

CONTROL-EF

AT70.18 : Software Architecture Design

Testing Document



Fathima Shafana (st121985)

Htoo Lwin (st120832)

Table of Contents

Table of Contents	2
Introduction	3
Test Approach	4
2.1 Process of Testing	4
2.2 Test Plan	4
2.3 Types of testing	4
2.4 Testing Approach	4
Test Environment	5
Testing Tools	7
4.1 Apache JMeter 5.4.1 tool	7
4.2 Cassandra-Stress Tool	7
Testing Scenario	7
5.1 Stress Testing for PostgreSQL database	7
5.2 Stress Testing for Apache Cassandra	11

1. Introduction

Control-Ef is a system that supports transcript searches across the video. The main focus of the system is to provide the exact timestamp of the keyword within videos that users search for. Therefore, the system mainly focuses on getting a quick response at the earliest possible. However, the transcript generated is of high volume and highly unstructured. Therefore, one of the main goal of the system is to choose an architecture that can provide a speedy response to the query made by the user, in this context is the provision of exact timestamp of the keyword searched when multitude of users are logged in and many videos with similar keywords exist in the system.

Stress testing is typically carried out to measure the performance of the system on its robustness and error handling capabilities under very heavy load conditions. Since, performance and scalability are key quality attributes of the system Control-Ef, this stress testing is paramount. Search for a keyword is the primary functionality of this system, hence, the stress testing is done to check the response time for the search of a keyword for increasing number of videos and the transcripts. The testing is performed for both of the proposed architectures which are based on classical PostgreSQL and Apache Cassandra.

2. Test Approach

2.1 Process of Testing

The testing is basically on the database of the system as the data is to be read from the database to fetch the timestamp of the keyword the user searches. Therefore, the time taken to retrieve the timestamps of the searched keyword, i.e. response time, is the factor that decides the choice of architecture of the system. The process of testing is basically to measure the latency for a search of keywords for varying transcript sizes.

2.2 Test Plan

The test plan is to simulate an increase in the number of transcript data entries starting from 2000 records in both databases until they reach 500000 entries for a single user. The test is carried out at multiple iterations and the average latency is considered as the time taken for the keyword search across the database.

2.3 Types of testing

As mentioned above, Stress testing is carried out to identify which of the database withstands for the increasing number of transcript entries.

2.4 Testing Approach

Stress testing is carried out with two of the popular automated testing tools available today. The testing for PostgreSQL database is done by Apache Jmeter whereas the Cassandra cluster is tested with Cassandra-Stress tool.

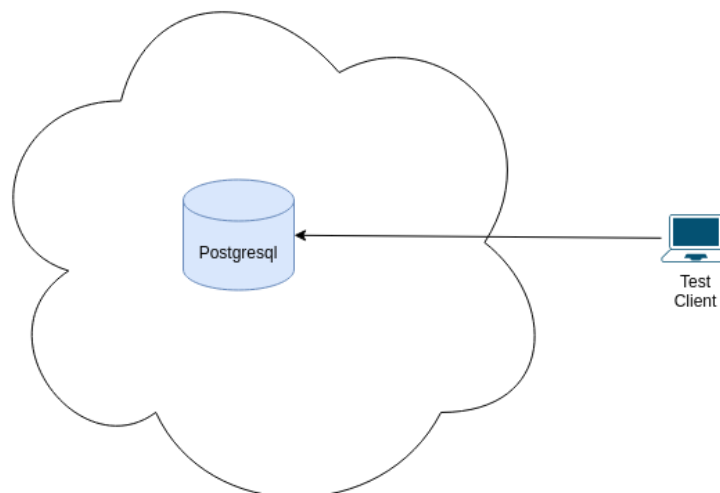
3. Test Environment

The tests are run on the Guppy server available at CSIM here at AIT. Guppy was chosen as the test environment because it provides a standard environment that can be used by all team members in testing out the databases in terms of the resources available to the machine hosting the databases. If a local machine was chosen, the results would vary as each developer machine was different.

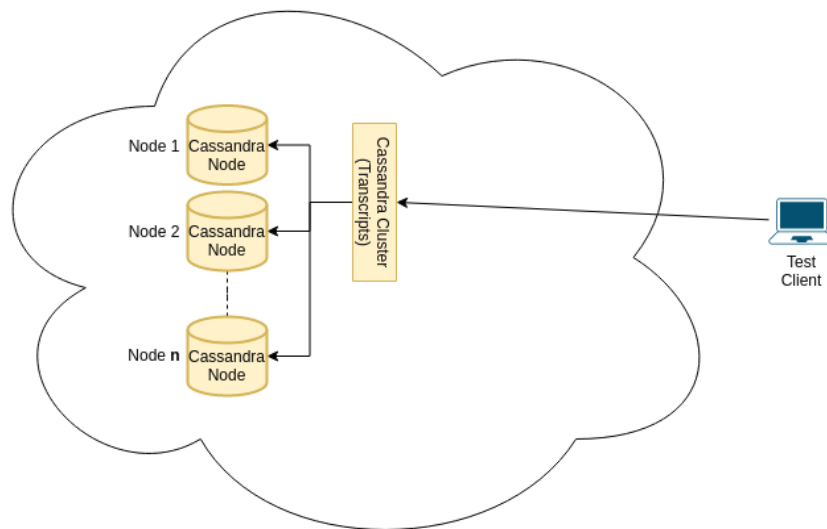
Furthermore, the databases themselves are located in Docker containers on Guppy. PostgreSQL uses one container while Cassandra uses as many containers as there are nodes in the Cassandra Cluster. Bear in mind, that all the nodes of the cluster are located on the same machine. An example of a Cassandra cluster up and running with some data is shown below. Each node has its own separate container. The connection to the containers is through an SSH tunnel.

```
st120832@guppy:~$ docker exec -it control-cassandra2 nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
// State=Normal/Leaving/Joining/Moving
-- Address      Load       Tokens     Owns (effective)  Host ID                               Rack
UN 172.19.0.3    16.28 MiB  256        48.0%             18caa8bd-0f90-4f56-b6da-1f057d354ef7  rack1
UN 172.19.0.2    18.82 MiB  256        52.0%             e833e23a-c5b1-4ec4-a888-1dcc379b7f50  rack1
```

The architecture of the Postgres testing environment is shown below. More details on how to set up the container can be found on this [README](#).



The architecture of the Cassandra testing environment is shown below. For our particular test experiment, a cluster of **two nodes** was used. More details on how to set up the cluster can be found on this [README](#).



4. Testing Tools

4.1 Apache JMeter 5.4.1 tool

For stress testing on PostgreSQL, Apache JMeter is used. JMeter is a widely used tool utilized for analyzing and measuring the performance of a variety of services. In the case of this study, it is used for the particular case of database stress testing.

JMeter natively supports most relational databases through JDBC driver plugins and this is the case for PostgreSQL as well.

4.2 Cassandra-Stress Tool

However, unlike PostgreSQL, Cassandra is not supported natively by Apache JMeter as found by our research and experiments. There are plugins such as this [one](#) but they are sorely out of date and do not work with the latest versions of both JMeter and the Cassandra driver.

Therefore, we found through our research, another tool called the [Cassandra Stress tool](#). This was found to be more suitable for our testing purposes and is officially supported by the Apache foundation. Other benchmarking tools such as [NoSQLBench](#) can also be used.

5. Testing Scenario

5.1 Stress Testing for PostgreSQL database

Phase 1: Prepare test data sets for Stress Testing

According to the Test Plan, a series of tables with records ranging from 2000, 5000, 10000, 25000, 50000, 100000, 300000 and 500000 were built in PostgreSQL database as transcript1, transcript2, transcript3, transcript4, transcript5, transcript6, transcript7 and transcript8 respectively as shown in Figure 1. The database was populated with dummy data on video transcripts obtained from Kaggle repository (*Source: <https://www.kaggle.com/goweiting/ted-talks-transcript>*). The data set consisted of transcripts in various languages. Hence, only the transcripts in English Language were selectively filtered for the testing procedure.

List of relations			
Schema	Name	Type	Owner
public	role	table	postgres
public	transcript1	table	postgres
public	transcript2	table	postgres
public	transcript3	table	postgres
public	transcript4	table	postgres
public	transcript5	table	postgres
public	transcript6	table	postgres
public	transcript7	table	postgres
public	transcript8	table	postgres
public	user_account	table	postgres
public	user_account_roles	table	postgres
public	video	table	postgres
(12 rows)			

Figure 1: List of Relations in PostgreSQL

Phase 2: Perform Stress Testing

As the database is built, Apache Jmeter 5.4.1 was used to test the JDBC Request.

Step 1:

Configure JDBC Connection for controlefdb

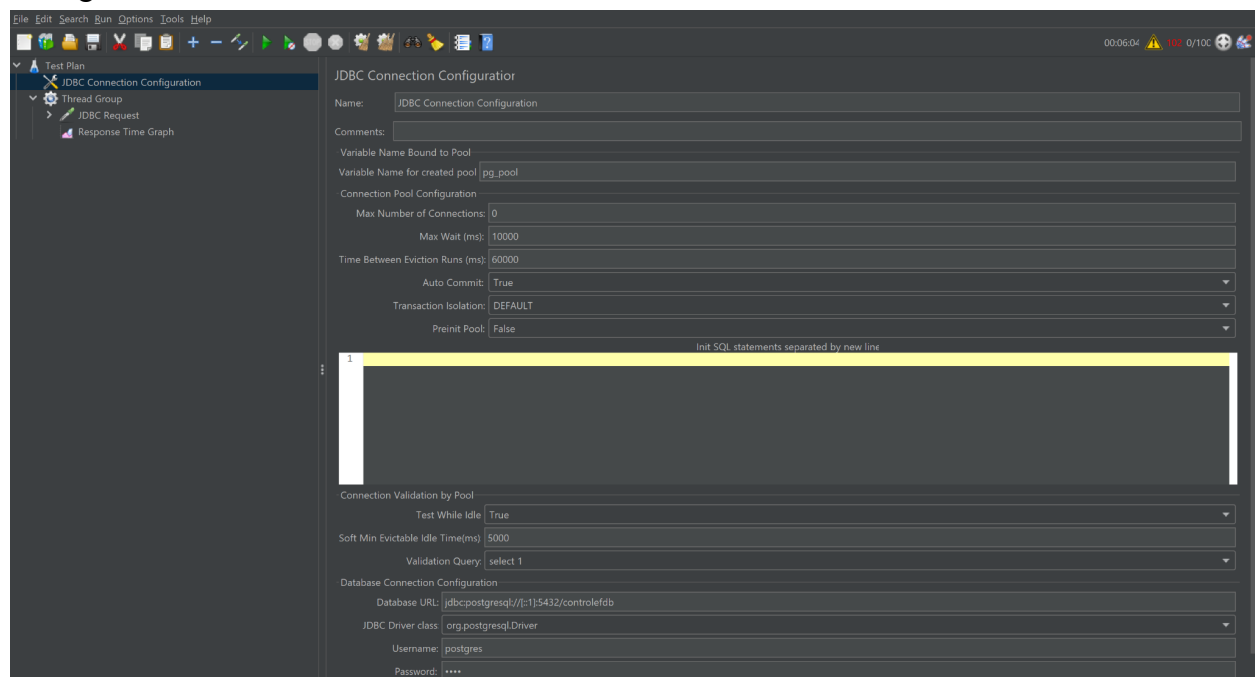
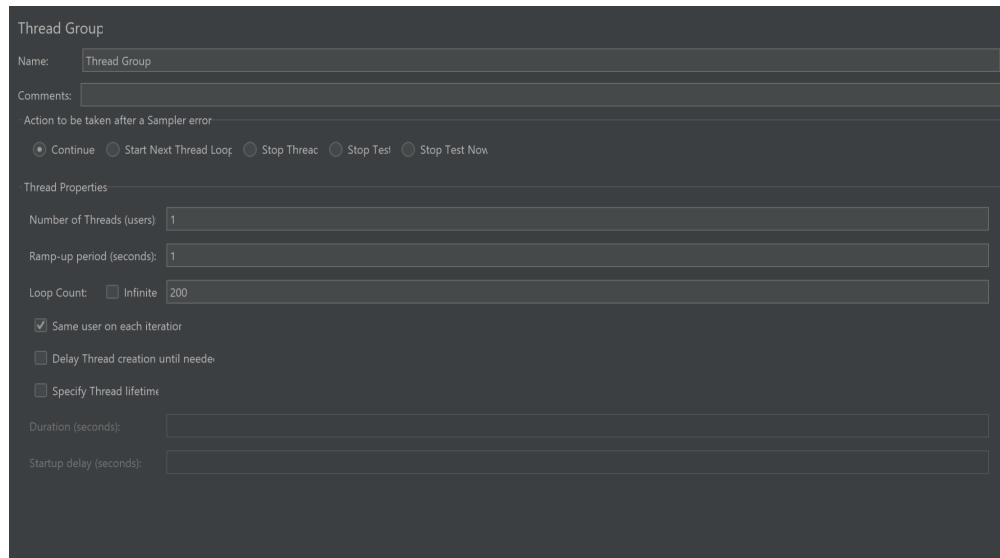


Figure 2 : JDBC Configuration

Step 2: Create Thread Group

Since load testing is not the main form of testing, the number of threads (users) was maintained at 1. The test was conducted in 200 iterations as an optimal value.

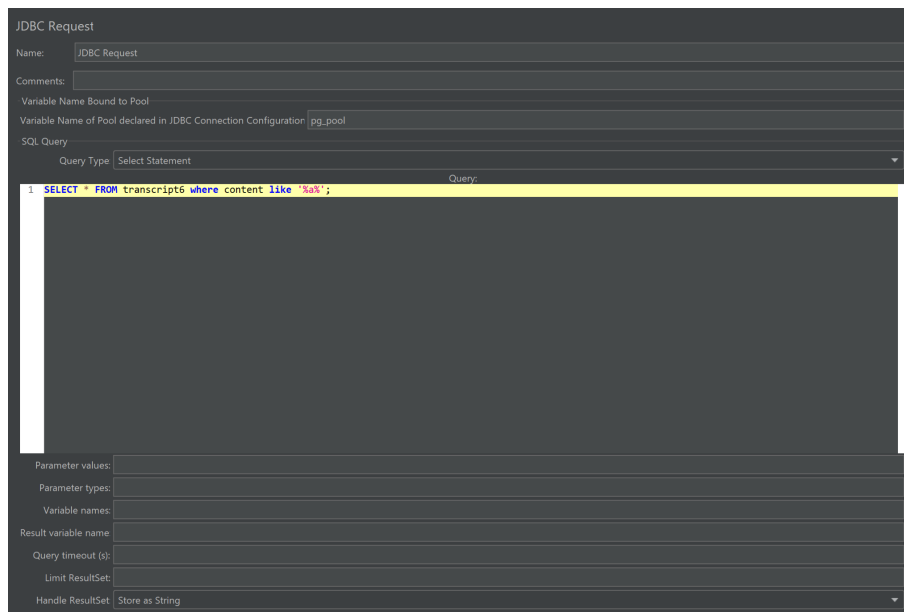


The screenshot shows the 'Thread Group' configuration window in JMeter. The 'Name' field is set to 'Thread Group'. The 'Comments' field is empty. Under 'Action to be taken after a Sampler error', the 'Continue' radio button is selected. The 'Thread Properties' section shows 'Number of Threads (users)' set to 1, 'Ramp-up period (seconds)' set to 1, and 'Loop Count' set to 200 with the 'Infinite' checkbox unchecked. The 'Same user on each iteration' checkbox is checked. The 'Delay Thread creation until needed' and 'Specify Thread lifetime' checkboxes are unchecked. The 'Duration (seconds)' and 'Startup delay (seconds)' fields are empty.

Figure 3 : Thread Group for test

Step 3: Add Samplers : JDBC Request

- JMeter natively supports PostgreSQL through JDBC driver plugins and this is the case for our system as well.
- The following SQL query was used to run the testing across all the 8 relations with transcripts available in the database.



The screenshot shows the 'JDBC Request' configuration window in JMeter. The 'Name' field is set to 'JDBC Request'. The 'Comments' field is empty. The 'Variable Name Bound to Pool' field is empty, and the 'Variable Name of Pool declared in JDBC Connection Configuration' field is set to 'pg_pool'. The 'SQL Query' field contains the query: `1. SELECT * FROM transcript6 where content like '%a%';`. The 'Query Type' dropdown is set to 'Select Statement'. The 'Parameter values', 'Parameter types', and 'Variable names' fields are empty. The 'Result variable name' field is empty. The 'Query timeout (s)' field is empty. The 'Limit ResultSet' field is empty. The 'Handle ResultSet' dropdown is set to 'Store as String'.

Figure 4: JDBC Request Setup

Step 4: Add Listeners

Many listeners such as View Result Tree, Aggregate Graph and View Results in table were added to interpret the results in a better way.

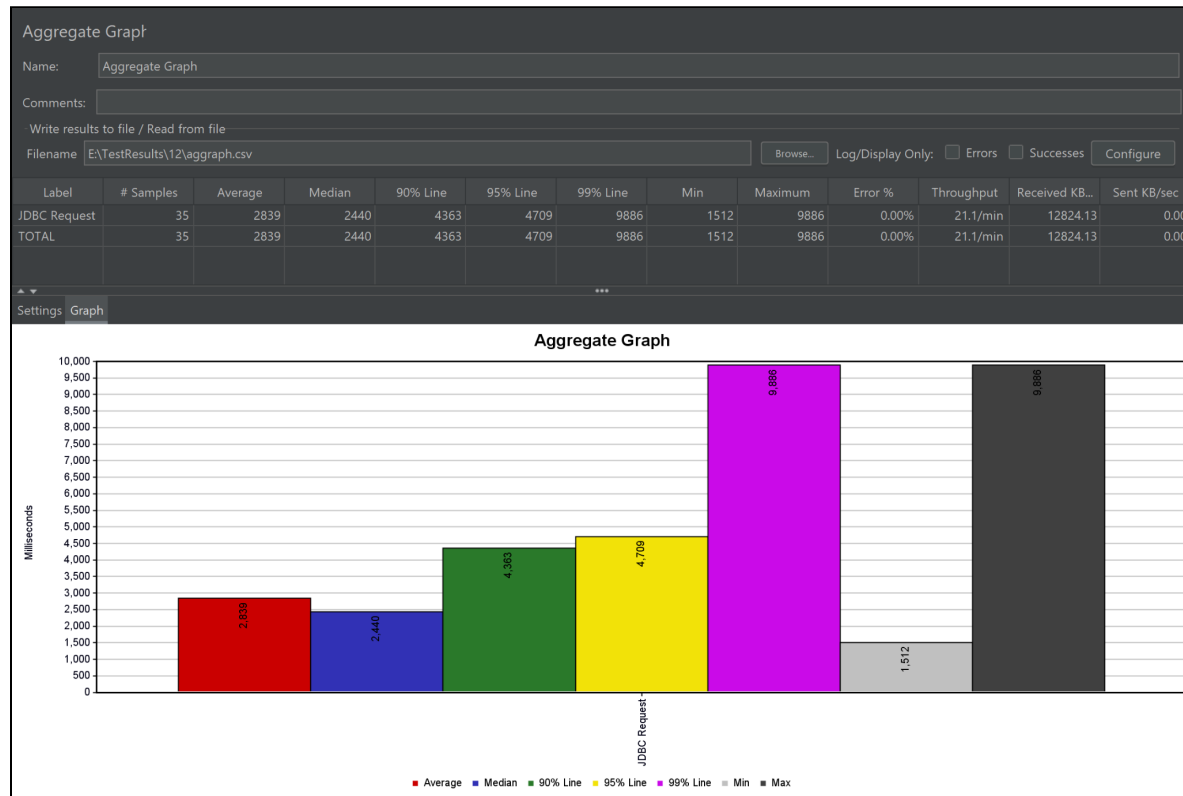


Figure 5: Aggregate Graph for a search query on 500,000 records

Step 4: Perform Stress Testing by running the configured test plan on each of the tables from 'transcript1' to 'transcript8' separately.

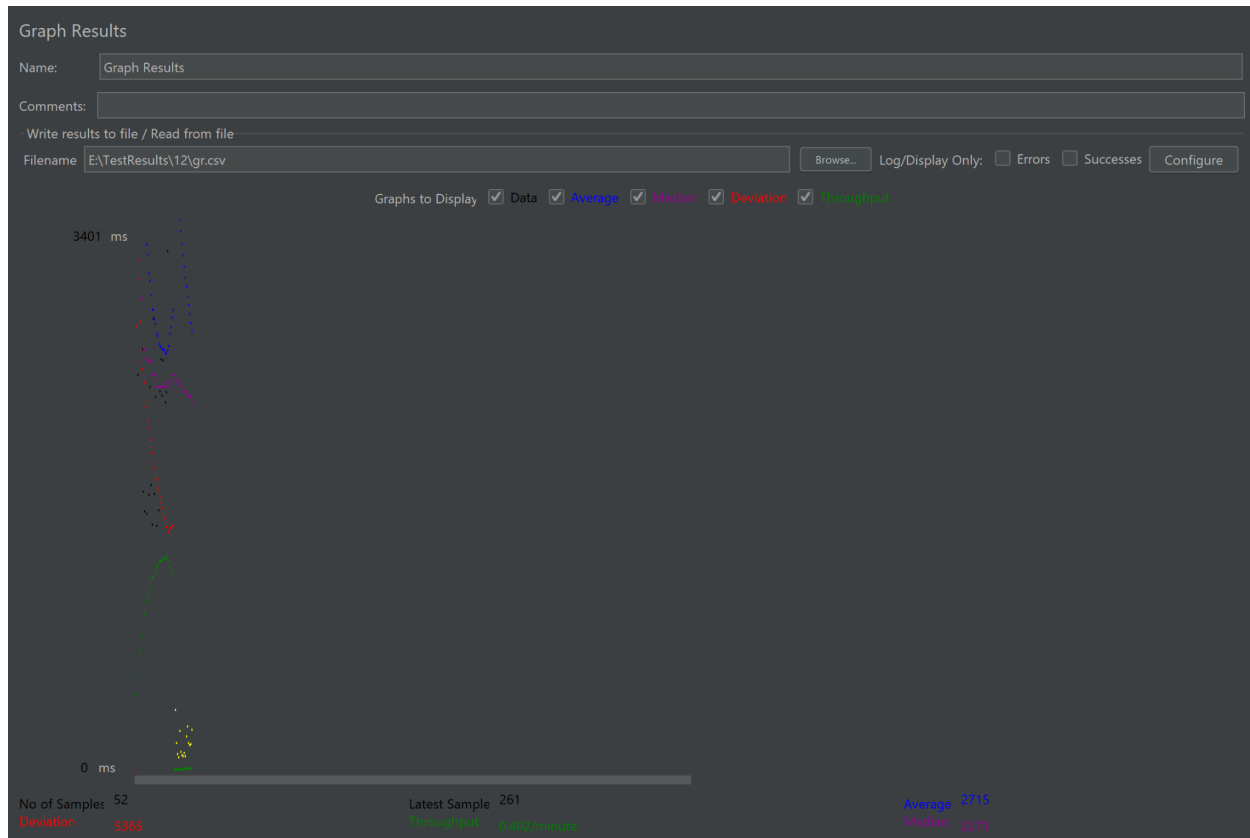


Figure 6: Graph Results for a search query on 500,000 records

5.2 Stress Testing for Apache Cassandra

The popular Cassandra-Stress tool has been utilized to perform Stress Testing on the Cassandra keyspace of the system. It is a Java-based stress testing utility used for basic benchmarking and load testing a Cassandra cluster. These may include:

- Quickly determine how a schema performs.
- Understand how your database scales.
- Optimize your data model and settings.
- Determine production capacity.

For our particular test scenario, a YAML-based approach by which we can easily customize the parameters of the stress test was utilized. The tool will be used to both generate and read data. In contrast, the data was imported to Postgres using a CSV file containing data acquired from Kaggle Repository. This however proved impossible for Cassandra since there were errors in the data. Therefore, generating the data via the stress tool seemed the most viable option. An example YAML file used is shown below.

```

keyspace: transcript
table: transcripts_by_content
columnspec:
  - name: content
    size: uniform(5..50)
  - name: video_id
    size: fixed(11)
insert:
  # How many partition to insert per batch
  partitions: fixed(1)
  # How many rows to update per partition
  select: fixed(1)/500
  # UNLOGGED or LOGGED batch for insert
  batchtype: UNLOGGED
queries:
  read1:
    cql: select * from transcripts_by_content where content like '%a%'
    fields: samerow

```

Figure 7: Sample YAML file

Step 1: Create a YAML file to set the keyspace and the schema of the relation that should be tested. The YAML also contains parameters to configure the data population as can be seen in the **columnspec** definition above. The **insert** definition is the query that inserts the generated data into the database.

Step 2: Set the parameters of cassandra-stress command.

Step 3: Run cassandra-stress to insert 'n' records into the database. The command is
/cassandra-stress user profile=controlef.yml n=100000 cl=ONE
ops\((insert=1\) **-rate threads=1 -graph file=test.html title=test**
revision=test1 -node localhost,localhost:4127. Here the parameter n can be customized.

Step 4. Run cassandra-stress to read 200 times. The average of these read times are taken as the final result.

Since the goal is to obtain the latency as the number of records increase, steps 3 and 4 (listed in bold above) are run for each level of data records, similar to that of PostgreSQL. More detailed information on the specific commands and parameters can be found in this [README](#).