HBase / Pig Integration

Bernard Lee Kok Bang

HBase / Pig Integration

- ***Move big data already in HDFS into HBase using Pig***
- Must create HBase table ahead of time
- ***New relation*** must have a ***unique key*** as its first column, followed by subsequent columns
- Both ***new relation*** and ***HBase table*** must have exact column
- USING clause allows us to STORE data into an HBase table
- Can work at scale – HBase is transactional on rows

1. Upload data into Ambari [HDFS]

- Login to Ambari
- Upload *u.user* file from movielens dataset into HDFS
- */user/maria_dev/ml-100k/* → **please change this to your own directory**

```
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
6|42|M|executive|98101
7|57|M|administrator|91344
8|36|M|administrator|05201
9|29|M|student|01002
10|53|M|lawyer|90703
```

u.user → pipe "|" delimited

Userid
Age
Gender
Occupation
Zipcode

In puTTy → Create HBase table

```
[ ]  # start the HBase interactive shell
     hbase shell

     # get the lists of existing tables on HBase instance
     list

     # create new table
     create 'users','userinfo'

     # run another list command to check for our newly created table
     list

     # exit out from the HBase shell
     exit
```

Create new table called "users" with column family name called "userinfo"

In command prompt [puTTY]

1. Make sure you are in your own directory

2. create a new pig script -> vi hbase.pig

3. Type these following code in the hbase.pig script

- *Remember to change the directory accordingly*

```
users = LOAD '/user/maria_dev/ml-100k/u.user'
USING PigStorage('|')
AS (userID:int, age:int, gender:chararray, occupation:chararray, zip:int);

STORE users INTO 'hbase://users'
USING org.apache.pig.backend.hadoop.hbase.HBaseStorage (
'userinfo:age,userinfo:gender,userinfo:occupation,userinfo:zip');
```

unique key

column family

HBase table

clause/code use to transmit data into HBase

Run the pig script using the following command

```
[ ]  pig hbase.pig
```

Return to HBase shell

```
hbase shell

# check whether our "users" table still exists
list

# view what's inside the "users" table
scan 'users'

# Once we are done, we can clean up the mess
disable 'users'
drop 'users'

# now the 'users' table shall no longer exists
list

# exit
exit
```

```
99                  column=userinfo:age, timestamp=1483035007446, value=20
99                  column=userinfo:gender, timestamp=1483035007446, value=M
99                  column=userinfo:occupation, timestamp=1483035007446, value
                    =student
99                  column=userinfo:zip, timestamp=1483035007446, value=63129
```

timestamp is automatically generated and appended; “versioning”

Cassandra Overview

A distributed non-relational database with no single point of failure;
no master node;
engineered for availability

Bernard Lee Kok Bang

Cassandra – NoSQL with a twist

- Unlike HBase, there is no master node at all – every node runs exactly the same software and performs the same functions
- Data model is similar to HBase
- It’s non-relational, but has a limited CQL query language as its interface

Cassandra's Design Choices

- The CAP Theorem says we can only have 2 out of 3: consistency, availability, partition-tolerance

  - And partition-tolerance is a requirement with 'big data', so we really only get to choose between consistency and availability

- Cassandra favors availability over consistency

  - It is "eventually consistent"

  - But we can specify our consistency requirements as part of our requests. So really it is "tunable consistency"

take FaceBook post as an example (consistency issue):
"Is it really the end of the world if everyone doesn't see my new post immediately?"

Weak vs Strong Consistency

- Imagine we have a library with multiple copies of the same book spread across different shelves.
- ***Strong consistency***: We can go to the librarian and request the book. The librarian will make sure that ***all the copies of the book*** in the library are ***the same*** before giving it to you. This ensures that we always get the most up-to-date version of the book, but it ***might take some time*** for the librarian to verify and retrieve it.

- ***Weak consistency***: Alternatively, we can go to any shelf, pick up the first copy of the book we find, and borrow it. In this case, we ***may not get the latest version*** of the book if there have been recent updates or changes made to it. However, we ***can get the book quickly*** because we don't have to wait for the librarian to check all the copies for consistency.

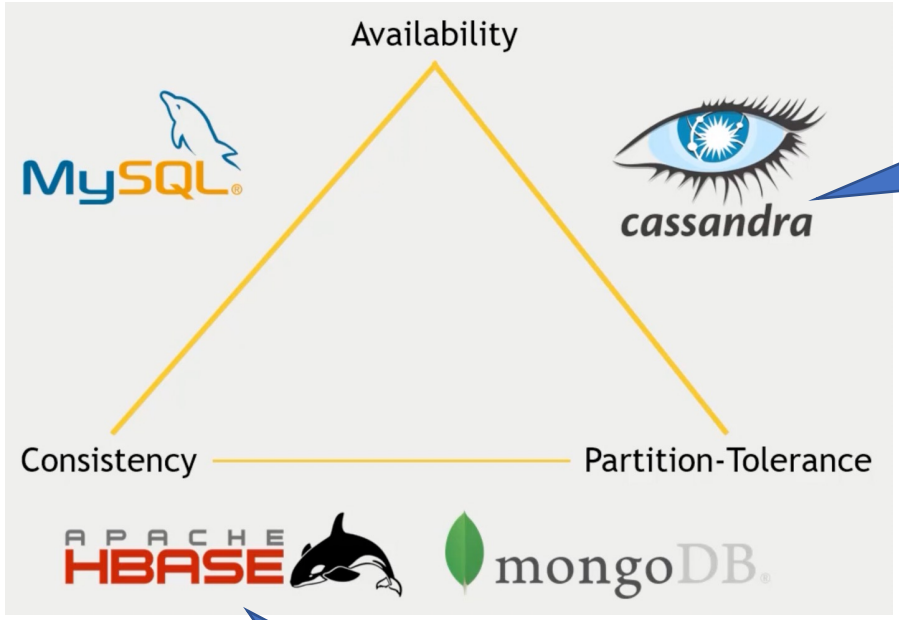
Cassandra -Weak vs Strong Consistency

Consistency level	Description	Characteristics
Strong consistency	High level of consistency in Cassandra	All data replicas are synchronized and up to date
	Ensures data synchronization before response	May introduce latency
Weak consistency	Lower level of consistency in Cassandra	Faster response time
	Possibility of reading slightly outdated or conflicting data	

CAP Theorem

- ***Consistency*** – when we write *"something"* to the database, we are going to *get that "something" back right away* no matter what happen
  - "completely consistent database" vs "eventually consistency"
  - *"If we make a new post to our friends and family on Facebook, is it really the end of the world if everyone doesn't see that post immediately"?*
  - Cassandra trade off consistency for availability [eventual consistency]
- ***Availability*** – a database that is reliable and always up and running, and have lots of redundancies are going to be more available
- ***Partition-tolerance*** – a database that can be easily split up and distributed across clusters *[non-negotiable for dealing with big data like HDFS]*

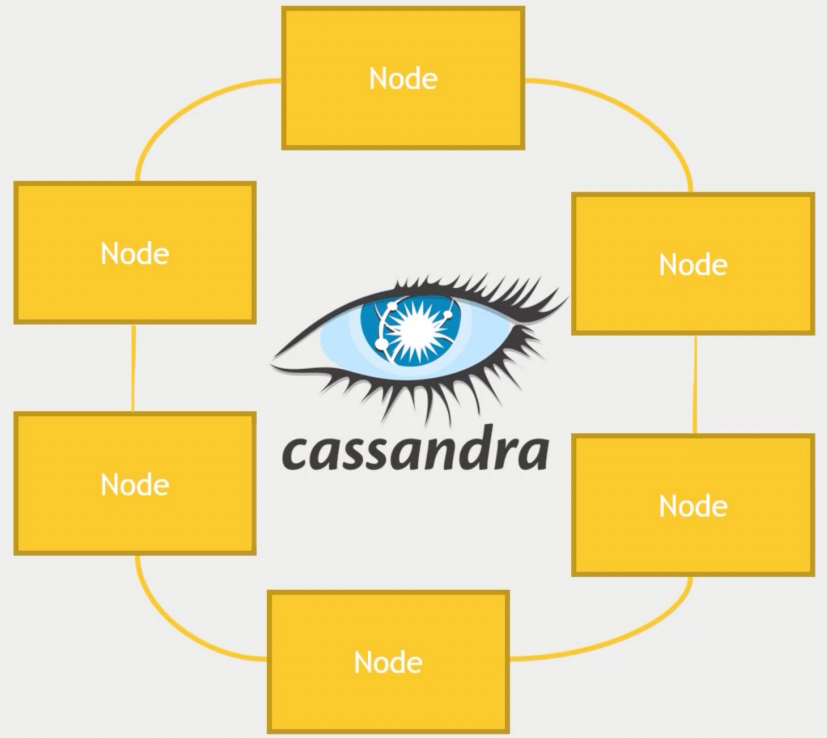
Where Cassandra fits in CAP tradeoffs



Cassandra values availability and partition-tolerance over consistency

HBase values consistency and partition-tolerance over availability; once the HBase master node or ZooKeeper goes down, we are done

Cassandra ring architecture → High availability



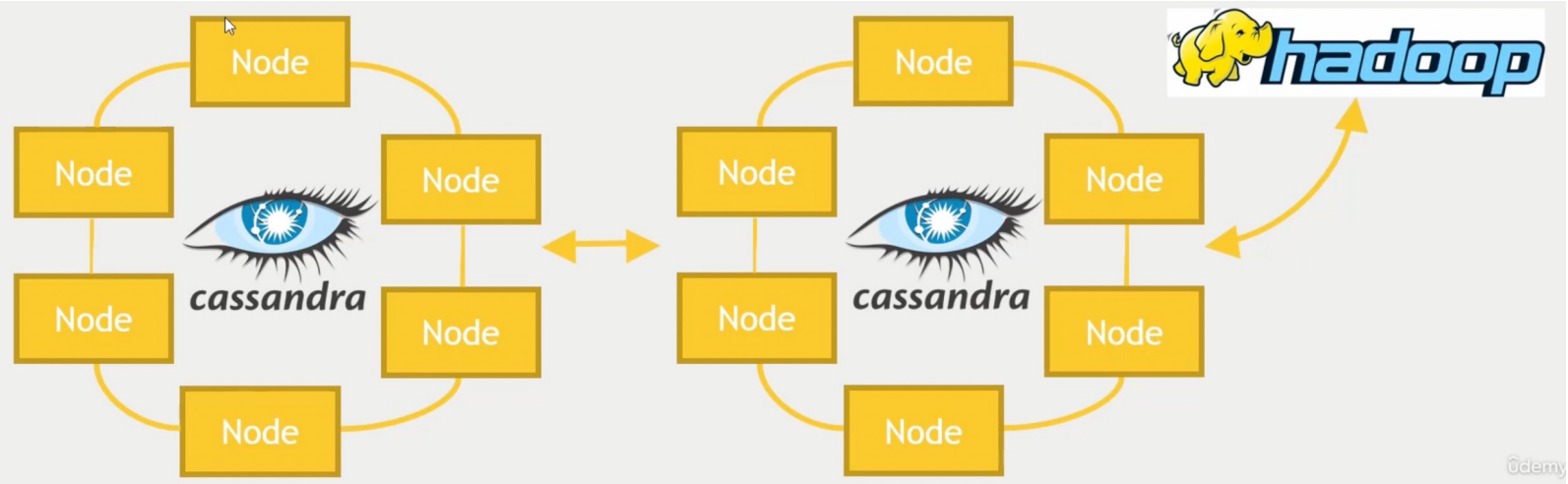
- Unlike HBase, Cassandra doesn't have any master node to keep track of what nodes serve what data
- Use gossip protocol – every node is communicating with each other every second to keep track of who's responsible for what bits of data, where the data is replicated to, etc…
- Every node in Cassandra cluster is running exactly the same software and doing exactly the same thing and performing exactly the same functions

- **doesn't mean all node have the same data**
- **the ring structure represents the distribution of data**
- **using the concept of primary keys**

Cassandra and HDFS cluster

- Cassandra's great for fast access to rows of information
- Get the best of both worlds – replicate Cassandra to another ring that is used for analytics and Spark integration

- **using separate racks and separate data centers - manage the exact replication using Cassandra**



Transactional queries *Analytical queries* **(integrate with Hive, Spark, Pig….)**

CQL (Cassandra Query Language)

- Cassandra's API is CQL, which makes it easy to look like existing database drivers to applications
- CQL is like SQL, but with some big limitations!
  - No JOINS
    - Our data must be de-normalized
    - So, it's still non-relational
  - All queries must be on some primary key
    - Secondary indices are supported
- CQLSH can be used on the command line to create tables, etc.
- All tables must be in a ***keyspace*** – keyspaces are like databases

Cassandra and Spark

- DataStax offers a Spark-Cassandra connector
- Allows users to read and write Cassandra tables as DataFrames
- Smart about passing queries on those DataFrames down to the appropriate level
- Use cases:
  - Use Spark for analytics on data stores in Cassandra
  - Use Spark to transform data and store it into Cassandra for transactional use

Let’s Play

- Install Cassandra on virtual Hadoop node
- Set up a table for MovieLens users
- Write into that table and query it from Spark

Installing Cassandra

Bernard Lee Kok Bang

1. Launch puTTY session

- Login to to putty as root account to install cassandra

  su root

- Make sure the python version in your machine is Python 2.7

  python –V # you should have python version 2.7.x

- Create a repository script to install Cassandra

  cd /etc/yum.repos.d

2. Create a datastax repository to install cassandra

- *vi datastax.repo*

```
[datastax]
name = DataStax Repo for Apache Cassandra
baseurl = http://rpm.datastax.com/community
enabled = 1
gpgcheck = 0
```

- *yum install dsc30*
- *service cassandra start* → **Launch Cassandra**

Cassandra Query Language (CQL)

- *cqlsh*

 # Create database

- *CREATE KEYSPACE movielens WITH replication = {'class': 'SimpleStrategy', 'replication_factor':'1'} AND durable_writes = true;*

- *USE movielens;*
- *CREATE TABLE users (user_id int, age int, gender text, occupation text, zip text, PRIMARY KEY (user_id));*
- *DESCRIBE TABLE users;*
- *SELECT * FROM users;*
- *exit*

Writing Spark Output into Cassandra

- *vi Cassandra_Spark.py*

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql import functions

def parseInput(line):
    fields = line.split('|')
    return Row(user_id = int(fields[0]), age = int(fields[1]), gender = fields[2], occupation = fields[3], zip = fields[4])

if __name__ == "__main__":
    # Create a SparkSession
    spark = SparkSession.builder.appName("CassandraIntegration").config("spark.cassandra.connection.host", "127.0.0.1").getOrCreate()
```

```
# Get the raw data
lines = spark.sparkContext.textFile("hdfs:///user/maria_dev/ml-100k/u.user")
# Convert it to a RDD of Row objects with (userID, age, gender, occupation, zip)
users = lines.map(parseInput)
# Convert that to a DataFrame
usersDataset = spark.createDataFrame(users)

# Write it into Cassandra
usersDataset.write\
    .format("org.apache.spark.sql.cassandra")\
    .mode('append')\
    .options(table="users", keyspace="movielens")\
    .save()

# Read it back from Cassandra into a new Dataframe
readUsers = spark.read\
.format("org.apache.spark.sql.cassandra")\
.options(table="users", keyspace="movielens")\
.load()

readUsers.createOrReplaceTempView("users")

sqlDF = spark.sql("SELECT * FROM users WHERE age < 20")
sqlDF.show()

# Stop the session
spark.stop()
```

 # To submit the python script
spark-submit --packages com.datastax.spark:spark-cassandra-connector_2.11:2.3.0 Cassandra_Spark.py

Output

```
|     624| 19|     M|       student|30067|
|     592| 18|     M|       student|97520|
|     434| 16|     F|       student|49705|
|     618| 15|     F|       student|44212|
|     471| 10|     M|       student|77459|
|     580| 16|     M|       student|17961|
|     223| 19|     F|       student|47906|
|     462| 19|     F|       student|02918|
|     550| 16|     F|       student|95453|
|     289| 11|     M|          none|94619|
|     521| 19|     M|       student|02146|
|     281| 15|     F|       student|06059|
|      36| 19|     F|       student|93117|
|     347| 18|     M|       student|90210|
|      68| 19|     M|       student|22904|
|     341| 17|     F|       student|44405|
|     179| 15|     M|entertainment|20755|
|     620| 18|     F|        writer|81648|
|     863| 17|     M|       student|60089|
|     904| 17|     F|       student|61073|
+-------+---+------+-------------+-----+
only showing top 20 rows
```

- *cqlsh*
- *USE movielens;*
- *SELECT * FROM users LIMIT 10;*
- *exit*
- *service cassandra stop*