

ASSIGNMENT 11.1

Task1

Explain the below concepts with an example in brief.

Nosql Databases: Nosql database refers to Not Only SQL databases. It is an approach to databases that represents a shift away from traditional database management systems (RDMS).

Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models.

As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS.

Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images.

Types of Nosql Databases:

- There are 4 basic types of NoSQL databases:
- Key-Value Store – It has a Big Hash Table of keys & values {Example- Riak, Amazon S3 (Dynamo)}
- Document-based Store- It stores documents made up of tagged elements. {Example- MongoDB, CouchDB}
- Column-based Store- Each storage block contains data from only one column, {Example- HBase, Cassandra}
- Graph-based- A network database that uses edges and nodes to represent and store data. {Example- Neo4J}
- **Key-value data stores:** It emphasizes simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned *replicated* across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data.

- **Document stores:** typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure.
- **Wide-column stores:** Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.
- **Graph stores:** A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index

CAP Theorem:--

The **CAP** theorem states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees.

Consistency: Every read receives the most recent write or an error.

Availability: Every request receives a (non-error) response – without guarantee that it contains the most recent write.

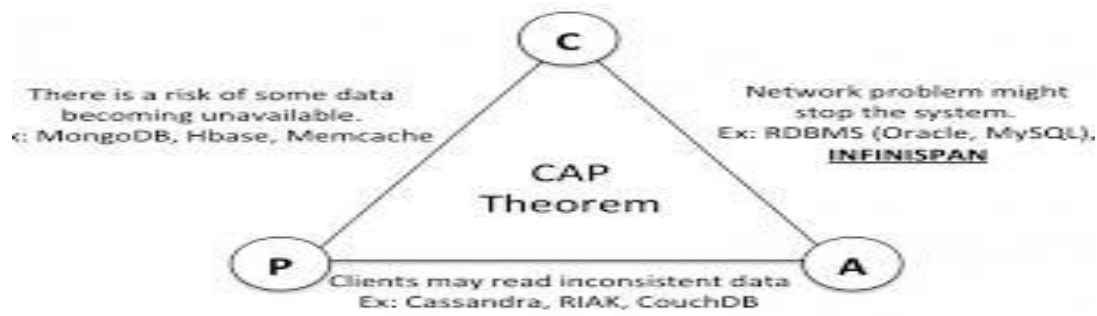
Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

Explanation: No distributed system is safe from network failures, thus network partitioning generally has to be tolerated. In the presence of a partition, one is then left with two options: consistency or availability.

When choosing consistency over availability, the system will return an error or a time-out if particular information cannot be guaranteed to be up to date due to network partitioning.

When choosing availability over consistency, the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up to date due to network partitioning.

In the absence of network failure – that is, when the distributed system is running normally – both availability and consistency can be satisfied.



CAP Theorem

HBase Architecture:---

HBase provides low-latency random reads and writes on top of HDFS. In HBase, tables are dynamically distributed by the system whenever they become too large to handle. The simplest and foundational unit of horizontal scalability in HBase is a Region. A continuous, sorted set of rows that are stored together is referred to as a region (subset of table data).

HBase architecture has a single HBase master node (HMaster) and several slaves i.e. region servers. Each region server (slave) serves a set of regions, and a region can be served only by a single region server.

Whenever a client sends a write request, HMaster receives the request and forwards it to the corresponding region server.

HBase can be run in a multiple master setup, wherein there is only single active master at a time.

HBase tables are partitioned into multiple regions with every region storing multiple table's rows.

Components of Apache HBase Architecture

HBase architecture has 3 important components- HMaster, Region Server and ZooKeeper.

i. HMaster

HBase HMaster is a lightweight process that assigns regions to region servers in the Hadoop cluster for load balancing. Responsibilities of HMaster –

- Manages and Monitors the Hadoop Cluster
- Performs Administration (Interface for creating, updating and deleting tables.)
- Controlling the failover
- DDL operations are handled by the HMaster
- Whenever a client wants to change the schema and change any of the metadata operations, HMaster is responsible for all these operations.

ii. Region Server

These are the worker nodes which handle read, write, update, and delete requests from clients. Region Server process, runs on every node in the hadoop cluster. Region Server runs on HDFS DataNode and consists of the following components –

- Block Cache – This is the read cache. Most frequently read data is stored in the read cache and whenever the block cache is full, recently used data is evicted.
- MemStore- This is the write cache and stores new data that is not yet written to the disk. Every column family in a region has a MemStore.
- Write Ahead Log (WAL) is a file that stores new data that is not persisted to permanent storage.
- HFile is the actual storage file that stores the rows as sorted key values on a disk.

iii. Zookeeper

HBase uses ZooKeeper as a distributed coordination service for region assignments and to recover any region server crashes by loading them onto other region servers that are functioning.

ZooKeeper is a centralized monitoring server that maintains configuration information and provides distributed synchronization.

Whenever a client wants to communicate with regions, they have to approach Zookeeper first. HMaster and Region servers are registered with ZooKeeper service, client needs to access ZooKeeper quorum in order to connect with region servers and HMaster.

ZooKeeper service keeps track of all the region servers that are there in an HBase cluster-tracking information about how many region servers are there and which region servers are holding which DataNode. HMaster contacts ZooKeeper to get the details of region servers. Various services that Zookeeper provides include –

- Establishing client communication with region servers.
- Tracking server failure and network partitions.
- Maintain Configuration Information
- Provides ephemeral nodes, which represent different region servers.

HBase vs RDBMS:--

H Base

1. Column-oriented
2. Flexible schema, add columns on the Fly
3. Good with sparse tables.
4. No query language
5. Wide tables
6. Joins using MR – not optimized
7. Tight – Integration with MR
8. De-normalize your data.
9. Horizontal scalable.
10. Consistent
11. No transactions.
12. Good for semi-structured data as well as structured data.

RDBMS

1. Row-oriented(mostly)
2. Fixed schema
3. Not optimized for sparse tables.
4. SQL
5. Narrow tables
6. optimized for Joins(small, fast ones)
7. Not really
8. Normalize as you can
9. Hard to share and scale.
10. Consistent
11. transactional
12. Good for structured data.

Task2


Execute the acadgild blog - ImportTSV Data from HDFS into HBase.

Step1: Inside Hbase shell give the following command to create table along with 2 column family.

create 'bulktable', 'cf1', 'cf2'

```
acadmild@localhost:~  
hbase(main):001:0>  
hbase(main):001:0> create 'bulktable', 'cf1','cf2'  
0 row(s) in 1.6440 seconds  
  
=> Hbase::Table - bulktable
```

mkdir -p hbase and change the directory by **cd hbase**

 acadgild@localhost:~/hbase

```
[acadgild@localhost ~]$ mkdir -p hbase
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$ cd hbase
[acadgild@localhost hbase]$ pwd
/home/acadgild/hbase
[acadgild@localhost hbase]$ █
```

Step3: Create a file inside the HBase directory named bulk_data.tsv with tab separated data inside it using below command in terminal.

cat > bulk_data.tsv

cat bulk_data.tsv

 acadgild@localhost:~/hbase

```
[acadgild@localhost hbase]$ cat > bulk_data.tsv
1      Amit      4
2      Giriya    3
3      Jatin     5
4      Swati     3
^C
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost hbase]$ cat bulk_data.tsv
1      Amit      4
2      Giriya    3
3      Jatin     5
4      Swati     3
[acadgild@localhost hbase]$ █
```

hadoop fs -mkdir /hbase

hadoop fs -put bulk_data.tsv /hbase/

hadoop fs -cat /hbase/bulk_data.tsv

acadgild@localhost:~/hbase

```
[acadgild@localhost hbase]$ hdfs dfs -ls /hbase
18/02/24 13:38:28 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
Found 9 items
drwxr-xr-x - acadgild supergroup 0 2018-02-23 21:30 /hbase/.tmp
drwxr-xr-x - acadgild supergroup 0 2018-02-24 13:37 /hbase/MasterProcWALs
drwxr-xr-x - acadgild supergroup 0 2018-02-23 21:27 /hbase/WALs
drwxr-xr-x - acadgild supergroup 0 2018-02-24 12:18 /hbase/archive
drwxr-xr-x - acadgild supergroup 0 2018-02-23 16:10 /hbase/corrupt
drwxr-xr-x - acadgild supergroup 0 2018-02-23 16:00 /hbase/data
-rw-r--r-- 1 acadgild supergroup 42 2018-02-23 15:59 /hbase/hbase.id
-rw-r--r-- 1 acadgild supergroup 7 2018-02-23 15:59 /hbase/hbase.version
drwxr-xr-x - acadgild supergroup 0 2018-02-24 13:38 /hbase/oldWALs
[acadgild@localhost hbase]$ hadoop fs -put bulk_data.tsv /hbase/
18/02/24 13:39:05 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
[acadgild@localhost hbase]$ hadoop fs -cat /hbase/bulk_data.tsv
18/02/24 13:39:43 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
1      Amit      4
2      Girija    3
3      Jatin     5
4      Swati     3
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost hbase]$
```

Step5: Now the data is present in HDFS. In terminal, we give the following command along with arguments <tablename> and <path of data in HDFS>

Command:

**hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
-Dimporttsv.columns=HBASE_ROW_KEY, cf1:name, cf2:exp bulktable
/hbase/bulk_data.tsv**

```
[acadgild@localhost ~]$
[acadgild@localhost ~]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hbase/bulk_data.tsv
2018-03-12 05:46:25,199 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.2.6/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2018-03-12 05:46:26,066 INFO [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x6025e1b6 connecting to ZooKeeper ensemble=localhost:2181
2018-03-12 05:46:26,086 INFO [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014 09:09 GMT
2018-03-12 05:46:26,086 INFO [main] zookeeper.ZooKeeper: Client environment:host.name=localhost
2018-03-12 05:46:26,086 INFO [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_151
2018-03-12 05:46:26,086 INFO [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
2018-03-12 05:46:26,086 INFO [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/java/jdk1.8.0_151/jre
2018-03-12 05:46:26,087 INFO [main] zookeeper.ZooKeeper: Client environment:java.class.path=/home/acadgild/install/hbase/hbase-1.2.6/conf:/usr/java/jdk1.8.0_151/lib/
ols.jar:/home/acadgild/install/hbase/hbase-1.2.6:/home/acadgild/install/hbase/hbase-1.2.6/lib/activation-1.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/aopallia
e-1.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/apacheds-i18n-2.0.0-M15.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/apacheds-kerberos-codec-2.0.0-M15.jar:
ome/acadgild/install/hbase/hbase-1.2.6/lib/api-asn1-api-1.0.0-M20.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/api-util-1.0.0-M20.jar:/home/acadgild/install/hbase/hb
ase-1.2.6/lib/asm-3.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/avro-1.7.4.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-beanutils-1.7.0.jar:/home/a
cadgild/install/hbase/hbase-1.2.6/lib/commons-beanutils-core-1.8.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-cli-1.2.jar:/home/acadgild/install/hbase/hb
e-1.2.6/lib/commons-codec-1.9.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-collections-3.2.2.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-comp
ss-1.4.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-configuration-1.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-daemon-1.0.13.jar:/home/a
cadgild/install/hbase/hbase-1.2.6/lib/commons-digester-1.8.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-el-1.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/
b/commons-httpclient-3.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-io-2.4.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-lang-2.6.jar:/home/a
cadgild/install/hbase/hbase-1.2.6/lib/commons-logging-1.2.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-math-2.2.jar:/home/acadgild/install/hbase/hbase-1.2.6
lib/commons-math3-3.1.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-net-3.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/disruptor-3.3.0.jar:/home/aca
ild/install/hbase/hbase-1.2.6/lib/findbugs-annotations-1.3.9-1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/guava-12.0.1.jar:/home/acadgild/install/hbase/hbase-1.
6/lib/guice-3.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/guice-servlet-3.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-annotations-2.5.1.jar:/home
acadgild/install/hbase/hbase-1.2.6/lib/hadoop-auth-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-client-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.
6/lib/hadoop-common-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-hdfs-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-mapreduce-client-
```

```

p-2.6.5/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.6.5.jar:/home/acadgild/install/hadoop/hadoop-2.6.5/contrib/capacity-scheduler
2018-03-12 05:46:26,090 INFO [main] zookeeper.ZooKeeper: Client environment:java.library.path=/usr/java/packages/lib/amd64:/usr/lib64:/lib
2018-03-12 05:46:26,090 INFO [main] zookeeper.ZooKeeper: Client environment:java.io.tmpdir=/tmp
2018-03-12 05:46:26,090 INFO [main] zookeeper.ZooKeeper: Client environment:java.compiler=<NA>
2018-03-12 05:46:26,090 INFO [main] zookeeper.ZooKeeper: Client environment:os.name=Linux
2018-03-12 05:46:26,091 INFO [main] zookeeper.ZooKeeper: Client environment:os.arch=amd64
2018-03-12 05:46:26,091 INFO [main] zookeeper.ZooKeeper: Client environment:os.version=2.6.32-696.18.7.el6.x86_64
2018-03-12 05:46:26,091 INFO [main] zookeeper.ZooKeeper: Client environment:user.name=acadgild
2018-03-12 05:46:26,091 INFO [main] zookeeper.ZooKeeper: Client environment:user.home=/home/acadgild
2018-03-12 05:46:26,091 INFO [main] zookeeper.ZooKeeper: Client environment:user.dir=/home/acadgild
2018-03-12 05:46:26,092 INFO [main] zookeeper.ZooKeeper: Initiating client connection, connectString=localhost:2181 sessionTimeout=90000 w
0x0, quorum=localhost:2181, baseZNode=/hbase
2018-03-12 05:46:26,131 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Opening socket connection to server localhost/0:0:0:0
pt to authenticate using SASL (unknown error)
2018-03-12 05:46:26,147 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Socket connection established to localhost/0:0:0:0:
n
2018-03-12 05:46:26,159 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Session establishment complete on server localhost/0:0:
= 0x16217907d5f0006, negotiated timeout = 90000
2018-03-12 05:46:31,790 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2018-03-12 05:46:31,891 INFO [main] client.ConnectionManager$HConnectionImplementation: Closing zookeeper sessionId=0x16217907d5f0006
2018-03-12 05:46:31,897 INFO [main-EventThread] zookeeper.ClientCnxn: EventThread shut down
2018-03-12 05:46:31,898 INFO [main] zookeeper.ZooKeeper: Session: 0x16217907d5f0006 closed
2018-03-12 05:46:32,024 INFO [main] client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
2018-03-12 05:46:32,458 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2018-03-12 05:46:35,757 INFO [main] input.FileInputFormat: Total input paths to process : 1
2018-03-12 05:46:35,926 INFO [main] mapreduce.JobSubmitter: number of splits:1
2018-03-12 05:46:35,970 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2018-03-12 05:46:36,161 INFO [main] mapreduce.JobSubmitter: Submitting tokens for job: job_1520813740819_0001
2018-03-12 05:46:36,724 INFO [main] impl.YarnClientImpl: Submitted application application_1520813740819_0001
2018-03-12 05:46:36,817 INFO [main] mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1520813740819_0001/
2018-03-12 05:46:36,818 INFO [main] mapreduce.Job: Running job: job_1520813740819_0001
2018-03-12 05:46:49,125 INFO [main] mapreduce.Job: Job job_1520813740819_0001 running in uber mode : false
2018-03-12 05:46:49,129 INFO [main] mapreduce.Job: map 0% reduce 0%
2018-03-12 05:46:56,316 INFO [main] mapreduce.Job: map 100% reduce 0%
2018-03-12 05:46:56,350 INFO [main] mapreduce.Job: Job job_1520813740819_0001 completed successfully
2018-03-12 05:46:56,546 INFO [main] mapreduce.Job: Counters: 31
File System Counters
FILE: Number of bytes read=0

```

```

2018-03-12 05:46:56,546 INFO [main] mapreduce.Job: Counters: 31
File System Counters
FILE: Number of bytes read=0
FILE: Number of bytes written=139463
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=146
HDFS: Number of bytes written=0
HDFS: Number of read operations=2
HDFS: Number of large read operations=0
HDFS: Number of write operations=0
Job Counters
Launched map tasks=1
Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=4604
Total time spent by all reduces in occupied slots (ms)=0
Total time spent by all map tasks (ms)=4604
Total vcore-seconds taken by all map tasks=4604
Total megabyte-seconds taken by all map tasks=4714496
Map-Reduce Framework
Map input records=4
Map output records=4
Input split bytes=106
Spilled Records=0
Failed Shuffles=0
Merged Map outputs=0
GC time elapsed (ms)=109
CPU time spent (ms)=1770
Physical memory (bytes) snapshot=169508864
Virtual memory (bytes) snapshot=2092298240
Total committed heap usage (bytes)=108527616
ImportTsv
Bad Lines=0
File Input Format Counters
Bytes Read=40
File Output Format Counters
Bytes Written=0

```


Scan 'bulkdata'

```
hbase(main):001:0> list
TABLE
bulktable
clicks
employee
htest
test
transactions_hbase
6 row(s) in 0.3250 seconds

=> ["bulktable", "clicks", "employee", "htest", "test", "transactions_hbase"]
hbase(main):002:0> scan "bulktable"
ROW                                COLUMN+CELL
1                                  column=cf1:name, timestamp=1520813785121, value=Amit
1                                  column=cf2:exp, timestamp=1520813785121, value=4
2                                  column=cf1:name, timestamp=1520813785121, value=Girija
2                                  column=cf2:exp, timestamp=1520813785121, value=3
3                                  column=cf1:name, timestamp=1520813785121, value=Jatin
3                                  column=cf2:exp, timestamp=1520813785121, value=5
4                                  column=cf1:name, timestamp=1520813785121, value=Swati
4                                  column=cf2:exp, timestamp=1520813785121, value=3
4 row(s) in 0.1730 seconds

hbase(main):003:0> █
```