Knowledge Summary

04/20/2016

by

Guan Zhou

Week 1: Hadoop Ecosystem & HDFS

1. What is Hadoop?

Hadoop is a free, Java-based, open-source software/programming framework that supports the processing of large data sets and running applications in a distributed computing environment/clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs.

(1)Open-source software. Open-source software is created and maintained by a network of developers from around the globe. It's free to download, use and contribute to, though more and more commercial versions of Hadoop are becoming available.

(2)Framework. In this case, it means that everything you need to develop and run software applications is provided – programs, connections, etc.

(3)Massive storage. The Hadoop framework breaks big data into blocks, which are stored on clusters of commodity hardware.

(4)Processing power. Hadoop concurrently processes large amounts of data using multiple low-cost computers for fast results.

2. Why do we need Hadoop and when should we use Hadoop?

Everyday a large amount of unstructured data is getting dumped into our machines. The major challenge is not to store large data sets in our systems but to retrieve and analyze the big data in theorganizations, that the data present in different machines at different locations. In this situation a necessity for Hadoop arises. Hadoop has the ability to analyze the data present in different machines at different locations very quickly and in a very cost effective way. It uses the concept of MapReduce which enables it to divide the query into small parts and process them in parallel.

Hadoop is ideal for storing large amounts of data, like terabytes and petabytes, and uses HDFS as its storage system. HDFS lets you connect nodes (commodity personal computers) contained within clusters over which data files are distributed. You can then access and store the data files as one seamless file system. Access to data files is handled in a streaming manner, meaning that applications or commands are executed directly using the MapReduce processing model.

Week 1: Hadoop Ecosystem & HDFS

3. Two major components of Hadoop:
HDFS and MapReduce are the two major components of Hadoop, where HDFS is from the 'Infrastructural' point of view and MapReduce is from the 'Programming' aspect.

4. What is HDFS? Key features? Namenode vs Datanode?
(1) HDFS is a distributed and scalable file system designed for storing very large files with streaming data access patterns, running clusters on commodity hardware. A HDFS cluster primarily consists of a NameNode that manages the file system metadata and DataNodes that store the actual data.
(2) Key features of HDFS: HDFS is highly fault-tolerant, with high throughput, suitable for applications with large data sets, streaming access to file system data and can be built out of commodity hardware.
(3) The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself.
A DataNode stores data in the [Hadoop File System]. A functional filesystem has more than one DataNode, with data replicated across them.

5. Assumptions and Goals/Objectives behind HDFS:
(1) Large Data Sets: It is assumed that HDFS always needs to work with large data sets. It will be an underplay if HDFS is deployed to process several small data sets ranging in some megabytes or even a few gigabytes. The architecture of HDFS is designed in such a way that it is best fit to store and retrieve huge amount of data. What is required is high cumulative data bandwidth and the scalability feature to spread out from a single node cluster to a hundred or a thousand-node cluster. The acid test is that HDFS should be able to manage tens of millions of files in a single occurrence.
(2) Write Once, Read Many Model: HDFS follows the write-once, read-many approach for its files and applications. It assumes that a file in HDFS once written will not be modified, though it can be access 'n' number of times (though future versions of Hadoop may support this feature too)! At present, in HDFS strictly has one writer at any time. This assumption enables high throughput data access and also simplifies data coherency issues. A web crawler or a MapReduce application is best suited for HDFS.

Week 1: Hadoop Ecosystem & HDFS

(3) Streaming Data Access: As HDFS works on the principle of 'Write Once, Read Many', the feature of streaming data access is extremely important in HDFS. As HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access. HDFS focuses not so much on storing the data but how to retrieve it at the fastest possible speed, especially while analyzing logs. In HDFS, reading the complete data is more important than the time taken to fetch a single record from the data. HDFS overlooks a few POSIX requirements in order to implement streaming data access.

(4) Commodity Hardware: HDFS assumes that the cluster(s) will run on common hardware, that is, non-expensive, ordinary machines rather than high-availability systems. A great feature of Hadoop is that it can be installed in any average commodity hardware. We don't need super computers or high-end hardware to work on Hadoop. This leads to an overall cost reduction to a great extent.

(5) Data Replication and Fault Tolerance: HDFS works on the assumption that hardware is bound to fail at some point of time or the other. This disrupts the smooth and quick processing of large volumes of data. To overcome this obstacle, in HDFS, the files are divided into large blocks of data and each block is stored on three nodes: two on the same rack and one on a different rack for fault tolerance. A block is the amount of data stored on every data node. Though the default block size is 64MB and the replication factor is three, these are configurable per file. This redundancy enables robustness, fault detection, quick recovery, scalability, eliminating the need of RAIDstorage on hosts and merits of data locality.

(6) High Throughput: Throughput is the amount of work done in a unit time. It describes how fast the data is getting accessed from the system and it is usually used to measure performance of the system. In Hadoop HDFS, when we want to perform a task or an action, then the work is divided and shared among different systems. So, all the systems will be executing the tasks assigned to them independently and in parallel. So the work will be completed in a very short period of time. In this way, the Apache HDFS gives good throughput. By reading data in parallel, we decrease the actual time to read data tremendously.

Week 1: Hadoop Ecosystem & HDFS

(7) Moving Computation is better than Moving Data: Hadoop HDFS works on the principle that if a computation is done by an application near the data it operates on, it is much more efficient than done far of, particularly when there are large data sets. The major advantage is reduction in the network congestion and increased overall throughput of the system. The assumption is that it is often better to locate the computation closer to where the data is located rather than moving the data to the application space. To facilitate this, Apache HDFS provides interfaces for applications to relocate themselves nearer to where the data is located.
(8) File System Namespace: A traditional hierarchical file organization is followed by HDFS, where any user or an application can create directories and store files inside these directories. Thus, HDFS's file system namespace hierarchy is similar to most of the other existing file systems, where one can create and delete files or relocate a file from one directory to another, or even rename a file. In general, HDFS does not support hard links or soft links, though these can be implemented if need arise.

6. Metadata: Metadata is data that describes other data. Meta is a prefix that in most information technology usages means "an underlying definition or description." Metadata summarizes basic information about data, which can make finding and working with particular instances of data easier.

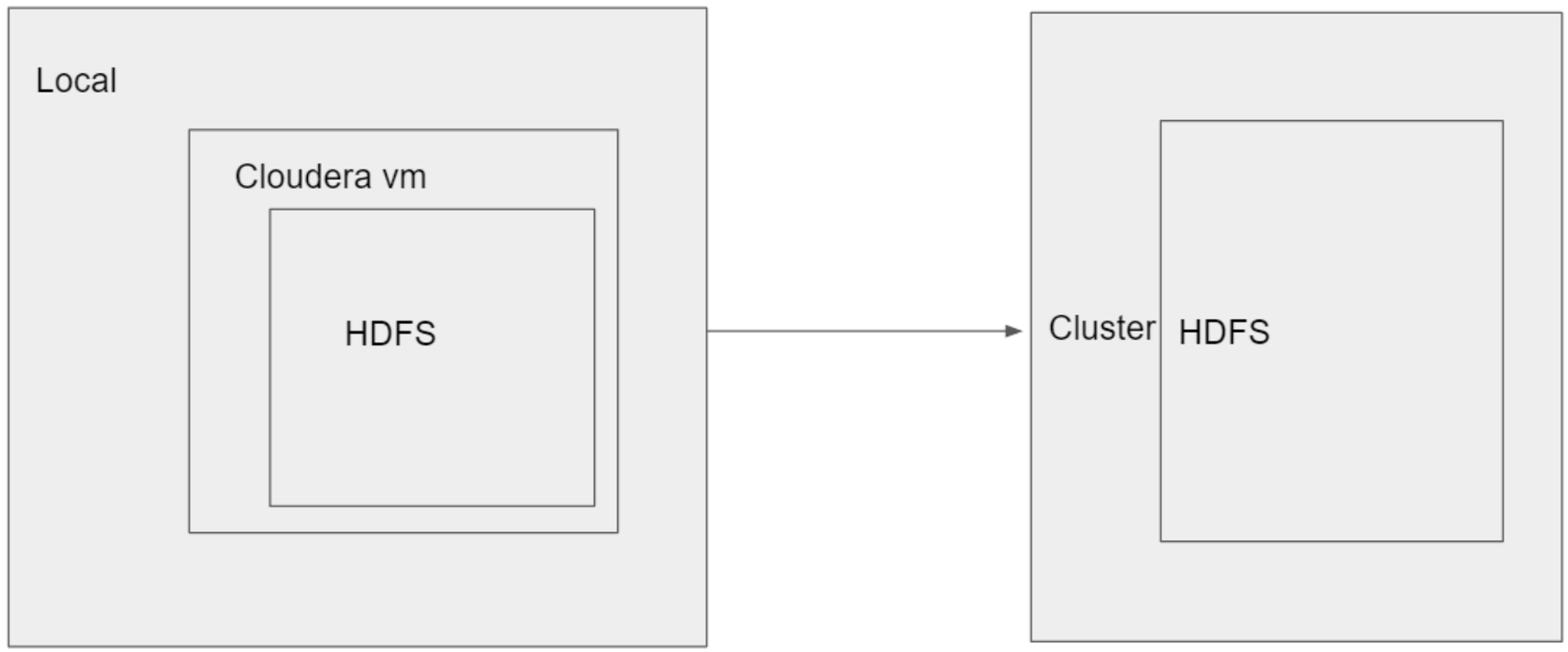
7. Familiar HDFS commands: hadoop fs vs hdfs dfs

8. What is Big Data?
Big Data is nothing but an assortment of such a huge and complex data that it becomes very tedious to capture, store, process, retrieve and analyze it with the help of on-hand database management tools or traditional data processing techniques.

9. Examples of Big Data?
There are many real life examples of Big Data: Facebook is generating 500+ terabytes of data per day, NYSE (New York Stock Exchange) generates about 1 terabyte of new trade data per day, a jet airline collects 10 terabytes of censor data for every 30 minutes of flying time. All these are day to day examples of Big Data!

Week 1: Hadoop Ecosystem & HDFS

10. Local vs HDFS vs cluster



windows: putty, ssh client, winscp

mac/ubuntu: terminal ssh, filezilla

Part I: Batch Processing

Batch data processing is an efficient way of processing high volumes of data is where a group of transactions is collected over a period of time. Data is collected, entered, processed and then the batch results are produced (Hadoop is focused on batch data processing).

Week 2: MapReduce & Yarn

1. What is MapReduce?

MapReduce is a framework for processing big data (huge data sets using a large number of commodity computers). Based on divide-and-conquer principles, it processes the data in two phases namely Map and Reduce phase. This programming model is inherently parallel and can easily process large-scale data with the commodity hardware itself. It is highly integrated with Hadoop distributed file system for processing distributed across data nodes of clusters.

2. What are 'maps' and 'reduces'?

'Maps' and 'Reduces' are two phases of solving a query in HDFS. 'Map' is responsible to read data from input location, and based on the input type, it will generate a key value pair, that is, an intermediate output in local machine.'Reducer' is responsible to process theintermediateoutputreceived from the mapper and generate the final output.

3. Where MapReduce works?

(1) The calculations should be possible to run the calculation on a subset of data, and merge partial results.

(2) The dataset size is big enough (or the calculations are long enough) that the infrastructure overhead of splitting it up for independent computations and merging results will not hurt overall performance.

4. Where MapReduce does NOT work?

(1) NOT applicable, in scenarios when the dataset has to be accessed randomly to perform the operation.

(2) NOT applicable for general recursive problems because computation of the current value requires the knowledge of the previous one, which means that you can't break it apart to sub computations that can be run independently.

Week 2: MapReduce & Yarn

5. What are the four basic parameters of a mapper and a reducer?
The four basic parameters of a mapper areLongWritable, text, text and IntWritable. The first two represent input parameters and the second two represent intermediate output parameters. The four basic parameters of a reducer aretext, IntWritable, text, IntWritable.The first two represent intermediate output parameters and the second two represent final output parameters.

6. What does a split do?
Before transferring the data from hard disk location to map method, there is a phase or method called the 'Split Method'. Split method pulls a block of data from HDFS to the framework. The Split classdoes not write anything, but reads data from the block and pass it to the mapper.Be default, Split is taken care by the framework. Split method is equal to the block size and is used to divide block into bunch of splits.

7. What does a MapReduce partitioner do?
A MapReduce partitioner makes sure that all the value of a single key goes to the same reducer, thus allows evenly distribution of the map output over the reducers. It redirects the mapper output to the reducer by determining which reducer is responsible for a particular key.
A partitioner works like a condition in processing an input dataset. The partition phase takes place after the Map phase and before the Reduce phase. The number of partitioners is equal to the number of reducers. That means a partitioner will divide the data according to the number of reducers. Therefore, the data passed from a single partitioner is processed by a single Reducer.

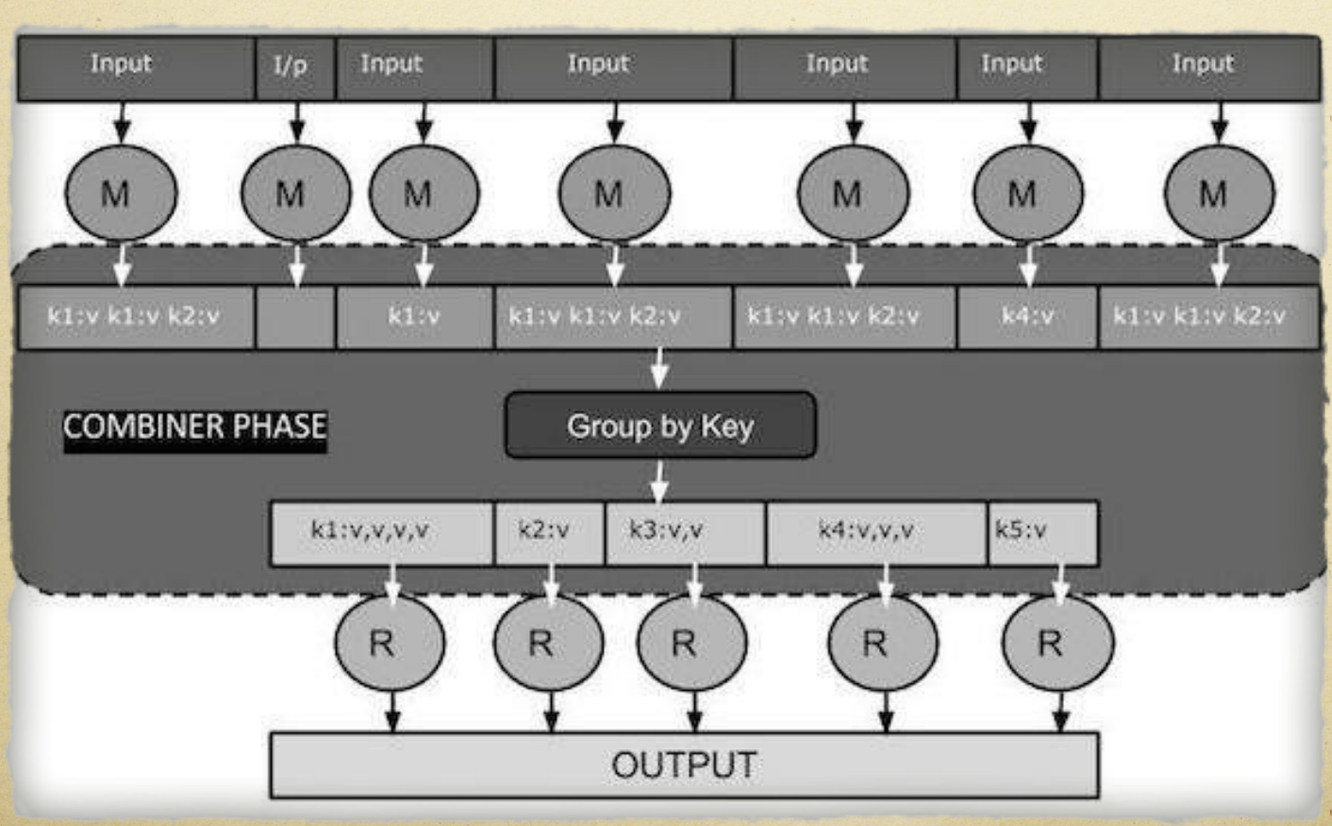
Week 2: MapReduce & Yarn

8. What is a Combiner?

A Combiner, also known as a semi-reducer, is an optional class that operates by accepting the inputs from the Map class (the mapper on a particular node) and thereafter passing the output key-value pairs to the Reducer class.

The main function of a Combiner is to summarize the map output records with the same key. The output (key-value collection) of the combiner will be sent over the network to the actual Reducer task as input.

The Combiner class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce. Usually, the output of the map task is large and the data transferred to the reduce task is high.



Week 2: MapReduce & Yarn

9. What is the difference between an HDFS Block and Input Split?
HDFS Block is the physical division of the data and Input Split is the logical division of the data.

10. What are data serialization and deserialization?
Serialization is the process of converting object data into byte stream data for transmission over a network across different nodes in a cluster or for persistent data storage. Deserialization is the reverse process of serialization and converts byte stream data into object data for reading data from HDFS. Hadoop provides Writables for serialization and deserialization purpose.

11. What are the constraints to Key and Value classes in MapReduce ?
Any data type that can be used for a Value field in a mapper or reducer must implement org.apache.hadoop.io.Writable Interface to enable the field to be serialized and
deserialized. By default Key fields should be comparable with each other. So, these must implement hadoop's org.apache.hadoop.io.WritableComparable interface which in turn extends Hadoop's Writable interface and java.lang.Comparable interfaces.

12. What are the main components of MapReduce Job ?
(1) Main driver class which provides job configuration parameters.
(2) Mapper class which must extend org.apache.hadoop.mapreduce.Mapper class and provide implementation for map () method.
(3) Reducer class which should extend org.apache.hadoop.mapreduce.Reducer class.

Week 2: MapReduce & Yarn

13. MapReduce Terms - 1

Term	Meaning
Job	The whole process to execute: the input data, the mapper and reducers execution and the output data
Task	Every job is divided among the several mappers and reducers; a task is the job portion that goes to every single mapper and reducer
Split	The input file is split into several splits (the suggested size is the HDFS block size, 64Mb)
Record	The split is read from mapper by default a line at the time: each line is a record. Using a class extending `FileInputFormat`, the record can be composed by more than one line
Partition	The set of all the key-value pairs that will be sent to a single reducer. The default partitioner uses an hash function on the key to determine to which reducer send the data

Week 2: MapReduce & Yarn

14. MapReduce Terms - 2

(1) Shuffle: After the first map tasks have completed, the nodes may still be performing several more map tasks each. But they also begin exchanging the intermediate outputs from the map tasks to where they are required by the reducers. This process of moving map outputs to the reducers is known as shuffling.

(2) Sort: Each reduce task is responsible for reducing the values associated with several intermediate keys. The set of intermediate keys on a **single node** is automatically sorted by Hadoop before they are presented to the Reducer.
Note: shuffling and sorting are performed locally, by each reducer, for its own input data, whereas partitioning is not local.

(3) Jobtracker: The primary function of the job tracker is resource management (managing the task trackers), tracking resource availability and task life cycle management (tracking its progress, fault tolerance etc.)

(4) Tasktracker: The task tracker has a simple function of following the orders of the job tracker and updating the job tracker with its progress status periodically. The task tracker is pre-configured with a number of slots indicating the number of tasks it can accept. When the job tracker tries to schedule a task, it looks for an empty slot in the tasktracker running on the same server which hosts the datanode where the data for that task resides. If not found, it looks for the machine in the same rack. There is no consideration of system load during this allocation.

Week 2: MapReduce & Yarn

(5) Scheduler: three schedulers are available in YARN: the FIFO, Capacity, and Fair Schedulers.

a. FIFO Scheduler: The FIFO Scheduler places applications in a queue and runs them in the order of submission(first in, first out). Requests for the first application in the queue are allocated first; once its requests have been satisfied, the next application in the queue is served, and so on.

The FIFO Scheduler has the merit of being simple to understand and not needing any configuration, but it's not suitable for shared clusters. Large applications will use all the resources in a cluster, so each application has to wait its turn. On a shared cluster it is better to use the Capacity Scheduler or the Fair Scheduler.

b. Capacity Scheduler: a separate dedicated queue allows the small job to start as soon as it is submitted, although this is at the cost of overall cluster utilization since the queue capacity is reserved for jobs in that queue. This means that the large job finishes later than when using the FIFO Scheduler.

c. Fair Scheduler: there is no need to reserve a set amount of capacity, since it will dynamically balance resources between all running jobs. Just after the first (large) job starts, it is the only job running, so it gets all the resources in the cluster. When the second (small) job starts, it is allocated half of the cluster resources

so that each job is using its fair share of resources.

Week 2: MapReduce & Yarn

15. What is YARN?

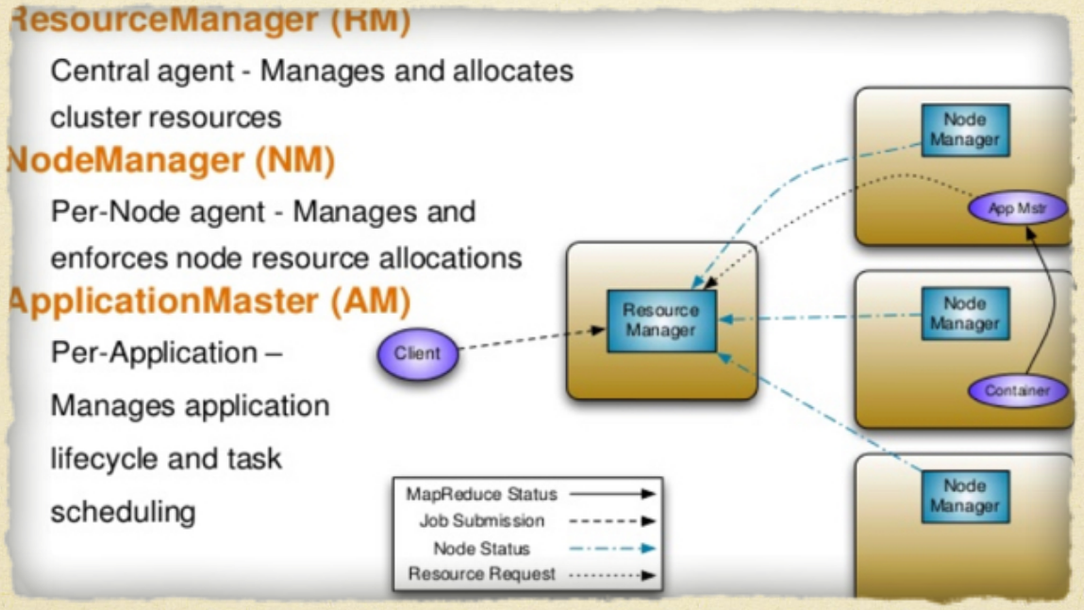
(1) YARN stands for Yet Another Resource Negotiator which is also called as Next generation Mapreduce or Mapreduce 2 or MRv2.

(2) The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons. The idea is to have a global ResourceManager (RM) and per-application ApplicationMaster (AM).

(3) In Yarn, the job tracker is split into two different daemons called Resource Manager and Node Manager (node specific). The resource manager only manages the allocation of resources to the different jobs apart from comprising a scheduler which just takes care of the scheduling jobs without worrying about any monitoring or status updates. Different resources such as memory, cpu time, network bandwidth etc. are put into one unit called the Resource Container. There are different AppMasters running on different nodes which talk to a number of these resource containers and accordingly update the Node Manager with the monitoring/status details.

Week 2: MapReduce & Yarn

(4) Yarn Architecture



(5) A comparison of MapReduce 1 and YARN components

MapReduce 1	YARN
Jobtracker	Resource manager, application master, timeline server
Tasktracker	Node manager
Slot	Container

Week 2: MapReduce & Yarn

16. Why was Yarn introduced and what are the benefits of using Yarn?

YARN was designed to address many of the limitations in MapReduce 1. The benefits to using YARN include the following:

(1) Scalability: YARN can run on larger clusters than MapReduce 1. MapReduce 1 hits scalability bottlenecks in the region of 4,000 nodes and 40,000 tasks, stemming from the fact that the jobtracker has to manage both jobs and tasks. YARN overcomes these limitations by virtue of its split resource manager/application master architecture: it is designed to scale up to 10,000 nodes and 100,000 tasks. In contrast to the jobtracker, each instance of an application —here, a MapReduce job—has a dedicated application master, which runs for the duration of the application. This model is actually closer to the original Google MapReduce paper, which describes how a master process is started to coordinate map and reduce tasks running on a set of workers.

(2) Availability: High availability (HA) is usually achieved by replicating the state needed for another daemon to take over the work needed to provide the service, in the event of the service daemon failing. However, the large amount of rapidly changing complex state in the jobtracker's memory (each task status is updated every few seconds, for example) makes it very difficult to retrofit HA into the jobtracker service. With the jobtracker's responsibilities split between the resource manager and application master in YARN, making the service highly available became a divide-and-conquer problem: provide HA for the resource manager, then for YARN applications(on a per-application basis). And indeed, Hadoop 2 supports HA both for the resource manager and for the application master for MapReduce jobs.

Week 2: MapReduce & Yarn

(3) Utilization

In MapReduce 1, each tasktracker is configured with a static allocation of fixed-size “slots,” which are divided into map slots and reduce slots at configuration time. A map slot can only be used to run a map task, and a reduce slot can only be used for a reduce task.

In YARN, a node manager manages a pool of resources, rather than a fixed number of designated slots. MapReduce running on YARN will not hit the situation where a reduce task has to wait because only map slots are available on the cluster, which can happen in MapReduce 1. If the resources to run the task are available, then the application will be eligible for them.

Furthermore, resources in YARN are fine grained, so an application can make a request for what it needs, rather than for an indivisible slot, which may be too big (which is wasteful of resources) or too small (which may cause a failure) for the particular task.

(4) Multitenancy

In some ways, the biggest benefit of YARN is that it opens up Hadoop to other types of distributed application beyond MapReduce. MapReduce is just one YARN application among many.

It is even possible for users to run different versions of MapReduce on the same YARN cluster, which makes the process of upgrading MapReduce more manageable.

Week 3: Pig

1. What is Pig?

Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

2. What Pig Can Do ?

•Extract-transform-load (ETL) data pipelines
•Research on raw data
•Iterative data processing
•Data Analysis

3. Interactive vs Batch Mode

•Run scripts
–pig -x local file.pig
–pig -x tez_local file.pig
–pig -x mapreduce file.pig
–pig -x tez file.pig
•Grunt shell
–pig -x local
–Pig -x mapreduce
–Pig -x tez
•PigServer - Called by Java

Week 3: Pig

4. What is Tez?

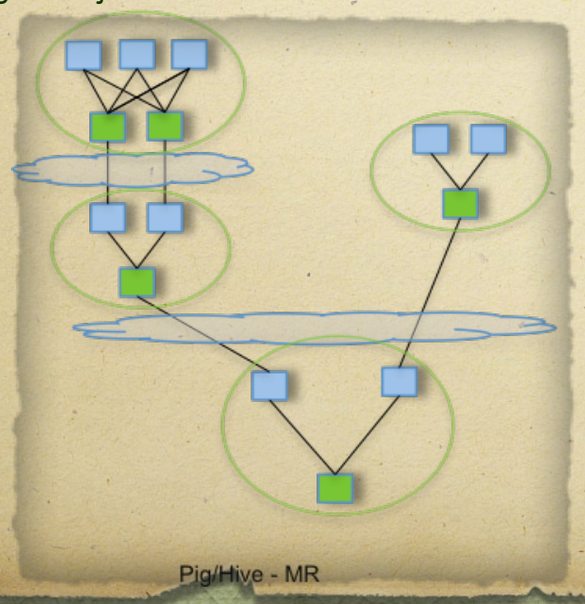
The Apache Tez™ project is aimed at building an application framework which allows for a complex directed-acyclic-graph of tasks for processing data.

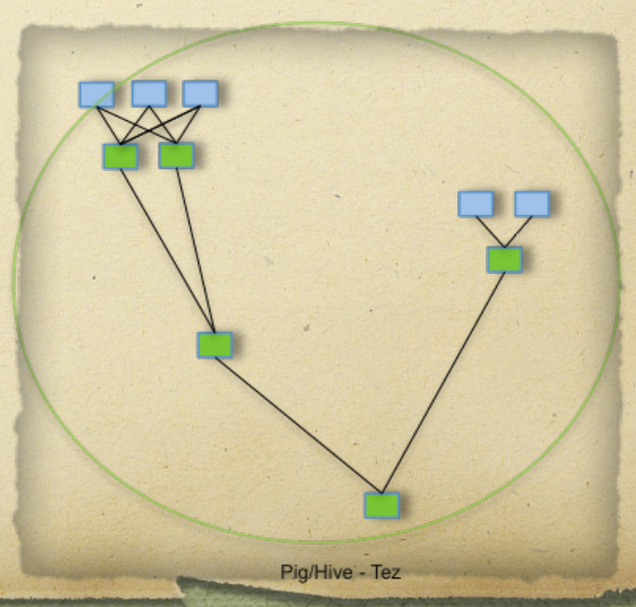
It is currently built atop Apache Hadoop YARN.

The 2 main design themes for Tez are:

- Empowering end users by:
  - Expressive dataflow definition APIs
  - Flexible Input-Processor-Output runtime model
  - Data type agnostic
  - Simplifying deployment
- Execution Performance
  - Performance gains over Map Reduce
  - Optimal resource management
  - Plan reconfiguration at runtime
  - Dynamic physical data flow decisions

By allowing projects like Apache Hive and Apache Pig to run a complex DAG of tasks, Tez can be used to process data, that earlier took multiple MR jobs, now in a single Tez job as shown below.





Week 3: Pig

5. bag, tuple, field

• A relation is a bag (more specifically, an outer bag).

• A bag is a collection of tuples.

• A tuple is an ordered set of fields.

• A field is a piece of data.

```
A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int, gpa:float);
DUMP A;
(John,18,4.0F)
(Mary,19,3.8F)
(Bill,20,3.9F)
(Joe,18,3.8F)
```

6. Loading & Save Data

•LOAD

•DUMP

•STORE

Week 3: Pig

7. Working with Data

- FILTER
- DISTINCT
- FOREACH GENERATE
- SAMPLE
- GROUP
- JOIN
- ORDER
- LIMIT
- UNION
- SPLIT

8. Debug

- SAMPLE
- DUMP
- DESCRIBE
- EXPLAIN
- ILLUSTRATE

Week 3: Pig

9. Piggybank - User Defined Pig Functions

REGISTER ‘yourPath/piggybank.jar';

10. Optimization

•Use TEZ

•Filter as early as possible

•Project as early as possible

•Avoid JOIN

11. Avro

Avro is an Apache™ open source project that provides data serialization and data exchange services for Hadoop®. These services can be used together or independently.

Week 3: Pig

12. Pig word count demo code:
WordCount:

```
input = load '/input/data' as (line:chararray);

words = foreach input generate flatten(TOKENIZE(line)) as word;

grpd = group words by word;

cntd = foreach grpd generate COUNT(words), group;

dump cntd;

store d into '/output/pig_wordcount';
```

13. Yahoo finance demo code:
See script

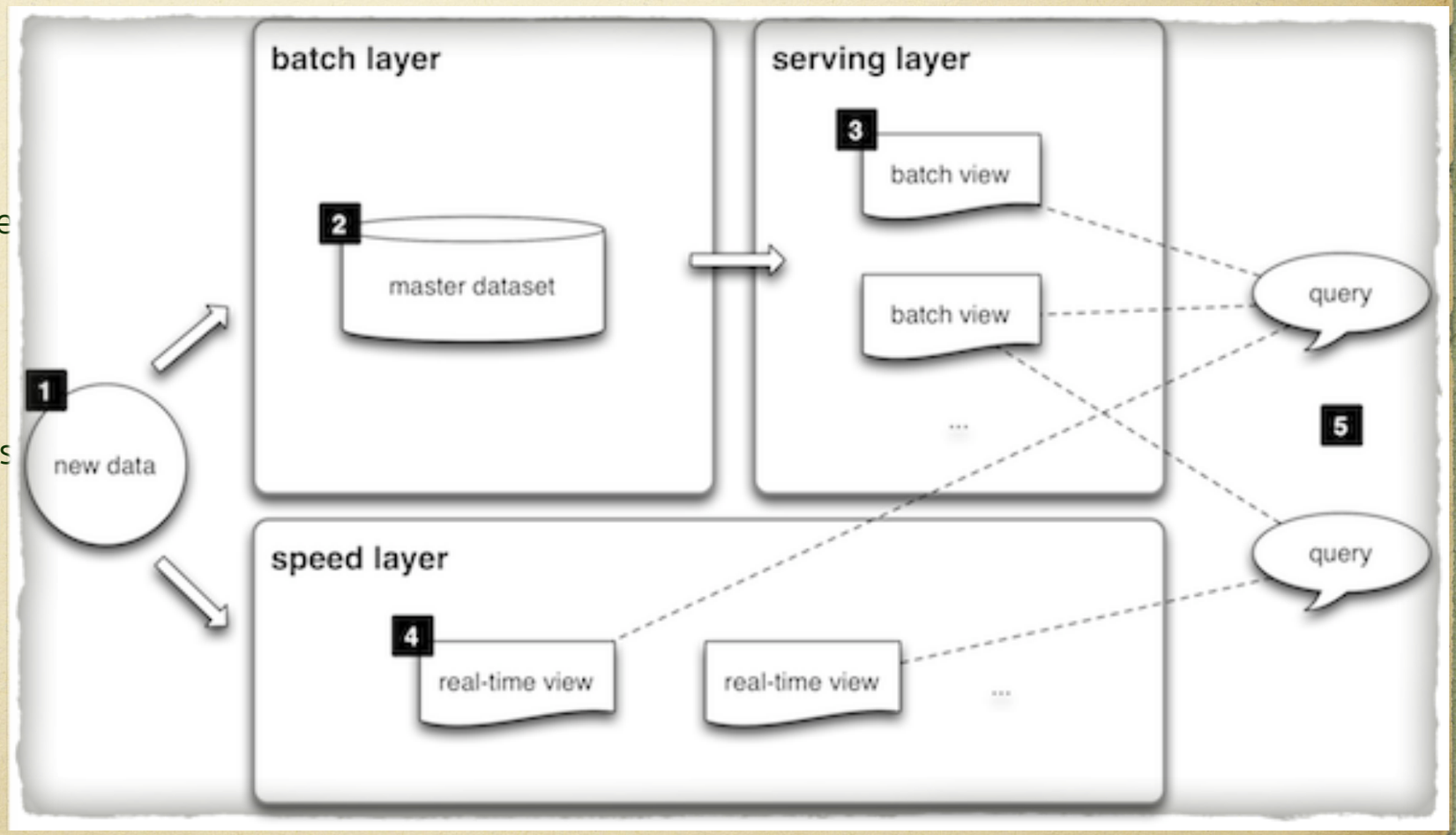
Week 3: Pig

14. What is the Lambda Architecture?

Nathan Marz came up with the term Lambda Architecture (LA) for a generic, scalable and fault-tolerant data processing architecture, based on his experience working on distributed data processing systems at Backtype and Twitter.

The LA aims to satisfy the needs for a robust system that is fault-tolerant, both against hardware failures and human mistakes, being able to serve a wide range of workloads and use cases, and in which low-latency reads and updates are required. The resulting system should be linearly scalable, and it should scale out rather than up.

Here's how it looks like, from a high-level perspective:



Week 3: Pig

14. What is the Lambda Architecture?

(1) All data entering the system is dispatched to both the batch layer and the speed layer for processing.

(2) The batch layer has two functions: (i) managing the master dataset (an immutable, append-only set of raw data), and (ii) to pre-compute the batch views.

(3) The serving layer indexes the batch views so that they can be queried in low-latency, ad-hoc way.

(4) The speed layer compensates for the high latency of updates to the serving layer and deals with recent data only.

(5) Any incoming query can be answered by merging results from batch views and real-time views.

Week 4: Hive

1. What is Hive?

Apache Hive is a data warehouse infrastructure to store structured data built on top of Hadoop for providing data summarization, query, and analysis.

Hive has three main functions: data summarization, query and analysis. It is designed for OLAP, supports queries expressed in a language called HiveQL, which automatically translates SQL-like queries into MapReduce jobs executed on Hadoop. In addition, HiveQL supports custom MapReduce scripts to be plugged into queries. Hive also enables data serialization/deserialization and increases flexibility in schema design by including a system catalog called Hive-Metastore. Hive is Familiar, fast, scalable and extensible.

2. Why Hive? (Motivation)

(1) MapReduce has limitations

- Have to use M/R model
- Not Reusable
- Long development type / overhead
- For complex jobs: Multiple M/R stages

(2) Bright side:

- MapReduce is scalable
- SQL has huge user base
- SQL is easy to code

=> Solution: Combine MapReduce and SQL

Week 4: Hive

3. What Hive is NOT?

(1) Not work with small data set (high latency)

(2) Not designed for online transaction processing

(3) Not offer real-time queries

(4) Not work as row level query

Hive is best used for batch jobs over large sets of append-only data (like web logs).

4. Files Hive supports

Hive supports text files (also called flat files), SequenceFiles (flat files consisting of binary key/value pairs) and RCFiles (Record Columnar Files which store columns of a table in a columnar database way.)

5. Hive Execution

(1) 3 ways:

- CLI: Hive CLI & Beeline
- Web UI: Hue
- JDBC/ODBC

(2) Run one query

```
hive -e 'SELECT DISTINCT username FROM temp.TwitterExampletextexample LIMIT 10'
```

(3) Run a hive query file

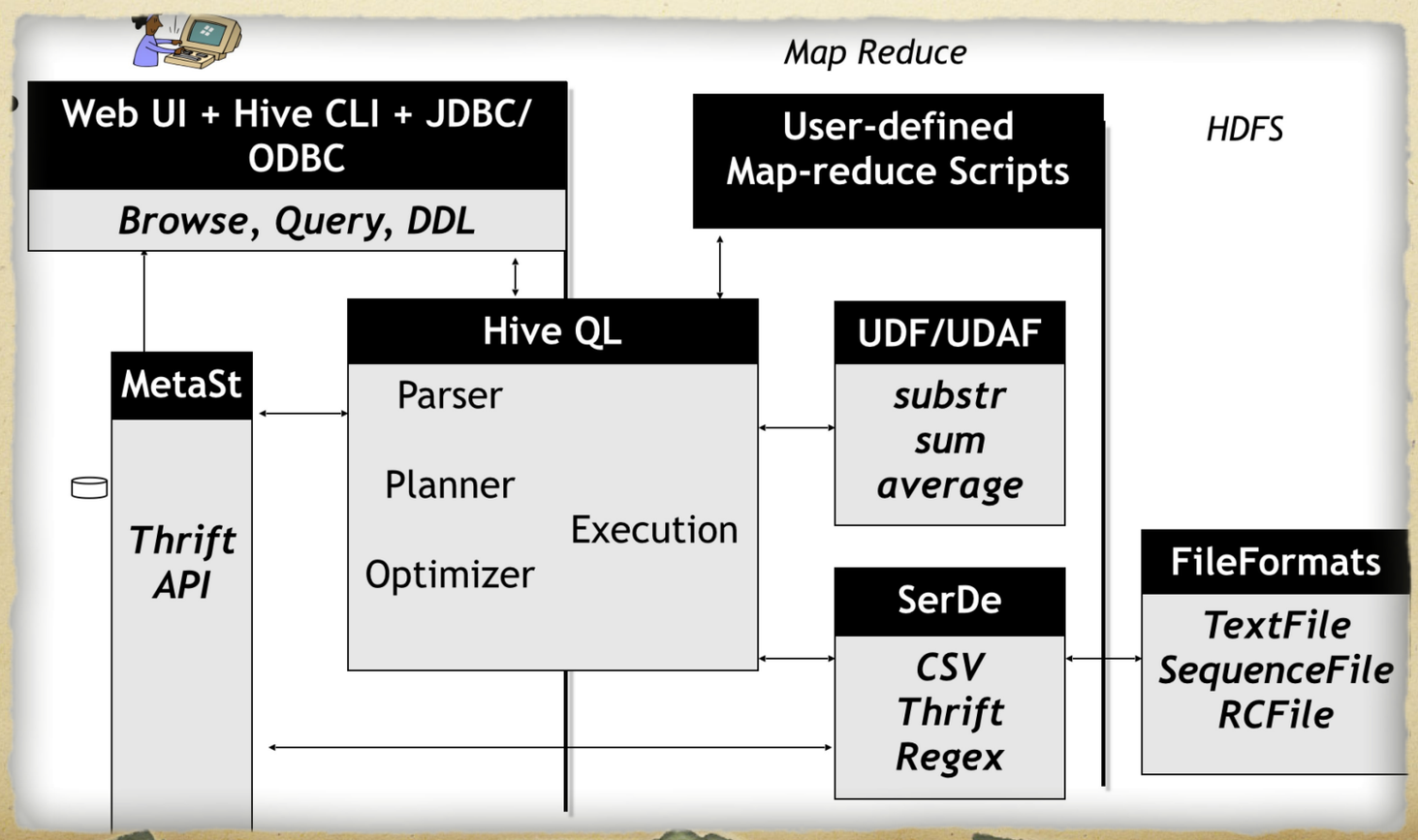
```
hive -f /tmp/demo_hive.sql
```

Week 4: Hive

6. Hive Architecture

Primarily The diagram represents CLI (Command Line Interface), JDBC/ODBC and Web GUI (Web Graphical User Interface). This represents when user comes with CLI(Hive Terminal) it directly connected to Hive Drivers,When User comes with JDBC/ODBC(JDBC Program) at that time by using API(Thrift Server) it connected to Hive driver and when the user comes with Web GUI(Ambari server) it directly connected to Hive Driver.

The hive driver receives the tasks(Queries) from user and send to Hadoop architecture.The Hadoop architecture uses name node,data node,job tracker and task tracker for receiving and dividing the work what Hive sends to Hadoop.



Week 4: Hive

7. Metastore & Metadata

(1) The Hive metastore service stores the metadata for Hive tables and partitions in a relational database, and provides clients (including Hive) access to this information via the metastore service API.

(2) Database namespace

Table definitions

- Schema info, physical location on HDFS

Partition data

ORM framework

- All the metadata can be stored in Derby by default
- Any database with JDBC can be configured

8. Data Model

• Tables

- Basic type columns(int, float, boolean)
- Complex type: list/map(associate array)

• Partitions

• Buckets

```
CREATE TABLE sales(
    id INT,
    items ARRAY<STRUCT<id:INT,name:STRING>
)
PARITIONED BY (ds STRING)
CLUSTERED BY (id) INTO 32 BUCKETS;

SELECT id FROM sales TABLESAMPLE (BUCKET 1 OUT OF 32);
```

Week 4: Hive

9. Data Type

(1) primitive

TINYINT

SMALLINT

INT

BIGINT

BOOLEAN

FLOAT

DOUBLE

BIGDECIMAL

STRING

BINARY

TIMESTAMP

(2) Complex Data Type

arrays: ARRAY<data_type>

maps: MAP<primitive_type, data_type>

structs: STRUCT<col_name : data_type [COMMENT col_comment], ...>

union: UNIONTYPE<data_type, data_type, ...>

```
CREATE TABLE test(col1 UNIONTYPE<INT, DOUBLE, ARRAY<VARCHAR>, STRUCT<a:INT,b:CHAR>>);

SELECT col1 FROM test LIMIT 1;

{0:1}                   // Matching INT types
{1:10.0}                  // Matching DOUBLE types
{2:["hello","world"]}       // Matching ARRAY with VARCHAR type
{3:{"a":50,"b":"Good"}}     // Matching STRUCT with INT & CHAR types
```

Week 4: Hive

10. Hive Database Features

- All about files
- Schema on read
- Fast when load data into DB
- Touch data only when run query
- Don't support delete/update

11. Internal Table vs External Table

(1) External table in HIVE (stores data on HDFS)

• External table stores files on the HDFS server but tables are not linked to the source file completely.

• If you delete an external table the file still remains on the HDFS server.

• The file and the table link is there but read only.

• As an example if you create an external table called "amandeep_test" in HIVE using HIVE-QL and link the table to file "flat_file.txt", then deleting "amandeep_test" from HIVE will not delete "flat_file.txt" from HDFS.

• External table files are accessible to anyone who has access to HDFS file structure and therefore security needs to be managed at the HDFS file/folder level.

•Meta data is maintained on master node and deleting an external table from HIVE, only deletes the metadata not the data/file.

Use external table if you:

• Want to manage the data outside HIVE e.g. you are planning to use an ETL tool to load/merge data files etc.

• Want to load the latest information to the table but still want to retain old dataset in a file on HDFS for regulatory/legal purposes.

• Are not planning to create a table from another table schema e.g. Create table1 as (Select * from table2)

Week 4: Hive

(2) Internal table in HIVE (stores data on HDFS but in a kind of restricted area)

• Stored in a directory based on settings in the following file: hive.metastore.warehouse.dir

by default internal tables are stored in the following directory “ /user/hive/warehouse” you can change it by updating the location in the config file mentionedabove.

e.g. in the following screen shot internal table “ tbl_batting“ is stored as a file on HDFSin warehouse folder.

• Deleting the table deletes the metadata &data from masternode and HDFS respectively

• Security needs to be managed within HIVE, probably at the schema level (depends on organisation to organisation). HDFS security is out of scope in this case.

Use internal table if you:

• Want to store the data temporary.

• Want to use HIVE to manage the lifecycle of tables and data.

12. Beeline

In its original form, Apache Hive was a heavyweight command-line tool that accepted queries and executed them utilizing MapReduce. Later, the tool split into a client-server model, in which HiveServer1 is the server (responsible for compiling and monitoring MapReduce jobs) and Hive CLI is the command-line interface (sends SQL to the server).

Later, the Hive community (with Cloudera engineers leading the charge) introduced HiveServer2, an enhanced Hive server designed for multi-client concurrency and improved authentication that also provides better support for clients connecting through JDBC and ODBC. Now HiveServer2, with Beeline as the command-line interface, is the recommended solution; HiveServer1 and Hive CLI are deprecated and the latter won’t even work with HiveServer2.

Week 4: Hive

13. High Performance Hive

- Use the ORC File Format
- Use the Tez Query Execution Engine
- Use Column Statistics and the Cost-Based Optimizer
- Use proper Compression techniques
- Better Workload Management by Using Queues

14. ORC file format

(1) The Optimized Row Columnar (ORC) file format provides a highly efficient way to store Hive data. It was designed to overcome limitations of the other Hive file formats. Using ORC files improves performance when Hive is reading, writing, and processing data.

(2) File Structure

An ORC file contains groups of row data called stripes, along with auxiliary information in a file footer. At the end of the file a postscript holds compression parameters and the size of the compressed footer.

The default stripe size is 250 MB. Large stripe sizes enable large, efficient reads from HDFS.

The file footer contains a list of stripes in the file, the number of rows per stripe, and each column's data type. It also contains column-level aggregates count, min, max, and sum.

(3) Stripe Structure

As shown in the diagram, each stripe in an ORC file holds index data, row data, and a stripe footer.

The stripe footer contains a directory of stream locations. Row data is used in table scans.

Index data includes min and max values for each column and the row positions within each column. (A bit field or bloom filter could also be included.) Row index entries provide offsets that enable seeking to the right compression block and byte within a decompressed block. Note thatORC indexes are used only for the selection of stripes and row groups and not for answering queries.

Week 4: Hive

15. HiveQL and Spark

```
import org.apache.spark.sql.hive.HiveContext
val hiveContext = new HiveContext(sc)
val results = hiveContext.sql(”…”)
//  prep
import org.apache.spark.sql.hive.HiveContext
val sc = new SparkContext(new SparkConf().setAppName(this.getClass.getName))
val hiveContext = new HiveContext(sc)

//  Create a table
hiveContext.sql("create table ratings ( ”
"user_id int, ”
"movie_id int, ”
"rating int, ”
"ts bigint) ”
"row format delimited ”
"fields terminated by ’*’ ”
"lines terminated by ’\n’ ”
"stored as textfile”)

//  load data into a table
hiveContext.sql("load data inpath ’/data/movielens_1m_simple/ratings/ratings.dat’ ”
"into table ratings")

//query on this table
Val genres = hiveContext.sql("select explode(genres) as genre ”
"from movies")
genres.take(10)
```

Week 4: Hive

16. Hive use cases

- Log processing
  - Daily Report
  - User Activity Measurement
- Data/Text mining
  - Machine learning (Training Data)
- Business intelligence
  - Advertising Delivery
  - Spam Detection
- Predictive Modeling, Hypothesis Testing

17. Hive code demo

(1) word count demo

```
DROP TABLE IF EXISTS doc;
-- a) create table to load whole file
CREATE TABLE doc(text STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\n' STORED AS textfile;
--b) loads plain text file
--if file is .csv then in replace '\n' by ',' in step no 1 (creation of doc table)
LOAD DATA INPATH '/user/randy01/demo/mapreduce/wordcount/input/big/hive/wordcount/' OVERWRITE INTO TABLE doc;
-- c) wordCount in single line
SELECT word, COUNT(*) AS cnt FROM doc LATERAL VIEW explode(split(text, ' ')) lTable AS word GROUP BY word ORDER BY cnt DESC LIMIT 200;
```

(2) Yahoo finance demo

Week 4: Hive

18. Pig vs Hive

Pig hadoop and Hive hadoop have a similar goal- they are tools that ease the complexity of writing complex java MapReduce programs. The data that gets loaded to Hive is usually a processed, cleansed, standardized, and structured data. All these processing and transformations are usually done using pig or mapreduce. One may also use Hive for this depending upon the structure of data.

There differences are:

Pig	Hive
Pig has a procedural data flow language (Pig Latin)	Hive has a declarative SQLish language (HiveQL)
Pig is mainly used for programming	Hive is mainly used for creating reports
Pig is generally used by researchers and programmers	Hive is mainly used by data analysts
Pig operates on the client side of any cluster	Hive operates on the server side of any cluster
Pig does not have a dedicated metadata database and the schema or data types will be defined in the script itself	Hive makes use of exact variation of the SQL DLL language by defining the tables beforehand and storing the schema details in any local
Pig supports Avro	Hive does not support Avro
Pig is also SQL-like but varies to great extent and thus it will take some time efforts to master pig	Hive directly leverage SQL expertise and thus can be learnt easily

Week 4: Hive

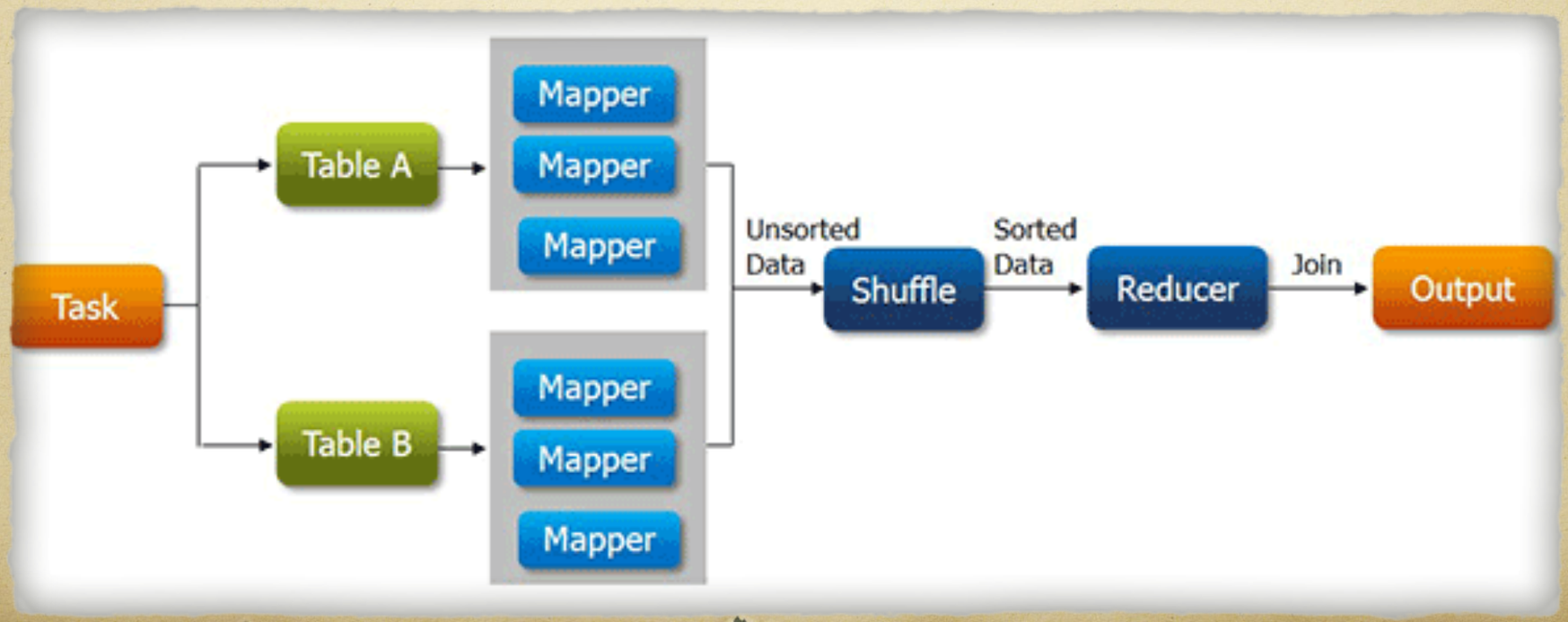
19. Map Side Join Vs. Join

(1) Join is a clause that combines the records of two tables (or Data-Sets).

Assume that we have two tables A and B. When we perform join operation on them, it will return the records which are the combination of all columns o f A and B.

(2) the functionality of normal join with an example..

Whenever, we apply join operation, the job will be assigned to a Map Reduce task which consists of two stages- a 'Map stage' and a 'Reduce stage'. A mapper's job during Map Stage is to "read" the data from join tables and to "return" the 'join key' and 'join value' pair into an intermediate file. Further, in the shuffle stage, this intermediate file is then sorted and merged. The reducer's job during reduce stage is to take this sorted result as input and complete the task of join.



Week 4: Hive

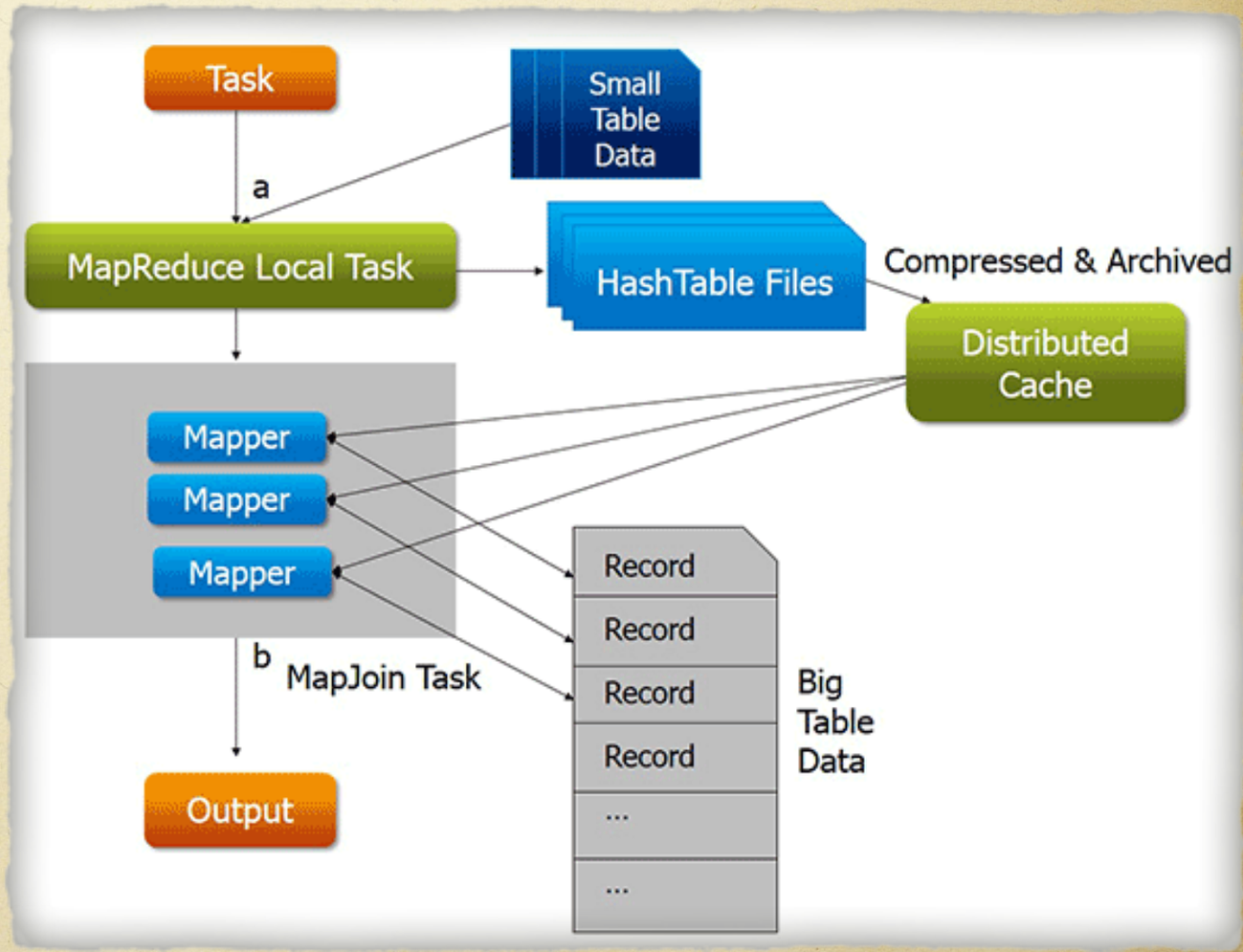
19. Map Side Join Vs. Join

(3)
- Map-side Join is similar to a join but all the task will be performed by the mapper alone.
- The Map-side Join will be mostly suitable for small tables to optimize the task.

(4) How will the map-side join optimize the task?

Assume that we have two tables of which one of them is a small table. When we submit a map reduce task, a Map Reduce local task will be created before the original join Map Reduce task which will read data of the small table from HDFS and store it into an in-memory hash table. After reading, it serializes the in-memory hash table into a hash table file.

In the next stage, when the original join Map Reduce task is running, it moves the data in the hash table file to the Hadoop distributed cache, which populates these files to each mapper's local disk. So all the mappers can load this persistent hash table file back into the memory and do the join work as before. The execution flow of the optimized map join is shown in the figure below. After optimization, the small table needs to be read just once. Also if multiple mappers are running on the same machine, the distributed cache only needs to push one copy of the hash table file to this machine.



Week 4: Hive

19. Map Side Join Vs. Join

(5) Advantages of using map side join:

- Map-side join helps in minimizing the cost that is incurred for sorting and merging in the shuffle and reduce stages.
- Map-side join also helps in improving the performance of the task by decreasing the time to finish the task.

(6) Disadvantages of Map-side join: (When should not use Map-side join)

Map side join is adequate only when one of the tables on which you perform map-side join operation is small enough to fit into the memory. Hence it is not suitable to perform map-side join on the tables which are huge data in both of them.

(7) Simple Example for Map Reduce Joins. Let us create two tables:

a. Emp: contains details of an Employee such as Employee name, Employee ID and the Department she belongs to.

b. Dept: contains the details like the Name of the Department, Department ID and so on.

c. Now, let us load the data into the tables.

d. Let us perform the Map-sideJoin on the two tables to extract the list of departments in which each employee is working.

e. Here, the second table dept is a small table. Remember, always the number of department will be less than the number of employees in an organization.

Week 4: Hive

19. Map Side Join Vs. Join

```
hive> select /*+ MAPJOIN(dept) */ emp.name, dept.deptname
    > from emp join dept on emp.deptid=dept.deptid;
Total MapReduce jobs = 1
Execution log at: /tmp/root/root_20130906024141_06bea72b-b254-4cda-bd09-8deda7b8
2080.log
2013-09-06 02:41:36     Starting to launch local task to process map join;     m
aximum memory = 1013645312
2013-09-06 02:41:36     Processing rows:        3       Hashtable size: 3       M
emory usage:    1565592 rate:   0.002
2013-09-06 02:41:36     Dump the hashtable into file: file:/tmp/root/hive_2013-0
9-06_02-41-35_114_6773657074648351797/-local-10002/HashTable-Stage-1/MapJoin-1--
.hashtable
2013-09-06 02:41:36     Upload 1 File to: file:/tmp/root/hive_2013-09-06_02-41-3
5_114_6773657074648351797/-local-10002/HashTable-Stage-1/MapJoin-1--.hashtable F
ile size: 451
2013-09-06 02:41:36     End of local task; Time Taken: 0.33 sec.
Mapred Local Task Succeeded . Convert the Join into MapJoin
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201309060215_0002, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201309060215_0002
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:80
21 -kill job_201309060215_0002
2013-09-06 02:41:38,967 Stage-1 map = 0%,  reduce = 0%
2013-09-06 02:41:40,980 Stage-1 map = 100%,  reduce = 0%
2013-09-06 02:41:41,989 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201309060215_0002
OK
akash   sales team
anand   sales team
pranav  Development team
akshay  Development team
ananth  sales team
Time taken: 6.962 seconds
```

Week 4: Hive

19. Map Side Join Vs. Join

```
hive> select emp.name, dept.deptname from emp join dept on emp.deptid=dept.deptid;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201309060248_0001, Tracking URL = http://localhost:50030/jobdet
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021
2013-09-06 02:49:59,474 Stage-1 map = 0%,  reduce = 0%
2013-09-06 02:50:03,537 Stage-1 map = 100%,  reduce = 0%
2013-09-06 02:50:12,613 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201309060248_0001
OK
pranav  Development team
akshay  Development team
ananth  sales team
akash   sales team
anand   sales team
Time taken: 17.689 seconds
```

Week 4: Hive

19. Map Side Join Vs. Join

(8) Summary:

While executing both the joins, you can find the two differences:

- Map-reduce join has completed the job in less time when compared with the time taken in normal join.
- Map-reduce join has completed its job without the help of any reducer whereas normal join executed this job with the help of one reducer.

Hence, Map-side Join is your best bet when one of the tables is small enough to fit in memory to complete the job in a short span of time.

In Real-time environment, you will be have data-sets with huge amount of data. So performing analysis and retrieving the data will be time consuming if one of the data-sets is of a smaller size. In such cases Map-side join will help to complete the job in less time.

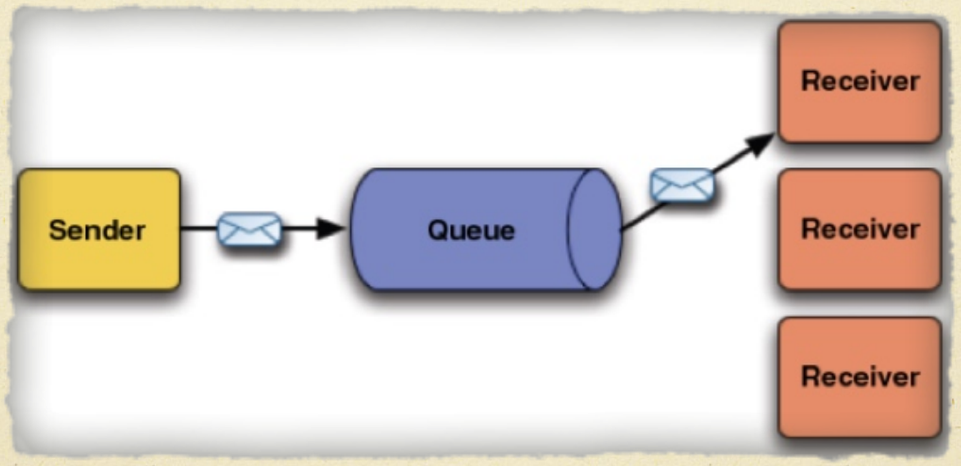
Part II: Real Time Processing

Real time data processing involves a continual input, process and output of data. Data must be processed in a small time period (or near real time).

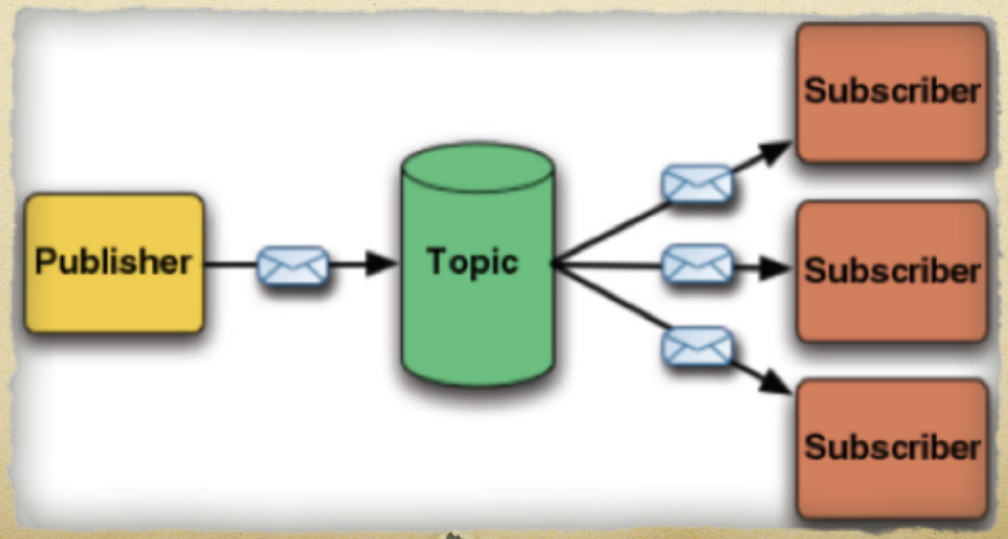
Week 5: Kafka & Storm

0. Before Kafka, let's introduce **Messaging System**:

(1) Point to Point Messaging (Queue)



(2) Publish-Subscribe Messaging (Topic)



Week 5: Kafka & Storm

Part I: Kafka

1. What is Kafka?

Apache Kafka is an open-source message broker project developed by the Apache Software Foundation written in Scala. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Kafka is a distributed publish-subscribe messaging system.

2. Features of Kafka

(1) Fast: A single Kafka broker can handle hundreds of megabytes of reads and writes per second from thousands of clients.

(2) Scalable: Kafka is designed to allow a single cluster to serve as the central data backbone for a large organization. It can be elastically and transparently expanded without downtime. Data streams are partitioned and spread over a cluster of machines to allow data streams larger than the capability of any single machine and to allow clusters of co-ordinated consumers.

(3) Durable: Messages are persisted on disk and replicated within the cluster to prevent data loss. Each broker can handle terabytes of messages without performance impact.

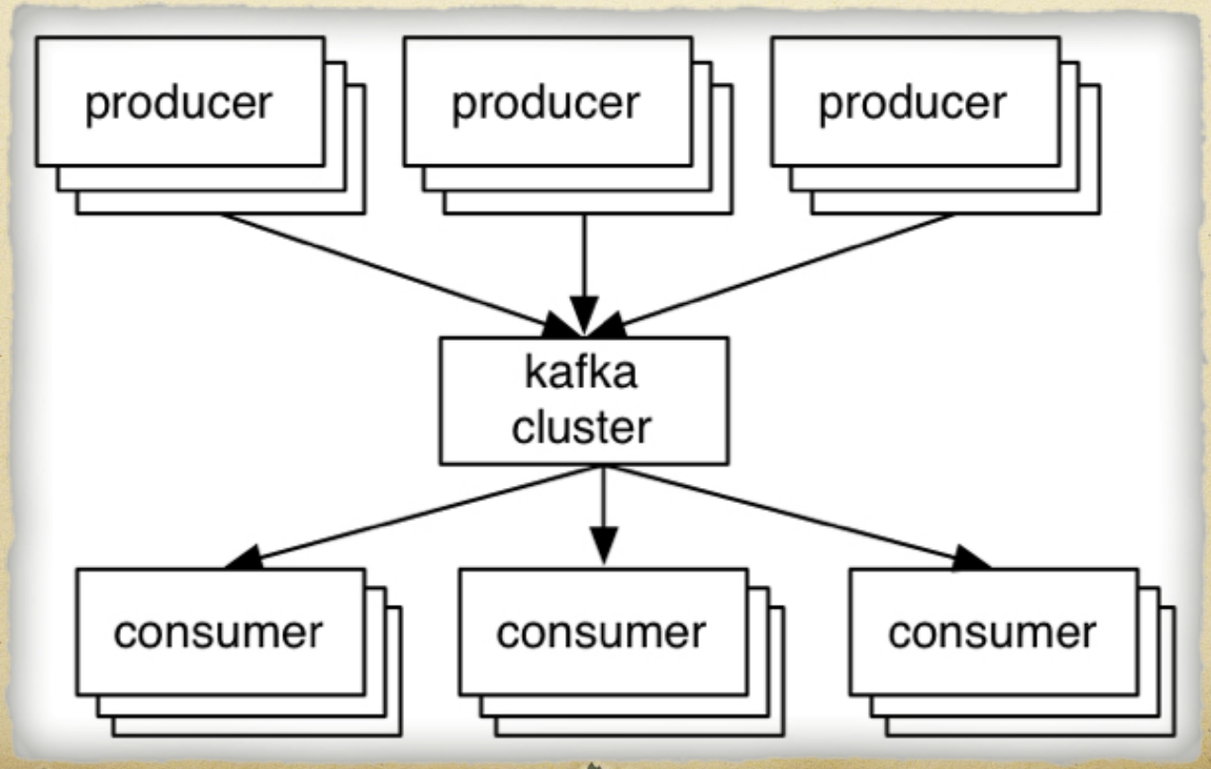
(4) Distributed by Design: Kafka has a modern cluster-centric design that offers strong durability and fault-tolerance guarantees.

Week 5: Kafka & Storm

3. Messaging Component

- Maintains feeds of messages in categories called **topics**
- Call processes that publish messages to a Kafka topic **producers**
- Call processes that subscribe to topics and process the feed of published messages **consumers**
- Run as a cluster comprised of one or more servers each of which is called a **broker**

4. Kafka Abstract



Week 5: Kafka & Storm

5. Major Use Case

- Messaging
- Website Activity Tracking
- Metrics – operational monitoring data
- Log Aggregation
- Stream Processing
- Event Sourcing
- Commit Log

6. kafka command lines

****** list topics
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh --list --zookeeper master1.nmobile.local:2181

****** describe a topic
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh --zookeeper master1.nmobile.local:2181 --describe --topic test_jason_test3

****** Create Kafka topic
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh --zookeeper master1.nmobile.local:2181 --create --topic test_jason_test3 --replication-factor 1 --partitions 2

****** produce some msg for test
/usr/hdp/current/kafka-broker/bin/kafka-console-producer.sh --broker-list master1.nmobile.local:6667 --topic test_jason_test3

****** read the message from consumer
/usr/hdp/current/kafka-broker/bin/kafka-console-consumer.sh --zookeeper master1.nmobile.local:2181 --topic test_jason_test3

Week 5: Kafka & Storm

7. What is Zookeeper?

- An Open source, High Performance coordination service for distributed applications
- Centralized service for
  - Configuration Management
  - Locks and Synchronization for providing coordination between distributed systems
  - Naming service (Registry)
  - Group Membership
- Features
  - hierarchical namespace
  - provides watcher on a znode
  - allows to form a cluster of nodes
- Supports a large volume of request for data retrieval and update.

8. Zookeeper Use cases

- Configuration Management
  - Cluster member nodes Bootstrapping configuration from a central source
- Distributed Cluster Management
  - Node Join/Leave
  - Node Status in real time
- Naming Service – e.g. DNS
- Distributed Synchronization – locks, barriers
- Leader election
- Centralized and Highly reliable Registry

Week 5: Kafka & Storm

9. Zookeeper Data Model

- Hierarchical Namespace
- Each node is called “znode”
- Each znode has data(stores data in byte[] array) and can have children
- znode
  - Maintains “Stat” structure with version of data changes , ACL changes and timestamp
  - Version number increases with each changes

Part II: Storm

1. What is Storm?

Apache Storm is a free and open source distributed realtime computation system. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing.

2. Characteristics of Storm

- Fast – benchmarked as processing one million 100 byte messages per second per node
- Scalable – with parallel calculations that run across a cluster of machines
- Fault-tolerant – when workers die, Storm will automatically restart them. If a node dies, the worker will be restarted on another node.
- Reliable – Storm guarantees that each unit of data (tuple) will be processed at least once or exactly once. Messages are only replayed when there are failures.
- Easy to operate – standard configurations are suitable for production on day one. Once deployed, Storm is easy to operate

Week 5: Kafka & Storm

3. Storm Cluster Components

- Nimbus node(master node, similar to the Hadoop JobTracker):
  - Uploads computations for execution
  - Distributes code across the cluster
  - Launches workers across the cluster
  - Monitors computation and reallocates workers as needed
- ZooKeeper nodes – coordinates the Storm cluster
- Supervisor nodes – communicates with Nimbus through Zookeeper, starts and stops workers according to signals from Nimbus

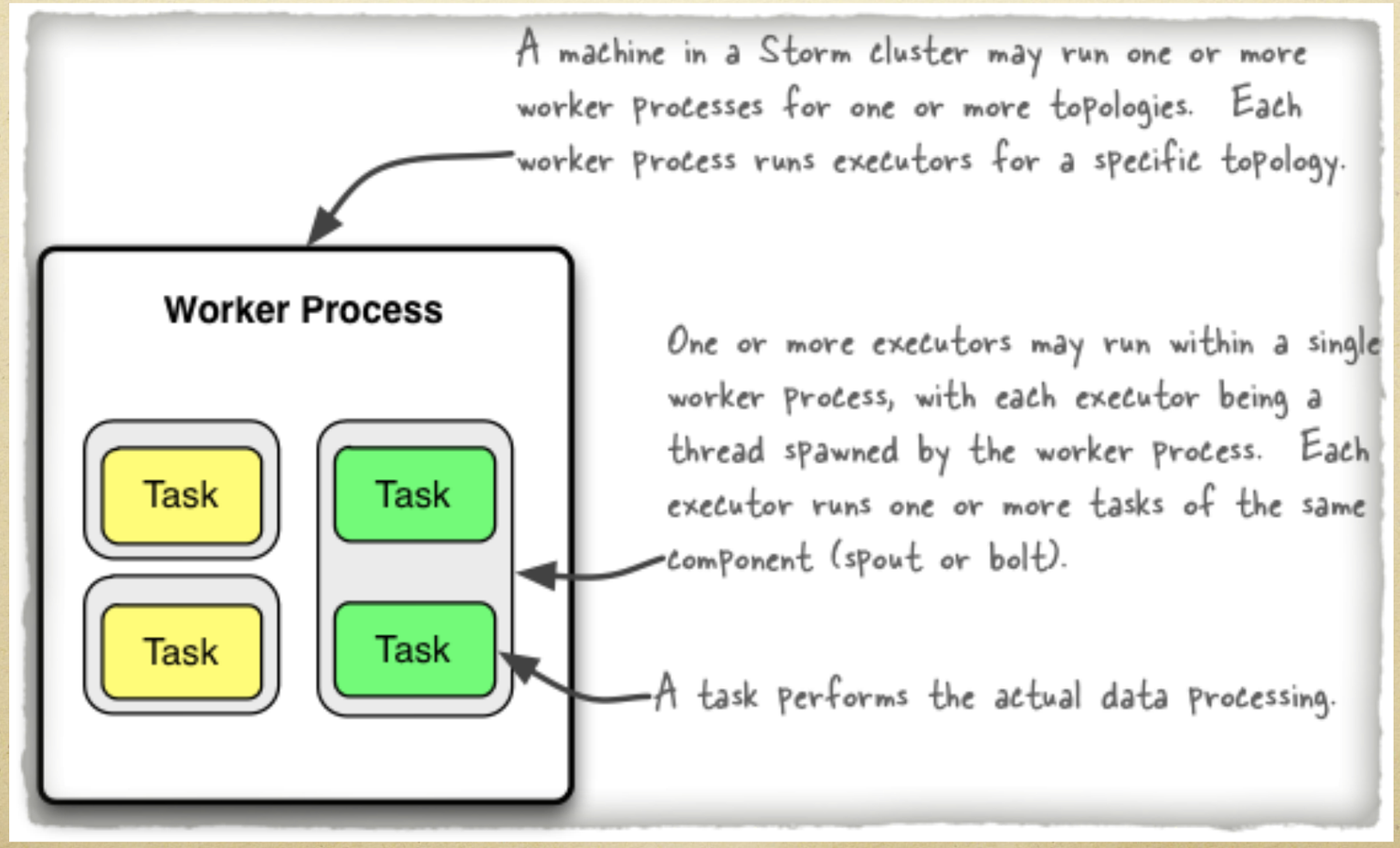
4. Topology

- Tuples– an ordered list of elements. For example, a “4-tuple” might be (7, 1, 3, 7). Tuple is the fundamental data structure in Storm.
- Streams – an unbounded sequence of tuples. Core abstraction in Storm and are what you “process” in Storm.
- Spouts –sources of streams in a computation (e.g. a Twitter API)
- Bolts – process input streams and produce output streams. They can: run functions; filter, aggregate, or join data; or talk to databases.
- Topologies – the overall calculation, represented visually as a network of spouts and bolts (as in the following diagram)

Week 5: Kafka & Storm

5. Command Line ToolStorm distinguishes between the following three main entities that are used to actually run a topology in a Storm cluster:

- Worker processes
- Executors (threads)
- Tasks



Week 5: Kafka & Storm

6. Topology Rebalancing:

(1) A nifty feature of Storm is that you can increase or decrease the number of worker processes and/or executors without being required to restart the cluster or the topology. The act of doing so is called rebalancing.

(2) You have two options to rebalance a topology:

a. Use the Storm web UI to rebalance the topology.

b. Use the CLI tool storm rebalance as described below.

(3) Here is an example of using the CLI tool:

Reconfigure the topology "mytopology" to use 5 worker processes, the spout "blue-spout" to use 3 executors and the bolt "yellow-bolt" to use 10 executors. -n means number of new workers

$ storm rebalance mytopology -n 5 -e blue-spout=3 -e yellow-bolt=10

7. Command Line Tool

storm jar topology-jar-path class ...

storm kill topology-name [-w wait-time-secs]

storm deactivate topology-name // Deactivates the specified topology's spouts.

storm activate topology-name // Activates the specified topology's spouts.

storm rebalance topology-name [-w wait-time-secs]

storm list

8. Storm word count vs MapReduce word count

Week 6: HBase

1. What is HBase?

HBase is a column-oriented database management system that runs on top of HDFS. It is well suited for sparse data sets, which are common in many big data use cases. Unlike relational database systems, HBase does not support a structured query language like SQL; in fact, HBase isn't a relational data store at all.

2. Key Features

- Schema-less, no-sql
- Column-oriented
- Good for semi-structure and structured data
- Random, real-time read/write
- Data replica across cluster

3. What HBase Good For

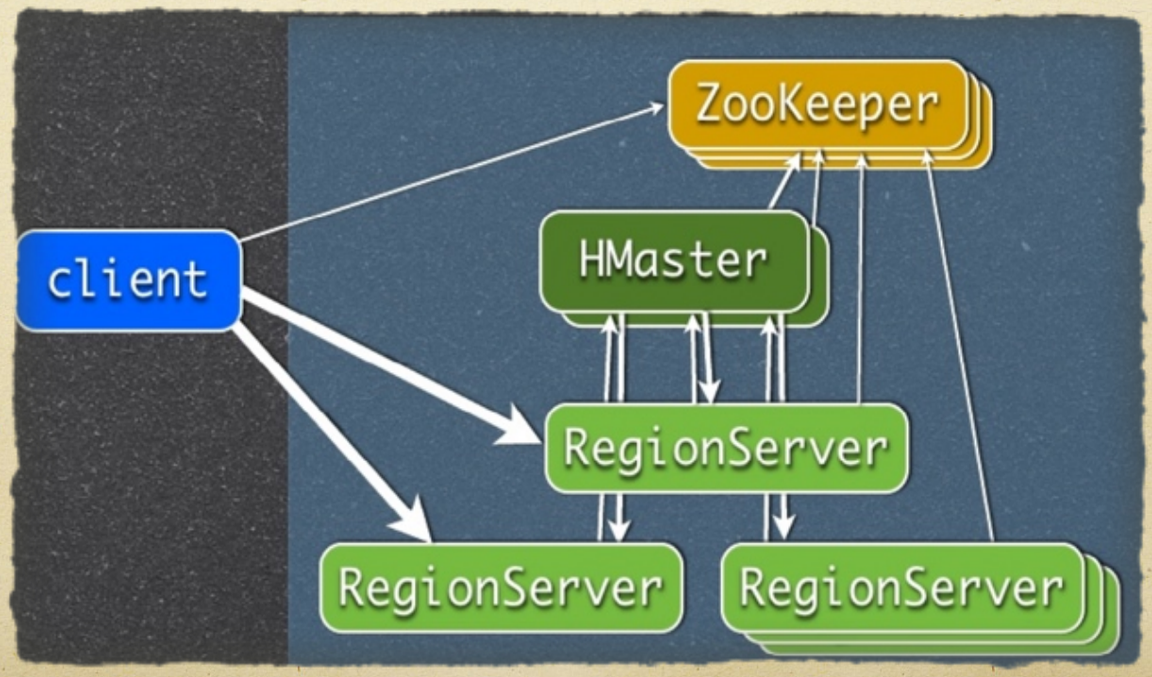
- Large datasets
- Spare datasets
- Loosely coupled records
- Lot of concurrent clients

Week 6: HBase

4. What HBase is NOT

- Not a SQL database
- Not relational
- No joins
- Not a engine for sophisticated query
- Not a replacement for legacy RDBMS

5. HBase Components



Week 6: HBase

6. Tables, Column and Column Family

	HadoopStore.com Product Table					
	ProductDetails Column Family			ProductAnalytics Column Family		
RowID	#InStock	Price	Weight	Sales1Mon	Sales3Mo	Bundle
Toy Elephant	25	5.99	0.5	183	600	USB Key
USB Key	50	7.99	0.01	421	1491	YARN Book
YARN Book	30	30.78	2.4	301	999	USB Key

1. Data in HBase Tables identified by a unique key.
2. Related Columns grouped into Column Families which are saved into different files.
3. For performance reasons, you should usually not use more than 3 column families.

Week 6: HBase

7. Flexible Schema

	HadoopStore.com Product Table							
	ProductDetails Column Family ③							
RowID	#InStock	#Pages	Ages	Author	Capacity	Color	Price	Weight
Toy Elephant	25		3+			Green	5.99	0.5
USB Key	50	①			8GB	Silver	7.99	0.01
YARN Book	30	400		Murthy		②	30.78	2.4

① Each Row can define its own columns, even if other rows do not use them.

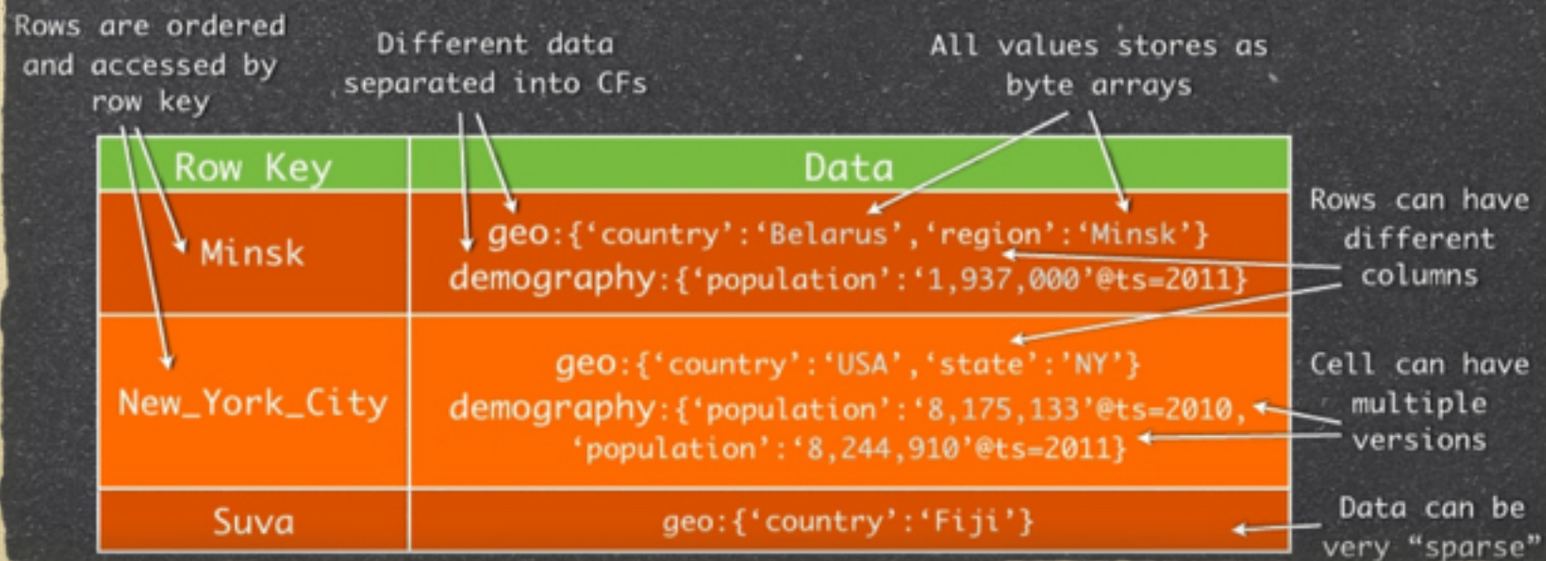
② Schema is not defined in advance, define columns as data is inserted.

③ Clients access columns using a family:qualifer notation, e.g. ProductDetails:Price

Week 6: HBase

8. Data Storage

- Row keys uninterpreted byte arrays
- Columns grouped in columnfamilies (CFs)
- CFs defined statically upon table creation
- Cell is uninterpreted byte array and a timestamp



Week 6: HBase

9. Sorted for Fast Access

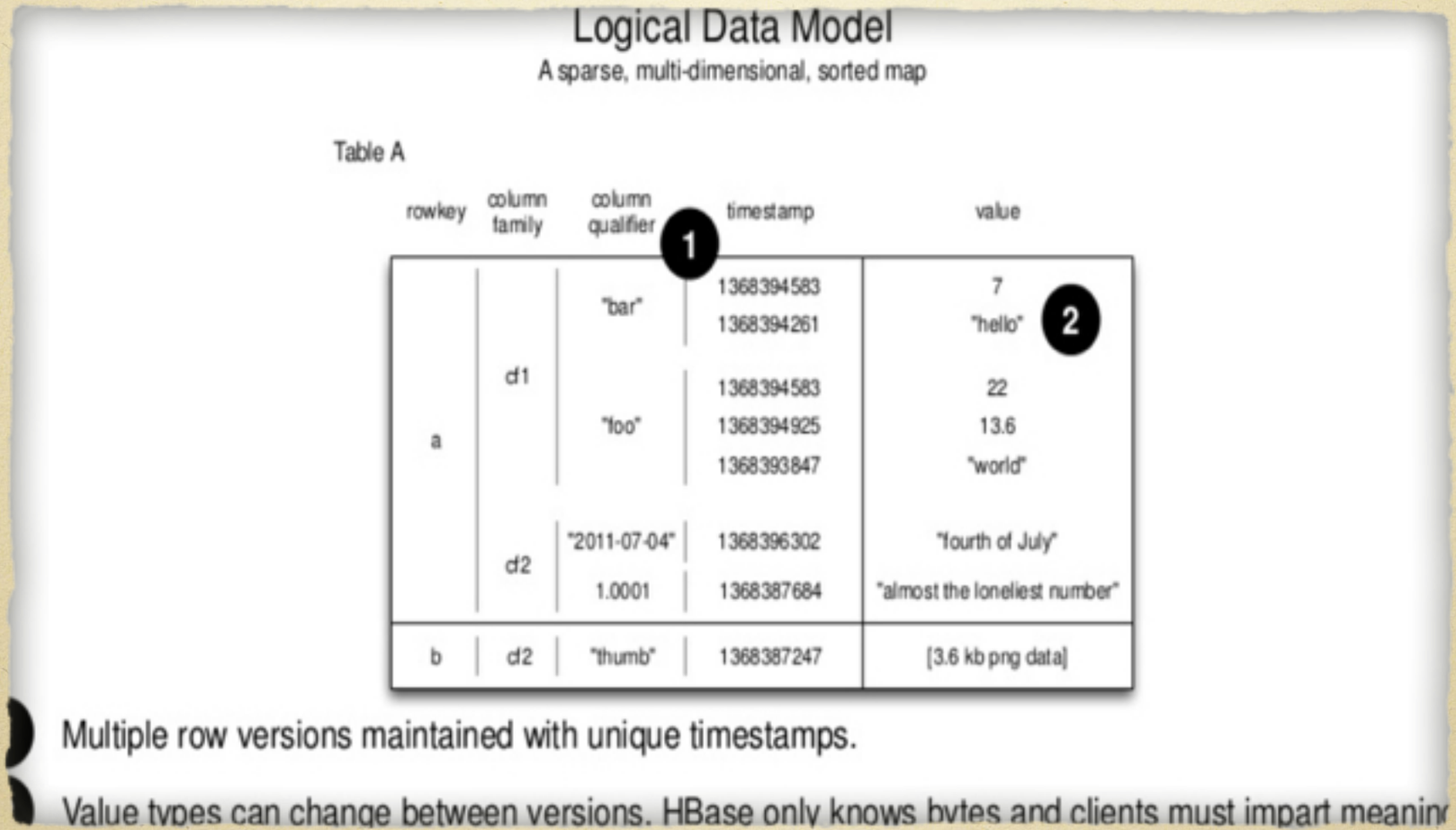
HadoopStore.com Product Table								
	ProductDetails Column Family							
RowID (2)	#InStock	#Pages	Ages	Author	Capacity	Color	Price	Weight
Toy Elephant	25		3+			Green	5.99	0.5
USB Key	50				8GB	Silver	7.99	0.01
YARN Book	30	400		Murthy			30.78	2.4

(1) Rows are sorted by key for fast range scans.

(2) Columns are sorted within Column Families.

Week 6: HBase

10. Multi-version, Type Evolution



Logical Data Model

A sparse, multi-dimensional, sorted map

Table A

rowkey	column family	column qualifier	timestamp	value
a	cf1	"bar"	1368394583	7
			1368394261	"hello"
		"foo"	1368394583	22
			1368394925	13.6
			1368393847	"world"
	cf2	"2011-07-04"	1368396302	"fourth of July"
		1.0001	1368387684	"almost the loneliest number"
b	cf2	"thumb"	1368387247	[3.6 kb png data]

1. Multiple row versions maintained with unique timestamps.
2. Value types can change between versions. HBase only knows bytes and clients must impart meanin

Week 6: HBase

11. Writing Data

- Row updates are atomic
- Updates across multi rows are NOT atomic, no transaction support out of the box
- HBase stores N versions of a cell (default 3)
- Tables are usually “sparse”

12. Reading Data

- Reader will always read the last written (and committed) values
- Reading single row: Get
- Reading multiple rows: Scan (very fast)
    - Scan usually defines start key and stop key
    - Rows are ordered, easy to do partial key scan
- Query predicate pushed down via server-side filters

Week 6: HBase

13. Data Model

Table A

Column Families

Rows

rowkey	column family	column qualifier	timestamp	value
a	cf1	"bar"	1368394583	7
			1368394261	"hello"
		"foo"	1368394583	22
			1368394925	13.6
			1368393847	"world"
	cf2	"2011-07-04"	1368396302	"fourth of July"
		1.0001	1368387684	"almost the loneliest number"
b	cf2	"thumb"	1368387247	[3.6 kb png data]

Week 6: HBase

14. Region & Region Server

Regions are the basic element of availability and distribution for tables, and are comprised of a Store per Column Family. Region Servers are the software processes (often called daemons) you activate to store and retrieve data in HBase.

15. Sharding the Data

- Automatic and configurable sharing of tables:
  - Tables partitioned into Regions
  - Region defined by start & end row keys
  - Regions are the “atoms” of distribution
- Regions are assigned to Region Servers (HBase cluster slaves)

Week 6: HBase

16. Master Server & Region Server

(1) Master Server

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers
- Maintains the state of the cluster by negotiating the load balancing
- Is responsible for schema changes and other metadata operations such as creation of tables and column families

(2) Region Server

- Communicate with the client and handle data-related operations
- Handle read and write requests for all the regions under it
- Decide the size of the region by following the region size thresholds

Week 6: HBase

17. More about Region Server

In production environments, each RegionServer is deployed on its own dedicated compute node. When you start using HBase, you create a table and then begin storing and retrieving your data.

However, at some point — and perhaps quite quickly in big data use cases — the table grows beyond a configurable limit. At this point, the HBase system automatically splits the table and distributes the load to another RegionServer.

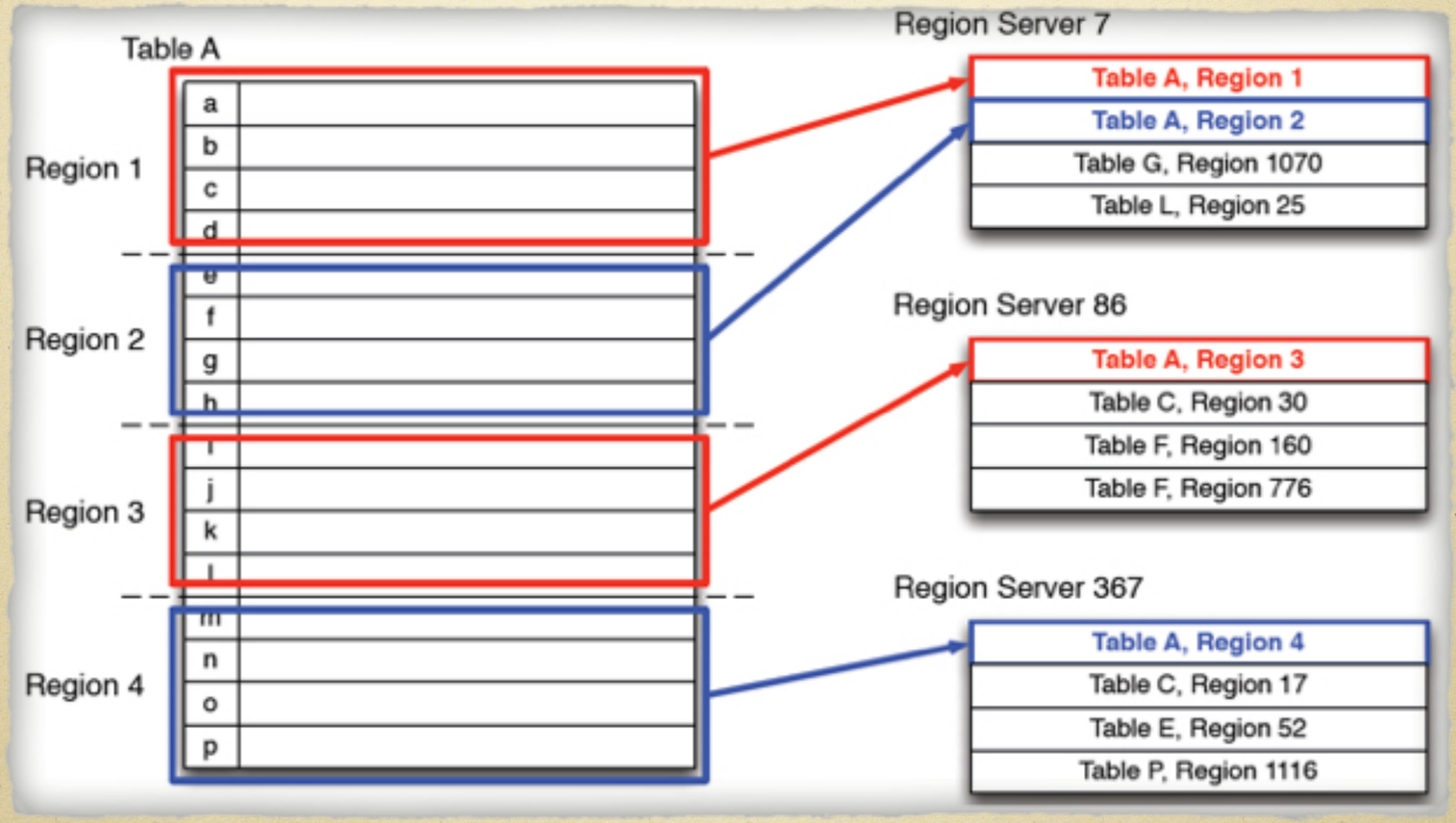
In this process, often referred to as auto-sharding, HBase automatically scales as you add data to the system — a huge benefit compared to most database management systems, which require manual intervention to scale the overall system beyond a single server. With HBase, as long as you have in the rack another spare server that's configured, scaling is automatic!

Why set a limit on tables and then split them? After all, HDFS is the underlying storage mechanism, so all available disks in the HDFS cluster are available for storing your tables. (Not counting the replication factor, of course.) If you have an entire cluster at your disposal, why limit yourself to one RegionServer to manage your tables?

Simple. You may have any number of tables large or small and you'll want HBase to leverage all available RegionServers when managing your data. You want to take full advantage of the cluster's compute performance. Furthermore, with many clients accessing your HBase system, you'll want to use many RegionServers to meet the demand.

Week 6: HBase

18. Logic Architecture



Week 6: HBase

19. Region Splits

- What is a Split
    - A “split” or “region split” is when a region is divided into 2 regions.
    - Usually because it gets too big.
    - The two splits will usually wind up on different servers.
- Region Split Strategies
    - Automatic (most common)
    - Manual (or Pre-Split)
- Pluggable Split Policy
    - Almost everyone uses “ConstantSizeRegionSplitPolicy”
    - Splits happen when a storefile becomes larger than HBase.hregion.max.filesize
    - Experts only: Other split policies exist and you can write your own.

20. High Availability

- Data is range partitioned across independent RegionServers
- All data is stored in HDFS with 3 copies
- If a Region Server is lost, data is automatically recover on a remaining Region Server
- Optionally, data can be hosted on multiple Region Servers, to ensure continuous read availability

Week 6: HBase

21. Difference between Hive & HBase:

(1) What They Do:

Apache Hive is a data warehouse infrastructure built on top of Hadoop. It allows for querying data stored on HDFS for analysis via HQL, an SQL-like language that gets translated to MapReduce jobs. Despite providing SQL functionality, Hive does not provide interactive querying yet - it only runs batch processes on Hadoop.

Apache HBase is a NoSQL key/value store which runs on top of HDFS. Unlike Hive, HBase operations run in real-time on its database rather than MapReduce jobs. HBase is partitioned to tables, and tables are further split into column families. Column families, which must be declared in the schema, group together a certain set of columns (columns don't require schema definition). For example, the "message" column family may include the columns: "to", "from", "date", "subject", and "body". Each key/value pair in HBase is defined as a cell, and each key consists of row-key, column family, column, and time-stamp. A row in HBase is a grouping of key/value mappings identified by the row-key. HBase enjoys Hadoop's infrastructure and scales horizontally using off the shelf servers.

Week 6: HBase

(2) Use Cases:

Apache Hive should be used for analytical querying of data collected over a period of time - for instance, to calculate trends or website logs. Hive should not be used for real-time querying since it could take a while before any results are returned.

Apache HBase is perfect for real-time querying of Big Data. Facebook use it for messaging and real-time analytics. They may even be using it to count Facebook likes.

Week 6: HBase

22. Hbase Shell

CREATE TABLES
hbase> create ‘t1’, {NAME => ‘fam1’}, {NAME => ‘fam2’}
hbase> create ‘t1’, ‘fam1’, ‘fam2’ # Shorthand

ADD/DELETE COLUMN FAMILY
hbase> alter 't1', NAME => 'f1', VERSIONS => 5
hbase> alter 't1', 'f1', {NAME => 'f2', IN_MEMORY => true}, {NAME => 'f3', VERSIONS => 5}
hbase> alter 'ns1:t1', NAME => 'f1', METHOD => 'delete'
hbase> alter 'ns1:t1', 'delete' => ‘f1'

enable_all/disable_all/drop_all: enable/disable/drop all the tables matching the given regex:
hbase> enable_all ’t.*'
hbase> disable_all ’t.*'
hbase> drop_all ’t.*'

exists: to check the existence of HBase table
hbase> exists ‘t1’

is_disabled/is_enabled: to know whether a HBase table is disabled or enabled
hbase> is_disabled ‘t1'
hbase> is_enabled ‘t1'

Week 6: HBase

DELETE a cell value at a specific table/row/column (optionally timestamp coordinates)
hbase> delete 't1', 'r1', 'c1', ts1

Using the “deleteall” command, you can delete all the cells in a row.
hbase> deleteall ‘<table name>’, ‘<row>’

PUT a cell value at a specific table/row/column (optionally timestamp coordinates)
hbase> put 't1', 'r1', 'c1', ‘value’

SCAN
To scan the rows of a table, use scan ‘tablename’. There are several **options** that can be used to restrict what data is returned:
(1) COLUMNS: To retrieve only certain columns:
scan 'tablename' , {COLUMNS => ‘cfamily:cqualifier’} OR scan 'tablename' , {COLUMNS => ['cf1:cq1' , 'cf2:cq2']}
scan 'stockYahoo', {COLUMNS=>’stockYahoo_data:close'}
(2) Reverse scan:
scan 't1', {REVERSED => TRUE}
(3) START ROW/STOP ROW: The row key to start or stop scanning from. LIMIT: The number of row keys to return. TIMESTAMP: A specific timestamp to search for (this is a long type).
hbase> scan 't1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STARTROW => ‘xyz’, TIMERANGE => [1303668804, 1303668904]}

Week 6: HBase

23. Ways of importing data into HBase

I. Put API

II. bulk load tool

III. customized MapReduce job

IV. open source tool sqoop

Week 6: HBase

I. Put:

(1) The easiest but not effective way

(2) not good for importing bulk data

(3) HBase was designed for big data storage, so Put is not the most effective way.

Week 6: HBase

II. bulk load:

(1) Most effective way of importing bulk data

(2) bulk load create a HBase HFile via MapReduce job, and load data to cluster

(3) The easiest way of using bulk load is importtsv

importtsv: a builtin tool

(4) Benefit of using bulk load:

Using bulk load will use less CPU and network resources than simply using the HBase API.

To load data via Puts (i.e., non-bulk loading):

$ bin/hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=a,b,c <tablename> <hdfs-inputdir>

To generate StoreFiles for bulk-loading:

$ bin/hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=a,b,c -Dimporttsv.bulk.output=hdfs://storefile-outputdir <tablename> <hdfs-data-inputdir>

Week 6: HBase

III. MapReduce:

MapReduce is flexible, and it will be the most effective way if data type is in other formats (not csv, tsv, etc) or if data is produced dynamically. Even though, the performance might be low if MapReduce jobs are not designed well.

Week 6: HBase

IV. sqoop:

Will be discussed in Sqoop section.

— Latest stable release is 1.4.6. Latest cut of Sqoop2 is 1.99.6. Note that 1.99.6 is not compatible with 1.4.6 and not feature complete, it is not intended for production deployment.

Week 6: HBase

24. Demo: Load data into HBase via ImportTsv

Usage: importtsv -Dimporttsv.columns=a,b,c <tablename> <inputdir>
Imports the given input directory of TSV data into the specified table.
Options:
-Dimporttsv.skip.bad.lines=false - fail if encountering an invalid line
'-Dimporttsv.separator=|' - eg separate on pipes instead of tabs
-Dimporttsv.timestamp=currentTimeAsLong - use the specified timestamp for the import
-Dimporttsv.mapper.class=my.Mapper - A user-defined Mapper to use instead of org.apache.hadoop.hbase.mapreduce.TsvImporterMapper

Demo yahooFinance data to HBase via ImportTsv

Week 6: HBase

25. Demo: Access HBase with Pig

Pig comes with some built-in **HBaseStorage** and **HBaseLoader**.

HBaseStorage:

Loads and stores data from an HBase table.

Syntax: HBaseStorage('columns', [‘options'])

columns: A list of qualified HBase columns to read data from or store data to.

The column family name and column qualifier are seperated by a colon (:).

Demo yahooFinance data from Pig to HBase

Week 6: HBase

26. SQL vs NoSQL

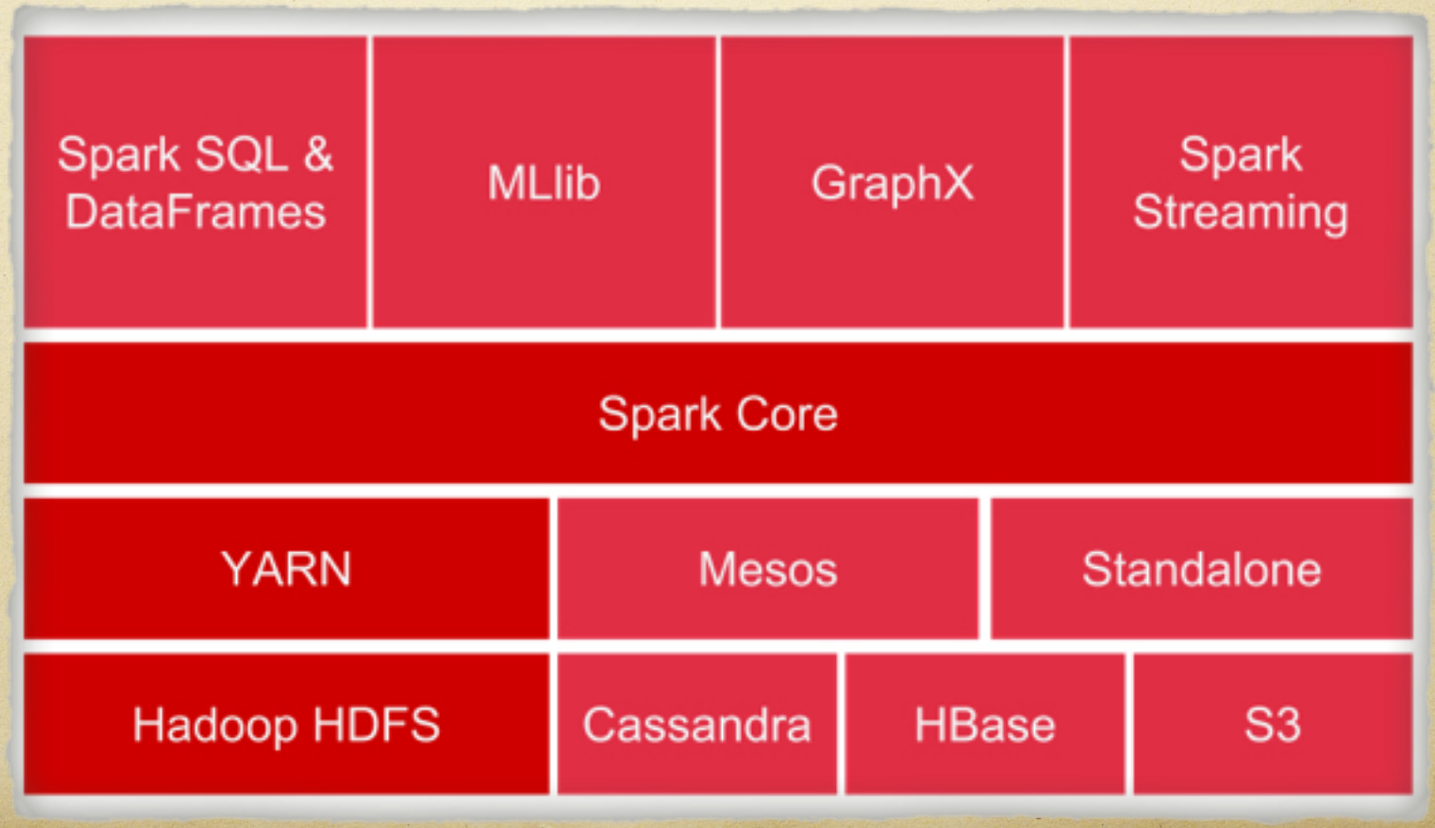
	SQL	NoSQL
Data storage	Stored in a relational model, with rows and columns. Rows contain all of the information about one specific entry/entity, and columns are all the separate data points; for example, you might have a row about a specific car, in which the columns are 'Make', 'Model', 'Colour' and so on.	The term "NoSQL" encompasses a host of databases, each with different data storage models. The main ones are: document, graph, key-value and columnar. More on the distinctions between them below.
Schemas and Flexibility	Each record conforms to fixed schema, meaning the columns must be decided and locked before data entry and each row must contain data for each column. This can be amended, but it involves altering the whole database and going offline.	Schemas are dynamic. Information can be added on the fly, and each 'row' (or equivalent) doesn't have to contain data for each 'column'.
Scalability	Scaling is vertical. In essence, more data means a bigger server, which can get very expensive. It is possible to scale an RDBMS across multiple servers, but this is a difficult and time-consuming process.	Scaling is horizontal, meaning across servers. These multiple servers can be cheap commodity hardware or cloud instances, making it alot more cost-effective than vertical scaling. Many NoSQL technologies also distribute data across servers automatically.
ACID Compliancy (Atomicity, Consistency, Isolation, Durability)	The vast majority of relational databases are ACID compliant.	Varies between technologies, but many NoSQL solutions sacrifice ACID compliancy for performance and scalability

Week 7: Spark & Machine Learning

1. What is Apache Spark?

Spark is essentially a fast and flexible data processing framework. It has an advanced execution engine supporting cyclic data flow with in-memory computing functionalities. Apache Spark can run on Hadoop, as a standalone system or on the cloud. Spark is capable of accessing diverse data sources including HDFS, HBase, Cassandra among others.

2. Spark Stack



Week 7: Spark & Machine Learning

3. Key features of Spark

- Spark allows Integration with Hadoop and files included in HDFS.
- It has an independent language (Scala) interpreter and hence comes with an interactive language shell.
- It consists of RDD's (Resilient Distributed Datasets), that can be cached across computing nodes in a cluster.
- It supports multiple analytic tools that are used for interactive query analysis, real-time analysis and graph processing. Additionally, some of the salient features of Spark include:
    - Lighting fast processing: When it comes to Big Data processing, speed always matters, and Spark runs Hadoop clusters way faster than others. Spark makes this possible by reducing the number of read/write operations to the disc. It stores this intermediate processing data in memory.
    - Support for sophisticated analytics: In addition to simple "map" and "reduce" operations, Spark supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms. This allows users to combine all these capabilities in a single workflow.
    - Real-time stream processing: Spark can handle real-time streaming. MapReduce primarily handles and processes previously stored data even though there are other frameworks to obtain real-time streaming. Spark does this in the best way possible.

Week 7: Spark & Machine Learning

4. What is “RDD”?

RDD stands for Resilient Distribution Datasets: a collection of fault-tolerant operational elements that run in parallel. The partitioned data in RDD is immutable and is distributed in nature. Automatically rebuilt after failure.

RDD Features:

a. Hold references to Partition objects

b. Each Partition object references a subset of your data

c. Partitions are assigned to nodes on your cluster

d. Each partition/split will be in RAM (by default)

5. How does one create RDDs in Spark?

In Spark, parallelized collections are created by calling the SparkContext “parallelize” method on an existing collection in your driver program.

```
val data = Array(4,6,7,8)
val distData = sc.parallelize(data)
```

Text file RDDs can be created using SparkContext’s “textFile” method. Spark has the ability to create distributed datasets from any storage source supported by Hadoop, including your local file system, HDFS, Cassandra, HBase, Amazon S3, among others. Spark supports text files, “SequenceFiles”, and any other Hadoop “InputFormat” components.

```
val inputfile = sc.textFile(“input.txt”)
```

Week 7: Spark & Machine Learning

6. What does the Spark Engine do?

Spark Engine is responsible for scheduling, distributing and monitoring the data application across the cluster.

7. Define “Partitions”

A “Partition” is a smaller and logical division of data, that is similar to the “split” in Map Reduce. Partitioning is the process that helps derive logical units of data in order to speed up data processing.

Here’s an example: val someRDD = sc.parallelize(1 to 100, 4)

Here an RDD of 100 elements is created in four partitions, which then distributes a dummy map task before collecting the elements back to the driver program.

8. What operations does the “RDD” support?

- Transformations
- Actions

Week 7: Spark & Machine Learning

9. Define “Transformations” and “Action” in Spark.

“Transformations” are functions applied on RDD, resulting in a new RDD. It does not execute until an action occurs. map() and filer() are examples of “transformations”, where the former applies the function assigned to it on each element of the RDD and results in another RDD. The filter() creates a new RDD by selecting elements from the current RDD.

In summary： create a new dataset from an existing one. Ex. map, filter, sample, union, join, repartition, etc.

An “action” helps in bringing back the data from the RDD to the local machine. Execution of “action” is the result of all transformations created previously. reduce() is an action that implements the function passed again and again until only one value is left. On the other hand, the take() action takes all the values from the RDD to the local node.

In summary： return a value to the driver program after running a computation on the dataset. Ex. reduce, collect, saveAsTextFile, etc.

10. What are the functions of “Spark Core”?

The “SparkCore” performs an array of critical functions like memory management, monitoring jobs, fault tolerance, job scheduling and interaction with storage systems.

It is the foundation of the overall project. It provides distributed task dispatching, scheduling, and basic input and output functionalities. RDD in Spark Core makes it fault tolerance. RDD is a collection of items distributed across many nodes that can be manipulated in parallel. Spark Core provides many APIs for building and manipulating these collections.

Week 7: Spark & Machine Learning

11. What is SparkContext?

"SparkContext" is the main entry point for Spark functionality. A "SparkContext" represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.

12. What is Hive on Spark?

Hive provides an SQL-like interface to data stored in the HDP. Spark users will automatically get the complete set of Hive's rich features, including any new features that Hive might introduce in the future.
The main task around implementing the Spark execution engine for Hive lies in query planning, where Hive operator plans from the semantic analyzer which is translated to a task plan that Spark can execute. It also includes query execution, where the generated Spark plan gets actually executed in the Spark cluster.

13. Name a few commonly used Spark Ecosystems.

- Spark SQL (Shark)
- Spark Streaming
- GraphX
- MLlib
- SparkR

Week 7: Spark & Machine Learning

14. What is “Spark Streaming”?

Spark supports stream processing, essentially an extension to the Spark API. This allows stream processing of live data streams. The data from different sources like Flume and HDFS is streamed and processed to file systems, live dashboards and databases. It is similar to batch processing as the input data is divided into streams like batches.

Business use cases for Spark streaming:Each Spark component has its own use case. Whenever you want to analyze data with the latency of less than 15 minutes and greater than 2 minutes i.e. near real time is when you use Spark streaming.

15. What is “GraphX” in Spark?

“GraphX” is a component in Spark which is used for graph processing. It helps to build and transform interactive graphs.

16. What is the function of “MLlib”?

“MLlib” is Spark’s machine learning library. It aims at making machine learning easy and scalable with common learning algorithms and real-life use cases including clustering, regression filtering, and dimensional reduction among others.

Week 7: Spark & Machine Learning

17. What is “Spark SQL”?

Spark SQL is a Spark interface to work with structured as well as semi-structured data. It has the capability to load data from multiple structured sources like “textfiles”, JSON files, Parquet files, among others. Spark SQL provides a special type of RDD called SchemaRDD. These are row objects, where each object represents a record.

Here’s how you can create an SQL context in Spark SQL:

SQL context: scala> var sqlContext=new SqlContext

HiveContext: scala> var hc = new HIVEContext(sc)

18. What is a “Parquet” in Spark?

“Parquet” is a columnar format file supported by many data processing systems. Spark SQL performs both read and write operations with the “Parquet” file.

19. What is an “Accumulator”?

“Accumulators” are Spark’s offline debuggers. Similar to “Hadoop Counters”, “Accumulators” provide the number of “events” in a program.

Accumulators are the variables that can be added through associative operations. Spark natively supports accumulators of numeric value types and standard mutable collections. “AggregrateByKey()” and “combineByKey()” uses accumulators.

20. Which file systems does Spark support?

- Hadoop Distributed File System (HDFS)
- Local File system
- S3

Week 7: Spark & Machine Learning

21. Benefits of Spark over MapReduce

- Due to the availability of in-memory processing, Spark implements the processing around 10-100x faster than Hadoop MapReduce.
- Unlike MapReduce, Spark provides in-built libraries to perform multiple tasks form the same core; like batch processing, steaming, machine learning, interactive SQL queries among others.
- MapReduce is highly disk-dependent whereas Spark promotes caching and in-memory data storage
- Spark is capable of iterative computation while MapReduce is not.

Additionally, Spark stores data in-memory whereas Hadoop stores data on the disk. Hadoop uses replication to achieve fault tolerance while Spark uses a different data storage model, resilient distributed datasets (RDD). It also uses a clever way of guaranteeing fault tolerance that minimizes network input and output.

22. What is a “Spark Executor”?

When “SparkContext” connects to a cluster manager, it acquires an “Executor” on the cluster nodes. “Executors” are Spark processes that run computations and store the data on the worker node. The final tasks by “SparkContext” are transferred to executors.

Week 7: Spark & Machine Learning

23. What is a “worker node”?

“Worker node” refers to any node that can run the application code in a cluster.

24. Define “PageRank”.

“PageRank” is the measure of each vertex in a graph.

25. Can we do real-time processing using Spark SQL?

Not directly but we can register an existing RDD as a SQL table and trigger SQL queries on top of that.

26. What is the biggest shortcoming of Spark?

Spark utilizes more storage space compared to Hadoop and MapReduce.

Also, Spark streaming is not actually streaming, in the sense that some of the window functions cannot properly work on top of micro batching.

27. Spark Configuration

Configuration on Spark Context:

- app name
- memory size (driver & executor)
- cores

Week 7: Spark & Machine Learning

28. Resource Manager / Cluster Managers

- Standalone
  - a simple cluster manager included with Spark that makes it easy to set up a cluster
- Apache Mesos
  - a general cluster manager that can also run Hadoop MapReduce and service applications
- Hadoop YARN
  - the resource manager in Hadoop 2

29. Spark UI

http://localhost:4040/

Jobs

Stage

Tasks

Storage

Environment

Executors

Week 7: Spark & Machine Learning

30. map vs flatMap in Spark

(1) map and flatMap are similar, in the sense they take a line from the input RDD and apply a function on it.

(2) The way they differ is that the function in map returns only one element, while function in flatMap can return a list of elements (0 or more) as an iterator.

(3) Also, the output of the flatMap is flattened. Although the function in flatMap returns a list of elements, the flatMap returns an RDD which has all the elements from the list in a flat way (not a list).

```
from pyspark import SparkContext
sc = SparkContext("spark://bigdata-vm:7077", "Map")
lines = sc.parallelize(["hello world", "hi"])
wordsWithMap = lines.map(lambda line: line.split(" ")).coalesce(1)
wordsWithFlatMap = lines.flatMap(lambda line: line.split(" ")).coalesce(1)
wordsWithMap.saveAsTextFile("hdfs://localhost:9000/user/bigdatavm/wordsWithMap"
wordsWithFlatMap.saveAsTextFile("hdfs://localhost:9000/user/bigdatavm/wordsWithFlatMap
```

The output of the map function in HDFS:

['hello', 'world']

['hi']

The output of the flatMap function in HDFS:

hello

world

hi

Week 8: Oozie & Sqoop

1. What is Oozie?

Oozie is a workflow scheduler system to manage Apache Hadoop jobs. Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions. Oozie Coordinator jobs are recurrent Oozie Workflow jobs triggered by time (frequency) and data availability. Oozie is integrated with the rest of the Hadoop stack supporting several types of Hadoop jobs such as Java MapReduce, Streaming MapReduce, Pig, Hive and Sqoop. Oozie is a scalable, reliable and extensible system.

2. What do NOT do with Oozie:

–Not a resource scheduler

–Submit job occasionally

3. oozie Workflow

A workflow definition is a DAG with control flow nodes or action nodes, where the nodes are connected by transitions arrows.

Week 8: Oozie & Sqoop

4. What are Control Flow Nodes?

– define the beginning and the end of a workflow
(start , end and fail nodes)

– control the workflow execution path
(decision , fork and join nodes)

5. What are Action Nodes?

– triggers the execution of task

- Map-reduce
- Pig
- HDFS
- Sub-workflow
- Java – Run custom Java code

Week 8: Oozie & Sqoop

6. word count example

```
<workflow-app nome='wordcount –wf'>
    <start to= 'wordcount'/>
    <action name='Wordcount'>
        <map-reduce>
            <job-tracker>foo.com:9001</job-tracker>
            <name-node>hdfs://bar.com:9000</name-node>
            <configuration>
                <property>
                    <name>mapred.input.dir</name>
                    <value>${inputDir}</value,>
                </property>
                <property>
                    <name>mapred.output.dir</name>
                    <value> ${outputDir}</value>
                </property>
            </configuration>
        </map-reduce>
        <ok to='end'/>
        <error to='kill'/>
    </action>
    <kill name='kill'/>
        <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
    </kill/>
    <end name='end'/>
</Workflow-app>
```

Week 8: Oozie & Sqoop

7. What is Sqoop and what are the features?

- Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases.
- Sqoop imports data from external structured datastores into HDFS or related systems like Hive and HBase.
- Sqoop can also be used to export data from Hadoop and export it to external structured datastores such as relational databases and enterprise data warehouses.
- Sqoop works with relational databases such as: Teradata, Netezza, Oracle, MySQL, Postgres, and HSQLDB.

8. Import data from MySQL to HBase command:

./sqoop import --connect jdbc:mysql://localhost:3306/hbase --table hly_temp_normal --hbase-table hly_temp --column-family cf1 --hbase-row-key id --username root -password 1 -m 1

Week 8: Oozie & Sqoop

9. Sqoop Debug

• sqoop eval --connect jdbc:mysql://db.example.com/corp --query "SELECT * FROM employees LIMIT 10“

• sqoop list-databases --connect jdbc:mysql://database.example.com/

• sqoop list-tables --connect jdbc:mysql://database.example.com/corp

10. Job Operations

• Submitting a Workflow, Coordinator or Bundle Job

$ oozie job -oozie http://localhost:8080/oozie -config job.properties -submit

• Starting a Workflow, Coordinator or Bundle Job

$ oozie job -oozie http://localhost:8080/oozie -start 14-20090525161321-oozie-joe

The start option starts a previously submitted workflow job, coordinator job or bundle job that is in PREP status.

• Running a Workflow, Coordinator or Bundle Job

$ oozie job -oozie http://localhost:8080/oozie -config job.properties -run

The run option creates and starts a workflow job, coordinator job or bundle job.

Week 8: Oozie & Sqoop

• Suspending a Workflow, Coordinator or Bundle Job

$ oozie job -oozie http://localhost:8080/oozie -suspend 14-20090525161321-oozie-joe

The suspend option suspends a workflow job in RUNNING status.

• Resuming a Workflow, Coordinator or Bundle Job

$ oozie job -oozie http://localhost:8080/oozie -resume 14-20090525161321-oozie-joe

The resume option resumes a workflow job in SUSPENDED status.

• Killing a Workflow, Coordinator or Bundle Job

$ oozie job -oozie http://localhost:8080/oozie -kill 14-20090525161321-oozie-joe

The kill option kills a workflow job in PREP , SUSPENDED or RUNNING status and a coordinator/bundle job in =PREP=, RUNNING , PREPSUSPENDED , SUSPENDED , PREPPAUSED , or PAUSED status.

• Rerunning a Workflow Job

$ oozie job -oozie http://localhost:8080/oozie -config job.properties -rerun 14-20090525161321-oozie-joe

The rerun option reruns a completed (SUCCCEDED , FAILED or KILLED) job skipping the specified nodes.

Reference

http://www.edureka.co/blog/introduction-to-apache-hadoop-hdfs/
http://www.tutorialspoint.com/map_reduce/map_reduce_combiners.htm
https://www.xplenty.com/blog/2014/05/hive-vs-hbase/
http://pig.apache.org/docs/r0.11.1/func.html#HBaseStorage
https://imiloainf.wordpress.com/2013/01/04/hbase-shell/
http://hadooptutorial.info/hbase-shell-commands-in-practice/
http://gethue.com/hadoop-tutorial-use-pig-and-hive-with-hbase/
https://github.com/romainr/hadoop-tutorials-examples
http://sqoop.apache.org
http://www.edureka.co/blog/top-apache-spark-interview-questions-you-should-prepare-for-in-2016
https://oozie.apache.org/docs/3.1.3-incubating/DG_CommandLineTool.html
http://www.edureka.co/blog/map-side-join-vs-join/