National Workshop on Recent Trends in Big Data Analytics
February 27, 2016

Hadoop EcoSystem and Big Data

By M. Saravanan

Hadoop and Its Ecosystem

- Hadoop is an open source framework for processing large amount of data in batches
- Hadoop is created for pipelining massive amount of data for data processing to achieve excellent end result
- The idea of Hadoop is to provide a cost-efficient High Performance Computing using the cloud infrastructure
- Hadoop is an Apache top level project, and licensed under Apache 2.0

Gartner mentioned "Big Data required new forms of processing to enable enhanced decision making, insight discovery and process optimization"

Core Component of Hadoop

Built in a modular approach, core component of Hadoop consists of:

- **Hadoop Common** – libraries and utilities that provides common functionality of Hadoop
- **Hadoop Distributed File System (HDFS)** – A distributed file system that provides high-throughput access to application data. A distributed file-system that stores data on multiple machines in the cluster. HDFS is made to be fault tolerant and capable of running on commodity hardware
- Hadoop **YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce** – a component model for large scale data processing in a parallel manner.

Data types

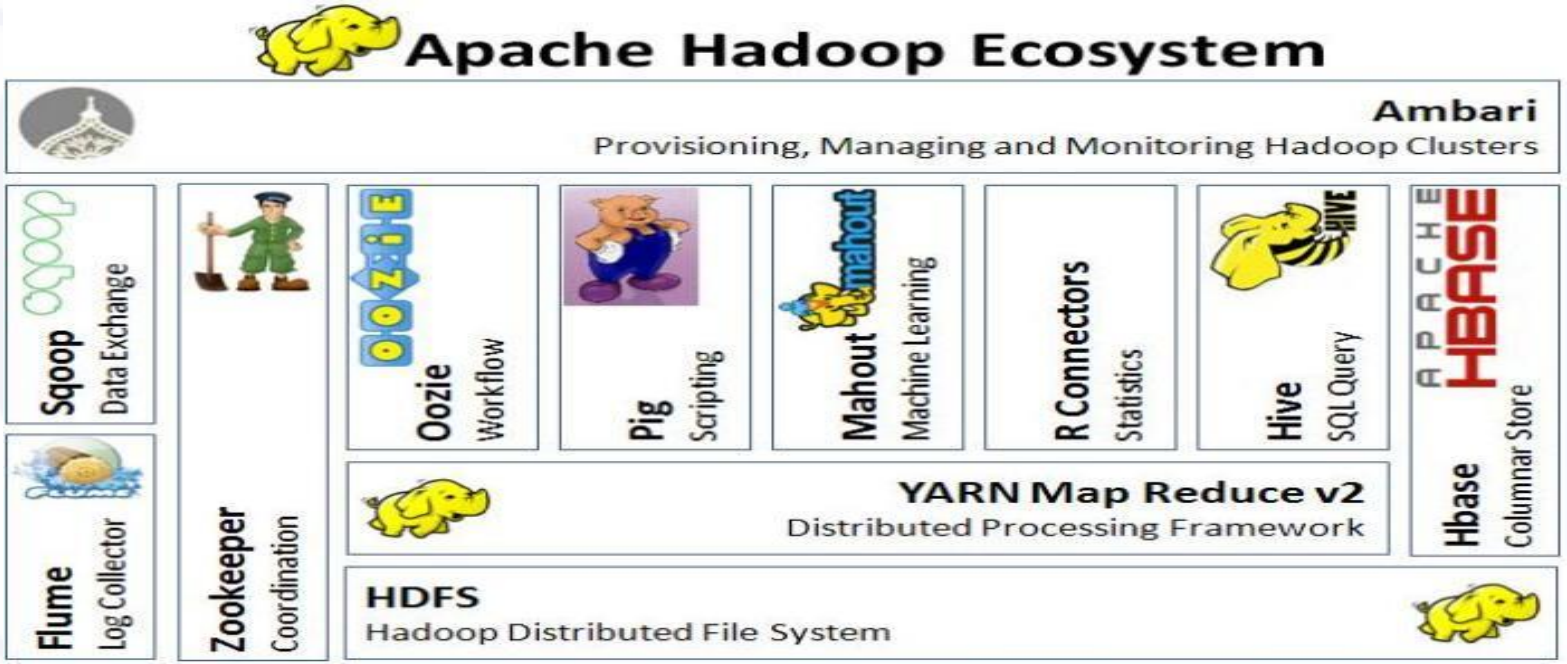
- **Structured**: Structured data has strong schema and schema will be checked during write & read operation. e.g. Data in RDBMS systems like Oracle, MySQL Server etc.
- **Unstructured**: Data does not have any structure and it can be any form - Web server logs, E-Mail, Images etc.
- **Semi-structured**: Data is not strictly structured but have some structure. e.g. XML files.

Depending on type of data to be processed, we have to choose right technology.

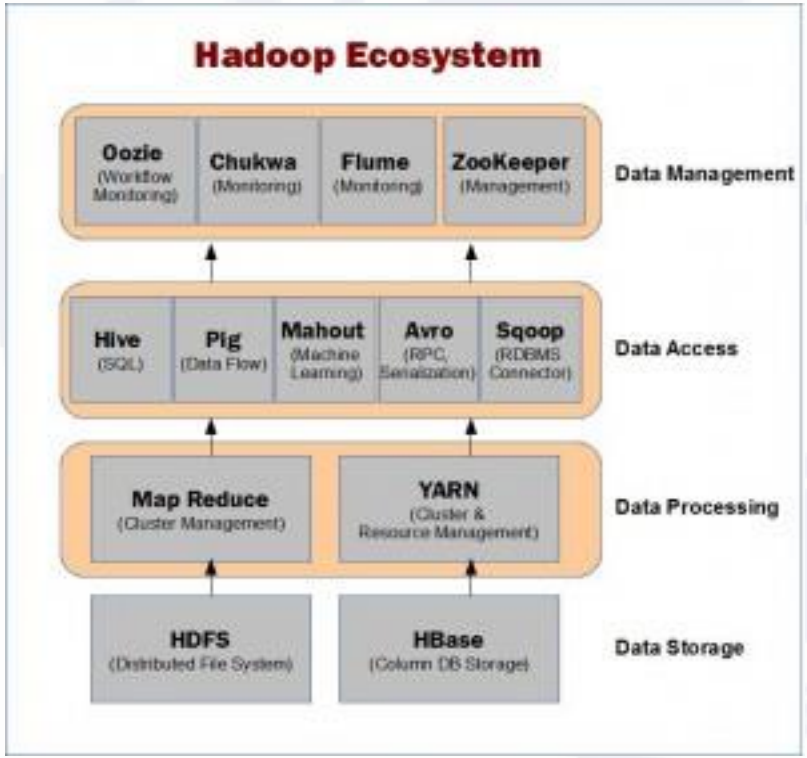
Parts of Hadoop

- **HBase:** A scalable, distributed database that supports structured data storage for large tables.
- **Hive:** A data warehouse infrastructure that provides data summarization and ad-hoc querying.
- **Pig:** A high-level data-flow language and execution framework for parallel computation.

Hadoop EcoSystem



4 Layers of Hadoop EcoSystem



- **Data Storage Layer**
  This is where the data is stored in a distributed file system, consist of HDFS and HBase ColumnDB Storage. HBase is scalable, distributed database that supports structured data storage for large tables.
- **Data Processing Layer**
  Is where the scheduling, resource management and cluster management to be calculated here. YARN job scheduling and cluster resource management with Map Reduce are located in this layer.
- **Data Access Layer**
  This is the layer where the request from Management layer was sent to Data Processing Layer. Some projects have been setup for this layer, Some of them are: Hive, A data warehouse infrastructure that provides data summarization and ad hoc querying; Pig, A high-level data-flow language and execution framework for parallel computation; Mahout, A Scalable machine learning and data mining library; Avro, data serialization system.
- **Management Layer**
  This is the layer that meets the user. User access the system through this layer which has the components like: Chukwa, A data collection system for managing large distributed system and ZooKeeper, high-performance coordination service for distributed applications.

About Hadoop EcoSystem

- About large data processing
- Focus on problem solving
- Map Reduce and Hadoop
- HBase
- Hive
- Less on Administration
- More on Programming

In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We shouldn't be trying for bigger and bigger computer, but for more systems of computers.

- Grace Hopper

Session 1 - Introduction

- Introduction
- Distributed computing
- Parallel computing
- Concurrency
- Cloud Computing
- Data Past, Present and Future
- Computing Past, Present and Future

What is Hadoop

- Framework written in Java
  - Designed to solve problem that involve analyzing large data (Petabyte)
  - Programming model based on Google's Map Reduce
  - Infrastructure based on Google's Big Data and Distributed File System
  - Has functional programming roots (Haskell, Erlang, etc.,)
  - Remember how merge sort work and try to apply the logic to large data problems

What is Hadoop

- Hadoop is an open source framework for writing and running distributed applications that process large amounts of data.
- Hadoop Includes
  - Hadoop Common utilities.
  - Avro: A data serialization system with scripting languages.
  - Chukwa: managing large distributed systems.
  - HBase: A scalable, distributed database for large tables.
  - HDFS: A distributed file system.
  - Hive: data summarization and ad hoc querying.
  - Map Reduce: distributed processing on compute clusters.
  - Pig: A high-level data-flow language for parallel computation.
  - Zookeeper: coordination service for distributed applications.

What is Hadoop

- Hadoop key differentiating factors
  - Accessible: Hadoop runs on large clusters of commodity machines or cloud computing services such as Amazon EC2
  - Robust: Since Hadoop can run on commodity cluster, its designed with the assumption of frequent hardware failure, it can gracefully handle such failure and computation don't stop because of few failed devices / systems
  - Scalable: Hadoop scales linearly to handle large data by adding more slave nodes to the cluster
  - Simple : Its easy to write efficient parallel programming with Hadoop

What is Hadoop

- What it is not designed for
  - Hadoop is not designed for random reading and writing of few records, which is type of load for online transaction processing.
- What is it designed for
  - Hadoop is designed for offline processing of large-scale data.
  - Hadoop is best used as write once and read many type of data store, similar to data warehousing.

Parallel Programming

- The Challenge so far
  - No framework represent the right level abstraction that represent parallelism natively so as to implement algorithm needed to effectively design, implement, and maintain applications that exploit parallelism.
  - Identifying work that can be done concurrently
  - Mapping work to processing units
  - Distributing the work
  - Managing access to shared data
  - Synchronizing various stages of activity
- Solutions prior to Hadoop
  - MPI
  - PVM
  - Etc
- As architecture
  - Single Instruction Multiple Data
  - Multiple Instruction Multiple Data

Parallel Programming Model

A way to structure parallel algorithm by selecting decompositions and mapping techniques in a manner to minimize interactions

- Different Models to Parallel Algorithms
  - Data Parallel : Independent process assigned data to work with.
  - Task graph : Task are mapped to nodes in data dependency graph, data moves from source to sink.
  - Work Pool : Pool of workers take data as an when available.
  - Master Slave : A master process assign work and sent data to set of other process.
  - Pipeline
  - Hybrid

Parallel Programming Model

- Different Models to Parallel Programming
  - Message passing
    - Independent tasks encapsulating local data
    - Tasks interact by exchanging messages
    - Most widely used, MPI, PVM, and considered as the most efficient form of parallel programming and has been effective in solving lot of SIMD and MIMD kind of problems both in data intensive and compute intensive problems.
  - Shared memory
    - Tasks share a common address space
    - Tasks interact by reading and writing this space asynchronously.
    - Mostly used on Symmetric Multiprocessing machine (multicore chips), they use shared address space and belong to single process with more threads.
  - Data parallelization
    - Tasks execute a sequence of independent operations
    - Data usually evenly partitioned across tasks
    - Also referred to as “Embarrassingly parallel”

Problems with Parallel Computing

- Synchronization
- Efficiency
- Reliability

Concurrency

- Problem: Run multiple applications in such a way that they are protected from one another
- Goal:
  - Keep User Programs from Crashing OS
  - Keep User Programs from Crashing each other

Concurrency

Application

Virtual Machine Interface

Operating System

Physical Machine Interface

Hardware

Concurrency

- Threads
  - Concurrency within a process
- Multiple Process
  - Multiple copy of same program

Concurrency - Communication

- Shared-Memory Mapping
  - Accomplished by mapping addresses to common DRAM
  - Read and Write through memory
  - Communication occurs by “simply” reading/writing to shared address page

Concurrency Communication

- Message Passing
  - Mechanism for processes to communicate and to synchronize their actions
  - processes communicate with each other without resorting to shared variables
- IPC facility provides two operations
  - send(message) – message size fixed or variable
  - receive(message)
- If P and Q wish to communicate, they need to
  - establish a communication link between them
  - exchange messages via send/receive
- Implementation of communication link

Cloud Computing

- Delivery of computing as service rather than a product, whereby shared resources, software, and other information are provided as service over a network.
- The consumer doesn't know the source of the infrastructure that provides him/her with computation, software, data access, storage, etc., this enables the consumer to consume a resource in a pay as you use model.

Cloud Computing

- Different forms of Cloud Computing
  - Software as a Service
    - Delivers software as a service without on internet, eliminating the need to install and run the application, example google apps like email for corporate, google docs.
  - Platform as a Service
    - Delivers a computing platform as a service, example would be http://www.heroku.com/ which provides a ruby stack as a service, we can build application on ruby stack and host them in heroku. Google Apps is another example.
  - Infrastructure as a service
    - Delivers computer infrastructure – a virtual machine or bunch of virtual machine, this is the core of the cloud computing, and other services are build on top of this. Example of this is amazon.com, linode, rackspace, etc.

Data Past Present Future

- Data Mining, or we can working on data has received much interest in recent years because of the new found processing power (Hadoop) to crunch huge volumes of data.
- Be it past, present or future, people worked on data to extract hidden information.
- People employed a set of algorithms to extract this hidden information.
- Hidden Information != Complex SQL Query
- Past it was tabular, today its not only tabular but an ever growing set of non standard or un-structured data and in the future its going to be continuous stream of related and data.

Data : Source, Apps and Algorithm

Data Source / Type	Applications	Format of Data	Algorithms
Hypermedia	Internet and Intranet	Hyper Text Data	Classification and Clustering
Ubiquitous	Mobile Apps, PDA, Camera, etc.,	Ubiquitous Data	Statistical Machine Learning
Multimedia Data	Audio Video Apps	Multimedia Data	Rule based classification
Spatial Data	Networking, Remote Sensing, GIS apps	Spatial Data	OLAP, Spatial Clustering
Time Series	Business and Financial apps	Time Series Data	Rule Induction

Session 2 - Understanding Hadoop Stack

- Understanding HDFS
- ○ MapReduce
- ○ Databases: Key Value, Document, Graph
- ○ HBase
- ○ Hive and Pig
- ○ HDFS

Goals of HDFS

- Very Large Distributed File System
  - 10K nodes, 100 million files, 10PB
- Assumes Commodity Hardware
  - Files are replicated to handle hardware failure
  - Detect failures and recover from them
- Optimized for Batch Processing
  - Data locations exposed so that computations can move to where data resides
  - Provides very high aggregate bandwidth

Distributed File System

- Single Namespace for entire cluster
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Files are broken up into blocks
  - Typically 64MB block size
  - Each block replicated on multiple DataNodes
- Intelligent Client
  - Client can find location of blocks
  - Client accesses data directly from DataNode

HDFS Basic Features

- The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware
- Highly fault-tolerant and is designed to be deployed on low-cost hardware
- Provides high throughput access to application data and is suitable for applications that have large data sets
- Can be built out of commodity hardware

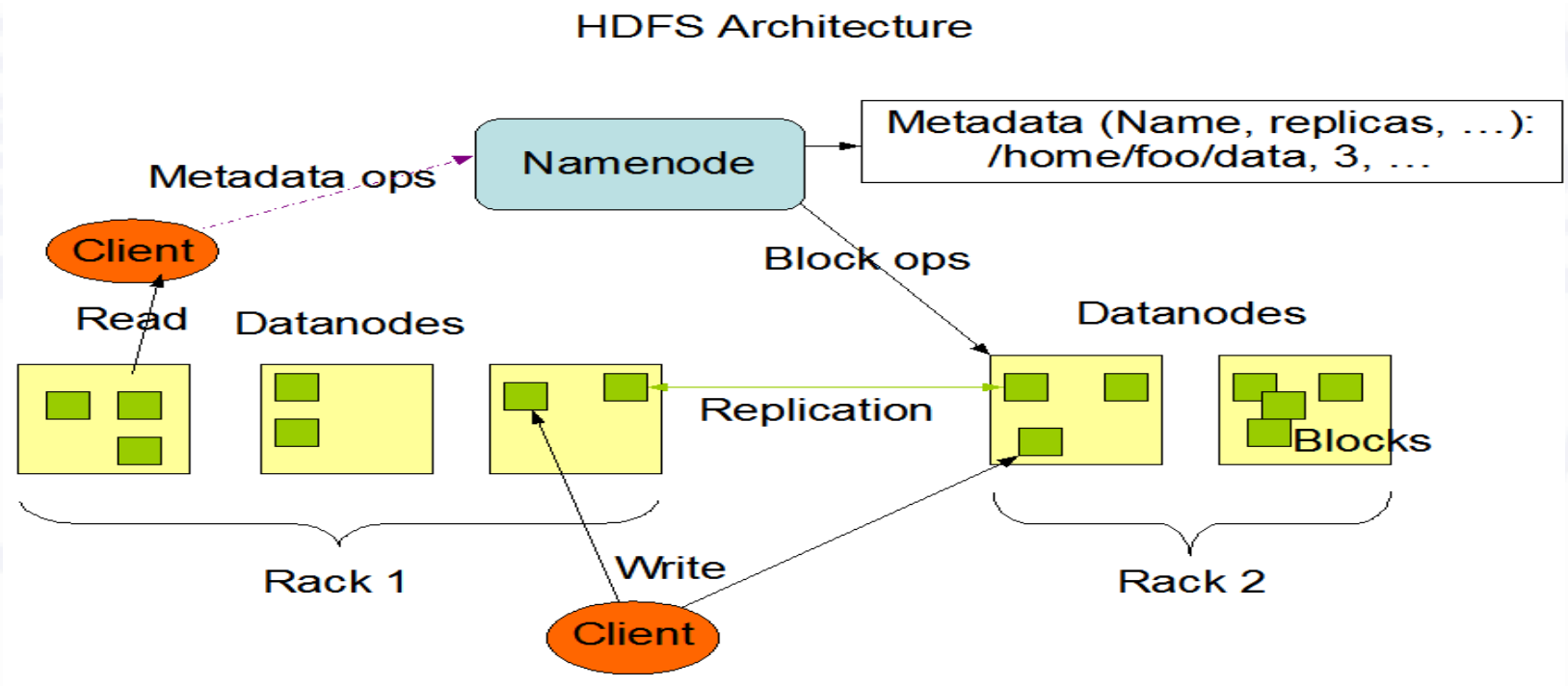
HDFS Design Principle - Failure

- Failure is the norm rather than exception
- A HDFS instance may consist of thousands of server machines, each storing part of the file system's data.
- At any point in time there is a high probability that few components are non-functional (due to the high probability of system failure).
- Detection of faults and quickly recovering from them in an automated manner is the core architectural goal of HDFS.

HDFS Architecture

- Master / Slave Architecture
- HDFS cluster consist of a single NameNode, this manages the file system namespace and regulates access to files by clients
- Many Data Nodes, typically one DataNode for a physical node
- DataNode manages storage attached to the node in which they run.
- HDFS exposes a file system namespace and allows user data to be stored in files.
- File is split into one or more blocks and set of such blocks are stored in each DataNode
- The NameNode executes file system namespace operations like opening, closing, and renaming files and directories
- NameNode determines the mapping of blocks to DataNodes
- DataNode serves read, write, perform block operation, delete, and replication upon request from NameNode
- The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

HDFS Architecture



NameNode Metadata

- Metadata in Memory
  - The entire metadata is in main memory
  - No demand paging of metadata
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

DataNode

- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores metadata of a block (e.g. CRC)
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data
  - Forwards data to other specified DataNodes

Block Placement

- Current Strategy
  - One replica on local node
  - Second replica on a remote rack
  - Third replica on same remote rack
  - Additional replicas are randomly placed
- Clients read from nearest replicas
- Would like to make this policy pluggable

Heartbeats

- DataNodes send hearbeat to the NameNode
  - Once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure

Replication Engine

- NameNode detects DataNode failures
  - Chooses new DataNodes for new replicas
  - Balances disk usage
  - Balances communication traffic to DataNodes

Data Pieplining

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next node in the Pipeline
- When all replicas are written, the Client moves on to write the next block in file

Rebalancer

- Goal: % disk full on DataNodes should be similar
  - Usually run when new DataNodes are added
  - Cluster is online when Rebalancer is active
  - Rebalancer is throttled to avoid network congestion
  - Command line tool

What is Map Reduce

- A programming model designed by Google, using which a sub set of distributed computing problems can be solved by writing simple programs.
- It provides automatic data distribution and aggregation.
- Removes problems with parallel programming, it believes in shared nothing architecture.
- To Scale, systems should be stateless.

Map Reduce

- A simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs
- Partitions input data
- Schedules execution across a set of machines
- Handles machine failure
- Manages inter-process communication

Map Reduce – Introduction

- Divide and conquer is the only feasible way to solve large scale data problems.
- Basic idea is to partition a large problem into smaller problems, to the extend that the sub problems are independent, they can be tackled in parallel by different worker. Intermediate results from workers are combined to produce the final result.
- This general principle behind divide and conquer is broadly applicable to a wide range of problems, **the impeding factor is in the details of implementation.**

Map Reduce – Introduction

- Issues in Divide & Conquer
  - How do we break a large problem into smaller tasks? How do we decompose the problem so that smaller task can be executed in parallel?
  - How do we assign task to workers distributed across a potentially a large number of machines?
  - How do we ensure workers get the data they need?
  - How do we coordinate synchronization among different workers?
  - How do we share partial result from one worker that is needed another?
  - How do we accomplish all of the above in the face of software error and hardware failure?

In traditional parallel programming, the developer needs to explicitly address many of the issues.

Map Reduce – Introduction

- Map Reduce are executed in two main phases, called mapping and reducing.
- Each phase is defined by a data processing function and these functions are called mapper and reducer.
- In map phase , MR takes the input data and feeds each data element into mapper.
- The Reducer process all output from mapper and arrives at final output.

Map Reduce – Introduction

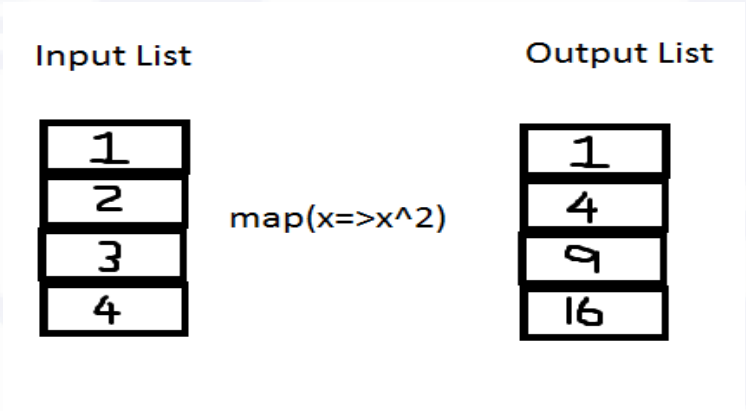
- In order for mapping, reducing, partitioning, and shuffling to seamlessly work together, we need to agree on a common structure for data being processed.
- It should be flexible and powerful enough to handle most of the target data processing application.
- Map Reduce use list and pair<key,value> as its fundamental primitive data structure.

Mapping

- This is probably the first step that Map Reduce starts with (apart from reading input files, configuration, etc)
- The map is provided with a list as input, the FileInputFormater takes care of providing this list.
- Inside the map, each element in the list is transformed and made available for next level.

Mapping

- Square is function applied to every element in the input list. The map function applies this and as a result we get a new list that has square of the input list.
- We did not modify the original list, we created a new list.

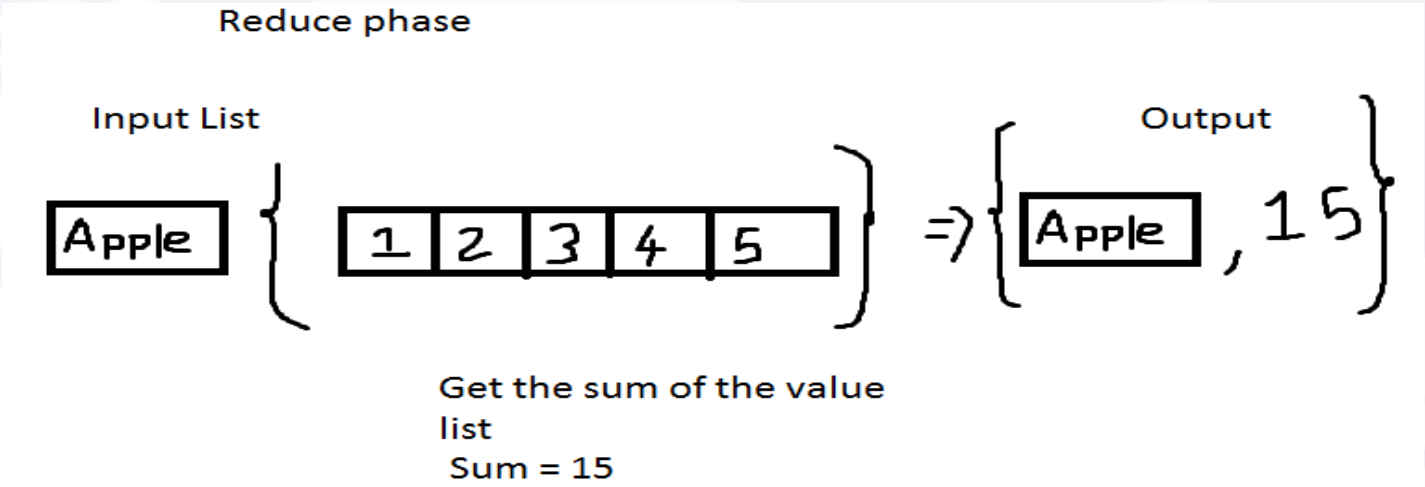


Reducer

- The next task done by map reduce is reducer (for now forget the intermediate shuffle and sort)
- Each reducer gets a key and a list of values for the key.
- The reducer job is transform / reduce those list of values into a single value.

Reducer

- Reduce aggregates the values in the list.
- Reduce receives Iterator as value, it combines them and emits a single value.
- The single value could be the biggest of all, smallest of all, sum of it, average of the list, etc.



Map and Reduce Phase

- Map is Parallel in nature, each element in the list is processed independently.
  - That is, key and values are scattered in the document space, each map job might look at some portion of data and find the key and its values, like this many map works in parallel.
- Reduce is sequential in nature, that is all values of the key are processed one after another.

Map Reduce – Introduction

	Input	Output
Map	<K1,V1>	List(<K2,V2>)
Reduce	<K2,List(V2)>	List(<K3,V3>)

Map Reduce - Introduction

- Map
  - Records from the data source (line output of files, rows of database, etc) are fed into the map function as <key, value> pairs, e.g (filename, line).
  - Map() function produces one or more intermediate values along with an output keys from the input.

Map Reduce – Introduction

```
Text word = new Text();
Public void map(LongWritable key, Text value, Context context) {

String line = value.toString();
StringTokenizer st = new StringTokenizer(line);

while(st.hasMoreElement())  {
    word.set(st.nextToken());
    context.write(word, new  IntWritable(1));
}
}
```

We will get to the details of this code shortly, as of now, the most important things to know is that we are creating intermediate <Key, Value> Pair.

Map Reduce - Introduction

- Reduce
  - After map phase is over, all intermediate values of a given output key are combined together into a list.
  - Reduce combine these intermediate values into one or more final values for the same output key.

Map Reduce - Introduction

```
Public void reduce(Text key, Iterator<IntWritable> values, Context context) {
Int sum = 0;
while( values.hasNext() )
  sum += values.next().get(); // ! = count , it's the value in the list
context.write(key, new IntWritable(sum));
}
```

Map Reduce - Introduction

- Parallelism – Joining the dots
  - Map function runs in parallel, creating different intermediate values from different input data sets.
  - Reduce function also runs in parallel, each working on a different output key.
  - All values are processed independently.

Map Reduce - Introduction

- We don't have to bother about sorting.
- We don't have to bother about shuffling.
- We don't have to bother about communication and scheduling.
- All we have to bother and care is about data and task.

Map Reduce - Introduction

- Before we start with map reduce, lets understand what it means by map and what it means by reduce in traditional sense.
  - Map is a function that transforms items in some kind of list to another kind of item and put them back in the same kind of list.
  - Example, if I have a list [1,2,3] and want to square each of them. The function is "Square" that is x = x ^ 2;
  - Traditionally we are used to think in terms of iteration using loops, which means we write the following code

```
int[] x = {1,2,3};
for(int i = 0; i<3; i++)
    x[i] = square(x[i]);
```

Map Reduce - Introduction

- Imagine if we can write like this

```
int[] x = {1,2,3];
x.map( x => x ^ 2);
```

- x=>x^2; is the function executed against each element in the array x, the program executes the function on each element in the list.

```
int[] x = {1,2,3];
x.map( x => x ^ 2);
```

- You might ask what difference it makes if we loop or have a map function?.
- We did not write a function called square, we called a map function and passed the logic (to square) into the function.
- Understanding the difference is important before we proceed further

Map Reduce - Introduction

- Reduce is a function which collects items in list and perform computation on all of them, thus reducing them to single value (most of the time, but we can use reduce phase to make transformation also).
- Example: Reduce, if I have a list [1,2,3] and want to find the sum of all the numbers in the list. In traditional manner we would write a code like this.

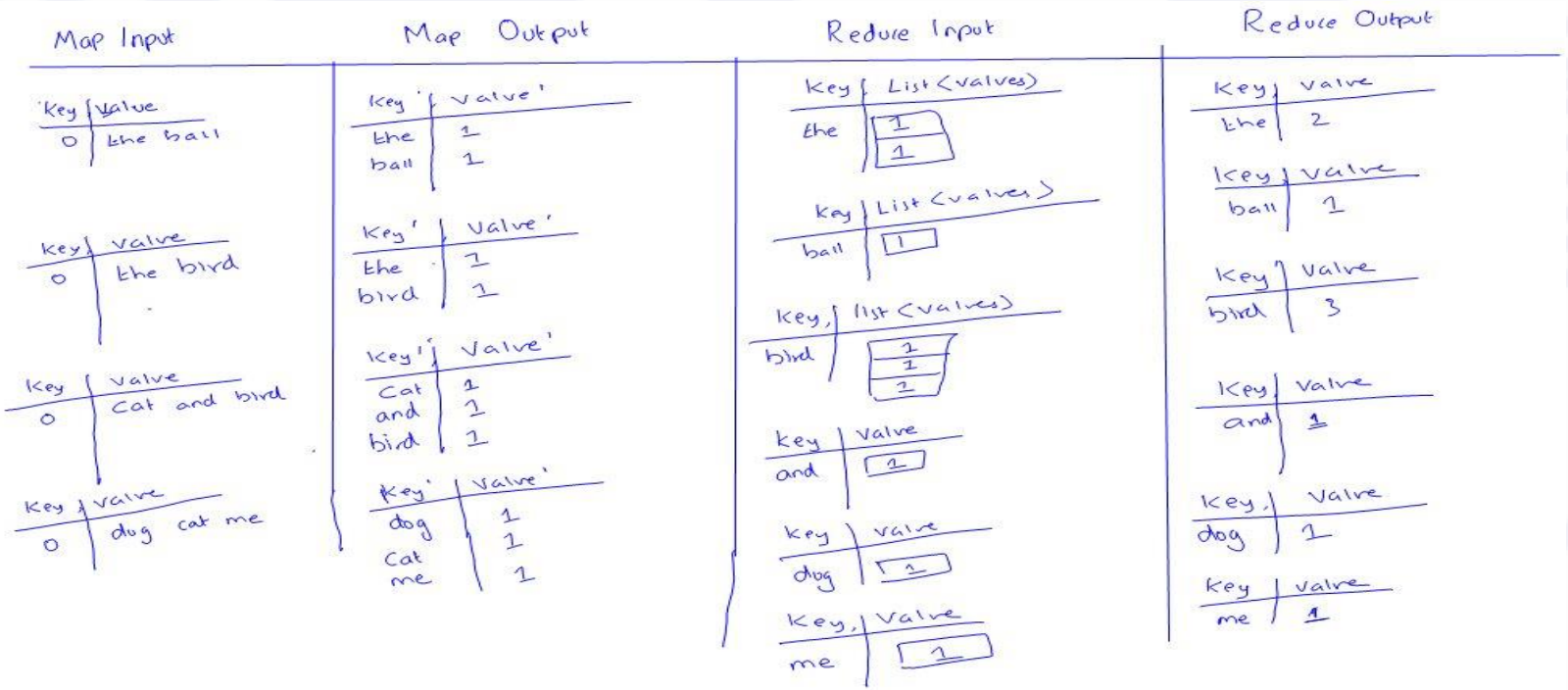
```
int[] x = {1,2,3};
    for(int i = 0; i<3; i++)
      sum + = x[i] ;
```

- Imagine if we can write in the following manner

```
int[] x = {1,2,3};
sum = x.reduce(0, (x,y) = > x + y)
```

- Reduce function is little difficult to comprehend than the map function, the above line states that, it tells, that take two values and reduce that to one value, and to start with one of them is 0.

Map Reduce - Illustration



Map Reduce - Introduction

- Map: If all we have to do is to iterate elements in a huge list and apply some function to each element, in such case the variable part is the function and not the iteration.
- Map abstracts the iteration and accepts the array on which it needs to iterate and the function it needs to apply on each element.
- Reducer: If all we need to do is to iterate all elements in a huge list and apply some combination function, the variable part is the array and the function and not the iteration.
- Reduce abstracts the iteration and accepts array on which it needs to iterate and the function that reduces the array elements into one single value.

Map Reduce - Introduction

- When all we need to do is to apply some function to all elements in array, does it really matter in what order you have to do it.
- It does not matter. We can suddenly split the list into two and give them to two thread and ask them to apply the map function on the elements in the list.
- Suddenly the job takes ½ the time it took before the list was split.
- See we don't have the burden on lock and synchronization. In Map Reduce an element (call it a row) is looked only by one mapper, and a key is processed by only one reducer, one reducer takes all values of a key and process them.
- Hadoop / MR abstracts the concepts behind the map and reduce logic and provided a framework so that, all we need to do is, give the logic that happens in map and the logic that happens in reduce, this is done by implementing map and reduce function.

Map Reduce - Introduction

- MR abstracts the concepts of looping.
- Please take the pain of learning what it means by functional programming, as without having an understanding on closure, function object, and functional programming it's very difficult to comprehend map reduce beyond the superficial level.
- To get to the depths of Map Reduce, one need to know the reason behind map and reduce and not just the API level details.

Map Reduce & Shuffle

Operation	Input	Function	Output
Map	Key, Value	Projection, Filter, Transform	Key', Value'
Shuffle	Key',Value'	Shuffle and Sort	sort(partition(List(Key',List(Vaue')))
Reduce	Key, List<Value>	Aggreation	Key, Value"

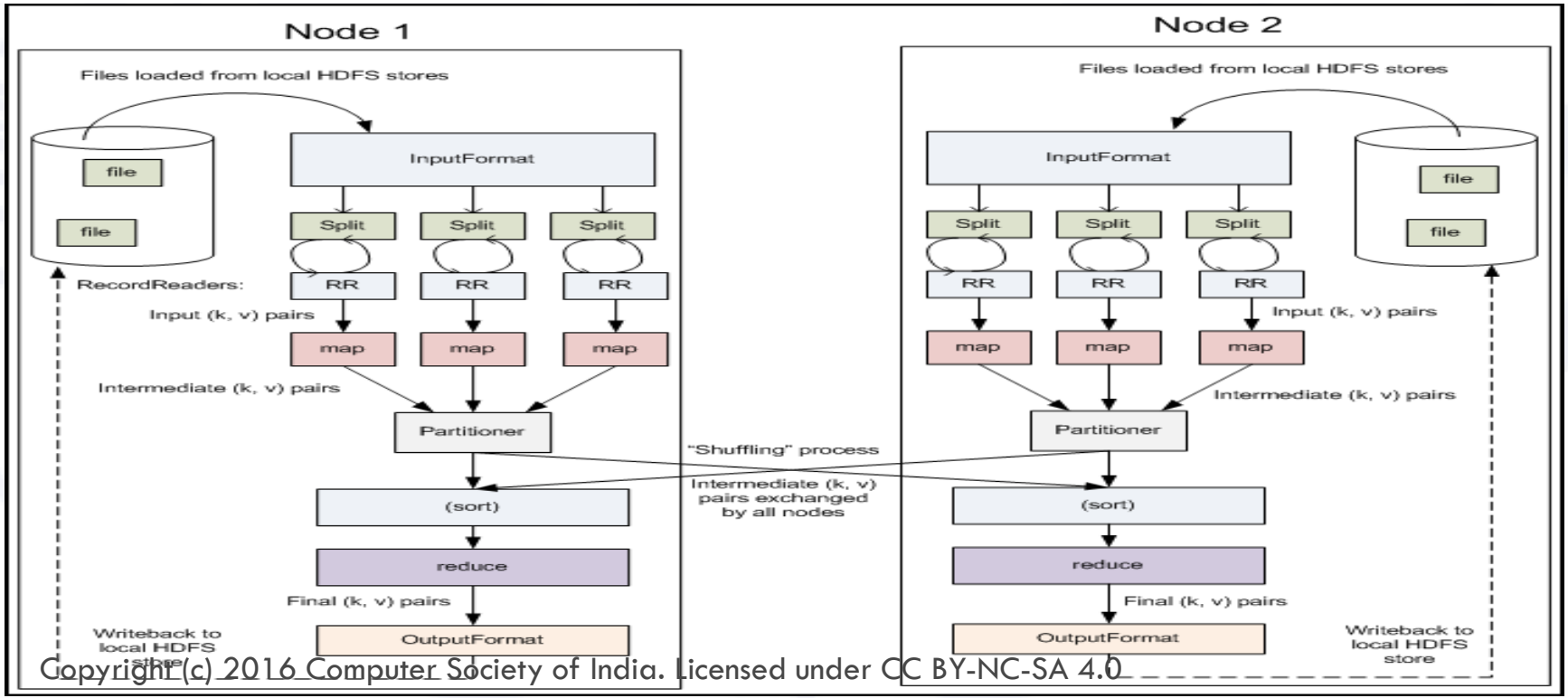
Map – Input

Input Formater	Key Type	Value Type
TextInputFormat	File Offset	Text (the entire line)
KeyValueTextInputFormat	Text (the first string split by tab)	The rest of the line is given as value as Text
SequenceFileInputFormat	User defined	User defined

Map – Output

Output Format	Format
TextOutputFormat	Key '\t' value '\n'
SequenceFileOutputFormat	Binary serialized keys and value , used by shuffle and sort by Map Reduce framework.
Null	Discard , as null output in linux

Map Reduce – Dataflow



Problem

User Event Time

---

mmaniga LOGIN 12345678

Ram LOGIN 12345678

Ram LOGOUT 12345975

…

Mmaniga LOGOUT 12345975

Dell LOGIN 12345977

..

DELL LOGOUT 12345999

Input to the problem is 1..n log files each of the above format, each line is a log event for a particular user, currently there are only two events, start and stop and each event have a time stamp associated with it. The task is to write a map reduce program to process this huge logs directory and produce one single file which has the total time spent by the user (that is start – stop)

Mapper

```
public class Mapper extends Mapper<LongWritable, Text, Text, LongWritable> {
        private final static LongWritable time = new LongWritable();
        private Text names = new Text();

        public void map(LongWritable key, Text value,Context context)
                                                throws IOException {
                String[] lineinput = value.toString().split(",");
                names.set(lineinput[0]);
                time.set(Long.parseLong(lineinput[2]));
                context.write(names, time);

        }

}
```

Reducer

```
public class UserTimeReducer extends
        Reducer<Text, LongWritable, Text, LongWritable> {
        long start = 0;
        long stop = 0;
        public void reduce(Text key, Iterator<LongWritable> values, Context context) throws IOException {
                start = values.next().get();
                stop = values.next().get();
                 // Note: Since we jave just start and stop as two
                 // event, this logic is okay, the day we have
                 // more event we have to iterate all the events
                 // and have a detailed code
                long time = (stop - start);
                context.write(key, new LongWritable(time));
        }
}
```

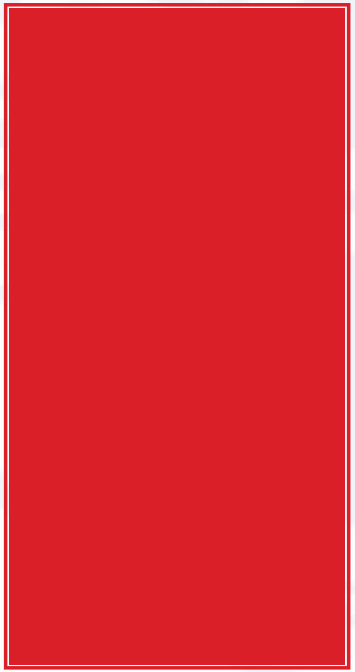
Understanding MapReduce

- Programming model based on simple concepts
  - Iteration over inputs
  - Computation of Key Value Pair from each piece of input
  - Grouping of intermediate value by key
  - Iteration over resulting group
  - Reduction of each group

Recap Hadoop

- Hadoop is a framework for solving large scale distributed application, it provides a reliable shared storage and analysis system.
- The storage is provided by Hadoop Distributed File System and analysis provided by Map Reduce, both are implementation of Google Distributed File System (Big Table) and Google Map Reduce.
- Hadoop is designed for handling large volume of data using commodity hardware and can process those large volume of data.

Session 3 – Hadoop Hands-On



Session 4 – Map Reduce Hands-on

Map Reduce – Problem Solving

- So far we have seen map reduce program where the job was to handle input from a single file location and multiple files.
- Real problems will have multiple input paths, each having different type of data.
- Input formatter for each of those files types might be different, one might have TextInputFormater and one might have Custom input formatter.
- Since each input path holds different type of data, it directly translates that they might need different mapper function, a single mapper function would no more be sufficient.
- We are going to look into problems that have multiple mappers and single reducer.

Map Reduce – Problem Solving

- Problem Statement:
  - A company has a list of user id (user name) and there mobile number. String:UserName, Number:MobileNUmber format.
  - The company gives the list of mobile number to a third part bulk sms vendor to send sms to all the mobile number, the company does not share the name of the user.
  - The third party produces a list of mobile number and their sms delivery status and this file is given to the primary company.
  - The primary company has to create a new file which has name , mobile number and their delivery status. A simple problem to solve in Database as long as data is small.
  - This is one example of map reduce and how its used to solve real world problems.
  - This is a join problems map reduce is designed to solve. What we also learn is to deal with multiple mappers and single reducer concepts.

Map Reduce – Problem Solving

- Problem Statement
  - The problem has two input file types, one is user details and one is delivery details and they have a common data called mobile number.
  - In database we can do a join to solve this problem.

Name	Mobile
Mani	1234
Vijay	2341
Ravi	3452

Mobile	Status
1234	0
2341	2
3452	3

Name	Mobile	Status
Mani	1234	0
Vijay	2341	2
Ravi	3452	3

Map Reduce – Problem Solving

- Mapper
  - We need two mappers, one to process the customer details and one to process the Delivery details.
  - The key concept is the key, to identify what is the key in each input data and make sure that data from different input files for the same key reaches the same reducer.
  - In our scenario, the key is the mobile number, so in each of the mapper job find the key from the data and make the remaining things as value and send to mapper.
- Mapper problem
  - How will reducer know which data belong to customer details and which data belongs to the delivery details.
  - How do we make sure multiple mappers are run and then the reducer kicks in.
  - What is there in map reduce framework to solve this problem.

Map Reduce – Problem Solving

- Multiple Mapper
  - Map reduce supports a mechanism to handle multiple input paths and associate a input formatter along with them.
    - MultipIInputs is the class to use
  - MultipleInputs also takes a mapper class as its arguments, so that we can directly tell the input path and the input formatter as well as its mapper functionality.

    static void addInputPath(Job job, org.apache.hadoop.fs.Path path, Class<? extends InputFormat> inputFormatClass, Class<? extends Mapper> mapperClass)

    Example:

```
MultipleInputs.addInputPath(job, new Path(args[0]),
TextInputFormat.class,
CustomerDetailsMapper.class);

MultipleInputs.addInputPath(job, new Path(args[1]),
TextInputFormat.class,
DeliveryStatusMapper.class);
```

Map Reduce – Problem Solving

- MultipleInputs
  - This class supports MapReduce jobs that have multiple input paths with a different InputFormat and Mapper for each path.
  - The problem with associating a mapper per input file type is solved using the map reduce framework.
  - The next problem is to identify the key, so each mapper should treat the same data as key, in our case it's the mobile number, so the CustomerDetailMapper and DeliveryStatusMapper should look for mobile number in respective input files and treat that as key and send the remaining things as value.
  - By doing this we get different values for the same key from different input source.

Map Reduce – Problem Solving

- MultipleInputs solved only one aspect of the problem, that is, it ran different mapper on different input source.
- It also solved the problem of extracting key and associating data and sending to reducer.
- The missing part is, how would reducer know which data is from customer details and which is data from delivery details. Without reducer knowing it, it would be difficult for it to process the data.
- To Solve this problem, in the mapper we enrich the data by associating some tag, or additional details like type of data, which the reducer can make use.

Map Reduce – Problem Solving

```
public class CustomerDetailsMapper extends
Mapper<LongWritable,Text, Text, Text>
{
private String cellNumber;
private String customerName;
private String tag="Cust~";
public void map(LongWritable key, Text value, Context
context) throws IOException,InterruptedException {
        String line = value.toString();
        String splitarray[] = line.split(",");
        cellNumber = splitarray[0].trim();
        customerName = splitarray[1].trim();
        context.write(new Text(cellNumber),
                new Text(tag+customerName));
    }
}
```

```
public class DeliveryStatusMapper extends
Mapper<LongWritable, Text, Text, Text>
{
private String cellNumber;
private String deliveryCode;
private String tag="Delivery~";

public void map(LongWritable key, Text value,
Context context)
throws IOException, InterruptedException
   {
       String line = value.toString();
       String splitarray[] = line.split(",");
       cellNumber = splitarray[0].trim();
       deliveryCode = splitarray[1].trim();
       context.write(new Text(cellNumber),
               new Text(tag+deliveryCode));
    }
}
```

Map Reduce – Problem Solving

```
public class CustDeliveryReducer extends
Reducer<Text, Text, Text, Text> {
private String customerName;
private String deliveryReport;

public void reduce(Text key, Iterator<Text> values, Context context)  throws IOException  {
    while (values.hasNext())  {
            String currValue = values.next().toString();
            String valueSplitted[] = currValue.split("~");
            if(valueSplitted[0].equals("Cust"))  customerName=valueSplitted[1].trim();
                        else if(valueSplitted[0].equals(" Delivery "))
                         deliveryReport = valueSplitted[1].trim();
    }
    context.write(new Text(customerName), new Text(deliveryReport));
}
}
```

Map Reduce – Graph Problem Solving

- Graph
  - Collection of nodes and links, a link connects two nodes.
  - Graph = (V,E)
    - V Represents the set of vertices or nodes
    - E Represents the set of edges or links
    - A Node represent some concept and a link represent some relation between two nodes.
  - Graphs Examples
    - Web
    - Network
    - Social Network
    - Semantic Net
    - Road Map
    - Etc.,

Map Reduce – Graph Problem Solving

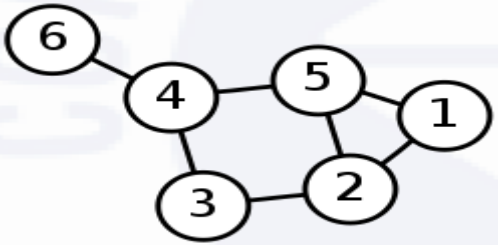
- Graph Algorithms
  - Traversal or visiting all the nodes
  - Traversal or visiting a specific node
  - Finding the shortest path to a particular node
  - Finding all path to a particular node
  - Processing link structure
  - Finding similarity inside graph
  - Processing Nodes and Links
  - Etc.,

Map Reduce – Graph Problem Solving

- Graph Representation
  - General Representation G = (V,E) is constrained that is, its not suitable to represent large graph and its not suitable to serialize the graph.
  - Adjacency Matrix is again constrained because in general it forms a sparse matrix
  - Adjacency list is the best form of graph representation as its efficient and can easily be serialized and retrieved and can be used to process in parallel

Map Reduce – Graph Problem Solving

Graph G = (V,E)



Adjacency Matrix

	1	2	3	4	5	6
1		1			1	
2	1				1	
3		1		1		
4					1	1
5	1	1		1		
6				1		

Adjacency List

1	2	5	
2	1	3	5
3	4	2	
4	5	3	
5	4	2	1
6	4		

Serializable Form

Nodes = {1, 2, 3, 4, 5, 6}

Links = {{1, 2}, {1, 5}, {2, 3}, {2, 5}, {3, 4}, {4, 5}, {4, 6}}

Map Reduce – Graph Problem Solving

- Problem Breadth First Search
  - Search algorithm on graph, we begin at the root node (or a starting node) and explore all the adjacent nodes, it then explores their unexplored adjacent nodes, and so on, until it finds the desired node or it completes exploring all the nodes.

Map Reduce – Graph Problem Solving

- Graph Representation
  - Adjacency list can be represented in multiple form
  - Version 1

    Nodes = {1, 2, 3, 4, 5, 6}

    Links = {{1, 2}, {1, 5}, {2, 3}, {2, 5}, {3, 4}, {4, 5}, {4, 6}}
  - Version 2

    {
    1{2, 5},
    2{ 3, 5},
    3{4},
    4{5,6}
    }
- Both these version would help as long as the graph is small and we can process them in single machine, these form of graph representation is not good for processing at large scale, distributed graph processing and using map reduce.

Map Reduce – Graph Problem Solving

- Graph Representation for Map Reduce
  - Simple adjacency is enriched with additional data so that a map reduce based graph processing algorithm could be worked out.
    - Nodes should become key
    - Adjacent nodes should become the values
    - Node visited (a color code – white, gray, black) is added as part of value
    - Distance from root node or source node is passed as value
- With these additional details we can think of processing this graph in distributed manner using map reduce.

Map Reduce – Graph Problem Solving

- Graph Map Reduce
  - Key : Node
  - Value : List<Node> | Distance From Source | Node Status (Color Code)
- List<Node> is comma delimited node id's that are connected to the node represented by the Node for Key.
- Distance is initially infinity (Integer.MAX_VALUE) except for source node which is zero.
- Node Status, which is an enumeration of {White, Gray, Black) representing not visited, visited, explored fully.

Map Reduce – Graph Problem Solving

- Example of Graph Map Reduce

Key	Value		
	Nodes	Distance	Status
1	2,5	0	GRAY
2	1,3,5	Integer.MAX_VALUE	WHITE
3	4,2	Integer.MAX_VALUE	WHITE
4	5,3	Integer.MAX_VALUE	WHITE
5	4,2,1	Integer.MAX_VALUE	WHITE
6	4	Integer.MAX_VALUE	WHITE

Map Reduce – Graph Problem Solving

- Mapper
  - The logic is simple, for all Gray node, the mapper emits a new gray node. The mapper explodes the gray nodes. The new gray nodes have distance that is incremented by one. The source node is emitted as black. It emits the other non gray node (that is white and black) as it is.

Map Reduce – Graph Problem Solving

Mapper Exploding Nodes

Original Input

Key	Value		
	Nodes	*Distance*	*Status*
1	2,5	0	GRAY
2	1,3,5	Integer.MAX_VALUE	WHITE
3	4,2	Integer.MAX_VALUE	WHITE
4	5,3	Integer.MAX_VALUE	WHITE
5	4,2,1	Integer.MAX_VALUE	WHITE
6	4	Integer.MAX_VALUE	WHITE

Exploded by Mapper – Iteration 1

Key	Value		
	Nodes	*Distance*	*Status*
1	2,5	0	BLACK
2	NULL	1	GRAY
5	NULL	1	GRAY
2	1,3,5	Integer.MAX_VALUE	WHITE
3	4,2	Integer.MAX_VALUE	WHITE
4	5,3	Integer.MAX_VALUE	WHITE
5	4,2,1	Integer.MAX_VALUE	WHITE
6	4	Integer.MAX_VALUE	WHITE

Map Reduce – Graph Problem Solving

- Reducer
  - As usual the reducer gets all the values for a particular key (the node)
  - Reducer creates new nodes for all the non null nodes, sets the distance as the minimum of all, and the max values of the color in the list.

Reducer Input

Key = 2
NULL | 1 | GRAY
1,3,5 | Integer.MAX_VALUE | WHITE

Map Reduce – Graph Problem Solving

- Reducer creates new nodes based on the following rule.
  - Pick up those rows that have non null list of nodes.
    - For Key = 2, we had two rows, one with nodes = null, and another with nodes = 1,3,5
    - Pick up the row that has nodes = 1,3,5
  - From all the rows pick the distance that is minimum of all
  - Choose the darkest color of all.
- Doing that we would end up creating the following row
  - 2 1,3,5 | 1 |GRAY

Graph After Reducer – Iteration 1

Key	Value		
	Nodes	Distance	Status
1	2,5	0	BLACK
2	1,3,5	1	GRAY
3	4,2	Integer.MAX_VALUE	WHITE
4	5,3	Integer.MAX_VALUE	WHITE
5	4,2,1	Integer.MAX_VALUE	WHITE
6	4	Integer.MAX_VALUE	WHITE

Map Reduce – Graph Problem Solving

- Mapper – Iteration 2
  - For every node that is gray emit another gray node, the new node is created by this rule
    - Distance = Distance + 1

Input After Map-Reduce Iteration 1

Key	Value		
	Nodes	*Distance*	*Status*
1	2,5	0	BLACK
2	1,3,5	1	GRAY
3	4,2	Integer.MAX_VALUE	WHITE
4	5,3	Integer.MAX_VALUE	WHITE
5	4,2,1	Integer.MAX_VALUE	WHITE
6	4	Integer.MAX_VALUE	WHITE

Exploded by Mapper – Iteration 2

Key	Value		
	Nodes	*Distance*	*Status*
1	2,5	0	BLACK
2	1,3,5	1	BLACK
1	NULL	2	GRAY
3	NULL	2	GRAY
5	NULL	2	GRAY
3	4,2	Integer.MAX_VALUE	WHITE
4	5,3	Integer.MAX_VALUE	WHITE
5	4,2,1	Integer.MAX_VALUE	WHITE
6	4	Integer.MAX_VALUE	WHITE

Map Reduce – Graph Problem Solving

- Reducer creates new nodes based on the following rule.
  - Pick up those rows that have non null list of nodes.
  - From all the rows pick the distance that is minimum of all
  - Choose the darkest color of all.
- Different reducers gets different portions of the data, the illustration as shown below.
- Reduce that gets keys 1,3 and 5 picks those rows highlighted by rectangle and apply the above rule to produce the next set of input for next map reduce phase

Key	Value		
	Nodes	*Distance*	*Status*
1	2,5	0	BLACK
1	NULL	2	GRAY

Key	Value		
	Nodes	*Distance*	*Status*
4	5,3	Integer.MAX_VALUE	WHITE

Key	Value		
	Nodes	*Distance*	*Status*
2	1,3,5	1	BLACK

Key	Value		
	Nodes	*Distance*	*Status*
5	NULL	2	GRAY
5	4,2,1	Integer.MAX_VALUE	WHITE

Key	Value		
	Nodes	*Distance*	*Status*
3	NULL	2	GRAY
3	4,2	Integer.MAX_VALUE	WHITE

Key	Value		
	Nodes	*Distance*	*Status*
6	4	Integer.MAX_VALUE	WHITE

Map Reduce – Graph Problem Solving

- Graph After Reducer – Iteration 2

Key	Value		
	Nodes	Distance	Status
1	2,5	0	BLACK
2	1,3,5	1	BLACK
3	4,2	2	GRAY
4	5,3	Integer.MAX_VALUE	WHITE
5	4,2,1	2	GRAY
6	4	Integer.MAX_VALUE	WHITE

Session 5 - HBase

- Advanced Introduction to Hbase
- Learning NoSQL
- Creating Table – Shell and Programming
- Understanding Column Families
- Unlearning Entity Relation
- Learning Column Value & Key Pair
- Unlearning Index & Query
- Learning Scan
- MapReduce and HBase – Importing into HBase

HBase – Advanced Introduction

- Open Source, horizontally scalable, sorted map data built on top of Apache Hadoop.
- Datamodel inspired and similar to Google's BigTable.
- Goal is to provide quick random read / write access to massive amount of structured data.
- Layered over HDFS
- Tolerant of Machine Failures
- Strong Consistency model

HBase Features

- Strongly consistent reads/writes.
- Automatic sharding: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- Automatic RegionServer failover
- Hadoop/HDFS Integration: HBase supports HDFS out of the box as it's distributed file system.
- MapReduce: HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- Java Client API: HBase supports an easy to use Java API for programmatic access.
- Thrift/REST API: HBase also supports Thrift and REST for non-Java front-ends.
- Operational Management: HBase provides build-in web-pages for operational insight as well as JMX metrics.

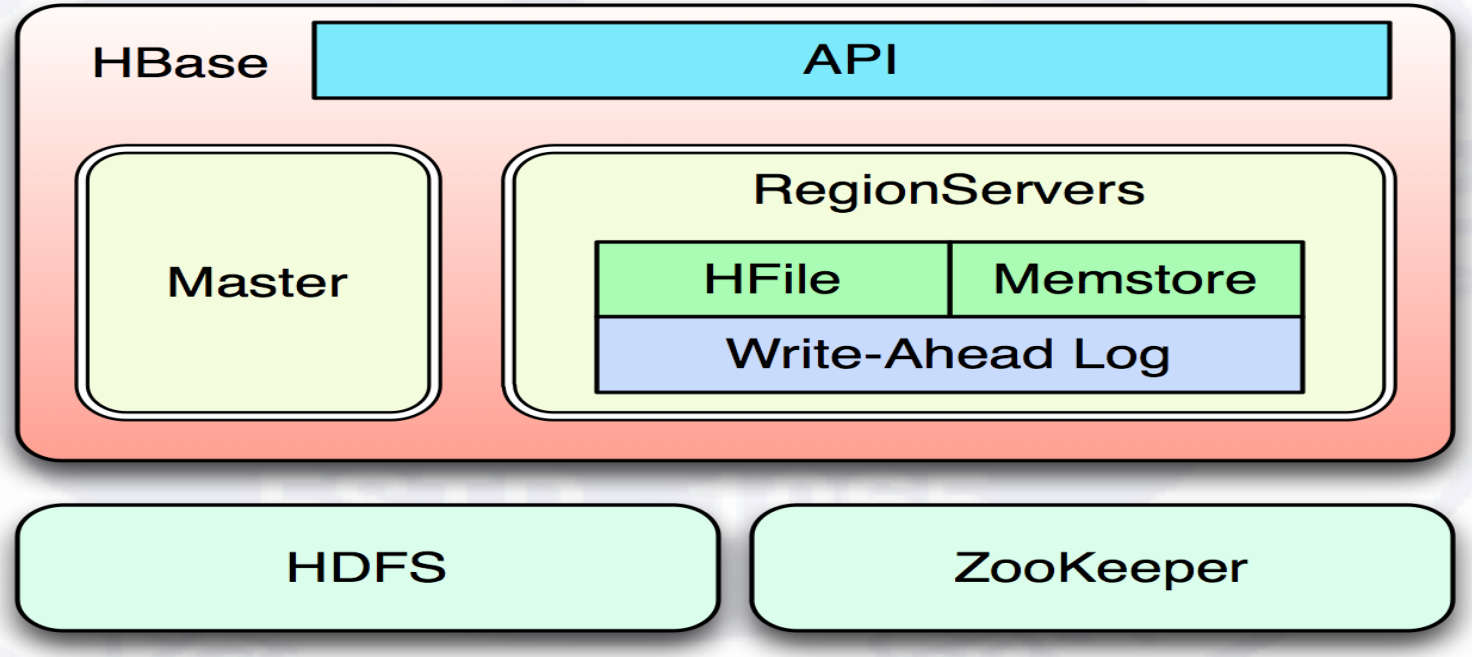
When HBase

- If we have hundreds of millions or billions of rows, then HBase is a good candidate.
- If we have a few thousand/million rows, then using a traditional RDBMS might be a better choice.
- If we can we live without the extra features that an RDBMS provides (e.g., typed columns, secondary indexes, transactions, advanced query languages, etc.)
- An application built against an RDBMS cannot be "ported" to HBase by simply changing a JDBC driver.
- Consider moving from an RDBMS to HBase as a complete redesign as opposed to a port.
- There should be enough hardware. Even HDFS doesn't do well with anything less than 5 DataNodes (due to things such as HDFS block replication which has a default of 3), plus a NameNode.
- HBase can run quite well stand-alone on a laptop - but this should be considered a development configuration only.

HDFS vs HBase

- HDFS is a distributed file system that is well suited for the storage of large files. It's documentation states that it is not, however, a general purpose file system, and does not provide fast individual record lookups in files.
- HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables.
- HBase internally puts data in indexed "StoreFiles" that exist on HDFS for high-speed lookups

HBase Architecture



HBase Architecture

- Uses HDFS as its reliable storage layer
- Access through native Java API, Avro, Thrift, REST, etc.,
- Master manages cluster
- RegionServer manages data
- ZooKeeper for bootstrapping and coordinating the cluster

HBase Architecture

- What is missing (in HDFS), though, is the ability to access data randomly and in close to real-time
- It's good with a few very, very large files, but not as good with millions of tiny files.
- The more files, the higher the pressure on the memory of the NameNode.
- Hbase was designed to drive interactive applications, such as Mail or Analytics, while making use of the same infrastructure and relying on HDFS for replication and data availability.
- The data stored is composed of much smaller entities
- Transparently take care of aggregating small records into very large storage files
- Indexing allows the user to retrieve data with a minimal number of disk seeks.
- Hbase could in theory can store the entire web crawl and work with MapReduce to build the entire search index in a timely manner.

Tables, Rows, Columns, and Cells

- the most basic unit is a *column*.
- One or more columns form a *row* that is addressed uniquely by a *row key*.
- A number of rows, in turn, form a *table*, and there can be many of them.
- Each column may have multiple versions, with each distinct value contained in a separate *cell*.
- Rows are composed of *columns*, and those, in turn, are grouped into *column families*.
- Columns are often referenced as *family:qualifier* with the qualifier being any arbitraryarray of bytes.

Tables, Rows, Columns, and Cells

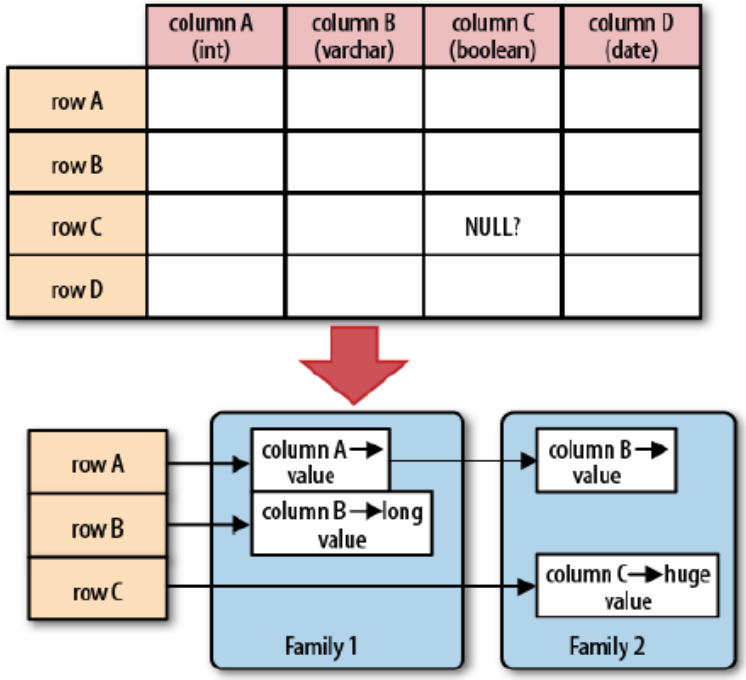
- All *rows* are always sorted lexicographically by their row key.
  - *Example The sorting of rows done lexicographically by their key*
  - hbase(main):001:0> scan 'table1'

```
ROW COLUMN+CELL
row-1      column=cf1:, timestamp=1297073325971 ...
row-10     column=cf1:, timestamp=1297073337383 ...
row-11     column=cf1:, timestamp=1297073340493 ...
row-2      column=cf1:, timestamp=1297073329851 ...
row-22     column=cf1:, timestamp=1297073344482 ...
row-3      column=cf1:, timestamp=1297073333504 ...
row-abc    column=cf1:, timestamp=1297073349875 ...
```

- In lexicographical sorting, each key is compared on a binary level, byte by byte, from left to right. Since row-1... is less than row-2..., no matter what follows, it is sorted first.

Tables, Rows, Columns, and Cells

- Rows and Column representation between RDBMS and Hbase
- In Hbase rows and columns are not arranged like the classic spreadsheet model, but rather use a tag metaphor, that is, information is available under a specific tag.
- The "NULL?" in Figure indicates that, for a database with a fixed schema, we have to store NULLs where there is no value, but for HBase's storage architectures, we simply omit the whole column;



Tables, Rows, Columns, and Cells

- We can express the access to data like :

  (Table, RowKey, Family, Column, Timestamp) → Value

- In a more programming language style, this may be expressed as:

  SortedMap<RowKey, List<SortedMap<Column, List<Value, Timestamp>>>>

- The first SortedMap is the table, containing a List of column families.
- The families contain another SortedMap, which represents the columns, and their associated values.
- These values are in the final List that holds the value and the timestamp it was set.
- The API, by default, automatically picks the most current value of each cell.

Representation of Data in a row

Row Key	Time Stamp	Column "data:"	Column "meta:" "mimetype"	Column "meta:" "size"	Column "counters:" "updates"
"row1"	t_3	"{ "name":"lars","address":...}"		"2323"	"1"
	t_6	"{ "name":"lars","address":...}"			"2"
	t_8		"application/json"		
	t_9	"{ "name":"lars","address":...}"			"3"

Auto Shrading

- Unit of scalability in Hbase is the Region
- Sorted, contiguous range of rows
- Spread “randomly ”across RegionServer
- Moved around for load balancing and failover
- Split automatically or manually to scale with growing data
- Capacity is solely a factor of cluster nodes vs. regionspernode

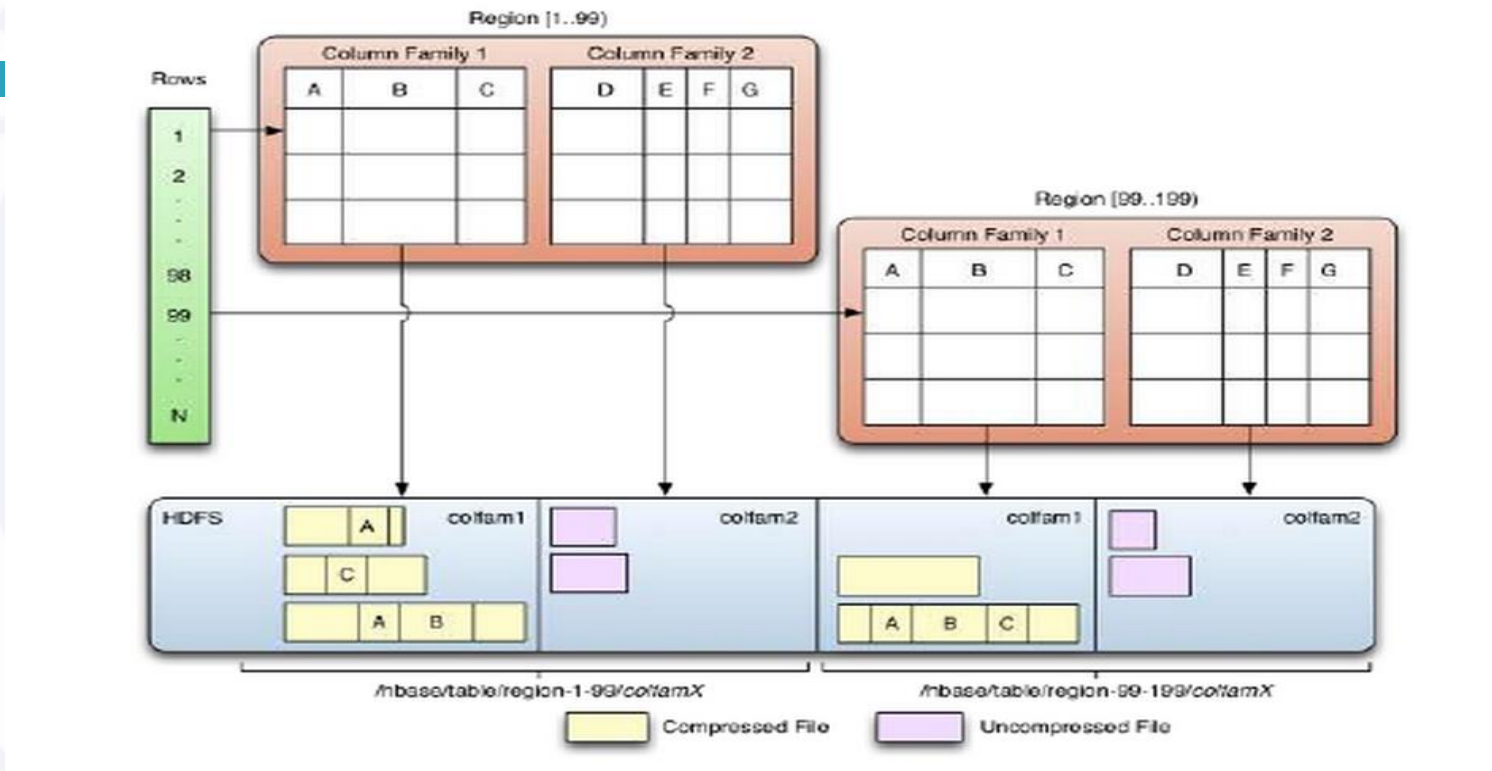
HBase ACID

- HBase **not ACID-compliant,** but does guarantee certain specific properties
  - **Atomicity**
    - All mutations are atomic within a row. Any put will either wholely succeed or wholely fail.
    - APIs that mutate several rows will *not* be atomic across the multiple rows.
    - The order of mutations is seen to happen in a well-defined order for each row, with no interleaving.
  - **Consistency** and **Isolation**
    - All rows returned via any access API will consist of a complete row that existed at some point in the table's history.

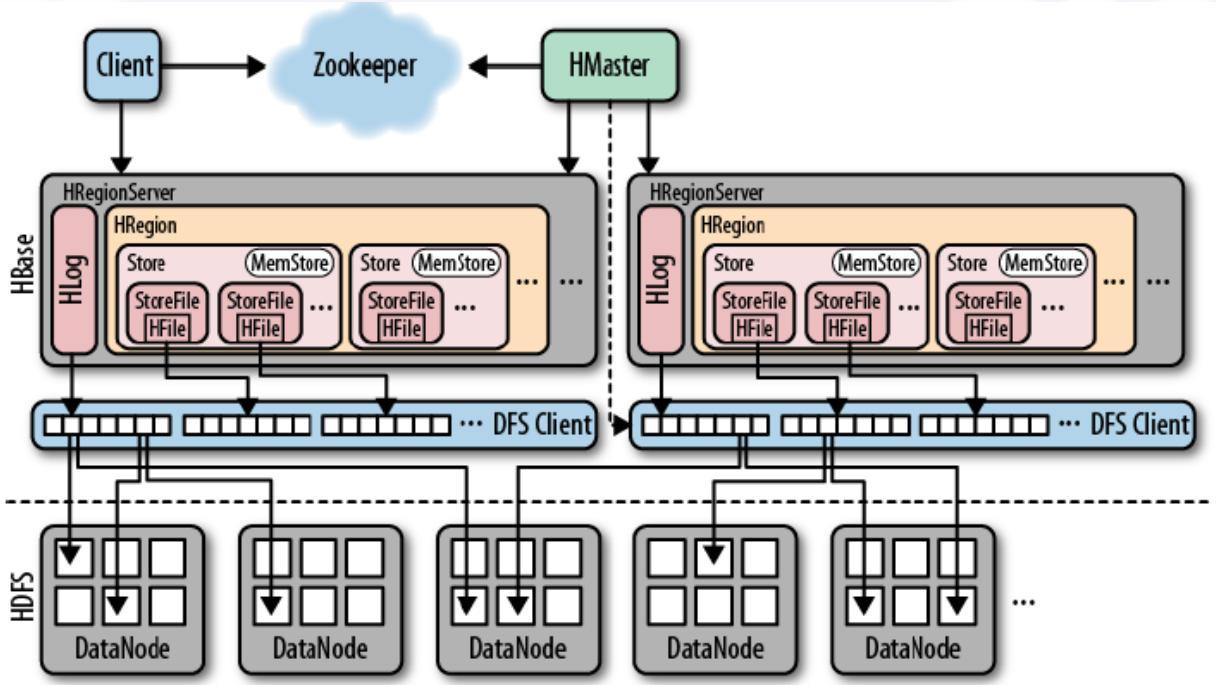
HBase ACID

- **Consistency** of Scans
  - A scan is not a consistent view of a table. Scans do not exhibit snapshot isolation.
  - Those familiar with relational databases will recognize this isolation level as "read committed".
- **Durability**
  - All visible data is also durable data. That is to say, a read will never return data that has not been made durable on disk.
  - Any operation that returns a "success" code (e.g. does not throw an exception) will be made durable.
  - Any operation that returns a "failure" code will not be made durable (subject to the Atomicity guarantees above).
  - All reasonable failure scenarios will not affect any of the listed ACID guarantees.

Column Family



HBase handling files in File system



Seek Versus Transfer

- Storage Architecture
  - RDBMS use B+ Tree
    - Operates at disk seek rate
    - Log(N) seeks per access (b+ tree depth is log(n)
  - NoSQL use Log Structured merge tree
    - Operates at disk transfer rate
    - Incoming data is stored in a logfile first, completely sequentially. It then updates an in-memory store that holds the most recent updates for fast lookup

Hive - Introduction

- If we need a large warehouse – Multi petabyte of data.
- Want to leverage SQL knowledge.
- Want to leverage hadoop stack and technology to do warehousing and analytics.

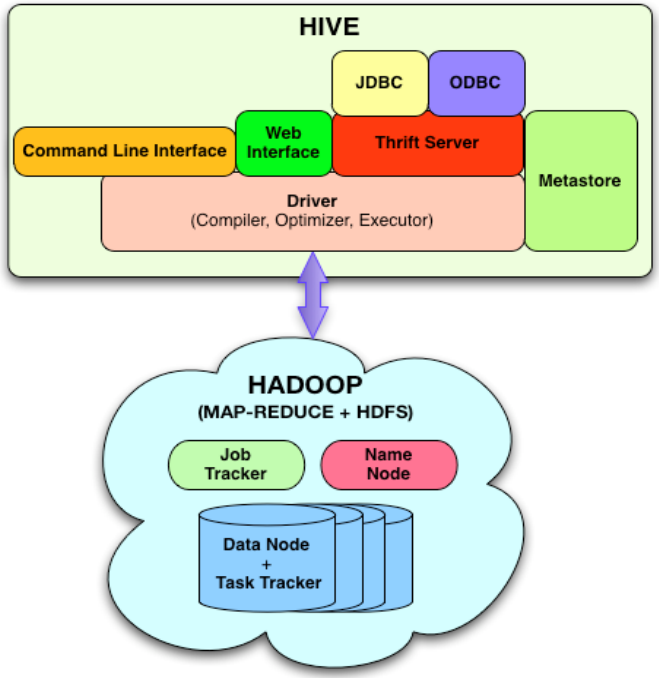
Hive - Introduction

- Hadoop based system for querying and analyzing large volumes of STRUCTURED Data.
- Hive uses map reduce for execution
- Hive uses HDFS for storage
- Hive uses RDBMS for metadata / schema storage, Derby is the default RDBMS but can be changed to MySQL.

Hive – Where not to use

- If data does not cross GB's.
- If we don't need schema or brining in schema is difficult or not possible on the data at hand.
- If we need response in seconds, and for low latency application.
- If RDBMS can solve, don't invest time in Hive.

Hive Architecture



Data Model

- Tables
  - Typed columns (int, float, string, boolean)
  - Also, list: map (for JSON-like data)
- Partitions
  - For example, range-partition tables by date
- Buckets
  - Hash partitions within ranges (useful for sampling, join optimization)

Metastore

- Database: namespace containing a set of tables
- Holds table definitions (column types, physical layout)
- Holds partitioning information
- Can be stored in Derby, MySQL, and many other relational databases

Physical Layout

- Warehouse directory in HDFS
  - E.g., /user/hive/warehouse
- Tables stored in subdirectories of warehouse
  - Partitions form subdirectories of tables
- Actual data stored in flat files
  - Control char-delimited text, or SequenceFiles

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

```
SELECT s.word, s.freq, k.freq FROM shakespeare s JOIN bible k ON (s.word = k.word)
WHERE s.freq >= 1 AND k.freq >= 1 ORDER BY s.freq DESC LIMIT 10;
```

the	25848	62394
I	23031	8854
and	19671	38985

Hive: Behind the Scenes

SELECT s.word, s.freq, k.freq FROM shakespeares JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1 ORDER BY s.freq DESC LIMIT 10;

(Abstract Syntax Tree)

(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (. (TOK_TABLE_OR_COL s) word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k) freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))

(one or more of MapReduce jobs)

Hive - Installation

- Pre-Request
  - Hadoop is installed
  - HDFS is available
- Download hive
- Extract Hive
- Configure Hive

Hive - Setup

tar – xvf hive-0.7.1-bin

cd hive-0.7.1-bin/

export HIVE_HOME=/home/guest/hive-0.8.1-bin

export PATH=/home/guest/hive-0.8.1-bin/bin/:$PATH

export PATH=/home/guest/hadoop/bin/:$PATH

Check Hadoop is in path

developer@ubuntu:~/hive-0.7.1-bin$ hadoop fs -ls

Found 2 items

drwxr-xr-x - developer supergroup 0 2011-06-02 04:01 /user/developer/input

drwxr-xr-x - developer supergroup 0 2011-06-16 02:12 /user/developer/output

Hive – Up and Running

```
hadoop fs -mkdir /user/hive/warehouse
hadoop fs -mkdir tmp
hadoop fs -chmod g+w /user/hive/warehouse
hadoop fs -chmod g+w tmp
hive
hive>
```

Hive – Create Table

```
hive> create table pokes(foo INT, bar STRING);
OK
Time taken: 8.487 seconds
hive>  show tables;
OK
pokes
Time taken: 0.197 seconds
hive> select * from pokes;
OK
Time taken: 0.198 seconds
```

Hive – Table Commands

```
hive> describe pokes;
OK
foo     int
bar     string
Time taken: 0.116 seconds
```

Hive – Import Data

```
hive> load data local inpath
    '/home/developer/hivedata.txt' overwrite into table
    pokes;
Copying data from file:/home/developer/hivedata.txt
Copying file: file:/home/developer/hivedata.txt
Loading data to table default.pokes
Deleted hdfs://localhost/user/hive/warehouse/pokes
OK
```

Hive – Select Command

```
hive> select * from pokes;
OK
100    Manigandan
200    Kannamma
300    Arkrish
400    NewName
Time taken: 0.159 seconds
```

Hive – Select commands

```
hive> select * from pokes where foo>200;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201112151812_0001, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201112151812_0001
Kill Command = /opt/hadoop-0.20.2/bin/../bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201112151812_0001
2011-12-15 19:58:52,692 Stage-1 map = 0%,  reduce = 0%
2011-12-15 19:58:55,734 Stage-1 map = 100%,  reduce = 0%
2011-12-15 19:58:58,763 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201112151812_0001
OK
300     Arkrish
400     NewName
Time taken: 12.278 seconds
```

Hive Commands – Alter Table

```
hive> alter table pokes add columns (boot INT);
OK
Time taken: 0.201 seconds
hive> describe pokes;
OK
foo     int
bar     string
boot    int
Time taken: 0.127 seconds
hive> select * from pokes;
OK
100     Manigandan NULL
200     Kannamma   NULL
300     Arkrish    NULL
400     NewName    NULL
Time taken: 0.141 seconds
```

Hive Commands - Sum

```
hive> select sum(a.foo) from pokes a;
Total MapReduce jobs = 1
Launching Job 1 out of 1

OK
1000
Time taken: 34.344 seconds
```

Hive – Table Creation

```
hive> create table u_data (userid INT, movieid INT, rating INT, unixtime STRING) row format
    delimited fields terminated by '\t' stored as textfile;
OK
Time taken: 0.037 seconds
hive> describe u_data;
OK
userid      int
movieid     int
rating      int
unixtime    string
Time taken: 0.107 seconds
```

Load Movie Data

```
hive> load data local inpath '/home/developer/ml-
    100k/u.data' overwrite into table u_data;
Copying data from file:/home/developer/ml-100k/u.data
Copying file: file:/home/developer/ml-100k/u.data
Loading data to table default.u_data
Deleted hdfs://localhost/user/hive/warehouse/u_data
OK
Time taken: 0.299 seconds
```

Hive - Queries

```
hive> select count(*) from u_data;
OK
100000
Time taken: 24.612 seconds
hive> select userid,count(userid) from u_data  group by userid ;
923     74
924     82
925     32
926     20
```

Hive - Transformation

- Create a new transformation table for u_data, this table has weekday (Monday to Sunday) instead of a time stamp. This weekday has to computed from timestamp, so an external mapper program has to be run to do this job.

```
hive> create table u_data_new (userid int, movieid int, rating int, weekday int) row format delimited fields terminated by '\t';
OK
Time taken: 7.405 seconds

hive> describe u_data_new;
OK
userid          int
movieid         int
rating          int
weekday         int
Time taken: 0.145 seconds
```

Hive – Extension / Functions

Function Type	Mapping	Example
User Defined Function – UDF	One to one mapping for each row	Concat('co11','col2')
User Defined Aggregate function – UDAF	Many to one row mapping	Sum(num_rating)
User Defined Table Function	One to many row mapping	Explore([1,2,3])

Hive

Hive as:

- a) A Data Warehouse Infrastructure
- b) Definer of a Query Language popularly known as HiveQL (similar to SQL)
- c) Provides us with various tools for easy extraction, transformation and loading of data.
- d) Hive allows its users to embed customized mappers and reducers.

What makes Hive Hadoop popular?

- Hive Hadoop provides the users with strong and powerful statistics functions.
- Hive Hadoop is like SQL, so for any SQL developer the learning curve for Hive will almost be negligible.
- Hive Hadoop can be integrated with HBase for querying the data in HBase whereas this is not possible with Pig. In case of Pig, a function named HbaseStorage () will be used for loading the data from HBase.
- Hive Hadoop has gained popularity as it is supported by Hue.
- Hive Hadoop has various user groups such as CNET, Last.fm, Facebook, and Digg and so on.

Hive

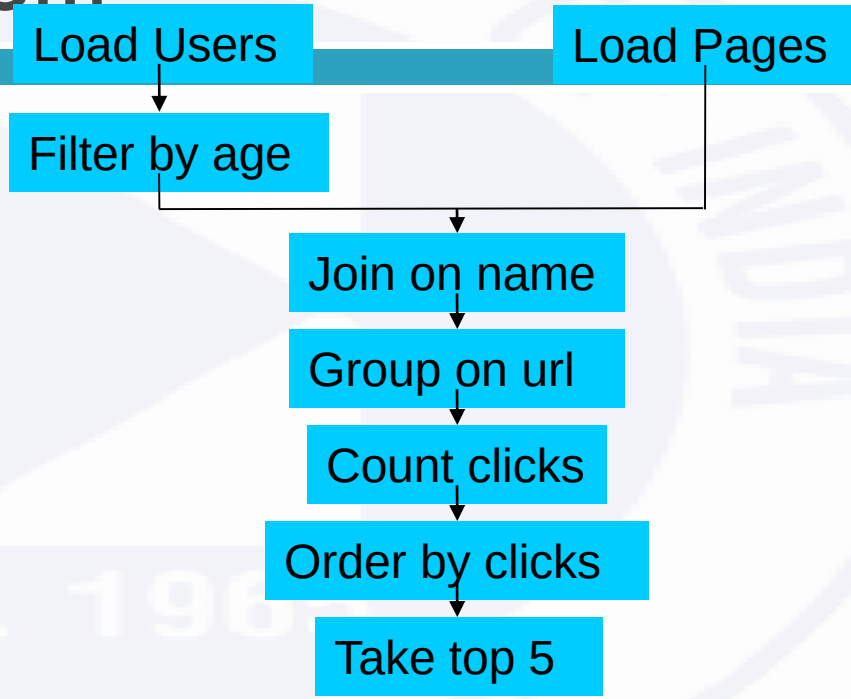
- Developed at Facebook
- Used for majority of Facebook jobs
- "Relational database" built on Hadoop
  - Maintains list of table schemas
  - SQL-like query language (HiveQL)
  - Can call Hadoop Streaming scripts from HiveQL
  - Supports table partitioning, clustering, complex data types, some optimizations

Pig

- Started at Yahoo! Research
- Now runs about 30% of Yahoo!'s jobs
- Features
  - Expresses sequences of MapReduce jobs
  - Data model: nested "bags" of items
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc.)
  - Easy to plug in Java functions

An Example Problem

- Suppose you have user data in a file, website data in another, and you need to find the top 5 most visited pages by users aged 18-25



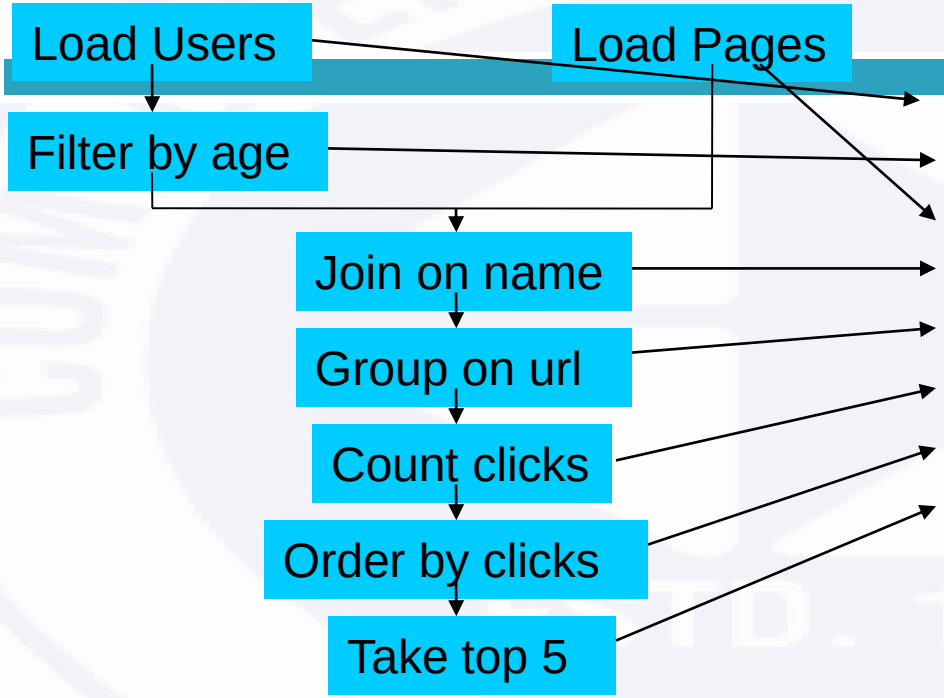
In MapReduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see
            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
                LongWritable key,
                Iterator<Text> iter,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
                new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
                new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

In Pig Latin

```
Users = load ‘users’ as (name, age);
Filtered = filter Users by age >= 18 and age <= 25;
Pages = load ‘pages’ as (user, url);
Joined = join Filtered by name, Pages by user;
Grouped = group Joined by url;
Summed = foreach Grouped generate group,
                 count(Joined) as clicks;
Sorted = order Summed by clicks desc;
Top5 = limit Sorted 5;
store Top5 into ‘top5sites’;
```

Ease of Translation




```
Users = load …
Fltrd = filter …
Pages = load …
Joined = join …
Grouped = group …
Summed = … count()…
Sorted = order …
Top5 = limit …
```