Week 1

Apache Cassandra

Week plan

1. What is Cassandra?
2. Install Apache Cassandra on Ubuntu
3. Work with Cassandra and Python

Tutorial

What is Cassandra?

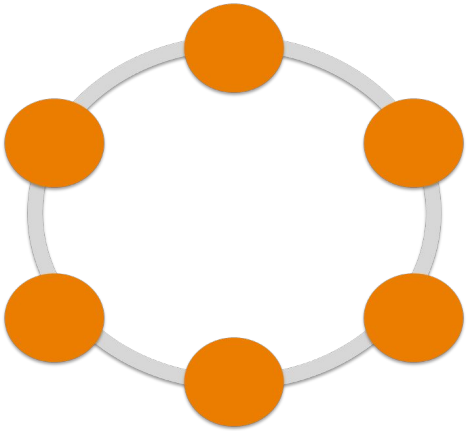
Apache Cassandra is a top level Apache project born at Facebook and built on Amazon's Dynamo and Google's BigTable, is a distributed database for managing large amounts of structured data across many commodity servers, while providing highly available service and no single point of failure.

Cassandra offers capabilities that relational databases and other NoSQL databases simply cannot match such as: continuous availability, linear scale performance, operational simplicity and easy data distribution across multiple data centers and cloud availability.

What is Cassandra?

Cassandra's architecture is responsible for its ability to scale, perform, and offer continuous uptime.

Rather than using a legacy master-slave or a manual and difficult-to-maintain sharded architecture, Cassandra has a masterless "ring" design that is elegant, easy to setup, and easy to maintain.



What is Cassandra?

In Cassandra, all nodes are equal, which means no master node, no master-slave relationships between nodes, no sharded system.

Cassandra's scalable architecture allows it to handle large amounts of data, thousands of user, and a great number of operations per second with ease. Even across multiple data storages.

Absence of master nodes and shards makes Cassandra resilient for node failures (no single point of weakness) and enables small uptime.

Features of Cassandra

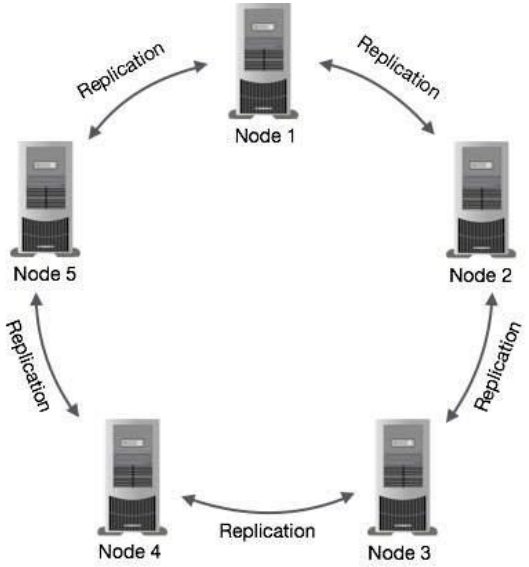
- **Elastic scalability** - add more nodes to accommodate more clients for data easily.
- **Always on architecture** - Business-critical applications cannot afford failures and Cassandra provides continuous availability without failure prone points.
- **Fast linear-scale performance** - Cassandra's ability to maintain quick response time by scaling load on nodes with their increase. More nodes - more throughput!
- **Flexible data storage** - all data formats (e.g. structured, semi-structured, and unstructured) are accommodated dynamically and with desired changes to them.
- **Easy data distribution** - flexibly distribute data across multiple data centers as needed.
- **Transaction support** - support for properties like Atomicity, Consistency, Isolation, and Durability (ACID).
- **Fast writes** - writes are blazingly fast, storing terabytes of data without loss of the read efficiency. Even on cheap commodity hardware.

Disadvantages

With all its shiny parts, Cassandra still has some let downs:

- A range scan implementation is far from perfect.
- A lot of adjustments are made at the cluster level.
- SSTable compaction, although occurs in the background, still spends a significant server resources and slows down.
- There is also a disadvantage related to communication between nodes, because that protocol does not able to transfer data as stream.

Data Replication in Cassandra

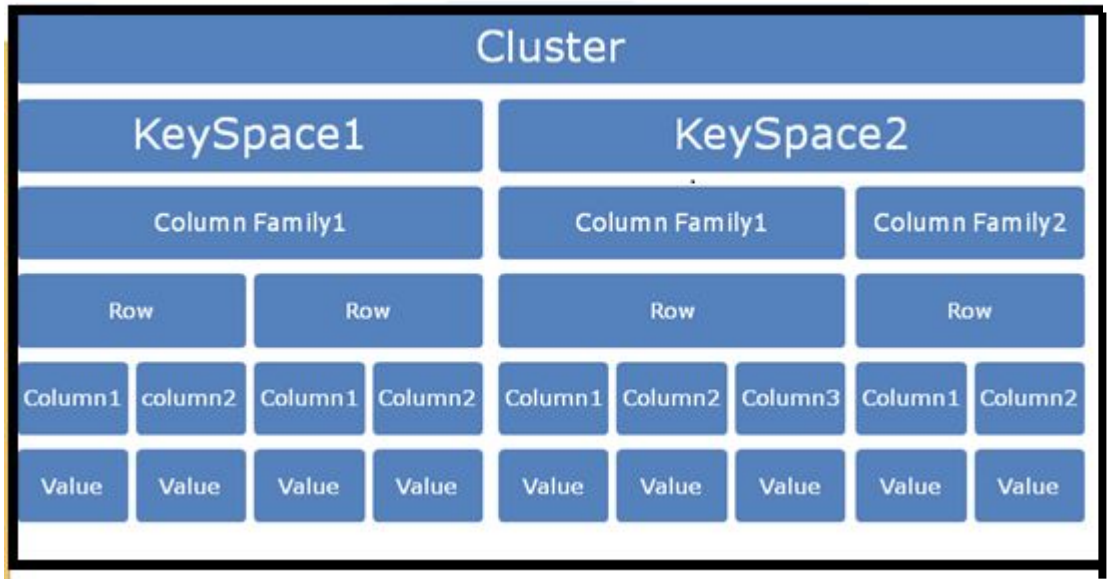


In Cassandra, replicas for a given piece of data are distributed on one or more of the nodes in a cluster. If it is detected that some of the nodes responded with an out-of-date value, Cassandra will return the most recent value to the client. After returning the most recent value, Cassandra performs update of the stale values in the background: process known as **read repair**.

Components of Cassandra

- **Node** − It is the place where data is stored, single machine.
- **Data center** − It is a collection of related nodes.
- **Cluster** − A cluster is a component that contains one or more data centers.
- **Commit log** − The commit log is a crash-recovery mechanism in Cassandra. Every write operation is written to the commit log.
- **Mem-table** − A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- **SSTable** − It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- **Bloom filter** − These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

Cassandra Architecture



Cluster

Cassandra database is distributed over several machines that operate together. The outermost container is known as the Cluster (collection of nodes). Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Keyspace

Keyspace is the outermost container for data in Cassandra. It is close in semantics to database.

Column Family

A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns. It has close meaning to a SQL table.

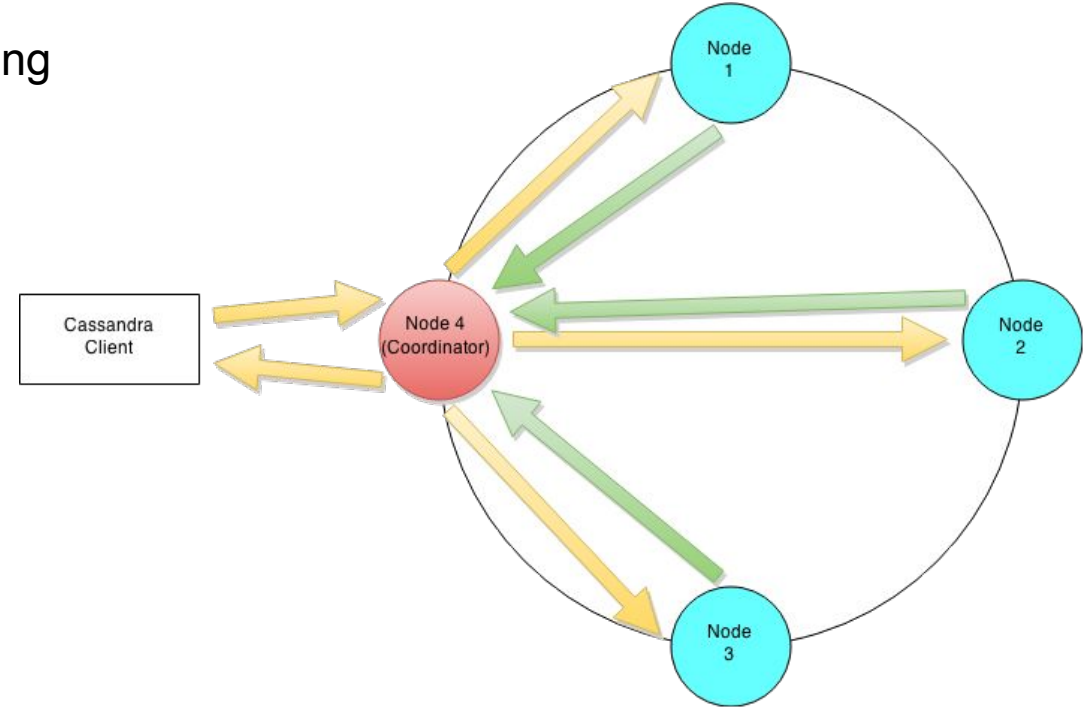
Cassandra Query Language

The Cassandra Query Language (CQL) is the primary language for communicating with the Cassandra database. CQL is purposefully similar to Structured Query Language (SQL) used in relational databases like MySQL and Postgres. This similarity lowers the barrier of entry for users familiar with relational databases. Many queries are very similar in these two. In fact, a lot of basic things are even exactly the same.

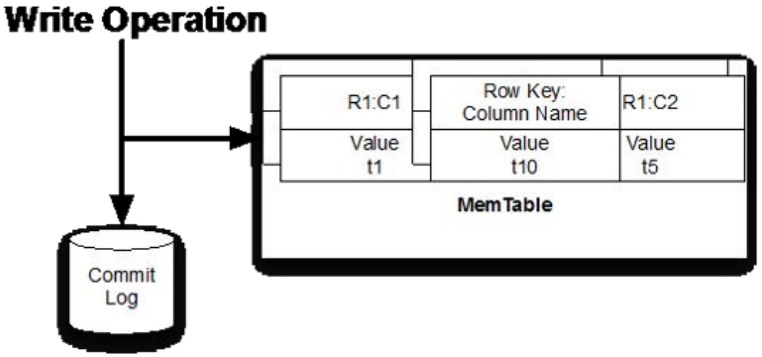
But Cassandra is a non-relational database, and so uses different concepts to store and retrieve data. Simplistically, a Cassandra keyspace is a SQL database, and a Cassandra column family is a SQL table (CQL allows you to interchange the words “TABLE” and “COLUMNFAMILY” for convenience).

Cassandra Operations

Let's find out how querying in Cassandra works.



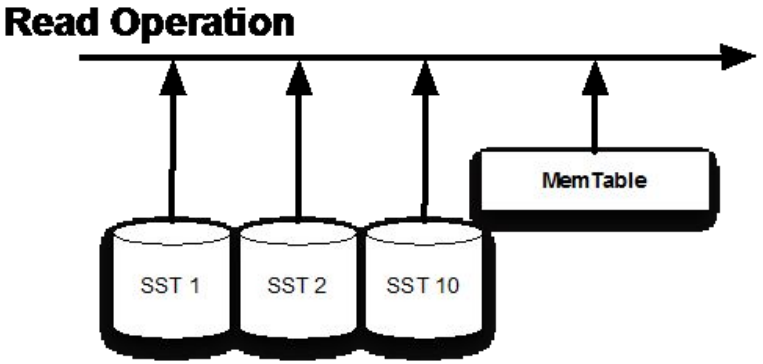
Write Operation



When write request comes to the node, first of all, it logs to the commit log. Then, Cassandra writes the data in the memtable. Mem-table is a temporarily stored data in the memory while Commit log save the transaction for backup purposes. When memtable is full, data is flushed to the SSTable data file.

In the case when remaining replicas lose data due to node downs or some other problem, Cassandra will make the row consistent by the built-in repair mechanism in Cassandra.

Read Operation



There are three types of read requests that are sent to replicas by a coordinator node.

1. Direct request
2. Digest request
3. Read repair request

The coordinator sends direct request to one of the nodes with replicas and gets the value that is being looked for. After that, the coordinator sends the digest request to the number of replicas specified by the consistency level and checks whether the returned data is an updated data.

Lastly, the coordinator sends digest request to all the remaining replicas. If any node gives out-of-date value, a background read repair request will update that data. This process is called read repair mechanism

Install Java on Ubuntu

Before installing Cassandra, make sure that Java is already installed on your computer. To make the Oracle JRE package available, you'll have to add a Personal Package Archives (PPA) using this command:

```
sudo add-apt-repository ppa:webupd8team/java
```

Update the package database:

```
sudo apt-get update
```

Install Java on Ubuntu

Then, install the Oracle JRE. Installing this particular package not only installs it but also makes it the default JRE. When prompted, accept the license agreement:

```
sudo apt-get install oracle-java8-set-default
```

After installing it, verify that it's now the default JRE:

```
java -version
```

Install Apache Cassandra on Ubuntu

We will use DataStax Community repository with a few simple steps to install Cassandra:

1. Add the DataStax Community repository to the /etc/apt/sources.list.d/cassandra.sources.list

sudo echo "deb http://debian.datastax.com/community stable main" | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list

Install Apache Cassandra on Ubuntu

2. Add the DataStax repository key to your aptitude trusted keys

sudo curl -L http://debian.datastax.com/debian/repo_key | sudo apt-key add -

3. Install the package:

sudo apt-get update
sudo apt-get install cassandra cassandra-tools

Install Apache Cassandra on Ubuntu

Because the Ubuntu packages start the Cassandra service automatically, you must stop the server and clear the data:

sudo service cassandra stop

Next remove the default cluster_name (Test Cluster) from the system table. All nodes must use the same cluster name.

*sudo rm -rf /var/lib/cassandra/data/system/**

sudo service cassandra start

Cassandra and Python

For managing Cassandra via Python you have to install cassandra-driver package from PyPI (Python Package Index), it can be done using pip.
Open console and run command:

```
pip install cassandra-driver
```

Create a Keyspace and Table

First, you need to connect to the Cluster. And check your connection. For executing this code you just need to run terminal and execute command "python".

After that write and execute line by line the following code:

```
>>> from cassandra.cluster import Cluster
>>> cluster = Cluster()
>>> session = cluster.connect()
>>> session
```

```
<cassandra.cluster.Session at 0x7f70d0909490>
```

Now let's create keyspace named "my keyspace", also don't forget to add parameters.

```
>>> query = session.execute("CREATE KEYSPACE my_keyspace WITH replication = {'class':'SimpleStrategy', 'replication_factor':1};")
```

Create a Keyspace and Table

Let's create a table, it will be called "users_".

```
>>> session.execute(
... """
... CREATE TABLE users_ (
...     id VARINT,
...     name TEXT,
...     born VARINT,
...     country TEXT,
...     PRIMARY KEY (id)
... );
... """
... )
```

```
<cassandra.cluster.ResultSet at 0x7f16285e0ed0>
```

Insert and Select Records

We are going to add data to our table in three different ways: a standard one, using variables and a dictionary.

```
>>> session.execute("USE users_")
>>> session.execute("INSERT INTO users_(id, name, born, country) VALUES (1, 'User_1', 1991, 'Ukraine')")
```

```
<cassandra.cluster.ResultSet at 0x7f16285e0590>
```

```
>>> name = "User_2"
>>> born = 1990
>>> country = "Poland"
>>> session.execute("INSERT INTO users_(id, name, born, country) VALUES (2, %s, %s, %s)", (name, born, country))
```

```
<cassandra.cluster.ResultSet at 0x7f1628622510>
```

```
>>> dict = {'name':'User_3','born': 1987,'country':'Czech Republic',}
>>> session.execute("""INSERT INTO users_ (id, name, born, country) VALUES (3, %(name)s, %(born)s, %(country)s)""", dict)
```

```
<cassandra.cluster.ResultSet at 0x7f1628544cd0>
```

Insert and Select Records

Let's select all the records to make sure that we have done everything right.

```
>>> query = session.execute("SELECT * FROM users_")
>>> for row in query:
...         print row
```

```
Row(id=2, born=1990, country=u'Poland', name=u'User_2')
Row(id=3, born=1987, country=u'Czech Republic', name=u'User_3')
Row(id=1, born=1991, country=u'Ukraine', name=u'User_1')
```

Now we are going to try to filter the entries, for example let's display data for a user named "User_1".

```
>>> query = session.execute("SELECT * FROM users_ WHERE name='User_1' ALLOW FILTERING;")
>>> for row in query:
…           print row
```

```
Row(id=1, born=1991, country=u'Ukraine', name=u'User_1')
```

Update and Delete Records

Let's change country for record with id = 3, and then check if something has changed in the table.

```
>>> session.execute("UPDATE users_ SET country='Canada' WHERE id=3")

>>> query = session.execute("SELECT * FROM users_")
>>> for row in query:
...         print row
```

```
Row(id=2, born=1990, country=u'Poland', name=u'User_2')
Row(id=3, born=1987, country=u'Canada', name=u'User_3')
Row(id=1, born=1991, country=u'Ukraine', name=u'User_1')
```

Update and Delete Records

Let's delete one of the records, such as having id = 2. And display all records from the table.

```
>>> session.execute("DELETE FROM users_ WHERE id=2")

>>>query = session.execute("SELECT * FROM users_")
>>> for row in query:
...         print row
```

```
Row(id=3, born=1987, country=u'Canada', name=u'User_3')
Row(id=1, born=1991, country=u'Ukraine', name=u'User_1')
```

Alter Table

Modify the column metadata of a table.

Adding a column

To add a column (other than a column of a collection type) to a table, use ALTER TABLE and the ADDinstruction as follows:

```
>>> session.execute("ALTER TABLE users_ ADD lastname text")
```

Dropping a column

To remove (*drop*) a column from the table, use ALTER TABLE and the DROP instruction.

```
>>> session.execute("ALTER TABLE users_ DROP lastname ")
```

Exercises

Task #1

Add a new record to the `users_` table with the following values: `id = 4`, `name = "User_4"`, `born = 1998`, `country = "France"`. And check what you've done using current code:

```
# type your code here
query = session.execute("SELECT * FROM users_")
for row in query:
    print row
```

To check the correctness of this and the following task you is proposed to solve few tests in the web page of the current course related with the respective task.

Task #2

Add new column into your table, let it be called "login", it should contain the same values as "name" column, but starting from lower case, type also should be the same. And display results using the code below.

```
# type your code here
for row in session.execute("SELECT * FROM users_"):
    print row
```

Task #3

Change country to "Ukraine" for user having id = 4 and display count of users from this country.

```
query = # type your code here
for row in query:
    print row
```


Thank You
For Your Attention