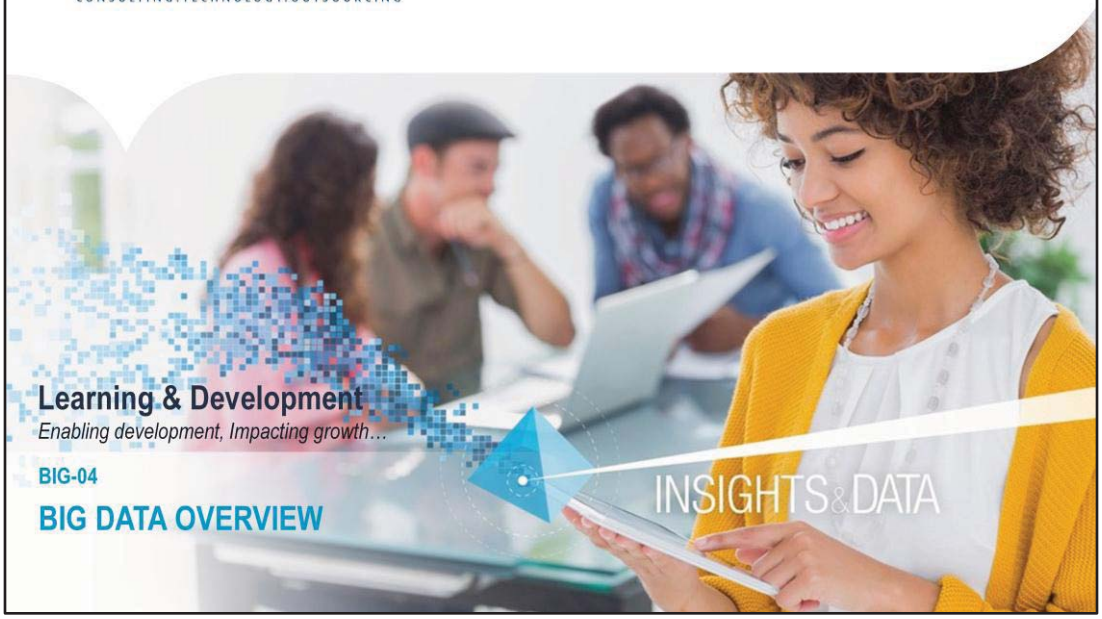
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

People matter, results count.



Learning & Development

Enabling development, Impacting growth…

BIG-04

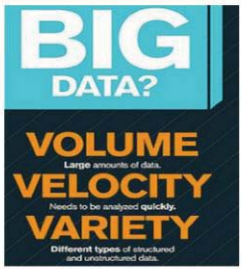
BIG DATA OVERVIEW

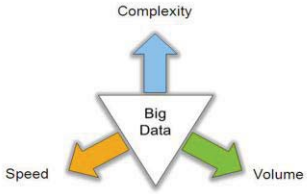
What's Big Data?

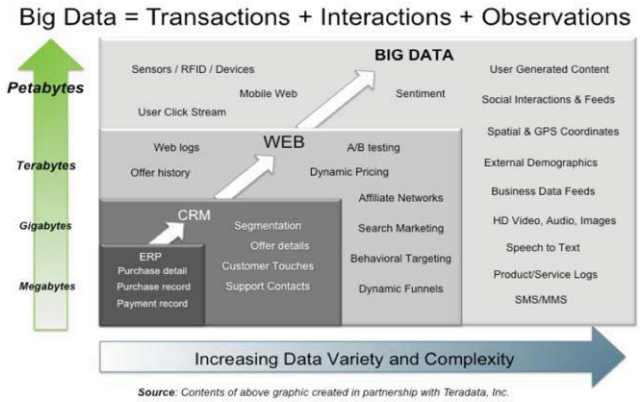
No single definition; here is from Wikipedia:

- **Big data** is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.
- The challenges include capture, curation, storage, search, sharing, transfer, analysis, and visualization.
- The trend to larger data sets is due to the additional information derivable from analysis of a single large set of related data, as compared to separate smaller sets with the same total amount of data, allowing correlations to be found to "spot business trends, determine quality of research, prevent diseases, link legal citations, combat crime, and determine real-time roadway traffic conditions."

Big Data: 3V's



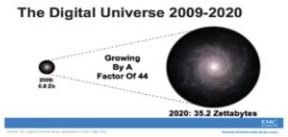


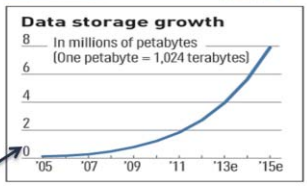


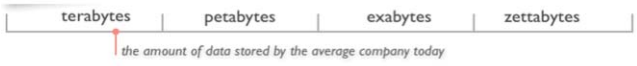
Source: Contents of above graphic created in partnership with Teradata, Inc.

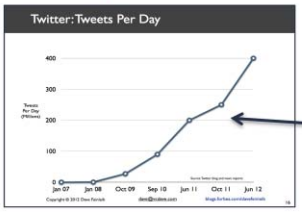
Volume (Scale)

- **Data Volume**
  - 44x increase from 2009 2020
  - From 0.8 zettabytes to 35zb
- Data volume is increasing exponentially



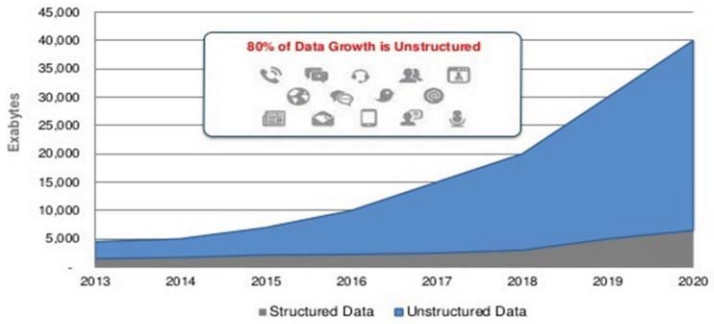






Exponential increase in collected/generated data

Volume (continued)



- By 2020, International Data Corporation predicts the number will reach 40,000 EB, or 40 Zettabytes (ZB) .
- The world's information is doubling every two years. By 2020, there will be 5,200 GB of data for every person on Earth.
- By 2020, the amount of high-value data worth analyzing will double and 60% of information delivered to decision makers will be actionable.

Data Sources for large volume

Velocity(continued)…Real-time/Fast Data

Social media and networks
(all of us are generating data)

Scientific instruments
(collecting all sorts of data)

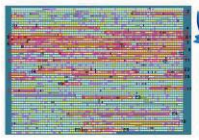
Mobile devices
(tracking all objects all the time)

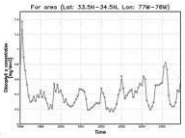
Sensor technology and networks
(measuring all kinds of data)

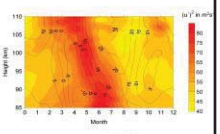
- The progress and innovation is no longer hindered by the ability to collect data
- But, by the ability to manage, analyze, summarize, visualize, and discover knowledge from the collected data in a timely manner and in a scalable fashion

Variety (Complexity)

- Relational Data (Tables/Transaction/Legacy Data)
- Text Data
- XML Data
- Streaming Data
  - Data changing within fraction of seconds
- Audio Data
- Video Data
- Logs Data
- Graph Data
  - Social Network
- A single application may generate/collect different types of data
- Big Public Data (online, weather, finance, etc)





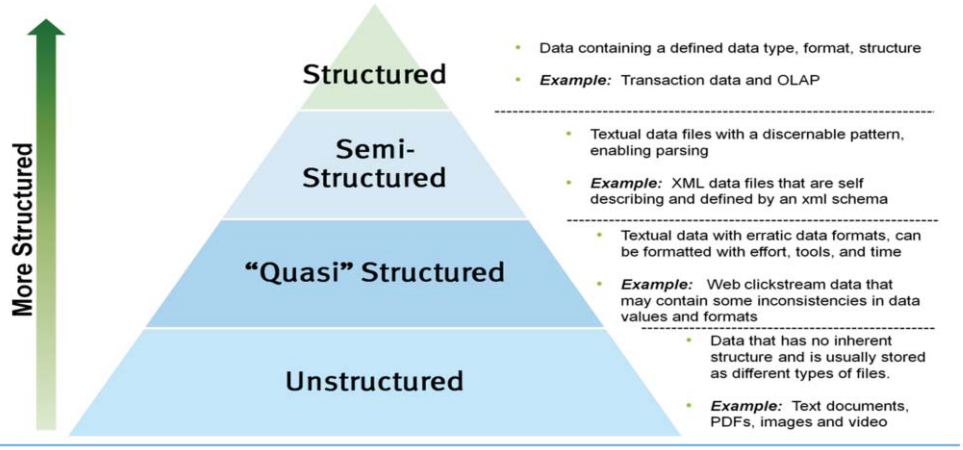


To extract knowledge➔ All these types of data need to linked together

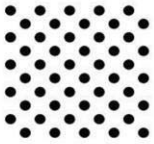
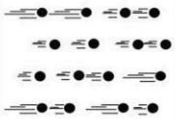
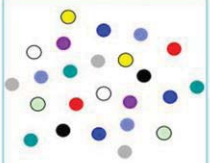
Variety(continued)…Types of Data in Big Data

Big Data Characteristics: Data Structures

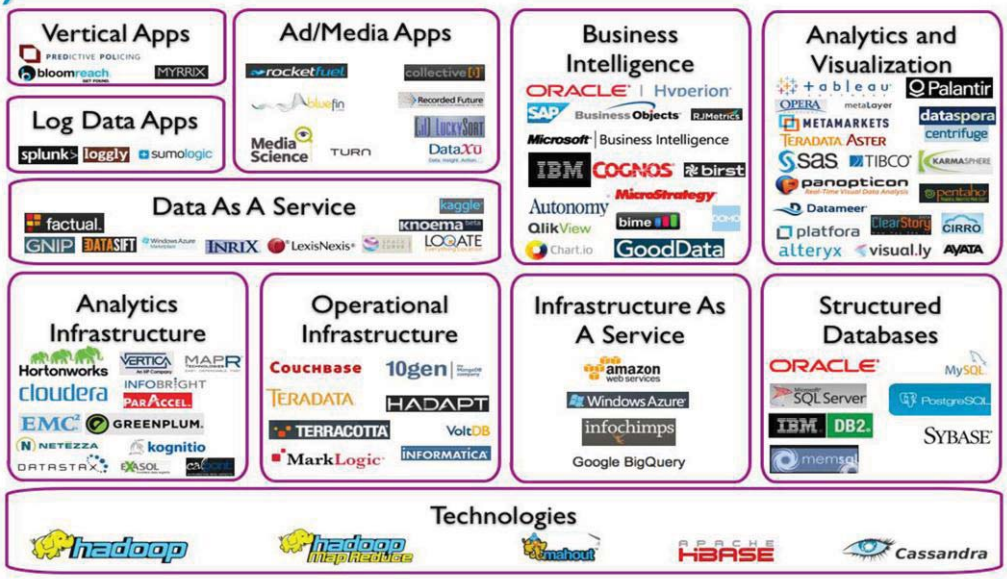
Data Growth is Increasingly Unstructured



Some Make it 4V's

Volume	Velocity	Variety	Veracity*
			
Data at Rest	**Data in Motion**	**Data in Many Forms**	**Data in Doubt**
Terabytes to exabytes of existing data to process	Streaming data, milliseconds to seconds to respond	Structured, unstructured, text, multimedia	Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations

The Big Data Landscape



Applications for Big Data Analytics

Smarter Healthcare

Multi-channel

Finance

Log Analysis

Homeland Security

Traffic Control

Telecom

Search Quality

Manufacturing

Trading Analytics

Fraud and Risk

Retail: Churn

Walmart has an application Savings Catcher: that alerts the customers whenever its neighboring competitor reduces the cost of an item the customer already bought. This application then sends a gift voucher to the customer to compensate the price difference.
Source: https://www.dezyre.com/article/how-big-data-analysis-helped-increase-walmart-s-sales-turnover/109
Ebay uses for click stream analysis.
Hadoop is used in Aadhar enrollment where citizen's data like personal details, photograph, retina etc is stored.

History of Hadoop

- Oct 2003: Google File system paper published
- Dec 2004: Jeffrey Dean & Sanjay Ghemawat from Google published MapReduce paper called “MapReduce: Simplified Data Processing on Large Clusters”
- Jan 2006: Above MapReduce Paper inspired Doug cutting, a yahoo employee then to develop an open source implementation of MapReduce framework
- Jan 2006: Hadoop subproject created as extension of Apache Nutch project, created by Doug Cutting.
- Apr 2006: Hadoop 0.1.0 released
- May 2006: Yahoo deploys 300 machine Hadoop cluster
- 2008: Cloudera, one of the major distributor of Hadoop founded

→ **Hadoop was named after Doug Cutting’s son’s toy elephant.**

History of Hadoop(continued)

- Apr 2007: Yahoo runs 2 clusters of 1,000 machines
- Jul 2008: Hadoop wins TeraByte sort benchmark
  (1st time a Java program won this competition)
- Jun 2010: Yahoo 4,000 nodes/70 petabytes
- Jun 2010: Facebook 2,300 clusters/40 petabytes
- Dec 2011: Apache Hadoop release 1.0.0 available
- 2011: Hortonworks, another major Hadoop distributor founded
- Oct 2013: Apache Hadoop release 2.2.0 (YARN)
- Dec 2015: Apache Hadoop release 2.6.3 available
- Feb 2016: Apache Hadoop release 2.6.4 available

- Source:
  - http://hadoop.apache.org/#News
  - https://en.wikipedia.org/wiki/Apache_Hadoop#Papers

Distributed File System (DFS)

Read 1 TB Data

1 Machine

50 Minutes

10 Machines

5 Minutes

What is Hadoop

- Hadoop is not a :
  - Database
  - Big Data
  - Networking Concept
  - Data warehouse
  - Programming Language

Then What Hadoop is.....??

What is Hadoop(continued)

- **Hadoop** is a framework that allows distributed processing of large data sets across clusters of commodity computers using simple programming models.

  Definition In Depth:

  - Distributed Processing :
    - Data is processed in multiple machines in a distributed manner
  - Large Data sets:
    - Large data sets in this context means files that are hundreds of megabytes, gigabytes, or terabytes in size
  - Clusters of commodity computers :
    - Cheap hardware (not expensive servers) are used to create a cluster
  - Simple Programming Model:
    - Map Reduce/Spark is used as a programming model to manipulate/process the data

Characteristics of Hadoop

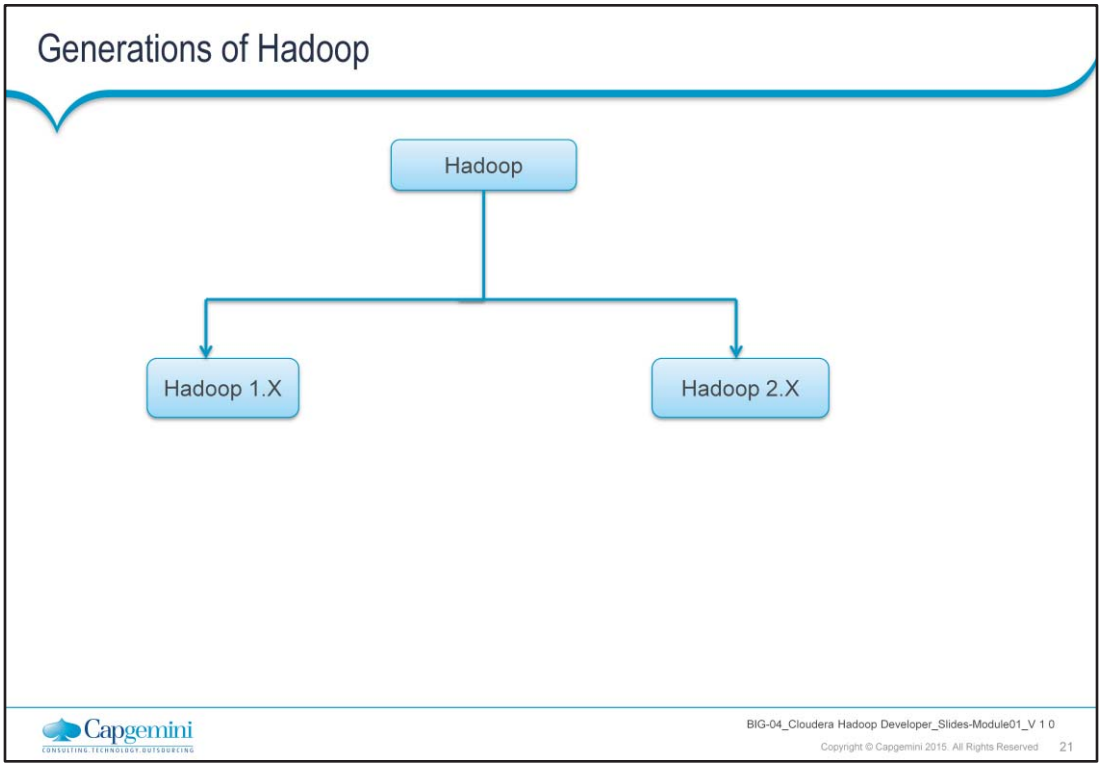
- **Scalable:** It can reliably store and process petabytes of data and can be scaled up anytime whenever required without any adverse impact of cluster.
- **Economical:** It distributes the data and processing across clusters of commonly available computers (in thousands).
- **Efficient:** By distributing the data, it can process it in parallel on the nodes where the data is located.
- **Reliable:** It automatically maintains multiple copies of data and automatically redeploys computing tasks based on failures.

These are the features making Hadoop among different organizations in various sectors.

Relational DB vs. Hadoop

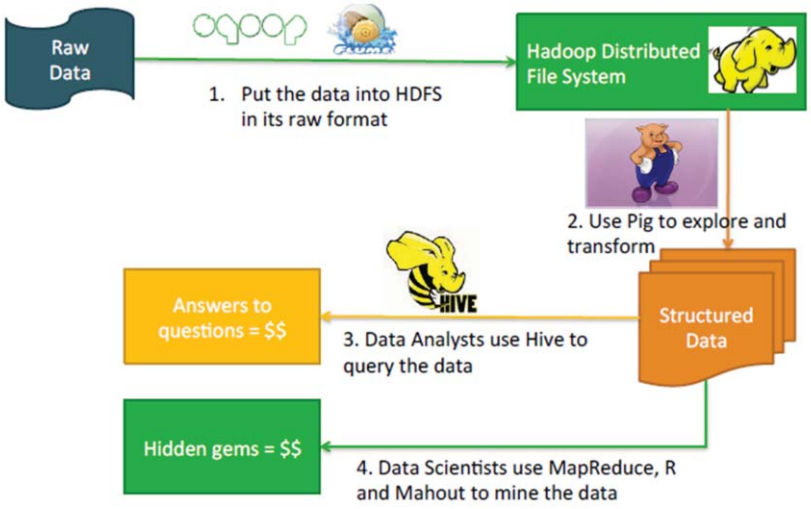
- Expensive dedicated HW
- Built for **performance**
- Designed for **high volumes** (eg 10s of TB)
- High availability
- Initially developed using Relational Data bases
- Supports only **modelled and structured data**
- Business As Usual ways to design, build and deliver
- Very mature solutions (skills, SW, HW, administration)
- Teradata, Oracle Exadata, IBM Netezza, ...

- Uses commodity PCs
- Built for **extreme scalability**
- Designed for **extreme volumes** (10s of PB and more)
- Very high availability (clouds like Amazon distributed all around the world)
- Initially developed by Google for storing Petabytes of web pages for ranking
- Not yet fully mature
- Hadoop = Data is distributed over many machines
- MapReduce = Computing is distributed and executed where data is (grid solution)
- Works on **Write Once read many times** approach



Hadoop 1.x is tested up to 4000 nodes cluster and Hadoop 2.x beyond that.

Usage of Hadoop and its eco-systems



Components of Hadoop

Hadoop is a platform for data storage and processing that is…

- ✓ Scalable
- ✓ Fault tolerant
- ✓ Open source

CORE HADOOP COMPONENTS

Hadoop Distributed File System (HDFS)

File Sharing & Data Protection Across Physical Servers

+

MapReduce

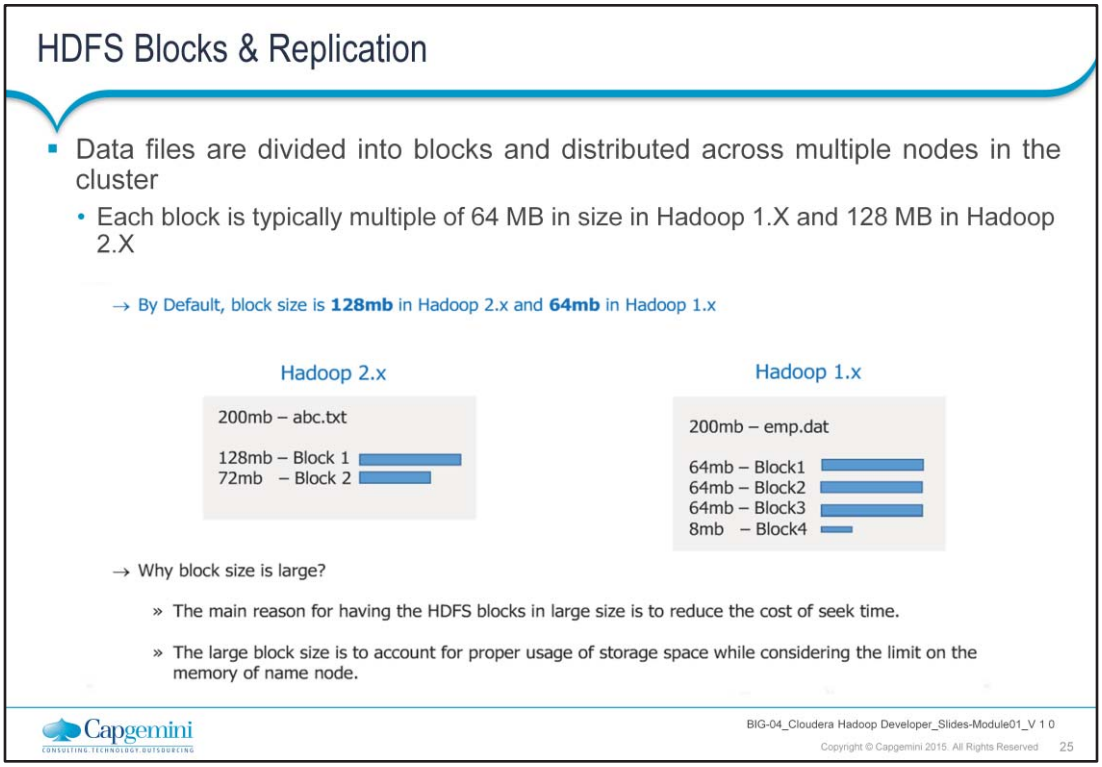
Distributed Computing Across Physical Servers

Components of Hadoop(continued)

- Two core components of Hadoop :
  1. Hadoop Distributed File System (HDFS)
  2. Map Reduce

- HDFS :
  - Is the storage part of Hadoop framework
  - Data is stored in files under directories
  - Files are always divided into blocks

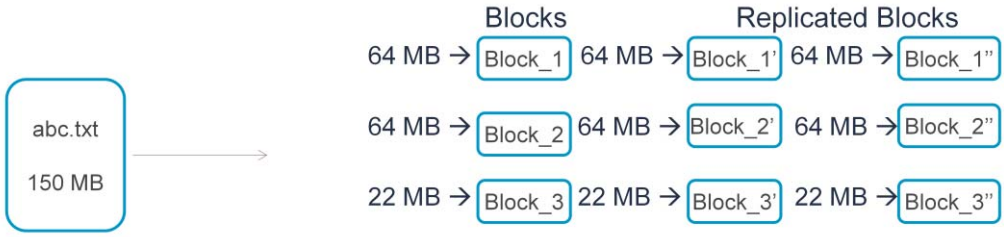
- Map Reduce :
  - Is the programming/processing model of Hadoop framework
  - Is responsible for manipulation or processing on data
  - Preferebly written as java programs



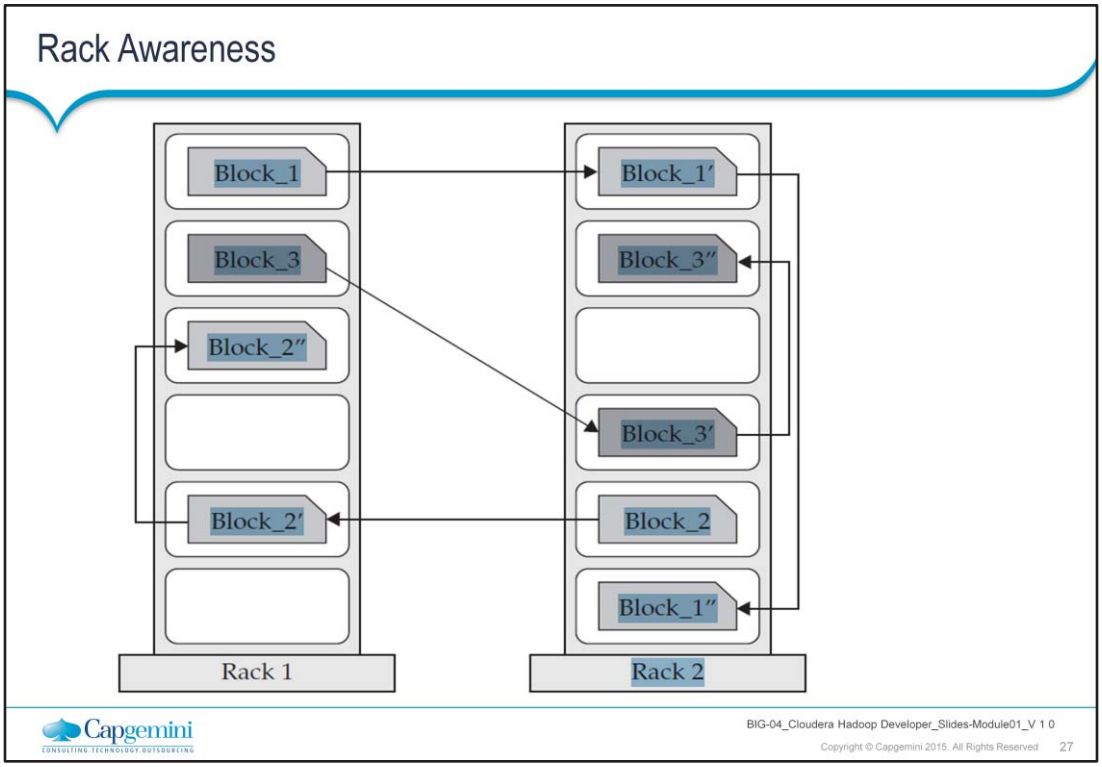
Earlier too, data was stored in distributed way in multiple servers/machines. But These machines were high configuration servers/machines. And no replication on any data was performed. Systems may have different topologies like star, mesh etc.
Eg: If a file of 100 MB is stored in a distributed manner in 5 machines i.e. each machine has 20 MB data. If any of the machine fails, there was no way to recover that. It has been overcome in Hadoop by replicating the data blocks in different machine and hence hadoop has better fault tolerance mechanism.

HDFS Blocks & Replication(continued)

- Each block is replicated multiple times
  - Default replication factor is 3 (configurable)
  - Replicas are stored on different nodes
  - This ensures both reliability and availability



Considered Hadoop 1.X in above example.



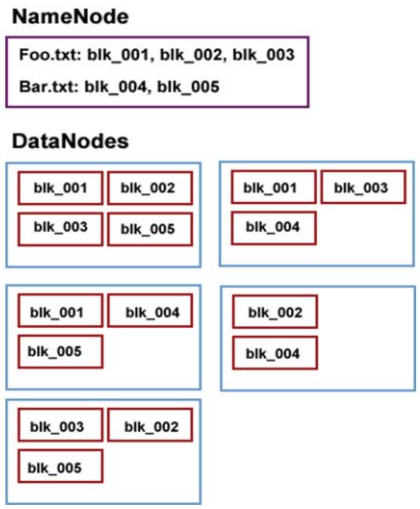
Block_1 will be stored in a data node of rack 1 and its replicated blocks Block_1’ and Block_1’’ will be in rack 2. This is done to overcome rack failure.
It will never happen that Block_1 and its any replicated blocks are stored in same rack.

How Files Are Stored

- Files are split into blocks
- Data is distributed across many machines at load time
  - Different blocks from the same file will be stored on different machines
  - This provides for efficient MapReduce processing.
- Blocks are replicated across multiple machines, known as Data Nodes
  - Default replication is three-fold meaning that each block exists on three different machines
- A master node called the Name Node keeps track of which blocks make up a file, and where those blocks are located
  - Known as the metadata

How Files Are Stored: Example

- Name Node holds metadata for the two files (Foo.txt and Bar.txt)
- Data Nodes hold the actual blocks
  - Each block will be 64 MB or 128 MB in size
  - Each block is replicated three times on the cluster



Fault Tolerance in Hadoop

- If a data node fails, the master(Name Node) will detect that failure and re-assign the task to a different node on the cluster
- Restarting a task does not require communication with nodes working on other portions of the data
- If a failed node recovers, it is automatically added back to the cluster and assigned new tasks
- If a node appears to be running slowly, the master can redundantly execute another instance of the same task
  - Results from the first to finish will be used
  - Known as 'speculative execution'

HDFS Commands

- HDFS Commands are used to
  - access Hadoop Distributed File System (HDFS)
  - list down files and directories present in HDFS
  - create and remove directory in HDFS
  - push the data present in landing zone(local machine) to HDFS
  - copy the data back to landing zone(local machine) from HDFS

HDFS Commands(continued)

- Get a directory listing of the HDFS root directory

```
hadoop fs –ls /
```

- Create a directory called input under the user directory

```
hadoop fs –mkdir /user/input
```

- Copy file foo.txt from local disk to the input directory in HDFS

```
hadoop fs -copyFromLocal foo.txt /user/input/foo.txt
```

HDFS Commands(continued)

- Display the contents of the HDFS file **/user/input/*foo.txt***

```
hadoop fs –cat /user/input/foo.txt
```

- Copy the file foo.txt from HDFS to local

```
hadoop fs –copyToLocal /user/input/foo.txt  foo.txt
```

- Delete the file bar.txt in HDFS

```
hadoop fs –rm /user/input/bar.txt
```

HDFS Commands(continued)

- Delete the input directory in HDFS

```
hadoop fs –rm [–r ] /user/input/
```

→ -r used for recursive, deletes directory even if some file is present under that.

For more commands please refer:

http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html

Hadoop Daemons

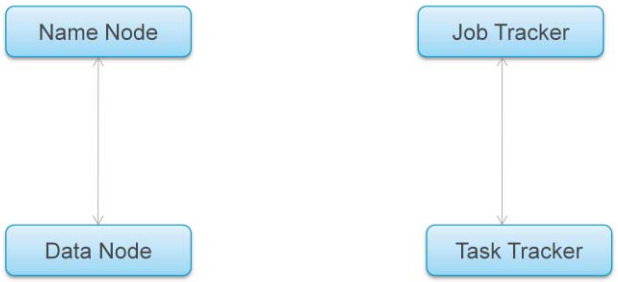
- Hadoop Works on Master-Slave pattern
- There are 5 daemons in Hadoop 1.x

1. Name Node → Master Node
2. Secondary Name Node → Master Node
3. Job Tracker → Master Node
4. Task Tracker → Slave Node
5. Data Node → Slave Node

Hadoop Daemons(continued)

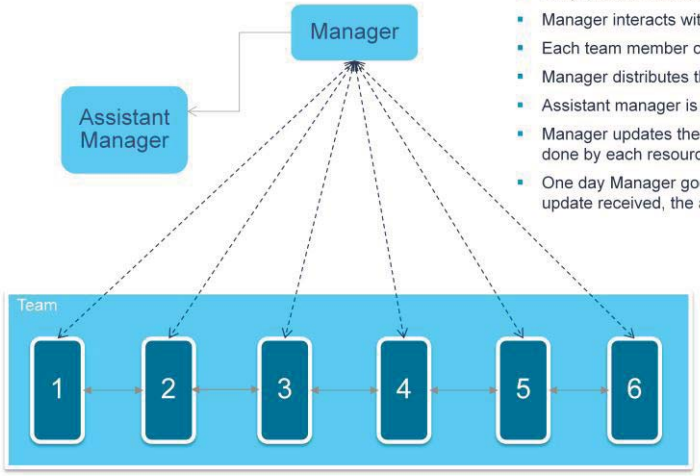
- **Name Node** stores metadata of cluster
  - Block details
  - Replication details
  - Load Balancing
  - High configuration machine, 1 Name Node per cluster
- **Secondary Name Node** is the backup of Name Node
  - Takes backup of Name Node in a regular interval
  - Not called as complete backup of Name Node
- **Job Tracker** is the daemon where hadoop jobs are submitted
- **Task Tracker** accomplish the task which are submitted to job tracker
- **Data Node** is the daemon where actual data in the form of blocks reside

Hadoop Daemons(continued)



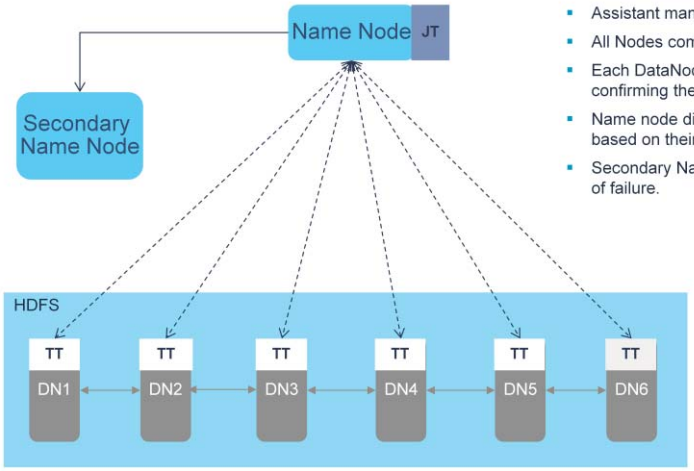
- Name Node coordinates with Data Nodes
  - In turn, all Data Nodes send heart beats to Name Node periodically
- Job Tracker coordinates with Task Tracker

Hadoop Daemons– Corporate Analogy



- Manager is appointed who will govern the project.
- Manager recruits 6 Members to Join the project team.
- They all communicate with each other.
- Manager interacts with each of the 6 Team members.
- Each team member confirms his presence to the manger everyday
- Manager distributes the work volume based on Team's utilization.
- Assistant manager is provisioned as a backup manager.
- Manager updates the Assistant manager every day about activities done by each resource.
- One day Manager goes on unplanned leave and based on the last update received, the asst. manager takes over the activities.

Hadoop Daemons(continued)



- Manager : Name node
- 6 Team Members : 6 data Nodes.
- Assistant manager : Secondary Name Node
- All Nodes communicate via SSH password less protocol.
- Each DataNode sends heartbeat signals to the Name node confirming they are live.
- Name node distributes the data Blocks across Datanodes based on their memory utilization and alive status.
- Secondary NameNode acts as a failover to avoid single point of failure.

1. When a MR program is submitted it goes to the Job Tracker(JT)
2. JT talks with NameNode to understand which Data Nodes have the required data blocks of the source files.
3. Once identified, JT talks to the TTs on those Nodes and use them to perform the assigned Tasks.
4. On task completion, JT updates the NameNode on the new Block location where the output is stored.


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
People matter, results count.
Learning & Development
Enabling development, Impacting growth…
BIG-04
MAP REDUCE
INSIGHTS&DATA

What is Map Reduce

- Programming model for data processing for clusters of commodity machines

- Pioneered by Google
  - Processes 20 PB of data per day
- Popularized by open-source Hadoop project
  - Used by Yahoo, Facebook, Amazon etc
- Simple Implementation model
  - Implementation can be done in Java, ruby, python, C++ etc.
  - No need to worry about shipment, rack, distribution, JVM etc.

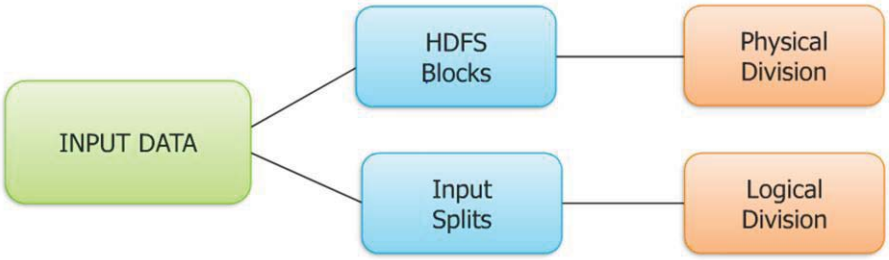
Map Reduce(continued)

- "Map" step:
  - Is just a data preparation step
  - Program split into pieces
  - Worker(Slave) nodes process individual pieces in parallel (under global control of the Job Tracker node)
  - Each worker node stores its result in its local file system where a reducer is able to access it
- "Reduce" step:
  - Data is aggregated (‘reduced” from the map steps) by worker nodes (under control of the Job Tracker)
  - Multiple reduce tasks can parallelize the aggregation
  - Aggregation logic is applied in this phase

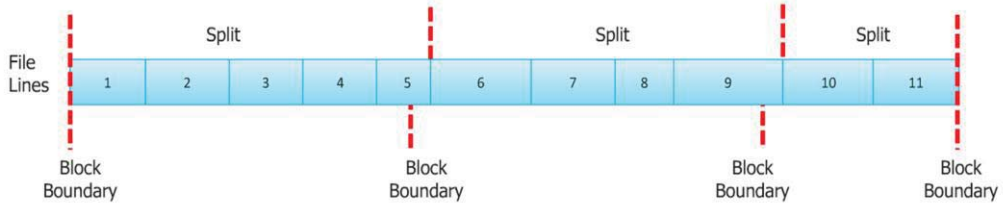
Map Reduce(continued)

- Input to Mapper (map phase) is always an input split.
- An input split is a logical division of data.
- Input and output of map reduce is always a Key-Value pair.
- Mapper is just a data preparation phase.
- Actual aggregation logic is applied in Reducer phase.
- Merge and sort is done by framework after map phase.

Input Splits



Relation between blocks and input splits



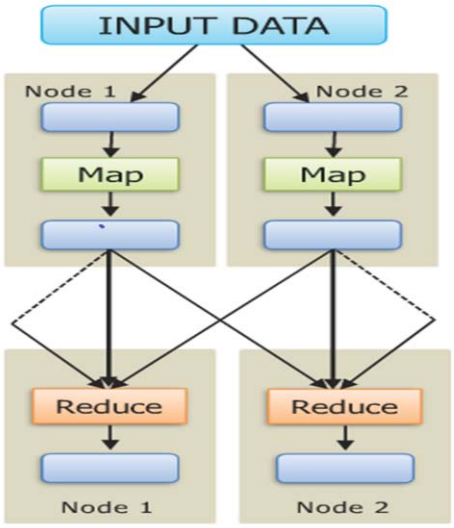
→ Logical records do not fit neatly into the HDFS blocks.

→ Logical records are lines that cross the boundary of the blocks.

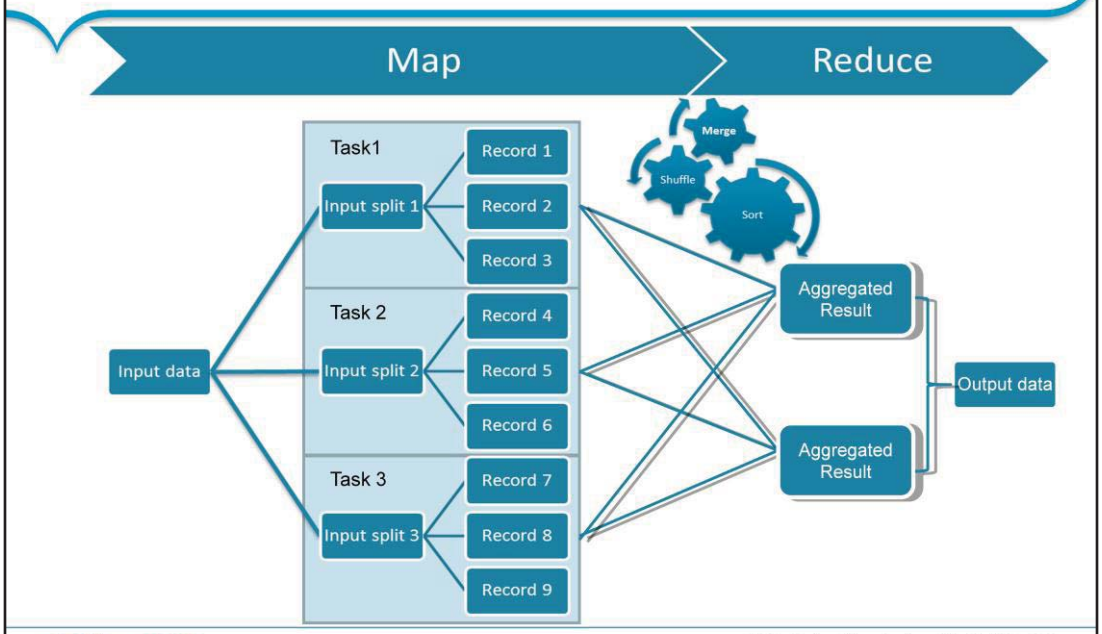
→ First split contains line 5 although it spans across blocks.

Map Reduce Job Submission

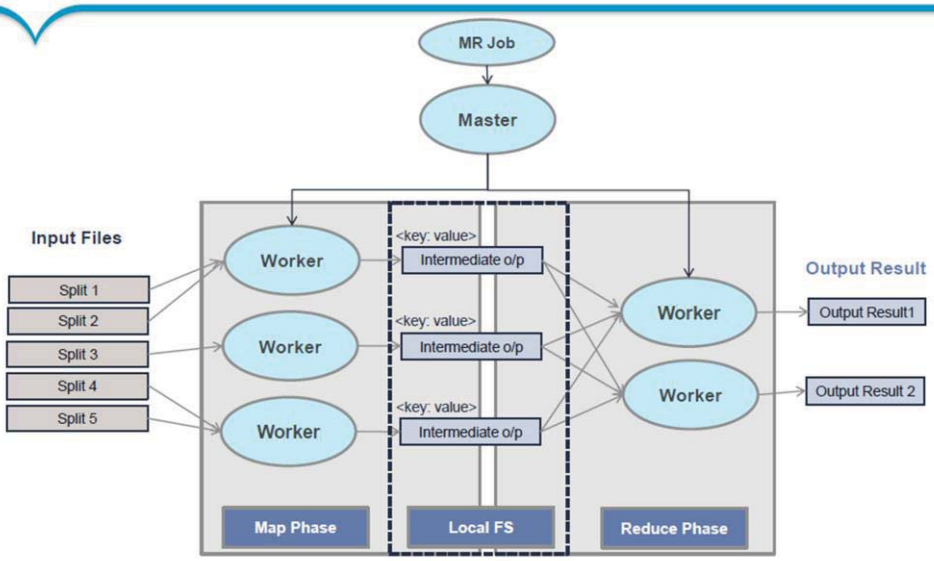
- Each Map task works on a split data
- Mapper outputs intermediate data
- Sort and merge is performed based on key
- Reducer output is stored in different nodes



Map Reduce(continued)



Map Reduce(continued)



Map Reduce Functions

- Map Function:

- Reduce Function:

Map Reduce in Hadoop-1

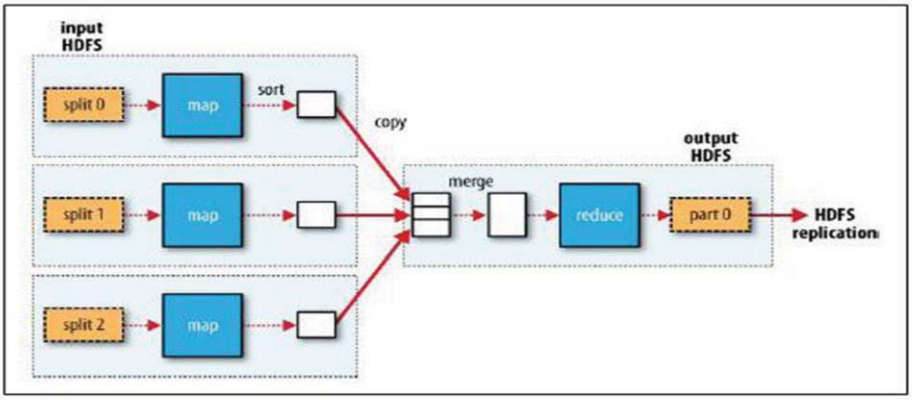
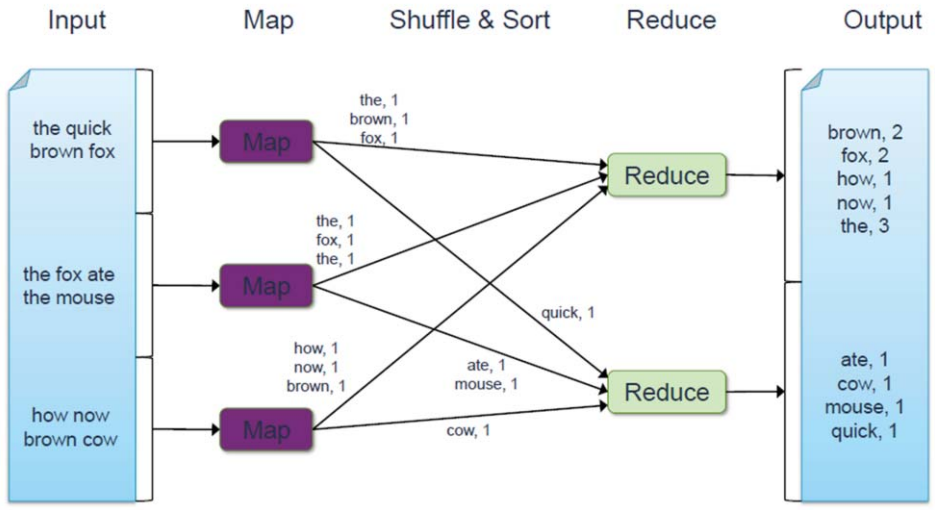


Figure 2-2. MapReduce data flow with a single reduce task

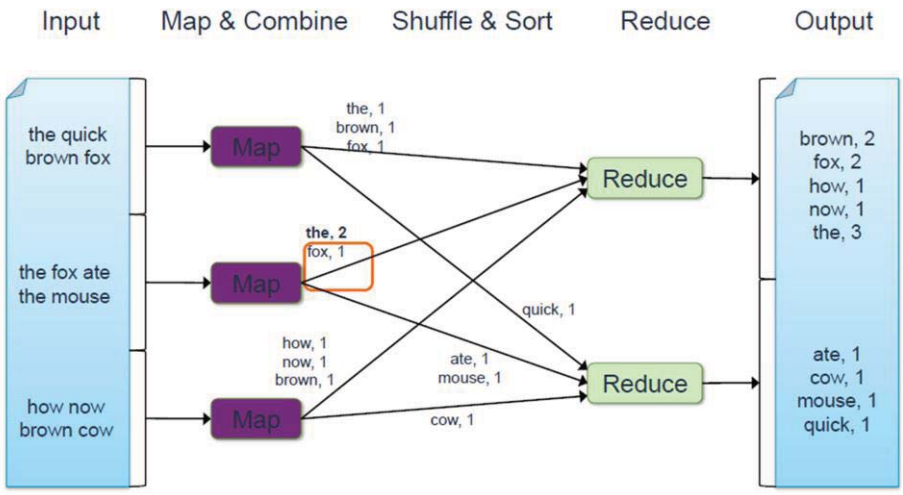
Word Count Execution



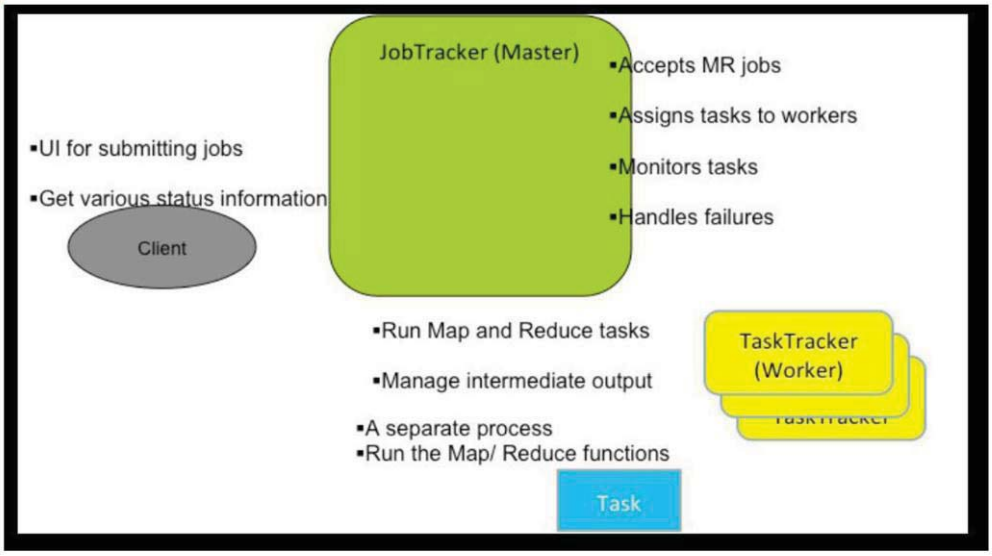
An Optimization: The Combiner

- A combiner is a local aggregation function for repeated keys produced by same map
- For associative ops. like sum, count, max
- Decreases size of intermediate data

Word Count with Combiner



Components in a Hadoop MR Workflow



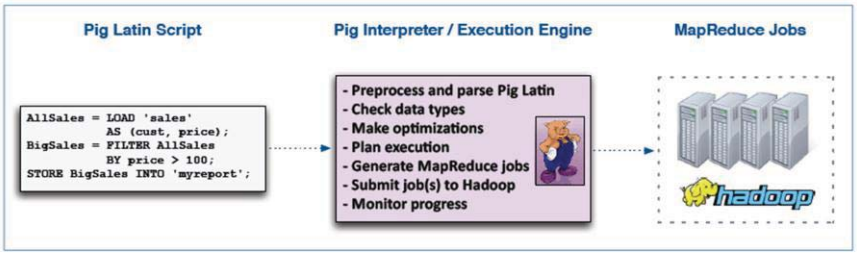
How are Organizations using Pig?

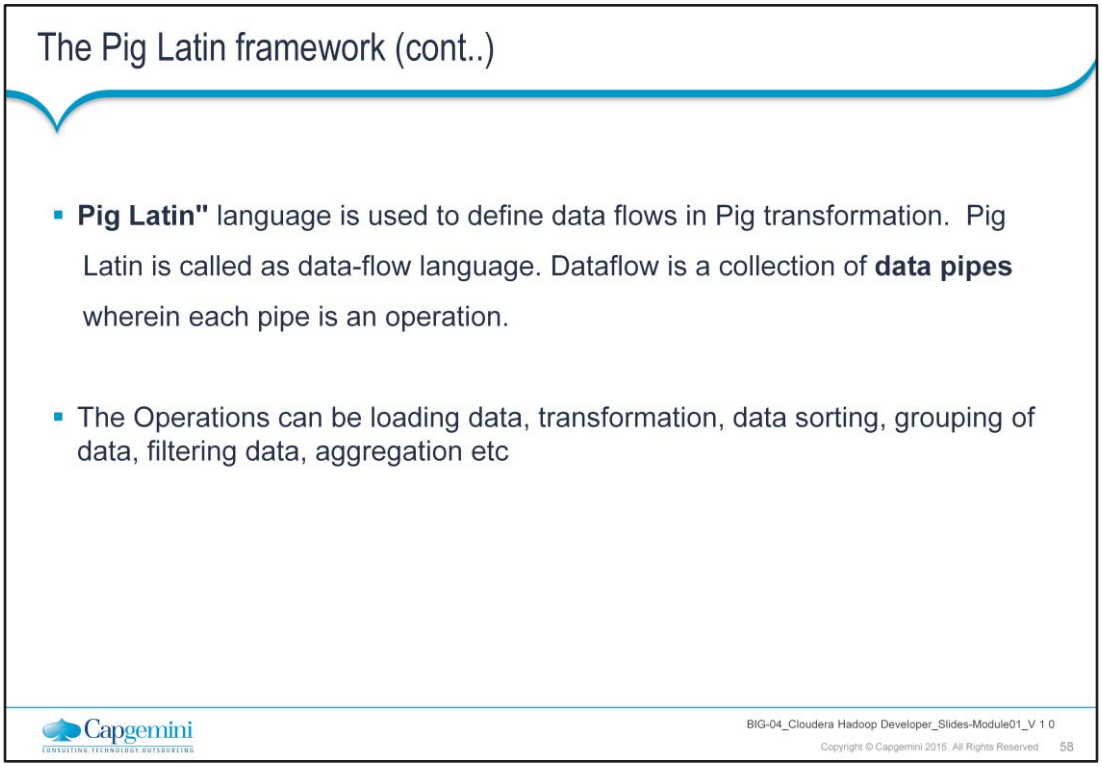
- Usage of Pig
  - Many Organization use Pig for data analyses to
  - Find relevant records in a massive data
  - Querying multiple data sets
  - Calculating values from input data
  - Pig also frequently used for data processing that includes
  - Reorganizing an existing data set
  - Joining dataset from multiple sources to produce a new data set.
- However, Organization uses 80% of Pig features for data processing purpose


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
People matter, results count.
Learning & Development
Enabling development, Impacting growth…
BIG-04
PIG
INSIGHTS&DATA

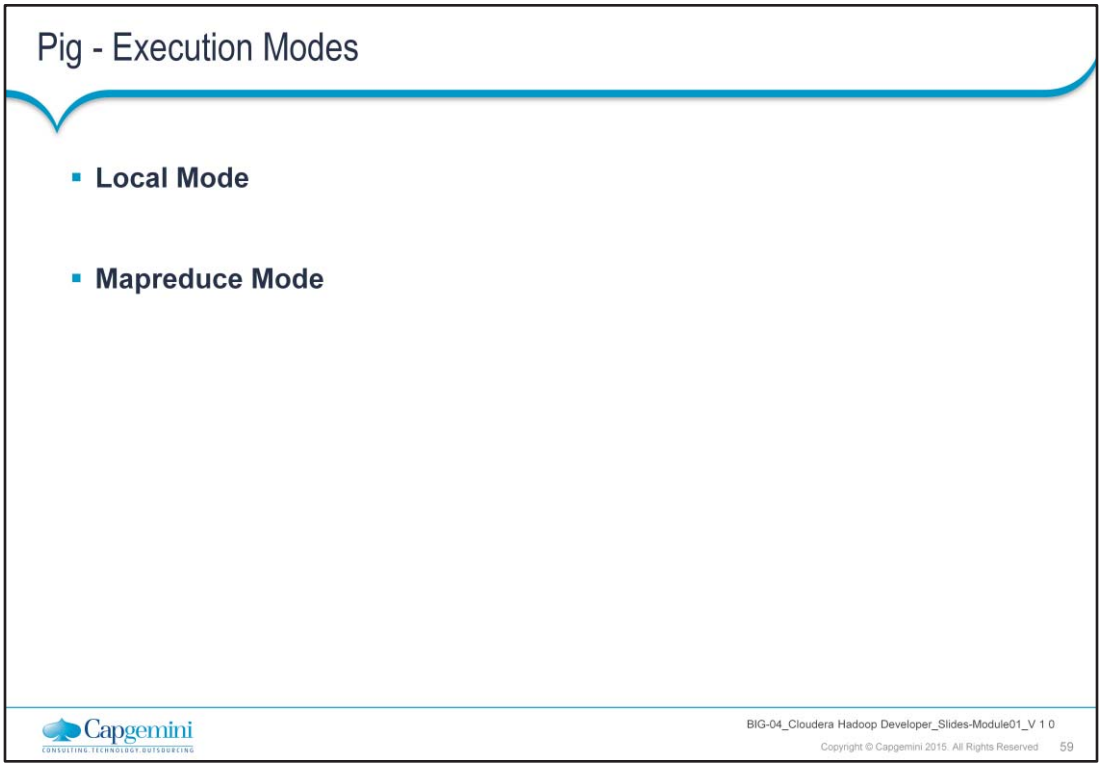
The Pig Latin framework

- Main components of Pig
  - The data flow language (Pig Latin)
  - The Interactive shell where you can type Pig Latin Statements (Grunt shell)





Java/C/C++ etc are **control flow structured language** where as Pig Latin is **data flow language**. In Pig Latin, without complex programming control structures, you can define complex data flows. Hence, from one pipe to another pipe, data transforms from one state to other state.



Local Mode - To run Pig in local mode, you need access to a single machine; all files are installed and run using your local host and file system. Specify local mode using the -x flag (pig -x local).
Mapreduce Mode - To run Pig in mapreduce mode, you need access to a Hadoop cluster and HDFS installation. Mapreduce mode is the default mode; you can, *but don't need to*, specify it using the -x flag (pig OR pig -x mapreduce).

Pig - Execution Modes (cont..)

- **Examples**
- This example shows how to run Pig in local and mapreduce mode using the pig command.
  - **local mode**

    $ pig -x local ...

  - **mapreduce mode**

    $ pig ... or
    $ pig -x mapreduce

Pig Interacting with HDFS

```
grunt> fs -mkdir sales/;
grunt> fs -put europe.txt sales/;
grunt> fs -get europt.txt ;
```

Running Pig in Interactive Mode

- You can run Pig in interactive mode using the Grunt shell.
- Invoke the Grunt shell using the "pig" command (as shown below) and then enter your Pig Latin statements and Pig commands interactively at the command line.
- The DUMP operator will display the results to your terminal screen.

```
grunt> A = load 'passwd' using PigStorage(':');

grunt> B = foreach A generate $0 as id;

grunt> dump B;
```

Pig Scripts

- Use Pig scripts to place Pig Latin statements and Pig commands in a single file
- It is good practice to create the file using the .pig extension, however it is not mandatory.

Comments in Script:

- You can include comments in Pig scripts:
- For multi-line comments use

/*myscript.pig My script is simple. It includes three Pig Latin statements .... */

- For single-line comments use

--

Eg: A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int, gpa:float); -- *loading data*

Data Types (cont..)

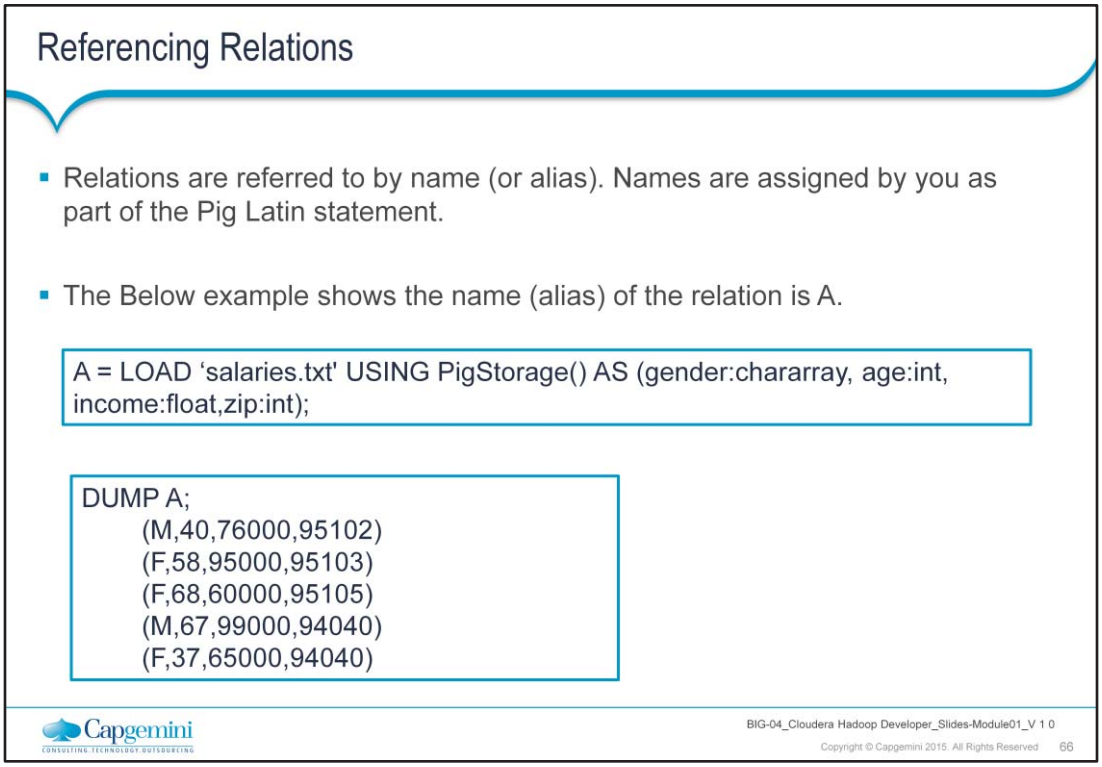
- fields default to type bytearray
- Implicit conversions are applied to the data depending on the context in which t
  - For example, in relation B, f1 is converted to integer because 5 is integer.
  - In relation C, f1 and f2 are converted to double because we don't know the type of either f1 or f2.hat data is used

```
A = LOAD 'salaries.txt ' AS (gender,age,income);

B = FOREACH A GENERATE age + 5;

C = FOREACH A generate  age + income;
```

Data Types (cont..)

- If a schema is defined as part of a load statement, the load function will attempt to enforce the schema.
- If the data does not conform to the schema, the loader will generate a null value or an error.

  A = LOAD 'salaries.txt' AS (gender:chararray, age:int, income:float,zip:int);



Pig Latin Relation Names

Each processing step of a Pig Latin script results in a new data set, referred to as a ***relation***. You assign names to relations, and the name of a relation is referred to as its ***alias***. For example, consider the following Pig Latin statement:

```
stocks = LOAD 'mydata.txt' using TextLoader();
```

The alias **stocks** is assigned to the relation created by the **LOAD** statement, which in this statement is a line of text from the **mydata.txt** file. The **stocks** alias now represents the collection of records in **mydata.txt**.

NOTE: **TextLoader** is a simple way of loading each line of text in a file into a record, no matter what the format of the data is.

Relation names (aliases) are not variables, even though they look like variables. You can reassign an alias to a different relation, but that is not recommended.

Both field names and relation names must satisfy the following naming criteria:

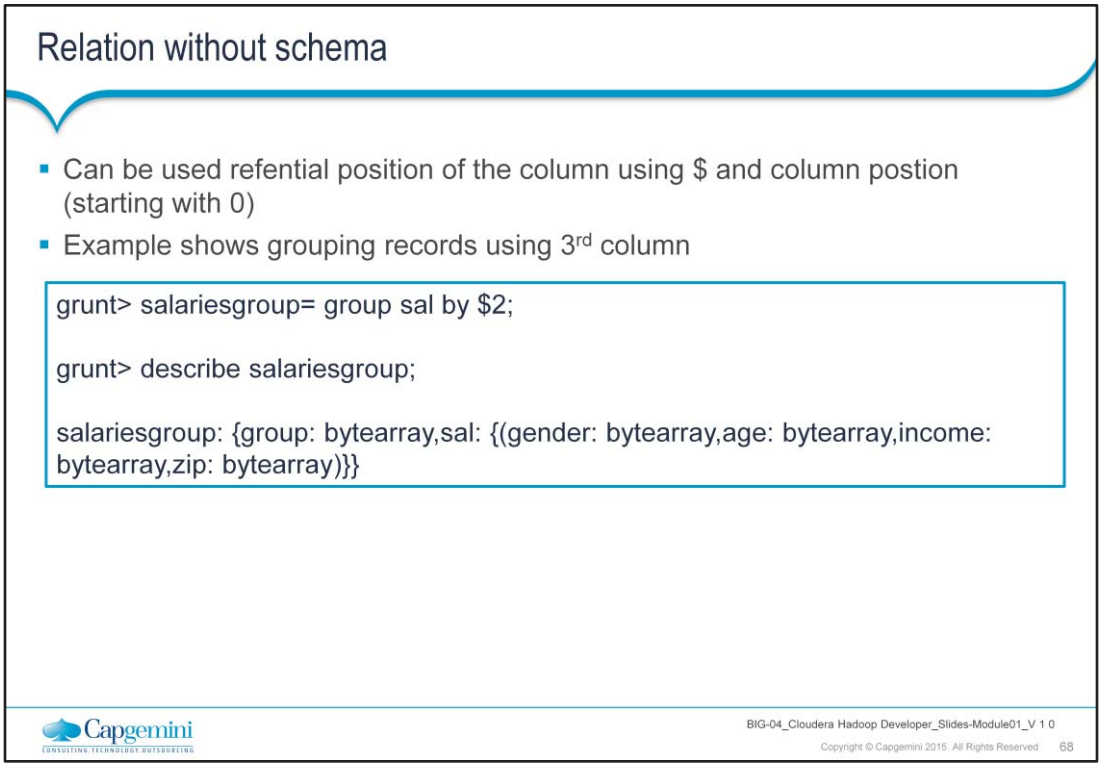
- Start with an alphabetic character
- Can contain alphabetic and numeric characters, as well as the underscore (_) character
- All characters must be ASCII

IMPORTANT: Field names and relation names are case sensitive in your Pig Latin scripts.
User Defined Functions (UDFs) are also case sensitive.
However, Pig Latin keywords (like LOAD and AS) are not case sensitive.

Referencing Fields

- Fields are referred to by positional notation or by name (alias).
- Positional notation is generated by the system. Positional notation is indicated with the dollar sign ($) and begins with zero (0); for example, $0, $1, $2.
- Names are assigned by you using schemas
  - for example, gender, age,income or $1,$3 etc

```
grunt> sal= load 'salaries.txt' USING PigStorage(',') as (gender,age,income,zip);

grunt>X= foreach sal generate age,$2;
grunt>DUMP X;

(44,96000)
(73,12000)
(55,32000)
(82,10000)
```



Relations without a Schema

If you do not define a schema, then the fields of a relation are specified using an index that starts at **$0**. This works well for datasets that have a lot of columns or for data that is not structured.

The following relation has four columns but does not define a schema:
salaries = LOAD 'salaries.txt' USING PigStorage(',');

Notice what the output is when you try to describe this relation:

> DESCRIBE salaries;
Schema for salaries unknown.
The following relation groups **salaries** by its fourth field:
salariesgroup = GROUP salaries BY $3;

Notice the **salariesgroup** relation does not have a schema for its **salaries** field:
> describe salariesgroup
salariesgroup: {group: bytearray,salaries: {()}}

Viewing Schema definition using DESCRIBE

- The DESCRIBE Command shows the structure of the data including names and types
- The following grunt session shows an example

```
grunt> sal =  load 'salaries.txt AS (gender,age,income , zip);

grunt> DESCRIBE sal

sal: {gender: bytearray,age: bytearray,income: bytearray,zip:
bytearray}
```

Loading Data to Pig

- Use the LOAD operator to load data from the file system to **relation**.

  Syntax : *LOAD 'data' [USING function] [AS schema];*

- Pig's Default loading function is called PigStorage
  - The name of the function is implicit when calling LOAD
  - PigStorage assumes text format with tab separated
  - Consider the following file in HDFS called **Salary.txt**

Loading Data to Pig (cont..)

- Example to load data
  - The two fields are separated by tab-characters

```
M   44      96000   94040
F   73      12000   95102
M   55      32000   94040
F   82      10000   95102
```

```
grunt> sal=LOAD 'salaries.txt' As (gender,age)
```

- The above Command loads file from the default HDFS directory

Viewing of Output using DUMP

- The Command used to handle output depends on its destination
  - DUMP: Sends output to the screen
  - STORE: Sends output to disk (HDFS)

- Example of DUMP

```
grunt> dump sal;
```

Storing Data with Pig - STORE

- The STORE commands is used to store data to HDFS
  - Similar to LOAD, but writes data instead of reading it
  - The directory must be present in the HDFS

  Syntax

  STORE alias INTO 'directory' [USING function];

- As with LOAD the use of PigStorage is implicit
  - The field delimiter also has default value (tab)

    STORE bigsales INTO 'myreport' ---> writes in default HDFS directory

- You may specify an alternate delimeter using PigStorage

  STORE bigsales INTO 'myreport' USING PigStorage('|');

Sorting Records and Limiting Output - Order By & Limit

- Use ORDER .....BY to sort the records in a bag in ascending order
  - Add DESC to sort in descending order instead
  - Take care to specify a schema - data type affects how data is sorted!

Syntax

alias = ORDER alias BY { * [ASC|DESC] | field_alias [ASC|DESC] [, field_alias [ASC|DESC] …] } [PARALLEL n];

```
grunt> sortedsal=  order sal by income desc;
```

```
grunt>dump sortedsal;

(M,67,99000,94040)
(M,44,96000,94040)
(F,58,95000,95103)
(M,31,95000,94041)
(M,48,91000,95102)
```


The ORDER BYOperator

The **ORDER BY** command allows you to sort the data in arelation:

```
salaries = LOAD 'salaries.txt' USING PigStorage(',')  AS
                (gender:chararray,age:int,salary:double,zip:chararray);  byage = ORDER salaries BY age  ASC;
```

The records in the **byage** relation will be sorted byage:

```
(M,19,0.0,95050)
(F,22,90000.0,95102)
(M,23,89000.0,95105)
(M,23,64000.0,94041)
(F,30,10000.0,95101)
(M,31,95000.0,94041)
```

You can use **DESC** or **ASC** in the **BY** clause. You can also order by multiple fields.For example:

```
agesalary = ORDER salaries BY age ASC, salary  ASC;
```

The output is similar to **byage**, except the **salary** field is sorted ascending. Comparethe two outputs of the records with age =23:

```
(M,19,0.0,95050)
(F,22,90000.0,95102)  (M,23,64000.0,94041)  (M,23,89000.0,95105)  (F,30,10000.0,95101)  (M,31,95000.0,94041)
```

NOTE: The resulting output of an **ORDER BY** relation is a *total ordering*, which means the data will be sorted across all output files. In other words, p[redacted]will contain the first set of ordered tuples, then p[redacted]will continue where the first records left off, and soon.

IMPORTANT: If you define a relation with an ordering, then process that relation in another expression, then the ordering is no longer guaranteed. Forexample:

```
A = ORDER visitors BY lastname DESC;  B = FILTER A BY age >=21;
```

The records in **B** are no longer guaranteed to be ordered by **lastname** descending.

Using Alternative Column Delimiters

- You can specify an alternate delimeter as an argument to PigStorage
- This example shows how to load comma-delimited data

```
grunt>sal= LOAD 'salaries.csv' USING PigStorage(',') AS (gender,age,income,
zip)
```

- To load (|) delimited file without specifying column name

```
grunt>sal= LOAD 'salaries.txt' USING PigStorage('|');
```

Arithmetic Comparison Operators - Examples

- Example the modulo operator is used with fields f1 and f2.

```
grunt >X = FOREACH A GENERATE f1, f2, f1%f2;

grunt> DUMP X;
   (10,1,0)
   (10,3,1)
   (10,6,4)
```

- Example: numeric

```
grunt > X = FILTER A BY (f1 == 8);
```

- Example: string

```
Grunt > X = FILTER A BY (f2 == 'apache');
```

- Example: matches

```
Grunt > X = FILTER A BY (f1 matches '.*apache.*');
```

Expressions

- Expressions are language constructs used with the FILTER, FOREACH, GROUP, and SPLIT operators as well as the eval functions.

- Example of an arithmetic expression:

```
X = GROUP A BY f2*f3;
```

- An Example of a string expression: *Note a and b are both chararrays:*

```
X = FOREACH A GENERATE CONCAT(a,b);
```

- Example of A boolean expression:

```
X = FILTER A BY (f1==8) OR (NOT (f2+f3 > f1));
```

Group Operator

Use GROUP BY to do this in Pig Latin

- The new relation has one record per unique group

```
grunt> salariesbyage = group sal by age;
```

```
grunt> dump salariesbyage;

(58,{(F,58,95000,95103)})
(65,{(F,65,70000,95102)})
(66,{(F,66,41000,95103),(M,66,84000,95103)})
(67,{(M,67,99000,94040),(M,67,81000,95101)})
(68,{(F,68,60000,95105),(M,68,15000,95103)})
(71,{(M,71,0,94041)})
```


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
People matter, results count.
Learning & Development
Enabling development, Impacting growth…
BIG-04
HIVE
INSIGHTS&DATA

Motivation to Hive

- MapReduce code is typically written in Java
  - Although it can be written in other languages using Hadoop Streaming API

- Requires:
  - A programmer
  - Who is a good Java programmer
  - Who understands how to think in terms of MapReduce
  - Who understands the problem they're trying to solve
  - Who has enough time to write and test the code
  - Who will be available to maintain and update the code in the future as requirements change

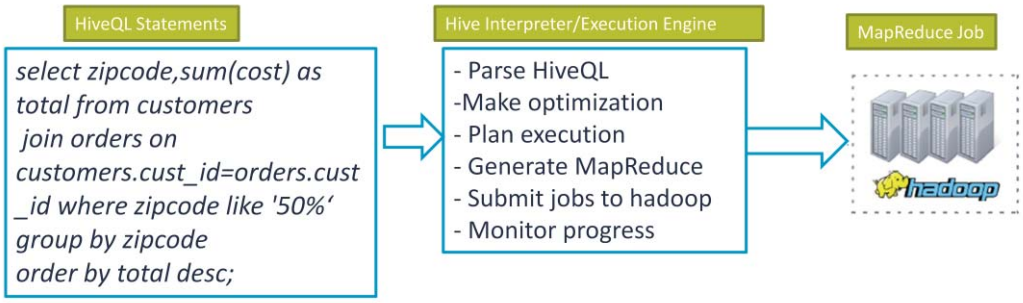
What is Hive?

- A data warehousing system to store structured data on Hadoop file system
- Provide an easy query these data by execution Hadoop MapReduce plans

- **Intuitive**
  - Make the unstructured data looks like tables regardless how it really lay out
  - SQL based query can be directly against these tables
  - Generate specify execution plan for this query

What is Hive? (cont..)

- Hive runs on the client machine
  - Turns HiveQL queries into MapReduce jobs
  - Submit those jobs to the cluster



Why do we use Apache Hive?

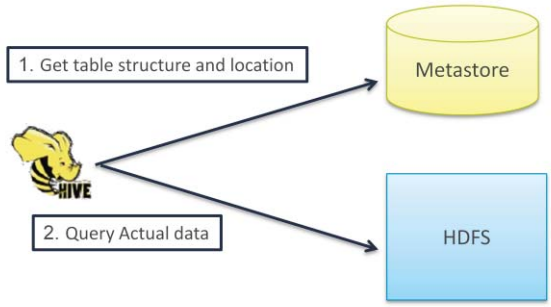
- More productive than writing MR jobs
  - Five lines of HiveQL might be equivalent to 100 lines or more of Java
- Brings large-scale data analysis to a broader audience
  - No Software development experience required
  - Leverage existing knowledge of SQL
- Offers interpretability with other systems
  - Extensible through java and external scripts
  - Many BI tools supports hive

Hive Vs RDBMS

Hive	RDBMS
SQL Interface.	SQL Interface.
Focus on analytics.	May focus on online or analytics.
No transactions.	Transactions usually supported.
Partition adds, no random INSERTs. In-Place updates not natively supported (but are possible).	Random INSERT and UPDATE supported.
Distributed processing via map/reduce.	Distributed processing varies by vendor (if available).
Scales to hundreds of nodes.	Seldom scale beyond 20 nodes.
Built for commodity hardware.	Often built on proprietary hardware (especially when scaling out).
Low cost per petabyte.	What's a petabyte?

Hive Data Model

- How Hive Stores loads data and store?
- Hive consults the metastore to determine data format and location
- The query itself operates on data stored on a filesystem (typically HDFS



Hive Data Types

Hive SQL Datatypes	Hive SQL Semantics
INT	SELECT, LOAD, INSERT from query
TINYINT/SMALLINT/BIGINT	Expressions in WHERE and HAVING
BOOLEAN	GROUP BY, ORDER BY, SORT BY
FLOAT	Sub-queries in FROM clause
DOUBLE	GROUP BY, ORDER BY
STRING	CLUSTER BY, DISTRIBUTE BY
TIMESTAMP	ROLLUP and CUBE
BINARY	UNION
ARRAY, MAP, STRUCT, UNION	LEFT, RIGHT and FULL INNER/OUTER JOIN
DECIMAL	CROSS JOIN, LEFT SEMI JOIN
CHAR	Windowing functions (OVER, RANK, etc.)
VARCHAR	INTERSECT, EXCEPT, UNION DISTINCT
DATE	Sub-queries in WHERE (IN/NOT IN, EXISTS/NOT EXISTS
	Sub-queries in HAVING

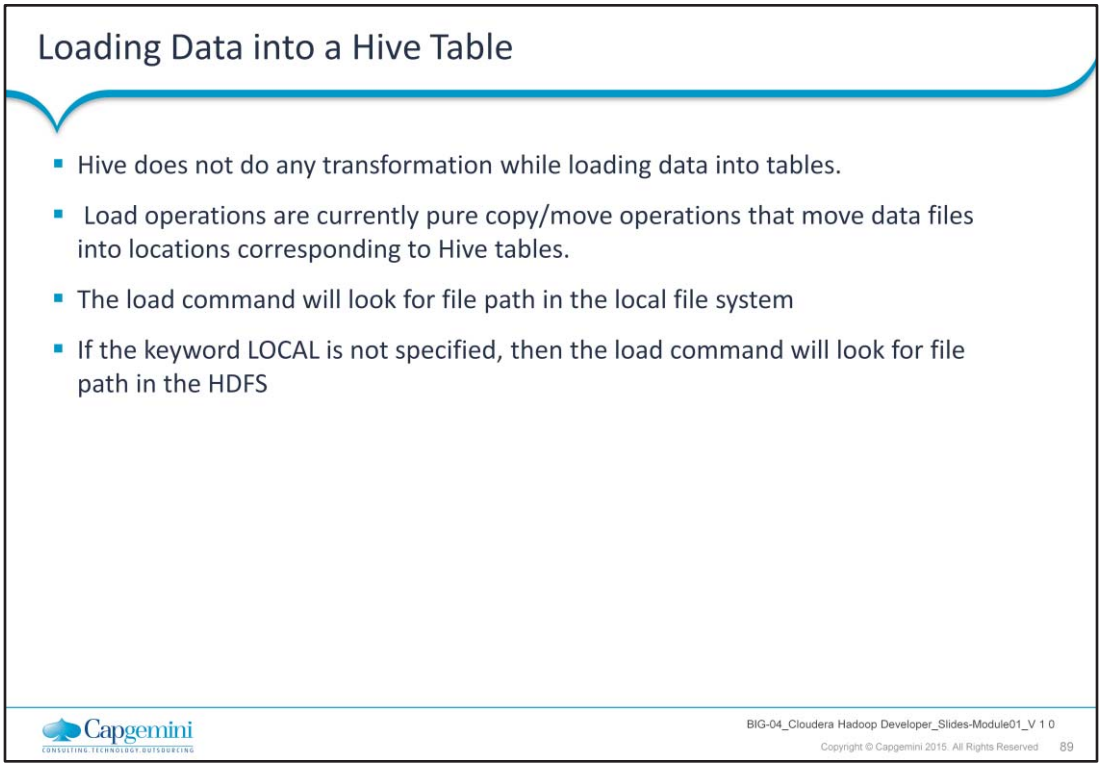
Hive 0.10
Hive 0.11
Future

Hive Basics: Creating Tables

```
hive> SHOW TABLES;

hive> CREATE TABLE shakespeare (freq INT, word STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;

hive> DESCRIBE shakespeare;
```

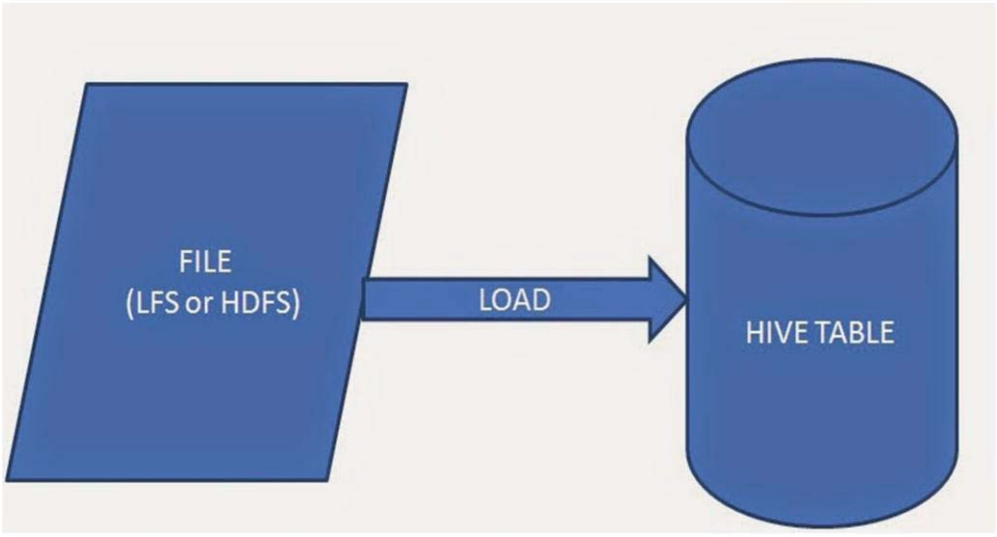
Hive Data: Physical Layout

- Hive tables are stored in Hive's 'warehouse' directory in HDFS
  - By default, /user/hive/warehouse



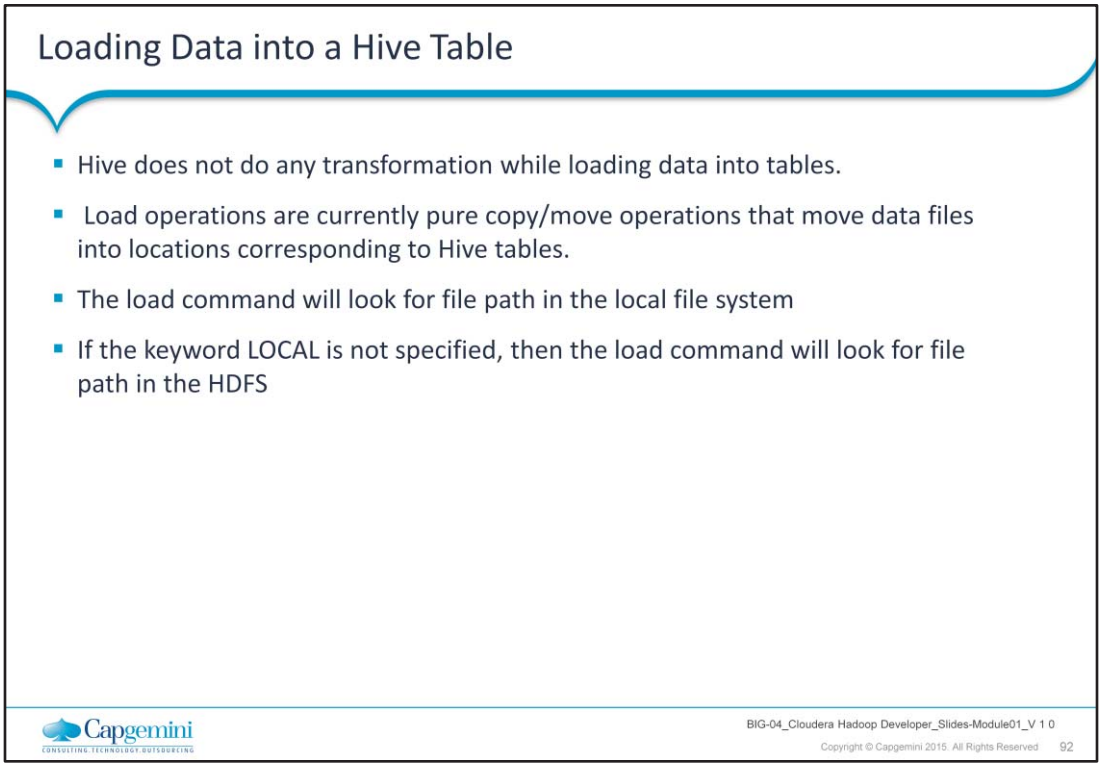
The **LOAD DATA** command can load files from the local file system (using the **LOCAL** qualifier) or files already in HDFS. For example, the following command loads a local file into a table named **customers**:

Loading files into Hive tables



Hive Basics: Creating Tables

```
hive> SHOW TABLES;

hive> CREATE TABLE shakespeare (freq INT, word STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;

hive> DESCRIBE shakespeare;
```



The **LOAD DATA** command can load files from the local file system (using the **LOCAL** qualifier) or files already in HDFS. For example, the following command loads a local file into a table named **customers**:

```
LOAD DATA LOCAL INPATH '/tmp/customers.csv' OVERWRITE INTO
TABLE customers;
```

If the data is already in HDFS, then leave off the **LOCAL** keyword:

```
LOAD DATA INPATH '/user/train/customers.csv' OVERWRITE INTO
TABLE customers;
```

In either case above, the file **customers.csv** is moved either into HDFS in a subfolder of **/apps/hive/warehouse** or to the table's **LOCATION** folder, and the contents of **customers.csv** are now associated with the **customers** table.

Basic SELECT Queries

- **Hive supports most familiar SELECT syntax**

```
hive> SELECT * FROM shakespeare LIMIT 10;

hive> SELECT * FROM shakespeare
WHERE freq > 100 ORDER BY freq ASC
LIMIT 10;
```