

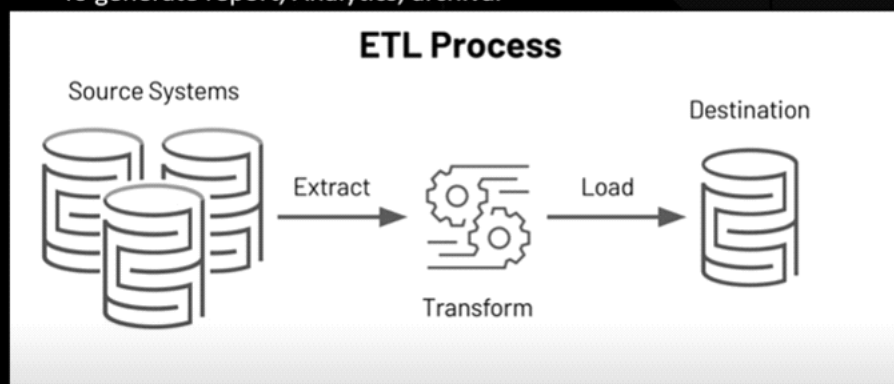
Spark Introduction

Wednesday, March 23, 2022 12:25 PM

- Data engineering means designing and building systems for collecting , storing , analyzing data. It's main tasks are data pipeline creation , data validation and data quality and other tasks like data archival.
- Data Engineering skills (Programming languages like Scala and python, SQL languages like RDBS or NoSQL , Cloud tools)
- We connect database to the front end using local host number and specific connection username and password.
- OLTP stores only transactional data, it doesn't store any other un important details.
- To generate the report oriented data we use OLAP database. OLAP is used to store the summary type of the dataset.
- In OLTP they do multiple vertical bisection to split the data.
- In transactional level at RDBMS we do normalization to stop the repetition data.
- In OLAP we try to minimizing the need for joins, reducing the number of tables, queries to be retrieved can be simpler, retrieving data is faster due to fewer joins.

ETL

- Daily Batch will move data from OLTP to OLAP
 - To generate report, Analytics, archival



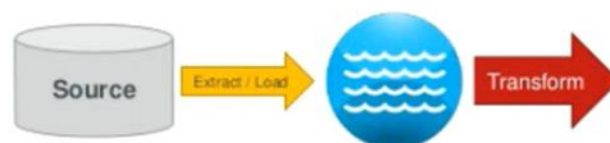
- In Transform phase we use those de-normalizing rules as discussed above.
- Data Lake: We have dumped raw data across the organization
- Data Warehouse: Here we will have data as summarized way. We will have current and historic data in this place.

Data Warehouses use a traditional ETL Process



Data is transformed when it enters the data warehouse

Data Lakes make use of the ELT Process



Data is transformed when it is retrieved from the data lake

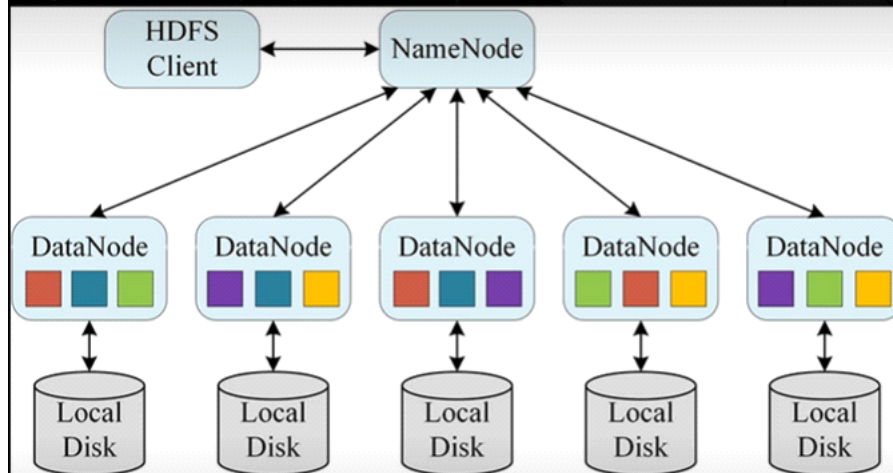
- New data formats that we are using now to share the data in big data operations are Parquet, Avro , ORC. And even we are having new type of file compressors.

Hadoop Ecosystem

- Distributed Processing across cluster of computers, distributed file system(Horizontal Scaling)
- Hadoop 1 (Deprecated)
 - Resource Manager and Data Processing (Mapreduce) are coupled
- Hadoop 2
 - YARN and MapReduce
- Some Tools in Hadoop Ecosystem :
 - Hive (Warehouse), SQOOP (Ingestion), Mahout (ML)
 - HDFS (Hadoop Distributed File System)

HDFS

Data partitioned in to blocks (128MB)



MapReduce: Map does the operations which happens inside a data block or partition (where we are having every record assigned with a key) And reduce does the aggregation part.

Cluster Manager: Manages cluster of computer and their resources like CPU, memory , storage, ports and other available resources. YARN is widely used cluster manager.

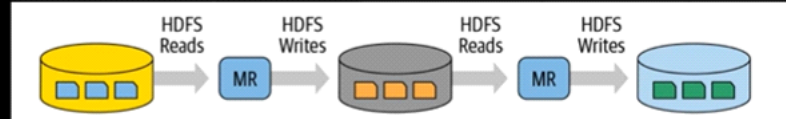
Hive

- Distributed, Fault tolerant data warehouse system
- SQL Like Interface
- Uses Tez, Spark, MapReduce as an engine to query data
- Schemas are created over data
 - Hive Managed table
 - External Table
- Bucketing
- Partitioning

- If we make any change on hive managed table we change the original data, where as external table the original data will not be affected.

Problem with Hadoop

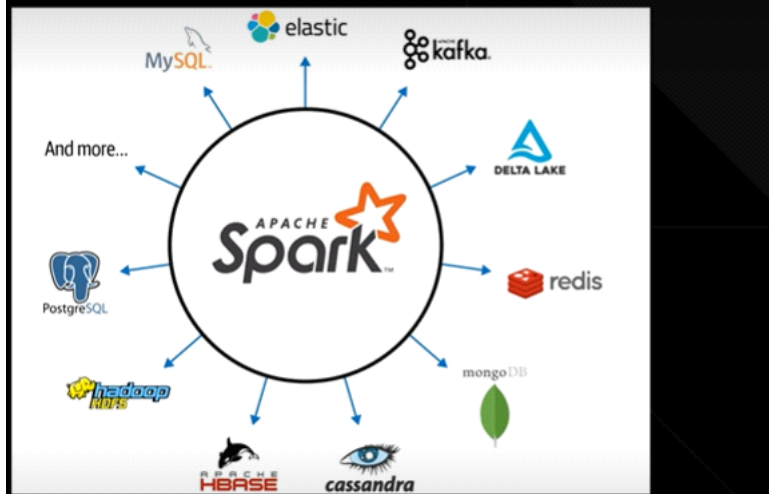
- No Caching



- Intermediate results need to be saved to disk

- No Support for Streaming

Apache Spark – Supported Datasources



Spark App – Package & Execution

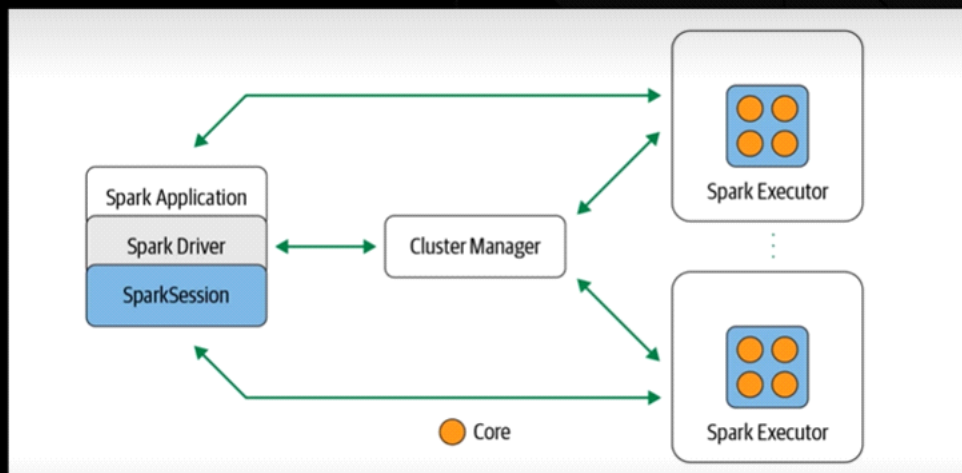
- Packaging

- Java & Scala Spark – jar
- PySpark – Zip file, Egg file
- Can be executed in Notebooks

- Execution

- Local mode – IDE Run
- Cluster mode - Trigger Spark app with API/schedule with tools like oozie/airflow

Spark Architecture



- Driver instructs the executor with the help of cluster manager. The working of cluster manager is an internal process.

- **Spark Driver**
 - Responsible for instantiating Spark Session.
 - Communicates with Cluster manager and request resources
 - Converts code into DAG , schedules the task, distributes across the executors
 - Communicates with executor
- **Spark Executor**
 - Runs on each worker node (requested by Spark driver)
 - Task execution – Sent from driver

Spark Cluster managers: Built in Standalone applications ,YARN , Mesos, Kubernetes .

Spark Application Entry Point

appName, master

• Spark 1.x

- SparkContext
- SQLContext
- HiveContext

• Spark 2.x

- SparkSession

```
conf = SparkConf() \
    .setAppName('app') \
    .setMaster(master)
sc = SparkContext(conf=conf)

from pyspark import SparkContext, HiveContext

conf = SparkConf() \
    .setAppName('app') \
    .setMaster(master)
sc = SparkContext(conf)
hive_context = HiveContext(sc)
hive_context.sql("select * from tableName limit 0")

from pyspark.sql import SparkSession

spark_session = SparkSession \
    .builder \
    .enableHiveSupport() \
    .getOrCreate()

# Two ways you can access spark context from spark session
spark_context = spark_session._sc
spark_context = spark_session.sparkContext
```

Spark Application - Design

- **Spark session creation**
 - E.g. `spark=SparkSession.builder.appName("Sample App").master("local").getOrCreate()`
- Read
- Transformation
- Write

- **RDD**
 - Low level API in Spark which holds
 - Dependencies Info
 - Partition Info
- **DataFrame :**
 - Inspired by Pandas
 - Distributed in-memory tables with columns and schema
 - Support to perform SQL operations
 - Dataset[Row] – Row(untyped JVM Object)
- **Dataset**
 - Imposing domain object
 - Can use language-native Expressions
 - Useful when complex business logic needs to be applied

Transformations	Actions
<code>orderBy()</code>	<code>show()</code>
<code>groupBy()</code>	<code>take()</code>
<code>filter()</code>	<code>count()</code>
<code>select()</code>	<code>collect()</code>
<code>join()</code>	<code>save()</code>

- Can be created from `sc` (SparkContext)
 - `sc.parallelize(List()/Seq())`
 - `sc.textFile()`
 - `sc.wholeTextFiles()`
- Or from `<dataframe>.rdd`
- RDD should be use when we want to use accumulator (counter) and broadcast variables.
- Benefits from Spark Optimizations aren't available.

RDD: Row delimiter

RDD Transformations	
Transformation	Meaning
map (<i>func</i>)	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .
filter (<i>func</i>)	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.
flatMap (<i>func</i>)	Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).
mapPartitions (<i>func</i>)	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T.
coalesce (<i>numPartitions</i>)	Decrease the number of partitions in the RDD to <i>numPartitions</i> . Useful for running operations more efficiently after filtering down a large dataset.
repartition (<i>numPartitions</i>)	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.
reduceByKey (<i>func</i> , [<i>numPartitions</i>])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type <code>(V,V) => V</code> . Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.

RDD Actions

Action	Meaning
reduce(func)	Aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(n)	Return an array with the first <i>n</i> elements of the dataset.
foreach(func)	Run a function <i>func</i> on each element of the dataset. This is usually done for side effects such as updating an Accumulator or interacting with external storage systems.
saveAsTextFile(path)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.

Dataset samples

Sample data

```

1 The MIT License (MIT)
2 Copyright (c) <year> <copyright holders>
3
4 Permission is hereby granted, free of charge, to any person obtaining a copy
5 of this software and associated documentation files (the "Software"), to deal
6 in the Software without restriction, including without limitation the rights
7 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8 copies of the Software, and to permit persons to whom the Software is
9 furnished to do so, subject to the following conditions:
10
11 The above copyright notice and this permission notice shall be included in
12 all copies or substantial portions of the Software.
13
14 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
18 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
19 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
20 THE SOFTWARE.

```

date	store	quantity	amount
1/1/2021	StoreA	320	10000
1/1/2021	StoreB	100	2000
1/1/2021	StoreC	100	5000
31/1/2021	StoreA	20	1000
31/1/2021	StoreB	100	200
31/1/2021	StoreC	200	15000
1/2/2021	StoreB	20	2000
1/2/2021	StoreC	1000	50000
28/2/2021	StoreA	40	750
28/2/2021	StoreB	10	1785
1/3/2021	StoreA	200	2100
1/3/2021	StoreC	100	6700
31/3/2021	StoreA	40	750
31/3/2021	StoreB	10	1500

```

22/03/17 05:58:14 INFO TaskSetManager: Starting task 14.0 in stage 111.0 (TID 1558, 172.30.4.8, executor 1, partition 14, PROCESS_LOCAL)
22/03/17 05:58:14 INFO TaskSetManager: Starting task 15.0 in stage 111.0 (TID 1559, 172.30.4.7, executor 2, partition 15, PROCESS_LOCAL)
22/03/17 05:58:14 INFO TaskSetManager: Starting task 16.0 in stage 111.0 (TID 1560, 172.30.4.11, executor 0, partition 16, PROCESS_LOCAL)
22/03/17 05:58:14 INFO TaskSetManager: Starting task 17.0 in stage 111.0 (TID 1561, 172.30.4.8, executor 1, partition 17, PROCESS_LOCAL)
22/03/17 05:58:14 INFO TaskSetManager: Starting task 18.0 in stage 111.0 (TID 1562, 172.30.4.7, executor 2, partition 18, PROCESS_LOCAL)
22/03/17 05:58:14 INFO TaskSetManager: Starting task 19.0 in stage 111.0 (TID 1563, 172.30.4.11, executor 0, partition 19, PROCESS_LOCAL)
22/03/17 05:58:14 INFO TaskSetManager: Starting task 20.0 in stage 111.0 (TID 1564, 172.30.4.8, executor 1, partition 20, PROCESS_LOCAL)
22/03/17 05:58:14 INFO TaskSetManager: Starting task 21.0 in stage 111.0 (TID 1565, 172.30.4.7, executor 2, partition 21, PROCESS_LOCAL)
22/03/17 05:58:14 INFO TaskSetManager: Starting task 22.0 in stage 111.0 (TID 1566, 172.30.4.11, executor 0, partition 22, PROCESS_LOCAL)
22/03/17 05:58:14 INFO TaskSetManager: Starting task 23.0 in stage 111.0 (TID 1567, 172.30.4.8, executor 1, partition 23, PROCESS_LOCAL)

```

Approach in Building a data application

• Easiest Approach

• Design the output table / schema first

- Word Count
- Log Analysis

Application Name:String Log Info:String LogDate:DateTime LogMessage:String

• Analyze Input data source – don't worry about format

- Row delimiter
 - Word Count - Each word is a record
 - Log Analysis - Each line is a record
- Column delimiter
 - How to split columns in each row ? (Map input record with defined output Schema)

• Data transformation & aggregation

- Word count -
 - Create a map(word,1) for every word. Sum up the value for each word
- Log Analysis
 - If no aggregation , takeout needed columns from the input line