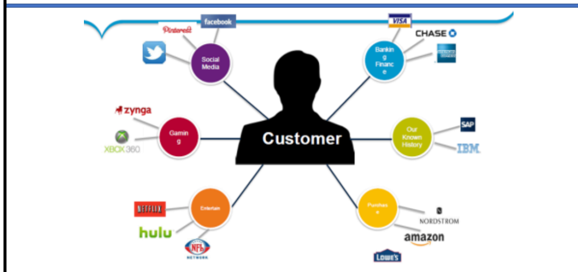
Big Data Analytics - Overview

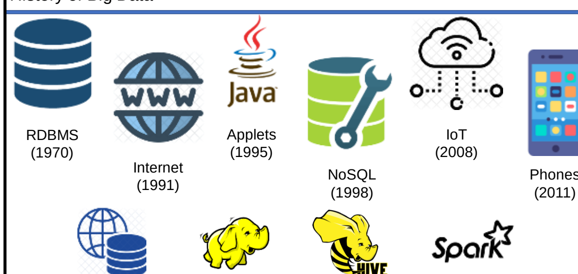
Trainer: Mr. Nilesh Ghule.

Source of Big Data



History of Big Data

History of Big Data

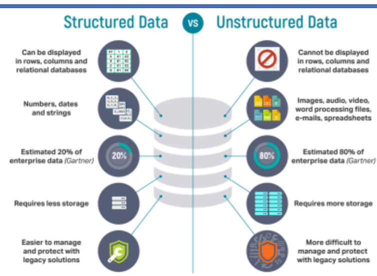


One internet minute

One internet minute

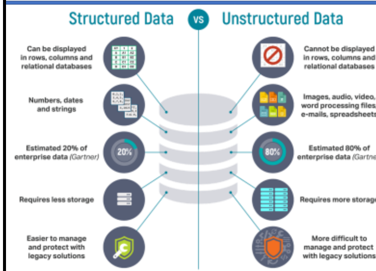
- KB
- MB
- GB
- TB
- PB
- XB
- ZB
- YB
- BB
- …

Structured vs Unstructured data



Sunbeam Infotech www.sunbeaminfo.com

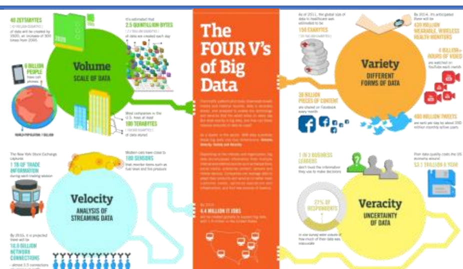
Structured vs Unstructured data vs Semi Structured data



- Semi-structured data
  - Flexible structured data
  - Better storage utilization
  - Usually hierarchical struct
  - Or Key-value format
  - Examples:
    - XML
    - JSON

Sunbeam Infotech www.sunbeaminfo.com

Big Data characteristics



Sunbeam Infotech www.sunbeaminfo.com

Big Data Spread

Sunbeam Infotech www.sunbeaminfo.com

Selective Ignorance

- Want to develop a software for NGO to organize Blood donation camp. Which information of donors to be collected?

Sunbeam Infotech www.sunbeaminfo.com

Selective Ignorance

- Blood Donors data to collect?

- Name
- Blood group
- Medical history
- Last blood donation date
- Gender
- Email
- Mobile
- Address

- Blood Donors data to collect?

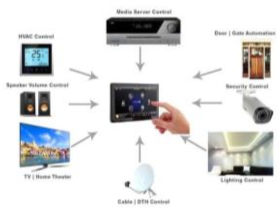
- Qualification
- Salary
- Occupation
- Religion
- Caste

Sunbeam Infotech www.sunbeaminfo.com

Big Data Technologies @ Sunbeam Infotech

Big Data & IoT

- Home Automation
  - ON/OFF appliance
  - Report state of appliance periodically.
- Huge Storage
  - High velocity, High volume & High variety.
- Process data
  - Analyze Time/Day of max/min usage.
  - Decide state of appliance - minimize electricity.
  - Invent new products, new marketing schemes.

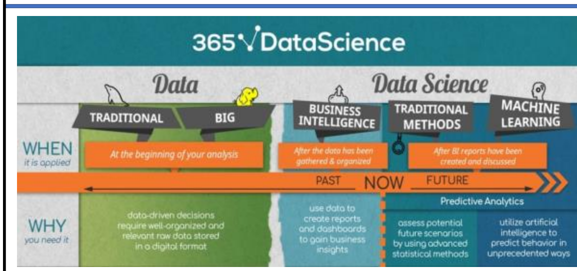


Big Data & Covid-19

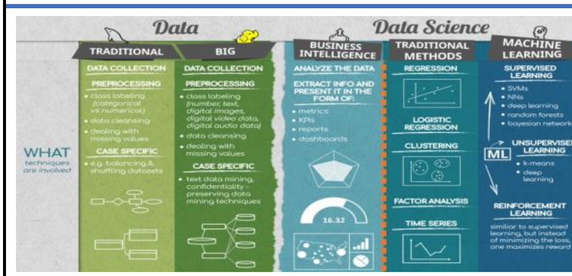
- Covid tracker apps
  - GPS & Bluetooth data from apps
  - Tracking patients
  - Contact tracing
  - Tracking quarantined people
- Covid testing devices
  - Digital thermometer
  - Swab testing devices
  - Pathology testing
- Covid treatment
  - Treatment details
  - Patient progress
- Covid Vaccine trials
  - Simulating drugs impact
  - Tracking effects & side-effects



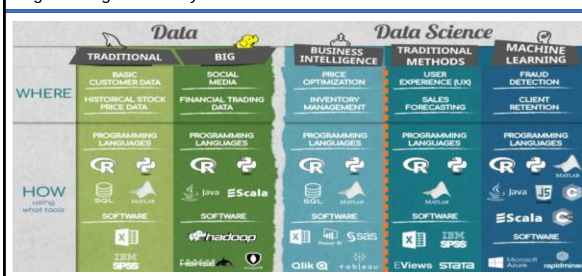

Jargon of Big Data Analytics & Data science



Jargon of Big Data Analytics & Data science



Jargon of Big Data Analytics & Data science



Jargon of Big Data Analytics & Data science

- In 2018, Big data analytics market was 169 billion dollar worth.
- By 2022, more than 274 billion dollar industry.
- Apart from technologies mentioned above few more terms are relevant
  - Artificial intelligence
  - Cloud computing
  - Parallel computing

Big Data Technologies @ Sunbeam Infotech

Jargon of Big Data Analytics & Data science

- Data analysis vs Data analytics
- Business analysis vs Business analytics
- Data analysis vs Data visualization
- Data analysis vs Data mining
- Artificial intelligence vs Machine learning vs Deep learning
- Data engineering vs Data science

Big Data & Analytics Spectrum

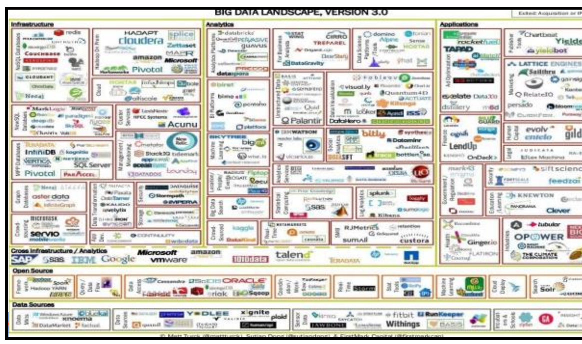
- Data storage
  - RDBMS & NoSQL databases
  - Data warehouse
  - S3, DFS, …
- Data Analysis & visualizations
  - Data Visualizations
  - Business reports
- Artificial Intelligence, Data Science & Data mining
  - Mathematics, Statistics & Computer algorithms
  - Machine learning & Deep learning
  - R Programming, Python
- Data Engineering
  - Hadoop, Hive, Spark, Kafka, BigTable, …
  - Parallel processing
  - Java, Scala, Python.
- Infrastructure
  - Linux, Cloud Computing

Big Data domains & opportunities

- Domains: Health-care, Retails, Trading/Share market, Finance, Security, Fraud, Search engines, Log Analysis, Telecom, Traffic Control, Manufacturing and lot more.
- Big Data is all about :- Think, Collect, Manage, Analyze, Summarize, Visualize, Discover Knowledge and Take Decisions.
- Job profiles:
  - Business Analyst/Intelligence
  - Database engineer / DWH
  - Big Data engineer
  - IT operations
  - AI/ML engineer
  - Data Scientist
  - Big Data Architect

- The sexiest job in the 21st century require a mixture of multidisciplinary abilities and suitable candidates must be prepared to learn and develop constantly.

-Ronald Van Loon



SUNBEAM

Big Data Frameworks

Trainer: Mr. Nilesh Ghule

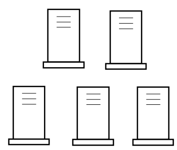
Big Data frameworks/software

- RDBMS databases
  - Oracle
  - SQL Server
  - MySQL, PostgreSQL
- NoSQL databases
  - MongoDb
  - Dynamo Db
  - Oracle KV store
  - Cassandra
  - HBase
  - Neo4J
- Big data storage
  - Amazon S3, Azure Blob.
  - Hadoop (HDFS)
  - GFS, GPFS, DBFS, MapR FS.
- Batch processing
  - Hadoop (Map Reduce)
  - Hive
  - Tez
  - Spark (Core, SQL)
  - Flink
  - Beam
- Interactive processing
  - Impala
  - Drill
- Stream processing
  - Flume
  - Kafka
  - Spark (Streaming)
  - Storm
  - Flink
  - Beam

Big Data Technologies @ Sunbeam Infotech

Distributed Systems

- Most of big data framework are distributed systems.
- Distributed system contains set of computers connected in a network (e.g. LAN). It is also referred as cluster. Each computer in cluster is referred as a node.
- Distributed systems provides
  - High availability, Fault tolerance, Rich computing, High memory.
  - High scalability (Horizontal scaling), Load balancing.
- There are two prime components of distributed system
  - Distributed storage
  - Distributed computing
- Major challenges for distributed systems
  - Node failure
  - Network failure
  - Distributed synchronization

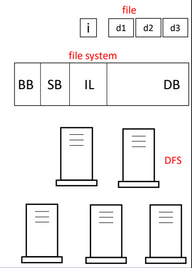


Cluster

Sunbeam Infotech www.sunbeaminfo.com

Distributed Storage

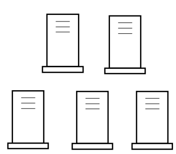
- Each file have data (contents) and metadata (info).
- Information about data blocks is stored into metadata. To read the file first metadata is accessed and then data blocks. To write data blocks are updated and metadata as well.
- Files are organized into file systems. File systems arrange file's data blocks and inodes in systematic manner for efficient storage and access.
- In distributed file system, data blocks and metadata can be scattered on multiple nodes in the cluster.
- This improves the processing speed of the data.
- However what if any node is failed (containing data) or metadata node is failed? DFS gracefully handle these concerns using replication and/or backup node features.



Sunbeam Infotech www.sunbeaminfo.com

Distributed Computing

- Traditionally program loads data to be processed from the source and perform operations on it.
- This approach in not suitable for Big Data, considering data size and read/write speed of storage.
- Since data is stored on multiple nodes (distributed storage), program is also executed on multiple nodes processing partial data. These partial results are collected on a node and processed to yield final result.
- Distributed computing follows map-reduce design pattern.
  - Map stage process each record individually.
  - Reduce stage performs aggregation operation.
- Where does individual nodes process the data in memory or on disk? What if any node fails?

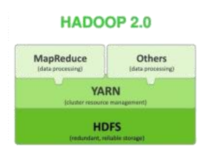


Distributed computing

Sunbeam Infotech www.sunbeaminfo.com

Apache Hadoop

- Hadoop is developed by Doug cutting.
  - Web crawler – Nutch
  - Distributed computing and storage needed to process huge data produced by the crawler.
  - Joined Yahoo. Developed and open sourced under Apache license.
- Hadoop 1.x
  - Distributed storage: HDFS
  - Distributed computing Map-reduce
- Hadoop 2.x
  - Distributed storage: HDFS
  - Distributed computing Map-reduce
  - Cluster manager: YARN
- Hadoop is like a Kernel/Platform on which many different applications are built (eco-systems).



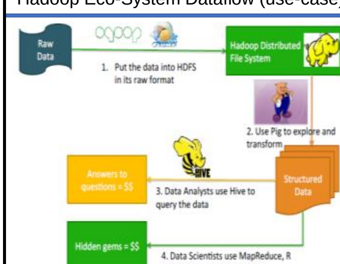
Sunbeam Infotech www.sunbeaminfo.com

Hadoop Eco-System & Hadoop distributions

- Eco-Systems
  - HBase
  - Sqoop
  - Flume
  - Pig
  - Hive
  - Impala
  - Hue
  - Oozie
  - ZooKeeper
  - Spark
- Hadoop Distributions
  - Cloudera
  - Hotonworks
  - AWS EMR
  - MapR

Sunbeam Infotech www.sunbeaminfo.com

Hadoop Eco-System Dataflow (use-case)



- Hadoop ELT
  - Extract data from sources (RDBMS or Live streaming)
  - Load (raw) data into HDFS.
  - Data is processed, transformed and/or summarized to convert into structured (tabular) format.
  - This structured data is loaded into the Hive.
  - Hive tables can be queried to analyse the data.
  - ML algorithms can be used on the data to get insights and/or predictions.
- Hadoop is batch processing framework.

Sunbeam Infotech www.sunbeaminfo.com

SUNBEAM

Big Data – Hadoop

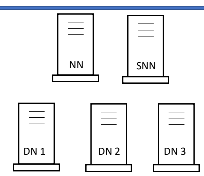
Trainer: Mr. Nilesh Ghule.

Apache Hadoop History

- Hadoop is developed by Doug cutting & Mike Cafarella.
  - Core code of distributed storage and distributed computing in Hadoop is borrowed from Nutch project.
  - Nutch project is web crawler developed by Doug & Mike.
  - Distributed computing and storage needed to process huge data produced by the crawler.
  - Doug Cutting joined Yahoo.
  - Hadoop 0.1.0 is released in April 2006.
  - Hadoop open sourced under Apache license.
- Development of Hadoop is inspired from Google white-papers on GFS (2003) & MapReduce (2004).
- Hadoop is implemented in Java.
- Hadoop is named after Doug Cutting's toy elephant.
- Hadoop has major components HDFS & MapReduce.

Hadoop Distributed File System

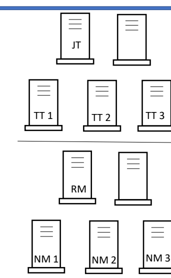
- HDFS is fault tolerant, redundant distributed file system.
- It is implemented following white-paper on Google File System.
- HDFS has three components
  - Name Node – Manage file metadata.
  - Data Node – Manage files data.
  - Secondary Name Node – Metadata backup.
- HDFS stores file's data into data blocks. Size of data block is 64 MB or 128 MB.
- Each HDFS block is replicated on 3 nodes (while write operation). It ensures that if any node fails, data can be taken from some replica node.



- The metadata backup is maintained on secondary name node. In case of name node failure, metadata can be retrieved from secondary name node.
- This makes HDFS fault-tolerant.

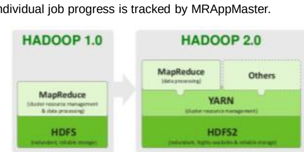
Hadoop Map-Reduce

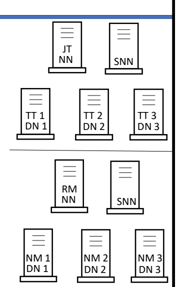
- Hadoop MR is implemented following Google's white paper MapReduce: Simplified Data Processing on Large Clusters.
- MR job processes data stored in HDFS.
- Hadoop MR job is implemented in Java or any other programming language (using Hadoop streaming).
- This MR job is submitted to Hadoop cluster (via HDFS) and its mapper and reducer components are scheduled to execute on multiple nodes in the cluster.
- Hadoop scheduler prefer to schedule the tasks in data local fashion.
- Hadoop MapReduce execution changed drastically across Hadoop 1.x and Hadoop 2.x.



Hadoop 1.x vs Hadoop 2.x

- HDFS of 2.x can be configured to be highly available.
  - Secondary NameNode process WAL to ensure that no data is lost.
  - Standby NameNode actively takes backup and can become active namenode immediately in case of failure of NameNode.
- MapReduce of 2.x introduce YARN scheduler.
  - YARN does uniform cluster/resource management.
  - Individual job progress is tracked by MRAppMaster.



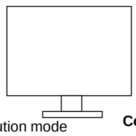


Hadoop 2.x Daemons & Hadoop installation modes

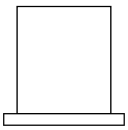
- Hadoop daemons are background processes implemented in Java.
- HDFS Daemons
  - DataNode
  - NameNode
  - SeconadaryNameNode
- YARN/MapReduce Daemons
  - NodeManager
  - ResourceManager
- All daemons are configurable via XML configuration files.
- Hadoop can be installed in 3 possible ways. It mainly differs in its applications and execution of Hadoop daemons.
- Local mode
  - All daemons runs in single Java process.
  - Can access only LocalFileSystem.
  - Used for unit testing of MR jobs & prototyping.
- Pseudo distribution mode (Single node cluster)
  - All daemons runs as independent Java processes on the single machine.
  - Used as developer machine setup.
- Full distribution mode (Multi node cluster)
  - All daemons runs as independent Java processes on the multiple machines in the network.
  - This is production cluster setup to run jobs.

Hadoop installation modes & Configuration files

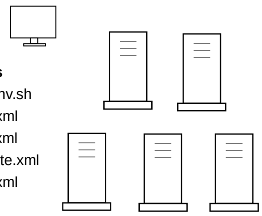
- Local mode



- Pseudo distribution mode



- Full distribution mode
  https://github.com/nilesh-g/hadoop-cluster-install



Config files
- hadoop-env.sh
- core-site.xml
- hdfs-site.xml
- mapred-site.xml
- yarn-site.xml
- ~/.bashrc

Sunbeam Infotech www.sunbeaminfo.com

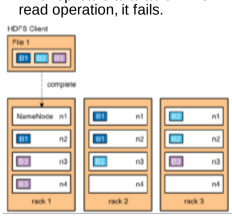
Using HDFS

- Before using HDFS need to be formatted. It create first (empty) file system image on NameNode.
  - terminal> hdfs namenode –format
- Start all HDFS daemons & verify them
  - terminal> start-dfs.sh
  - terminal> jps
  - browser: http://localhost:50070
- While metadata is loaded into NameNode memory, HDFS is not ready for use. This state is safe mode.
- HDFS user commands
  - terminal> hadoop fs –help
  - syntax: hadoop fs genericoptions command
- Generic options
  - -conf, -fs, …
- HDFS user commands categories
  - ingestion/retrieval: put, get, getmerge
  - directory handling: ls, mkdir, rmdir
  - file data handling: cat, tail, rm, truncate, touchz, stat
  - metadata handling: chmod, chown, setrep
- HDFS admin commands
  - terminal> hdfs –help
  - terminal> hdfs dfsadmin –help

Sunbeam Infotech www.sunbeaminfo.com

HDFS Replication

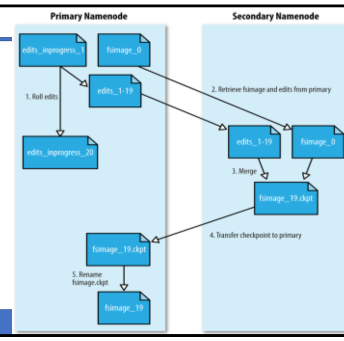
- Default replication factor for HDFS is 3.
  - hdfs-site.xml – hdfs-site.xml
- Each data block is copied on 3 different data nodes.
- Data nodes are stored across the racks for more reliability. Data nodes are chosen by name node considering load balancing.
- NameNode ensure availability of datanodes by the periodic heartbeat signal.
- If number of replicas are less than replication factor, it is under-replica. If number of replicas are more than replication factor, it is over-replica.
- Hadoop auto adjust replicas to the replication factor over the time by creating more replicas or deleting them depending on scenario.
- Replication is done while write operation.
- If no replica is available while read operation, it fails.



Sunbeam Infotech www.sunbeaminfo.com

HDFS Internals

- NameNode loads all metadata into RAM from HDFS fsimage.
- Each change in metadata is fetched by SNN with last fsimage.
- Applying those changes in fsimage, SNN creates next fsimage checkpoint (.ckpt)
- This checkpoint is transferred to NameNode.
- NameNode rename it to consider as new fsimage and deletes old fsimage.
- This ensures that NN & SNN maintains same metadata.



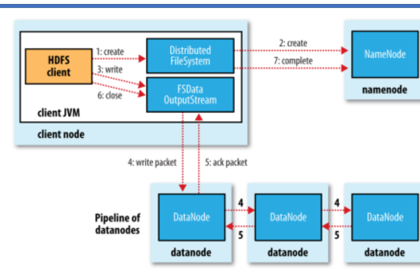
Sunbeam Infotech

HDFS Java API

- HDFS can be accessed or manipulated using Java API.
- DistributedFileSystem class represent HDFS, while LocalFileSystem class represent local file system.
- Mainly two types of APIs
  - FileSystem API
  - File-IO API
- FileSystem API
  - Deals with metadata & directories.
  - FileStatus object contains metadata of file or directory.
  - Most of FileSystem APIs don't need access to DataNode (as metadata is maintained on NameNode itself).
- File IO API
  - Deals with data of the files.
  - FSDataInputStream class for reading the file, while FSDataOututStream class for writing the files.
  - They provide abstraction like replication process, network access, etc.
- Write/Read text files

Sunbeam Infotech www.sunbeaminfo.com

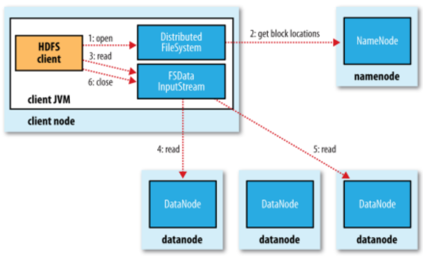
HDFS Java API: write operation



Sunbeam Infotech www.sunbeaminfo.com

Big Data Technologies @ Sunbeam Infotech

HDFS Java API: read operation



Map-Reduce thought process

- Design pattern for distributed computing.
- Typical data flow of MR program:
- Word count program
  - red green blue
  - green blue black
  - red red green
  - green red green

Implementing MR job

- Implement Mapper class
  - Handle individual record
  - class MyMapper extends Mapper<KeyIn, ValueIn, KeyOut, ValueOut> { … }
  - Override map() method
  - Input from InputFormat record by record and key-value pair output to merge stage
- Implement Reducer class
  - Perform aggregation on set of values (corresponding to each key)
  - class MyReducer extends Reducer<KeyIn, ValueIn, KeyOut, ValueOut> { … }
  - Override reduce() method
  - Input from merge stage in key-values pair and key-value pair output to OutputFormat
- Hadoop Writable
  - Like java wrapper classes, but optimized for serialization over the network.
  - IntWritable, ByteWritable, ShortWritable, LongWritable, DoubleWritable, BooleanWritable, Text
  - ArrayWritable, MapWritable, NullWritable

Implementing MR job

- Create MR job
  - Job and Jar
  - Mapper class & its output
  - Reducer class & its output
  - Input & Output format
  - Combiner, Partitioner
  - Submit job
- Configured class
  - Associate configuration object with the driver
  - getConf() and setConf()
- Tool and ToolRunner
  - Tool is standard way of implementing any processing on Hadoop – run() method
  - ToolRunner is helper to execute the Tool.
- Generic options
  - hadoop jar <jar-path> <generic-options> <cmd-line args to main-class>
  - hadoop jar <jar-path> <main-class> <generic-options> <cmd-line args to main-class>
  - Generic options:
    - -conf
    - -D
    - -fs
    - -jt
    - -files
    - -libjars
  - GenericOptionsParser

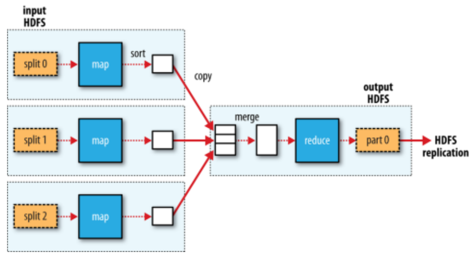
Executing MR

- hadoop jar command
  - hadoop jar <jar-path> <generic-options> <cmd-line args to main-class>
  - hadoop jar <jar-path> <main-class> <generic-options> <cmd-line args to main-class>
- MR job configurations
  - fs.defaultFS = hdfs://namenode:9000/
  - mapreduce.framework.name = yarn
  - yarn.resourcemanager.address = resourcemanager:8032
- Understanding MR summary
  - Number of mapper & reducer tasks.
  - Number of input & output records for mapper
  - Number of input & output records for reducer
  - Custom Job counters
- MR log files review
  - $HADOOP_HOME/logs

Input/Output Format and Input Splits

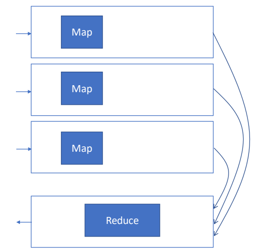
- InputFormat – how to read data
  - FileInputFormat, TextInputFormat
  - KeyValueTextInputFormat, NLineInputFormat
  - DBInputFormat
- RecordReader – Logical division of record
- Number of mappers = Number of input splits
- Number of input splits ≈ Number of HDFS blocks
- Output format – how to write data
  - FileOutputFormat, TextOutputFormat
  - DBOutputFormat
- RecordWriter – Write individual record
- Number of output files = Number of reducers
- Output written on HDFS (replicated)

Data flow of MR job (Single reducer)



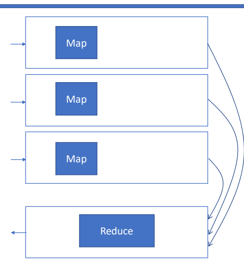
Combiner

- Combiner is a local reducer i.e. runs reducer (aggregation logic) within mapper task process.
- Minimize output for mapper task
  - Less merge & shuffle
  - Less network transfer
  - Less aggregation in reducer
- Combiner is optional.
- Works only for commutative & associative aggregate functions only.
  - A + B = B + A
  - A + (B + C) = (A + B) + C

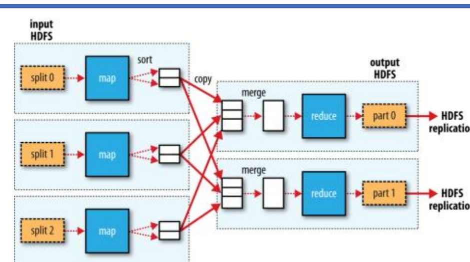


Partitioner

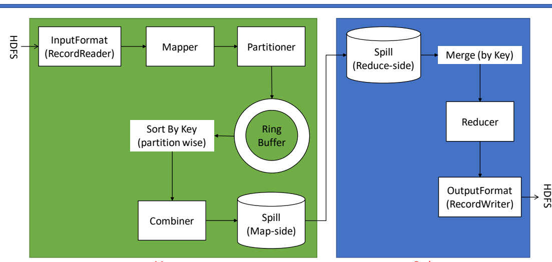
- By default MR job have single reducer.
- Having huge data for aggregation may lead to out of memory error.
- Number of reducers can be configured in job configuration file or in driver code.
  - job.setNumReduceTasks(2);
  - mapreduce.job.reduces = 2
- Number of partitions = Number of reducers
- Output of mapper is divided into multiple partitions based produced key
- By default HashPartitioner is used, that distributes mapper output in number of partitions uniformly.



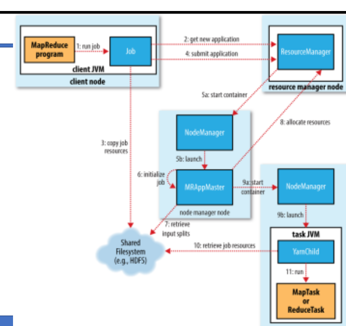
Data flow of MR job (Multiple reducers)



Hadoop MR data flow (detailed)



MR on YARN



Uber job

- By default for each MR job separate MRAppMaster, Mapper(s) and Reducer(s) processes (containers) are created.
- For small data processing this process creation and their communication is overhead.
- Uber mode allows running such small jobs in single container i.e. MRAppMaster.
- Mapper(s) & Reducer runs in same process. As no IPC involves, these small jobs are executed quickly.
- It is configured using settings
  - mapreduce.job.ubertask.enable (default: false)
  - mapreduce.job.ubertask.maxmaps (default: 9)
  - mapreduce.job.ubertask.maxreduces (default: 1)

Sunbeam Infotech www.sunbeaminfo.com

Chained Map Reduce Job

- Complex processing may not be completed in single MR job.
- Such processing can be done by feeding output of first job as input to second job. This way multiple jobs can be chained to each other.
- Need to update driver code. Run second job, only if first job is successful.
- Friends recommendation on Facebook or any social network site. Steps:
- Get count of common friends of X & Y (but X & Y are not friends of each other).
- For X, recommend top N Y's (based on common friends count).
- You may consider a minimal threshold for suggesting as friend.

Input (HDFS) → Mapper1 → Reducer1 → Job1 Output (HDFS) → Mapper2 → Reducer2 → Output (HDFS)

Sunbeam Infotech www.sunbeaminfo.com

Chained Map Reduce Job – Friends Recommendation

A	B, C, D, G
B	A, E
C	A, E
D	A, E
E	B, C, D

B, C	1
B, D	1
B, G	1
C, D	1
C, G	1
D, G	1
A, E	1
A, E	1
A, E	1
B, C	1
B, D	1
C, D	1

A,E	1 1 1
B,C	1 1
B,D	1 1
B,G	1
C,D	1 1
C,G	1
D,G	1

A,E	3
B,C	2
B,D	2
B,G	1
C,D	2
C,G	1
D,G	1

A,E	3
B,C	2
B,D	2
B,G	1
C,D	2
C,G	1
D,G	1

A	E,3
B	C,2
B	D,2
B	G,1
C	D,2
C	G,1
D	G,1

A	E,3
B	C,2 | D,2 | G,1
C	D,2 | G,1
D	G,1

A	E,3
B	C,2
B	D,2
C	D,2
C	G,1
D	G,1

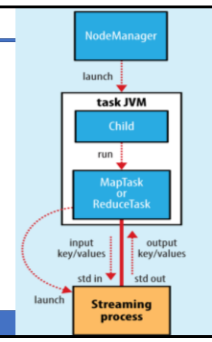
Sunbeam Infotech www.sunbeaminfo.com

Hadoop Streaming

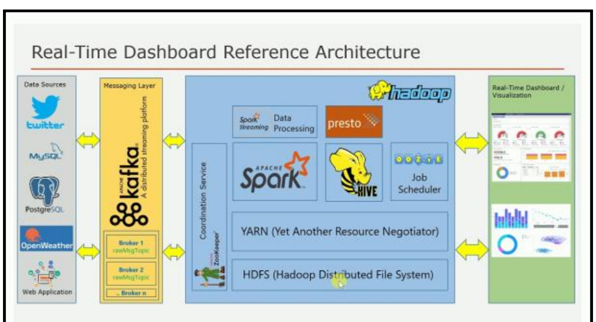
```
#!/usr/bin/python3
# mapper.py
import sys
for line in sys.stdin:
    words = line.split()
    for word in words:
        print(f"{word}\t1")
```

```
#!/usr/bin/python3
# reducer.py
di = dict()
for line in sys.stdin:
    (word,cnt) = line.split()
    newcnt = di.get(word, 0) + int(cnt)
    di[word] = newcnt
for word,total in di.items():
    print print(f"{word}\t{total}")
```

```
hadoop jar $HADOOP_HOME/share/.../hadoop-streaming-2.7.3.jar \
-files mapper.py,reducer.py \
-input /user/nilesh/wc/input \
-output /user/nilesh/wc/output \
-mapper mapper.py -reducer reducer.py
```



Sunbeam Infotech



Apache Hive

Sunbeam Infotech

Sunbeam Infotech www.sunbeaminfo.com

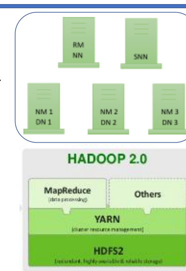
Agenda

- Hive is data warehouse built on Hadoop framework. It convert SQL-like queries into Hadoop MR jobs.
- Getting ready
  - Hadoop Introduction
  - Hive Introduction
  - Applications
  - Architecture
  - Installation
  - Hive CLI vs Beeline
- Hive QL
  - Features
  - Schema on Read
  - Hive QL types
  - Hive DDL and DQL queries
  - Hive Joins optimization

Sunbeam Infotech www.sunbeaminfo.com

Hadoop Overview

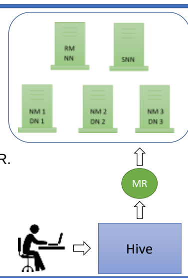
- Hadoop is Big Data framework developed at Yahoo!
  - Developed by Doug Cutting in 2006.
  - Designed after Google white papers on GFS & MR.
  - Can process huge amount of structured and unstructured data.
  - HDFS: Distributed storage
    - Fault tolerant Redundant storage
    - Write once Read multiple times
  - MapReduce: Distributed computing
    - Map stage: process individual records
    - Reduce stage: aggregate operations.
- Hadoop challenges
  - Need Java programming expertise
  - Any moderate operation need cascaded jobs
  - MR job development, testing and maintenance
  - HDFS is read-only (write once)



Sunbeam Infotech www.sunbeaminfo.com

Hive Introduction

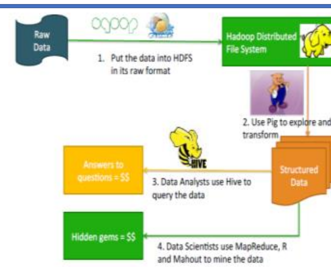
- History
  - Facebook data ingestion into Hadoop
    - 10s GB/day – 2006
    - 1 TB/day – 2007
    - MySQL/Oracle database limitations
  - Processing Hadoop data using MR is complex
  - Developed Hive to convert SQL queries into MR
  - Open sourced under Apache license (2010)
- Hive is client software that convert Hive QL queries to MR.
- Hive QL is similar to SQL with many extended features.
- Hive manage structured data.
- Hive is data warehouse (OLAP) built for Hadoop.



Sunbeam Infotech www.sunbeaminfo.com

Hive advantages and limitations

- Advantages
  - Data warehouse - data analysis
  - Long running queries.
  - Fault tolerant environment.
- Limitations
  - Slower response time.
  - Data manipulation is not supported (fully).
- Applications
  - Batch processing (SQL based)
  - ETL jobs
  - Business Intelligence (Reports)
  - Predictive Modeling
  - Data mining
  - Log processing



Sunbeam Infotech www.sunbeaminfo.com

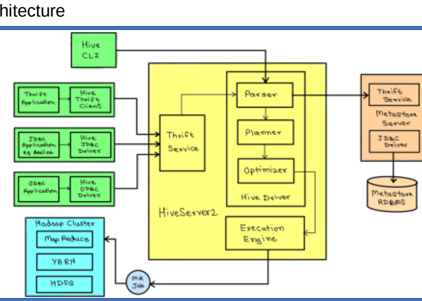
Traditional ETL vs Hadoop ELT

- ETL stands for Extract, Transform and Load.
- The ETL process typically extracts data from the source/transactional systems, transforms it to fit the model of data-warehouse and finally loads it to the data warehouse.
- The transformation process involves cleansing, enriching and applying transformations to create desired output.
- Data is usually dumped to a staging area after extraction.

- ELT stands for Extract, Load and Transform.
- As opposed to loading just the transformed data in the target systems, the ELT process loads the entire data into the data lake. This results in faster load times.
- The load process can also perform some basic validations and data cleansing rules.
- The data is then transformed for analytical reporting as per demand.

Sunbeam Infotech www.sunbeaminfo.com

Hive Architecture



Sunbeam Infotech www.sunbeaminfo.com

Hive Architecture

- Hive Server 2: Accept queries and execute them
  - Hive Driver = Parser + Planner + Execution Engine
  - Hive Execution engine
    - MR or Tez or Spark
- Hive Metastore
  - Store table structure & other metadata
  - Embedded mode
    - Easy to configure
    - By default available with Java (JDK)
    - Derby is single user RDBMS
  - Local/Remote mode
    - Can be accessed from local or remote machine
- Hive Clients
  - Hive CLI
  - Hive Beeline
  - Java/Python programs

Hive Installation & Getting started

- Install Hadoop.
- Install Hive
  - hive-site.xml
  - set PATH in ~/.bashrc
- Start metastore service.
- Start hive CLI.
- Start hiveserver2 service.
- Start hive beeline.

Hive QL

- Hive QL is extended SQL.
- Supports DQL, DML, DDL and DCL.
- DQL supports filtering, ordering, grouping, joins, etc.
  - Data will be read from HDFS.
  - Store query result into HDFS (or in another table).
- Supports views and indexes
- Manage tables, partitions & buckets
- Provide various hive data types
- Follows Schema on Read for better performance
  - While loading the data no schema is verified.
  - While processing individual records schema is verified.
  - If data is not compatible with the type, value is considered null.

Hive data types

- Primitive Types:
  - BOOLEAN (1)
  - Integers: TINYINT (1), SMALLINT(2), INT(4), BIGINT(8)
  - Floating Point: FLOAT (single precision), DOUBLE (double precision), DECIMAL(m,n)
  - Characters: CHAR(n), VARCHAR(n), STRING
  - Date & Time: TIMESTAMP, DATE, DATETIME
- Collection Types:
  - ARRAY: collection of same type of data
  - STRUCT: collection of different type of data
  - MAP: collection of key-value pairs

Hive INSERT

- Inserts new records into hive table.
- Internally creates new files under HDFS (table directory).
- Produce MR job to insert data.
- While INSERT hive follows schema on write.

File formats

- Hive data is stored in HDFS in supported file formats.
- The file formats decide data processing in generated MR job.
- Popular file formats
  - TEXTFILE
    - If not mentioned or "STORED AS TEXTFILE", the hive table data is stored in text format.
    - Internally it use InputTextFormat/OutputTextFormat for processing it. By default one record is one line.
  - RC
  - ORC
    - Text processing is not efficient.
    - ORC file format is designed for optimized execution of Hive queries.
    - ORC (Optimized Row Columnar) stores data in columnar way in binary format.

Hive SerDe

- Serde is Serializer & Deserializer.
  - Internally encapsulate Hadoop InputFormat (& RecordReader) and OutputFormat (& RecordWriter).
  - Types: Built-in Serdes (e.g. OpenCSVSerde), Third party Serdes, Custom Serdes
- OpenCSVSerde
  - Loads CSV file into hive table
    - Comma separated file
    - If data contains comma, cell is enclosed in double quote.
    - If data contains double quotes, it is escaped by "\".
- RegexSerde
  - Only Deserializer i.e. only used to read records.
  - Mainly used for data cleansing/extraction.

Hive Views

- Hive views are same as RDBMS views.
- Hive 2.x views are not materialized.
  - Only Hive view create query is stored in Hive metastore.
- Hive 3.x support materialized views.
  - `CREATE MATERIALIZED VIEW mv_booksummary AS SELECT subject, SUM(price) total, AVG(price) avgprice FROM books_orc GROUP BY subject;`
  - `ALTER MATERIALIZED VIEW mv_booksummary REBUILD;`
  - `SHOW MATERIALIZED VIEWS;`
  - `DROP MATERIALIZED VIEW mv_booksummary;`
- Applications
  - Simplify few queries.
  - Security.
  - Improve performance.

Hive Joins

- Hadoop supports two types of joins i.e. Map-side and Reduce-side join.
- In map-side join, smaller table is copied on all processing nodes and connected with other table in mapper only. Reducer does only aggregation.
- In reduce-side join, both tables are processed by individual mappers and produce common key. Using this reducer connect both tables and also does aggregation job.
- SET hive.auto.convert.join=false;
- SET hive.auto.convert.join.noconditionaltask=false;

Movie Recommendation

movies

id	title	...
...	...	...

ratings

uid	mid	rating
17	70	3.0
35	21	1.0
49	19	2.0
49	21	1.0
49	70	4.0
87	19	1.0
87	19	1.0
87	21	2.0
98	19	2.0

user movies

m1	m2	r1	r2
21	70	1.0	4.0
19	70	2.0	4.0
19	21	2.0	1.0
19	21	1.0	2.0

corr movies

m1	m2	r1	r2
21	70	1	0.0
19	70	1	0.0
19	21	2	-1.0

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}}$$

Hive scripts

- Hive scripts contains multiple Hive QL statements to executed sequentially (like .sql script).
- It includes Hive QL queries as well as configurations.
- Running using Hive CLI
  - terminal> hive -f /path/to/.hql
  - hive> SOURCE /path/to/.hql
- Running using Hive beeline
  - terminal> beeline -u ... -n ... -f /path/to/.hql
  - beeline> !run /path/to/.hql

Managed Table vs External Tables

CREATE TABLE statement.	CREATE EXTERNAL TABLE statement.
Located in HDFS warehouse directory	Located in HDFS directory.
Drop table operation drop table data (from HDFS) as well as table structure (from metastore).	Drop table operation drop only table structure (from metastore). Data in HDFS is intact.
Loading data explicitly into the table (HDFS) after table creation.	When data is already present in HDFS and need to process it using HiveQL. Multiple tables (metadata) can refer to the same data files in HDFS.

Partitioning

- Data is divided into multiple sub-directories under HDFS (table location) based on value of one or more columns.
- When query is fired for given value of column, only respective sub-directories data will be processed. This significantly improves performance.
- Examples:
  - Emp partitioned dept-wise
  - Emp partitioned job-wise
  - Emp partitioned dept-job-wise
- Static partitioning
  - Data is ingested partition-wise.
  - Very fast operation.
- Dynamic partitioning
  - Data is ingested in staging table.
  - Data is loaded partition-wise into main table using MR.

Partitioning

Bucketing

- Data in bucketed tables is divided into multiple files.
- When data is processed using MR job, number of reducers will be same as number of buckets.
- To insert data into bucketed table, it must be uploaded via staging table.
- Usually buckets are created on unique column(s) to uniformly divide data across multiple reducers.
- It provides better sampling and speed-up map side joins.
- It is mandatory for DML operations.

Bucketing

Hive functions

- Hive have many built-in functions.
  - Single Row Functions
    - Row → Function → Row
    - e.g. LENGTH(), CONCAT(), ROUND(), ...
  - Group/Aggregate Functions
    - Rows → Function → Row
    - e.g. SUM(), AVG(), COUNT(), ...
  - Table generation Functions
    - Row → Function → Rows
    - e.g. EXPLODE(), ...
- Hive function help is available in Hive documentation:
  - https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF
- Hive UDF can be written in Java/Python.

Hive Indexes

- Similar to RDBMS index.
- To speed up SELECT queries (searching & grouping).
- Indexes internally store addresses of records for given column values.
- Creating index is time-taking job (for huge data). If indexing is done under load, then clients query performance is too low.
- In Hive indexes are created, but deferred for build (using ALTER statement).
- CREATE INDEX query doesn't create index, rather keep ready for building later.
- Index building should be triggered explicitly, when server is less loaded.

Hive Indexes

- In hive indexes are stored in HDFS (as hive tables).
- These indexes are build by different index handlers e.g. BITMAP, CompactHandler, …
- Compact:
  - Stores combination of indexed column value & its HDFS block id.
- Bitmap:
  - Stores combination of indexed column value & list of rows as bitmap.
  - Bitmap indexes work faster than Compact.
- Hive indexes are not supported from Hive 3.x onwards. Use materialized view instead to improve the performance.

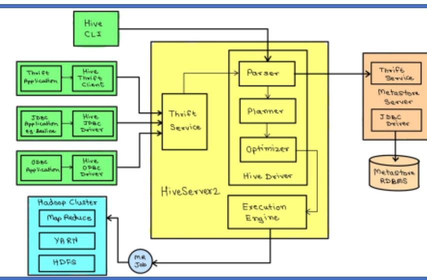
Hive DML

- Hive was designed for OLAP.
- HDFS is write once read multiple times (no edit).
- Hive doesn't allow UPDATE or DELETE operations (before 0.14).
- Newer versions enable operations. Should be avoided for efficiency.
- Need to set transaction manager, concurrency and compactor properties into hive-site.xml.
- UPDATE/DELETE is allowed only if table meets following conditions.
  - ORC file format, Bucketing, Transactional attribute
- Each DML operations save modifications into delta files. Hive process them when data is queries.
- The delta files are merged over the period by compactor threads.

Hive connectivity

- HiveServer2 must be running to connect Hive from external programs.
- The thrift service in HiveServer2 accept client query and execute it on server.
- Hive can be connected from Python with multiple packages.
- pyhive package is most popular option for it.
  - sudo apt-get install python3.7-dev libsasl2-dev
  - python3.7 -m pip install thrift sasl thrift_sasl pyhive
- Hive can be connected from Java using JDBC.
  - JDBC Jar is available in $HIVE_HOME/jdbc and it should be added in project classpath.

Hive Architecture



Apache Spark

Sunbeam Infotech

Agenda

- Introduction
- Architecture
- Spark Core
- Spark RDD
- Spark Cluster
- Spark Dataframe
- Spark SQL
- Spark on Cloud
- Spark Streaming
- Spark ML

Introduction

- Spark is Distributed computing framework, that can process huge amount of data.
- Spark can be used as eco-system of Hadoop or can be used as independent distributed computing framework.
- Developed by UCB AMPlabs division.
- Further developed/maintained by DataBricks.
- Popular Spark vendors
  - DataBricks, AWS EMR, Cloudera, MapR
- Spark Toolkit
- Spark Philosophy
  - Unified
  - Compute Engine
  - Libraries

Hadoop vs Spark

• Distributed framework • Distributed storage + Distributed computing	• Distributed framework • Distributed computing • Not tied up with particular storage
• Hadoop is developed in Java (JVM based).	• Spark is developed in Scala (JVM based).
• Designed for commodity hardware. • Data is processed in RAM and spills on disk.	• Needs better hardware config. • Data is processed fully in RAM to achieve faster execution.
• In MapReduce job, mappers & reducers are executed as independent JVM processes.	• In Spark job, tasks are executed as threads in Executor process.

PySpark Development

- terminal> python3 –m pip install pyspark
- In ~/.profile
  - export PYSPARK_PYTHON=python3
  - export PYSPARK_DRIVER_PYTHON=python3
  - export SPARK_HOME=$HOME/.local/lib/python3.6/site-packages/pyspark
  - export PATH=$HOME/.local/bin:$PATH
- terminal> pyspark
  - file = sc.textFile("/home/nilesh/spark-2.4.4-bin-hadoop2.7/LICENSE")
  - lines = file.map(lambda line: line.lower())
  - words = lines.flatMap(lambda line: line.split())
  - word1s = words.map(lambda word: (word,1))
  - wordcounts = word1s.reduceByKey(lambda acc,cnt: acc + cnt)
  - result = wordcounts.collect()
  - print(result)

PySpark Development (PyCharm)

- PyCharm -> New Project
  - Select project location
  - Existing interpreter -> Python3.x
- Create Python file (hello.py)
  - from pyspark import SparkConf
  - from pyspark import SparkContext
  - conf = SparkConf().setAppName("Demo01").setMaster("local")
  - sc = SparkContext(conf=conf)
  - file = sc.textFile("/home/nilesh/spark-2.4.4-bin-hadoop2.7/LICENSE")
  - lines = file.map(lambda line: line.lower())
  - words = lines.flatMap(lambda line: line.split())
  - word1s = words.map(lambda word: (word,1))
  - wordcounts = word1s.reduceByKey(lambda acc,cnt: acc + cnt)
  - result = wordcounts.collect()
  - print(result)

Spark RDD

- Resilient Distributed Dataset
  - Resilient
  - Distributed
  - Dataset
- RDD characteristics
  - Immutable
  - Lazily evaluated
  - Resilient

Spark RDD – creation

- sc.parallelize(collection, partitions)
  - convert collection into rdd with given partitions
- sc.textFile(path)
  - hdfs or local or s3 file or directory
- sc.wholeTextFiles(path)
  - hdfs or local or s3 directory, one file = one record
- sc.binaryRecords(path, recLen)
  - hdfs or local or s3 file or directory, recLen bytes = one record
- sc.wholeBinaryFiles(path)
  - hdfs or local or s3 directory, one file = one record
- sc.hadoopFile(path, inputFormat, …)
  - hdfs file or directory, number of partitions = number of input splits

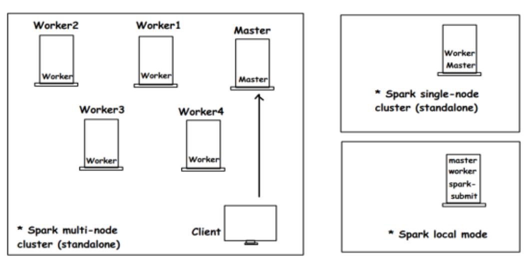
RDD Operations

- Transformations: Returns RDD
  - distinct()
  - filter()
  - map()
  - flatMap()
  - sortBy(keyValue, ascending, numOfPartitions)
  - sortByKey()
  - pipe()
  - keyBy()
  - countByKey()
  - mapValues()
  - groupByKey()
  - aggregateByKey()
  - cogroup()
  - zip()
  - join() – joins two RDDs by key.
  - reduceByKey()

RDD Operations

- Transformations
  - Narrow transformations
    - A partition of new RDD is computed from single partition of source RDD
  - Wide transformations
    - Partition of new RDD is computed from multiple partitions of source RDD
    - These transformations cause shuffling data across partitions.
- Actions: Returns non-RDD
  - count()
  - countApprox()
  - reduce()
  - countByValue()
  - first()
  - max()
  - min()
  - collect()
  - take() – collect n elements
  - takeOrdered()
  - top()
  - saveAsTextFile()
  - saveAsObjectFile()
  - lookup() – lookup by key

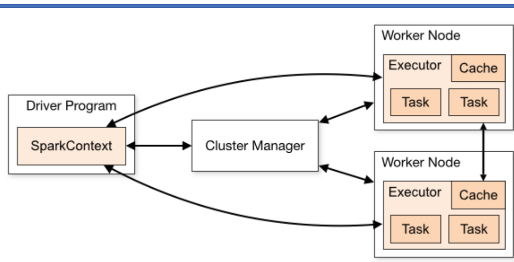
Spark Installation Modes



Spark Single Node Cluster

- Download & extract spark-x.y.z-bin-hadoop2.7.tgz.
- In ~/.bashrc
  - export SPARK_HOME=$HOME/spark-x.y.z-bin-hadoop2.7
  - export PATH=$SPARK_HOME/bin:$SPARK_HOME/sbin:$PATH
- In $SPARK_HOME/conf/spark-env.sh
  - export SPARK_MASTER_HOST=localhost
  - export SPARK_LOCAL_IP=localhost
- In $SPARK_HOME/conf/spark-defaults.conf
  - spark.master   spark://localhost:7077
- Start master & slaves.
  - terminal> start-master.sh
  - terminal> start-slaves.sh
  - terminal> jps
- Using cluster:
  - pyspark --master spark://localhost:7077
  - spark-submit

Spark Cluster Manager



Spark Execution – Terminologies

- RDD: Resilient Distributed Dataset
  - Distributed across nodes -- Partition.
- DAG: Execution plan
  - Graph: Nodes=RDDs, Edges=Operations
- SparkContext: In-charge of execution
  - Maintain all info for execution/communication
  - Creates RDDs
- Application: Set of jobs
- Job: A DAG of execution
  - An action triggers job
- Stages: DAG is divided into stages
  - Stages are separated by shuffle operations (wide transformations)
- Task: Set of operation performed in a stage on a partition
  - A thread is created for each task.

Spark Execution – Terminologies

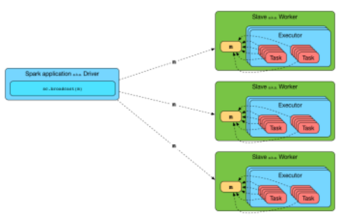
- Driver: Process that creates SparkContext
  - Create DAG
  - Submit DAG to the Master for execution.
  - Keep track of execution progress.
- Master: Cluster manager
  - It is a Java process.
  - It manage and allocate resources.
  - Spark standalone mode: Master URL = spark://master:7077/
- Worker: Each node in cluster
  - It is a Java process.
  - It executes application in separate process -- executor.
- Executor
  - It is a Java process.
  - Created by the worker for execution of application.
  - All tasks (threads) are created in executor.

Spark Multi Node Cluster (Standalone)

- On Master machine, conf/slaves make entries of all slaves.
- In all machines, conf/spark-defaults.conf
  - spark.master spark://master:7077
- On master machine, set SPARK_LOCAL_IP & SPARK_MASTER_HOST to be set (in conf/spark-env.sh) to the IP address of network.
  - export SPARK_LOCAL_IP=master
  - export SPARK_MASTER_HOST=master
- On each slave machine, set SPARK_LOCAL_IP (in conf/spark-env.sh) to the IP address of network.
  - export SPARK_LOCAL_IP=vmX
- from master
  - terminal> start-master.sh
  - terminal> start-slaves.sh
- Using spark cluster
  - terminal> spark-submit --master spark://master:7077 --deploy-mode client app.py

RDD Broadcast variable and Accumulators

- Broadcast variable is used to send data from driver code to all the workers.
  - Data will be wrapped in broadcast variable object which will be copied on all worker nodes (in executor processes).
  - It will accessed using brVar.value.
- Accumulators are job counters.
  - Used to collect info (counter type) from the workers.
  - They are incremented (using acc.add(1)) individually on worker nodes.
  - Finally collected on driver.



RDD advanced operations

- Repartitioning
  - repartition(n) – HashPartitioner & Heavy shuffle. Balanced across partitions.
  - coalesce(n) – Merge RDDs quickly to repartition into fewer partitions. May not be balanced.
- Caching & Persistence
  - rdd.cache()
  - rdd.persist(mode)
    - MEMORY_ONLY
    - MEMORY_ONLY_SER
    - MEMORY_AND_DISK
    - MEMORY_AND_DISK_SER
    - DISK_ONLY
  - rdd.checkpoint(dirpath)
    - recover from disk, even when application is terminated/aborted.
- Lineage
  - toDebugString()

SUNBEAM

Spark – Structured API

Sunbeam Infotech

Introduction

- Spark Structured API is abstraction on Lower level concepts (RDD & DAG).
- It includes: Dataframe & Dataset.

```
books = spark.read\
        .option("header", "true")\
        .option("delimiter", ",")\
        .option("inferSchema", "true")\
        .csv("/path/to/books_hdr.csv")

result = books\
        .select("subject", "price")\
        .groupBy("subject").sum("price")\
        .orderBy("subject")
```

SparkSession

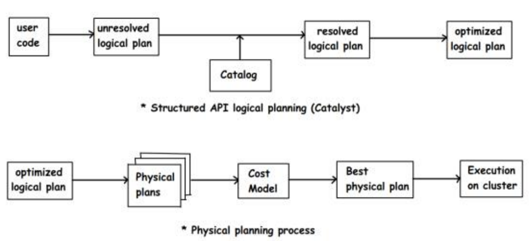
- Wrapper on SparkContext. It can encapsulate additional contexts as needed e.g. SQLContext, HiveContext, StreamingContext, ...
- Spark 2.4 deprecates SparkContext.
- SparkSession is singleton i.e. one application will have single SparkSession.
- It is created using builder design pattern.

```
spark = SparkSession.builder()\
       .appName("myApp")\
       .master("local[*]")\
       .getOrCreate()
…
spark.stop()
```


Data Frames

- Created using SparkSession.
- Abstraction/wrapper on RDD.
- Similar to Pandas dataframes or R dataframes or RDBMS table.
- Dataframe have structure (metadata) and rows & columns (data).
- The operations on dataframes is similar to SQL operations e.g. select(), groupBy(), orderBy(), limit(), where(), join(), ...

Spark Dataframe Execution



Dataframe creation

- Using DataFrameReader
  - df = spark.read.format("csv").option("key", "value").load()
  - df = spark.read.option("key", "value").csv("path")
- From Java List<Row>, RDD<Row>, NamedTuple or Dict
  - df = spark.createDataframe(data, schema)
- Schema can be inferred or can given manually.
  - df = spark.read.schema(my_schema).option("key", "value").csv("path")

Dataframe columns

- Columns are expressions.
- Expression can be column name or some arithmetic expression or some sql fn processing expression.
  - e.g. "job", "sal", "sal + comm", "sal + ifnull(comm,0)", "1 as one", "*", ...
- Selecting columns/expression
  - df.select(c1, c2, ...)
  - df.selectExpr(e1, e2, ...)
  - df.withColumn("colname", "expression") -- add extra column
- Drop column
  - df.drop("colname")

Dataframe rows

- Internally Dataframe is RDD or Row type.
- Row is StructType.
  - e.g. Row(job='CLERK', sum(sal)=4150.0, sum(comm)=None, sum(income)=4150.0)
- Individual column in Row can be accessed using index [n].

Dataframe operations

- DF operations are transformations or actions.
  - Transformations produce new dataframe.
  - Actions cause execution plan preparation & execution.
- Transformations
  - select(), selectExpr(), where()
  - orderBy(), sort() -- asc/desc and one/more columns
  - limit(), distinct()
  - groupBy("col").someAggOp("col")
  - join()
  - repartition(), coalesce()
- Actions
  - df.show()
  - df.first(), df.take(), df.collect()
  - df.write.format("csv").option("path", "dirpath").save(), df.write.csv("dirpath")
  - df.write.saveAsTable()

Spark SQL Functions

- Numeric functions
  - abs(), floor(), ceil(), round(), pow(), ...
- String functions
  - substring(), lower(), upper(), concat(), ...
- Null value functions
  - ifnull(), isnull(), ...
- Date Time functions
  - from_unixtime(), to_timestamp(), to_date(), ...
  - current_date(), current_timestamp(), ...
  - date_diff(), ...
- Aggregate functions
  - sum(), avg(), count(), min(), max(), stddev_pop(), corr(), ...
- Complex types
  - explode(), array_contains(), ...
- Window functions
  - rank(), dense_rank(), ...

Spark data formats

- Hive use SerDe to write/read data from hive table.
- Dataframes are created using DataframeReader (spark.read) and can be saved using DataframeWriter (df.write).
- Supported formats
  - csv, json, text
  - orc
    - columnar file format
    - designed & optimized for hive
  - parquet
    - columnar file format
    - designed & optimized for spark
    - default format (i.e. if no format is mentioned)
    - efficient than CSV/JSON data.
    - parquet-cli is python package to read parquet file format.
  - jdbc
    - read/write data from/to RDBMS.

Spark SQL

Sunbeam Infotech

Introduction

- Based on Spark structured API i.e. dataframes.
- Enable writing SQL queries on Spark dataframes as views/tables.
- Before Spark 2.x, SQLContext provides SQL functionality.
- Spark 2.x SparkSession encapsulate SparkContext.
- SparkContext use Hive metastore to maintain metadata.

Spark Tables

- Spark dataframes can be saved as table.
  - df.saveAsTable("tablename")
  - Table metadata is stored in metastore and data stored in spark warehouse directory.
- Spark tables can be partitioned by one or more column.
  - df.write.partitionBy("col_name").saveAsTable("part_tablename")
  - Partitions are sub-directories (directory name col=value) in which data is divided by column value.
- Spark tables can be bucketed by a column.
  - df.write.bucketBy(numOfBuckets, colname).saveAsTable("buck_tablename")
  - Buckets divide data into multiple data files by column value.
- Spark tables can be partitioned as well as bucketed.
  - emp.write. partitionBy("col1").bucketBy(numOfBuckets, col2).saveAsTable("tablename")
- Buckets are supported only as spark managed tables.

Spark Views

- View is abstraction on spark dataframes.
- Created using df.createOrReplaceTempView("viewName")
- createOrReplaceTempView()
  - Creates view if not available.
  - If available, replace with new view.
- View treats dataframe as in memory table & create a view (like SQL view) to fire SQL queries on it.
- The temporary view is in memory only, its info not stored in metastore. It is attached to current sparkSession.
- df.createOrReplaceGlobalTempView("viewName") creates global view, which can be shared across multiple sessions.

Spark SQL – setup

- Copy hive-site.xml into $SPARK_HOME/conf
  - javax.jdo.option.ConnectionURL = spark/hive metastore path (derby/mysql)
  - javax.jdo.option.ConnectionDriverName = derby/mysql driver
  - javax.jdo.PersistenceManagerFactoryClass = persistence manager factory
  - hive.metastore.warehouse.dir/spark.sql.warehouse.dir = local/hdfs directory path
- Start spark master and slaves.
  - start-master.sh
  - start-slaves.sh
- Start spark thrift-server.
  - start-thriftserver.sh
- Start spark beeline.
  - beeline -u jdbc:hive2://localhost:10000 -n $USER

Spark SQL architecture

Spark Hive Integration

- Spark metastore is compatible with Hive 1.2.1.
- Spark can access tables from Hive directly. However all dependencies of Hive are not shipped with Spark.
- To access Hive tables from spark application, Hive config should be associated with application and HiveContext should be activated.

```
spark = SparkSession.builder.appName("app")\
  .config("javax.jdo.option.ConnectionURL","jdbc:derby:;databaseName=/path/to/metastore")\
  .config("javax.jdo.option.ConnectionDriverName", "org.apache.derby.jdbc.EmbeddedDriver")\
  .config("hive.metastore.warehouse.dir", "/path/to/spark-warehouse")\
  .enableHiveSupport().getOrCreate()

tables = spark.catalog.listTables()

books = spark.read.table("sbooks")

spark.stop();
```


Spark Streaming

Sunbeam Infotech

Batch processing vs Stream processing

• Processing finite set of data (data at rest).	• Processing live stream of data (data in motion).
• Incremental data load is managed by programmer.	• Data processing is managed by framework.
• Cluster should be planned as per data size. High throughput.	• Less throughput.
• Job run once by batch.	• Job is running forever.

Stream processing

- Applications
  - Notifications & Alerts: Shipping alert, Fire alert, ...
  - Incremental ETL: Load live data from twitter/fb and process, ...
  - Real time reporting: Live dashboard, ...
  - Real time decisions: Customer management, ...
  - Online ML: Training ML model with live data, fraud detection, ...
- Advantages
  - Batch processing need to execute periodically (manually or scheduler).
  - Processing with lower latency.
  - Efficient handling of Incremental data.

Stream processing

- Challenges of stream processing
  - Maintain large amount of state.
  - Data throughput.
  - Exactly once processing.
  - Process out-of-order data.
  - Low latency processing.
  - Load imbalance.
  - Join with external data.
  - Producing output.
- Design considerations
  - Record at a time vs Declarative APIs
  - Event time vs Processing time
  - Continuous processing vs Micro-batch processing

Stream processing

Spark Streaming

- Spark is originally designed by micro-batch processing.
- Spark Streaming APIs
  - Spark DStream
  - Spark Structured Streaming

Spark DStream

- Micro-batches of RDDs (Small RDDs).
- Developed in 2012. Most popular Streaming framework in 2016.
- RDD based programming.
- Limitations
  - Based on RDDs (in JVM). Not efficient in Python.
  - No support for event time processing.
  - Only micro-batch processing.
- Examples
  - Twitter stream
  - Socket stream processing

Spark Structured Streaming

- Developed in 2016.
- Stable in Spark 2.2.
- Spark Structured Streaming is based on dataframes.
- Works seamlessly with other Spark APIs i.e. Spark SQL & Spark ML.
- Advantages
  - Optimized (Catalyst engine)
  - Event time processing is supported
  - Support for continuous processing
  - Same query/code works for batch processing & stream processing
  - Exactly once processing mode is available
  - Fault tolerance

Spark Structured Streaming

- Spark Structured Streaming consider dataframe to be unbounded (infinitely growing).
- Transformations & Actions
  - Transformations are same as spark dataframe. Few transformations are not yet implemented.
  - Action is starting the stream & print results.
- Input sources
  - socket, rate, files, flume, kinesis, kafka
- Output sinks
  - console, memory, files, flume, kafka, foreach

Spark Structured Streaming

- Output modes
  - Every mode is not supported for every type of query.
  - append: output result of current micro-batch is available. (not supported for aggregate operations).
  - complete: complete result including prev result & current micro-batch result is available.
  - update: only results modified in current micro-batch are available.
- Triggers
  - By default, micro-batches are processed one after another.
  - Trigger can specify time duration after which each batch is to be processed.
- Event time processing
  - Time at which event is generated at source, is "event time".
  - Can process out-of-order data.
  - Watermark feature is used define for how much time data should be considered (how much time should be wait before processing data).

SUNBEAM

Apache Kafka

Sunbeam Infotech

Introduction

- Kafka is a distributed messaging system.
- Developed at LinkedIn and open sourced in 2011.
- Implemented in Scala.
- Used by LinkedIn, Twitter, Uber, Yahoo, airbnb, …

- https://www.slideshare.net/jhols1/kafka-atlmeetuppublicv2
- https://www.slideshare.net/AkashVacher/kafka-meetup-ppt-1

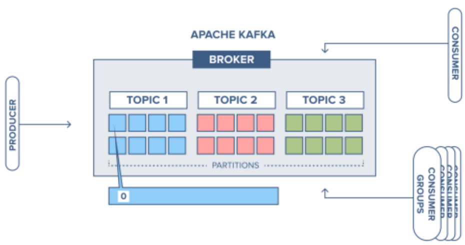
Apache Kafka

- Advantages
  - High throughput
  - Disk based
  - Durable
  - Scalable
  - Low latency
  - Finite retention
  - Strong ordering
  - Exact once delivery
- Applications
  - Stream processing
  - Log processing
  - Notifications
  - Customer tracking

Kafka Terminologies

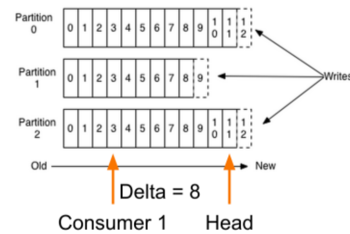
- Broker
  - Each node in kafka cluster is called as "kafka broker".
  - Each broker have its own id (configured in $KAFKA_HOME/config/server.properties).
  - Broker is JVM process -- Kafka -- running on port 9092.
- Topic
  - The messages in kafka are logically divided under topics.
  - The topics are configured as per need of application.
- Partitions
  - Topics are physically divided under partitions.
  - The messages data is actually stored into log files (.log) for each partition under /tmp/kafka-logs (default directory).
  - Each partition each message is numbered (indexed) from 0,1,2,... called as "offsets".
  - Partitions have replication factor. If replication factor is 3, partition data will be copied on two more brokers.
  - In case of replication of partitions, one of the partition is leader while others will be followers/replicas. These replicas are always try to be in sync with leader.
  - The replicas which are not too behind the leader and said to "in-sync replicas" (ISRs).

Kafka Terminologies



Sunbeam Infotech www.sunbeaminfo.com

Kafka Topic



Sunbeam Infotech www.sunbeaminfo.com

Kafka Terminologies

- Producer
  - Client process of kafka which send messages to the broker, is said to be producer.
  - These client processes can be implemented in Java, Python, C/C++, ...
  - Producer publish data to the topics, which is sent to partitions, so that partitions are load balanced. It may use round-robin algorithm or may do key-based division.
  - Producer always write data to the leader of the partition.
- Consumer
  - Client process of kafka which receive messages from the broker, is said to be consumer.
  - These client processes can be implemented in Java, Python, C/C++, ...
  - Clients subscribe to the kafka topics.
  - It will read the data from any of the replica, which have data required for that client.
  - The data in a partition is guaranteed to be received in sequence (of writing). However data in different partitions is not guaranteed to read in sequence of writing.

Sunbeam Infotech www.sunbeaminfo.com

Kafka Terminologies

- Consumer groups
  - Each topic can have one or more consumers -- divided into one or more consumer groups.
  - Consumer group -- kafka ensure than only one consumer from a consumer group will get a message.
    - if there are three consumer groups as follows:
      - G1 -- C1a, C1b, C1c
      - G2 -- C2a, C2b
      - G3 -- C3a, C3b, C3c, C3d
    - If message "m0" is received into kafka, it will be received by exactly one consumer from each group.
  - Possible organization of consumers in a group:
    - example1:
      - G1 -- C1
      - G2 -- C2
      - G3 -- C3
      - each message is received by each consumer.
      - broadcast system.
    - example2:
      - G1 -- C1a, C1b, C1c
      - each message is received by only one consumer.
      - parallel processing of records.

Sunbeam Infotech www.sunbeaminfo.com

Kafka Installation

- Download and extract Kafka.
- In ~/.bashrc
  - export KAFKA_HOME=/path/to/kafka
  - export PATH=$KAFKA_HOME/bin:$PATH
- In $KAFKA_HOME/config/server.properties
  - Uncomment line → listeners=PLAINTEXT://:9092
- Start Kafka
  - terminal> zookeeper-server-start.sh $KAFKA_HOME/config/zookeeper.properties
  - terminal> kafka-server-start.sh $KAFKA_HOME/config/server.properties

Sunbeam Infotech www.sunbeaminfo.com

Using Kafka

- Create topic
  - kafka-topics.sh --zookeeper localhost:2181 --create --topic spark03 --replication-factor 1 --partitions 2
  - Verify: ls /tmp/kafka-logs
- List topics
  - kafka-topics.sh --zookeeper localhost:2181 --list
- Write to topic (producer)
  - echo "one" | kafka-console-producer.sh --broker-list localhost:9092 --topic spark03
- Read from topic (consumer)
  - kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic spark03
  - kafka-console-consumer.sh --bootstrap-server localhost:9092 --partition 0 --offset 2 --topic spark03
  - kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --topic spark03
- Consumer Groups
  - kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic spark03 --group 1
  - kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic spark03 --group 1
  - kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic spark03 --group 2

Sunbeam Infotech www.sunbeaminfo.com

Spark Kafka Integration

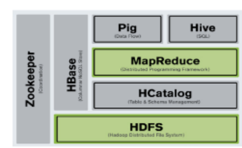
- Spark dataframes are divided into the partitions. When dataframe created from kafka, one dataframe partition is mapped to one kafka partition.
- Spark dataframe partitions are on workers i.e. each worker will read corresponding partition from kafka topic partition. Each worker will also maintain its own offset.
- When a streaming query is executed, a consumer is created.
  - one query(job) = one consumer group --> each group will have consumers equal to number of workers used to read from kafka.
- If two different jobs are processing two different kafka topics, number of dataframe partitions in each job is equal to number of kafka partitions in that topic. All tasks (threads) processing the partitions corresponding to one topic will be in one consumer group.
- If two different jobs are processing same kafka topic (for doing different processing), still there will be two different consumer groups. Each group will contain number of consumers equal to number of dataframe partitions (which in turn equal to number of kafka partitions). Note that data will be read twice from each kafka partition to create two different dataframe partitions (one for each job).

Apache HBase

Sunbeam Infotech

HBase Introduction

- HBase is an open-source non-relational distributed database.
- It is modeled after Google's Big-Table and open-source.
- HBase is developed in Java as part Apache Hadoop project.
- HBase contributed by developers at Facebook, Cloudera, Hortonworks, etc.
- It is cluster based database running on top of Hadoop HDFS.
- HBase data files are stored in HDFS.
- HBase development:
  - 2006: Google Big Table paper
  - 2007: Hadoop's contrib
  - 2008: Hadoop's sub-project
  - 2010: Apache top level project
  - 2011: HBase 0.92 release



HBase vs HDFS

- Distributed systems to scale to thousands of nodes.
- HDFS – Batch processing over big files
  - Not good for record lookup.
  - Not good for small incremental batches.
  - Not designed for update & delete.
- HBase – Distributed Column database
  - Low latency record lookup (by row id/key).
  - Support for inserting & updating records.

	HDFS + MR	Hbase
Write pattern	Append only	Random write, Bulk loading
Read pattern	Scan whole file	Random read, small range read or full table scan
SQL performance	Very good	Slower
Structured storage	User-defined, Avro or Sequential files	Sparse column family data model
Max data size	30+ PB	~ 1 PB

HBase vs RDBMS

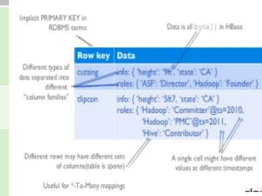
	RDBMS	HBase
Data layout	Row oriented	Column oriented
Transaction	Multi-Row ACID	Single Row only
Query Language	SQL	get/put/scan/…
Security	Authentication	HDFS+Auth
Indexes	On any column	Only on row-key
Max data Size	TBs	~ 1PB
Read/Write Throughput	1000 queries/sec	million queries/sec

HBase - NoSQL

- HBase being NoSQL, is a schema-less database. Columns can be added on the fly.
- Sparse tables have lot of *null* values and not stored by HBase to save disk space.
- It persists all its data in underlying HDFS. Hence it is reliable, scalable, high performance at cost of distributed servers.
- Each record is associated with a key and is stored in sorted order of keys.
- Data store can store one or more tables.
- Each row in table is indexed by row key.
- Column oriented database:
  - Each table can have one or more column families.
  - Each column family can have one or more columns.
  - Columns in family may be different for each row.
- Columns can be added in family dynamically.
- Multiple versions of the values (for each update) as per timestamps (by default).

HBase Data Model

Row ID	Column Family -- Coulmn Qualifiers
001	name : { fname : 'nilesh', lname : 'ghule' } details : { email : 'nilesh@sunbeaminfo.com', mobile : '9527331338', mobile2: '7722093091' }
002	name : { fname : 'nitin', lname : 'kudale' } details : { mobile : '9881208115' }
003	name : { fname : 'sunbeam' } details : { site : 'www.sunbeaminfo.com' mobile : '9881208115', mobile2: '9881208114' phone : '02024260308' }



HBase Data Model

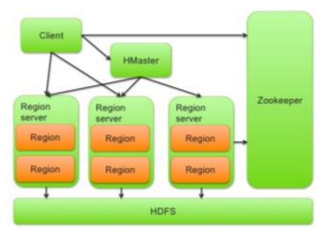
- Intersection of row and column is a cell.
- All cells, row ids, even table, column family & column names are stored as byte array.
- Thus any data type of any size can be stored in each cell.
- With each edit, a new version of the cell is created with new time stamps. Internally stores versions in desc order of time-stamps.
- Key & Version numbers are replicated with each column family.
- Empty cells aren't stored.

Sorted on disk by Row key, Col key, descending timestamp

Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

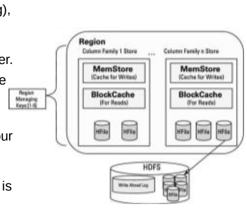
HBase Data Storage

- Different column families can have different properties and access patterns.
- Configurable column properties are cache usage, compression (none, gzip, LZO), version retention policies.
- Each column family is stored in a separate file (called HFile), which is then partitioned into **regions**.



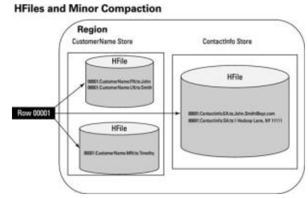
HBase RegionServers

- They are software processes/daemons responsible for store and retrieve data in HBase. Running on each node in cluster.
- When a table grows beyond threshold (as per config), it is auto split & distributed the load to another node. This is called auto-sharding.
- Each split is separate region managed by a rgn server.
- Each column family store has a read cache called the BlockCache and a write cache called the MemStore.
- BlockCache helps with random read performance.
- The Write Ahead Log (WAL, for short) ensures that our Hbase writes are reliable.
- The design of HBase is to flush column family data stored in the MemStore to one HFile per flush. HFile is finally stored in HDFS blocks.



HBase Compactions

- Compaction, the process by which HBase cleans up itself.
- Minor compactions combine (a configurable number of smaller HFiles into one larger HFile. Due to combining data together, disk access speed increases.
- Major compaction combine all HFiles into one large HFile. It also removes extra versions & deleted cells as per config.



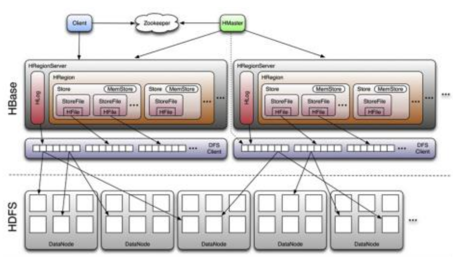
HBase Master

- Monitor region servers in cluster.
- Handle metadata operations.
- Assign regions (after split) & balance load.
- Manage region server failover.
- Manage and clean catalog tables.
- Clear the WAL (Write Ahead Logs).
- Usually a back copy of master server is maintained to handle failover of master.

HBase ZooKeeper

- ZooKeeper is a distributed cluster that provides reliable coordination & synchronization services for clustered applications.
- This team consists of ZooKeeper Leader and ZooKeeper Follower(s).
- HBase comes with an instance of ZK that coordinates operations of master server, region server(s) and client(s).

HBase on HDFS



HBase: CAP and ACID properties

- C : Consistency
  - HBase is consistent as same data is visible from any node (because of HDFS).
- P : Partition Tolerance
  - HBase it tolerant i.e. if any node goes down the data can be still accessible (because of Replica).
- A : Available
  - HBase doesn't guarantee 100% uptime i.e. each request may not get response (success/failure).

- HBase is not ACID compliant like RDBMS as it doesn't support Isolation.
- A : Atomic - HBase guarantees update single record atomically
- C : Consistent - HBase is in consistent stage - No PK/FK relations.
- D : Durable - HBase changes are durable in HDFS.
- I : Isolation - HBase doesn't guarantee changes sequentially.

HBase Access

- HBase shell - ruby shell with set of commands
- Web interface - browser on port 60010
- Java API - native APIs for HBase
- REST (HTTP) - REST API on port 8080
  - hbase rest start -p <port>
- Thrift - enable access from other language
- Hive/Pig for Analytics

- HBase Shell Commands:
  - https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/

HBase Applications

- HBase is preferred for random reads, random writes or both.
- Do not use HBase if random access is not required.
- HBase can perform thousands of operations per seconds on TBs of data.
- HBase performs well is access pattern is well known & simple.
- Used by: Facebook, Mozilla, Twitter, OpenLogic, Meetup, …

Apache Airflow

Sunbeam Infotech

Introduction

- A platform created by community to programmatically author, schedule and monitor workflow.
- Airflow is used to develop, orchestrate and monitor complex ETL pipelines.
- Airflow is completely developed in Python, so it can work with any of the Python library.
- Initiated by Airbnb in 2014 and open-sourced under Apache in 2016.
- Apache top level project in 2019.
- Used by more than 250 companies world-wide - Amazon, Citi, JPMorgan, Salesforce, Drillinginfo, Unitedhealth, ...
- Managed services in GCP and AWS cloud.

Traditional ETL

- Extract, Transform and Load
- Common schedulers: cron, oozie (hadoop), luigi (spotify)
- BigData has complex pipelines.
- Cron limitations
  - Error handling
  - Not maintainable (runs on a machine)
  - Execution dependency (delayed tasks)
  - Transperency (No centralized log)
  - Task tracking (No centralized monitoring)
  - Handle historical data
- Oozie
  - Error handling (n retries)
  - Execution dependency
  - Better tracking & monitoring
  - XML based jobs - difficult to read/understand

Airflow Terminologies

- DAG
  - Directed Acyclic Graph
  - Vertices are tasks and Edges are dependencies
  - Can be parallelized.
- Operators
  - Single dedicated task in workflow/DAG.
  - Example: Bash command, Python Function, Database Operation, Email send, etc.
- Task
  - Instantiated operator and assigned to some worker.
  - Failed task can be retried.
- Workflow
  - Sequence of task arranged in control dependencies.
  - Workflow and DAG words are interchangeable.

Airflow DAG

Airflow Workflow

- Airflow workflows are written as Python code in form of DAG.
  - DAG includes multiple tasks in form of operators.
  - Operators are connected in complex sequence.
- DAG implemented in Python, but can execute variety of tasks.
  - Programs written in any language.
  - Analytical/ML task
  - ETL task
  - Big Data components/programs
- DAGs are more dynamic, manageable, testable and collaborative.
- Airflow is job scheduler that execute the tasks on defined time and/or followed by its dependencies.

Airflow Advantages

- Dynamic DAG
  - Implemented in code - flexible
  - Can be configured: parallelism, params, templates
- Extensible
  - Plenty of operators/executors
  - Shell script, Big data task, OS command, etc.
  - Custom task/plugins
- Scalable
  - Modular architecture
  - Can handle any number of DAGs (with multi-node cluster)
- Configurable
  - airflow.cfg - admin settings
  - Centralized configurations
- Monitoring
  - Elegant Web UI
  - Handle failures (n retries)
  - Email alerts
- Open Source
  - Active community
  - New plugins

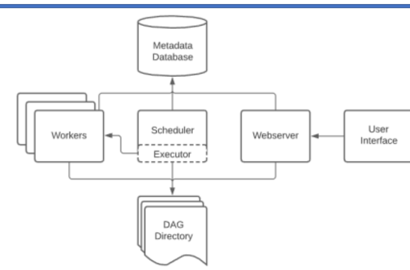
Airflow Applications

- Task scheduler
- ETL pipelines
- Periodic backups
- Automate DevOps operations
- ML jobs
- Recommendation engines
- ...

Airflow Installation

- Airflow is installed as Python package – refer docs.
- Airflow directory/files
  - config
    - airflow.cfg
      - dags folder
      - timezone
      - executor type
      - parallelism
  - dags
    - dag definition (python files)
  - script
    - entrypoint.sh (with docker image)
    - to initiate airflow - web-server & scheduler.
    - enviroment variables
  - metastore
    - airflow.db
      - using sqlite (default) database for airflow metastore.

Airflow Architecture



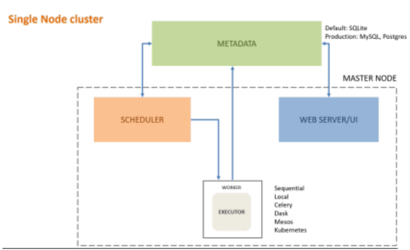
Airflow Architecture

- Metadata
  - RDBMS that stores historical and current DAGs and tasks details.
  - Also maintains information used resources.
  - Default RDBMS used is SQLite, but it is single-user database.
  - In production MySQL or Posgre-SQL is preferred option.
- Scheduler
  - Instructs and trigger task execution on worker node.
  - Implemented into Python.
  - Reads DAG file, task config and schedules the task on worker nodes in sequence.
  - It monitor tasks state from metadata and handle the task failure (as per config).
- Web Server
  - Flask-based UI for Airflow to monitor DAG.
  - Communicate with metadata to fetch the task state.
  - Task state is rendered in various formats including DAG, graphs, time/numbers, etc.

Airflow Architecture

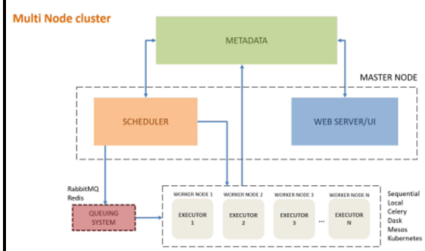
- Executor
  - Carry the task scheduled by the scheduler.
  - Runs on Worker node.
  - Various types:
    - Sequential (default) - For testing
    - Local - For single node cluster - Use python process.
    - Celery - Recommended for multi-node cluster.
      - Job queue written in Python.
      - Used in scalable distributed system.
    - Dask
    - Mesos
    - Kubernetes

Airflow – Single node installation



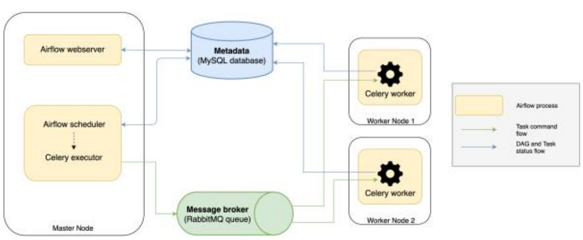
- Single worker node.
- Not scalable (max all resources available on the system).
- Quite fast for limited number of DAGs and data size.
- Use local executor.
- Direct communication between scheduler and executor.

Airflow – Multi node installation



- Master node runs scheduler and web-server.
- Multiple worker nodes - running executors.
- Highly scalable for huge data and many nodes.
- Recommended executor is Celery.
- Scheduler and workers communicate with external queue system like Rabbit MQ / Redis.

Airflow – Multi node installation



Airflow Working

- Scheduler periodically pings for DAG folder and communicate with metastore.
- If any DAG is available for execution, scheduler starts a DAG run for it. DAG run is an object representing an instantiation of DAG.
- Scheduler update DAG state as "Running".
- For each task, task object is instantiated. The task state is updated as "Scheduled".
- Scheduler assigns priority to each task in DAG (as per config) and push them into queuing system. The task state is updated to "Queued".
- Worker pull the task from queue, set its state as "Running" and start executing it.
- Upon completion of each task, task state is updated as succeed or failed.
- When all tasks in DAG run are executed, status of DAG run is updated as succeed or failed.
- Airflow web-server periodically get the data from meta-store and render it for users.
- If any new DAG is found it DAGs folder, scheduler begin its execution (as above).

Airflow Operators

- DAG only describe how to run the workflow, but do not perform actual computation.
- Operator describes single task in workflow.
- Operator characteristics
  - Atomic (usually) - standalone (not sharing resources with other operators)
  - Idempotent - produce same result for each run.
  - Operator in execution (instantiated) is referred as task.
  - Inherited from airflow's BaseOperator class.
  - Operators can communicate using XCom.
- Operator categories
  - Sensor operators - Wait for certain criteria
    - HdfsSensor, FileSensor, ...
  - Transfer operators - Transfer the data
    - MySqlToHiveOperator, ...
  - Action operators - Do specified task
    - BashOperator, PythonOperator, HiveOperator, MySqlOperator, EmailOperator, ...

SUNBEAM

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>