ILLINOIS DATA SCIENCE INITIATIVE

TECHNICAL REPORT

Introduction to Cassandra

Author:
Bhuvan Venkatesh

March 2, 2017

Introduction to Cassandra

Bhuvan Venkatesh[1] **and Robert Brunner**[2]

[1] *National Center For Supercomputing Applications (NCSA)*
[2] *Laboratory for Computation, Data, and Machine Learning*

Compiled March 2, 2017

https://github.com/lcdm-uiuc

INTRODUCTION

Apache Cassandra is an Open Source distributed database. The characteristics of Cassandra is that it is a high availability and fault tolerant database. The use cases for Cassandra are when you are familiar with the typical SQL databases and you need a way of modeling

INSTALLATION

For those of you on docker to run all you need to do is

```
$ docker pull cassandra
$ docker run cassandra
```

If not, the non-dockerized installation is below.

Prerequisites

1. Java 1.6 or Above
2. Python 2.7 or Above

First you will need to set a new user for cassandra, letting cassandra run under normal users is not recommended, so we create a new user

```
$ useradd cassandra # Name for the user
```

Navigate to *http : //cassandra.apache.org/download/* and select the most recent stable version of cassandra (Usually the most recent or the second most recent version) and execute

```
$ wget <url>
$ tar zxvf <file >.tar.gz
$ mv <folder> /opt/cassandra # May need root
$ export CASSANDRA_HOME=/opt/cassandra
$ chown -R cassandra $CASSANDRA_HOME
```

After this, create the directories that cassandra keeps its data in.

```
$ mkdir /var/lib/cassandra
$ chown cassandra /var/lib/cassandra
$ mkdir /var/log/cassandra
$ chown cassandra /var/log/cassandra
```

You can change this in the settings explained in a future section.

To run cassandra switch the cassandra user, and execute the cassandra executable

```
$ sudo su cassandra
$ cd $CASSANDRA_HOME
$ ./bin/cassandra -f # or if you want to run
    it in the background -d
```

And you have cassandra running! To make sure that everything is working on a different terminal

```
$ cd $CASSANDRA_HOME
$ ./bin/cqlsh
>       #If there are no errors, your single
   node cluster is up.
```

CONFIGURING MULTINODE CLUSTERS

Backing up date on previous

First, if you have a single node cluster you **must** complete these instructions or your system will be in a halfway state. Delete all the data associated with your cassandra instance. If you have important data, there are ways to back it up.

```
> ./bin/cqlsh
> COPY table_name TO '/user/you/file_name'
# After the switch to multinode you can do
> COPY table_name FROM '/user/you/file_name'
```

If you are comfortable with your data backup, delete the old datastore

```
sudo rm -rf /var/lib/cassandra/*
```

Setting up the configurations

Got to the $CASSANDRA_HOME/cassandra.yaml file. You can update the following values to set up a multi node system

```
# Find these values in the existing file
cluster_name: 'MustBeNamed'

# Some fields may be added
seed_provider:
  - class_name: org.apache.cassandra.locator.
      SimpleSeedProvider
    parameters:
         - seeds: "this_server_ip,server_ip_2
             ,..."
```

```
# Firewall Addresses
rpc_address: this_server_ip
listen_address: this_server_ip

# Snitches
endpoint_snitch: GossipingPropertyFileSnitch
```

1. cluster_name: The name of your cluster. In order for other nodes to join it must be named the **same thing** on each server.

2. seed_provider: This property is how cassandra knows about which nodes are directly connected to which. It will assume that the nodes are connected in a ring where the first ip is connected to the second, the second to the third and so on and the last is connected to the first.

3. rpc_address, listen_address: These are the two IPs that cassandra will do its communication on. These are usually localhost and default to two ports.

4. endpoint_snitch: A snitch in computing shares metadata about the cluster (the health of each node, the free space etc) by occasionally telling its neighbors (fingertable accessible). This is the gossip protocol in work and that is why we go with the GossipingPropertyFileSnitch and not the SimpleSnitch for fault tolerance.

For those on linux systems iptables are the usualy firewall the kernel has up to block incoming traffic, we will need to circumvent that.

```
# Redo the following command for each
# machine connected to a single machine.

$ iptable -A INPUT \ # We are going to be
    receiving data
  -p tcp -s connected_machine_ip \ # Replace
      the IP as goes
  -m multiport --dports 7000,9042 \ # Default
      port
  -m state --state NEW,ESTABLISHED -j ACCEPT
$ service iptables-persistent restart
```

To see if it worked try the following command.

```
$ $CASSANDRA_HOME/bin/cqlsh ANY_SERVER 9042
```

CQLSH/DRIVERS

There are two usual ways of interacting with cassandra, either the cqlsh executable or drivers for a specific programming language.

The CQLSH command line is mainly for maintenance or executing queries during setup/backup/destruction. This is because it is easy to use, easy to script, and easy to check for errors. But, there are drawbacks to using it. One drawback is CQLSH will *always* connect the instance that you supply it with. Cassandra thrives on the ability for any client to connect to any node, reducing the point of failures and helps load balance the users among the cluster which decreases the probability of failures.

This is where community drivers come in. One can find drivers for many popular programming languages. One such driver for python (the highly popular datastax cassandra-python) is showcased in the code snippet below.

```
from cassandra.cluster import Cluster

cluster = Cluster(
    ['machine_ip_1', ...],
    port=...,
    )
try:
  sess = cluster.connect()
  sess.set_keyspace('...')
  rows = sess.execute('SELECT␣name␣FROM␣users␣\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣WHERE␣email=%s', ['email1@gmail.com'])
  for user_row in rows:
      print(user_row.name)
except:
  pass
finally:
  # No Need to close!
```

The convention is that there is one cluster object that connects once at the start of the program, then there is one session per keyspace that executes queries.

The cluster object represents an object that can connect to any node in the cluster. The session object manages the session, there is no need to open or close it. And mostly, you can execute queries very easily in cassandra SQL.

OVERVIEW OF DATA MODELING

If you are familiar with more traditional Relational Database Management Systems (RDBMS) then you are familiar with the concept of tables and joins where the table represents spreadsheets. Imagine Cassandra as a traditional RDBMS without the ability to do JOINs and where every statement ought to have a LIMIT BY for performance (there are ways around it).

If you are otherwise not familiar with traditional data stores, cassandra stores its data in tables, much like a spreadsheet. The tables live in entities called keyspaces. Keyspaces are ways for cassandra to keep track of resillience levels in tables in cqlsh syntax here is how you create one.

```
> CREATE KEYSPACE "NAME" WITH replication =
      {
        'class': 'SimpleStrategy', # The
            default
        'replication_factor' : 3 # For example
      };
```

To start using a keyspace, do the following

```
> USE KEYSPACE "KEYSPACE"
```

To create a table it is pretty similar as well. One of the technical notes to data modeling is that you want to design your table with the idea that as much information as possible should be in one row of the table. Cassandra likes wide column stores because it cannot do JOINs with other tables. This means that it preferes to have denormalized data, repeated data and so on versus a single representation for everything. There are some basic crud operations below (The should look similar to SQL for those of you familiar).

```
> CREATE TABLE "SAMPLE" (
    sample_identifier text,
    sample_time timestamp,
    sample_size int,
    sample_data blob,
    PRIMARY_KEY (sample_identifier,
        sample_time) # You must have a key!
)
```

```
> INSERT INTO SAMPLE VALUES ("1", toTimestamp(now()), 10, {"data"})
> DELETE FROM SAMPLE WHERE sample_size = 10
```

TUNING PARAMS

There are many basic ways to tune Cassandra.

First, During your queries, make sure to set the appropriate consistency level. The consistency level tells the query to return succeeded or failed based on the number of nodes that get written to. The other nodes may eventually be replicated, but for your queries to return, this is the standard way. The most popular consistency levels are as follows.

1. ONE/TWO/THREE, this means the read/write only has to go to one/two/three node(s) before returning
2. QUORUM, this means the read/write has to go to half the nodes plus one. This is because for workloads with equal reads and writes, having a quorum is guaranteeing consistency.
3. ALL, this means the read/write has to go to all the nodes in the cluster (or almost all) before returning. This is usefull to pair with a ONE consistency level in the case of optimize workloads (more in the next section).

Workloads - Sessions

For read heavy/write light workloads, set the *read session* consistency to ONE with he following command in one session.

```
> CONSISTENCY ONE
```

And the write consistency to ALL with the following command

```
> CONSISTENCY ALL
```

For write heavy/read light workloads, set the consistency vice versa as above. For the case where you need a balanced workload, set both consistencies in the session to QUORUM, so you get consistent results in both reads and writes with high probability.

USUAL DO NOTS

Although cassandra gives you a lot of tunability parameters there are practices that should be avoided at all cost because it severly impacts the performance or the reliability of the database.

1. Do not put cassandra's data files on network attached storage
2. Try not to host cassandra's data files directly on hadoop either, cassandra has reliability built into it.
3. Try not to store your data on multiple separate tables to join later. Cassandra is part of the wide column store family meaning that

REFERENCES

https://www.digitalocean.com/community/tutorials/how-to-run-a-multi-node-cluster-database-with-cassandra-on-ubuntu-14-04
https://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dml$_{config_{consistency_{c}}}$.html