total 80
drwxr-xr-x 2 root root 4096 Dec  6 02:48 .
drwxr-xr-x 3 root root 4096 Dec  6 02:48 ..
-rw-r--r-- 1 root root     8 Dec  6 02:48 __SUCCESS.crc
-rw-r--r-- 1 root root   132 Dec  6 02:48 .part-00000.crc
-rw-r--r-- 1 root root   140 Dec  6 02:48 .part-00001.crc
-rw-r--r-- 1 root root   140 Dec  6 02:48 .part-00002.crc
-rw-r--r-- 1 root root      0 Dec  6 02:48 __SUCCESS
-rw-r--r-- 1 root root 15558 Dec  6 02:48 part-00000
-rw-r--r-- 1 root root 16665 Dec  6 02:48 part-00001
-rw-r--r-- 1 root root 16671 Dec  6 02:48 part-00002
```

# RDD Persistence - cache/persist

- Cache data to memory/disk. It is a lazy operation
- cache = persist(MEMORY)

`persist(MEMORY_AND_DISK)`



```
val u1 = sc.parallelize(1 to 1000).filter(_%2==0)
u1.cache()
u1.count
```

## Storage Level for persist

# Storage Level for persist

| Storage Level                             | Purpose                                                                                                                                                                                                                                          |
|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MEMORY_ONLY (Default level)               | Keep RDD in available cluster memory as deserialized Java objects. Some partitions may not be cached if there is not enough cluster memory. Those partitions will be recalculated on the fly as needed.                                          |
| MEMORY_AND_DISK                           | This option stores RDD as deserialized Java objects. If RDD does not fit in cluster memory, then store those partitions on the disk and read them as needed.                                                                                     |
| MEMORY_ONLY_SER                           | This option stores RDD as serialized Java objects (One byte array per partition). This is more CPU intensive but saves memory as it is more space efficient. Some partitions may not be cached. Those will be recalculated on the fly as needed. |
| MEMORY_ONLY_DISK_SER                      | This option is same as above except that disk is used when memory is not sufficient.                                                                                                                                                             |
| DISC_ONLY                                 | This option stores the RDD only on the disk                                                                                                                                                                                                      |
| MEMORY_ONLY_2,<br>MEMORY_AND_DISK_2, etc. | Same as other levels but partitions are replicated on 2 slave nodes                                                                                                                                                                              |

*By default, spark uses algorithm of Least Recently Used (LRU) to remove old and unused RDD to release more memory. You can also manually remove unused RDD from memory by using **unpersist()***

# Spark Variable - Broadcast

- Like Hadoop DC, broadcast variables let programmer keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. For example, to give every node a copy of a large input dataset efficiently
- Spark also attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost

```
val broadcastVar = sc.broadcast(Array(1, 2, 3))
broadcastVar.value
```

```
val fn = sc.textFile("hdfs://filename").map(_.split("|"))
val fn_bc = sc.broadcast(fn)
```

# Spark Variable - Accumulators

- Accumulators are variables that can only be “added” to through an *associative* operation Used to implement counters and sums, efficiently in parallel
- Spark natively supports accumulators of numeric value types and standard mutable collections, and programmers can extend for new types
- Only the driver program can read an accumulator’s value, not the tasks

```
val accum = sc.accumulator(0)
sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)
accum.value
```

## Revisit word count demo

37



# Revisit word count demo

Self Lab



<http://www.it21learning.com>

37

# Two Main Spark Abstractions

RDD – Resilient Distributed Dataset

**DAG – Direct Acyclic Graph**



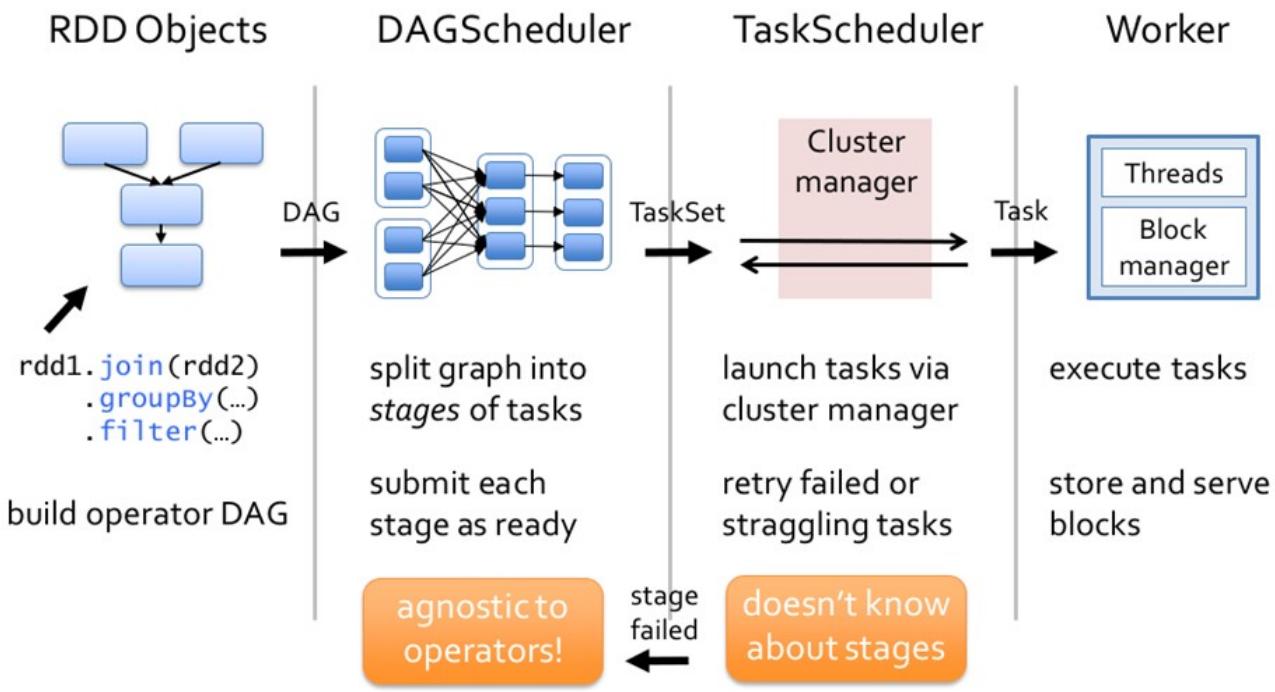
<http://www.it21learning.com>

# DAG

- ***Direct Acyclic Graph*** – sequence of computations performed on data
  - ❖ *Node*: RDD partition
  - ❖ *Edge*: Transformation on top of data
  - ❖ *Acyclic*: Graph cannot return to the older partition
  - ❖ *Direct*: transformation is an action that transitions data partition state (from A to B)

## How Spark Works by DAG

# How Spark Works by DAG



The first layer is the interpreter, Spark uses a Scala interpreter, with some modifications. As you enter your code in spark console (creating RDD's and applying operators), Spark creates a operator graph. When the user runs an action (like collect), the Graph is submitted to a DAG Scheduler. The DAG scheduler divides operator graph into (map and reduce) stages. A stage is comprised of tasks based on partitions of the input data. The DAG scheduler pipelines operators together to optimize the graph. For e.g. Many map operators can be scheduled in a single stage. This optimization is key to Sparks performance. The final result of a DAG scheduler is a set of stages. The stages are passed on to the Task Scheduler. The task scheduler launches tasks via cluster manager. (Spark Standalone/Yarn/Mesos). The task scheduler doesn't know about dependencies among stages. The Worker executes the tasks. A new JVM is started per job. The worker knows only about the code that is passed to it.

# Spark SQL

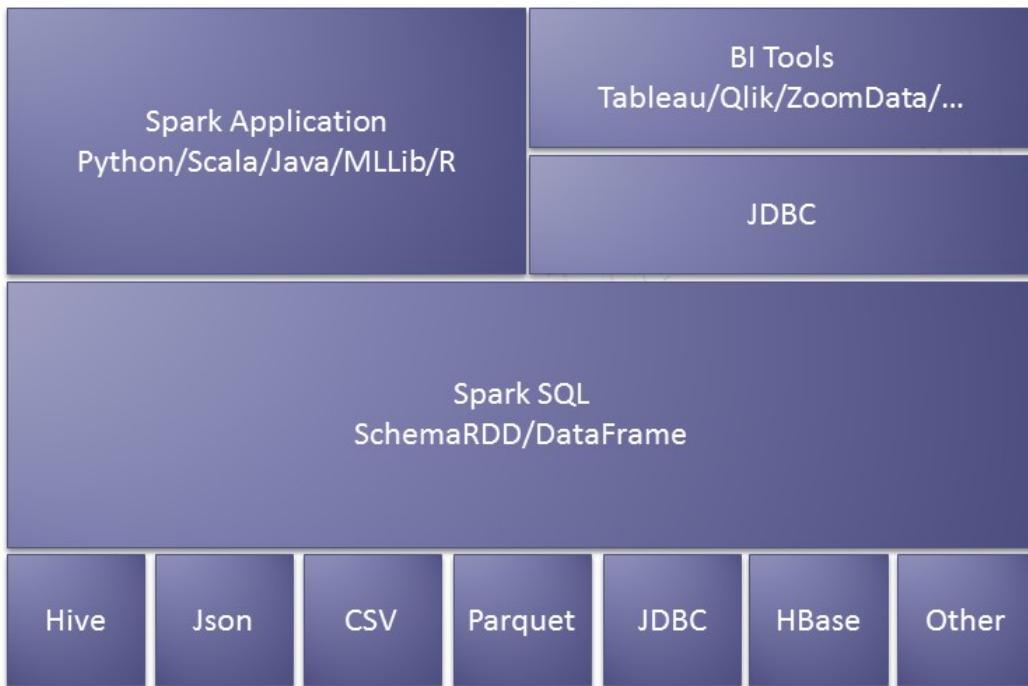
SQL over Apache Spark



<http://www.it21learning.com>

## Apache Spark SQL Stack

# Apache Spark SQL Stack



# Spark SQL History

*Stop at mid of 2014*

Shark

Development ending;  
transitioning to Spark SQL

*Released at  
April of 2014*

Spark SQL

A new SQL engine designed  
from ground-up for Spark

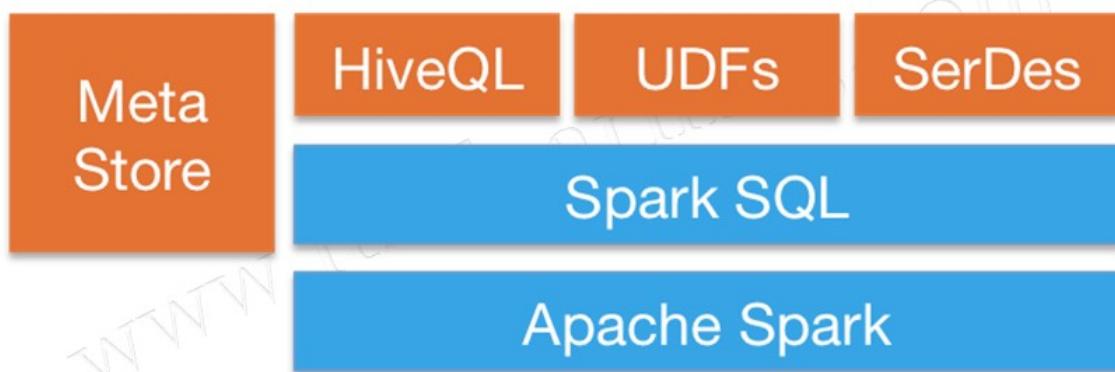
*Ready at  
Beginning of 2015*

Hive on Spark

Help existing Hive users  
migrate to Spark

# Spark SQL with Hive Meta

*Support almost all hive query – HiveContext, which is superset of SQLContext  
SparkSQL uses Hive's meta store, but his own version of thrift server*



# Spark SQL with BI Tools

*Leverage existing BI Tools*

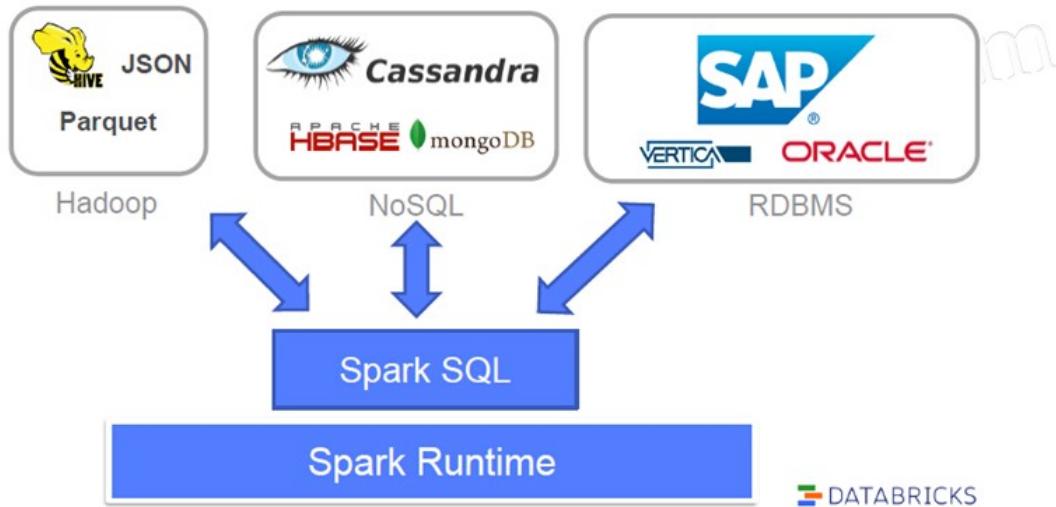
BI Tools

JDBC / ODBC

Spark SQL

# Spark SQL with Other Integrations

Will facilitate deeper integration with other systems



*Picture from databricks.com*

# Spark SQL Supported Keywords

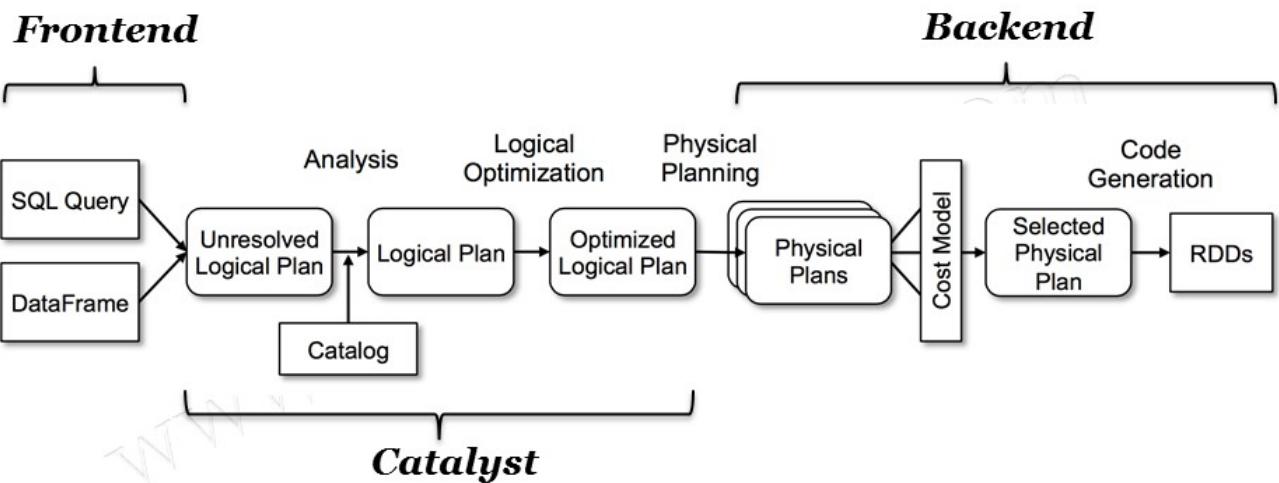
*Support SQL 92*

|                |             |             |
|----------------|-------------|-------------|
| map            | reduce      | sample      |
| filter         | count       | take        |
| groupBy        | fold        | first       |
| sort           | reduceByKey | partitionBy |
| union          | groupByKey  | mapWith     |
| join           | cogroup     | pipe        |
| leftOuterJoin  | cross       | save        |
| rightOuterJoin | zip         | ...         |

*Picture from databricks.com*

## Spark SQL – How It Works

# Spark SQL - How It Works



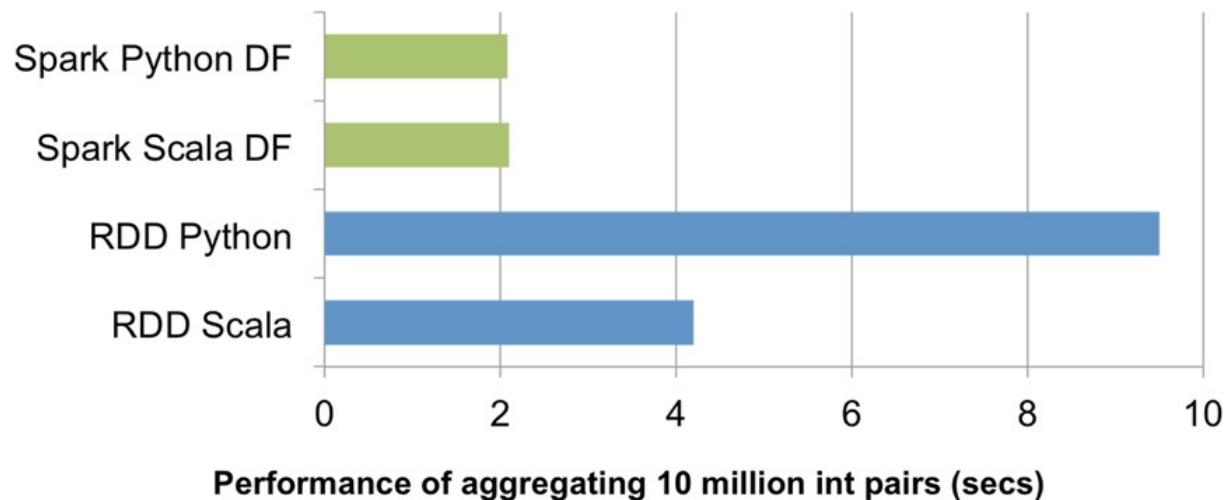
Optimizing query planning for Spark Takes Dataframe operations and ‘compiles’ them down to RDD operations Often faster than writing RDD code manually Use Dataframes whenever possible (v1.4+)

# DataFrame (DF) RDD

- Schema on read
- Works with structured and semi-structured data
- Simplifies working with structured data
- Keep track of schema information
- Read/Write from structure data like JSON, Hive tables, Parquet, etc.
- SQL inside your Spark App
- Best Performance and more powerful operations API
- Access through SQLContext
- Integrated with ML

Works with structured and semi-structured data  
DataFrame simplifies working with structured data  
Read/Write from structure data like JSON, Hive tables, Parquet, etc.  
SQL inside your Spark App  
Best Performance and more powerful operations API

# DF - Performance



*Picture from databricks.com*

# DF Example - Direct from files

*SqlContext directly deals with DF*

```
val df = sqlContext.read.json("/demo/data/data_json/customer.json")
df: org.apache.spark.sql.DataFrame = [age: bigint, name: string]
df.printSchema()
scala> df.printSchema
root
| -- age: long (nullable = true)
| -- name: string (nullable = true)

df.show()
+---+-----+
|age|  name|
+---+-----+
21	Mike
38	Ted
35	Will
35	Lily
42	Joy
13	Michael
33	Bob
33	Lucy
+---+-----+
df.select("name").show
df.filter(df("age") >= 35).show()
df.groupBy("age").count().show()
```

## DF Example – Direct from Hive

# DF Example - Direct from Hive

*SqlContext directly deals with DF*

```
val a = sqlContext.table("sample_07")
a: org.apache.spark.sql.DataFrame = [code: string, description: string, total_emp: int, salary: int]
a.printSchema()
root
 |-- code: string (nullable = true)
 |-- description: string (nullable = true)
 |-- total_emp: integer (nullable = true)
 |-- salary: integer (nullable = true)

df.show(1)
+-----+-----+-----+
| code|description|total_emp|salary|
+-----+-----+-----+
|00-0000>All Occupations|134354250| 40690|
+-----+-----+-----+
```

# DF Example - Indirect from Other RDD

*SqlContext directly deals with DF*

```
val rdd = sc.textFile("/demo/data/data_cust/data1.csv").  
flatMap(_.split(",")).map(x=>(x,1)).reduceByKey(_+_)

//auto schema  
val df1 = sqlContext.createDataFrame(rdd)  
val df2 = rdd.toDF

//specify schema  
val schemaString = "word cnt"  
val schema = StructType(StructField("word", StringType, true) :: StructField("cnt",  
IntegerType, false) :: Nil)  
val rowRDD = rdd.map(p => Row(p._1,p._2))  
val dfcust = sqlContext.createDataFrame(rowRDD, schema)
```

# Spark SQL Shell Demo

Self Practice



<http://www.it21learning.com>

**Thank You**

55

*Spark 1.5=>1.6*

**Thank You**

Will Du  
[sparkera.ca](http://sparkera.ca)



<http://www.it21learning.com>

55

## Hadoop Data Model & Application Architecture

# Hadoop Data Lake

Wayne Shen



<http://www.it21learning.com>

<http://www.slideshare.net/StampedeCon/choosing-an-hdfs-data-storage-format-avro-vs-parquet-and-more-stampedecon-2015>

# Hadoop Data Model - Text

- Text files normally come in CSV, JSON, Fixed-Length Flat formats
- Convenient formats to use to exchange with other applications or scripts that produce or read delimited files.
- Human readable and parsable
- Data stores is bulky and not as efficient to query
- Do not support block compression

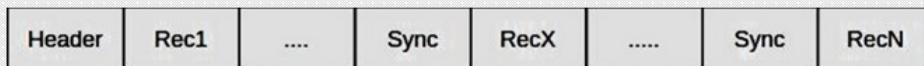
# Hadoop Data Model - SequenceFile (1)

- SequenceFile:
  - Provide a persistent data structure for binary key-value pairs;
  - Row based;
  - Commonly used to transfer data between MapReduce jobs;
  - Can be used as an archive to pack small files in Hadoop;
  - Support splitting even when the data is compressed.
- Types
  - Key Class Type
  - Value Class Type

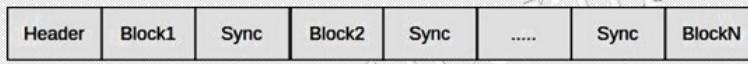
# Hadoop Data Model - SequenceFile (2)

- SequenceFile Format

- Record Format

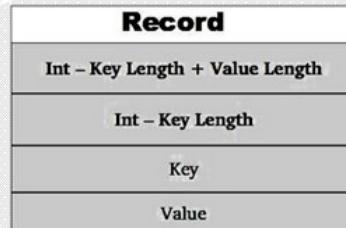


- Block Format



- Compression:

- Record Level
    - Block Level



- JavaAPI:

- `org.apache.hadoop.io.SequenceFile`

| <b>Sequence File Header</b>                            |  |
|--------------------------------------------------------|--|
| 3 Byte (SEQ) + 1 Byte (Version)<br>(e.g. SEQ4 or SEQ6) |  |
| Text – Key Class Name                                  |  |
| Text – Value Class Name                                |  |
| Boolean - Is Compressed                                |  |
| Boolean – Is blockCompressed                           |  |
| CompressionCodec Class Name                            |  |
| MetaData                                               |  |
| Sync Marker                                            |  |

## Hadoop Data Model - SequenceFile (3)

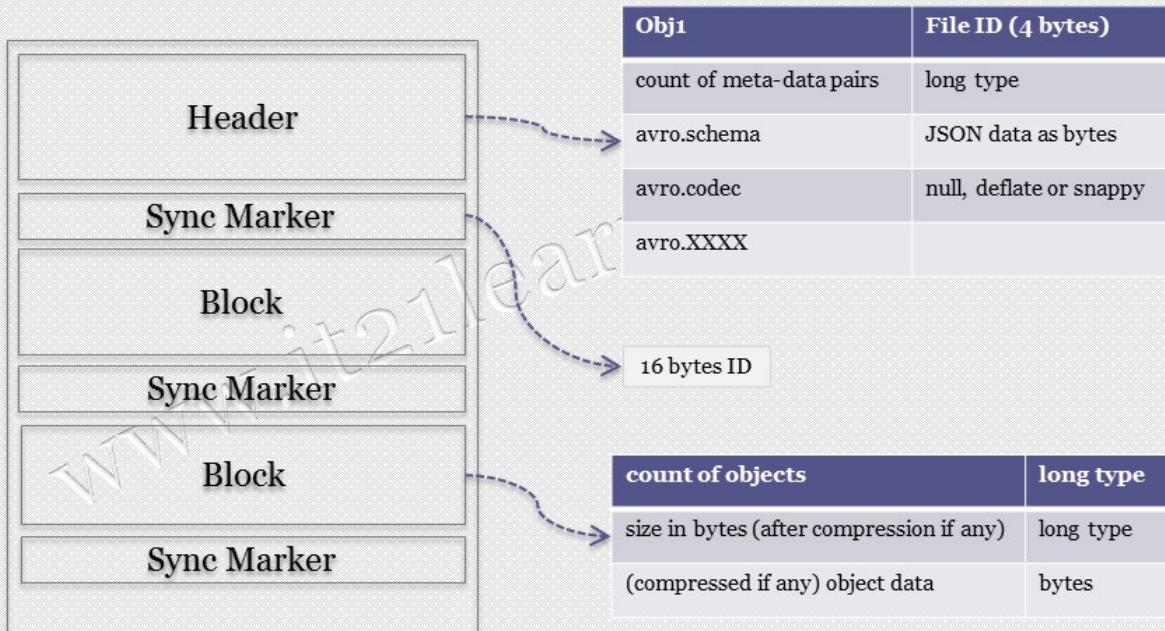
- Sequence Files are not an optimal solution for Hive since it saves a complete row as a single binary value. Hive has to read a full row and decompress it even if only one column is being requested.
- Load SequenceFile with Apache Pig:
  - REGISTER /usr/lib/pig/piggybank.jar;
  - DEFINE SequenceFileLoader org.apache.pig.piggybank.storage.SequenceFileLoader();
  - log = LOAD '/data/twitters' USING SequenceFileLoader AS (...)

# Hadoop Data Model - AVRO (1)

- AVRO File
  - Stores data-definition in JSON
  - Store data in binary format
  - Support Data Types:
    - Primitive types – null, Boolean, int, long, float, double, bytes, strings, and binary data;
    - Complex types – arrays, maps, enumerations, records
  - Schema Evolution:
    - Cleanly handles schema changes like missing fields, added fields and changed fields;
    - Old programs can read new data and new programs can read old data;

# Hadoop Data Model - AVRO (2)

- AVRO File Format



# Hadoop Data Model - AVRO (2)

- Hive Table with AVRO

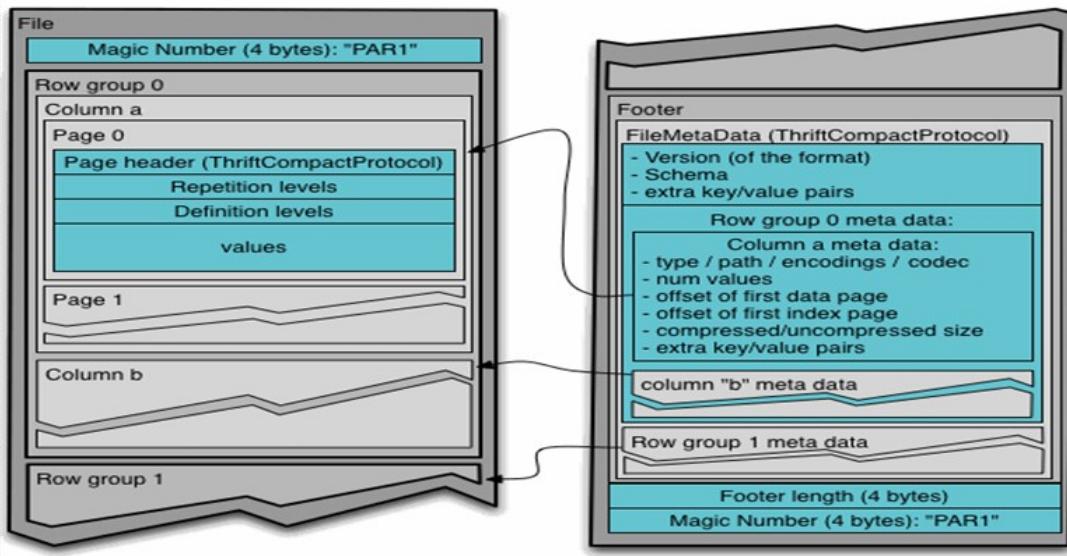
```
hive> create table CUSTOMERS
> row format serde 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
> stored as inputformat 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
> outputformat 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
> tblproperties ('avro.schema.literal'='{
    > "name": "customer", "type": "record",
    > "fields": [
        > {"name": "firstName", "type": "string"},
        > {"name": "lastName", "type": "string"},
        > {"name": "age", "type": "int"},
        > {"name": "salary", "type": "double"},
        > {"name": "department", "type": "string"},
        > {"name": "title", "type": "string"},
        > {"name": "address", "type": "string"}]}');
```

# Hadoop Data Model - Parquet (1)

- Parquet File
  - Columnar storage format for Hadoop.
  - Uses the record shredding and assembly algorithm described in the Dremel paper
  - Each data file contains the values for a set of rows;
  - Efficient in terms of disk I/O when specific columns need to be queried.
- Types:
  - Primitives: BOOLEAN, INT32, INT64, INT96, FLOAT32, DOUBLE64, BYTE\_ARRAY (string)
  - Logical Types

# Hadoop Data Model - Parquet (2)

- Parquet File Format



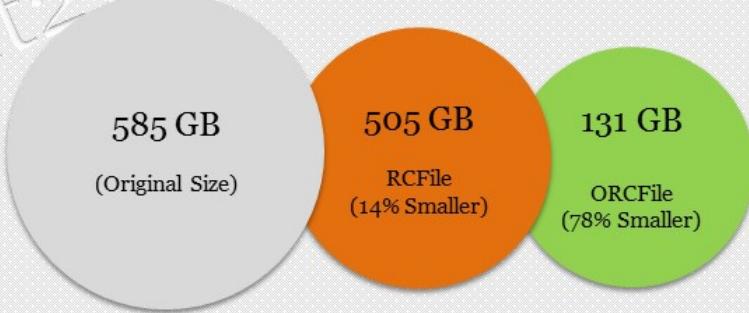
- Format details - <https://parquet.apache.org/documentation/latest/>

# Hadoop Data Model - Parquet (3)

- Hive table with Parquet File(s)
  - `CREATE EXTERNAL TABLE parquet_table_name (x INT, y STRING)  
ROW FORMAT SERDE 'parquet.hive.serde.ParquetHiveSerDe'  
STORED AS  
INPUTFORMAT "parquet.hive.DeprecatedParquetInputFormat"  
OUTPUTFORMAT "parquet.hive.DeprecatedParquetOutputFormat"  
LOCATION '/hive-warehouse/twitters';`
- Pig Manipulation of Parquet File(s)
  - `A = LOAD '/data/twitters'  
USING parquet.pig.ParquetLoader AS (x: INT, y STRING);`
  - `B = LIMIT A 100;`
  - `DUMP B;`

# Hadoop Data Model - RC & ORC (1)

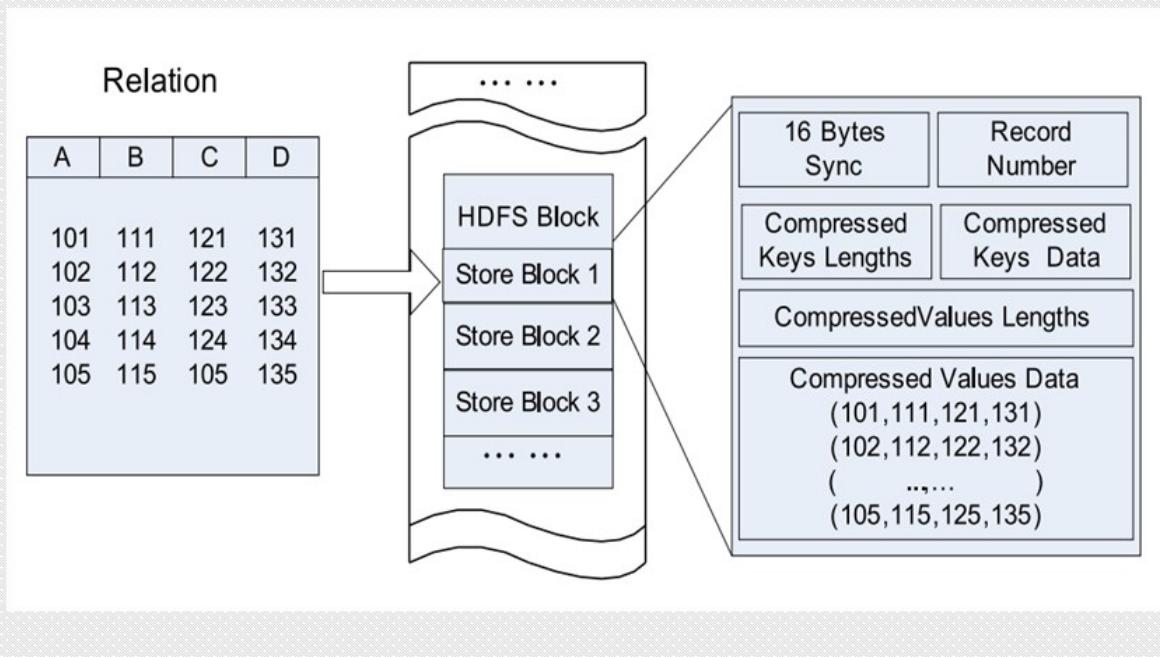
- Record Columnar / Optimized Row Columnar
  - Stores collections of rows and within the collection the row data is stored in columnar format
  - Introduces a lightweight indexing that enables skipping of irrelevant blocks of rows;
  - Splittable: allows parallel processing of row collections;



Additionally, compression on a column base is more efficient. It can take advantage of similarity of the data in a column.

# Hadoop Data Model - RC & ORC (2)

- RCFile Format



# Hadoop Data Model - RC & ORC (3)

- Hive table with RC/ORC

```
CREATE TABLE Twitters
(
    tweet_id BIGINT,
    user_name STRING,
    tweet STRING,
    time_stamp STRING
)
COMMENT 'This is the Twitter streaming data'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS RCFILE;
```

- Pig Manipulation of RC/ORC files

```
A = LOAD '/user/data/twitters' USING OrcStorage();
```

## File Format Comparison (1)

# File Format Comparison (1)

| FileType     | Splittable | Block Compressible | Schema Evolution* | Hive | Pig | Remark           |
|--------------|------------|--------------------|-------------------|------|-----|------------------|
| Text/CSV     | Yes        | No                 | No                | Yes  | Yes |                  |
| XML/JSON     | No         | No                 | Yes               | ?    | Yes |                  |
| AvroFile     | Yes        | Yes                | Yes               | Yes  | Yes |                  |
| SequenceFile | Yes        | Yes                | Yes               | ?    | Yes |                  |
| RCFile       | Yes        | Yes                | No                | Yes  | Yes | Columnar Storage |
| ORCFile      | Yes        | Yes                | No                | Yes  | Yes | Columnar Storage |
| ParquetFile  | Yes        | Yes                | Yes               | Yes  | Yes | Columnar Storage |

\* → Schema Evolution means schema used to read a file does not need to match the schema used to write the file.

# File Format Comparison (2)

- Normally write time is not significant concern.
- For Read:
  - Avro – Query datasets that have changed over time
  - Parquet – Query a few columns on a wide table.
  - Parquet & ORC optimize read performance at the cost of write performance
  - Text is slow to read.
  - Hive Query (Fastest → Slowest)
    - ORC → Parquet/SequenceFile → Text → Avro

## Data Compression in Hadoop

# Data Compression in Hadoop

| Compression | Codec                                      | Extension | Splittable |
|-------------|--------------------------------------------|-----------|------------|
| Deflate     | org.apache.hadoop.io.compress.DefaultCodec | .deflate  | No         |
| Gzip        | org.apache.hadoop.io.compress.BzipCodec    | .gz       | No         |
| Bzip2       | org.apache.hadoop.io.compress.BZip2Codec   | .gz       | Yes        |
| LZO         | org.apache.hadoop.io.compress.LzopCodec    | .lzo      | No         |
| LZ4         | org.apache.hadoop.io.compress..Lz4Codec    | .lz4      | No         |
| Snappy      | org.apache.hadoop.io.compress.SnappyCodec  | .snappy   | No         |

www.it21learning.com

# What is Data Lake?

- A **data lake** is a large storage repository and processing environment.
- A **data lake** is an enterprise-wide data management platform for analyzing disparate sources of data in their native format.
- Data Lake's Three Key Attributes:
  - **Collect everything** - A Data Lake contains all data, both **raw** sources over extended periods of time as well as any **processed** data.
  - **Dive in anywhere** - A Data Lake enables users across multiple business units to refine, explore and enrich data on their terms.
  - **Flexible access** - A Data Lake enables multiple data access patterns across a shared infrastructure: batch, interactive, online, search, in-memory and other processing engines.

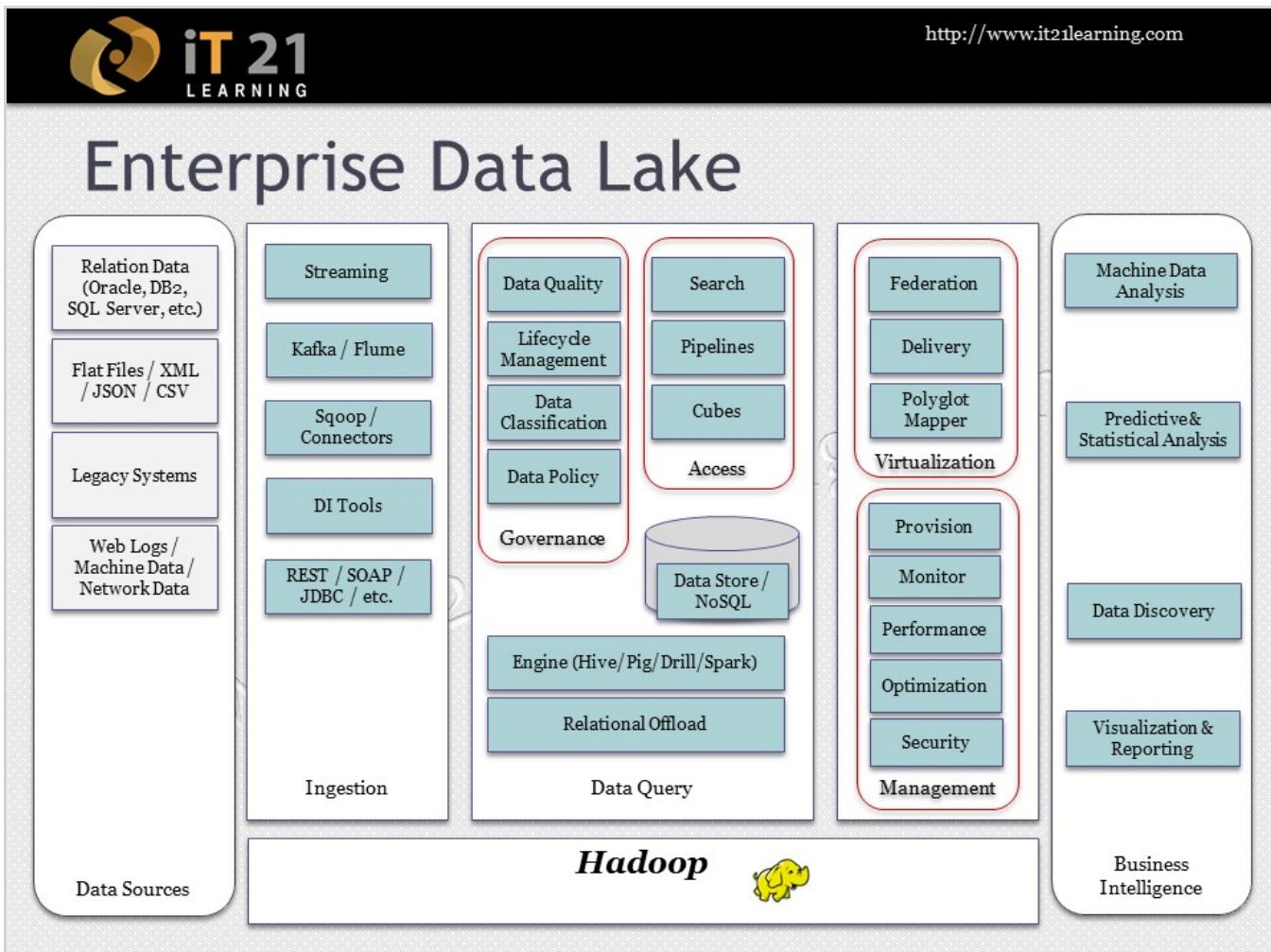
# Why EDL, not EDW?

- EDW (Enterprise Data Warehouse) has served as the foundation for business intelligence and data discovery, but
  - Falling out-of-sync with times
  - Predictable, restricted in scope and ability
  - Cost-intensive
  - Unable to handle data complexities
- EDL (Enterprise Data Lake) offers a strategic, adaptive and intuitive data management and analysis platform:
  - Cheap massive data storage
  - Super power of data processing
  - Data governance & integration
  - **Schema on Read** – *discovering unique insights is only possible when data is not limited by structures.*

# EDL Benefits

- The Active Archive
  - Provide a unified data storage system to store all Enterprise data, in any format, at any volume, indefinitely.
  - Address compliance requirement
  - Deliver data on demand to satisfy internal and external regulatory demands;
- Self-Service Exploratory Business Intelligence
  - Allow to explore data with full security by using traditional interactive business intelligence tools.
- Advanced Analytics
  - Offer the capacity to search or analyze data on a large scale and at a microscopic or granular level for effective analysis.

## Enterprise Data Lake



Apache Kafka is HDFS based Message Queuing; Apache Drill is a SQL engine for Hadoop, NoSQL and Cloud Storage. A query can join a HDFS file, with data from NoSQL such as MongoDB etc. Cubes – Cubes functions; Relational Offload – Tools and solutions to help enterprises offload expensive computing from relational data warehouses to big data warehouses;

# EDL Components (1)

- **Data Sources:**

- Relational database (such as Oracle, DB2, PostgreSQL, SQL Server and the like);
- Multiple disparate, unstructured and semi-structured data sources in formats such as flat files, XML, JSON or CSV;
- Integration data in EDI or other B2B exchange formats;
- Machine data and network elements for voluminous data generation.

- **Hadoop Distribution:**

- Most popular choice for big data today, Hadoop is available in open source Apache and commercial distribution packages;
- Consists of a file system called HDFS – the key data storage layer of the big data warehouse.

# EDL Components (2)

- **Data Ingestion:**

- For data assimilation from various sources to the Hadoop file system – reliable and scalable data ingestion mechanisms;
- For connecting relational database – Sqoop and database-specific connectors;
- For data streaming – Apache Kafka (**HDFS-based message queue**) and Flume;
- For topology creation of streaming source, ingestion, in flight transformation and data persistence – common CEP (Complex Event Processing) or streaming engines such as Apache Storm or StreamAnalytic;
- For existing Data Integration (DI) connector creation – custom scripts for integration using REST, SOAP, or JDBC components.

# EDL Components (3)

- **Data Query:**
  - For the data resident in HDFS, a multitude of query engines such as Pig, Hive, Apache Drill (**Schema-free SQL Query Engine for Hadoop, NoSQL and Cloud Storage**) and Spark SQL utilized.
  - Tools and solutions (from organization like impetus Technologies) to help enterprises offload expensive computing from relational data warehouses to big data warehouses.
- **Data Stores:**
  - Data-store coupling or NoSQL database like HBase, Cassandra in the big data warehouse for additional functions and capabilities.

# EDL Components (4)

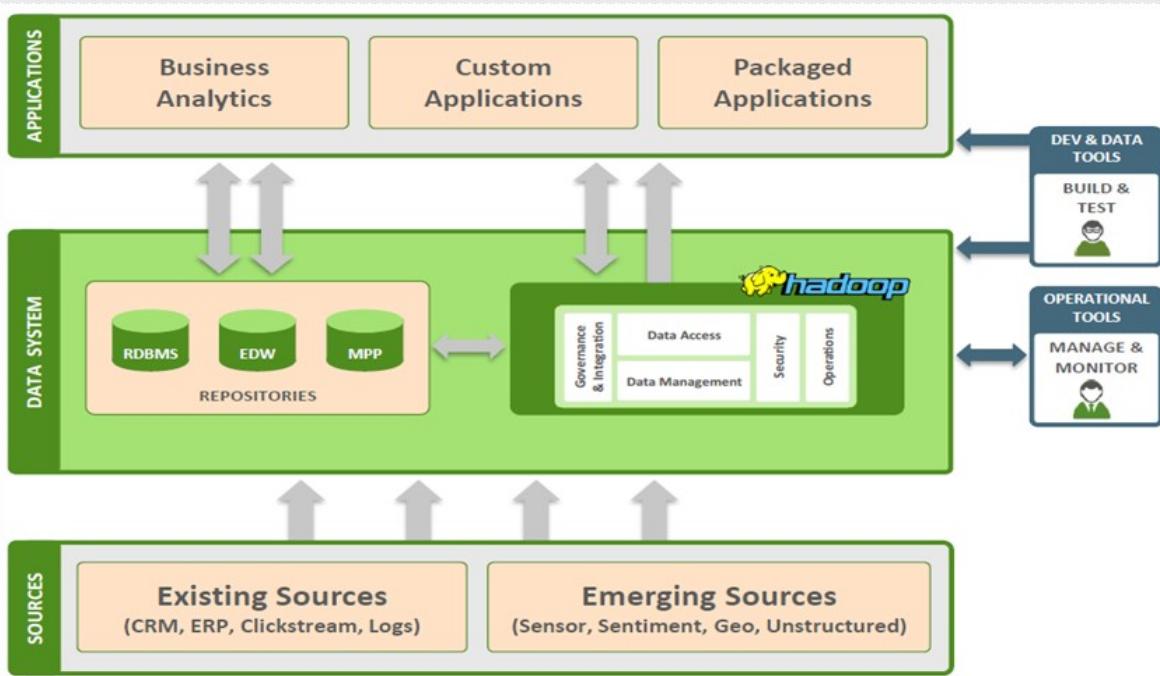
- **Access:**
  - Complex access requirement management – includes features from the traditional world like search and cubes functions
  - New tools to manage complex job pipeline (output of one query may be fed as input into another)
- **Governance:**
  - Data quality maintenance;
  - Data quality regulation adherence;
  - Data lifecycle management;
  - Comprehensive data classification;
  - Enterprise level information policy definition.

# EDL Components (5)

- **Visualization:**
  - For consistent results with appropriate polyglot querying, data and delivery mechanism federation maintenance.
- **Management:**
  - Cluster monitoring – via tools and dashboards for cluster management;
  - Optimal speed and minimal resource consumption – via MapReduce jobs and query performance diagnosis;
- **Business Intelligence:**
  - Various visualization and reporting tool deployment;
  - Data pattern discovery using predictive/statistical algorithms and machine data analytics.

## A Modern Data Architecture

# A Modern Data Architecture



As data continues to grow exponentially, then Enterprise Hadoop and EDW can provide a strategy for both efficiency in a modern data architecture, and opportunity in an Enterprise Data Lake

## Any Question?

# Any Question?



## **Appendix A - Hadoop Ecosystem**

# Hadoop Ecosystem Overview

Will Du

Oct. 29, 2015@Toronto



<http://www.it21learning.com>

## Outline



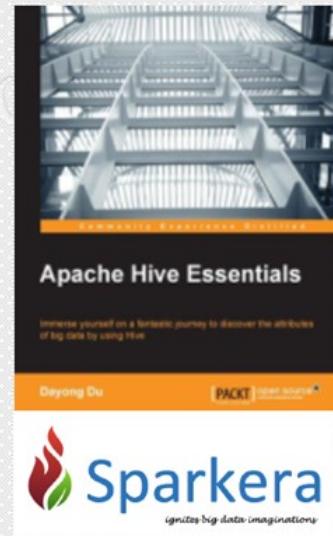
<http://www.it21learning.com>

# Outline

- Ecosystem – Technical Stack
- Ecosystem – Distribution, Vendors, Solutions

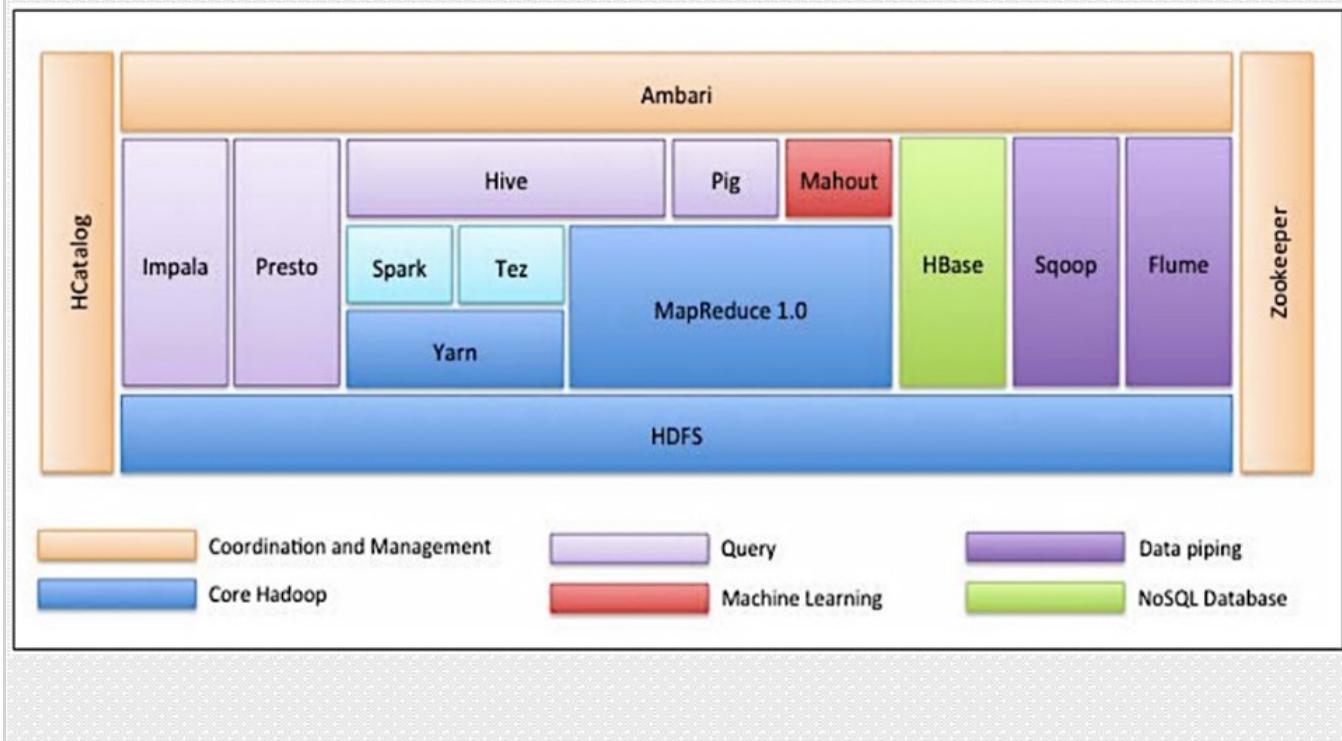
# Who Am I - Will Du

- A big data practitioner, leader, and developer
- 10+ years of experience in enterprise data warehouse and big data architecture
- The author of *Apache Hive Essentials*, the 1st book on Hive v1.0 in the world
- The first few certified Cloudera Hadoop Developer for CDH 4&5
- [www.sparkera.ca](http://www.sparkera.ca)

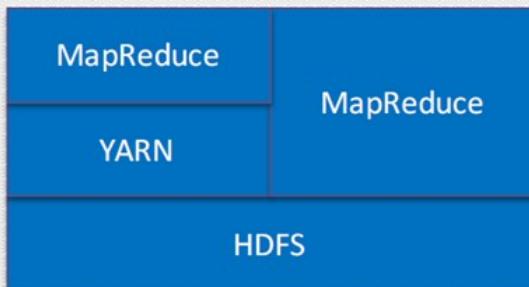


## Ecosystem - Technical Stack

# Ecosystem - Technical Stack



## Core Hadoop



# Core Hadoop

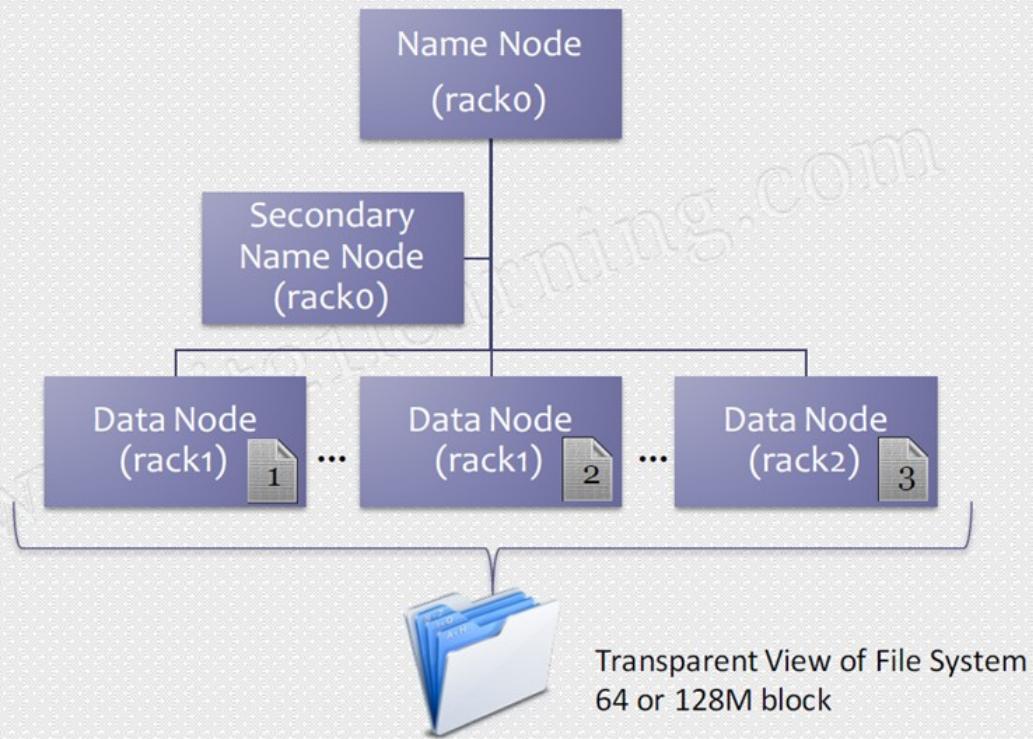
- HDFS – Hadoop Distributed File System
- MapReduce
- YARN – Yet Another Resource Negotiator



<http://www.it21learning.com>

## Hadoop Distributed File System

# Hadoop Distributed File System



# HDFS Cont.

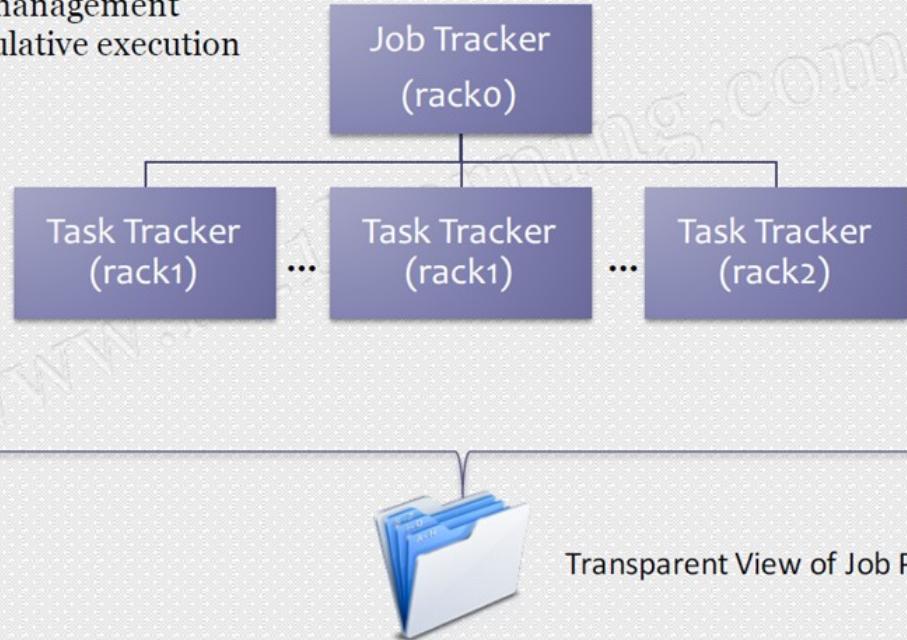
- Linux like command line
  - `hdfs dfs –help <command_name>`
  - `hdfs dfs –ls <URI>`
  - `hdfs dfs –cat <file_name>`
  - `hdfs dfs –rm [-r] <URI>`
  - `hdfs dfs –put <source_file> <URI>`
  - `hdfs dfs –get <source_uri> <local_destination>`
  - `hdfs dfs –getmerge <source_uri> <local_destination>`
- Dynamic and transparent load balance
- Dynamic and transparent data replication&recover
- Dynamic and transparent routing request
- Dynamic and transparent nodes maintenance

## MapReduce - Physical View

# MapReduce - Physical View

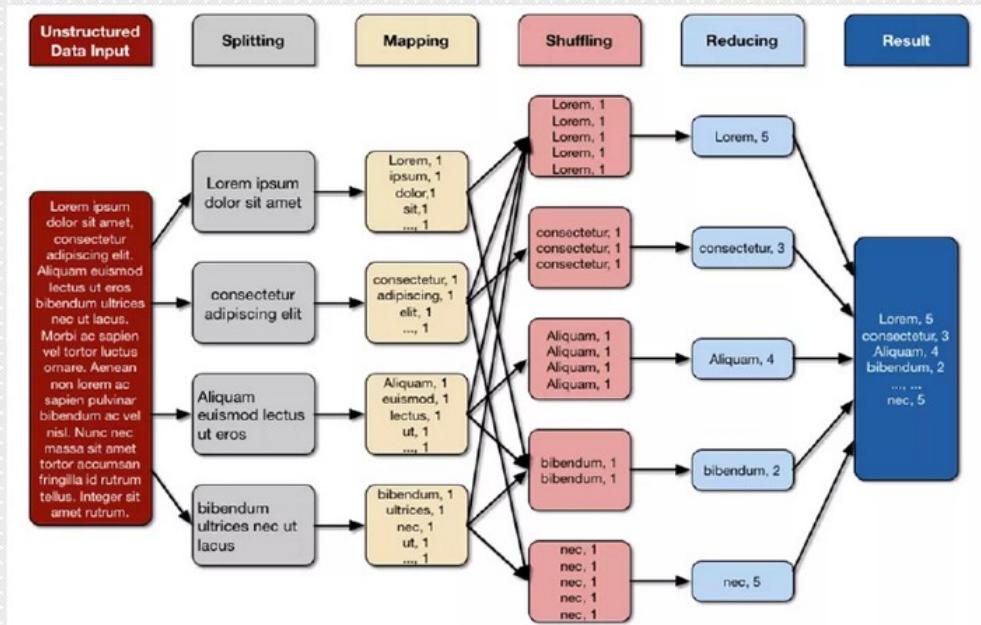
Dynamic and transparent

- Load balance
- Job management
- Speculative execution



## MapReduce - Logic View

# MapReduce - Logic View

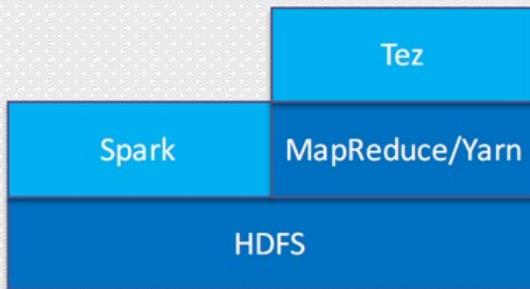


- Job is splitable
- Job is independent
- Move algorithm to data

# YARN - A Way Out for Hadoop

|                            |                                                                                                                                                                                                                                                                                                            |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Multi-tenancy</b>       | YARN allows multiple access engines (either open-source or proprietary) to use Hadoop as the common standard for batch, interactive and real-time engines that can simultaneously access the same data set.<br><br>Multi-tenant data processing improves an enterprise's return on its Hadoop investments. |
| <b>Cluster utilization</b> | YARN's dynamic allocation of cluster resources improves utilization over more static MapReduce rules used in early versions of Hadoop                                                                                                                                                                      |
| <b>Scalability</b>         | Data center processing power continues to rapidly expand. YARN's ResourceManager focuses exclusively on scheduling and keeps pace as clusters expand to thousands of nodes managing petabytes of data.                                                                                                     |
| <b>Compatibility</b>       | Existing MapReduce applications developed for Hadoop 1 can run YARN without any disruption to existing processes that already work                                                                                                                                                                         |

## Alternative Computing



# Alternative Computing

- Apache Tez
- Apache Spark



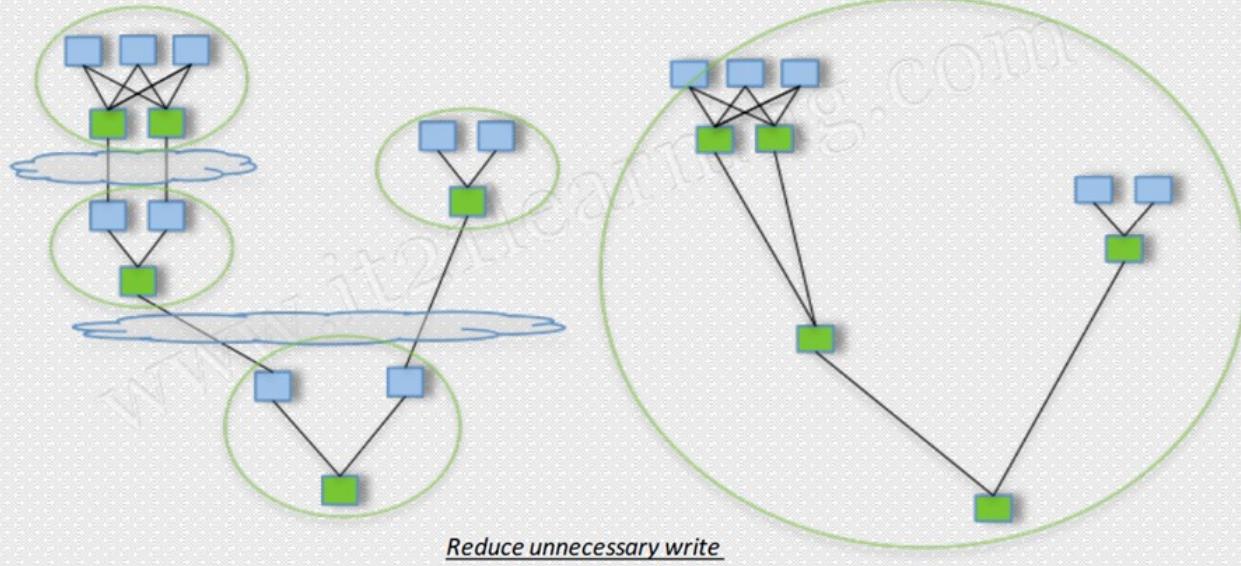
<http://www.it21learning.com>

## Apache Tez



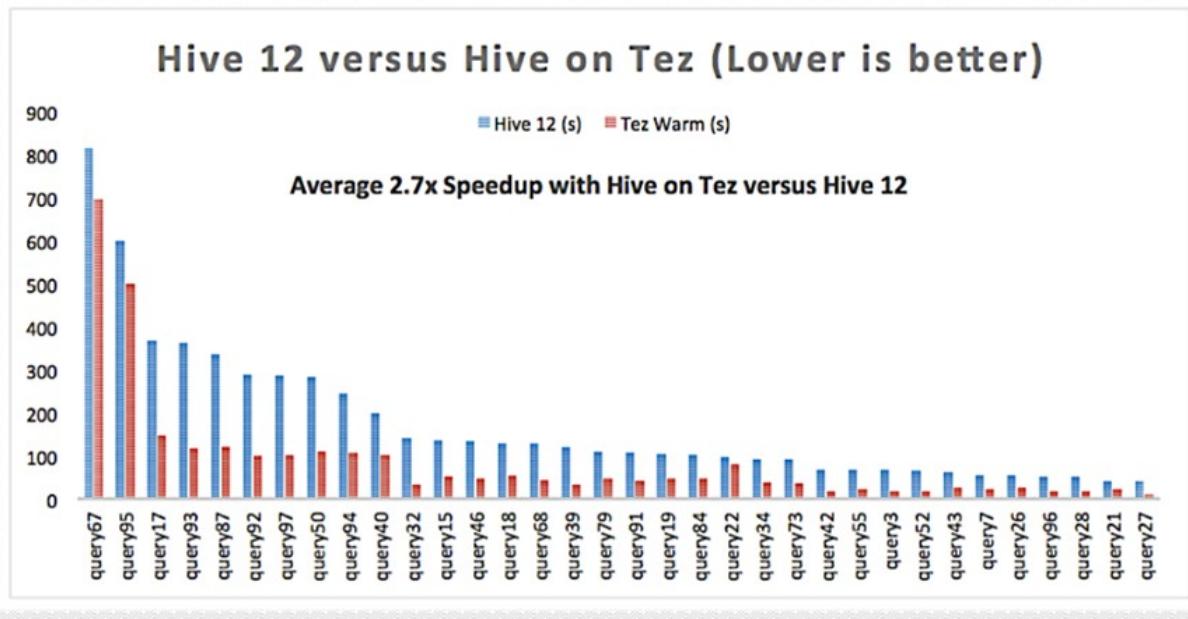
# Apache Tez

Building an application framework which allows for a complex directed-acyclic-graph of tasks for processing data. It is currently built on top of YARN. Tez is to further optimize MapReduce (fight against Impala).



# Tez Over MapReduce

As Stringer Initiative Phase I – *speed up Apache Hive for 100x*



# Apache Spark



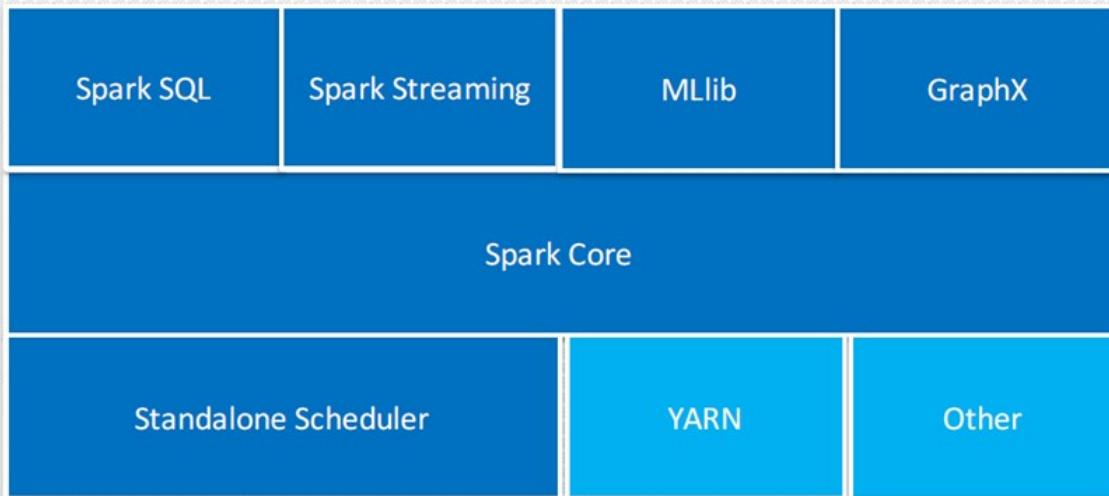
- Started in 2009 as a research project in UC Berkeley RAD lab which became AMP Lab.
- Spark researchers found that Hadoop MapReduce was inefficient for iterative and interactive computing.
- Spark was designed from the beginning to be fast for interactive, iterative with support for in-memory storage and fault-tolerance.
- Apart from UC Berkeley, Databricks, Yahoo! and Intel are major contributors.
- Spark was open sourced in March 2010 and transformed into Apache Foundation project in June 2013.
- Spark handles batch, interactive, and real-time within a single framework
- Native integration with Java, Python, Scala
- Programming at a higher level of abstraction
- Leverage Resilient Distributed Dataset—RDD to process data in memory
  - ❖ Resilient – Data can be recreated if it is lost in the memory
  - ❖ Distributed – Stored in the memory across cluster
  - ❖ Dataset – Can be created from file or programmatically

## Apache Spark Stack



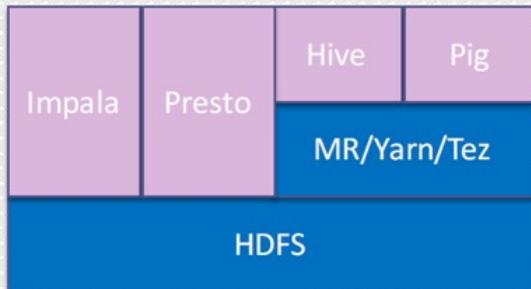
<http://www.it21learning.com>

# Apache Spark Stack



*Full stack and great ecosystem*

## Query Simplification



# Query Simplification

- Apache Hive
- Apache Pig
- Apache Impala
- Apache Presto



<http://www.it21learning.com>

## Apache Hive

# Apache Hive



- Data Warehousing Solution built on top of Hadoop
- Provides SQL-like query language named HiveQL - HQL, minimal learning curve
- Early Hive development work started at Facebook in 2007
- Today Hive is an top Apache project under Hadoop at [hive.apache.org](http://hive.apache.org)
- Hive provides a simpler query model with less coding than MapReduce
- Hive provides lots of functions that lead to easier analytics usage
- Hive supports running on different computing frameworks – MR, TEZ, Spark
- Hive supports ad hoc querying data on HDFS and HBase
- Hive supports user-defined functions, scripts, and customized I/O format to extend functionality
- Matured JDBC and ODBC drivers allow many applications to pull Hive data for seamless reporting
- Hive has a well-defined architecture for metadata management, authentication, optimizations
- There is a big community of practitioners and developers working on and using Hive

## Apache Hive vs. MapReduce

# Apache Hive vs. MapReduce

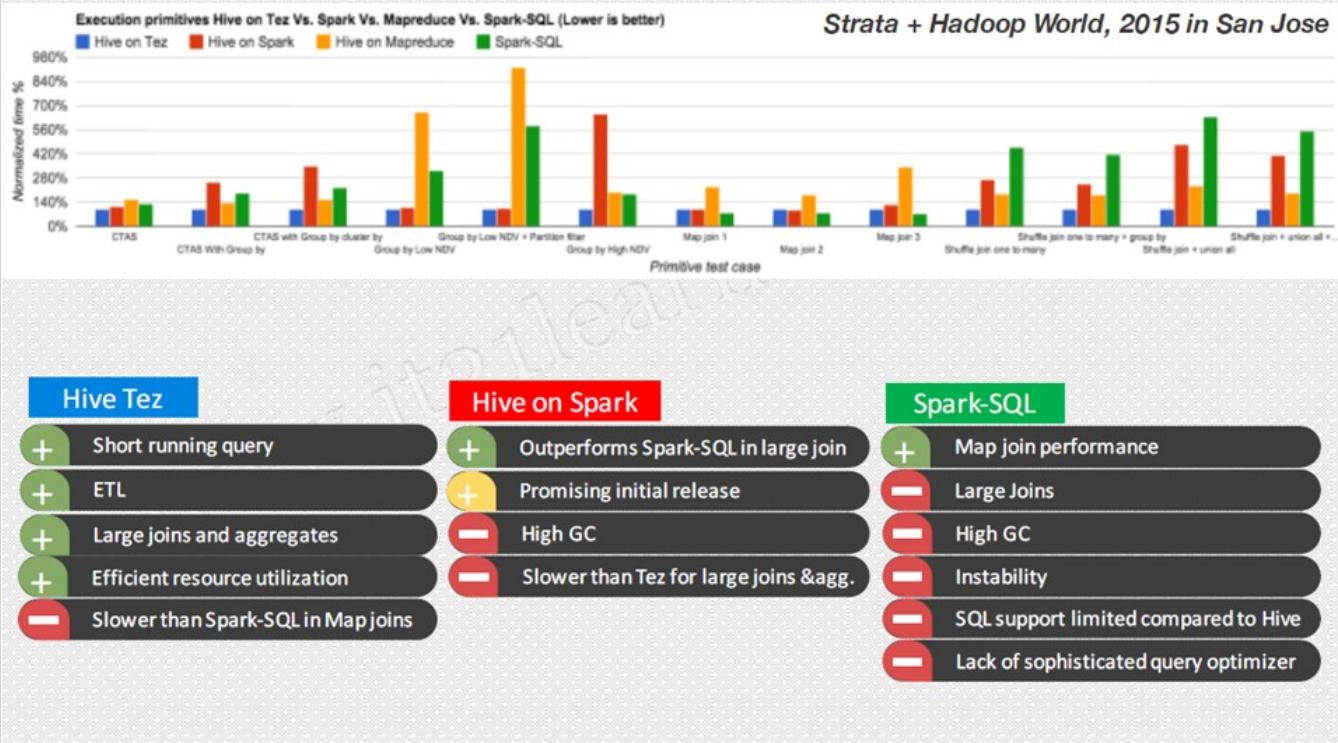
```
1 package org.apache.hadoop.examples;
2 import java.io.IOException;
3 import java.util.StringTokenizer;
4 import org.apache.hadoop.conf.Configuration;
5 import org.apache.hadoop.fs.Path;
6 import org.apache.hadoop.io.IntWritable;
7 import org.apache.hadoop.mapreduce.Job;
8 import org.apache.hadoop.mapreduce.Mapper;
9 import org.apache.hadoop.mapreduce.Reducer;
10 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12 import org.apache.hadoop.mapreduce.lib.options.GenericOptionsParser;
13
14 public class WordCount {
15
16     public static class TokenizerMapper
17         extends Mapper<Object, Text, IntWritable>{
18
19     private final static IntWritable one = new IntWritable(1);
20     private Text word = new Text();
21
22     public void map(Object key, Text value, Context context
23                     ) throws IOException, InterruptedException {
24         StringTokenizer itr = new StringTokenizer(value.toString());
25         while (itr.hasMoreTokens()) {
26             word.set(itr.nextToken());
27             context.write(word, one);
28         }
29     }
30 }
31
32     public static class IntSumReducer
33         extends Reducer<Text,IntWritable,Text,IntWritable> {
34     private IntWritable result = new IntWritable();
35
36     public void reduce(Text key, Iterable<IntWritable> values,
37                        Context context
38                        ) throws IOException, InterruptedException {
39         int sum = 0;
40         for (IntWritable val : values) {
41             sum += val.get();
42         }
43         result.set(sum);
44         context.write(key, result);
45     }
46 }
47
48     public static void main(String[] args) throws Exception {
49     Configuration conf = new Configuration();
50     String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
51     if (otherArgs.length != 2) {
52         System.err.println("Usage: wordcount <in> <out>");
53         System.exit(2);
54     }
55
56     Job job = new Job(conf, "word count");
57     job.setMapperClass(WordCount.class);
58     job.setMapperClass(TokenizerMapper.class);
59     job.setCombinerClass(IntSumReducer.class);
60     job.setReducerClass(IntSumReducer.class);
61     job.setOutputKeyClass(Text.class);
62     job.setOutputValueClass(IntWritable.class);
63     job.setFileInputFormat(FileInputFormat.class);
64     FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
65     System.exit(job.waitForCompletion(true));
66 }
67 }
```

## Simplified and Optimized

```
1 ---define metadata for the source
2 CREATE EXTERNAL TABLE lines(Line STRING);
3 LOAD DATA INPATH 'book' OVERWRITE INTO TABLE lines;
4
5 --- word count
6 SELECT word, count() as word_count
7 FROM lines
8 LATERAL VIEW explode(split(text, ' ')) t1 as word
9 GROUP BY word;
```

## Apache Hive vs. Spark

# Apache Hive vs. Spark



# Apache Pig



- It is an open source data flow language
- Pig Latin is used to express the queries and data manipulation operations in simple scripts
- Pig converts the scripts into a sequence of underlying Map Reduce jobs
- Pig is a **Data Flow** language, thus it is most suitable for:
  - ❖ Quickly changing data processing requirements
  - ❖ Processing data from multiple channels
  - ❖ Quick hypothesis testing
  - ❖ Time sensitive data refreshes
  - ❖ Data profiling using sampling
- Comparing to Hive, pig does not depend on meta data

# Apache Pig - Simple Example

Query : Get the list of web pages visited by users whose age is between 20 and 29 years.

```
$ pig
grunt> cat /training/playArea/pig/user.txt
willddy@hotmail.com 35
michael@gmail.com 28

grunt> cat /training/playArea/pig/pages.txt
http://www.google.ca willddy@hotmail.com 2015-10-01 10:34:45
http://www.yahoo.ca michael@gmail.com 2015-10-11 13:36:15

grunt> USERS = load 'users' as (uid, age);
grunt> USERS_20s = filter USERS by age >= 20 and age <= 29;
grunt> PVs = load 'pages' as (url, uid, timestamp);
grunt> PVs_u20s = join USERS_20s by uid, PVs by uid;
grunt> dump PVs_u20s;

...
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 50% complete
2015-10-14 17:36:22,040 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 100% complete
2015-10-14 17:36:22,040 [main] INFO
...
michael@gmail.com 28 michael@gmail.com 2015-10-11 13:36:15

grunt>
```

Also support STORE, FOREACH, DISTINCT, JOIN, LIMIT, GROUP etc.

# Apache Impala



- General-purpose SQL query engine from Cloudera since 2012
- Runs directly within Hadoop without MapReduce
- High performance in C++
- Runs as a distributed service in cluster: one Impala daemon on each node with data
- User submits query via ODBC/Beeswax Thrift API to any of the daemons
- Query is distributed to all nodes with relevant data
- If any node fails, the query fails
- Impala uses Hive's metadata interface, connects to Hive's metastore

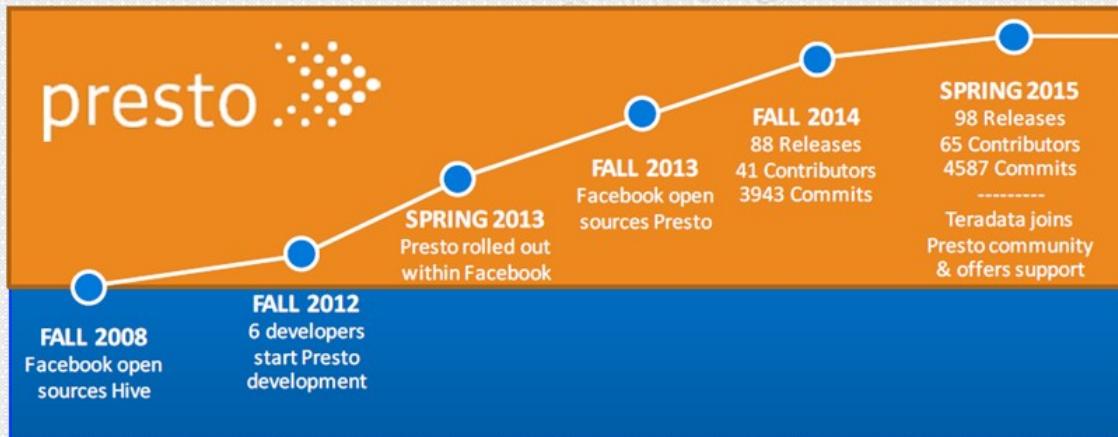
# Apache Impala SQL

- SQL support:
  - ❖ Patterned after Hive's version of SQL
  - ❖ Select, Project, Join, Union, Subqueries, Aggregation and Insert. etc.
  - ❖ Different type of JOIN support
  - ❖ DDL support
- Functional limitations:
  - ❖ Custom UDFs since v1.2
  - ❖ No beyond SQL (buckets, samples, transforms, arrays, structs, maps, xpath, json)
  - ❖ No Lateral views.

# Apache Presto



- Open source distributed ANSI SQL engine for Big Data
- Optimized for low latency, Interactive querying
- Cross platform query capability, MySQL, Hive, HDFS, etc
- Distributed under the Apache license, now supported by Teradata



# Apache Presto SQL

- Supported SQL features:
  - ❖ aggregation, scalar, most popular joins
  - ❖ Most data types
  - ❖ structural data types: row, map, array
- Not supported SQL features:
  - ❖ Scalar subqueries not supported: "where x = (select y from ...)"
  - ❖ Non-equal joins only supported for inner join: ("n\_name" < "p\_name")
  - ❖ EXISTS, EXCEPT, INTERSECT not supported
  - ❖ ROLLUP is not supported
  - ❖ LIMIT ALL and OFFSET clause not supported
- Other – lack of connector for BI/ETL tools

## Data Piping



# Data Piping

- Apache Flume
- Apache Sqoop



<http://www.it21learning.com>

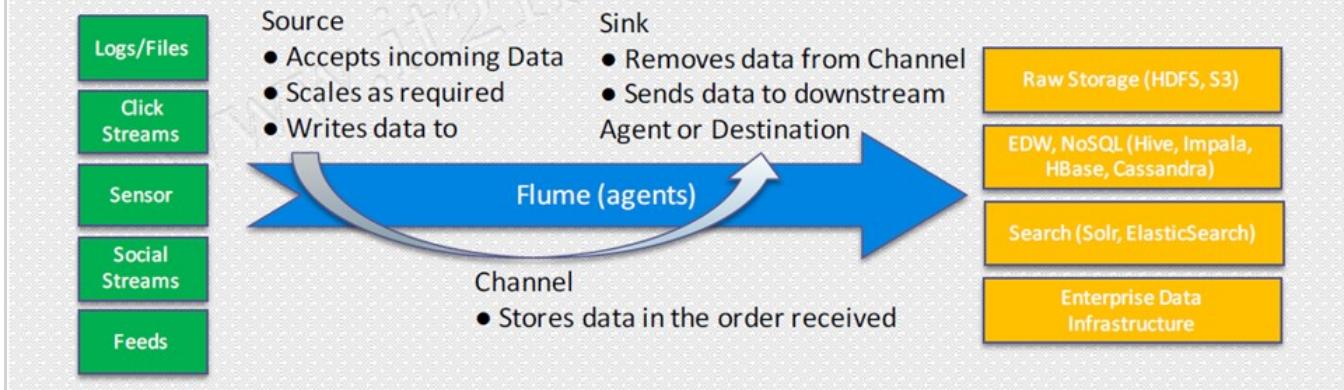
## Apache Flume



# Apache Flume

Apache Flume is a continuous data ingestion system that is...

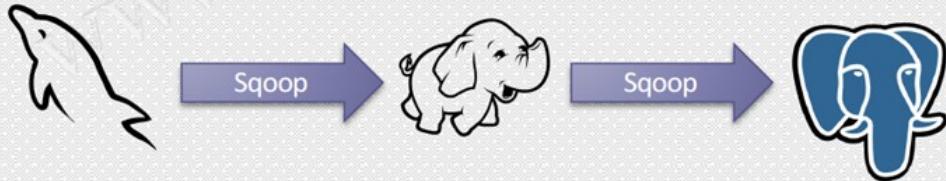
- Open-source
  - Reliable
  - Scalable
  - Manageable
  - Customizable
- ... and designed for Big Data ecosystem.



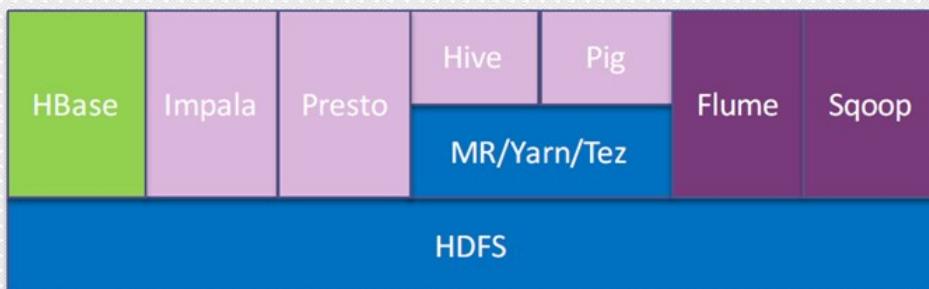
# Apache Sqoop



- Apache Top-Level Project
- Data bridge to Hadoop
- Tool to export/import data from relational databases to Hadoop
- Teradata, MySQL, PostgreSQL, Oracle, Netezza, DB2 are supported
- HDFS (text, sequence file), Hive, HBase, Avro are supported
- Sqoop convert job into mapreduce jobs
- Sqoop has good performance for transfer big volume data
- Not an ETL tool, but support simple transfer tracking support



## Database - Apache HBase



# Database

- Apache HBase

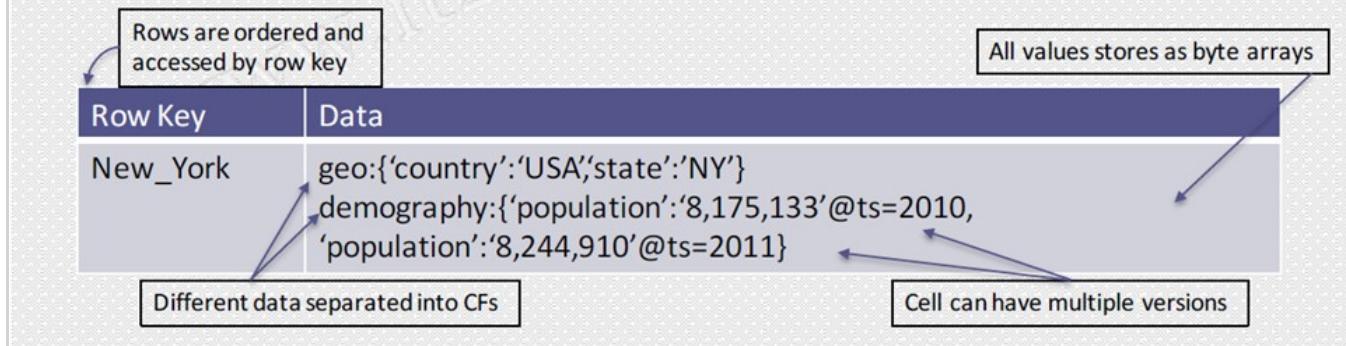


<http://www.it21learning.com>

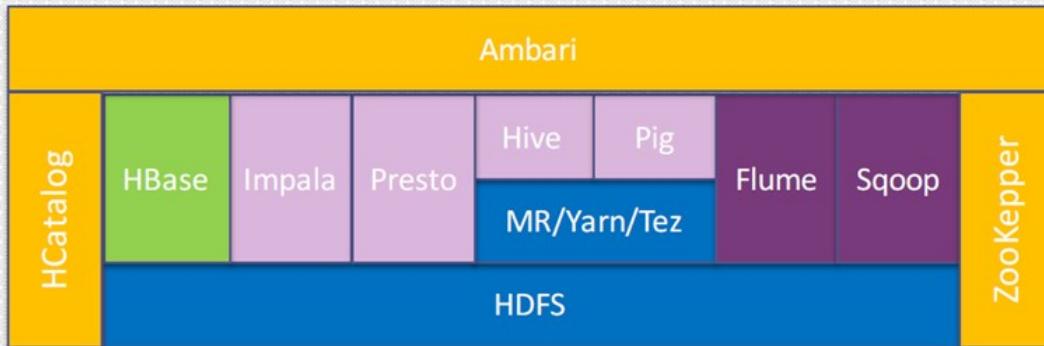


# Apache HBase

- An open source distributed NoSQL database on top of Hadoop
- Versioned and key-value, column store
- Real time **read** and write
- High scalable, fault tolerant, and HA – ZooKeeper
- Auto-sharding
- Strong consistent
- Out of box support for history date – designed row key & timestamp
- Rest and Java API support (SQL support from Phoenix)



## Management & Coordination



# Management & Coordination

- Apache HCatalog – expose Hive meta datastore
- Apache Zookeeper
- Apache Ambari



<http://www.it21learning.com>

# Apache ZooKeeper



- An open source, high-performance coordination service for distributed applications.
- Exposes common services in simple interface:
  - ❖ naming
  - ❖ configuration management
  - ❖ locks & synchronization
  - ❖ group services
- Build according to your requirement for specific needs.
- *It can be used in*
  - ❖ Locking - allow for serialized access to a shared resource
  - ❖ Synchronization – synchronizing shared resource
  - ❖ Configuration Management – centralize distributed system
  - ❖ Leader Election – dynamically leader selection

## Apache Ambari



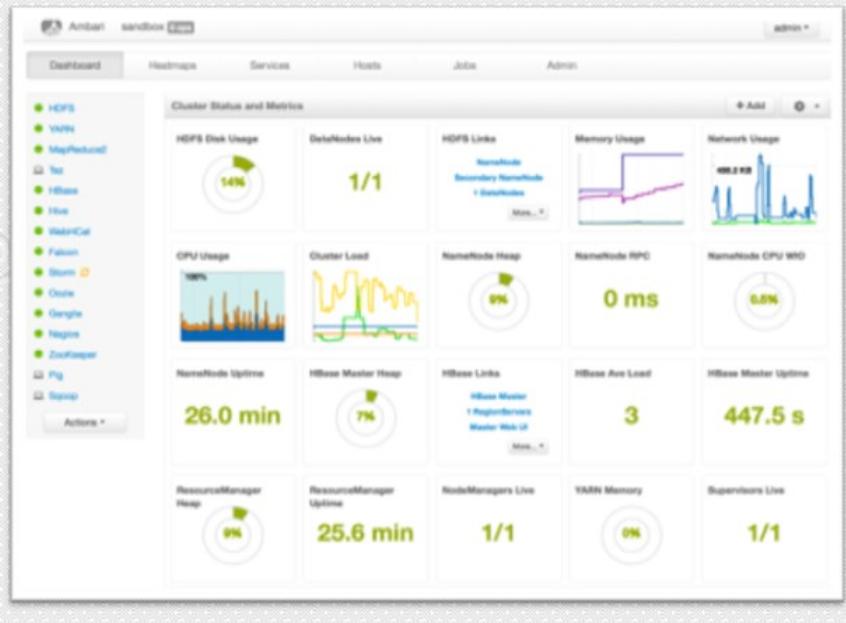
<http://www.it21learning.com>

# Apache Ambari



Apache Ambari is a platform to provision, manage and monitor Hadoop clusters – a free version of Cloudera Manager.

- Integrate with the Enterprise Robust API for integration with existing enterprise systems, such as Teradata Viewpoint and Microsoft Sys Center.
- Extend for the Ecosystem Provide extensible platform for Customers, Partners and the Community to, such as Stacks and Views.



# Ecosystem - Platform & Distribution



- As leader in the Hadoop solution provider
- CDH 5.\* as leading product
- Impala, Cloudera Search, Cloudera Manager
- Cloudera lab



- Fully open sourced Hadoop solution provider
- HDP 2.\* as leading product
- Tez, Data Flow, Ambari, ORC
- Bring Hadoop to windows

# Ecosystem - Platform & Distribution

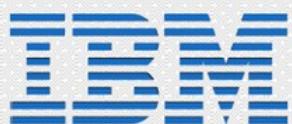


- Redesign Hadoop
- M3, M5, M7 as leading product
- Apache Drill, MapR DB, MapR FS
- Secure and High Performance Hadoop



- Biggest market share
- EMR.\* as leading product
- EC2, S3, Redshift, Lambda
- Bring Hadoop to Cloud

## Ecosystem - Platform & Distribution



- From competitor to player
- IBM BigInsights as leading product
- Big SQL, Big Sheet, Spark?
- Big Data University



- Leverage EMC and VMware
- Big Data Suit as leading product
- GemFire, Pivotal DB, HAWQ
- Leading SQL over Hadoop



- Hadoop from Windows Cloud
- HDInsight as leading product
- Use HDP version
- Close integration with MS product suite

## Question - Thank you

Thank You

Questions?



<http://www.it21learning.com>

## **Appendix B - Hadoop Data Tool - Apache Sqoop**

# Hadoop Data Tool - Sqoop

Wayne Shen



<http://www.it21learning.com>

# What is Sqoop?

- Sqoop is a tool designed to transfer data between Hadoop and relational databases or mainframes.
  - Import Data from RDBMS to HDFS
  - Export Data from HDFS to RDBMS
    - Sqoop uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.
  - Target Users
    - System and application programmers
    - System administrators
    - Database administrators
    - Data analysts
    - Data engineers

# Import RDB TABLE to HDFS

- **Import a table from a RDB**

- `sqoop import`  
`--connect jdbc:mysql://localhost/hr`  
`--table user`  
`--username root`  
`--password 12345`  
`--target-dir /data/user`  
`--m 3`
  - `--connect` → connection string
  - Database → hr
  - `--table` → user
  - `--target-dir` → HDFS target folder
  - `--m` → the number of Mappers

- **sqoop-import** is an alias of **sqoop import**

# Import by WHERE

- Import a table by filtering with WHERE

- sqoop import  
--connect jdbc:mysql://localhost/hr  
--table orders  
--where "order\_date > '2015-10-10'"  
--username root  
--password 12345  
--delete-target-dir  
--target-dir /data/orders  
--m 3
- WHERE for filtering records

## Import by COLUMNS



<http://www.it21learning.com>

# Import by COLUMNS

- Import a table from a RDB by COLUMNS

- sqoop import  
--connect jdbc:mysql://localhost/hr  
--table user  
--columns "host,name,age"  
--username root  
--password 12345  
--delete-target-dir  
--target-dir /data/user  
--m 3
  - --columns → the column list
  - --delete-target-dir → delete the target directory if it exists

## Import by Query



<http://www.it21learning.com>

# Import by Query

- **Import a table from a RDB by QUERY**

- ```
sqoop import
--connect jdbc:mysql://localhost/hr
--query "select * from user where host != '127.0.0.1' and \$CONDITIONS"
--username root
--password 12345
--split-by host
--delete-target-dir
--target-dir /data/user
--m 3"
```
- --query → all the queries should end up with \\$CONDITIONS, which is used by sqoop internally to distribute record ranges to all mappers.
- --split-by host → the host column is used to split work units.

# Incremental Import with Sqoop

- **Incremental Mode:**

- **append** – append all matched records (may create duplicates in target)
- **last-modified** – append new records and update modified records from all matched records.
- sqoop import  
--connect jdbc:mysql://localhost/hr  
--table orders  
--username root  
--password 12345  
--incremental append  
--check-column order\_date  
--last-value '2015-10-10'  
--target-dir /data/orders  
--m 3

# File Format

- **Target File Format:**

- **--as-avrodatafile** → Imports data to Avro Data Files
- **--as-sequencefile** → Imports data to SequenceFiles
- **--as-textfile** → Imports data as plain text (default)
- **--as-parquetfile** → Imports data to Parquet Files
- sqoop import  
--connect jdbc:mysql://localhost/hr  
--table orders  
--username root  
--password 12345  
--delete-target-dir  
--target-dir /data/orders  
--m 3  
--as-sequencefile

## Import into Hive



<http://www.it21learning.com>

# Import into Hive

- ```
sqoop import
--connect jdbc:mysql://localhost/hr
--table orders
--username root
--password 12345
--hive-import
--create-hive-table
--hive-table eShop.orders
--hive-overwrite
--m 3
--as-parquetfile
```

  - --hive-import → import into Hive
  - --create-hive-table → create a new hive table. If target table exists, error out;
  - --hive-overwrite → overwrite existing target data

# Partition Data

- **Partition Data when importing into Hive**

```
▫ sqoop import  
  --connect jdbc:mysql://localhost/hr  
  --query "select order_id, order_status from orders  
    where order_date >= '2014-07-24' and order_date < '2014-07-25'  
      and \$CONDITIONS"  
  --username root  
  --password 12345  
  --target-dir /user/data/orders  
  --split-by order_status  
  --hive-import  
  --hive-table eShop.orders  
  --hive-partition-key "order_date"  
  --hive-partition-value "20140724"  
  --m 3
```

## Any Question?

# Any Question?

