Implementing and Evaluating Joins over NoSql Cassandra DBs

HaridimosKondylakis
Forth-Institute
Heraklion, Crete, Greece
kondylak@ics.forth.gr

ApostolosPlanas
Computer Science Department
University of Greece
planas@csd.uoc.gr

DimitrisPlexousakis
Forth-Institute
Heraklion, Crete, Greece
dp@ics.forth.gr

Table of Contents

1- INTRODUCTION 1
2- RELATED WORK 2
3- IMPLEMENTING MEMORY JOINS OVER CASSANDRA DBS 4
4- EVALUATION 5
5- CONCLUSION 11
6- REFERENCES 12

1. Introduction

In the last few years there has been an explosion of interest in using NoSql databases[1] for big data. There are a lot of different NoSql databases such as Apache HBase[2] , MongoDB[3] and Cassandra[4].
Initialy NoSql databases designed to support only single-table queries and excluded the support for join operations however modern applications increasingly require the efficient combination of information from multiple tables and column-families. Our interest lies with Apache Cassandra an open source, distributed and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world that provides high availability, scalability, fault-tolerance and consistent. In 2015 DataStax started to claim that Cassandra can now perform joins just as well as an RDBMS can. Yet, no timing measurements have been provided for these joins, no open source for these joins and no algorithms have yet been published. In this thesis my job was to create data for benchmarking and run time measurements with the help of a Java API to perform joins called CassandraJoins[15]. The queries for these tests were taken by BSBM[5] in Sql translated to Cql with the help of Sqoop[6] and this queries tested and time-measured by CassandraJoins in a cluster of multiple nodes.

2. Related Work

What is NoSql (Not Only Sql)?

NoSql encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications:

- Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.
- Long gone is the twelve-to-eighteen month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.
- Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally to millions of users.

The Benefits of NoSql

When compared to relational databases, NoSql databases are more scalable and provide superior performance, and their data model addresses several issues that the relational model is not designed to address:

- Large volumes of rapidly changing structured, semi-structured, and unstructured data
- Agile sprints, quick schema iteration, and frequent code pushes
- Object-oriented programming that is easy to use and flexible
- Geographically distributed scale-out architecture instead of expensive, monolithic architecture

What is aSql join?

SQL is a special-purpose programming language designed for managing information in a relational database management system (**RDBMS**). The word relational here is key; it specifies that the database management system is organized in such a way that there are clear relations defined between different sets of data.
ASql join is a Structured Query Language (**SQL**) instruction to combine data from two sets of data (e.g. two tables).

What is Apache Cassandra?

Apache Cassandra is an open source, distributed and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world. It provides highly available service with no single point of failure.
Listed below are some of the notable points of Apache Cassandra:

- It is scalable, fault-tolerant, and consistent.
- It is a column-oriented database.
- Its distribution design is based on Amazon’s Dynamo and its data model on Google’s Bigtable.
- Created at Facebook, it differs sharply from relational database management systems.

- Cassandra implements a Dynamo-style replication model with no single point of failure, but adds a more powerful “column family” data model.
- Cassandra is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, eBay, Twitter, Netflix, and more.

What is a Cassandra Query Language (CQL)?

Users can access Cassandra through its nodes using Cassandra Query Language (CQL). CQL treats the database (Keyspace) as a container of tables. Programmers use cqlsh: a prompt to work with CQL or separate application language drivers.
Clients approach any of the nodes for their read-write operations. That node (coordinator) plays a proxy between the client and the nodes holding the data.

What is Big Data?

Big data is a term for data sets that are so large or complex that traditional data processing applications are inadequate to deal with them.
Big data can be described by the following characteristics:
Volume
- The quantity of generated and stored data. The size of the data determines the value and potential insight- and whether it can actually be considered big data or not.

Variety
- The type and nature of the data; this helps people who analyze it to effectively use the resulting insight.

Velocity
- In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development.

Variability
- Inconsistency of the data set can hamper processes to handle and manage it.

Veracity
- The quality of captured data can vary greatly, affecting accurate analysis.

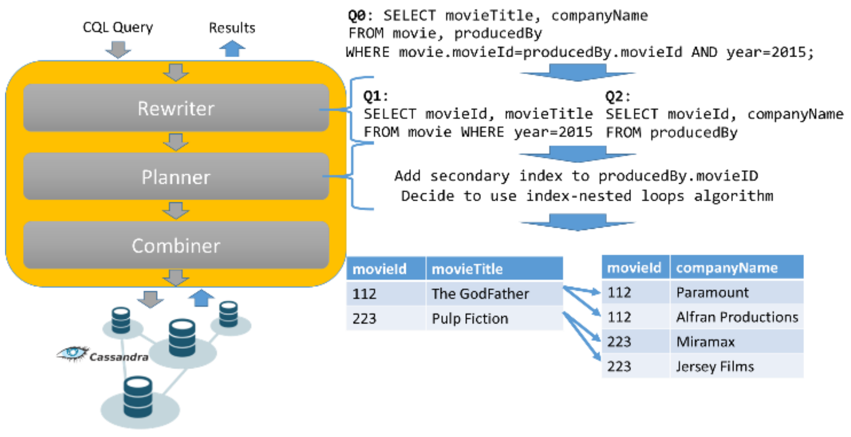
On 24 of March in 2015,DataStax published an article[7] stating that now it’s possible to do joins with two different ways. Using Apache Spark[8] and Cassandra or using the new SimbaODBC driver[9] and Cassandra and in the same article they stated that they will announce soon a way to do Joins using the Titan open source Graph database. Yet, no timing measurements have been provided for these joins, no analysis has been given and no open source code for these joins. Also their product Simba-DatastaxSql ODBC driver, a program that translates Sql to Cql doesn’t offer full support for joins and runs extremely slow. For this reason we present a solution that efficiently supports joins over such databases. More specifically, we present a query optimization and execution module placed on top of Cassandra clusters that is able to efficiently combine information stored in different column-families. Its safe to say that our approach is the first and the only available open source solution allowing joins over NoSQL Cassandra databases and its much faster than ODBC driver.

3. Implementing memory Joins over Cassandra DBs

Our query optimization and execution module can be placed on top of any Cassandra cluster and is composed of the following components, shown in Fig. 1:

a) Rewriter: The rewriter accepts the CQL query containing joins and creates the queries for accessing each individual column-family/tables.For example assuming that Q0is issued by the user, this module produces as output Q1 and Q2 as shown in Fig. 1.

b) Planner: This component plans the execution of the individual queries as constructed by the rewriter. First it identifies the available indices on the queried column-families and tries to comply with R2. For example, if the queries don't restrict all partition keys they can only run if there are available secondary indices on these keys. To satisfy this restriction the planner automatically generates secondary indices on the required fields. In our running example, a secondary index will be automatically generated to the producedBy.movieID column.



```
Q0: SELECT movieTitle, companyName
FROM movie, producedBy
WHERE movie.movieId=producedBy.movieId AND year=2015;
```

```
Q1:                                   Q2:
SELECT movieId, movieTitle            SELECT movieId, companyName
FROM movie WHERE year=2015            FROM producedBy
```

```
Add secondary index to producedBy.movieID
Decide to use index-nested loops algorithm
```

movieId	movieTitle
112	The GodFather
223	Pulp Fiction

movieId	companyName
112	Paramount
112	Alfran Productions
223	Miramax
223	Jersey Films

Figure 1. Components of the optimization & execution module

Besides trying to comply with all Cassandra restrictions the planner identifies which join algorithms should be used for executing the various joins by comparing the cost of left-deep trees. Currently two join algorithms have been implemented: a) a variation of Index-Nested Loops taking advantage of the existing indexes and additionally allowing joins over collection sets – indexed collections of elements (maps, sets and lists) supported after the Cassandra version 2.1b) the sort-merge join allowing the join to be implemented in one pass over the data when the joined relations are indexed. When joining two column-families, if only one of them has an index on the joined field, the optimizer reads all rows from the non-indexed one and then uses the index for searching the indexed column-family. On the other hand when both column-families are sorted on the join column, the Sort-Merge join algorithm would be faster and is preferred by the optimizer.

c) Combiner: This component executes the queries, calculates the join using the selected algorithm and returns the results to the users.

4. Evaluation

The first step we had to do it was to create data for the benchmark. For this we used the BSBM[16]. The Berlin SPARQL Benchmark (BSBM) defines a suite of benchmarks for comparing the performance of these systems across architectures. The benchmark model is built around an e-commerce use case in which a set of products is offered by different vendors and consumers have posted reviews about products. The benchmark query mix illustrates the search and navigation pattern of a consumer looking for a product. In our example we use these arguments: -fc -pc 1000 -s Sql –fn mydata. Where –s is the output format , -pc number of products , -fn for the file name of the generated dataset.
So my output format is Sql and the files are created in the mydata are 20mb size. These Sql files are converted to Cql to run in the Cassandra Cluster.

One of the biggest problems we had was to convert the Sql files to Cql because Cassandra doesn't support Sql. After research we decided to use the sqoop tool. Sqoop is a tool designed for efficiently transferring bulk data between Hadoop and structed datastores such as relational databases. One of these options was to transfer data from MySql to Cql. We installed DataStax Enterprise 5.0 that contains both Cassandra, Hadoop and Sqoop. There are not a lot of documentation of how to run the Sqoop just an a demo[17]. The demo runs Sql commands to put the data from a CSV file or MySql file into a MySql table in a MySql database. You then import the Sql data from MySql to a Cql table in Cassandra. The demo exports the data from MySql, and then uses a subset of Sqoop commands to import the data into a Cql table. To import data to Cql, the keyspace and Cql table must exist prior to the importation. If the Cql table contains data prior to the importation, Cql-import updates the data. The problem with Sqoop was the import options that they were needed. . Depending on the job you want to do the import options should change also. In every machine, hostname, username, passwords, tables, keyspaces, mapping, etc, import options should change accordingly. The Sqoop needs a jdbc driver installed in a specific folder else Sqoop doesn't run. Also the Cassandra host name that can be found if in the command line we type "hostname –f" . Our import options and our data can be found if needed in the link below :
https://www.dropbox.com/sh/ra4d8xbv9urlh8n/AABDa605UBooqF-tcr61cGrca?dl=0
Given below is the import options for Sqoop demo too:

import.options file

Contents	Description
cql-import	Perform an import operation.
--table	A SQL table name follows this option.
npa_nxx	SQL table name for the demo.
--cassandra-keyspace	A keyspace name follows this option.
npa_nxx	The keyspace name for the demo.
--cassandra-table	A Cassandra table name follows this option.
npa_nxx_data	The Cassandra table name for the demo.
--cassandra-column-mapping	A CQL:SQL column mapping follows this option.
npa:npa,nxx:nxx,latitude:lat,longitude:lon,state:state,city:city	The Cassandra column names:corresponding MySQL column names, cql1:sql1,cql2:sql2, . . .
--connect	The JDBC connection string follows this option.
jdbc:mysql://<mysql_host>/npa_nxx_demo	The JDBC connection string.
--username	A MySQL user name follows this option.
<mysql_user>	The user name you configured as the MySQL admin.
--password	A MySQL password follows this option.
<mysql_password>	The MySQL administrative password you configured.
--cassandra-host	The IP address of the MySQL host node follows this option.
<cassandra_host>	The IP address of the host node. For example, 127.0.0.1. A fully-qualified domain name if using Kerberos.

The next problem was to change-update the code that was creating the joins in java code. Some changes were mostly about the libraries that had to be updated because they were older version of Datastax Enterprise 5.0 and creating problems and some code that changed because the com.datastax.driver.core.Date; was not existing anymore

For example in some functions:

```
ClusterConn.getMetadata().getKeyspace(KeySpace).getTable(tableName).getColumn(columnName).
getIndex()!=null
```

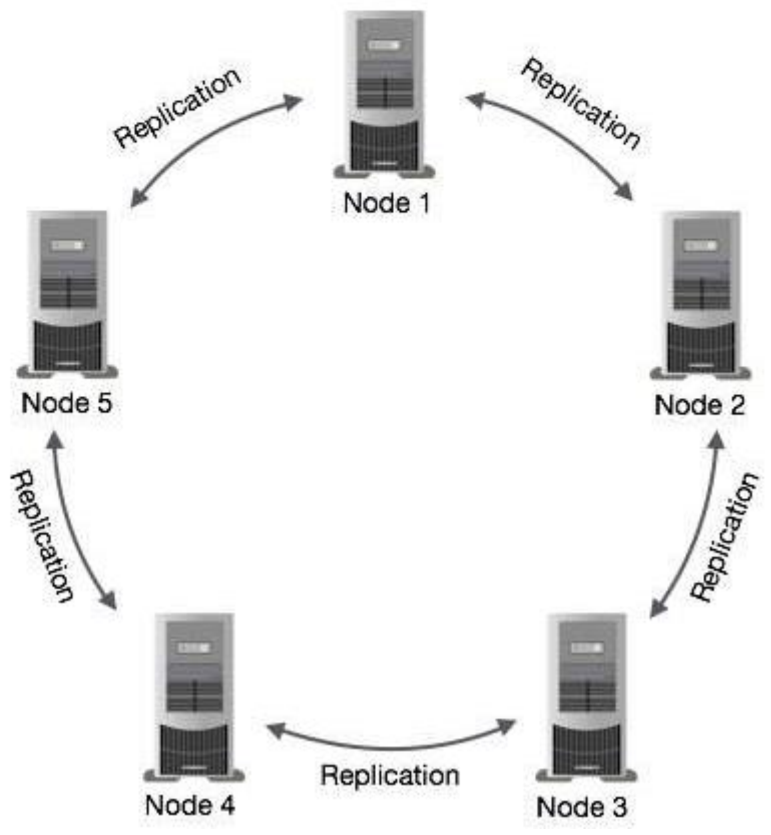
Changed to:

```
ClusterConn.getMetadata().getKeyspace(KeySpace).getTable(tableName).getColumn(columnName).
getName()!=null
```

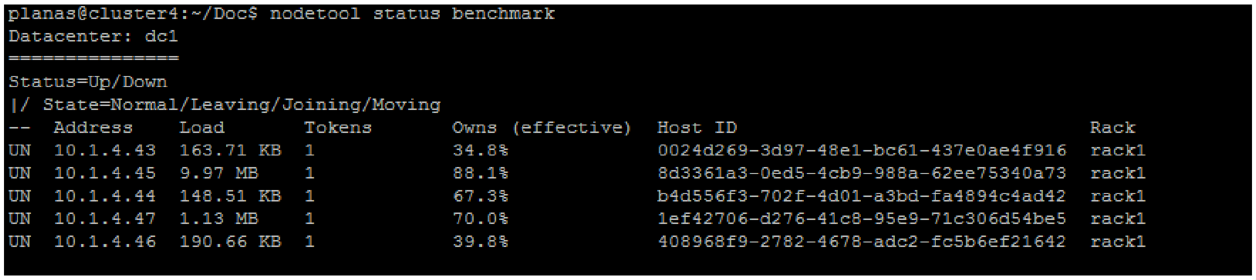
Afterwards our move was to create the Cluster of Cassandra.

Cassandra database is distributed over several machines that operate together. The outermost container is known as the Cluster. For failure handling, every node contains a replica, and in case of a failure, the replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

The following figure shows a schematic view of how Cassandra uses data replication among the nodes in a cluster to ensure no single point of failure.



Cassandra uses the **Gossip Protocol** in the background to allow the nodes to communicate with each other and detect any faulty nodes in the cluster.
Also every replica owns a percentage of the data and depending on the query, Cassandra takes the data that needs from the replica that will do it faster. The figure below demonstrates these percentages from a 5 node Cluster.



```
planas@cluster4:~/Doc$ nodetool status benchmark
Datacenter: dc1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.1.4.43  163.71 KB  1            34.8%             0024d269-3d97-48e1-bc61-437e0ae4f916  rack1
UN  10.1.4.45  9.97 MB    1            88.1%             8d3361a3-0ed5-4cb9-988a-62ee75340a73  rack1
UN  10.1.4.44  148.51 KB  1            67.3%             b4d556f3-702f-4d01-a3bd-fa4894c4ad42  rack1
UN  10.1.4.47  1.13 MB    1            70.0%             1ef42706-d276-41c8-95e9-71c306d54be5  rack1
UN  10.1.4.46  190.66 KB  1            39.8%             408968f9-2782-4678-adc2-fc5b6ef21642  rack1
```

Our Evaluation started with selecting the queries. For the evaluation the queries that were used was the BSBM queries[14]. Some attributes or in specific cases such as nested select, don't exist in Cassandra. So in these cases these queries didn't used for our tests.

Here we have a table that shows which queries we runned and which we didn't:

BSBM Queries		
IN / NOT IN	Query 1 , Query 3 , Query 4	Not supported by Cassandra
Joins	Query 2 , Query 8 , Query 9 ,Query 10,Query 12	Works
Nested Select	Query 5 , Query 7	Not supported by Cassandra
Like	Query 6	Not supported by Cassandra
Simple Query	Query 11	Works

The queries that have join of three tables, the only way to do it is to run the join of two and then the result will do join again with the third. This is expensive but the only way. Also a big issue in the queries was the primary key. Its very important to use the correct primary key because Cassandra depending on that partitions the data. Even though that the primary key was very important, in the BSBM queries the key was specified. This make the things more complex. When we were running the tests with the primary keys as specified a specific error message was received :

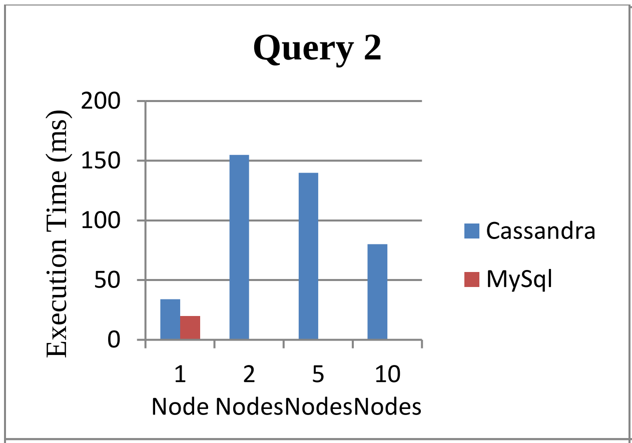
Bad Request: Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING.

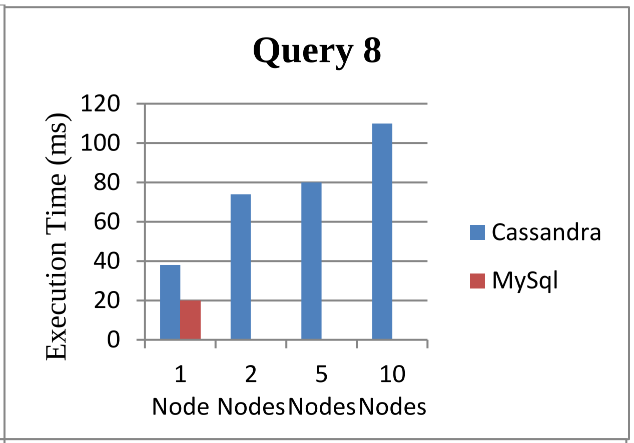
Cassandra knows that it might not be able to execute the query in an efficient way. It is therefore warning you: “Be careful. Executing this query as such might not be a good idea as it can use a lot of your computing resources”.
The only way Cassandra can execute this query is by retrieving all the rows from the table blogs and then by **filtering** out the ones which do not have the requested value for the requested column.
If your table contains for example a 1 million rows and 95% of them have the requested value from a column, the query will still be relatively efficient and you should use ALLOW FILTERING.
On the other hand, if your table contains 1 million rows and only 2 rows contain the requested value from the column, your query is extremely inefficient. Cassandra will load 999, 998 rows for nothing. If the query is often used, it is probably better to add an index on the specific column.
Unfortunately, Cassandra has no way to differentiate between the 2 cases above as they are depending on the data distribution of the table. Cassandra is therefore warning you and relying on you to make the good choice.
One solution was to add a secondary index in this specific column that requests a value that needs allow filtering. In this situation Cassandra will return all the blogs that have this value and will not request ALLOW FILTERING. This is due to the fact that Cassandra can use the secondary index on the column to find the matching rows and does not need to perform any filtering. Although that it’s a good solution it doesn’t work in all the situations. In the case that a value is asked from a column that is a secondary index and then another value is asked Cassandra will request allow filtering. Because it will have to first find and load the rows containing the first value asked, and then to filter out the ones which do not have the column equal to the second value.
The smartest choice in this situations and the only solution is to think your data, your model and what you are trying to do and depending on that make the correct primary keys in the queries that is needed which is the thing we did. We used in specific queries indexes and in the other queries we created a new table with a new primary key and we put the old data of the table there.

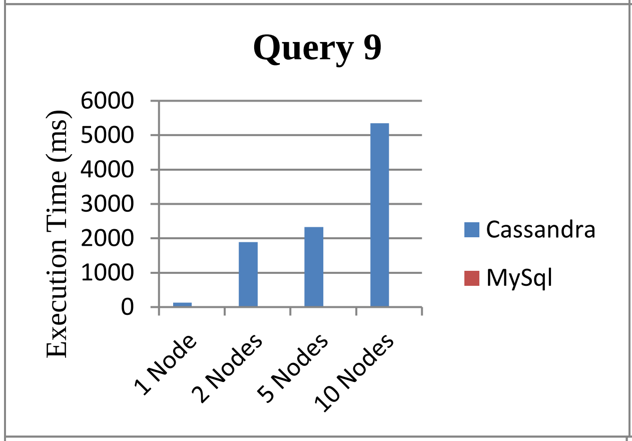
The rest of the queries that had a simple join tested by running the join function. The join functions ,that is a function from CassandraJoins[15] , joins two tables and creates a new table with the result columns. While running this function an algorithm that time measures is running simultaneously. Also all these queries were tested in multiple nodes and each time was the average after three runs. So from these tests the times measures that we got is given in the table below:

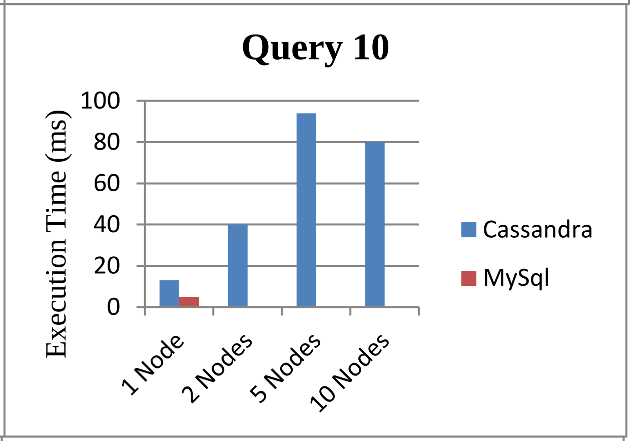
Cassandra	Node 1	Node 2	Node 5	Node 10
Query 2	34 msec	155 msec	140 msec	80 msec
Query 8	38 msec	74 msec	80 msec	110 msec
Query 9	27 msec	1 sec 890 msec	2 sec 330 msec	5 sec 350 msec
Query 10	13 msec	40 msec	94 msec	80 msec
Query 11	5 msec	4 msec	3 msec	3 msec
Query 12	39 msec	32 msec	25 msec	80 msec

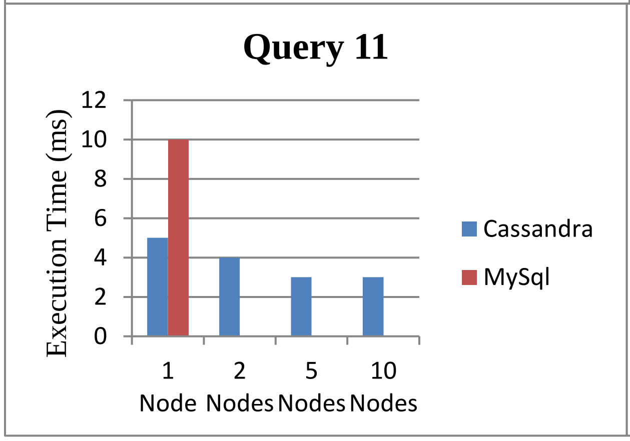
MySql	Node 1
Query 2	20 msec
Query 8	20 msec
Query 9	20 msec
Query 10	5 msec
Query 11	10 msec
Query 12	10 msec

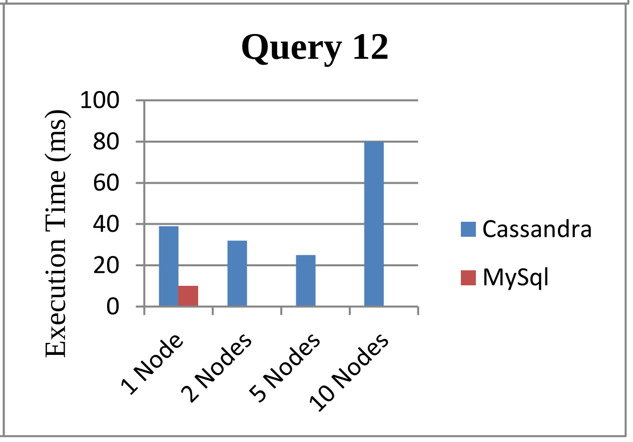












5. Conclusion

In this thesis my job was to create data for benchmarking and run time measurements with the help of a Java API to perform joins called CassandraJoins[15]. The queries for these tests were taken by BSBM[5] in Sql translated to Cql with the help of Sqoop[6] . Then a cluster with multiple nodes initialized and deployed from whom we tested and time-measured the queries from BSBM by using the CassandraJoinsApi. The future work that must been done is to integrate joins in Cql because adding joins directly to Cql will allow join operations to run faster than running through an added layer. Also the next step is to implement additional joins algorithms such as the Hash Joins algorithm that Cassandra already is doing hashing on the partition keys, so Hash join could be fast in many cases and easy to implement.

Joins will be able to run even more fast in future Cassandra versions if a graph database[7] is added. If you understand what a graph database can do, then you know that one thing it does very well is handle the traversal of multiple relationships between vertices (entities in an RDBMS world) without any need to create indexes or materialized views to overcome join performance inefficiencies in an RDBMS.In short, a graph database represents the ultimate in joins where ease of use and performance are concerned.

6. References

[1] https://en.wikipedia.org/wiki/NoSQL

[2] https://hbase.apache.org/

[3] https://www.mongodb.com/

[4] http://cassandra.apache.org/

[5] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/BenchmarkRules/index.html%23datagenerator

[6] http://sqoop.apache.org/

[7] http://www.datastax.com/2015/03/how-to-do-joins-in-apache-cassandra-and-datastax-enterprise

[8] http://spark.apache.org/

[9] http://www.simba.com/drivers/cassandra-odbc-jdbc/

[10] https://www.mongodb.com/nosql-explained

[11] http://www.sql-join.com/

[12] https://www.tutorialspoint.com/cassandra/cassandra_introduction.htm

[13]https://www.tutorialspoint.com/cassandra/cassandra_architecture.htm

[14] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/20110607/ExploreUseCase/index.html - queriesRelational

[15]https://www.dropbox.com/sh/b3x9f50loh7ty0t/AAByW0jrGph0IFIpMO6IsZ8Pa?dl=0

[16]http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/BenchmarkRules/index.html%23datagenerator

[17] https://docs.datastax.com/en/datastax_enterprise/5.0/datastax_enterprise/ana/sqoopDemo.html