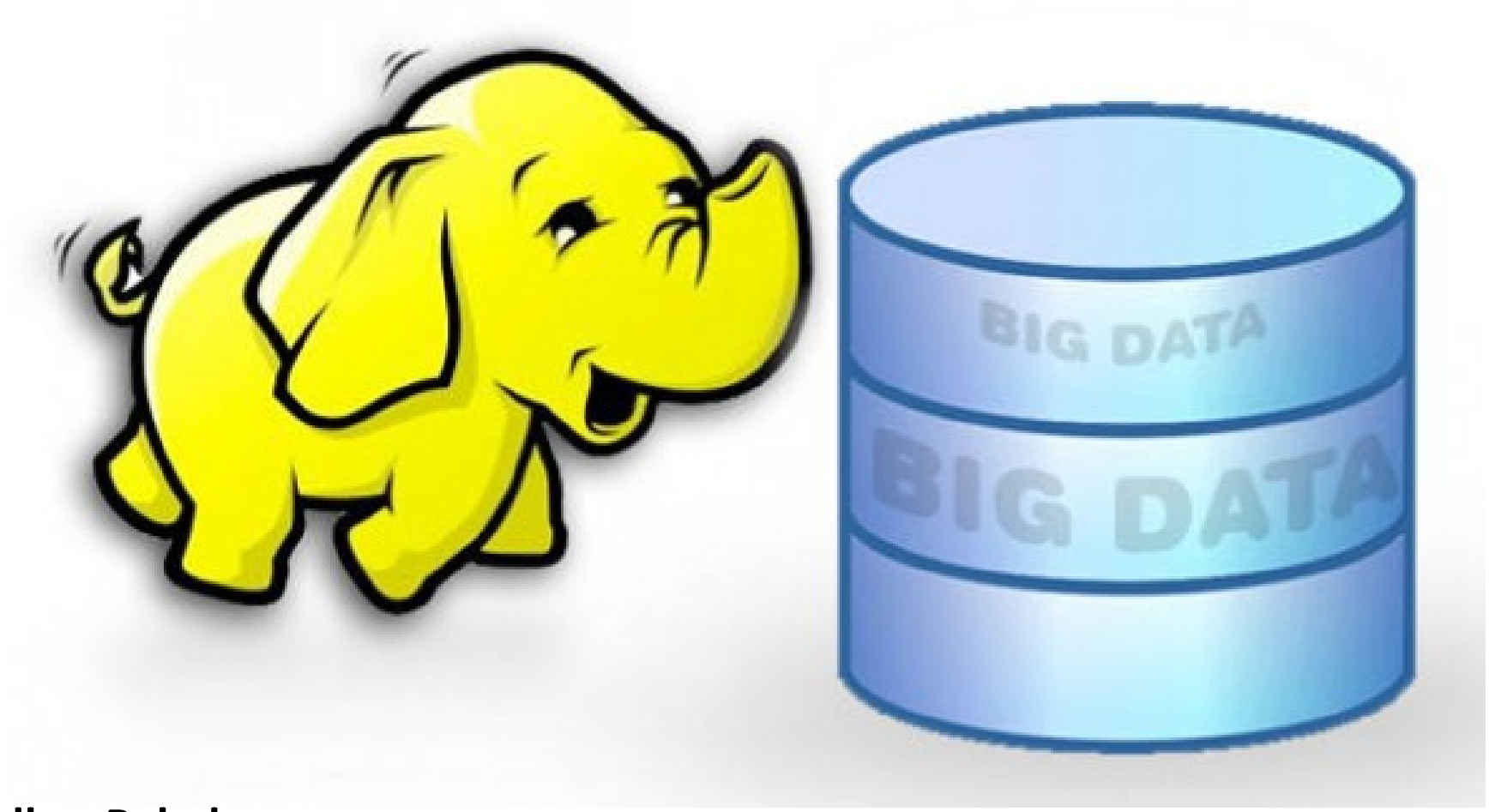
Big Data – Introduction to Apache™ Hadoop®



Sridhar Paladugu
http://www.linkedin.com/in/sridharpaladugu/

Agenda

- Big Data Concept
- What is Hadoop
- Architecture of Hadoop
- Basic Hadoop maintenance tasks
- Hadoop Sample Application
- Hadoop Eco System
- Pig Overview and sample application
- Hive Overview and sample application
- Hbase Overview
- Conclusion
- Questions

BIG DATA is often defined by 4 V's

Volume

Terabytes or even petabytes of data for processing and ever increasing yet

Velocity

Pace at which the data are to be consumed.

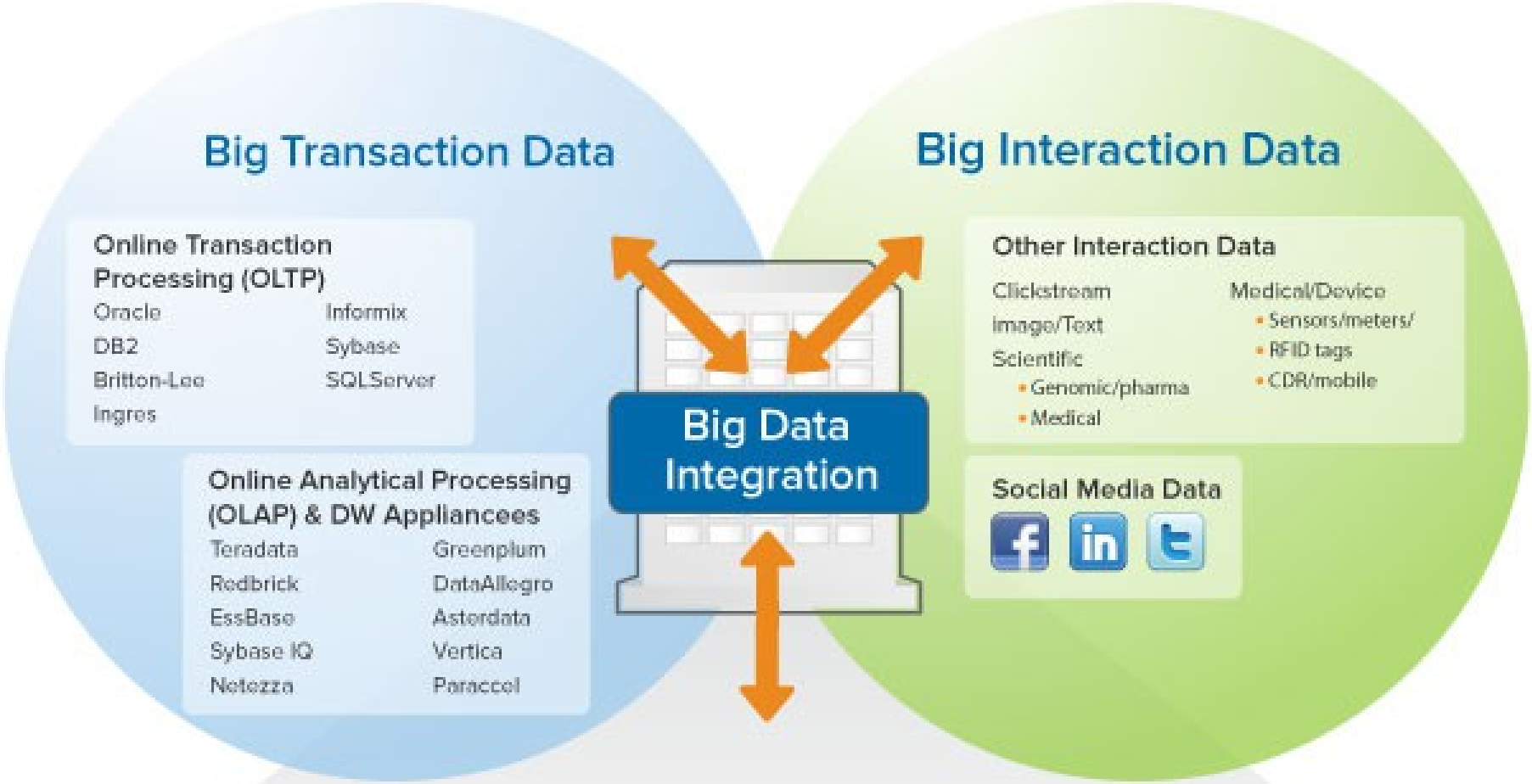
Variety

Structured and unstructured data

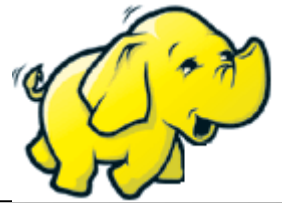
Veracity

Data inconsistency and incompleteness

Definition: Big data is the confluence of the three trends consisting of Big Transaction Data, Big Interaction Data and Big Data Processing

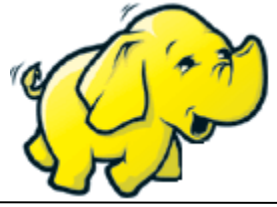


What is Hadoop?

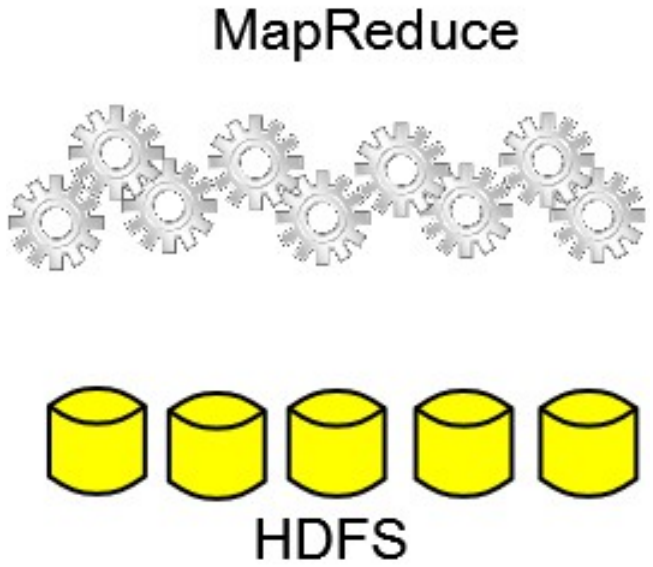


- open source project for big data processing.
- provides an infrastructure for a distributed storage and a distributed computation.
- Highly Scalable
- High degree of fault tolerant.
- Cost effective
- Flexible data format support.
- Redundant and reliable.

Core Components of Hadoop



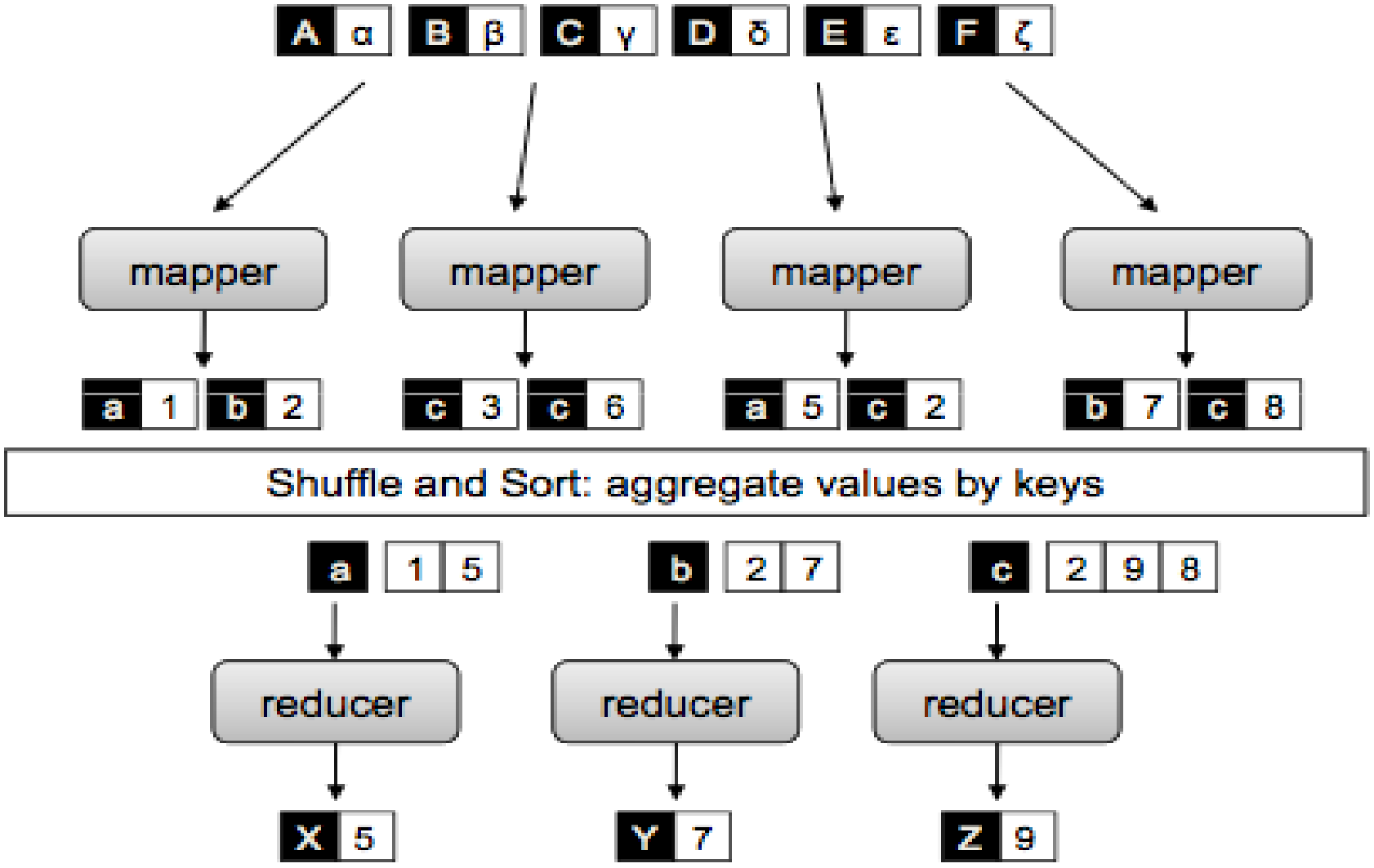
- Distributed File System (HDFS)
- Parallel computing framework (Map/Reduce)
- Hadoop commons - Job scheduler and Management features



MapReduce

- Programming model designed to process large data sets.
- Input & Output: each a set of key/value pairs
- **Map**
  - Processes input key/value pair and Produces set of key/value pairs
- **Reduce**
  - Shuffle map output and Combines all intermediate values for a each key.
  - Merge and produce output.

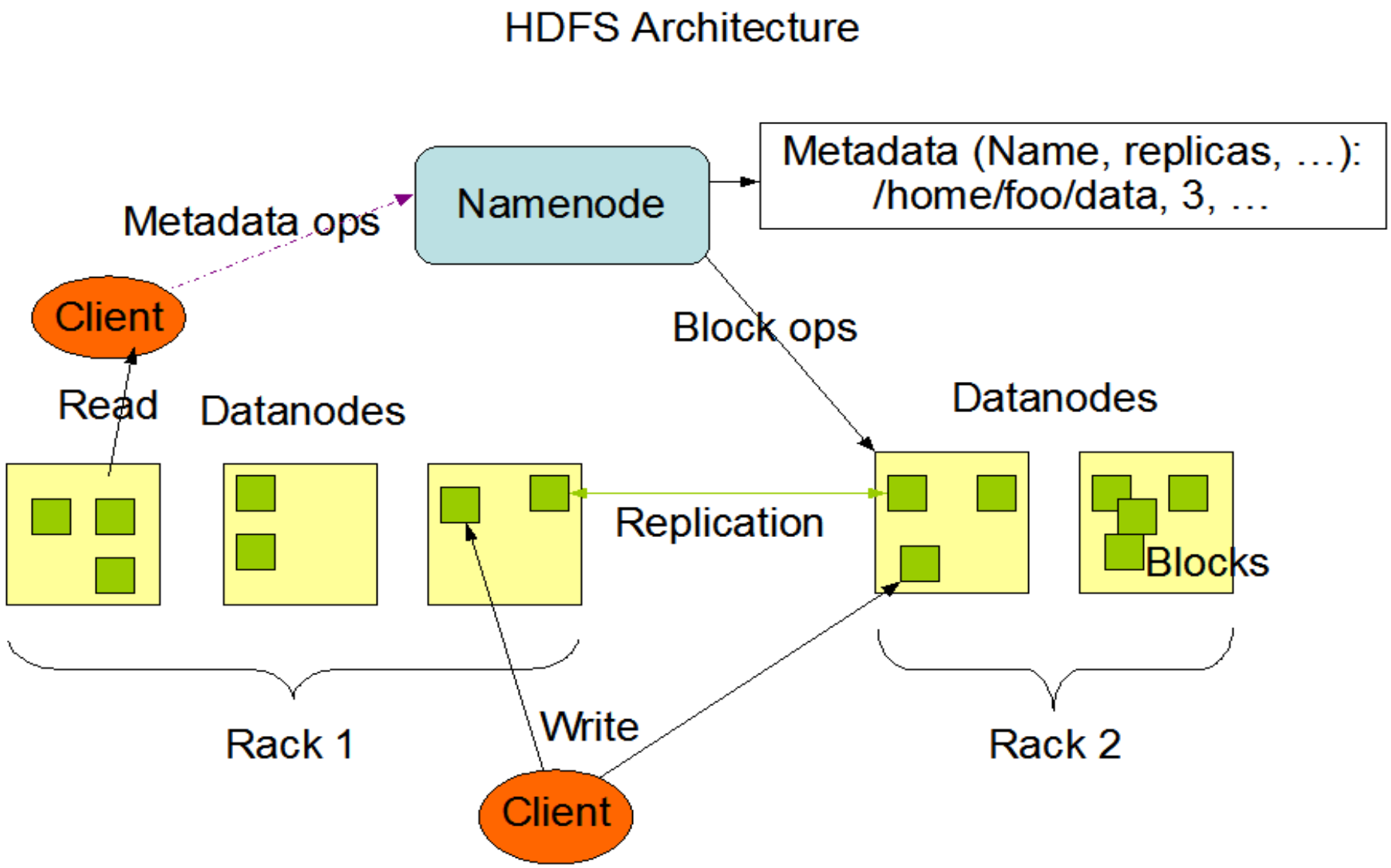
Map Reduce


A α
B β
C γ
D δ
E ε
F ζ
mapper
mapper
mapper
mapper
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8
reducer
reducer
reducer
X 5
Y 7
Z 9

HDFS

- **Distributed File system**
- **Write-once, Read-many model**
- **Data access via MapReduce streaming**
- **Fault Tolerance - Automatically and seamlessly recover from failures**
- **Load Balancing - Place data intelligently for maximum efficiency and utilization.**
- **Tunable Replication - Multiple copies of each file provide data protection and computational**
- **Security - POSIX-based file permissions for users and groups with optional LDAP integration**

HDFS


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

Working with HDFS

- Using Linux Shell we can run hdfs file system commands.
  - **$hadoop fs -ls /user/sridhar**
- Storing a file to HDFS
  - **$hadoop fs -copyFromLocal myfile.txt /user/sridhar/volume1**
- Retrieving a file from HDFS
  - **$hadoop fs -copyToLocal /user/sridhar/volume1 myfile.txt**
- Copy file to HDFS replication
  - **$hadoop distcp hdfs://namenodeA/user/sridhar hdfs://namenodeB/user/sridhar**
- Merge output
  - **$hadoop fs –getmerge /user/sridhar/equityPricing eqtyPricing_11302012.bcp**

Working with HDFS

- Using Linux Shell we can run hdfs file system commands.
  - **$hadoop fs -ls /usr/sridhar**
- Storing a file to HDFS
  - **$hadoop fs -copyFromLocal myfile.txt /usr/sridhar/volume1**
- Retrieving a file from HDFS
  - **$hadoop fs -copyToLocal /usr/sridhar/volume1 myfile.txt**
- Copy file to HDFS replication
  - **$hadoop distcp hdfs://namenodeA/usr/sridhar hdfs://namenodeB/usr/sridhar**
- Merge output
  - **$hadoop fs –getmerge /usr/sridhar/equityPricing eqtyPricing_11302012.bcp**

Working with HDFS : Sqoop

- **Open source tool to transfer data in and out from hadoop tp RDBMS.**
- **Utilizes MapReduce to do parallel execution and fault tolerance.**
- **Support Hive and hbase.**
- **Import from table or query**
  - **Divide table in to range using primary key or specified column.**
  - **Create mappers for each range and executes in parallel.**
  - **Can output in all formats supported by hdfs.**
  - **Generates java class for generated hdfs file.**
- **Export to Database**
  - **Utilize MapReduce to read data concurrently.**
  - **Utilize bulk insert using Parallelism.**

Import and Export using Sqoop

- **Import using query**

  - **$sqoop import**

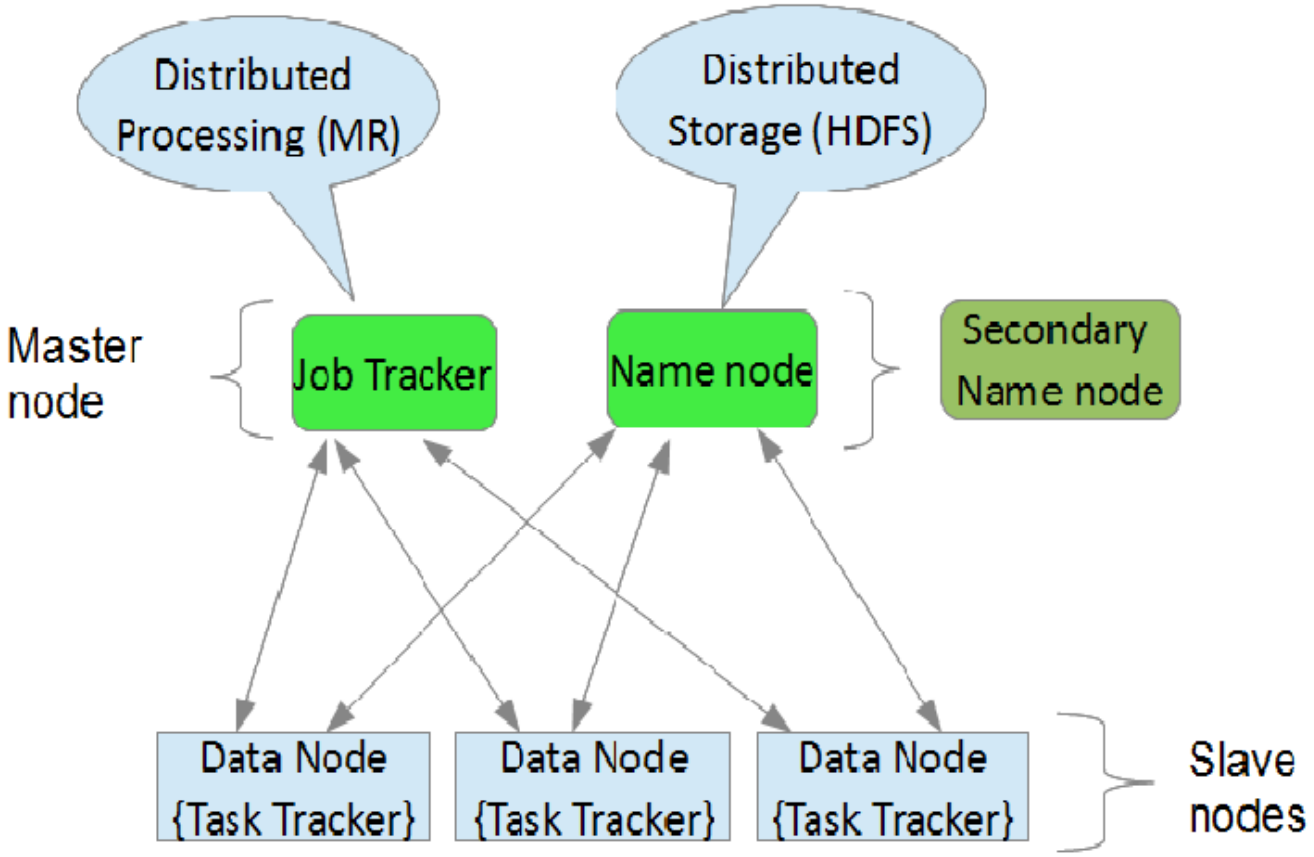
```
--connect  jdbc:oracle:thin:@tradeNode-1:1521:eqty
--username eqtyTrader --password passw0rd
--as -textfile
--query 'select t.tradeId,b.bookName, cp.cunterparty, t.tradeDate,t.mtm'\
        ' from      Trade t '\
        ' join Counterparty cp on t.counterpartyFk=cp.id '\
        ' join eodMarks20121130 m on t.tradeFk = m.tradeFk '\
--split-by t.tradeId
--fields-terminated-by '\t' --lines-terminated-by '\n' \
--targetDir /usr/eqty/mtm/20121130
```

  - ***[Incremental, last-value, hive-import,hive-overwrite, options-file]***

- **Export to database**

```
$sqoop export
--connect  jdbc:oracle:thin:@tradeNode-1:1521:eqty
--username eqtyTrader --password passw0rd
--table EOD_MTMSummary
--update-key id --update-mode=allowinsert
--export-dir /path/to/data
```

Hadoop Cluster view



Hadoop : Basic Configuration

Hadoop is mainly configured using files core-site.xml , hdfs-site.xml, mapred-site.xml

Core-site.xml; specifies the name node related configuration.

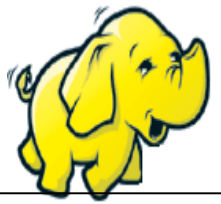
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
      <name>hadoop.tmp.dir</name>
      <value>/usr/local/hadoop-1.0.4/tmp</value>
   </property>
   <property>
     <name>fs.default.name</name>
     <value>hdfs://hdp-master.localdomain.com:54310</value>
   </property>
</configuration>
```

Hadoop : Basic Configuration

- **Hdfs-site.xml specify information related to hdfs**

```
<?xml version="1.0"?>
<configuration>
<property>
   <name>dfs.name.dir</name>
   <value>/usr/local/hadoop-1.0.4/hdfs/name</value>
 </property>
 <property>
   <name>dfs.data.dir</name>
   <value>/usr/local/hadoop-1.0.4/hdfs/data</value>
 </property>
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property>
</configuration>
```

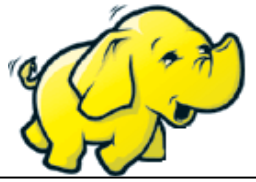
Hadoop : Basic Configuration



- **Mapred-site.xml** configuration related to job tracker is specified here

```
<?xml version="1.0"?>
<configuration>
<property>
 <name>mapred.job.tracker</name>
 <value>hdp-master.localdomain.com:54311</value>
 <description>job tracker addres</description>
</property>
<property>
  <name>dfs.hosts.exclude</name>
  <value>/path/to/hadoop/dfs_excludes</value>
  <final>true</final>
</property>
<property>
  <name>mapred.hosts.exclude</name>
  <value>/path/to/hadoop/mapred_excludes </value>
  <final>true</final>
</property>
<property><name>mapred.reduce.parallel.copies</name> <value>5<value> </property>
```

Hadoop : Basic maintenance tasks



- Start Hadoop
  - Standalone mode: run as a single process on a single node.
  - Pseudo-distributed mode: run all services in separate processes on a single node.
  - Fully-distributed mode: run all services in separate processes across multiple nodes.
    - **sridhar@hdp-standalone:/usr/local/hadoop/bin/start-all.sh**

sridhar@hdp-standalone:/usr/local/hadoop-1.0.4/myscripts$ jps

15020 Jps
24033 NameNode
24815 TaskTracker
24274 DataNode
24504 SecondaryNameNode
24589 JobTracker
9359 RunJar

- Stop Hadoop

$/usr/local/hadoop/bin/stop-all.sh

Hadoop : Basic maintenance tasks

- Add nodes to cluster
  - Setup new linux machine and install the distribution; hdp-node-3

    sridhar@hdp-master:/usr/local/hadoop-1.0.4/conf$ vi slaves

    hdp-node-1.localdomain.com

    hdp-node-2.localdomain.com

    hdp-node-3.localdomain.com
  - In new slave machine, run below commands

    sridhar@hdp-node-3:/usr/local/hadoop-1.0.4/bin/start datanode

    sridhar@hdp-node-3:/usr/local/hadoop-1.0.4/bin/start tasktracker
  - In Master node, run

    sridhar@hdp-master:/usr/local/hadoop-1.0.4/bin/start-balancer.sh

Hadoop : Basic maintenance tasks

- **Decommission a node**

sridhar@hdp-master:/usr/local/hadoop-1.0.4/conf$**vi dfs_excludes**

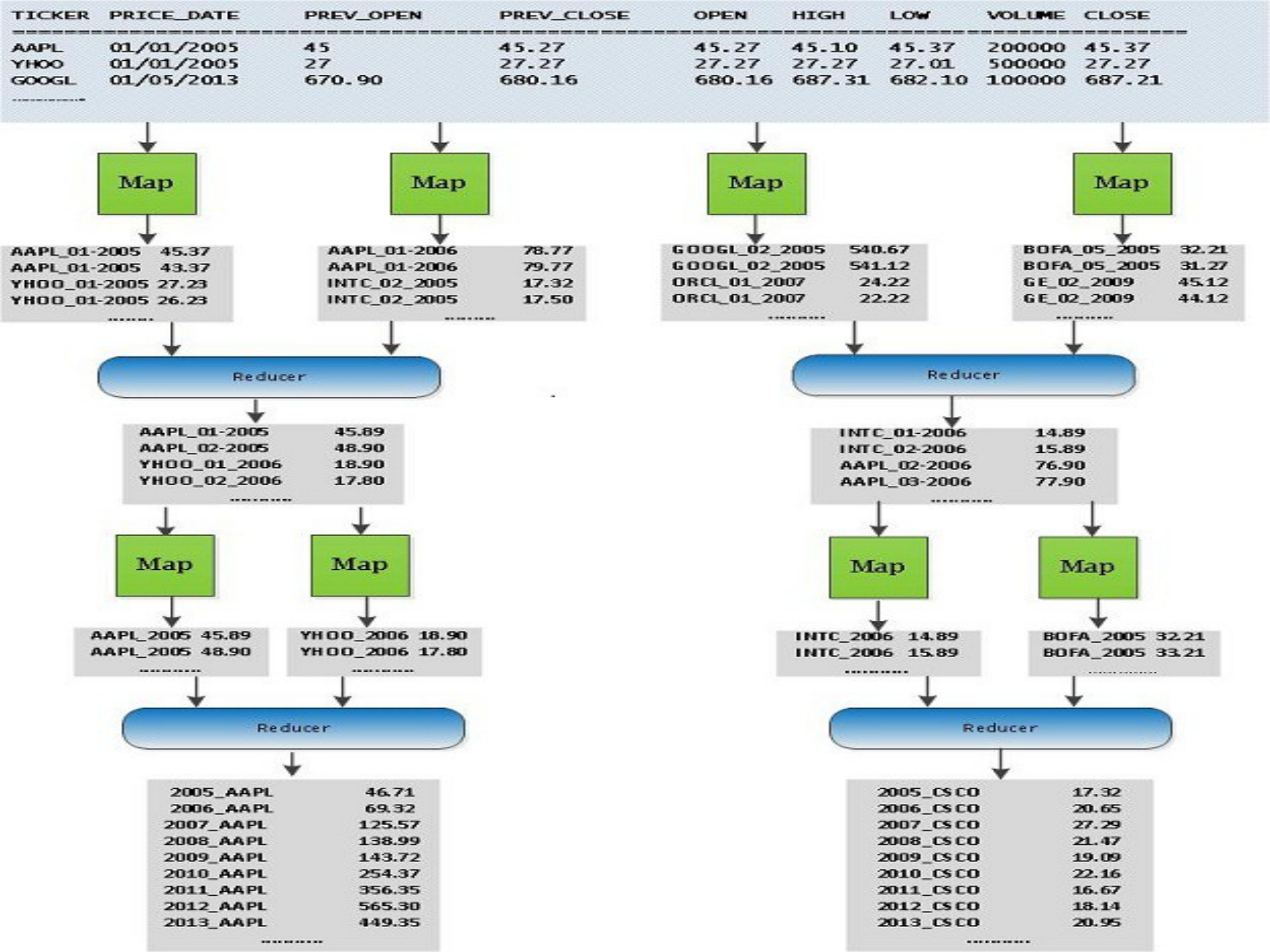
hdp-node-3.localdomain.com

sridhar@hdp-master:/usr/local/hadoop-1.0.4/conf$**vi mapred_excludes**

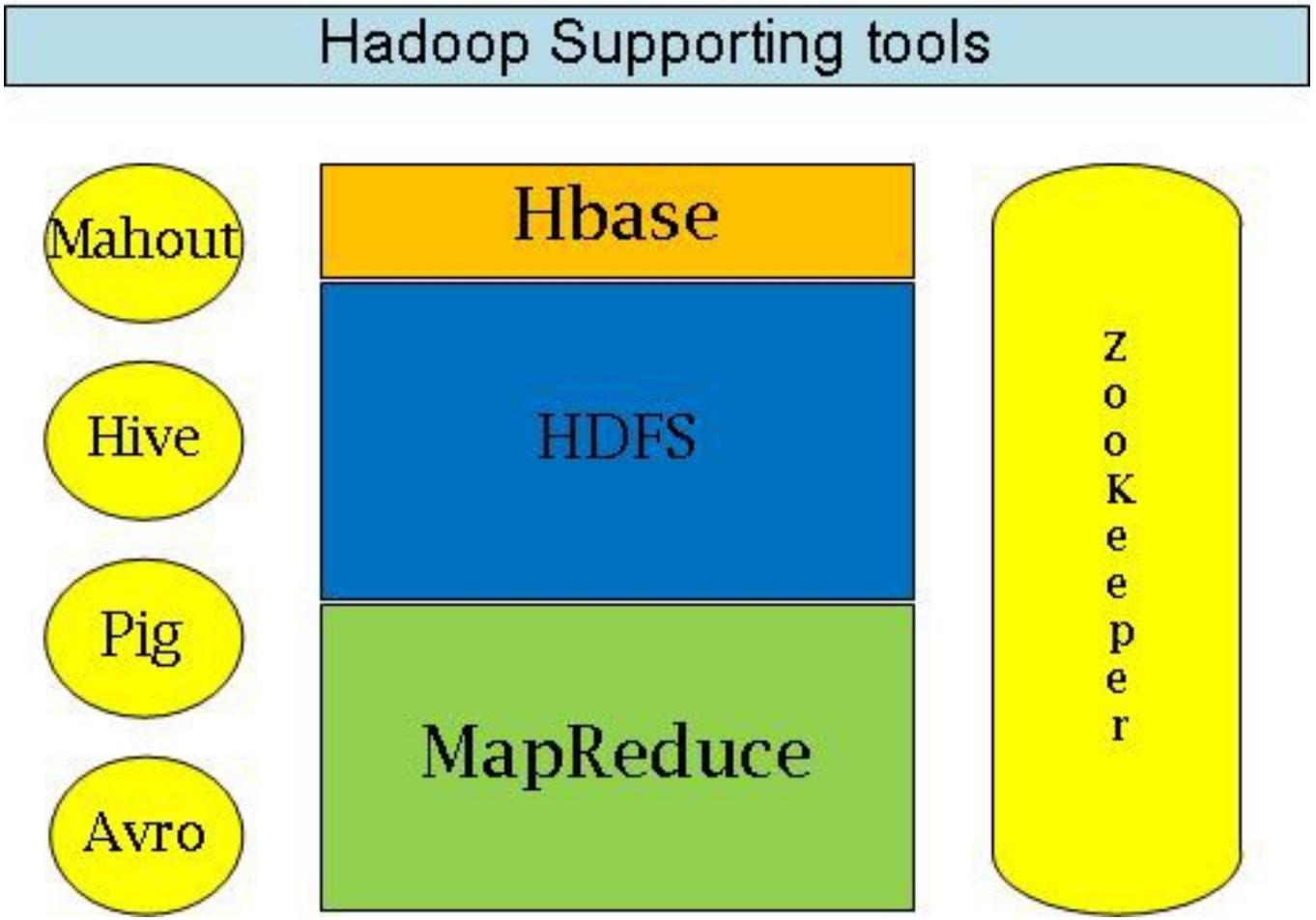
hdp-node-3.localdomain.com

sridhar@hdp-master:/usr/local/hadoop-1.0.4/bin$**hadoop dfsadmin –refreshNodes**

sridhar@hdp-master:/usr/local/hadoop-1.0.4/bin$**hadoop mradmin –refreshNodes**

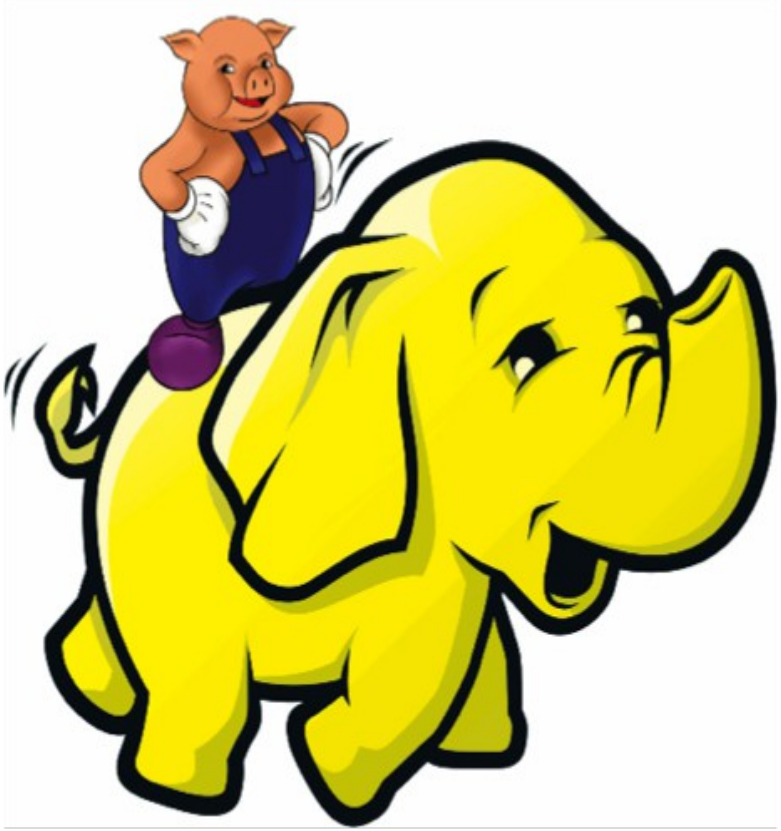

TICKER PRICE_DATE PREV_OPEN PREV_CLOSE OPEN HIGH LOW VOLUME CLOSE
AAPL 01/01/2005 45 45.27 45.27 45.10 45.37 200000 45.37
YHOO 01/01/2005 27 27.27 27.27 27.27 27.01 500000 27.27
GOOGL 01/05/2013 670.90 680.16 680.16 687.31 682.10 100000 687.21
Map
Map
Map
Map
AAPL_01-2005 45.37
AAPL_01-2005 43.37
YHOO_01-2005 27.23
YHOO_01-2005 26.23
AAPL_01-2006 78.77
AAPL_01-2006 79.77
INTC_02_2005 17.32
INTC_02_2005 17.50
GOOGL_02_2005 540.67
GOOGL_02_2005 541.12
ORCL_01_2007 24.22
ORCL_01_2007 22.22
BOFA_05_2005 32.21
BOFA_05_2005 31.27
GE_02_2009 45.12
GE_02_2009 44.12
Reducer
Reducer
AAPL_01-2005 45.89
AAPL_02-2005 48.90
YHOO_01_2006 18.90
YHOO_02_2006 17.80
INTC_01-2006 14.89
INTC_02-2006 15.89
AAPL_02-2006 76.90
AAPL_03-2006 77.90
Map
Map
Map
Map
AAPL_2005 45.89
AAPL_2005 48.90
YHOO_2006 18.90
YHOO_2006 17.80
INTC_2006 14.89
INTC_2006 15.89
BOFA_2005 32.21
BOFA_2005 33.21
Reducer
Reducer
2005_AAPL 46.71
2006_AAPL 69.32
2007_AAPL 125.57
2008_AAPL 138.99
2009_AAPL 143.72
2010_AAPL 254.37
2011_AAPL 356.35
2012_AAPL 565.30
2013_AAPL 449.35
2005_CSCO 17.32
2006_CSCO 20.65
2007_CSCO 27.29
2008_CSCO 21.47
2009_CSCO 19.09
2010_CSCO 22.16
2011_CSCO 16.67
2012_CSCO 18.14
2013_CSCO 20.95

Hadoop Eco System



Apache PIG

- High level structured query language for analyzing data stored in Hadoop cluster.
- Provide a run-time time environment to generate Map Reduce programs from a high level data-flow statements.
- Pig Latin, the data-flow language
- Grunt pig shell, run-time
- Not a full featured programming language, or not equivalent to SQL.
- Extensible
- Why Pig?



Apache PIG Run-time environment

- Pig run-time is started using ;
  - $**/usr/local/pig/pig**

$**grunt**>

$**grunt**> data = LOAD(...) AS(...)

$**grunt**> filter_by_v = FILTER data by (vendor = 'xyz')

$**grunt**> group_by_dt = GROUP filer_by_v BY (shop_date)

$**grunt**>transactions = FOR EACH group_by_dt GENERATE

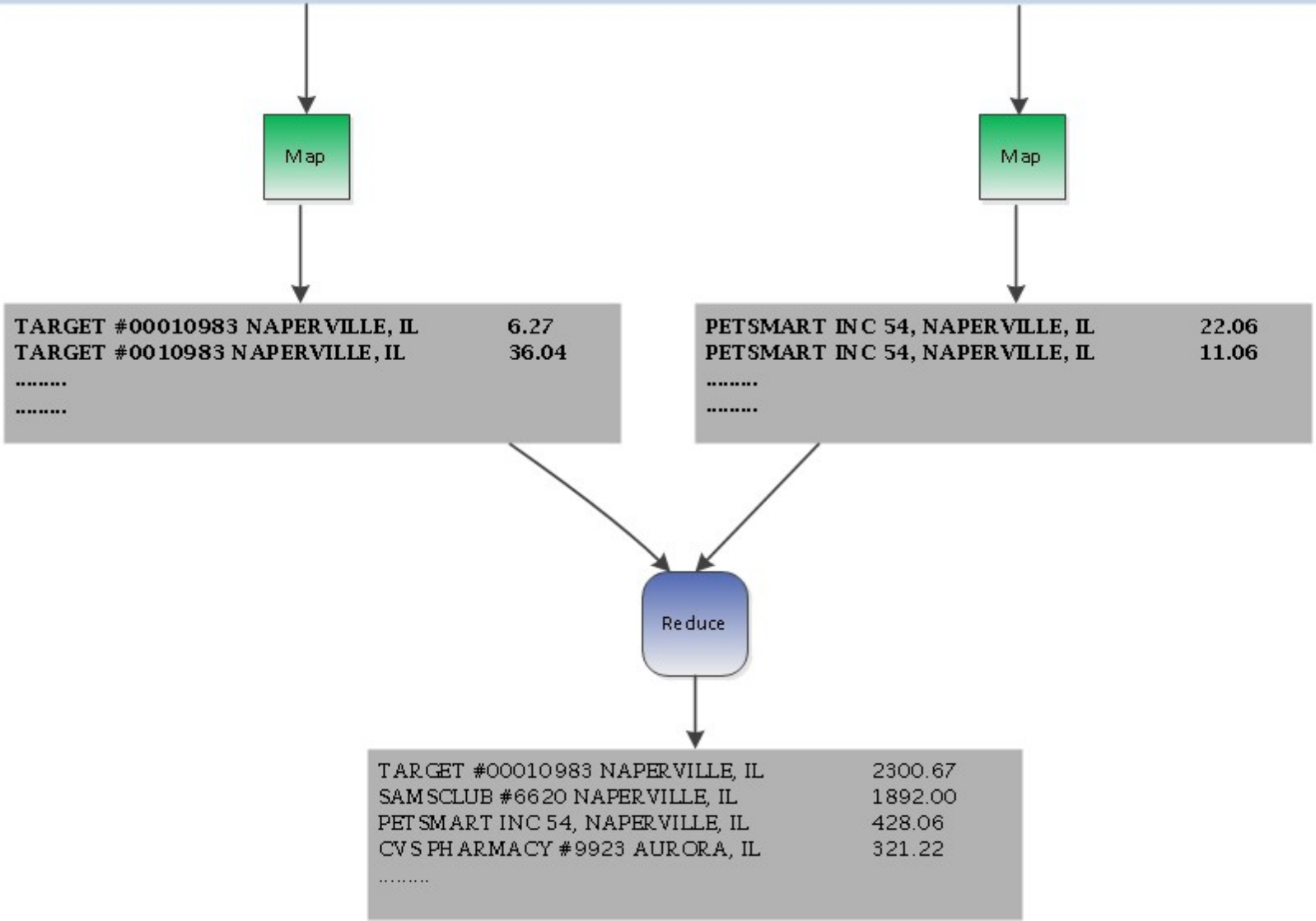
group, sum(amt)

$**grunt**> STORE group_by_v INTO (....)

Apache Pig – Development and Testing

- Pig run time supports User defined functions defined in Java, Ruby, Python code
- Running a pig script
  - **$/usr/local/pig/pig -x sampleScript_1.pig**
- Describe -- print the schema of a data structure
  - **DESCRIBE data_alias;**
- Explain – print how map reduce jobs are being generated for a script
  - **Explain data_alis**
  - **/usr/local/pig/pig -x local -e 'explain -script sampleScript_1.pig'**
- Illustrate --takes a sample of your data and runs it through your script
  - **/usr/local/pig/pig -x local -e 'illustrate -script illustrate.pig'.**

Transaction Date	Description	Amount	Comments
07/21/2012	TARGET 00010983 NAPERVILLE, IL	5.11	
07/20/2012	PAYMENT	-1330.33	Return processed***
07/20/2012	TRU HOLIDAY EXPRESSQPS NAPERVILLE, IL	10.74	
07/13/2012	TARGET 00010983 NAPERVILLE, IL	6.27	
07/10/2012	TARGET 00010983 NAPERVILLE, IL	36.04	



Apache PIG : Sample script

```
all_credit_tx = LOAD 'hdfs://hdp-
   master.localdomain.com:54310/user/sridhar/data/cred
   it_card_tx.txt' AS(transactionDate:chararray,
   transactionDesc:chararray, amount:float);

filter_returns = FILTER all_credit_tx BY
   (transactionDesc != 'PAYMENT');

group_by_vendor = GROUP filter_returns by
   transactionDesc;

sum_vendor_exp = FOREACH group_by_vendor GENERATE
   group, SUM(filter_returns.amount);

STORE sum_vendor_exp INTO
   '/usr/sridhar/pig/out/SPENDING_ANALYSIS';
```

Apache PIG : Sample script Results

Run the script using

- $pig -f ExpByVendor.pig

SKECHERS-USA #119 CHICAGO, IL 2127021.75

SIX FLAGS GREAT AMERICA IL IL 2127777.10

76 10115095 NAPERVILLE, IL 2122521.27

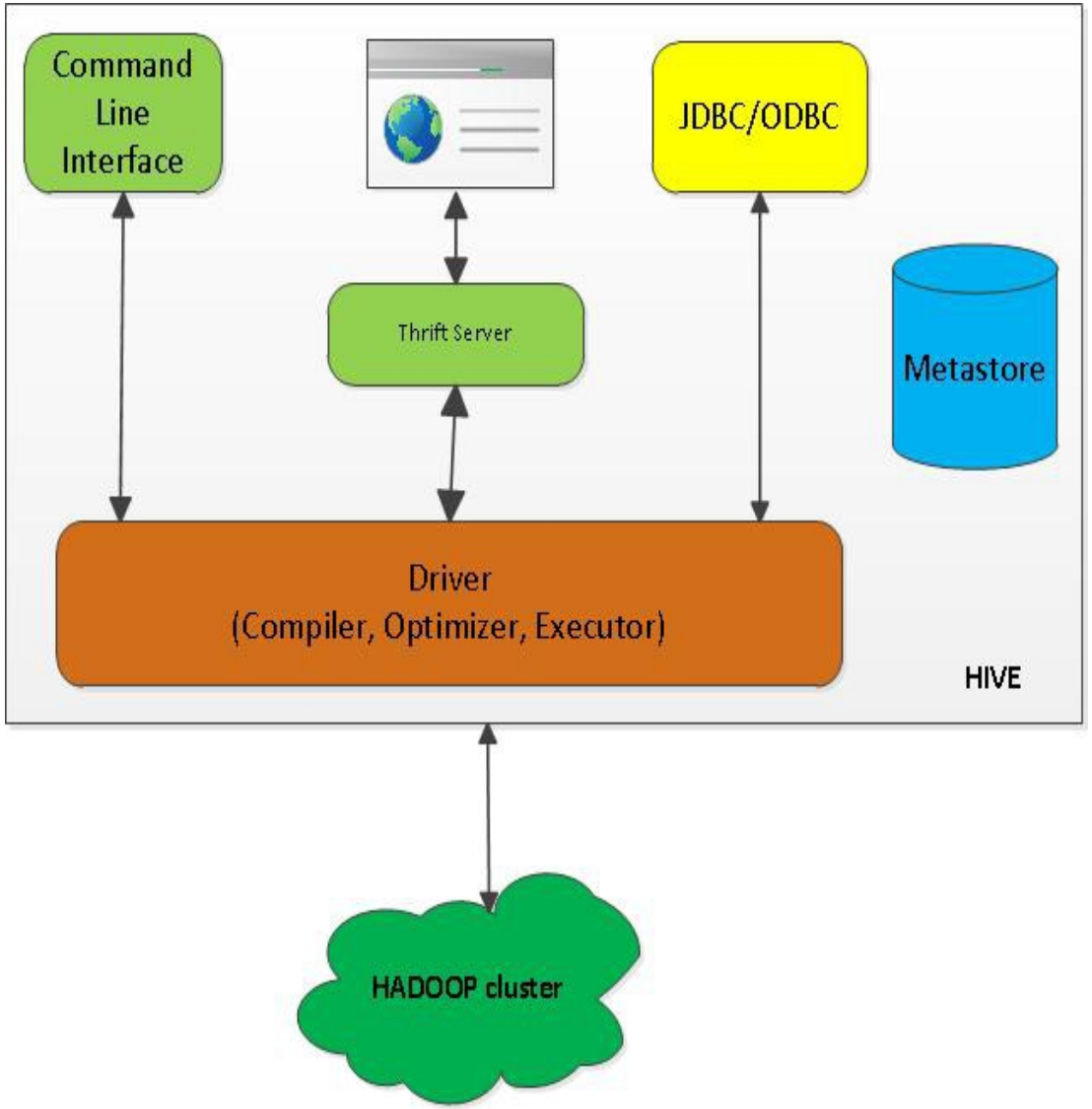
BABIES R US #6447 QPS NAPERVILLE, IL 2126748.77

Apache PIG: Conclusion

- provides easy way of implementing Map reduce jobs.
- Provides extensible features via UDF
- Provides less complex way of creating job flows
- Provides Lazy evaluation
- Ability to store data anytime during flow.
- Provides out of box performance enhancers

Apache HIVE

- A distributed data warehouse engine built on top of Hadoop
- Provides HQL interface to HDFS data
- Utilize Map-Reduce to execute queries
- Work with data in various formats
- Extensible
- Supports Partitions and Joins



Apache HIVE : Data Types

- Domain objects
  - Tables, partitions, Buckets
- Data Types
  - Primitives
    - TINYINT, SMALLINT, INT, BIGINT, BOOLEAN,FLOAT, DOUBLE, STRING
  - Complex types
    - STRUCT, MAP, ARRAY

Apache HIVE – DDL operations

- Start hive CLI
  - $hive
- hive>DROP TABLE IF EXISTS creditcard_transactions;
- hive>CREATE EXTERNAL TABLE creditcard_transactions (

transactionDate STRING, description STRING, amount FLOAT, comments STRING)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'

LOCATION '/usr/sridhar/creditcard_txs/';

- hive>show tables;
- hive> describe creditcard_transactions;

Apache HIVE : DML statements

- **Selecting data**

```
select * from creditcard_transactions
where UNIX_TIMESTAMP(transactionDate,'yyyy-MM-dd') >=
    UNIX_TIMESTAMP('2012-01-01', 'yyyy-MM-dd')
and UNIX_TIMESTAMP(transactionDate,'yyyy-MM-dd') <=
    UNIX_TIMESTAMP('2012-01-30', 'yyyy-MM-dd')
```

- **Supports Group BY, Having and Limit**
  - Select description, sum(amount)
    from creditcard_transactions
    where transactionDate='2012-01-02'
    group by description
    having sum(amount) < 3000
    sort by description desc LIMIT 10;
- **Writing data out to table or file system**
  - INSERT INTO OVERWRITE saleBy_Vendaor SELECT ….
  - INSERT INTO OVERWRITE [LOCAL] directory path_to_write SELECT...

Apache HIVE: Table Partitioning

- Hive organizes table as partitions.
- each partition as a separate file
- Data defined in one partition belong to a key, making access fast
- Partitions can have sub-partitions.
- While loading data into partition table we need to specify partition.
- Dynamic partitioning is allowed*

```
CREATE TABLE mgd_creditcard_transactions (
description STRING,  amount FLOAT,   comments STRING)
PARTITIONED BY(transactionDate STRING);
```

Apache HIVE: Populating a Partition

- Exclusive partition load

```
INSERT OVERWRITE TABLE mgd_creditcard_transactions
PARTITION (transactionDate='2012-01-02')
SELECT description,amount,comments,transactionDate
FROM creditcard_transactions
WHERE transactionDate='2012-01-02';
```

- Dynamic partition load

```
SET hive.exec.dynamic.partition = true;
SET hive.exec.dynamic.partition.mode = nonstrict;
SET hive.exec.max.dynamic.partitions.pernode = 200;
set hive.exec.max.dynamic.partitions = 200000;

INSERT OVERWRITE TABLE mgd_creditcard_transactions
PARTITION (transactionDate)
SELECT description,amount,comments,transactionDate
    FROM creditcard_transactions;
```

Apache HIVE: Extensibility

- Support custom code via UDF**.**

```
Package com.sridhar.hadoop.samples.hive;
import java.math.BigDecimal;
import java.text.NumberFormat;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
public class CurrencyFormatter extends UDF {
 public Text evaluate(final Text input) {
 if (input == null) { return null; }
 BigDecimal d = new BigDecimal(input.toString());
    return new Text(NumberFormat.getCurrencyInstance().format(d));
 }}
```

- **Register the function**

  CREATE FUNCTION CU_FMT AS com.sridhar.hadoop.samples.hive.CurrencyFormatter

- **Use in hql**

```
select transactionDate, Description, CU_FMT(amount)
 from mgd_creditcard_transactions where transactionDate='2012-01-02';
```

Apache HIVE : Conclusion

- **Advantages**
  - Provides easy interface to HDFS by hiding complexity of Map Reduce.
  - Support SQL-based queries and thus less less learning curve.
  - provides External Table feature to leverage to keep data in HDFS
  - Provides meta store which allows decoupling application logic from physical data structure
  - Support for data partitioning for faster access
  - Easy to plug-in custom implementation for user defined functions
- **Disadvantages**

  Not designed for OLTP

  Latency issues because of need to generate Map Reduce jobs

Apache HBase

Implemented using Google's Big Table paper.

A column-oriented database stores data in key value pairs.

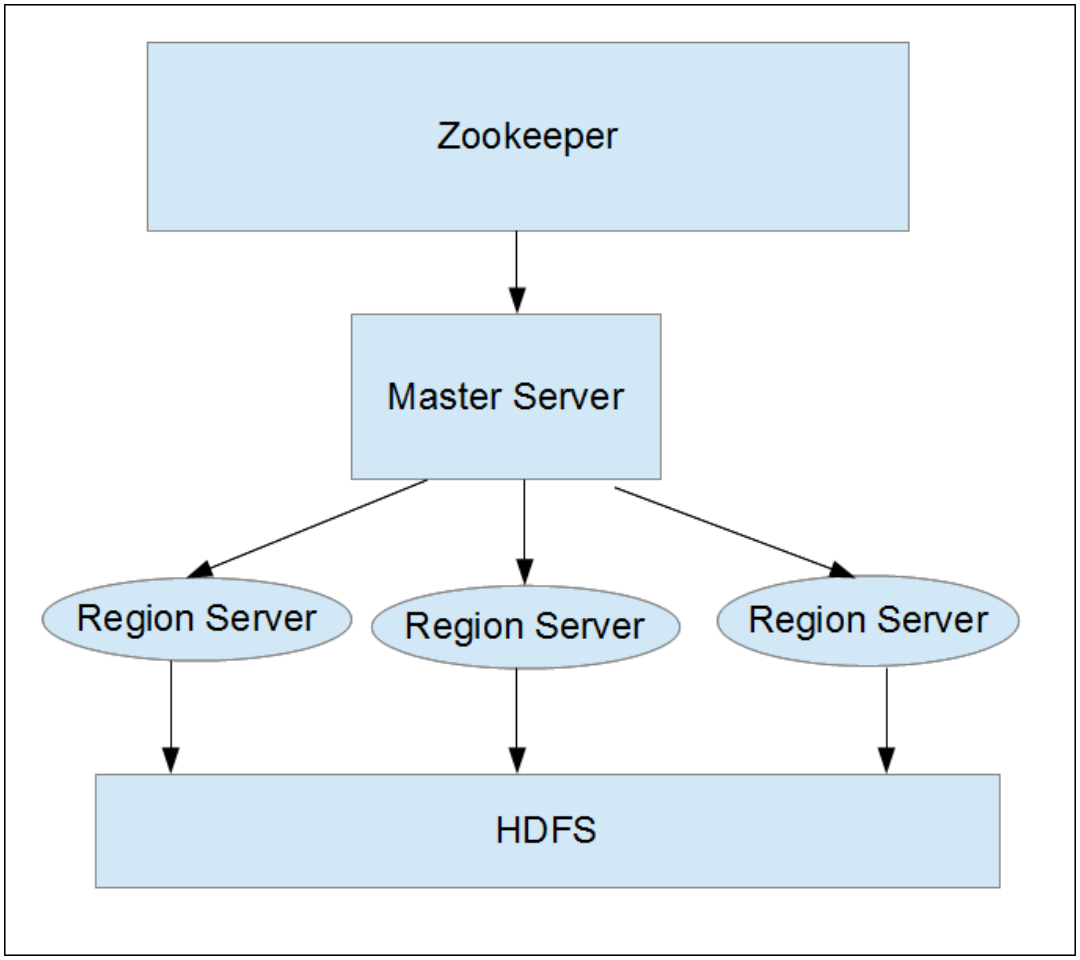
Characteristics:

- Sparse
- Distributed
- Sorted
- Multidimensional
- Version aware

Features:

- Vertical and and horizontally scalable
- Automatic Fail-over
- Consistent reads and Writes
- Integration with Map Reduce framework
- Simple Java API for Client access

Apache HBase - Architecture



Apache Hbase – Data Model

- Data is stored in Table
- Row = {key, value}
  - Key can be primitive type or custom data structure
  - Value is typically a set of column families
- A Column family is a map of column cells
  - row_1 {col_family {cell1, cell2....} }
- Column cell is a key value pair with version
  - Row_1 {col_family_1: {cell_1, LIST{cell_value, cell_version}

    col_family_1: {cell_2, LIST{cell_value, cell_version}

    …….. }

    …..}
- Java construct:

  SortedMap<Key, List<SortedMap<key, List<Value, timestamp>>

Apache Hbase – Table Storage

Logical View:

- Table is made of regions.
- Region is a range of Keys, key range is inclusive.
- Each Region Server hosts one or more regions
- Regions grow dynamically as data grows

Physical View:

- WAL (Write Ahead Log)
- MemStore – in memory storage
- HFile – an indexed file. Each flush operation creates one file
- -ROOT-
- .META – Region meta-base

Apache HBase : Hbase Shell

Login to Shell

```
sridhar@hdp-master:/usr/local/hbase-0.94.8/bin$ hbase shell
```

Create Table

```
hbase(main):006:0> create 'books', {NAME => "info", VERSIONS => 5}
```

Insert data to hbase table using put

```
hbase(main):006:0> put 'books', 'book2', 'info:title','HBase In Action'
```

Fetch data:

```
hbase(main):007:0> scan 'books'
ROW                 COLUMN+CELL
book1               column=info:title, timestamp=1371505821795, value=Hadoop Definitive guide
                                                                      2nd version
book2               column=info:title, timestamp=1371525002008, value=HBase InAction

2 row(s) in 0.1590 seconds
```

Fetch Data for a specific cell value:

```
hbase(main):002:0> scan 'books', { COLUMNS => 'info:title', LIMIT => 10, FILTER => "ValueFilter( =,
'binaryprefix:Hadoop' )" }
ROW                 COLUMN+CELL
book1               column=info:title, timestamp=1371505821795, value=Hadoop Definitive guide 2nd version

1 row(s) in 0.1550 seconds
```

Find rows in a table:

```
hbase(main):003:0> count 'books'
2 row(s) in 0.0140 seconds
```

Apache HBase: Import/Export Data

- Export/Import MapReduce jobs

  Export [-D<property=value>]* <tablename> <outputdir> [<versions>
  [<starttime> [<endtime>]] [^[regex pattern] or [Prefix] to filter]]

  - sridhar@hdp-master:hbase org.apache.hadoop.hbase.mapreduce.Export -Dmapred.output.compress=true -Dmapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec -Dmapred.output.compression.type=BLOCK 'books' /home/sridhar/Downloads/hbaseExport

  Import <tablename> <inputdir>

  - sridhar@hdp-master:hbase org.apache.hadoop.hbase.mapreduce.Import 'books_restored' /home/sridhar/Downloads/hbaseExport

Apache HBase: Import Data

- Using ImportTsv:
  - create 'daily_stock_prices', {NAME => "price_info", VERSIONS => 5}

sridhar@hdp-master:/usr/local/hbase-0.94.8/bin$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY, price_info:exchange,price_info:ticker,price_info:priceDate,price_info:open,price_info:high,price_info:low,price_info:close,price_info:volume,price_info::adjustedClose daily_stock_prices /user/sridhar/data/nasdaq.csv

  - hbase(main):002:0> get 'daily_stock_prices', 'NASDAQ_ACAT_2006-10-05'

COLUMN CELL
price_info::adjustedClose timestamp=1371535356480, value=16.18
price_info:close timestamp=1371535356480, value=17.09
price_info:exchange timestamp=1371535356480, value=NASDAQ
price_info:high timestamp=1371535356480, value=17.47
price_info:low timestamp=1371535356480, value=16.94
price_info:open timestamp=1371535356480, value=17.4
price_info:priceDate timestamp=1371535356480, value=2006-10-05
price_info:ticker timestamp=1371535356480, value=ACAT
price_info:volume timestamp=1371535356480, value=120800

  - 9 row(s) in 0.5280 seconds

Apache HBase: Conclusion

- HBase is not a RDBMS replacement.
- Use only when data is huge.
- Use if application does not use and RDBMS features.
- Use when data is loaded and accessed using key.
- Use when application has a variable schema.
- Use when data volumes are expected to grow and need to scale cost effectively.

Conclusion

- At present Hadoop is only cost effective parallel computing environment for Big Data
- Hadoop is not a replacement for current database systems.
- Hadoop is 10 years old and growing market share.
- Hadoop MR2 is maturing and coming up with more robust features.

Thank you.

Sridhar Paladugu
http://www.linkedin.com/in/sridharpaladugu/