Introduction to BigData, Hadoop, and HDFS

© Pristine – www.edupristine.com
Big Data

Definition:

- Big data is a collection of both structured and unstructured data that is too large, fast and distinct to be managed by traditional database management tools or traditional data processing applications.

For e.g.,

- Data managed by eBay for request search, consumer/customer recommendations, current trend and merchandising
- Data managed by Facebook, Twitter, Linkedin for providing social network platform
- Data managed for real-time auction/bidding in online Advertisement platforms
- Data captured in the Blackbox of flights etc...
Big Data

Characteristics of Big Data:

- **Volume**
  - System/Users generating Terabytes, PetaBytes and ZetaBytes of data

- **Velocity**
  - System generated streams of data
  - Multiple sources feeding data for one system

- **Variety**
  - Structured data
  - Unstructured data - Blogs, Images, Audio etc.

Storage

Processing

Presentation
Big Data

Value Chain of Big Data:

1. **Data Generation**
   - Source of data e.g., Users, Enterprises, Systems etc

2. **Data Collection**
   - Companies, Tools, Sites aggregating data e.g., IMS

3. **Data Analysis**
   - Research and Analytics Firms e.g., MuSigms etc

4. **Application of insights**
   - Management consulting firms, MNC’s
Big Data-Hadoop

Platform For Managing Big Data

Hadoop:

- Hadoop is an open-source software framework from Apache that supports scalable distributed applications.
- Hadoop supports running applications on large clusters of commodity hardware and provide fast and reliable analysis of both structured and unstructured data.
- Hadoop uses simple programming model.
- Hadoop can scale from single servers to thousands of machines, each offering local computation and storage

Hadoop Comprises of multiple softwares. Some examples are given below:

- Hadoop Distributed File System
- Hadoop MapReduce
- PIG
- HIVE
- ZooKeeper
Hadoop Distributed File System (HDFS)

- HDFS is the implementation of Hadoop filesystem, the Java abstract class org.apache.hadoop.fs.FileSystem that represents a filesystem in Hadoop.
- HDFS is designed to work efficiently in conjunction with MapReduce

Definition:
- A distributed file system that provides big data storage solution through high-throughput access to application data.
- When data can potentially outgrow the storage capacity of a single machine, partitioning it across a number of separate machines is necessary for storage or processing. This is achieved using a distributed File systems.

Potential challenges:
- Ensuring data integrity
- Data data retention in case of nodes failure
- Integration across multiple nodes and systems
Hadoop Distributed File System-HDFS

Hadoop Distributed File System (HDFS)

- HDFS is designed for storing very large files* with streaming data access* patterns, running on clusters of commodity hardware*.

Very large files :

- Very large means files that are hundreds of MB, GB, TB or PB in size.

Streaming data access:

- HDFS implements write-once, read-many-times pattern. Data is copied from source for analyses over time. Each analysis involves a large proportion of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

Commodity hardware

- Hadoop runs on clusters of commodity hardware (commonly available hardware). HDFS is designed to carry on working without a noticeable interruption to the user in the case of node failure.
Hadoop Distributed File System-HDFS

Where HDFS doesn’t work well:
- HDFS is not designed for following scenarios:

Low-latency data access:
- HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency

Lots of small files:
- Filesystem metadata is stored in memory, hence the limit to the number of files in a filesystem is governed by the amount of memory on the namenode.
- As a rule of thumb, each file, directory, and block takes about 150 bytes.

Multiple updates in the file
- Files in HDFS may be written to by a single writer at the end of the file. There is no support for multiple writers, or for modifications at arbitrary offsets in the file
Hadoop Distributed File System-HDFS

HDFS Concepts

- Blocks
- Namenodes
- Datanodes
- HDFS Federation
- HDFS High Availability
- Failover
- Fencing
Hadoop Distributed File System-HDFS

**HDFS Concepts - Blocks:**

- Files in HDFS are broken into blocks 64 MB (default) and stored as independent units.
- File in HDFS that is smaller than a single block does not occupy a full block’s storage.
- HDFS blocks are large compared to disk blocks to minimize the cost of seeks.
- Map tasks in MapReduce operate on one block at a time.
- Block as a unit of abstraction rather than a file simplifies the storage subsystem which tracks the meta data information.
- Blocks fit well with replication for providing fault tolerance and availability.
- HDFS’s `fsck` command understands blocks.

For example, command to list the blocks that make up each file in the file system:

- `% hadoop fsck / -files -blocks`
HDFS Concepts –Namenodes & Datanodes

- An HDFS cluster has two types of node operating in a master-workers pattern:
- A namenode (the master) and a number of datanodes (workers).
- The namenode manages the filesystem namespace. The filesystem cannot be used without the namenode. It maintains the filesystem tree and the metadata for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files as below:
  - Namespace image
  - Edit log
- The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts.
- A client accesses the filesystem on behalf of the user by communicating with the namenode and the datanodes
Hadoop Distributed File System-HDFS

HDFS Concepts –Namenodes & Datanodes Continued...

- Datanodes are the workhorses of the filesystem.
- Datanodes store and retrieve blocks when requested by clients or the namenode. They report back to the namenode periodically with lists of blocks that they are storing
- It is important to make the namenode resilient to failure as without the namenode, the filesystem cannot be used. Hadoop provides two mechanisms for this:
  1. Back up the files that make up the persistent state of the filesystem metadata
  2. Run a secondary namenode, which despite its name does not act as a namenode. Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large
Hadoop Distributed File System-HDFS

HDFS Concepts – Namenodes & Datanodes Continued...

1-Open

2-Get block location

3-Read

4-Read

5-Read

6-Close

Client

 DFS

FSDataInputStream

Datanode

Datanode

Datanode

Namenode
Hadoop Distributed File System - HDFS

HDFS Concepts – HDFS Federation

- HDFS Federation allows a cluster to scale by adding namenodes, each of which manages a portion of the filesystem namespace.

- For example, one namenode might manage all the files rooted under /user, and a second namenode might handle files under /share.

- Under federation, each namenode manages a namespace volume, which is made up of the metadata for the namespace, and a block pool containing all the blocks for the files in the namespace.

- Namespace volumes are independent of each other i.e Namenodes do not communicate with one another and Failure of one namenode does not affect the availability of the namespaces managed by other namenodes.

- Block pool storage is not partitioned, however, so datanodes register with each namenode in the cluster and store blocks from multiple block pools.

- To access a federated HDFS cluster, clients use client-side mount tables to map file paths to namenodes. This is managed in configuration using the ViewFileSystem, and viewfs:// URIs.
Hadoop Distributed File System-HDFS

HDFS Concepts – HDFS High Availability

- HDFS provides features to recover from a failed namenode by adding support for HDFS high-availability (HA).
- It provides a pair of namenodes in an active-standby by configuration to take over its duties to continue servicing client requests without a significant interruption.

This is achieved through following implementations:

- Using highly-available shared storage to share the edit log.
- Ensuring that datanodes send block reports to both namenodes
- Clients must be configured to handle namenode failover
HDFS Concepts – Failover and Fencing

- The transition from the active namenode to the standby is managed by the failover controller.
- Failover controllers are pluggable, but the first implementation uses ZooKeeper to ensure that only one namenode is active.
- Each namenode runs a lightweight failover controller process whose job is to monitor its namenode for failures (using a simple heartbeating mechanism) and trigger a failover should a namenode fail.
- Failover may also be initiated manually by an administrator, in the case of routine maintenance, also known as a graceful failover.
- In the case of an ungraceful failover, however, it is impossible to be sure that the failed namenode has stopped running. For example, a slow network or a network partition can trigger a failover transition, even though the previously active namenode is still running, and thinks it is still the active namenode.
- The HA implementation ensures that the previously active namenode is prevented from doing any damage and causing corruption—a method known as fencing. The system employs
  - a range of fencing mechanisms, including killing the namenode’s process, revoking its access to the shared storage directory and disabling its network port via a remote management command.
Hadoop Distributed File System-HDFS

HDFS File Permissions

- HDFS has a permissions model for files and directories.

There are three types of permission:

- Read permission (r)
- Write permission (w)
- Execute permission (x)

- The read permission is required to read files or list the contents of a directory.
- The write permission is required to write a file, or for a directory, to create or delete files or directories in it.
- The execute permission is ignored for a file since you can’t execute a file on HDFS, and for a directory it is required to access its children.
- Each file and directory has an owner, a group, and a mode.
- The mode is made up of the permissions for the user who is the owner, the permissions for the users who are members of the group, and the permissions for users who are neither the owners nor members of the group.
Hadoop Installation Steps

Step 1: Download and install VMware from the following link

VMware player: https://www.vmware.com/tryvmware/?p=player

Step 2: Download Hadoop Setup file (Cloudera CDH3-Cloudera VM file) from following link:

https://docs.google.com/file/d/0B_P02gj6T2mTGZKS3JzUTM3bjA/edit?usp=sharing

Extract the file in local drive

Step 3: Open VMware and access the “Open Virtual Machine” link. Browse and open the path where Cloudera setup file is saved.

Step 4: Use following login credentials:

Username – cloudera
Password - cloudera
Hadoop HDFS and MR sample problem
1. HDFS is the file system
2. MR is the job which runs on file system
3. The MR job helps the user to ask question from HDFS files
4. Pig and Hive are two projects built to replace coding the map reduce
5. Pig and Hive interpreter turns the script and sql queries "INTO" MR job
6. To save the map and reduce only dependency to be able to query on HDFS - Impala and Hive
7. Impala Optimized for high latency queries-Near real time
8. Hive optimized for batch processing job
9. Sqoop: Can put data from a relation DB to Hadoop ecosystem
10. Flume can send data generated from external system to move to HDFS- Apt for high volume logging
11. Hue: Graphical frontend to cluster
12. Oozie: Workflow management tool
When a 150 mb file is being fed to Hadoop ecosystem it breaks itself into multiple parts to achieve parallelism.

It breaks itself into chunks where default chunk size is 64 mb.

Data node is the demon which takes care of all the happening at an individual node.

Name node is the one which keeps a track on what goes where and when required how to collect the same group together.

Now think hard what could be the possible challenges?
HDFS

- When a 150 mb file is being fed to Hadoop ecosystem it breaks itself in multiple parts to achieve parallelism.
- It breaks itself in to chunks where default chunk size is 64 mb.
- Data node is the demon which takes care of all the happening at an individual node.
- Name node is the one which keeps a track on what goes where and when required how to collect the same group together.
- Now think hard what could be the possible challenges?

Is There a Problem?

- Network Failure
- Disk Failure on DN
- Not all DN used
- Block sizes differ
- Disk Failure on NN
HDFS

Problems:
1. If one of the node fails, then the data stored at that node goes missing
2. Network Failure
3. Single Point of Failure: Name Node – Heart of Hadoop ecosystem

Solution:
1. Hadoop solves this problem by replicating every data fragment thrice and save it on different nodes, so even if one node fails, it can still recover
2. Network Failure is an important issue as lot of shuffle happens
3. Name node failure was at first reduced by storing the name node data on NFS(Network file Server) which could be recovered in case of Name node crash
HDFS

- Later one same problem was solved finally by introducing the concept of secondary node. Though Data redundancy seemed to be an issue with this approach, still was best way to solve the network/Hardware irregularity.
Application of HDFS - Reading data file to be analyzed

```
[training@localhost data]$ ls
access_log.gz  purchases.txt
[training@localhost data]$ hadoop fs -ls
[training@localhost data]$ hadoop fs -put purchases.txt
[training@localhost data]$ hadoop fs -ls
```

- `log of sell across store`  
- `So far nothing in Cluster!!!`  
- `Lets push that file to hadoop cluster`  
- `Pushed!!!`  

```
31 17:59 Norfok Toys 164.34 MasterCard
2012-12-31 17:59 Chula Vista Music 380.67 Visa
2012-12-31 17:59 Hialeah Toys 115.21 MasterCard
2012-12-31 17:59 Indianapolis Men's Clothing 158.28 MasterCard
2012-12-31 17:59 Norfolk Garden 414.09 MasterCard
2012-12-31 17:59 Baltimore DVDs 467.3 Visa
2012-12-31 17:59 Santa Ana Video Games 144.73 Visa
2012-12-31 17:59 Gilbert Consumer Electronics 354.66 Discover
2012-12-31 17:59 Memphis Sporting Goods 124.79 Amex
2012-12-31 17:59 Chicago Men's Clothing 386.54 MasterCard
2012-12-31 17:59 Birmingham CDs 118.04 Cash
2012-12-31 17:59 Las Vegas Health and Beauty 420.46 Amex
2012-12-31 17:59 Wichita Toys 383.9 Cash
2012-12-31 17:59 Tucson Pet Supplies 268.39 MasterCard
2012-12-31 17:59 Glendale Women's Clothing 68.05 Amex
2012-12-31 17:59 Albuquerque Toys 345.7 MasterCard
```

- `How my data looks like!!!`
Application of HDFS-Moving data file for analysis

Moving a file in hadoop

Moved file in Hadoop ecosystem
HDFS – Application and analysis using MR

- Lets Solve a problem using concepts we just learned
- Say we want to sum up all the sales across store/location. How would you approach this problem? Excel? Hash Table? What could be possible limitation if the data size is in TB?

How can we perform analytics on huge sales data?
For a moment assume the problem in hand is you have a pile of postcards where on every postcard a city name is written and some one want to know how many postcard per city?

Easy right ?? Just have many buckets with a city name on each and keep putting as and when you encounter?

What if 3 diff people are doing the same work ? Well may be then you need to have a set of bucket for each one of them to avoid too much co ordination

Same is the concept of mapper and reducer.

Now for the problem in Hand
MR- Workflow

To summarize what we tried...

MAP REDUCE

MAPPERS

SHUFFLE & SORT

INTERMEDIATE RECORDS (KEY, VALUES)

REDUCERS

KEY, VALUES

RESULTS
Run the Mapper and Reducer code

```
drwxr-xr-x - training supergroup 0 2013-09-12 21:16 myinput
[training@localhost code]$ hadoop fs -ls myinput
Found 1 items
-rw-r--r-- 1 training supergroup 211312924 2013-09-12 21:16 myinput/purchases.txt
[training@localhost code]$ ls
mapper.py reducer.py
[training@localhost code]$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.6.0-mr1-cdh4.1.1.jar -mapper mapper.py -reducer reducer.py -file mapper.py -file reducer.py -input myinput -output joboutput
packageJobJar: [mapper.py, reducer.py, /tmp/hadoop-training/hadoop-unjar815901048942447581/] [] /tmp/stream ob2183126376193420464.jar tmpDir=null
13/09/12 21:23:25 WARN snappy.LoadSnappy: Snappy native library is available
13/09/12 21:23:25 INFO streaming.StreamJob: To kill this job, run:
13/09/12 21:23:25 INFO streaming/streaming.Job: UNDEF/bin/hadoop job -Dmapred.job.tracker=0.0.0.0:8021 -kill j
13/09/12 21:23:25 INFO streaming/streaming.Job: Tracking URL: http://0.0.0.0:50030/jobdetails.jsp?jobid=job_201309111631_0001
13/09/12 21:23:26 INFO streaming/streaming.Job: map 0% reduce 0%
13/09/12 21:23:36 INFO streaming/streaming.Job: map 16% reduce 0%
13/09/12 21:23:39 INFO streaming/streaming.Job: map 24% reduce 0%
13/09/12 21:23:42 INFO streaming/streaming.Job: map 33% reduce 0%
```
Part-0000 is the file which has the output we expect:
## MR-Output

### Part-0000 OUTPUT FILE

<table>
<thead>
<tr>
<th>Location</th>
<th>Zip Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portland</td>
<td>10007635.77</td>
</tr>
<tr>
<td>Raleigh</td>
<td>10061442.54</td>
</tr>
<tr>
<td>Reno</td>
<td>10079955.16</td>
</tr>
<tr>
<td>Richmond</td>
<td>9992941.59</td>
</tr>
<tr>
<td>Riverside</td>
<td>10006695.42</td>
</tr>
<tr>
<td>Rochester</td>
<td>10067606.92</td>
</tr>
<tr>
<td>Sacramento</td>
<td>10123468.18</td>
</tr>
<tr>
<td>Saint Paul</td>
<td>10057233.57</td>
</tr>
<tr>
<td>San Antonio</td>
<td>10014441.7</td>
</tr>
<tr>
<td>San Bernardino</td>
<td>9965152.04</td>
</tr>
<tr>
<td>San Diego</td>
<td>9966038.39</td>
</tr>
<tr>
<td>San Francisco</td>
<td>9995570.54</td>
</tr>
<tr>
<td>San Jose</td>
<td>9936721.41</td>
</tr>
<tr>
<td>Santa Ana</td>
<td>10050309.93</td>
</tr>
<tr>
<td>Scottsdale</td>
<td>10037921.85</td>
</tr>
<tr>
<td>Seattle</td>
<td>9936267.37</td>
</tr>
<tr>
<td>Spokane</td>
<td>10083362.98</td>
</tr>
<tr>
<td>St. Louis</td>
<td>10002105.14</td>
</tr>
<tr>
<td>St. Petersburg</td>
<td>9986495.54</td>
</tr>
<tr>
<td>Stockton</td>
<td>10006412.64</td>
</tr>
<tr>
<td>Tampa</td>
<td>10106428.55</td>
</tr>
<tr>
<td>Toledo</td>
<td>10020768.88</td>
</tr>
<tr>
<td>Tucson</td>
<td>9998252.47</td>
</tr>
<tr>
<td>Tulsa</td>
<td>10064955.9</td>
</tr>
<tr>
<td>Virginia Beach</td>
<td>10086553.5</td>
</tr>
<tr>
<td>Washington</td>
<td>10139363.39</td>
</tr>
<tr>
<td>Wichita</td>
<td>10083643.21</td>
</tr>
</tbody>
</table>
def mapper():
    for line in sys.stdin:
        data = line.strip().split("\t")

        if len(data) == 6:
            date, time, store, item, cost, payment = data

            print "{0}\t{1}".format(store, cost)
Reducer Code

```python
def reducer():
    salesTotal = 0
    oldKey = None
    for line in sys.stdin:
        data = line.strip().split("\t")
        if len(data) != 2:
            continue
        thisKey, thisSale = data
        if oldKey and oldKey != thisKey:
            print "{0}\t{1}".format(oldKey, salesTotal)
            salesTotal = 0
        oldKey = thisKey
        salesTotal += float(thisSale)

        Miami  12.34
        Miami  99.07
        Miami  3.14
        NYC    99.77
        NYC    88.99
```
Hadoop Ecosystem

Another most common problem solved using Hadoop cluster is log processing.

Sample Example copied below: Solve for finding the number of index page hits?
Hadoop Ecosystem

1. Demo Video will show how you could look at the way nodes are involved when the mapper reducer job is running

2. Hs is an alias created for command Hadoop –jar /path/to/jar –mapper mapper.py –reducer reducer.py –file mapper.py
Thank you!

Contact:

EduPristine
702, Raaj Chambers, Old Nagardas Road, Andheri (E), Mumbai-400 069. INDIA
www.edupristine.com
Ph. +91 22 3215 6191