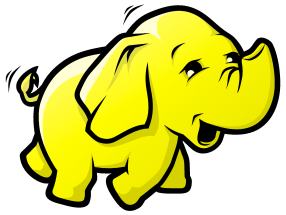
Association of Data Science and Analytics



Workshop X: Introduction to Hadoop

Ken Taylor, Agrible

October 4, 2015

Topics

- Top 10
- History
- HDFS
- MapReduce
- Pig Example
- Hive Example
- Berkeley AMPLab
- Flume
- References

Top 10 Reasons for Using Hadoop

10. It's cool
9. It's free
8. It's open source
7. It's fast (relatively), but not real-time
6. It's magnetic
5. It handles billions of records
4. It's scalable
3. It's secure
2. It's used by everybody

Yahoo, eBay, LinkedIn, Twitter, comScore, Facebook, Amazon, IBM, Hulu, Netflix

Top 10 Reasons for Using Hadoop

1. It has embraced SQL

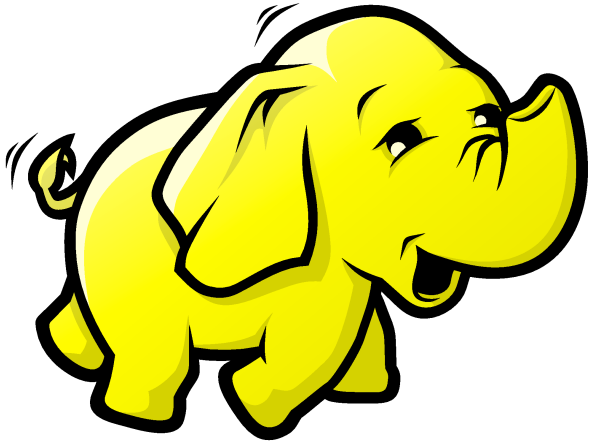
Where did Hadoop Originate?

- Google invents Google File System (GFS) and MapReduce in 2003 & 2004, respectively. Publishes papers on both.
- Based on these papers, Doug Cutting develops the Nutch Distributed File System and Nutch MapReduce as part of an open source web search engine in 2004-2006
- Nutch was migrated to Apache in 2006
- Yahoo! puts major emphasis on Hadoop development in 2006 after hiring Doug Cutting
- Yahoo! implements web-scale processing using Hadoop in early 2008

Why the Name Hadoop?

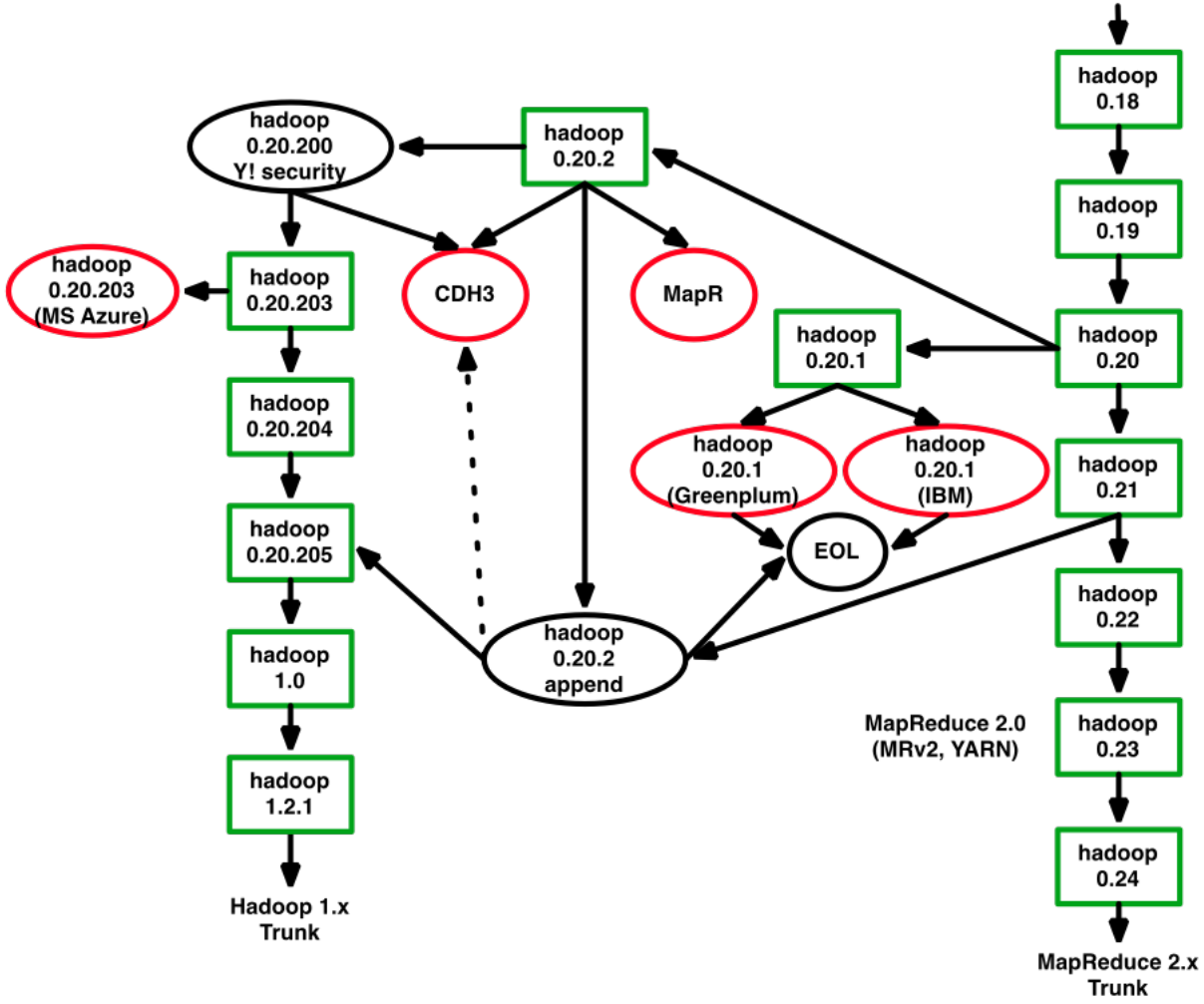
- Doug Cutting had a young son who had a toy elephant named Hadoop

Source: Doug Cutting



Source: Ken Taylor

Hadoop Releases

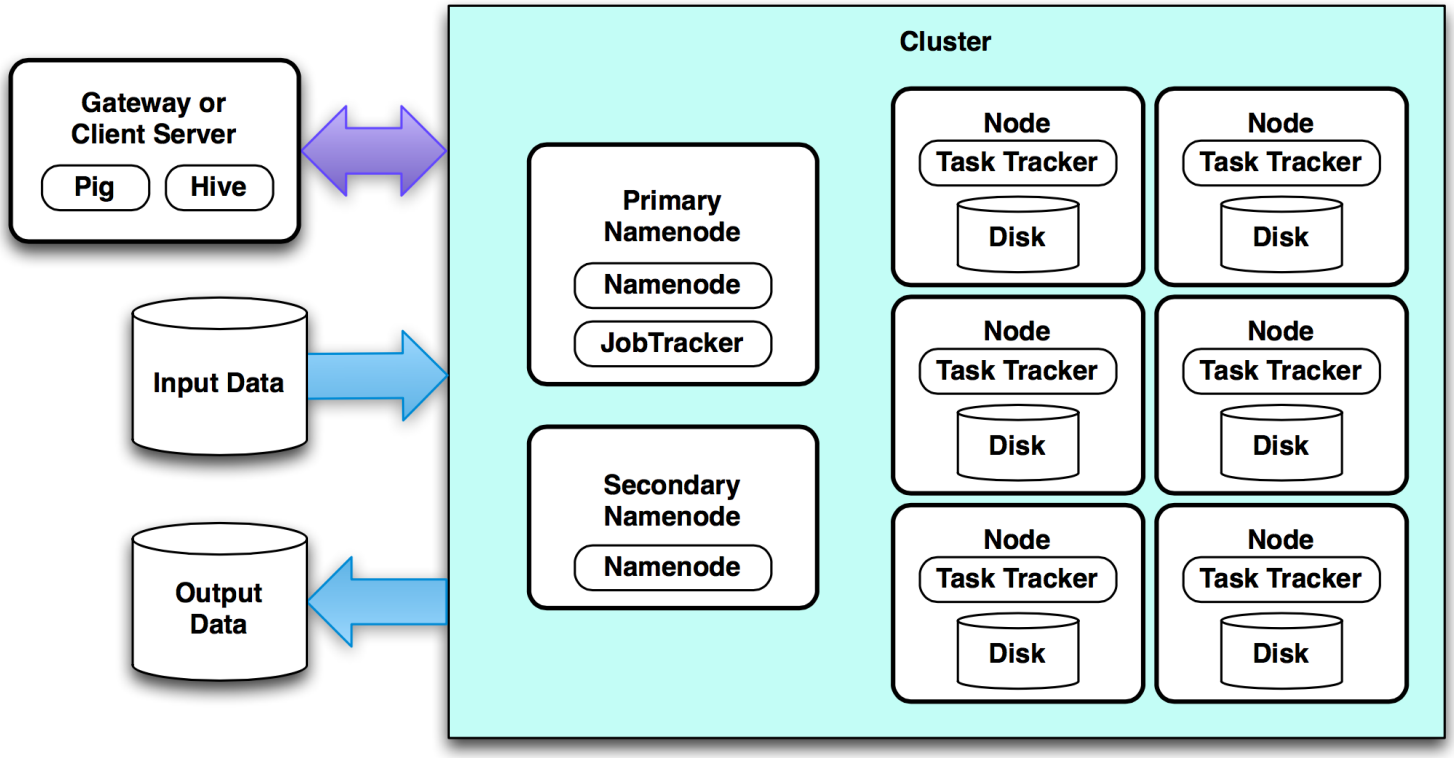


http://nosql.mypopescu.com/post/16015072005/hadoop-versions-take-2-what-you-wanted-to-know-about

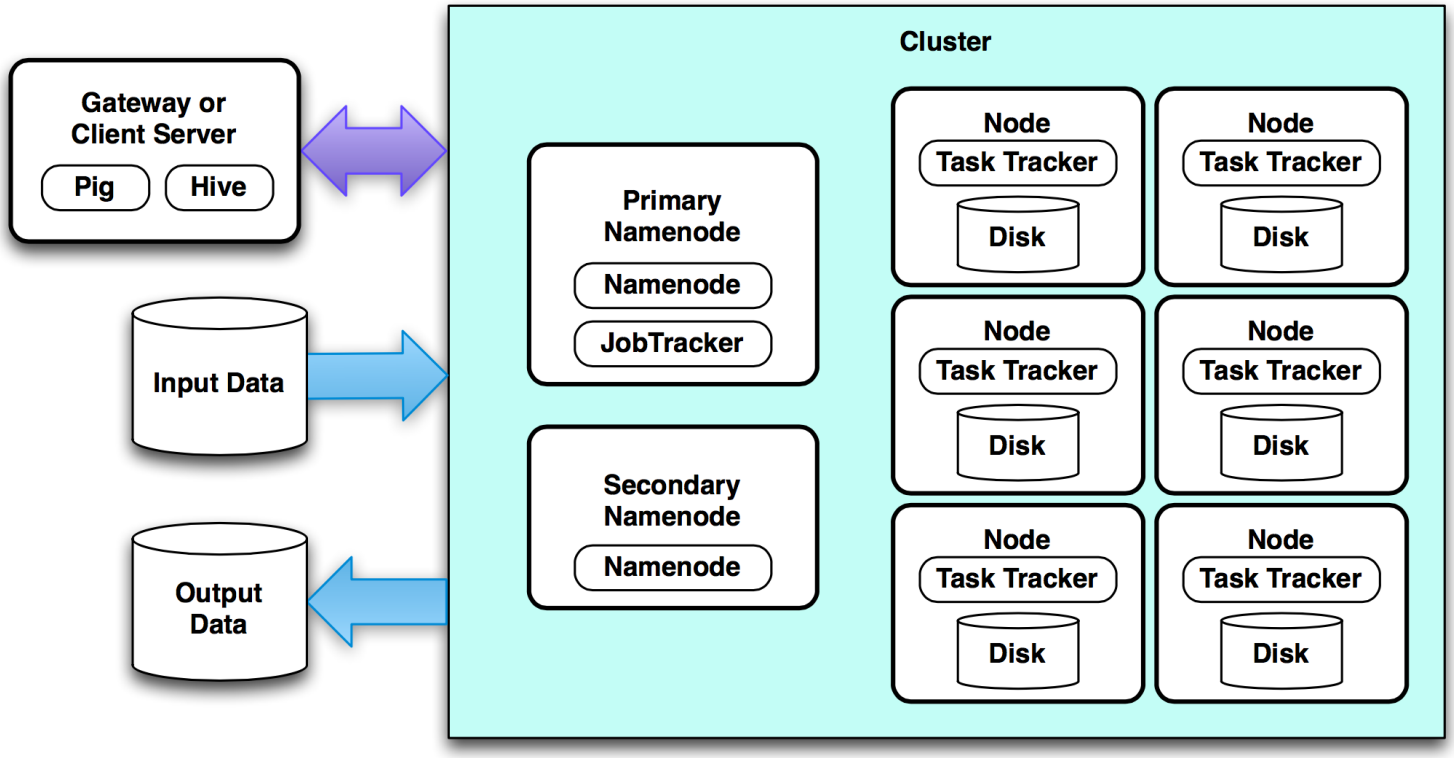
Hadoop

- Hadoop is a platform for large-scale parallel processing
- Two main components are the storage mechanism and the programming framework
- Storage mechanism is Hadoop Distributed File System (HDFS)
- Programming framework is MapReduce
- Collection of computers is called a cluster

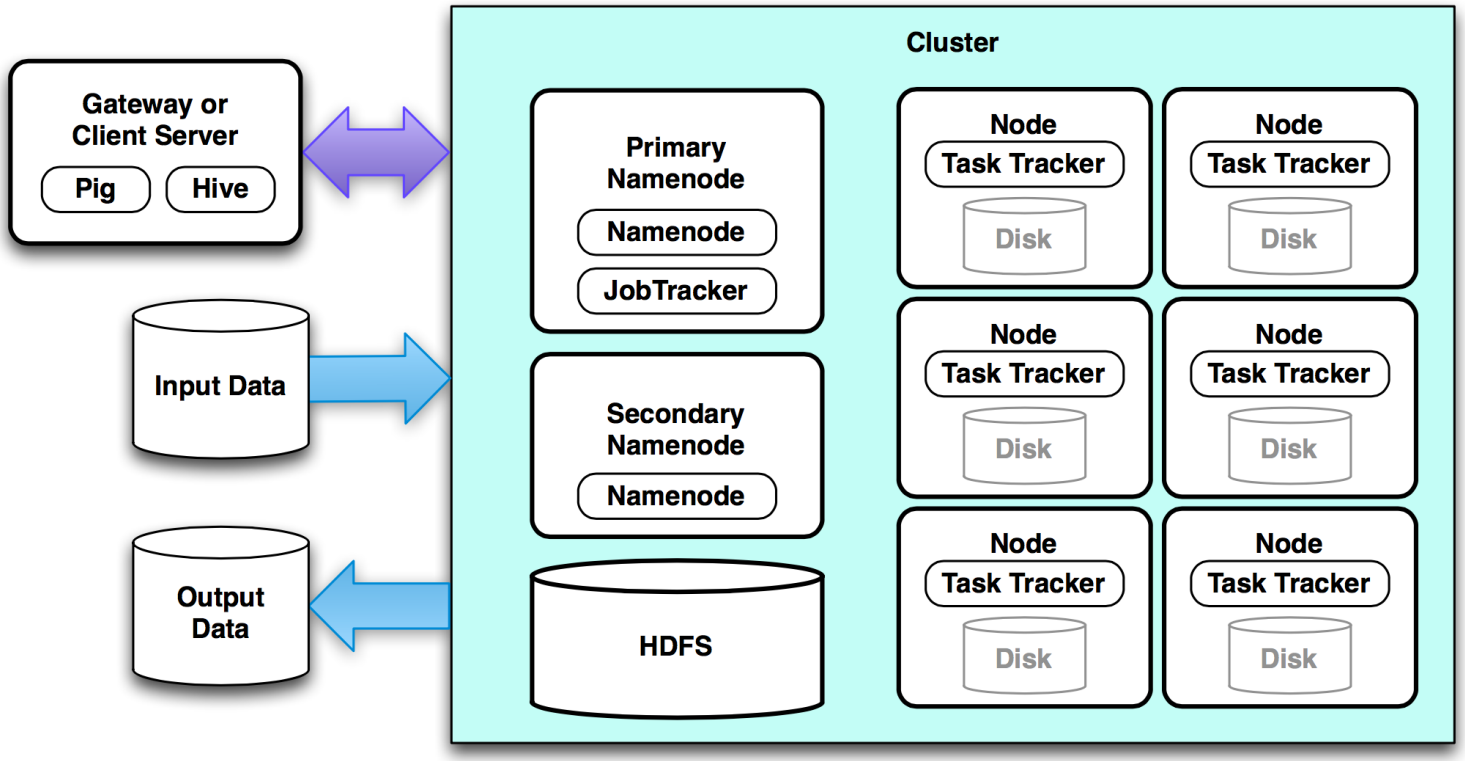
Hadoop Cluster



HDFS


Gateway or
Client Server
Pig
Hive
Input Data
Output
Data
Cluster
Primary
Namenode
Namenode
JobTracker
Secondary
Namenode
Namenode
Node
Task Tracker
Disk
Node
Task Tracker
Disk
Node
Task Tracker
Disk
Node
Task Tracker
Disk
Node
Task Tracker
Disk
Node
Task Tracker
Disk

HDFS


Gateway or
Client Server
Pig
Hive
Input Data
Output
Data
Cluster
Primary
Namenode
Namenode
JobTracker
Secondary
Namenode
Namenode
HDFS
Node
Task Tracker
Disk
Node
Task Tracker
Disk
Node
Task Tracker
Disk
Node
Task Tracker
Disk
Node
Task Tracker
Disk
Node
Task Tracker
Disk

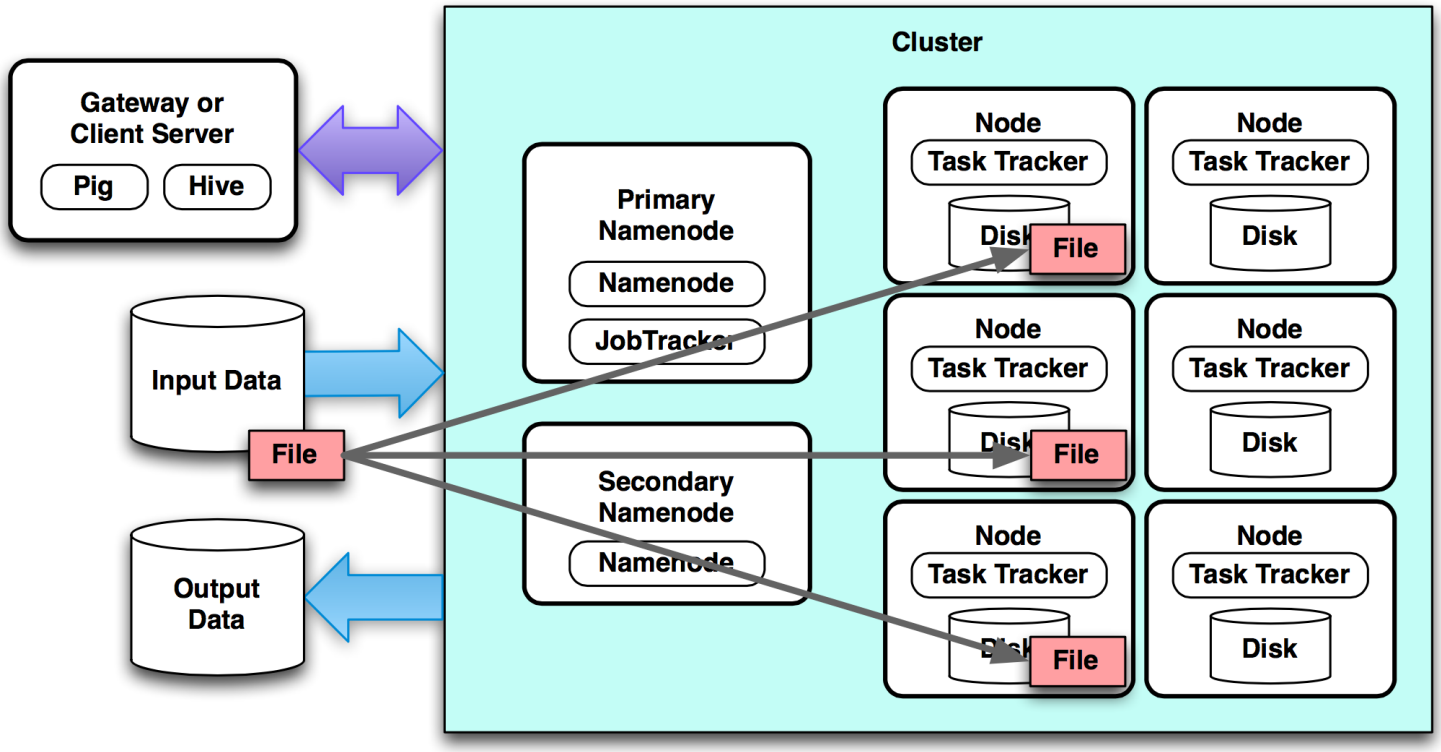
Hadoop Distributed File System

- HDFS stores files in the Hadoop cluster
- Files are duplicated, typical 3 copies are made in case a node fails
- Large block sizes are typical: 64, 128, or 256GB
- Disks read faster in sequential mode, not seek, so large block sizes benefit large-scale processing since you typically scan the entire file
- Configurable

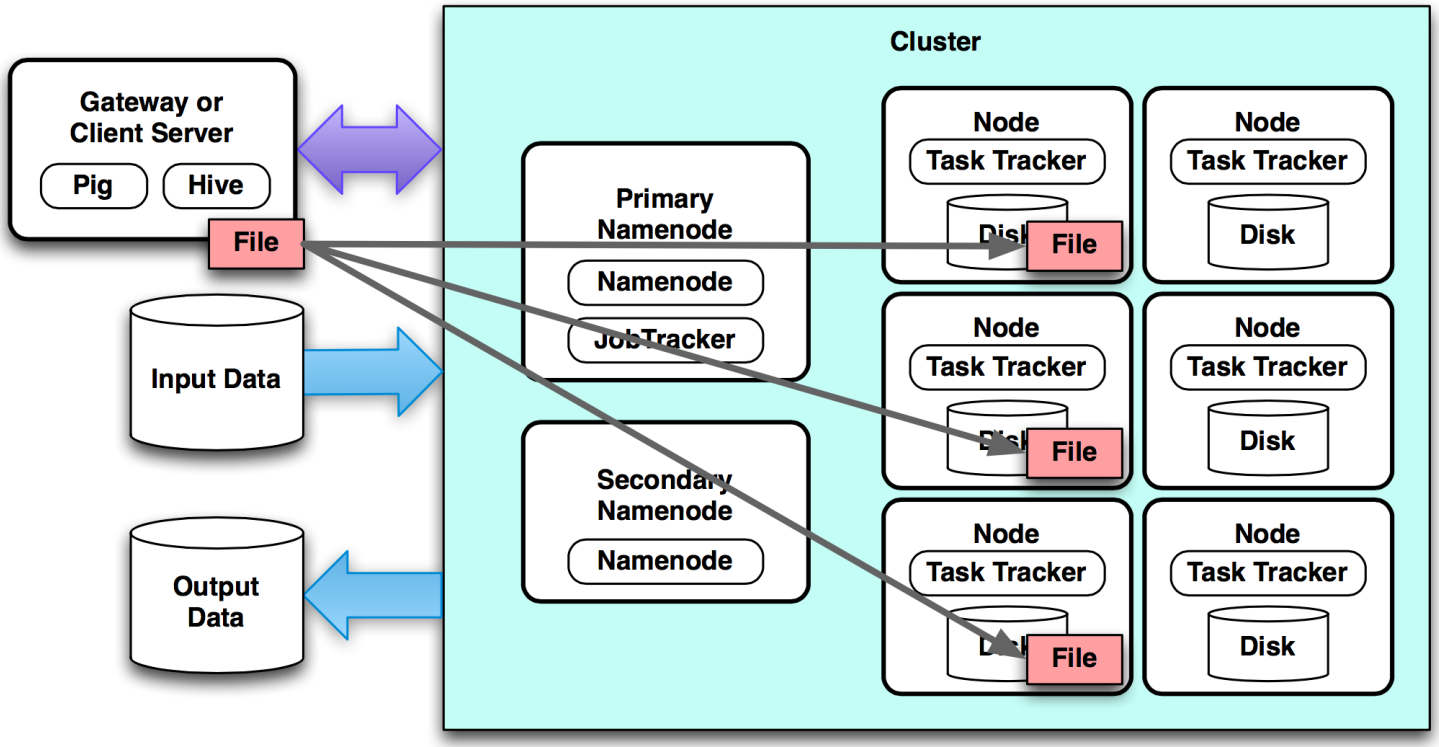
Hadoop Distributed File System

- Write-once, read-many environment
- Consider the data as a heavy object, immovable
- Move the code to the data, not data to the code
- Updates to the files are not allowed
- An update operation is a rewrite of the file

File Load


Cluster
Gateway or Client Server
Pig
Hive
Input Data
File
Output Data
Primary Namenode
Namenode
JobTracker
Secondary Namenode
Namenode
Node
Task Tracker
Disk
File
Node
Task Tracker
Disk
Node
Task Tracker
Disk
File
Node
Task Tracker
Disk
Node
Task Tracker
Disk
File
Node
Task Tracker
Disk

File Load


Cluster
Gateway or Client Server
Pig
Hive
File
Input Data
Output Data
Primary Namenode
Namenode
JobTracker
Secondary Namenode
Namenode
Node
Task Tracker
Disk
File

Hadoop Distributed File System

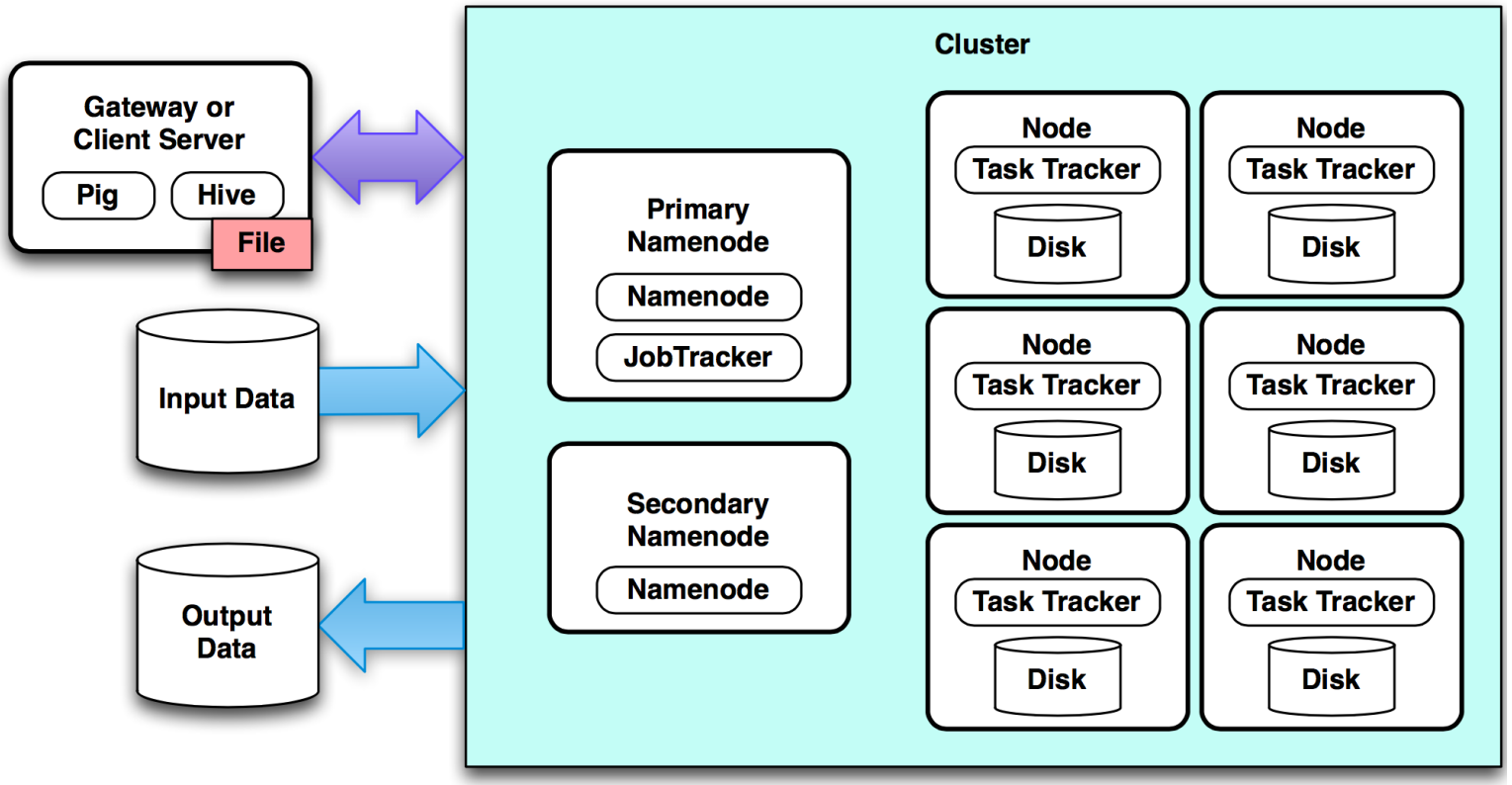
- Namenode keeps track of HDFS
- Replicates files
- Maintains location of blocks
- Maps filenames to blocks
- You never see the replication

Hadoop Distributed File System

- HDFS provides a command interface, similar to POSIX
- Since you are on the Gateway, you can view local gateway files and HDFS files

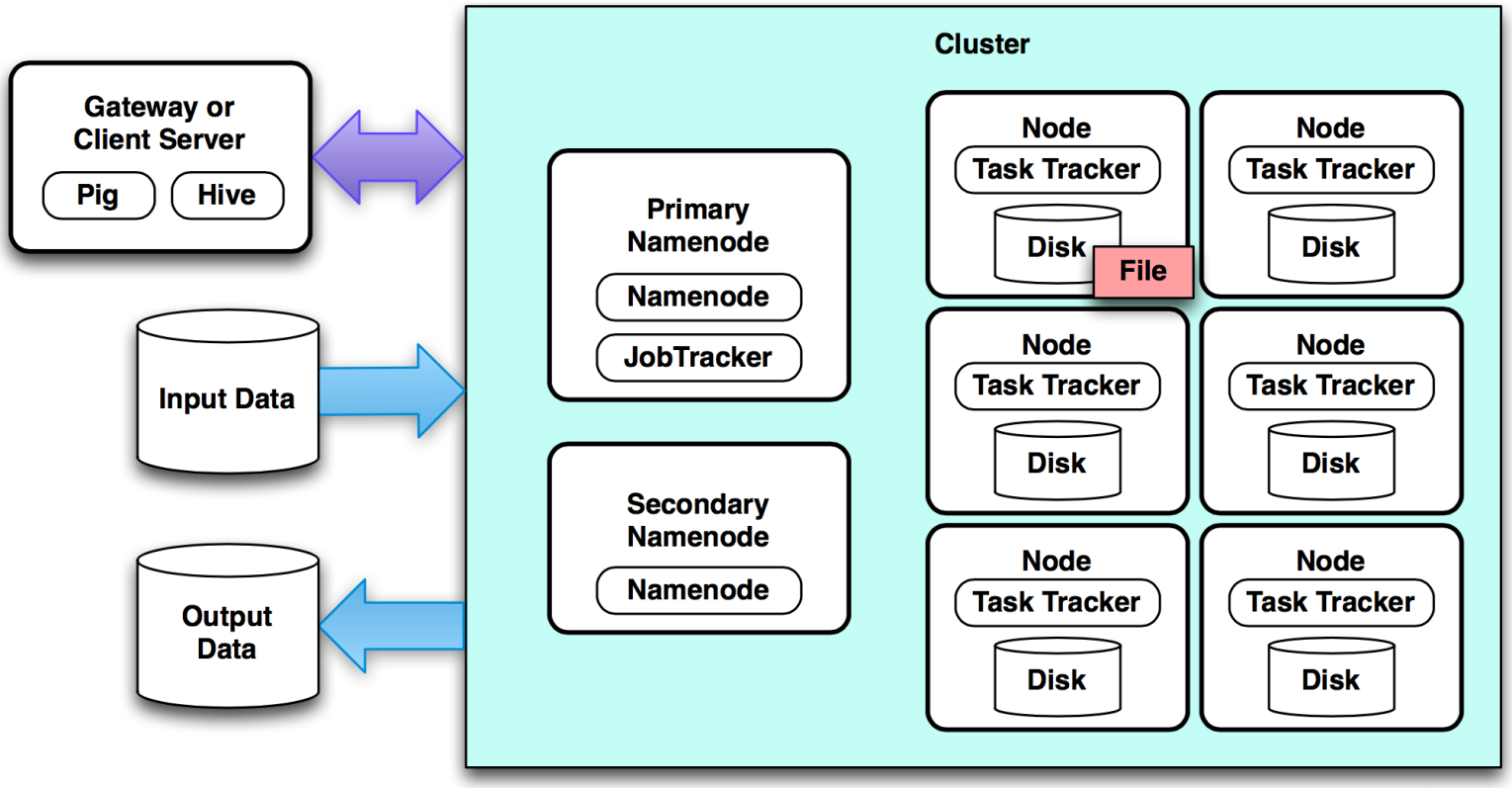
Local Gateway Files

- `$ ls file`



HDFS Files

- `$ hadoop fs -ls /user/myuser/file`



Hadoop Distributed File System

- HDFS command format

```
$ hadoop fs <–cmd> <args>
```

- List files on HDFS

```
$ hadoop fs –ls <args>
```

- Create a directory

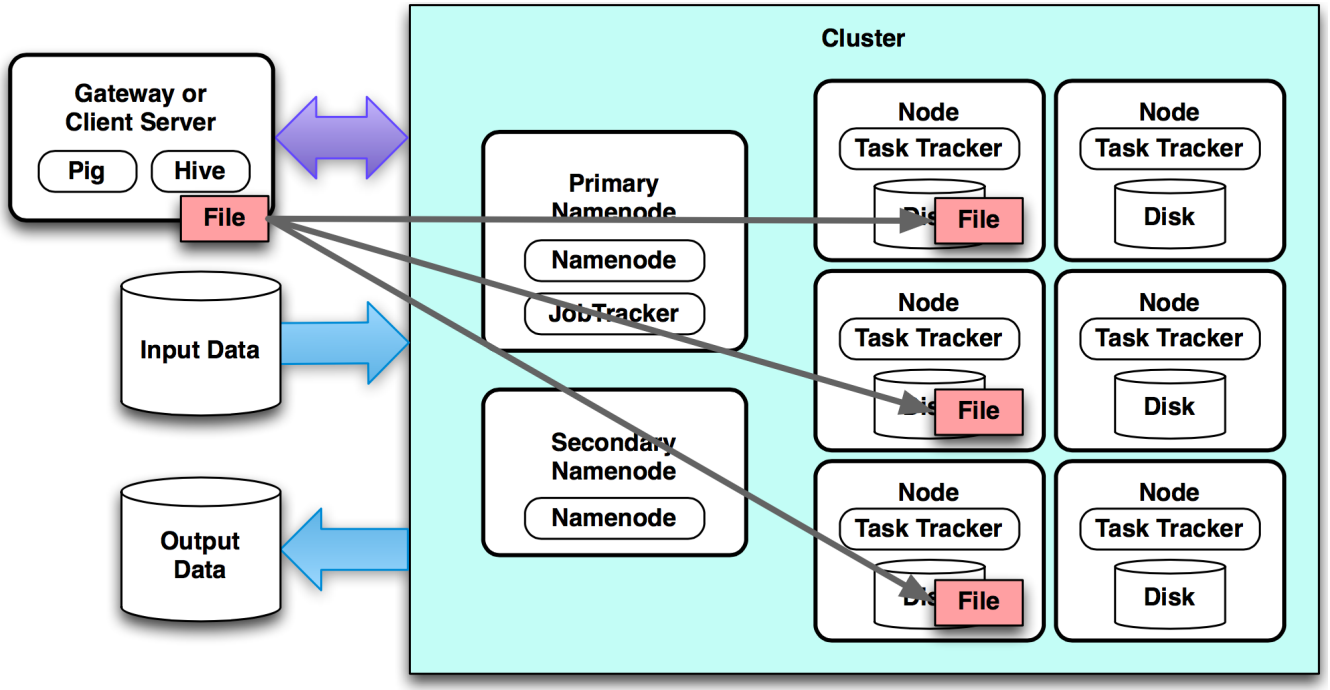
```
$ hadoop fs –mkdir <paths>
```

- Copy a local file onto HDFS

```
$ hadoop fs –copyFromLocal <localsrc> URI
```

HDFS Files

- `$ hadoop fs -copyFromLocal file /user/myuser/file`



Hadoop Distributed File System

- cat
- chgrp
- chmod
- chown
- copyFromLocal
- copyToLocal
- cp
- du
- dus
- expunge
- get
- getmerge
- ls
- lsr
- mkdir
- movefromlocal
- mv
- put
- rm
- rmr
- setrep
- stat
- tail
- test
- text
- touchz

Alternate File Systems

- Amazon S3
- MapR maprfs
- Quantcast QFS (1.5x expansion)
- Kosmix Cloudstore

MapReduce Processing

- Created a cluster
- Loaded your files into HDFS
- Process the files
- Some systems will run Map operations
- Some systems will run Reduce operations

MapReduce Processing

- MapReduce programming operates on key/value pairs
- Mapping examines the key and forms a value for the key
- Initial values may not exist

  (k1) → Mapper → (k2, v2)
- In general, we see the following

  (k1, v1) → Mapper → (k2, v2)

MapReduce Processing

- Reduce operations groups keys together

  (k1, v1) → Mapper → (k2, v2)

  { (k2, v2), (k2, v3), (k2, v4) } → Reducer → (k2, v5)

- In between Mapping and Reducing, there is a Shuffle phase where (k,v) pairs are sorted

MapReduce Processing

- Adding the Shuffle phase to a Mapper and Reducer, we see the following

(k1, v1) → Mapper → (k2, v2)
Shuffle
(k2, v2) → Reducer → (k3, v3)

MapReduce Processing

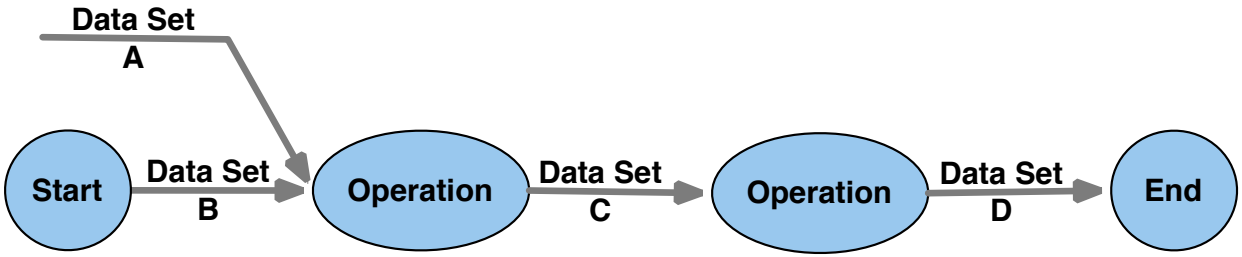
- Let's look at a word count example
- Input is text separated by white space
- Output is a count of each word from the text
  - Input: …the…the…
  - Output: (2, the)
- We can have multiple mappers and reducers
  - Divide and conquer using a series of MR operations
- We need a programming language
  - Pig

Pig

- A platform for analyzing large data sets
- ‘Pig’ doesn’t stand for anything, it was just a fun name
- Developed at Yahoo starting in 2007
- Open source Apache project
- High-level language
- Scripting language
- Performance is typically very close to native java

Pig

- Pig describes data flow
- Directed Acyclic Graph (DAG) / Complex workflow
  - Edges are data flows, Nodes are operators



- Operating on data sets, not variables

Scalar Data Types

- Data Types used within Data Sets
- Data Types
  - int
  - long
  - float
  - double
  - chararray
  - bytearray
- Nulls are allowed
  - Nulls define an unknown value, similar to SQL

Complex Data Types – Data Sets

- maps
  - Chararray to data element mapping
  - key:value
  - [’name’#’joe’, ’age’#30]
- tuples
  - Fixed-length ordered collection of data elements
  - Analogous to a row in SQL
  - (’joe’,’smith’,’100 Main Street’,’Champaign’,’Illinois’)
- bags
  - Unordered collection of tuples
  - {(’joe’,30), (’zoe’,40),(’tim’,32)}
  - {(10),(19),(25),(2)}

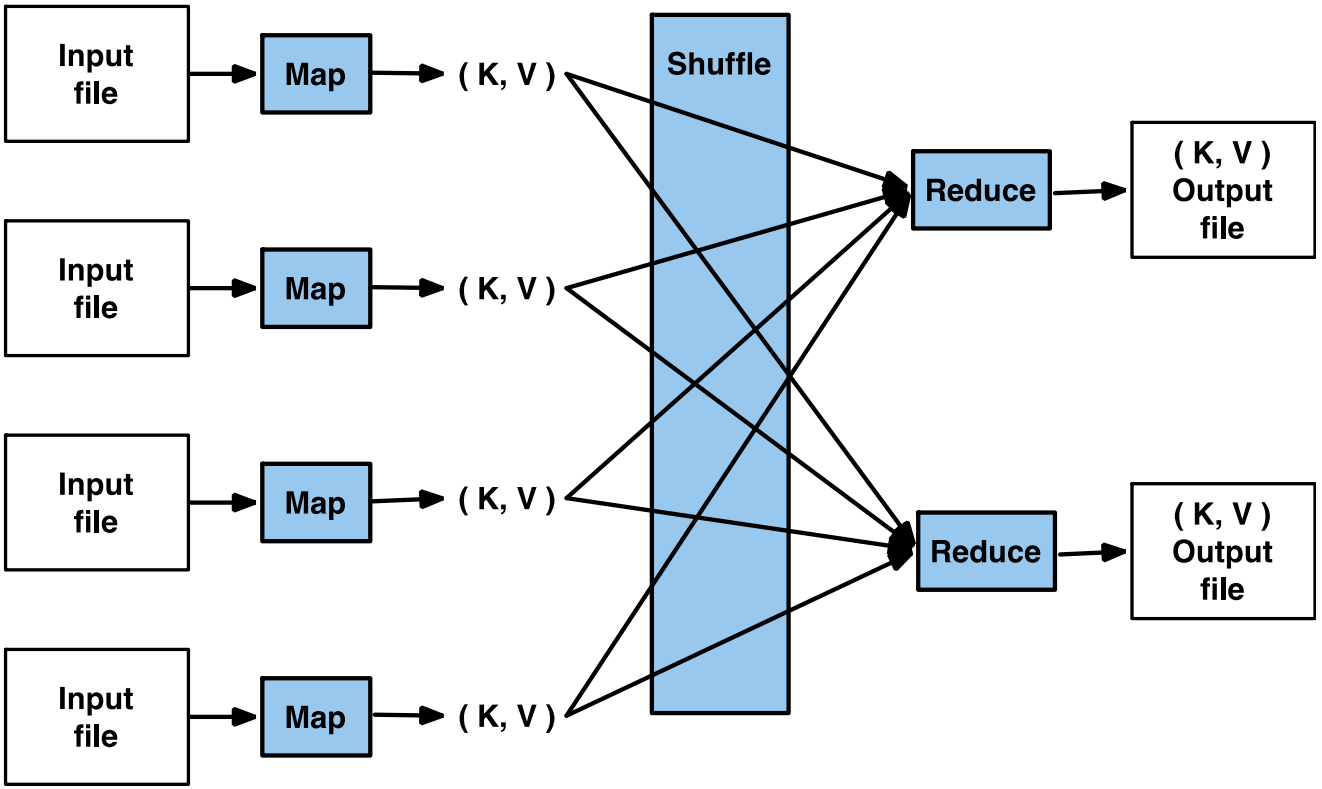
Pig Latin

- SQL-like, but unstructured data
- Ease of programming
  - Complex operations have been developed to simplify the code
  - Join, Filter, Group by, Order by, Union, Flatten, Foreach, Cogroup, Cross, MapReduce
- Script compiles to Java
- Built-in optimization with user settings
- Environment is extensible via User Defined Functions (UDF)

Pig Use Cases

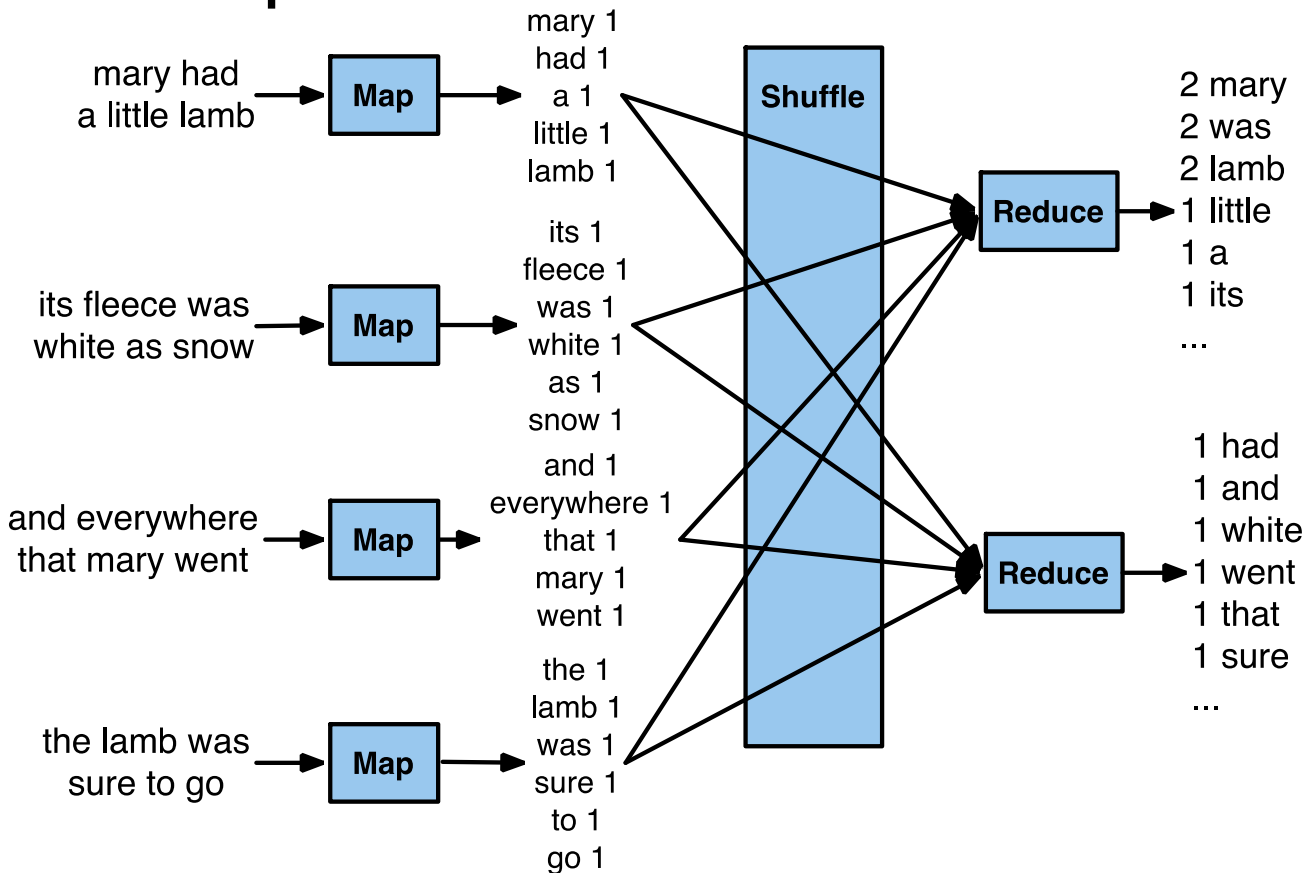
- Data pipelines
- Prediction models
- Ad-hoc queries
- Iterative processing models
- Mostly used on larger data sets
  - Smaller data sets, standard DB tools

MapReduce Processing



Programming Pig, by Alan Gates, O'Reilly. Copyright 2011 Yahoo! Inc.

MapReduce Processing



MapReduce Processing

- Let's try it
- Input file

```
mary had a little lamb
whose fleece was white as snow
and everywhere that mary went
the lamb was sure to go
```

- Pig script to count the words
- Single Hadoop node on laptop

Word Count Example

```
-- Load data from a file
input_lines = LOAD ’text1' AS (line:chararray);

-- Extract words from each line and put them into
a pig bag datatype, then flatten the bag to get
one word on each row
words = FOREACH input_lines GENERATE
FLATTEN(TOKENIZE(line)) AS word;

-- Filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w
+';
```

Word Count Example

```
-- create a group for each word
word_groups = GROUP filtered_words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE
COUNT(filtered_words) AS count, group AS word;

-- order the records by count
ordered_word_count = ORDER word_count BY count
DESC;
```

Word Count Example

```
-- Store the results into a file
STORE ordered_word_count INTO
'word_count_result';

-- Display the results
-- (This line is optional)
DUMP ordered_word_count;
```

Source: http://en.wikipedia.org/wiki/Pig_(programming_tool)

Word Count Example

- Copy the text file to HDFS

```
$ hadoop fs –copyFromLocal text1
/user/hadoop/text1
```

- Run the Pig script

```
$ pig word_count.pig
```

Word Count Example

- Output from the script

(2,mary)
(2,was)
(2,lamb)
(1,a)
(1,as)
(1,go)
(1,to)
(1,and)
(1,had)
(1,the)
(1,snow)
(1,sure)
(1,that)
(1,went)
(1,white)
(1,whose)
(1,fleece)
(1,little)
(1,everywhere)

Word Count Example

- Run the demo
- Hadoop environment on laptop

Code Walk-Through

```
-- Load data from a file
input_lines = LOAD 'text1' AS
(line:chararray);
```

- Comments indicated with leading dash-dash
- A file is loaded into a data set called input_lines (a bag) consisting of a set of lines
- Each line is defined as a chararray
- Ability to define schema

Code Walk-Through

```
-- Extract words from each line and put them
-- into a pig bag datatype, then flatten the
-- bag to get one word on each row
words = FOREACH input_lines GENERATE
FLATTEN(TOKENIZE(line)) AS word;
```

- The input_lines data set is parsed into individual words, each word becomes a row

1. (’mary had a little lamb’)
2. (’mary’,’had’,’a’,’little’,’lamb’)
3. (’mary’)
   (’had’)
   (’a’)…

Code Walk-Through

```
-- Filter out any words that
-- are just white spaces
filtered_words = FILTER words BY
word MATCHES '\\w+';
```

Code Walk-Through

```
-- create a group for each word
word_groups = GROUP filtered_words
BY word;
```

- Group similar words together
  - (’mary’,’mary’)
  - (’was’,’was’)
  - (’little’)
  - (’had’)
  - …

Code Walk-Through

```
-- count the entries in each group
word_count = FOREACH word_groups GENERATE
COUNT(filtered_words) AS count, group AS word;
```

- Create a new data set word_count based on the count of each group
  - (2, 'mary')
  - (1, 'a')
  - (2, 'was')
  - (1, 'had')
  - (2, 'lamb')
- Note the (k,v) – count is now k, word is now v

Code Walk-Through

```
-- order the records by count
ordered_word_count = ORDER
word_count BY count DESC;
```

- Create a new data set that is ordered
- Largest count first

Code Walk-Through

```
-- Store results into a file
STORE ordered_word_count INTO
'word_count_result';
```

- Write the ordered data set into an output file

Code Walk-Through

```
-- Display the results
-- (This line is optional)
DUMP ordered_word_count;
```

- Write the ordered data set to the screen
- MapReduce example in 7 lines

Sample Program Operations

- Map

  k1 → k1,v1

- Reduce

  $\left\{ \begin{array}{l} (k2, v1) \\ (k2, v2) \\ (k2, v3) \end{array} \right\} \rightarrow (k2, v4)$

Sample Program Operations

- Flatten

(k1, k2, k3) → { k1, k2, k3 }

- Group

{ k1, k1, k2, k1, k2 } → (k1, k1, k1), (k2, k2)

Where's the MapReduce?

- Each Pig line of code is considered a MapReduce operation
- Some operations don't require a reduction
  - Also called a zero reducer
- Pig will attempt to optimize the MapReduce operations, with additional environment settings that can influence the optimization

Why Use Pig?

- Easy to read
- Comparable speed performance to Java
- MapReduce example in 7 lines

Development Savings

- One author ran a simple experiment
  - 9 lines of Pig Latin, 15 minutes to debug
  - 170 lines of Java MapReduce, 4 hours to debug
- 19:1 ratio in code reduction
- 16:1 ratio in development time reduction
- Yahoo example
  - ~120 lines of Pig Latin, 3 days development
  - ~3000 lines of Java MapReduce, 3 months development
- 25:1 ratio in code reduction
- 20:1 ratio in development time reduction

Word Count Example in Hive

```
CREATE TABLE lines(line STRING);
LOAD DATA INPATH 'text1' OVERWRITE INTO
TABLE lines;
  SELECT count(*), word
    FROM lines
 LATERAL VIEW explode(split(text,'\s'))
         subView AS word
GROUP BY word
ORDER BY count DESC;
```

Hive

- Query execution in MapReduce
- Flexible data structure mapping
- Direct access to files in HDFS or HBase
- Easy data extract/transform/load (ETL) operation
- SQL-like language called HiveQL
- Not for Online Transaction Processing (OLTP)
- Ad-hoc queries
- Use other scripting languages to build up successive queries

But wait, there’s more!

Computing Trends

- Why can't we have it all?
  - Batch
  - Interactive
  - Streaming
- Combinations of models
  - Batch, Interactive, Streaming together
- Sophisticated algorithms
- Compatible with existing Hadoop environments & programming languages

Faster Computation Goals

- What if we exploit in-memory computation to improve processing speed
- Avoid disk I/O – true bottleneck
- Focus on smaller data sets
  - MB/GB ranges
- Add Increased parallelism – more maps
- Add low latency scheduler
- Add optimization for parallel communication patterns
- Add recovery from failures and stragglers

Trade-offs for Speed

- With large amounts of data in memory, performance can still be difficult
- Scanning 500 GB of data could still take 10s of seconds
- Approximate results are acceptable, given that you characterize the accuracy of the response

Provide a Richer Environment

- Allow faster computation environment
- Allow single environment supporting easy combination of batch, interactive and streaming models
- Allow easy development of sophisticated algorithms
  - Python and Scala shell
  - High level abstraction for graph-based and machine learning algorithms
- Allow for compatibility with Hadoop, HDFS, Hive, Flume, etc.

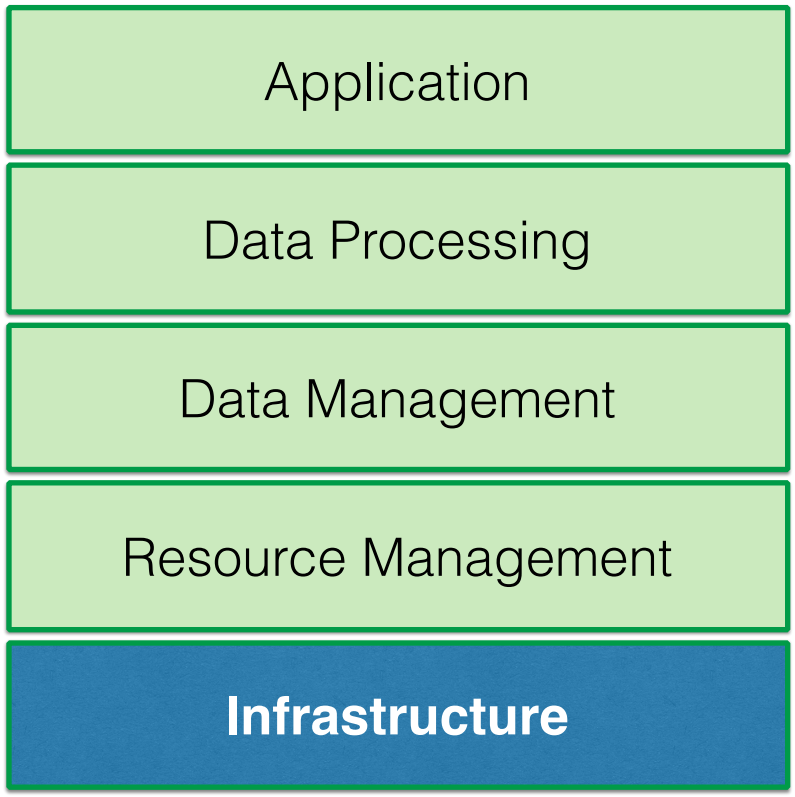
Berkeley Data Analytics Stack (BDAS)

- Project to achieve a richer environment
- Modify the Hadoop environment to add features, yet keep compatibility
- Introduce a Resource Management layer to share infrastructure across frameworks

Berkeley Data Analytics Stack (BDAS)

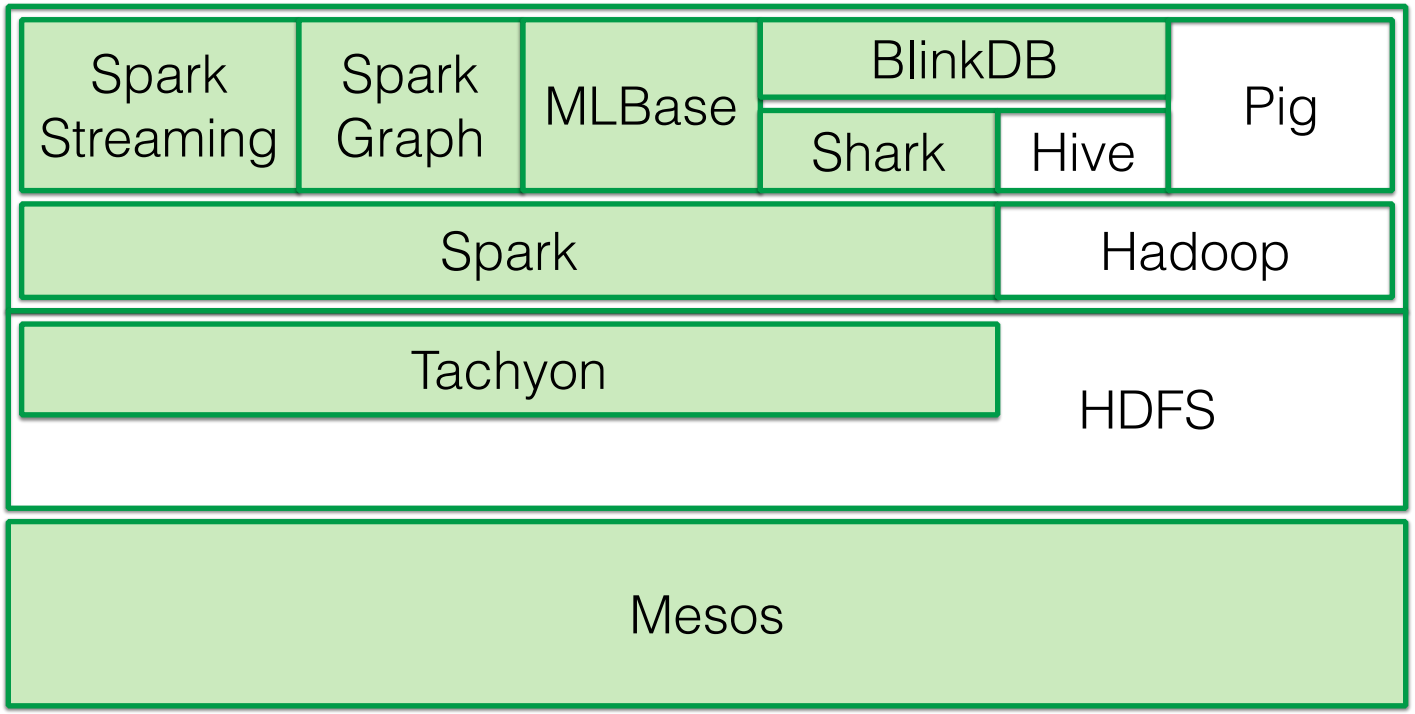
- Modify Data Management layer for data sharing across layers
- Modify Data Processing layer for in-memory processing
- Modify Application layer for high-level approaches for improved programming capabilities

Berkeley Data Analytics Stack (BDAS)



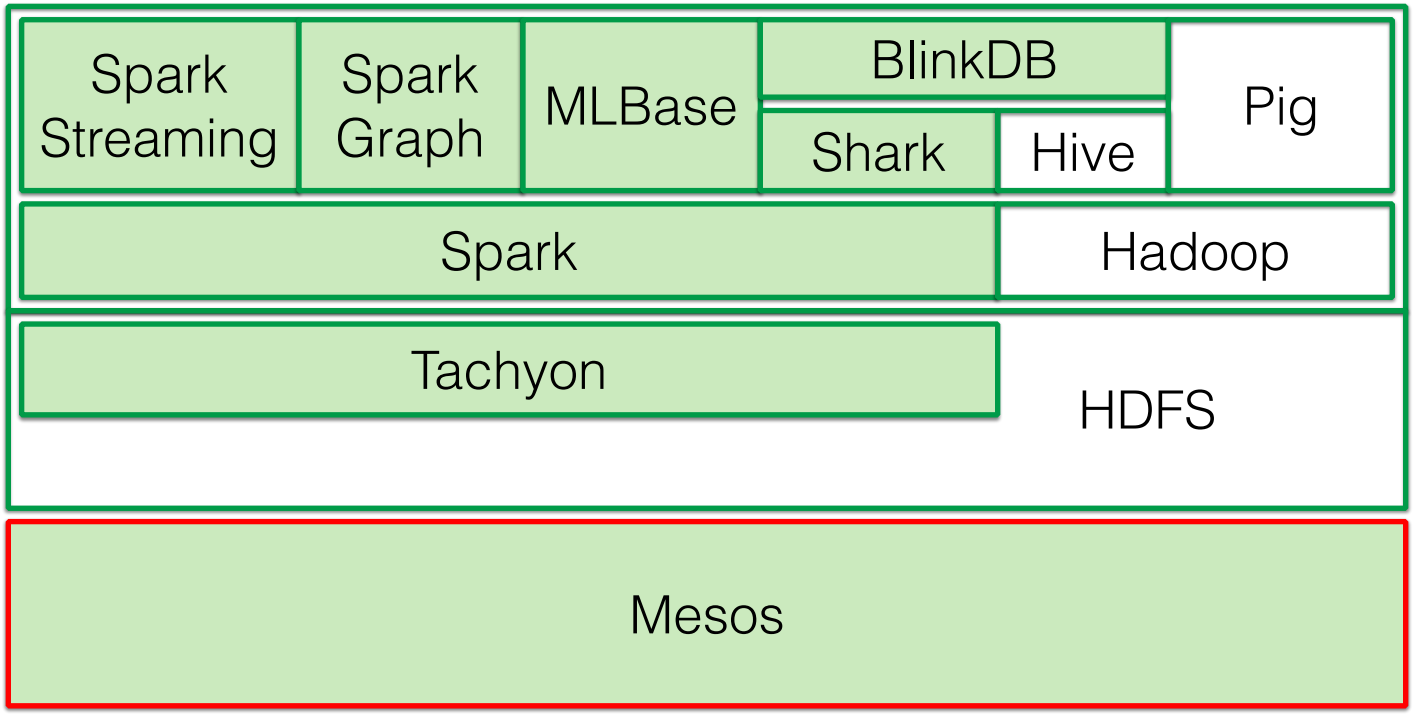
Source https://amplab.cs.berkeley.edu/software/

Berkeley Data Analytics Stack (BDAS)



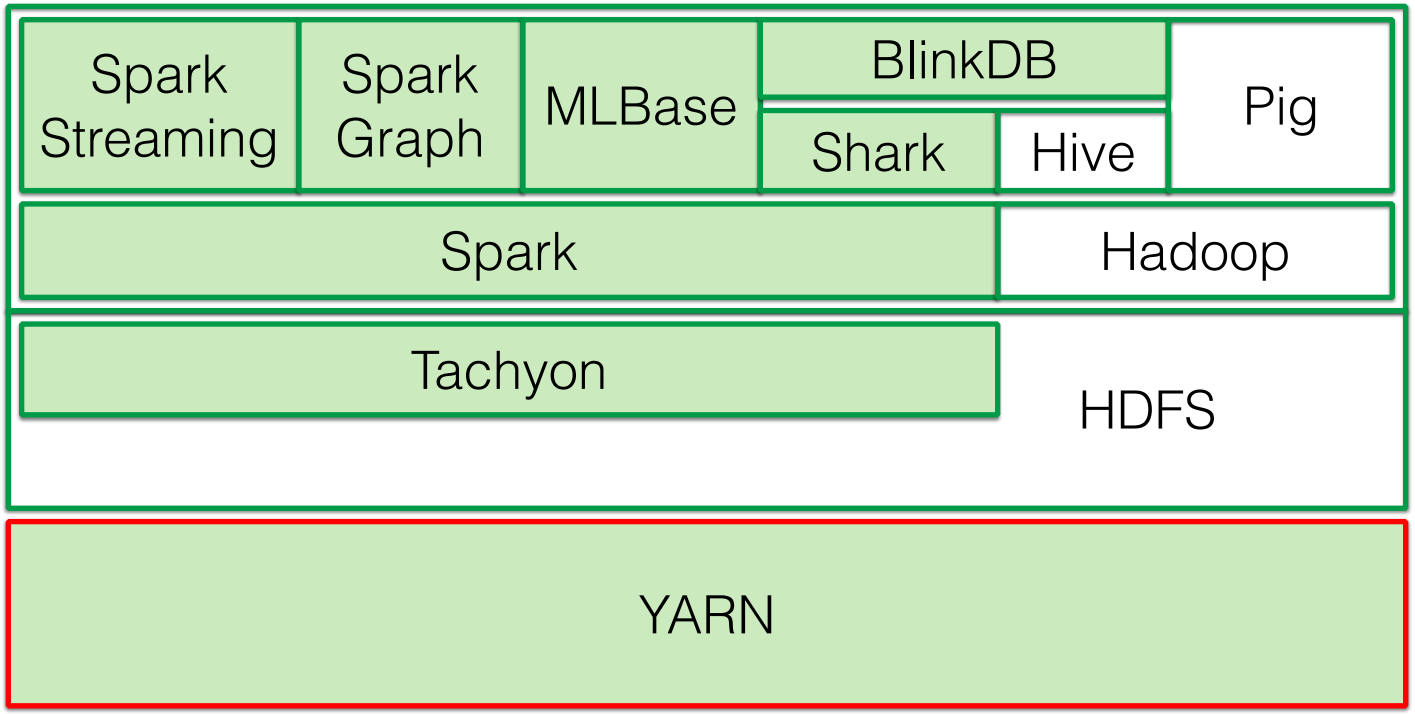
Source https://amplab.cs.berkeley.edu/software/

Mesos is the new Resource Manager



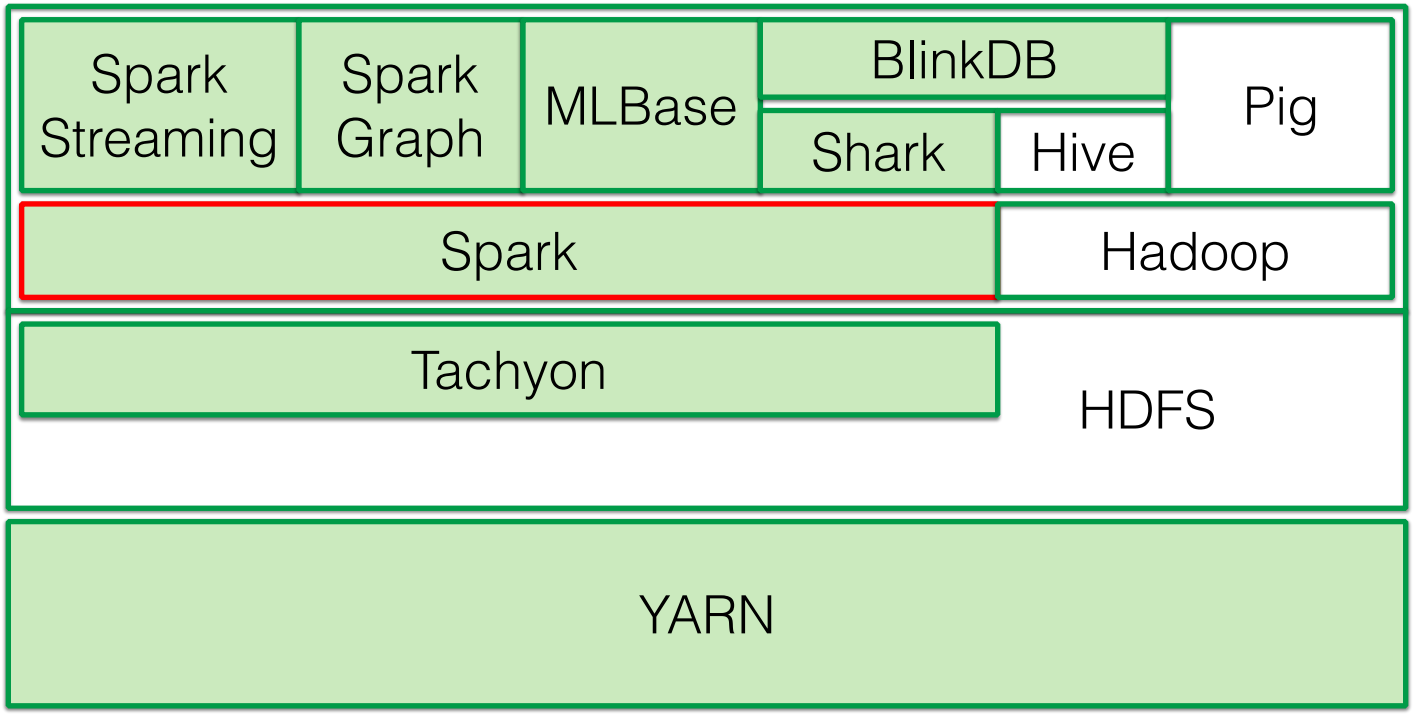
Source https://amplab.cs.berkeley.edu/software/

YARN – Yet Another Resource Negotiator



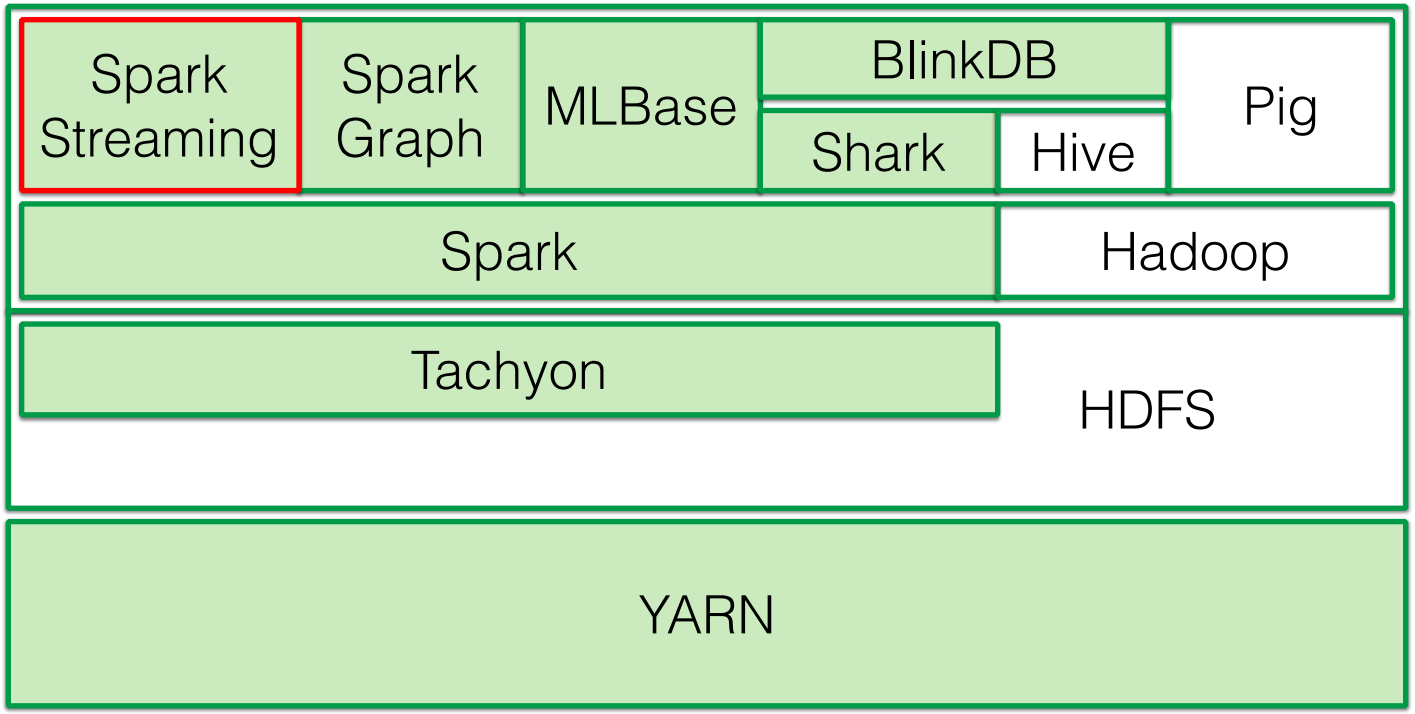
Source https://amplab.cs.berkeley.edu/software/

Spark is the in-memory framework



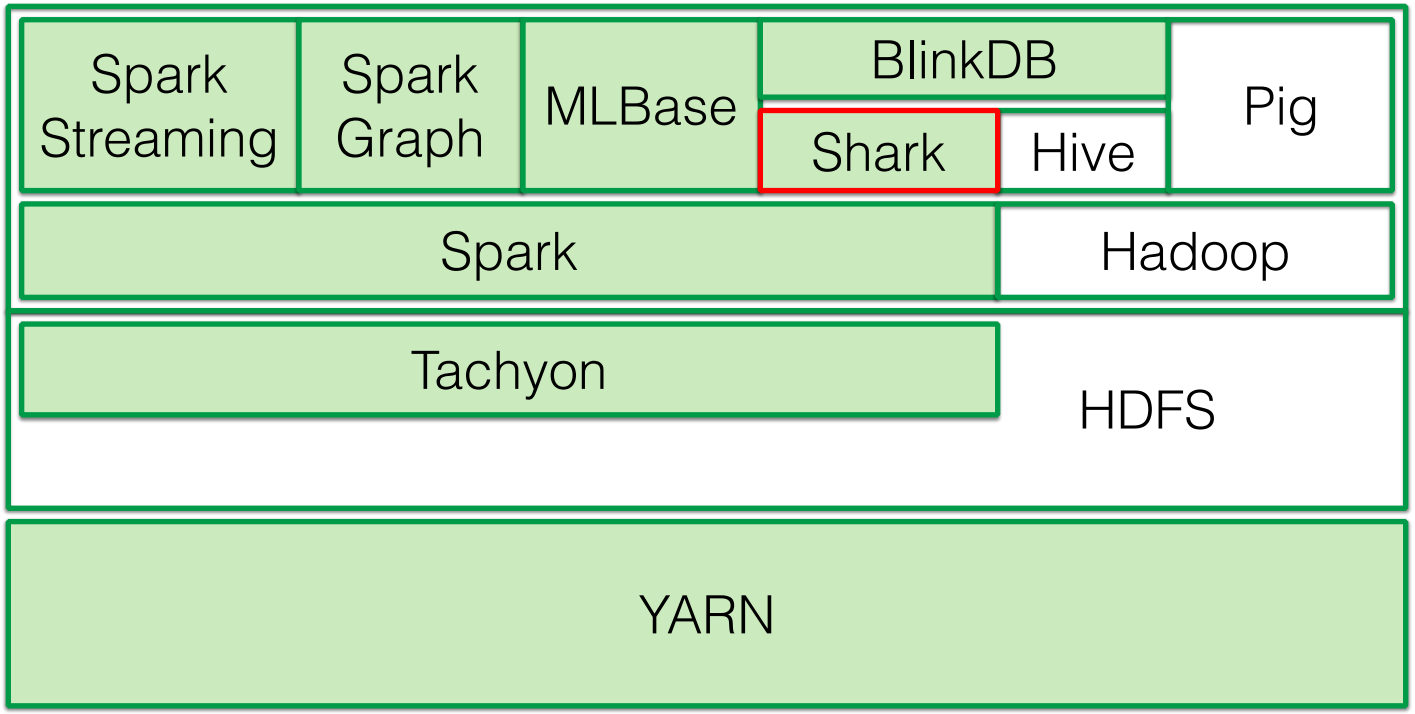
Source https://amplab.cs.berkeley.edu/software/

Spark Streaming for streaming models



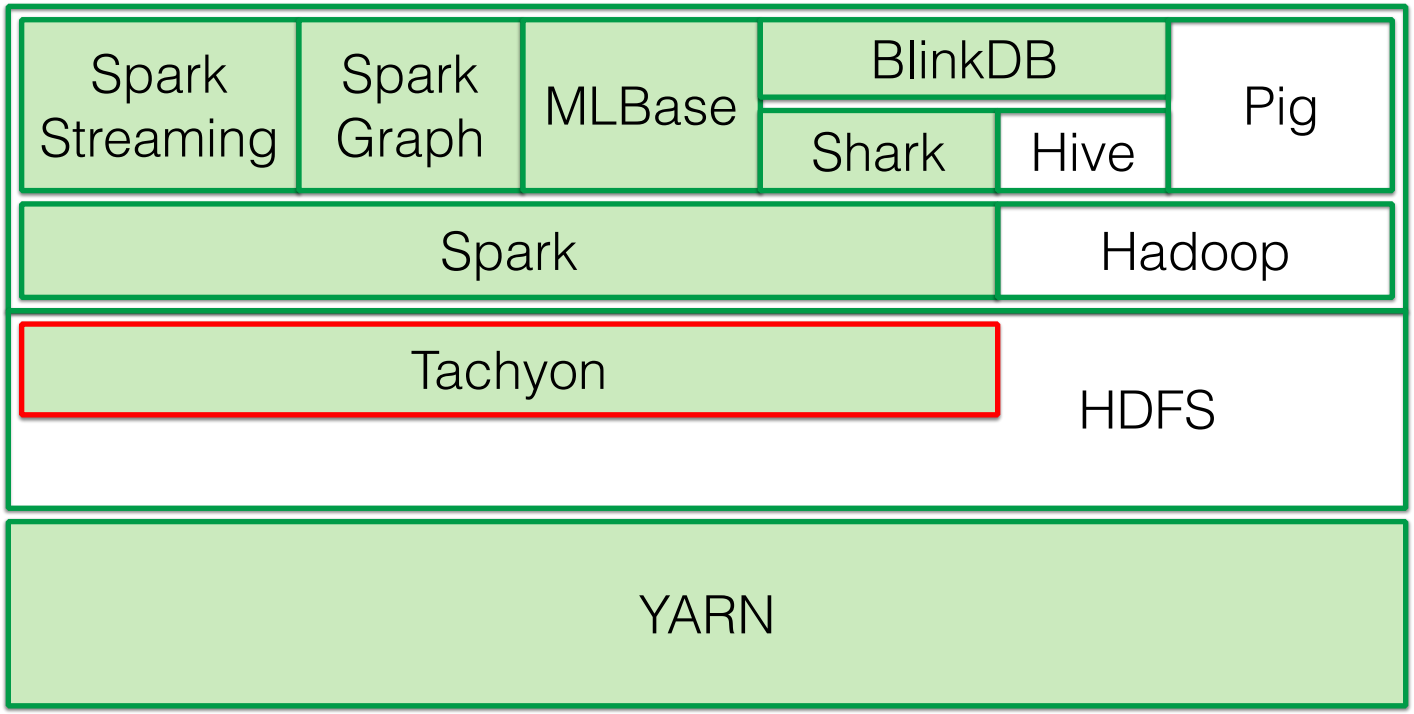
Source https://amplab.cs.berkeley.edu/software/

Shark allows Hive to use Spark



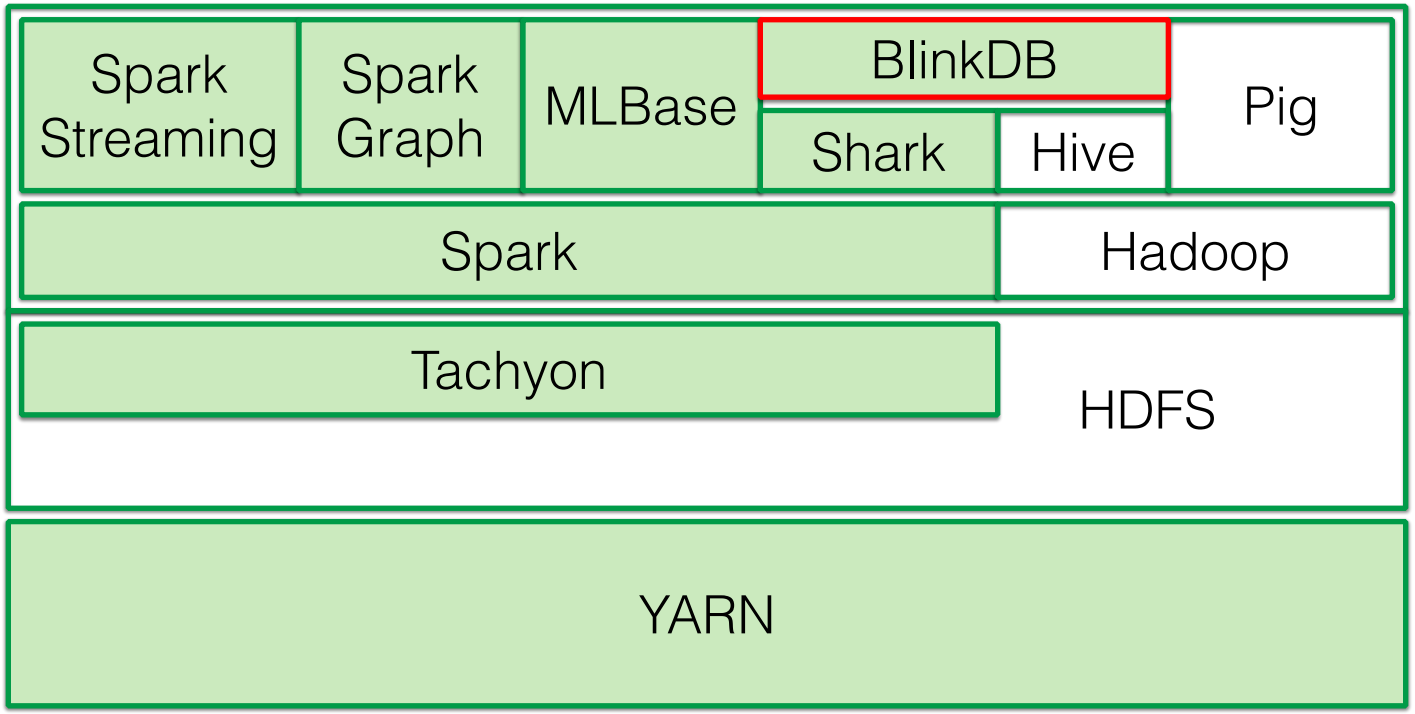
Source https://amplab.cs.berkeley.edu/software/

Tachyon is the in-memory interface to HDFS



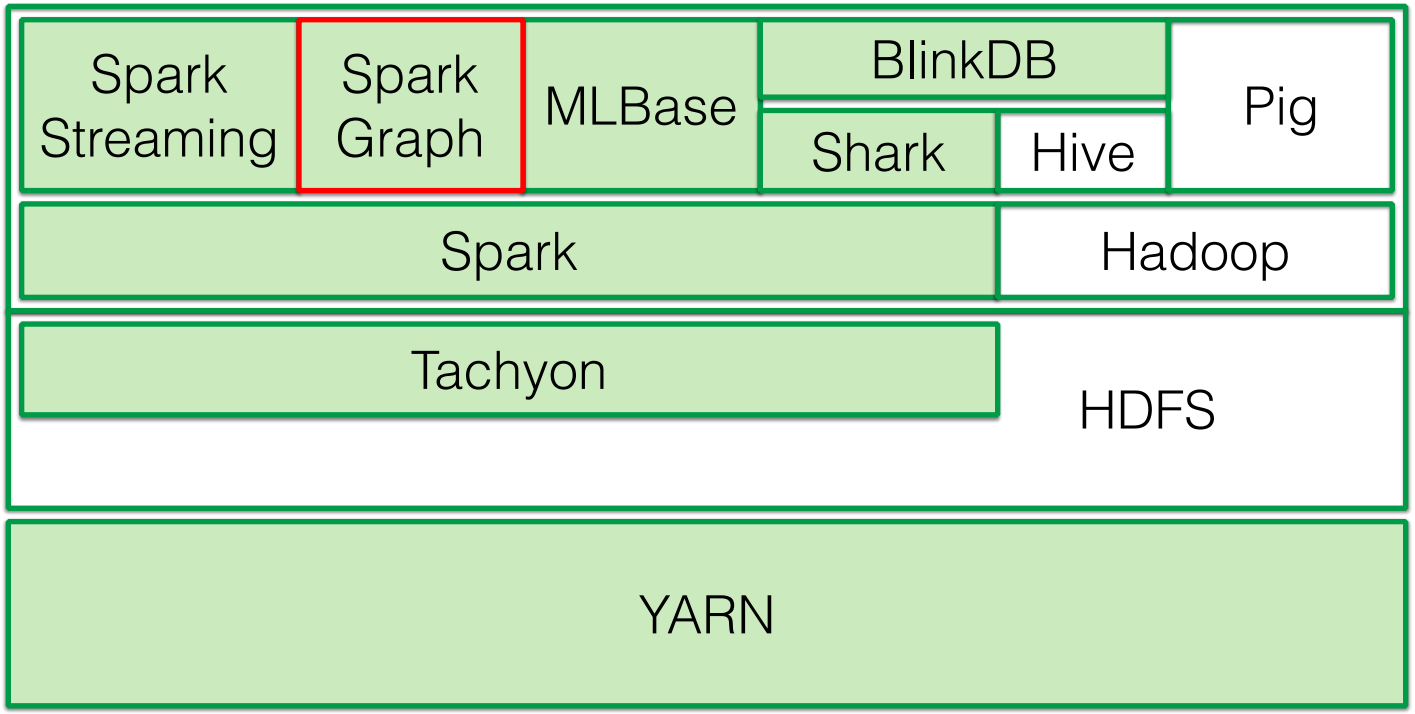
Source https://amplab.cs.berkeley.edu/software/

BlinkDB is the approximation engine



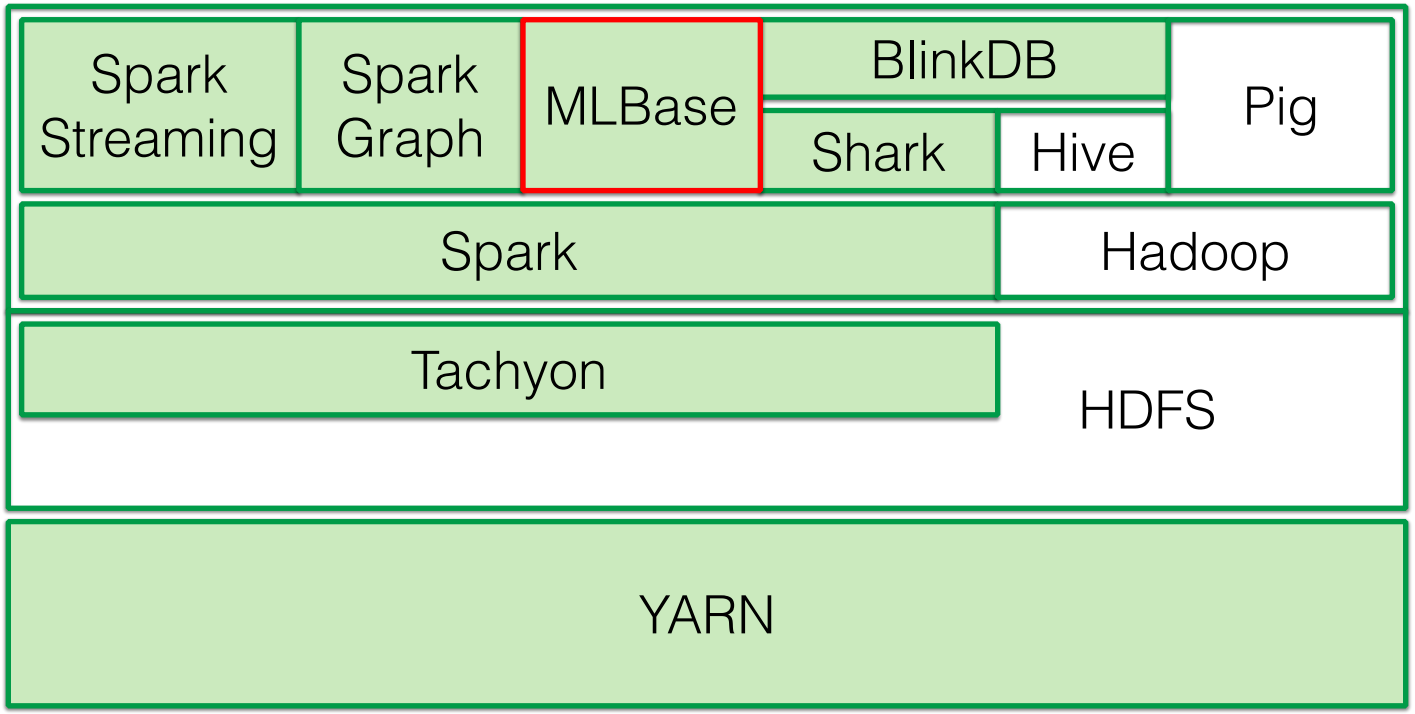
Source https://amplab.cs.berkeley.edu/software/

Spark Graph is a GraphLAB API



Source https://amplab.cs.berkeley.edu/software/

MLBase is machine learning for non-experts



Source https://amplab.cs.berkeley.edu/software/

Spark and Shark on Amazon

- Amazon now supports Spark and Shark on Elastic MapReduce

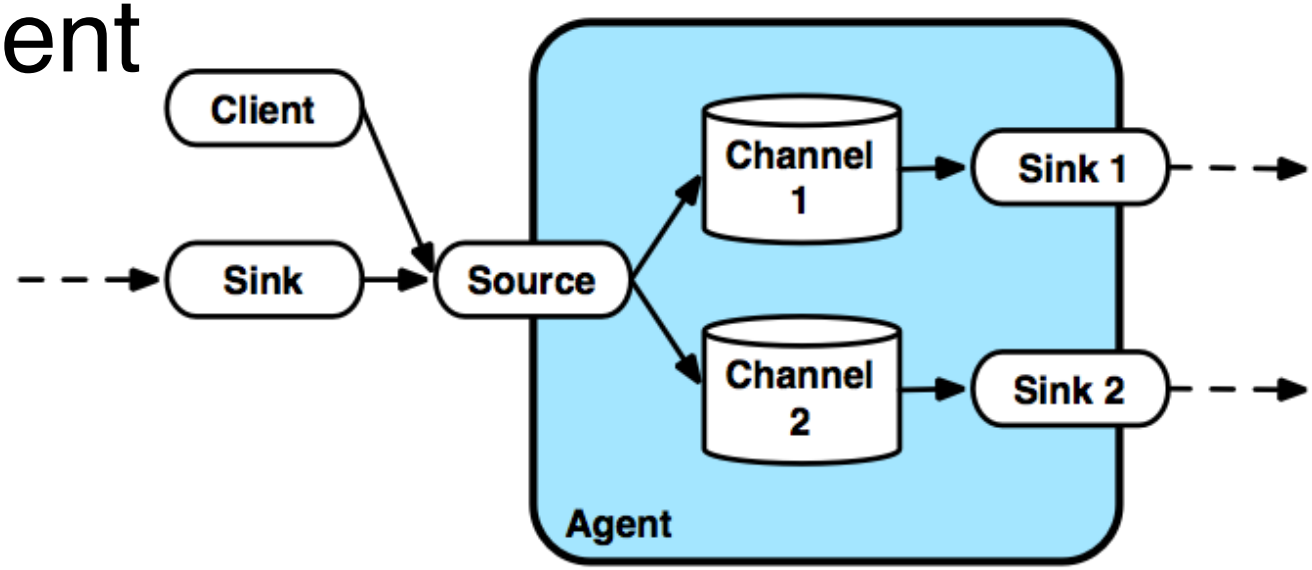
That’s not all. If you call within the next 10 minutes, we’ll throw in…

Flume

- System to collect data and load it onto Hadoop
- Used to migrate log data
- Reliable message delivery system
  - Message is defined as an Event
- Defines Flow Pipeline with Source and Sink connection points
- Sources and Sinks belong to an Agent

Flume

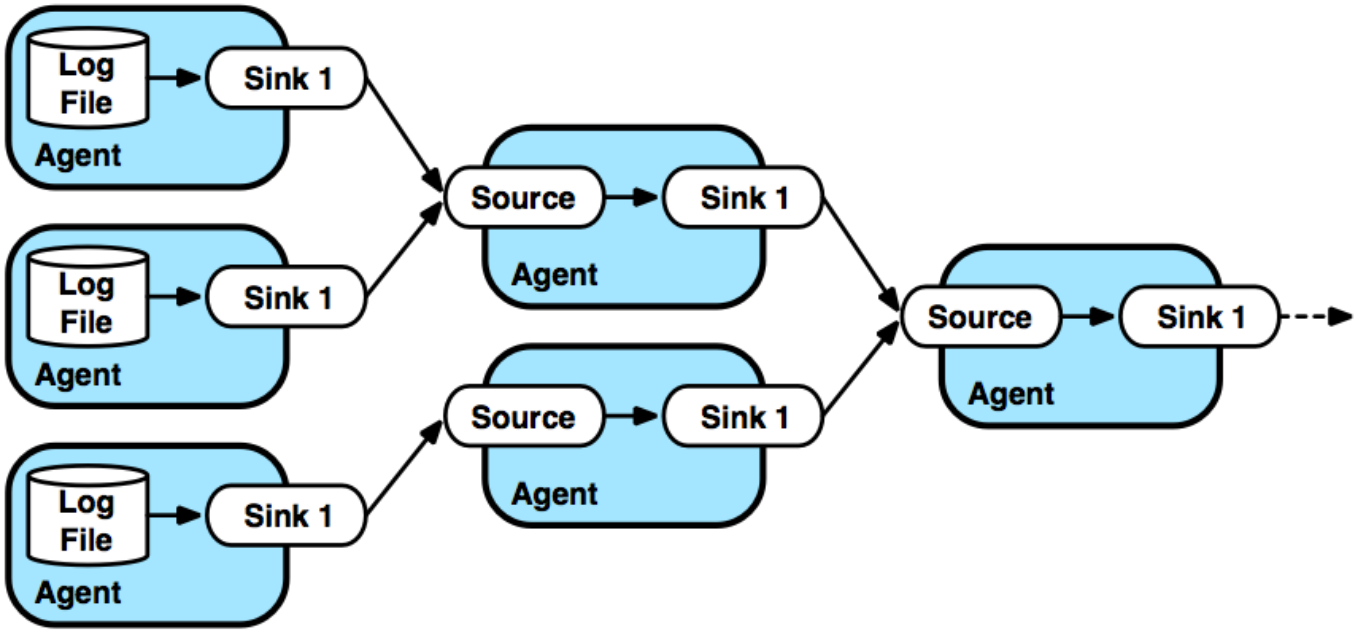
- Sink and Source connections in an Agent



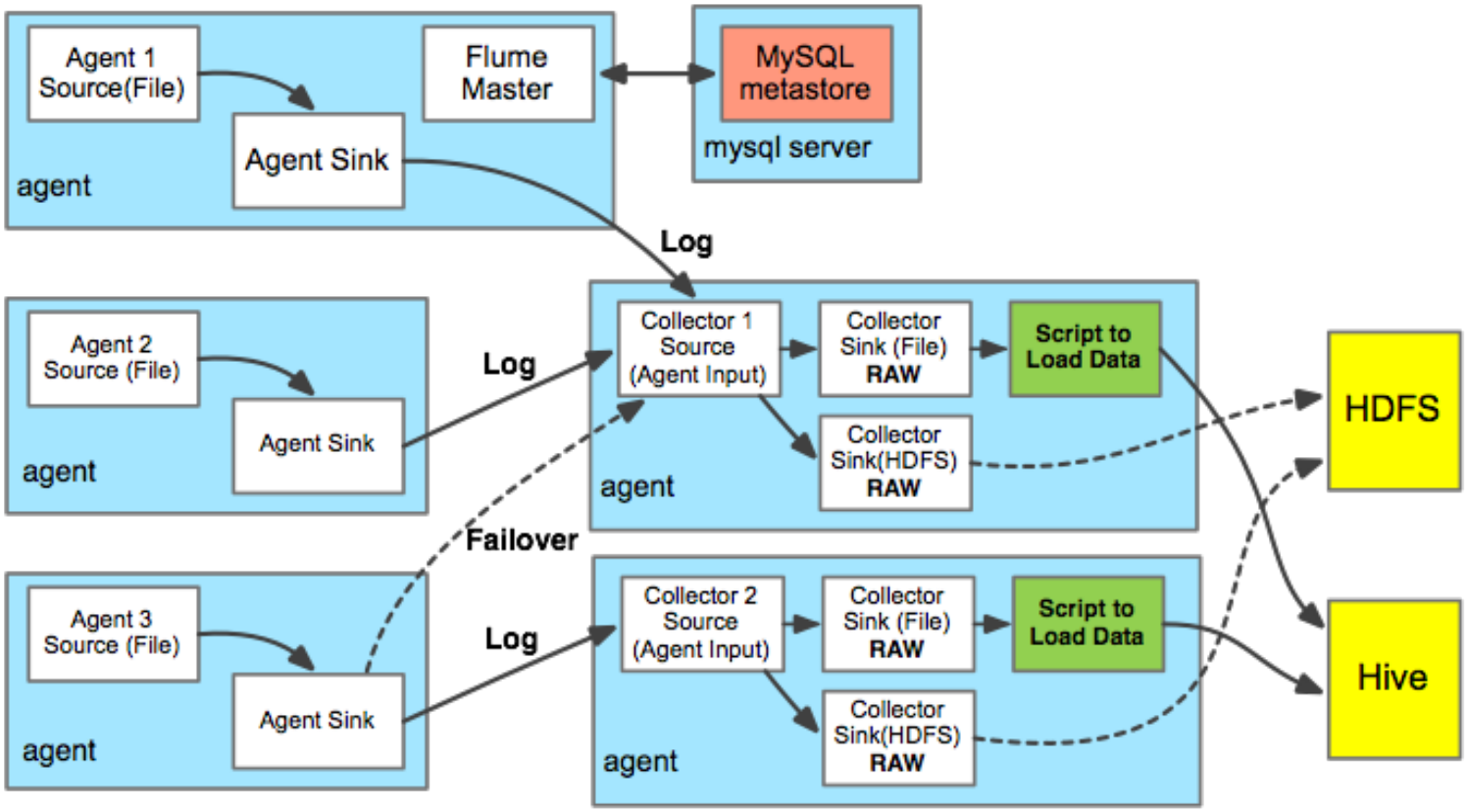
Source: http://blog.cloudera.com/blog/2011/12/apache-flume-architecture-of-flume-ng-2/

Flume

- Connecting Sinks to Sources creates fan-in data model to load data into Hadoop



Sample Configuration

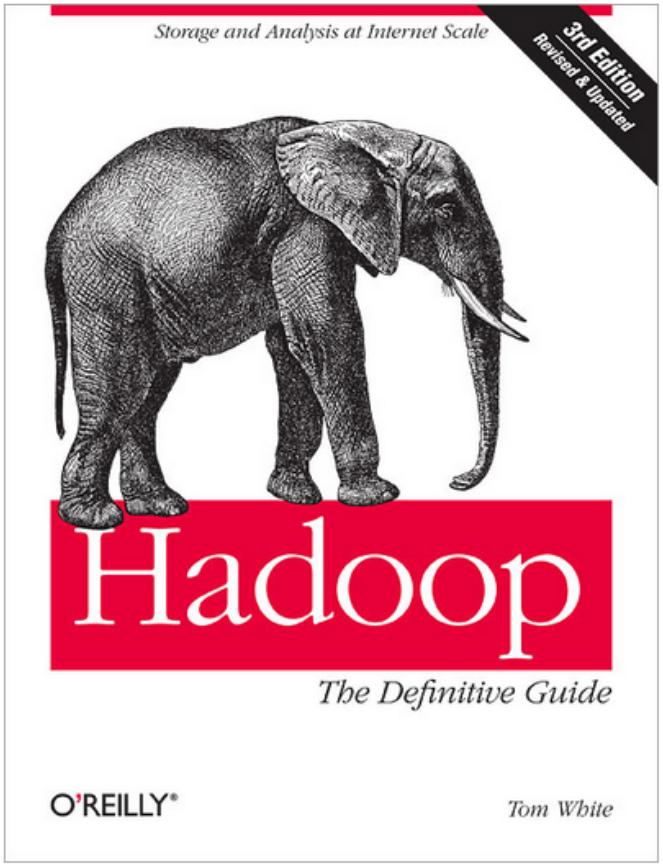


References

- Apache Hadoop Home
- http://hadoop.apache.org/

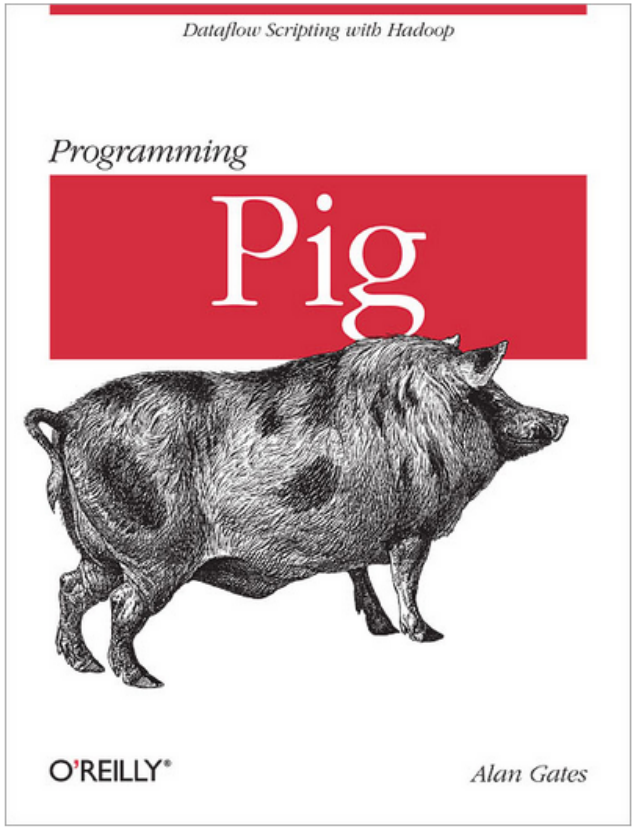
References

- Hadoop: The Definitive Guide, by Tom White



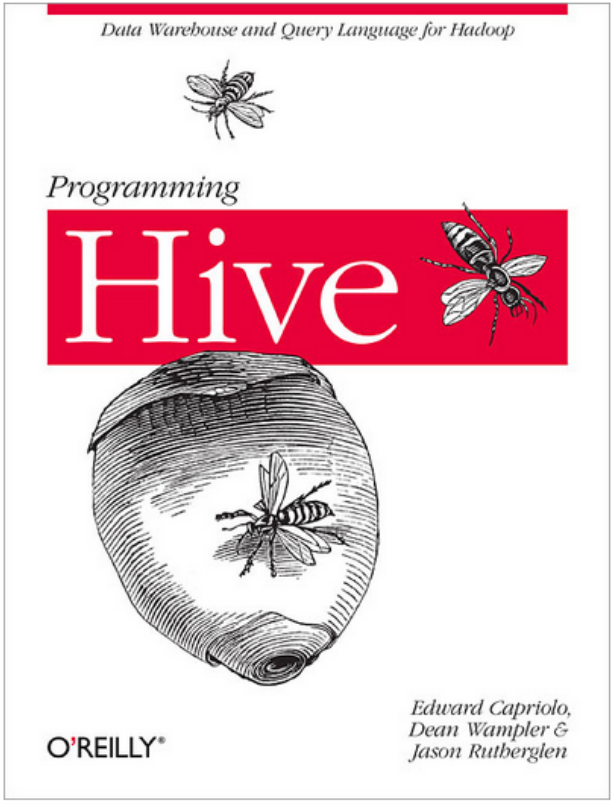
References

- Programming Pig, by Alan Gates



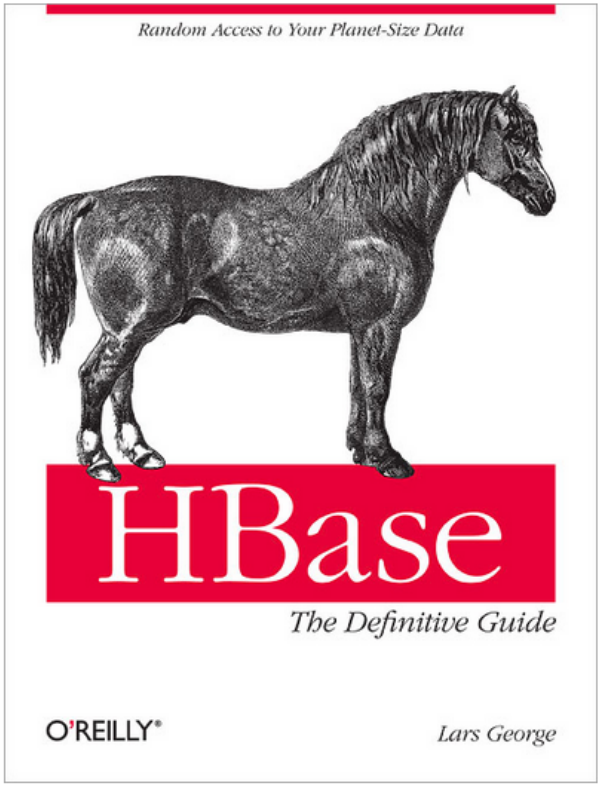
References

- Programming Hive, by Edward Capriolo, Dean Wampler & Jason Rutherglen



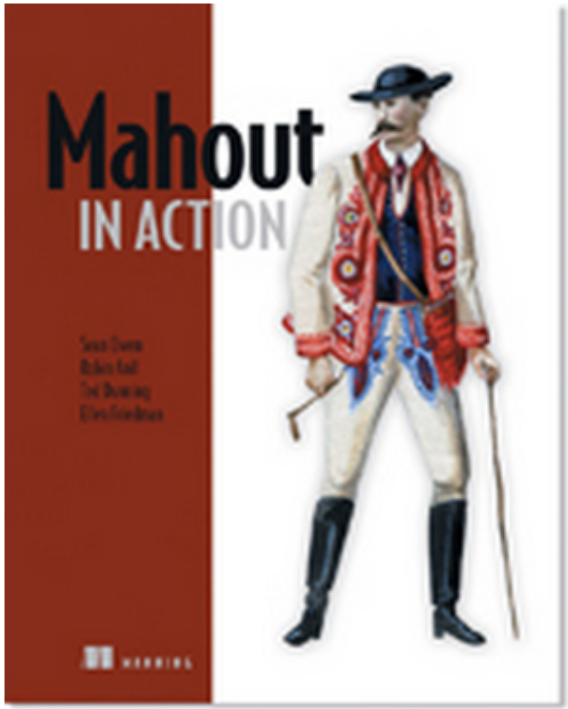
References

- Hbase: The Definitive Guide, by Lars George



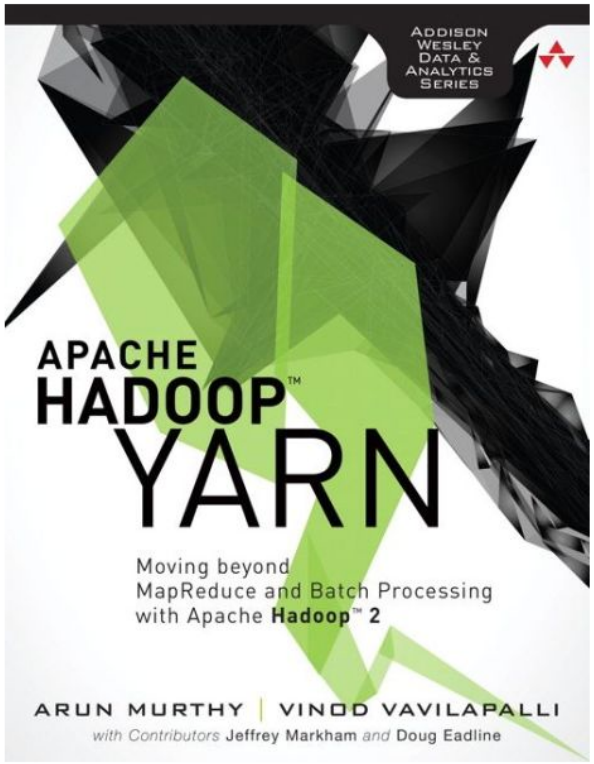
References

- Mahout in Action, by Sean Owen, Robin Anil, Ted Dunning and Ellen Friedman



References

- Apache Hadoop YARN,
  by Arun Murthy and Vinod Vavilapalli
  (Jan 2014)



References

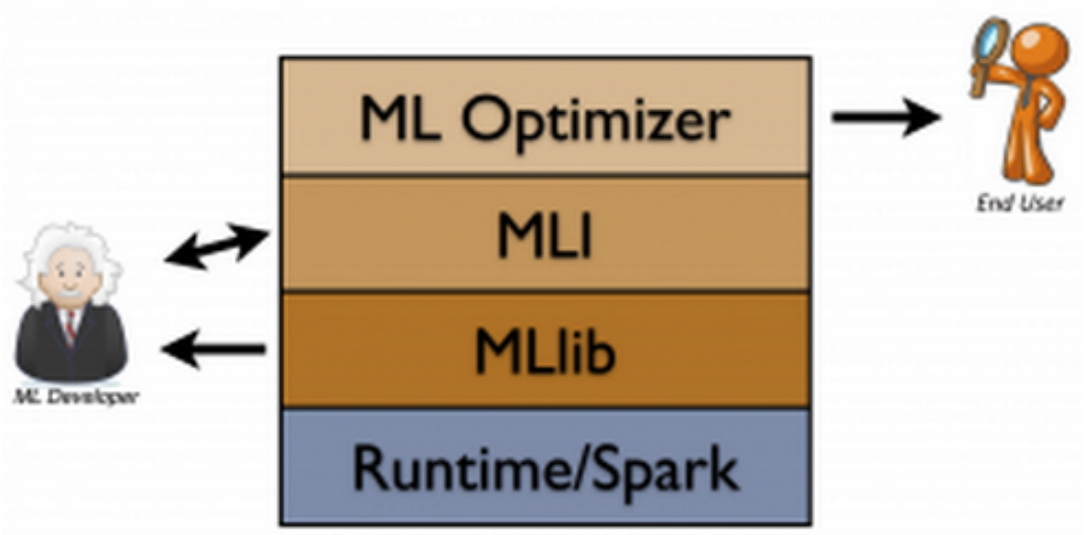
- Flume User Guide
- http://flume.apache.org/FlumeUserGuide.html

References

- Spark Project
- http://spark-project.org/
- http://spark-project.org/docs/latest/

References

- MLBase
- http://mlbase.org/



References

- Python on Hadoop
- http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python

Summary

- Wonderful world of MapReduce
- Word count example in Pig & Hive
- Berkeley AMPLab
- Flume log handling
- Join U-CHUG
  - http://www.meetup.com/Urbana-Champaign-Hadoop-User-Group-U-CHUG

Thank you

ken@agrible.com
LinkedIn: kentaylor7