Introduction to Big Data

Pooya Jamshidi
pooya.jamshidi@ut.ac.ir

Ilam University
School of Engineering,
Computer Group

April 18, 2025

Ilam University

Map-Reduce and the New Software Stack

MapReduce

- Much of the course will be devoted to large scale computing for data mining.
- **Challenges:**
  - How to distribute computation?
  - Distributed/parallel programming is hard.
- **Map-reduce** addresses all of the above.
  - Google's computational/data manipulation model.
  - Elegant way to work with big data.

Seminal MapReduce Paper: Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM, 2008.

The MapReduce Paradigm

- Platform for reliable, scalable parallel computing.
- Abstracts issues of distributed and parallel environment from programmer.
- Runs over distributed file systems.
    - Google File System

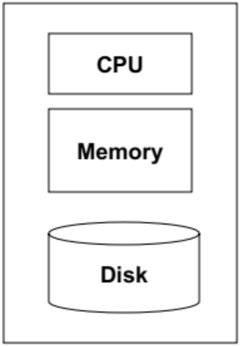
MapReduce: Insight

- Consider the problem of counting the number of occurrences of each word in a large collection of documents.
- How would you do it in parallel?
- **Solution:**
    - Divide documents among workers.
    - Each worker parses document to find all words, outputs (word, count) pairs.
    - Partition (word, count) pairs across workers based on word.

Single Node Architecture

CPU	Memory

Machine Learning, Statistics



Motivation: Google Example

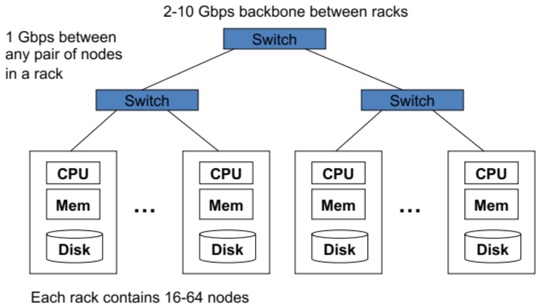
- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from a spinning disk.
  - ~4 months to read the web
- ~100 hard drives to store the web.
- Takes even more to do something useful with the data!
- Today, a standard architecture for such problems is emerging:

Cluster Architecture

- 1 Gbps between any pair of nodes in a rack.
- 2-10 Gbps backbone between racks.
- **Switch**
  - Each rack contains 16-64 nodes.
  - CPU, Memory, Disk

In 2011 it was estimated that Google had 1M machines, *`http://bit.ly/Shh0RO`*

Cluster Architecture


2-10 Gbps backbone between racks
1 Gbps between any pair of nodes in a rack
Switch
Switch
Switch
CPU
Mem
Disk
...
Each rack contains 16-64 nodes

Large-scale Computing

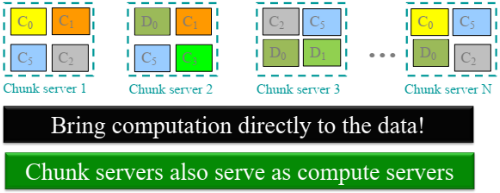
- Large-scale computing for data mining problems on commodity hardware.
- **Challenges:**
  - How do you distribute computation?
  - How can we make it easy to write distributed programs?
  - Machines fail:
    - One server may stay up 3 years ($\sim$1,000 days)
    - If you have 1,000 servers, expect to lose 1/day. Why?
    - People estimated Google had $\sim$1M machines in 2011.
    - How many machines fail every day?

Idea and Solution

- **Issue: Copying data over a network takes time**
- **Idea:**
  - Bring computation close to the data.
  - Store files multiple times for reliability.
- **Map-reduce** addresses these problems
  - Google's computational/data manipulation model.
  - Elegant way to work with big data.
- **Storage Infrastructure – File system.**
  - Google: GFS / HDFS
- **Programming model**
  - Map-Reduce

Distributed File System

- **Reliable distributed file system**
- Data kept in “chunks” spread across machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure



Programming Model: MapReduce

- Inspired from map and reduce operations commonly used in functional programming languages like Lisp.
- **Warm-up task:**
  - We have a huge text document.
  - Count the number of times each distinct word appears in the file.
- **Sample application:**
  - Analyze web server logs to find popular URLs.

Task: Word Count

- **Case 1:**
  - File too large for memory, but all <word, count >pairs fit in memory.
- **Case 2:**
  - Count occurrences of words:
    - `words(doc.txt) | sort | uniq -c`
    - where `words` takes a file and outputs the words in it, one per line.
- **Case 2 captures the essence of MapReduce:**
  - Great thing is that it is naturally parallelizable.

MapReduce: Overview

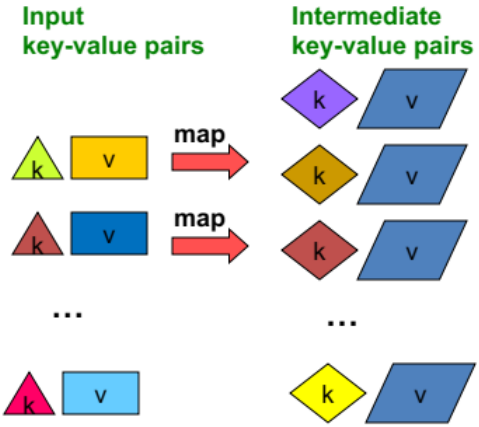
- Sequentially read a lot of data
- **Map:**
  - Extract something you care about $<$k, v $>$
- **Group by key:** Sort and Shuffle
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result

Outline stays the same, Map and Reduce change to fit the problem.

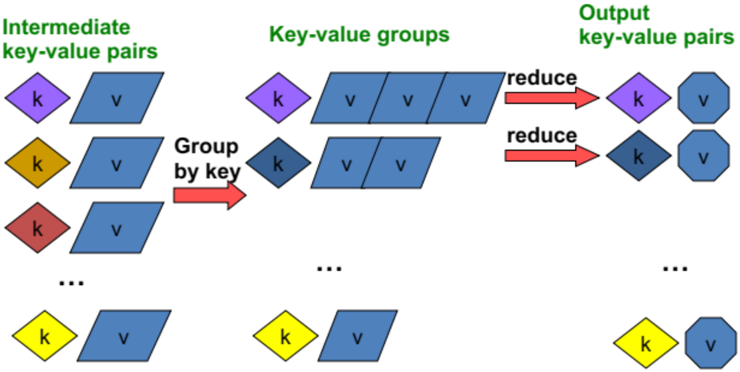
MapReduce: Overview (cont'd)

- **Input:** a set of key-value pairs
- Programmer specifies two methods:
  - **Map(k, v)** $\rightarrow < k', v' >^*$:
    - Takes a key-value pair and outputs a set of key-value pairs.
    - E.g., key is the filename, value is a single line in the file.
    - There is one Map call for every (k, v) pair.
  - **Reduce**(k', $< v' >^*$) $\rightarrow < k', v''^{>} *$
    - All values *v'* with the same key *k'* are reduced together and processed in *v'* order.
    - There is one Reduce function call per unique key *k'*.

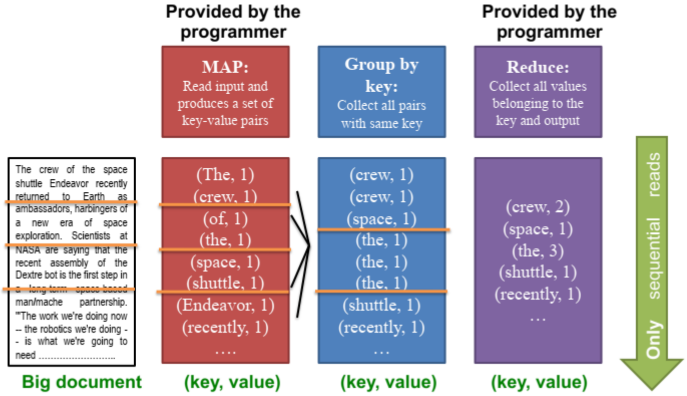
MapReduce: The Map Step



MapReduce: The Reduce Step



MapReduce: Word Counting


The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. "'The work we're doing now -- the robotics we're doing -- is what we're going to need ....

Word Count Using MapReduce

map(key, value):

- key: document name; value: text of the document
- for each word *w* in value:
    - output(w, 1)

reduce(key, values):

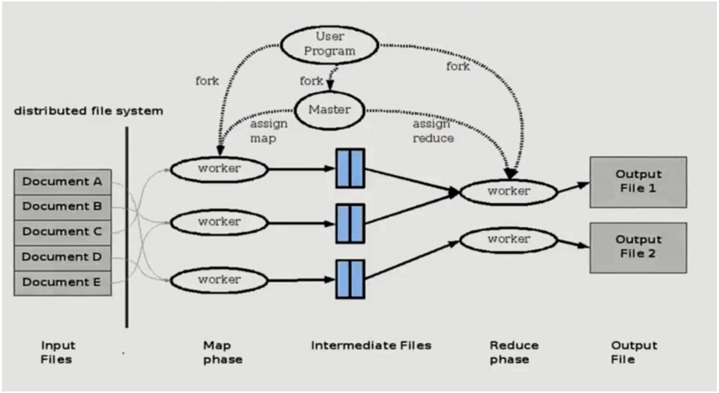
- key: a word; value: an iterator over counts
- result $= 0$
- for each count v in values:
    - result $+= v$
- output(key, result)

Map-Reduce: Environment

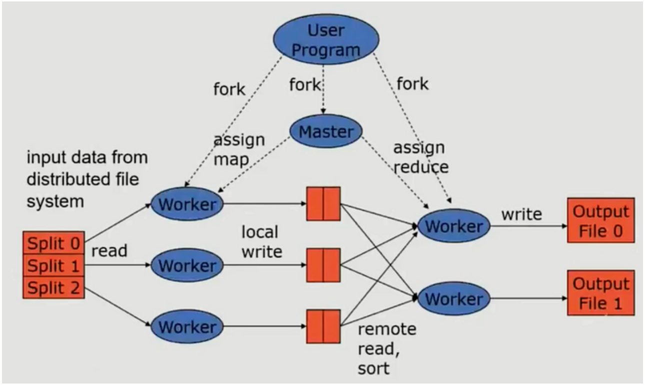
Map-Reduce environment takes care of:

- Partitioning the input data.
- Scheduling the program's execution across a set of machines.
- Performing the group by key step.
- Handling machine failures.
- Managing required inter-machine communication.

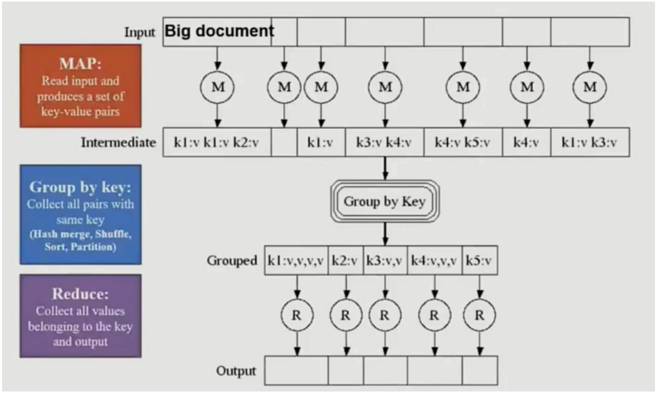
MapReduce: Execution overview



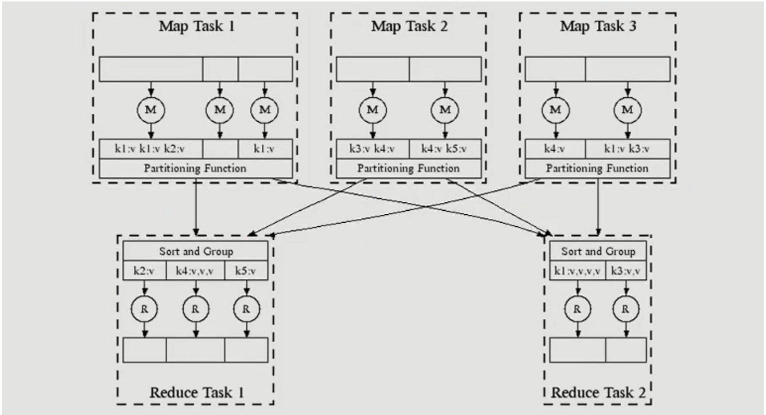
Distributed Execution overview



MapReduce: A diagram



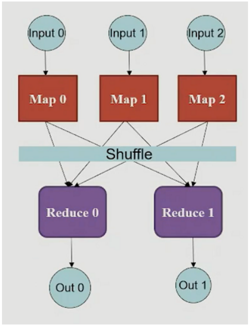
MapReduce: A Parallel



All phases are distributed with many tasks doing the work

MapReduce

- **Programmer specifies:**
  - Map and Reduce and input files
- **Workflow:**
  - Read inputs as a set of key-value pairs
  - **Map** transforms input `kv`-pairs into a new set of `k'v'`-pairs
  - Sorts & Shuffles the `kv'`-pairs to output nodes
  - All `k'v'`-pairs with a given `k'` are sent to the same **reduce**
  - **Reduce** processes all `k'v'`-pairs grouped by key into new `k"v"`-pairs
  - Write the resulting pairs to files
- All phases are distributed with many tasks doing the work



Data Flow

- Input and final output are stored on a distributed file system (FS):
    - Scheduler tries to schedule **map** tasks "close" to physical storage location of input data.
- **Intermediate results** are stored on local FS of Map and Reduce workers.
- Output is often input to another MapReduce task.

Coordination: Master

- Master node takes care of coordination:
  - Task status: (idle, in-progress, completed)
  - Idle tasks get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its R intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures
- **How to determine number of workers in Hadoop?**
  - `mapred.map.tasks = m`
  - `mapred.reduce.tasks = r`

Dealing with Failures

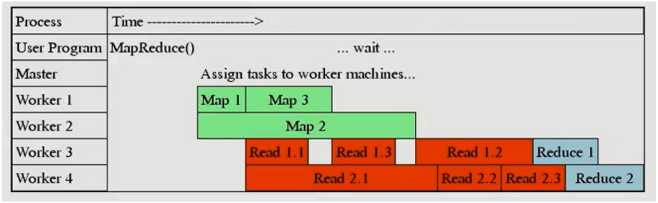
- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle.
  - Reduce workers are notified when task is rescheduled on another worker.
- **Reduce worker failure**
  - Only in-progress tasks are reset to idle.
  - Reduce task is restarted.
- **Master failure**
  - MapReduce task is aborted and client is notified.

How many Map and Reduce jobs?

- M map tasks, R reduce tasks
- **Rule of a thumb:**
  - Make M much larger than the number of nodes in the cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures.
- **Usually R is smaller than M. Why?**
  - Because output is spread across R files

Task Granularity & Pipelining

- **Fine granularity tasks:** map tasks $\gg$ machines
  - Minimizes time for fault recovery
  - Can do pipeline shuffling with map execution
  - Better dynamic load balancing

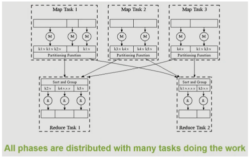


Refinements: Backup Tasks

- **Problem:**
  - Slow workers significantly lengthen the job completion time:
    - Other jobs on the machine
    - Bad disks
    - Weird things
- **Solution:**
  - Near the end of the phase, spawn backup copies of tasks:
  - Whichever one finishes first “wins.”
- **Effect:**
  - Dramatically shortens job completion time.

Refinement: Combiners

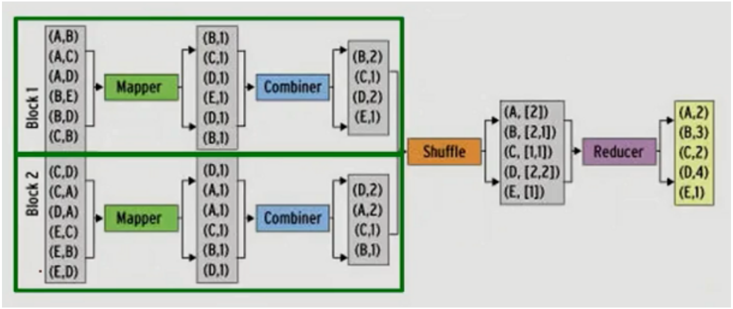
- Often a Map task will produce many pairs of the form $(k, v_1), (k, v_2), \ldots$ for the same key k
  - E.g., popular words in the word count example
- Can save network time by pre-aggregating values in the mapper:
  - `combine(k, list(v))` $\rightarrow$ `v2`
  - Combiner is usually the same as the reduce function
- Works only if the reduce function is commutative and associative:
  - Commutative: $a + b = b + a$
  - Associative: $a + (b + c) = (a + b) + c$



Refinement: Combiners (cont'd)

Back to our word counting example:

- Combiner combines the values of all keys of a single mapper (single machine):



- Much less data needs to be copied and shuffled!

Refinement: Partition Function

- **Want to control how keys get partitioned**
  - Inputs to map tasks are created by contiguous splits of input file.
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker.
- **System uses a default partition function:**
  - `hash(key) mod R` $\rightarrow \{0, 1, ..., R\text{-}1\}$
- **Sometimes useful to override the hash function:**
  - E.g., `hash(hostname(URL)) mod R` ensures URLs from a host end up in the same output file.
  - Sometimes the key is not a string and you need to cast it to string first.
  - Sometimes you need a better hashing function than the built-in `hash()`. What makes a good hashing function?

Problems Suited for Map-Reduce

Example: Host Size

- Suppose we have a large web corpus
- Look at the metadata file
    - Lines of the form: `(URL, size, date, ...)`
- For each host, find the total number of bytes:
    - That is, the sum of the page sizes for all URLs from that particular host
- **Other examples:**
    - Link analysis and graph processing
    - Machine Learning algorithms

Example: Language Model

- **Statistical machine translation:**
  - Need to count number of times every 5-word sequence (5-gram) occurs in a large corpus of documents
- **Very easy with MapReduce:**
  - **Map:** Extract (5-word sequence, count) from document
  - **Reduce:** Combine the counts

Example: Join By Map-Reduce

- Compute the natural join `R(A,B)` $\bowtie$ `S(B,C)`
  - `R` and `S` are each stored in files
  - Tuples are pairs `(a,b)` or `(b,c)`

A	B
a1	b1
a2	b1
a3	b2
a4	b3

$\bowtie$

B	C
b2	c1
b2	c2
b3	c3

$\equiv$

A	C
a3	c1
a3	c2
a4	c3

Map-Reduce Join

- Use a hash function h from B-values to $1 \ldots k$
- A Map process turns:
  - Each input tuple $R(a, b)$ into key-value pair $(b, (a, R))$
  - Each input tuple $S(b, c)$ into $(b, (c, S))$
- Map processes send each key-value pair with key b to Reduce process $h(b)$
- Each Reduce process matches all the pairs $(b, (a, R))$ with all $(b, (c, S))$ and outputs $(a, (b, c))$

Note

This is a tough process and requires you to know Java to implement your mapper and reducer. We'll see how to easily do this via Hive later.

MapReduce vs. Parallel Databases

- **MapReduce widely used for parallel processing**
  - Google, Yahoo, and 100's of other companies
  - Example uses: compute PageRank, build keyword indices, do data analysis of web click logs, ...
- **Database people say:** but parallel databases have been doing this for decades
- **MapReduce people say:**
  - We operate at scales of 1000's of machines
  - We handle failures seamlessly
  - We allow procedural code in map and reduce and allow data of any type

Implementations

- **Google MapReduce**
  - Not available outside Google
- **Hadoop**
  - An open-source implementation in Java.
  - Originally developed at Yahoo!
  - Uses HDFS for stable storage.
  - In 2010, Facebook claimed that they had the largest Hadoop cluster in the world with 21 PB of storage.
  - In June 2012, it had grown to 100 PB and was growing by roughly half a PB per day.
  - As of 2013, Hadoop adoption had become widespread: more than half of the Fortune 50 companies used Hadoop.
  - Download: `https://hadoop.apache.org/`
- **Teradata Aster**
  - Cluster-optimized SQL Database that also implements MapReduce

Quiz

Question

You are given two tables: Orders(OrderID, CustomerID, Amount) and Customers(CustomerID, Name). Write a MapReduce algorithm to calculate the total Amount spent by each customer (i.e., sum of Amount from the Orders table) and output the CustomerID along with the Name and the total Amount they spent.
Describe the steps for the map and reduce phases for this operation.

OrderID	**CustomerID**	**Amount**
1	1	150.00
2	2	200.00
3	1	350.00
4	3	450.00
5	2	100.00

CustomerID	**Name**
1	Peyman
2	Mehdi
3	Milad

SQL Solution

SQL Query:

SELECT Customers.CustomerID, Customers.Name, SUM(Orders.Amount)

FROM Orders JOIN Customers ON Orders.CustomerID = Customers.CustomerI

GROUP BY Customers.CustomerID;

Sample Output:

CustomerID	**Name**	**Total Amount**
1	Peyman	500.00
2	Mehdi	300.00
3	Milad	450.00

MapReduce Solution

MapReduce Process:

- Map Phase:
  - Emit (CustomerID, Amount) for each order from the ‘Orders‘ table.
  - Emit (CustomerID, Name) for each customer from the ‘Customers‘ table.
- Shuffle and Sort Phase:
  - Group records by ‘CustomerID‘.
- Reduce Phase:
  - Sum the ‘Amount‘ values for each ‘CustomerID‘.
  - Retrieve the ‘Name‘ for each ‘CustomerID‘.
  - Output the result as (CustomerID, Name, Total Amount).

Apache Hive

Hadoop Usage @ Facebook

- Types of Applications:
  - Summarization
    - Eg: Daily/Weekly aggregations of impression/click counts
  - Ad hoc Analysis
    - Eg: how many group admins broken down by state/country
  - Data Mining (Assembling training data)
    - Eg: User Engagement as a function of user attributes

Twitter
6000
Tweets Every Second
350,000
Tweets Per Minute
50 Million
Tweets Per Day
facebook
1.55 Billion
Active users Per Month
1.39 Billion
Mobile active User
Every minute on Facebook
510
Comments Posted
293,000
Statuses Are Updated
136,000 Photos Are Uploaded.

Motivation for Another Data Warehousing System

- Problem: Data, data and more data
  - Several TBs/PBs of data everyday
- The Hadoop Experiment:
  - Uses Hadoop File System (HDFS)
  - Scalable/Available
- Problem
  - Lacked Expressiveness
  - Map-Reduce hard to program
- Solution: Facebook created **Hive** in 2007 and made it open source in 2008.

What is Hive?

- A system for querying and managing structured data and unstructured data (as if it were structured) built on top of Hadoop data.
  - Uses Map-Reduce for execution
  - HDFS for Storage
  - Note that Hadoop is not a database.
- Key Building Principles
  - SQL as a familiar data warehousing tool
  - Extensibility
    - Pluggable map/reduce scripts in the language of your choice
    - Rich and User Defined Data Types and User Defined Functions
  - Interoperability (Extensible Framework to support different file and data formats)
  - Performance

Type System

- **Primitive types**
  - Integers: TINYINT, SMALLINT, INT, BIGINT.
  - Boolean: BOOLEAN.
  - Floating point numbers: FLOAT, DOUBLE.
  - String: STRING.
- **Complex types**
  - Recursively build up using Composition/Maps/Lists
  - Structs: {`a INT; b INT`}.
  - Maps: `M`

    “group”

    .
  - Arrays: `['a', 'b', 'c']`
    - A[1] returns ‘b’.

Data Model: Tables

- Analogous to tables in relational DBs.
- Each table has corresponding directory in HDFS.
- Example
  - Table name: `my_table`
  - HDFS directory:
    - `/user/hive/warehouse/my_table`
  - HQL:

```
CREATE TABLE t1(
ds string,
ctry float,
li list<map<string, struct<p1:int, p2:int>>
);
```

Data Model: Partitions (1)

- Partitions
  - Analogous to dense indexes on partition columns
  - Nested sub-directories in HDFS for each combination of partition column values
  - Allows users to efficiently retrieve rows
- Example
  - Partition columns: `ds`, `ctry`
  - HDFS for `ds=20120410`, `ctry=US`
    - /user/hive/warehouse/my_table/ds=20120410/ctry=US
  - HDFS for `ds=20120410`, `ctry=IR`
    - /user/hive/warehouse/my_table/ds=20120410/ctry=IR

Data Model: Partitions (2)

- Creating partitions

```
CREATE TABLE test_part(c1  string, c2  string)
PARTITIONED  BY (ds string, hr int);
```

- Add a new partition:

```
INSERT OVERWRITE TABLE test_part
PARTITION(ds='2009-01-01', hr=12)
SELECT * FROM t;
```

OR

```
ALTER TABLE test_part
ADD PARTITION(ds='2009-02-02', hr=11);
```

Data Model: Partitions (3)

```
SELECT * FROM test_part
    WHERE ds='2009-01-01';
```

will only scan all the files within the /user/hive/warehouse/test_part/ds=2009-01-01 directory.

```
SELECT * FROM test_part
    WHERE ds='2009-02-02' AND hr=11;
```

will only scan all the files within the /user/hive/warehouse/test_part/ds=2009-02-02/hr=11 directory.

Data Model: Buckets

- Split data based on hash of a column – mainly for parallelism
- Data in each partition may in turn be divided into Buckets based on the value of a hash function of some column of a table.
- Example
  - Bucket column: user into 32 buckets
  - HDFS file for user hash 0:
    - /user/hive/warehouse/my_table/ds=20120410/cntr=US/part-00000
  - HDFS file for user hash bucket 20:
    - /user/hive/warehouse/my_table/ds=20120410/cntr=US/part-00020

Data Model: External Tables

- Point to existing data directories in HDFS
- Can create table and partitions
- Data is assumed to be in Hive-compatible format
- Dropping external table drops only the metadata
- Example: create external table

```
CREATE EXTERNAL TABLE test_extern(c1  string, c2  int)
LOCATION '/user/mytables/mydata';
```

Serialization/Deserialization

- Generic (De)Serialization interface *SerDe*.
- Uses LazySerDe.
- Flexible interface to translate unstructured data into structured data.
- Designed to read data separated by different delimiter characters.
- The SerDes are located in hive_contrib.jar
- You can write your own SerDe:

```
add jar /jars/myformat.jar
CREATE TABLE t2 ROW FORMAT SERDE 'com.myformat.MySerDe'
```

Hive File Formats

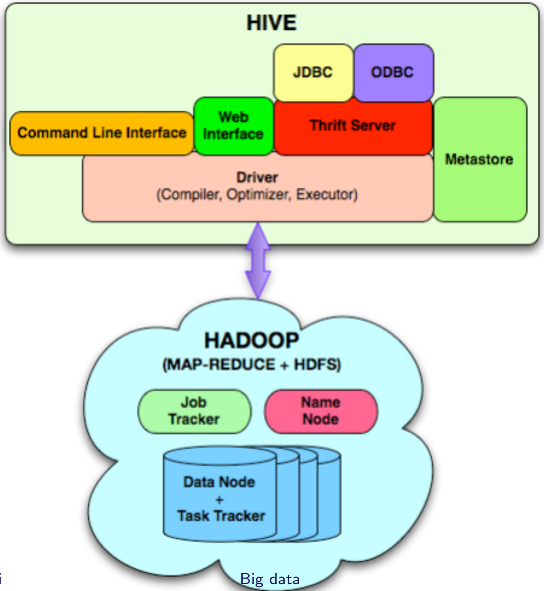
- Hive lets users store different file formats.
- Helps in performance improvements.
- Supports compressed file format for on-the-fly compression:
  - https://cwiki.apache.org/confluence/display/Hive/CompressedStorage
- SQL Example:

```
CREATE TABLE dest1(key INT, value STRING)
STORED AS

INPUTFORMAT 'org.apache.hadoop.mapred.SequenceFileInputFormat'

OUTPUTFORMAT 'org.apache.hadoop.mapred.SequenceFileOutputFormat'
```

System Architecture and Components (1)

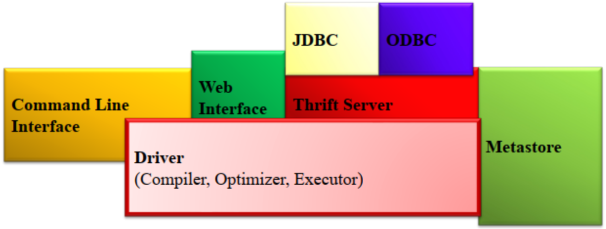


System Architecture and Components: Metastore

- The component that stores the system catalog and meta data about tables, columns, partitions, etc.
- Stores Table/Partition properties:
  - Table schema and SerDe library
  - Table Location on HDFS
  - Logical Partitioning keys and types
  - Other information
- Thrift API
  - Current clients in Php (Web Interface), Python (old CLI), Java (Query Engine and CLI), Perl (Tests)
  - Stored on a traditional RDBMS:
    - DataNucleus for open source Hive.
    - MySQL for internal Facebook Hive.

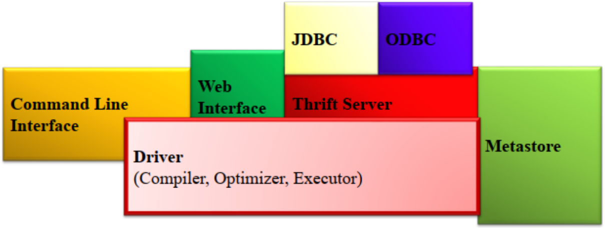
System Architecture and Components: Driver

- The component that manages the lifecycle of a HiveQL statement as it moves through Hive.
- The driver also maintains a session handle and any session statistics.



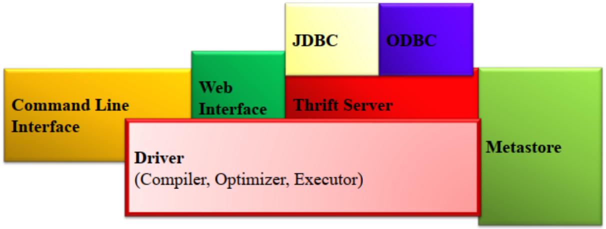
System Architecture and Components: Query Compiler

- The component that compiles HiveQL into a directed acyclic graph of map/reduce tasks.



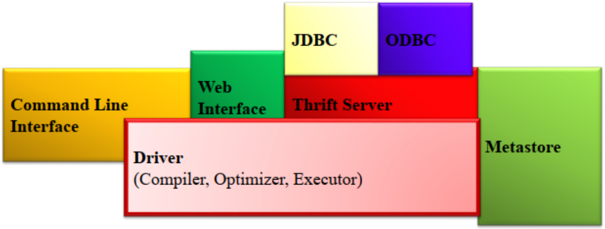
System Architecture and Components: Optimizer

- Consists of a chain of transformations such that the operator DAG resulting from one transformation is passed as input to the next transformation.
- Performs tasks like Column Pruning, Partition Pruning, Repartitioning of Data.



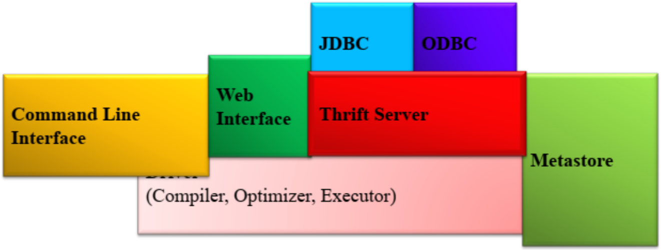
System Architecture and Components: Execution Engine

- The component that executes the tasks produced by the compiler in proper dependency order.
- The execution engine interacts with the underlying Hadoop instance.



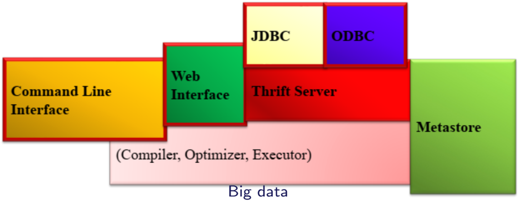
System Architecture and Components: HiveServer

- The component that provides a **Thrift interface** and a JDBC/ODBC server
- Provides a way of integrating Hive with other applications.



System Architecture and Components: Client Components

- Client component like Command Line Interface (CLI), the web UI and JDBC/ODBC driver.
- Hive CLI:
  - **Data definition language (DDL):**
    - Create table/drop table/rename table
    - Alter table add column
  - **Browsing:**
    - Show tables
    - Describe table
    - Cat table
  - Loading Data
  - Queries



Hive Query Language (HQL)

- **Philosophy**
  - SQL like constructs + Hadoop Streaming
- **Basic SQL**
  - From clause sub-query
  - ANSI JOIN (equi-join only)
  - Multi-Table insert
  - Multi group-by
  - Sampling
  - Objects Traversal
- **Extensibility**
  - Pluggable Map-reduce scripts using `TRANSFORM`
- **Output** of these operators can be to mappers/reducers, another Hive Table, HDFS files, and local files.

Insertion

- Insertion

```
INSERT OVERWRITE TABLE t1
SELECT * FROM t2;
```

- INSERT INTO will append to the table or partition, keeping the existing data intact.
  - It is only available starting in version 0.8.
- Insertion into external sources:

```
INSERT OVERWRITE TABLE sample1 '/tmp/hdfs_out'
SELECT * FROM sample WHERE ds='2012-02-24';

INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out'
SELECT * FROM sample WHERE ds='2012-02-24';

INSERT OVERWRITE LOCAL DIRECTORY '/tmp/hive-sample-out

SELECT * FROM sample;
```

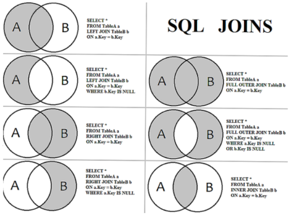
Joins

- **Joins**

```
FROM page_view pv
JOIN user u ON (pv.userid =
u.id)
INSERT INTO TABLE pv_users
SELECT pv.*, u.gender, u.age
WHERE pv.date = 2008-03-03;
```

- **Outer Joins**

```
FROM page_view pv
FULL OUTER JOIN user u ON
(pv.userid = u.id)
INSERT INTO TABLE pv_users
SELECT pv.*, u.gender, u.age
WHERE pv.date = 2008-03-03;
```



```
SELECT *
FROM TableA a
LEFT JOIN TableB b
ON a.Key = b.Key
```

```
SELECT *
FROM TableA a
LEFT JOIN TableB b
ON a.Key = b.Key
WHERE b.Key IS NULL
```

```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
ON a.Key = b.Key
```

```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
ON a.Key = b.Key
WHERE a.Key IS NULL
```

```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
ON a.Key = b.Key
```

```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
ON a.Key = b.Key
WHERE a.Key IS NULL
OR b.Key IS NULL
```

```
SELECT *
FROM TableA a
INNER JOIN TableB b
ON a.Key = b.Key
```

Aggregations and Multi-Table Inserts

```
FROM pv_users
INSERT INTO TABLE pv_gender_uu
SELECT
pv_users.gender,
count(DISTINCT pv_users.userid)
GROUP BY(pv_users.gender)

INSERT INTO TABLE pv_ip_uu
SELECT
pv_users.ip,
count(DISTINCT pv_users.id)
GROUP BY(pv_users.ip);
```

Running Custom Map/Reduce Scripts

```
FROM (
FROM pv_users
SELECT TRANSFORM (pv_users.userid, pv_users.date)
USING 'map_script' AS (dt, uid)
CLUSTER BY (dt) ) map
INSERT INTO TABLE pv_users_reduced
SELECT TRANSFORM (map.dt, map.uid) USING 'reduce_script'
AS (date, cnt);
```

Inserts into Files, Tables and Local Files

```
FROM pv_users
INSERT INTO TABLE pv_gender_sum
SELECT pv_users.gender, count_distinct(pv_users.userid)
GROUP BY(pv_users.gender)

INSERT INTO DIRECTORY ‘/user/facebook/tmp/pv_age_sum.dir’
SELECT pv_users.age, count_distinct(pv_users.userid)
GROUP BY(pv_users.age)

INSERT INTO LOCAL DIRECTORY ‘/home/me/pv_age_sum.dir’
FIELDS TERMINATED BY ‘,’ LINES TERMINATED BY \013
SELECT pv_users.age, count_distinct(pv_users.userid)
GROUP BY(pv_users.age);
```

Conclusion

- **Pros**
    - Familiar language (i.e., HiveQL) with good documentation: Hive Language Manual
    - No Java experience is necessary.
    - Architecture is well explained.
    - Good tool chain.
- **Cons**
    - Accepts only a subset of ANSI SQL queries.
    - Fragile (external tables)
    - Slow.

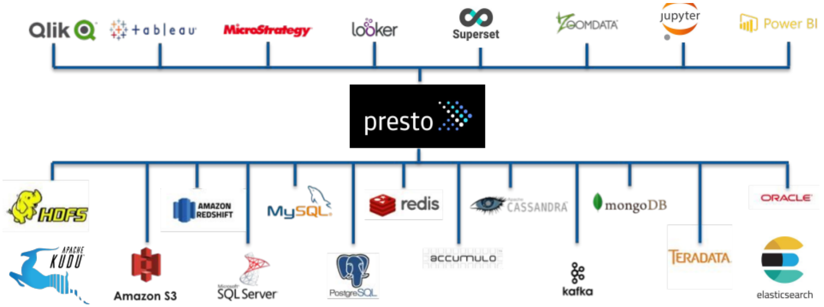
Presto

What is Presto?

- A Massively Parallel Processing *(MPP)* SQL query engine
    - Started by Facebook as a successor of HiveMapReduce in 2012
    - Open-sourced in 2013
- Typically run on top of a Hadoop cluster
- Designed and written from the ground up for interactive analytical queries
- Scales to the sizes needed by organizations like Facebook
- Query data where it lives
- Hadoop Distribution agnostic
- Extensible

Presto: SQL-on-Anything

- Deploy Anywhere, Query Anything



Presto = Performance

- Horizontal scale out
- Query execution is pipelined throughout memory
- Vectorized columnar processing
- Optimized data source readers
- Presto is written in highly tuned Java
  - Efficient in-memory data structures
  - Very careful coding of inner loops
  - Bytecode generation

Presto Users

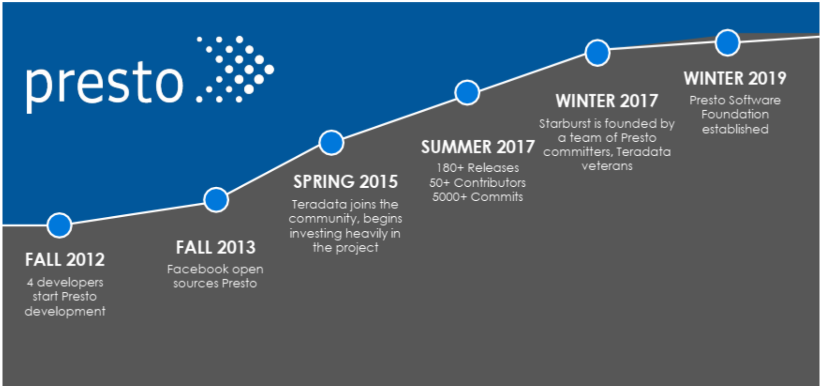
- Facebook, META

Checkr
amazon
YAHOO! JAPAN
JD.COM 京东
FreeWheel
airbnb
mercado Libre
Alibaba Group
allegro
jampp
TiVo
Marin SOFTWARE
NELE INTERNATIONAL
SmartNews
Eventbrite
zomato
TREASURE DATA
looker
LinkedIn
Supership
zuora
WB
comcast
Starburst
UBER
glassdoor
GROUPON
Walmart
WiX
FINRA
Bloomberg
去哪儿 Qunar.Com
Grab
Zillow GROUP
lyft
twitter
Qubole ELEMENTAL TO BIG DATA
GREE
cogo labs
SCHIBSTED MEDIA GROUP
AdRoll
trulia
shopify
Atlassian
slack
SHAZAM
Pinterest
Akamai
openspan
NASDAQ
Dropbox
CUEBIQ
NETFLIX
LINE

Presto in production

- **Facebook:** 1000s of nodes, HDFS (ORC, RCFile), sharded MySQL, 1000s of users
- **Uber:** 800+ nodes (2 clusters on premises) with 200K+ queries daily over HDFS (Parquet/ORC)
- **Twitter:** 800+ nodes (several clusters on premises) for HDFS (Parquet)
- **LinkedIn:** 350+ nodes (2 clusters on premises), 40K+ queries daily over HDFS (ORC), 600+ users
- **Netflix:** 250+ nodes in AWS, 40+ PB in S3 (Parquet)
- **Lyft:** 200+ nodes in AWS, 20K+ queries daily, 20+ PBs in Parquet
- **Yahoo! Japan:** 200+ nodes (4 clusters on premises) for HDFS (ORC), ObjectStore, and Cassandra
- **FINRA:** 120+ nodes in AWS, 4PB in S3 (ORC), 200+ users

Presto History

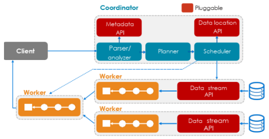

presto
FALL 2012
4 developers start Presto development
FALL 2013
Facebook open sources Presto
SPRING 2015
Teradata joins the community, begins investing heavily in the project
SUMMER 2017
180+ Releases
50+ Contributors
5000+ Commits
WINTER 2017
Starburst is founded by a team of Presto committers, Teradata veterans
WINTER 2019
Presto Software Foundation established

Project History: Presto Fork

- **January 2019:**
  - Presto Software Foundation established after key developers quit Facebook.
  - Presto forks: PrestoDB from Facebook and PrestoSQL from Presto Software Foundation.
- **September 2019:** Facebook donates PrestoDB to Linux foundation, establishing the Presto Foundation.
- **December 2020:** PrestoSQL is rebranded as Trino, since Facebook (and subsequently Linux Foundation) holds a trademark on the name "Presto".
  - PrestoDB: `https://prestodb.io/`
  - Trino: `https://trino.io/`

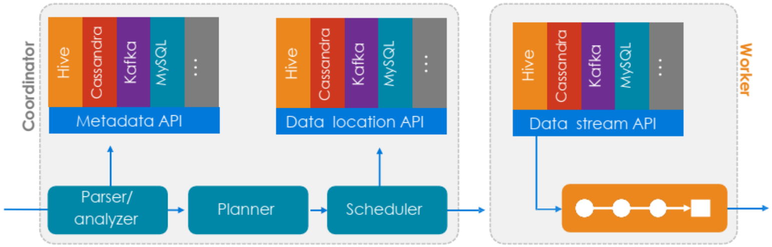
Architecture

- **Coordinator**
  - Receives a query from the client. How?
  - Analyzes, parses, plans and schedules to workers
- **Worker**
  - Executes scheduled tasks
  - Reads and writes to/from data sources
  - Shuffles data between other workers
  - Sends the final results to the client
- **Client**
  - Application
  - Presto console
- **Data source plugins**
  - Data Location API/Data Stream API
  - Custom plugins can be easily added



Presto Extensibility – Connectors

- A connector provides Presto an interface to access an arbitrary data source.
- Each connector provides a table-based abstraction over the underlying data source.
- Presto connectors are plug-ins loaded by each server at startup.



Differences between Presto and SparkQL/HiveQL

- Spark/Hive was designed for batch or long running extract, transform, load (ETL) jobs, when Presto targets on interactive queries with low-latency start up time.
  - Cluster starts on demand
  - Presto runs as a shared cluster all the time
  - You have to declare resources needed
  - Presto gives all that is available at the moment to satisfy the query
- SparkSQL currently supports JDBC driver available only for Hive tables
- Third party storage plugins in Spark
  - Centrally managed plugins by the project
  - Eventually will work with a new version
- In general the purpose of both is different and both plays an important role...

Presto vs. Hive/Spark Summary

Hive/Spark

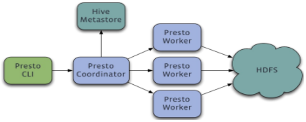
- ETL
- Machine Learning
- Scale

Presto

- Exploratory
- Interactive
- Reporting
- Audits

Presto for Hadoop in Practice

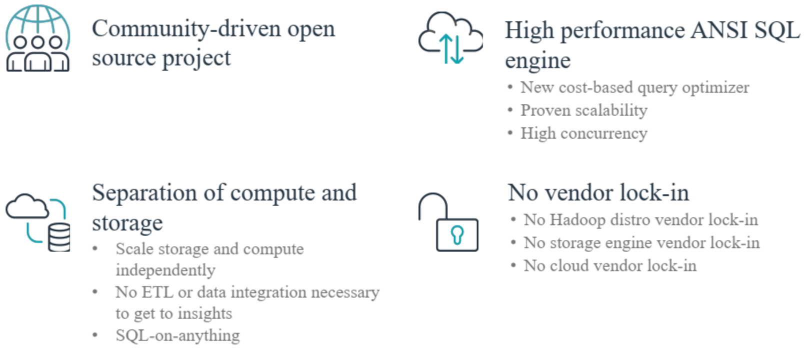
- Has dedicated connector for HDFS via Hive metastore
  - Only data mapped via Hive tables can be accessed
  - Already existing HDFS folders can be easily mapped to Hive (if schema is coherent)
- Each connector type can have multiple instances (called plugins)
  - Multiple hives (Hadoop clusters) can be accessed simultaneously
  - `SELECT * FROM hive_hadalytic.my_schema.my_table`
- Interfaces
  - Presto shell
- JDBC or ODBC for binding with applications



Beyond ANSI SQL

- Presto offers a wide variety of built-in functions including:
  - regular expression functions
    - `SELECT REGEXP_EXTRACT_ALL('1a 2b 14m', '\d+');` – [1, 2, 14]
  - lambda expressions and functions
    - `SELECT FILTER(ARRAY[5, -6, NULL, 7], x -> x > 0);` – [5, 7]
    - `SELECT TRANSFORM(ARRAY[5, 6], x -> x + 1);` – [6, 7]
  - geospatial functions
    - `SELECT c.city_id, count(*) AS trip_count FROM trips_table AS t JOIN city_table AS c ON ST_CONTAINS(c.geo_shape, ST_POINT(t.dest_lng, t.dest_lat)) WHERE t.trip_date = '2018-05-01' GROUP BY 1;`
- **Complex data types:**
  - JSON
  - ARRAY
  - MAP
  - ROW / STRUCT

Summary



Community-driven open source project

High performance ANSI SQL engine

- New cost-based query optimizer
- Proven scalability
- High concurrency

Separation of compute and storage

- Scale storage and compute independently
- No ETL or data integration necessary to get to insights
- SQL-on-anything

No vendor lock-in

- No Hadoop distro vendor lock-in
- No storage engine vendor lock-in
- No cloud vendor lock-in

Quiz

Question

- You are given a Comment table and a User table for a social network as shown below.
  - **a)** Write a HiveQL (HQL) query to calculate the average number of comments any user makes every day.
  - **b)** Assuming that these tables are stored in HDFS, which one of the following tools are better to run your query: Hive or Presto? Why?

Comment table:

CommentID	UserID	Comment	Date
8979	768	'Hello!'	'04-11-2022'
5432	998	'Happy birthday!'	'04-11-2022'
6546	768	'Whats up?'	'04-10-2022'
...			

User table:

UserID	Age	Name	Gender
768	25	Taban	Female
150	26	Matin	Male
600	29	Nazanin	Female
...			

Apache Spark