Practical evaluation of NoSQL/NewSQL systems in the environment emulating interactive social networking actions

Final Project Report

Submitted as partial fulfillment of the requirements towards an

M.Sc. degree in Computer Science

The Open University of Israel

Computer Science Division

By

Yuri Gurevich

Prepared under the supervision of Prof. Ehud Gudes

February 2015

Contents

Abstract

Cassandra is an open source distributed database system that is designed for storing and managing large amount of data across commodity servers [2]. Cassandra is considered to be NoSQL data store and according to data model classification it belongs to column family category. As most of the modern NoSQL databases, Cassandra doesn't provide ACID guarantees and is not based on the traditional relational model.

The NuoDB database is SQL compliant and has been called 'NewSQL' [3]. NewSQL databases are considered to be fast enough to handle big data challenges, they are horizontally scalable as NoSQL databases, although they initially designed to provide ACID [4].

BG is a benchmarking system that rates data store by processing interactive social networking actions [9]. In this project we implemented clients for Cassandra and NuoDB so that we will be able to evaluate these two data stores using BG benchmarking system.

Introduction

Nowadays we're witnessing a huge growth of social networking applications. Sites like Facebook, LinkedIn and Twitter provide various types of person to person communication to millions of users.

This enormous amount of data related to social networking usually poses new challenges for data storage and processing and demands a different approach to building data stores that will be able to handle such "big data". The term big data was first introduced in early 2000 and was referring to the following 3 major components: Big Variety, Big Velocity, and Big Volume [1]. As a result of this new requirement NoSQL movement started to develop in the last decade or two.

Initially the abbreviation NoSQL ("not only SQL") was used to describe type of data store which is not ACID compliant and does not fully support SQL language as its primary interface.

In the research seminar "Database Systems" [Appendix A] we have surveyed number of scientific articles describing NoSQL principles as well as the criticism of NoSQL movement [19]. In addition to that in the seminar report we have also presented an analysis of the architecture of Apache Cassandra and various techniques which enabled this distributed data store efficiency. Cassandra is an open source, peer-to-peer key-value data store system that can scale over thousands of nodes while providing highly available service without a single point of failure [3]. This NoSQL data store is very scalable, fast and applicable for particular applications, mainly for those that involve huge amount of non-structured data and allow some degree of "inconsistency" or

eventual consistency. Apart from Facebook, Cassandra is now widely used by a range of other most advanced Internet companies including: Twitter, Digg and Formspring.

Another topic which was analyzed in the seminar was a definition of the NewSQL databases [Appendix A]. This type of data bases started to appear just recently and it's an answer to an increasing demand of OLTP applications that deal with "big data", but along with that require ACID. NewSQL is a class of modern relational database management that provides the same scalable performance of NoSQL systems for online transaction processing (read-write) workloads while maintaining the ACID guarantees of a traditional database system [5]. In the seminar we have surveyed VoltDB which is one example for a NewSQL RDMBS aimed to fit the modern high velocity applications [6].

In this project we're going to analyze the architecture and performance of another member of NewSQL family – NuoDB. Main characteristics of this DB is that it's a distributed database designed with global application challenges in mind. Besides that, it is a true SQL service: it supports ACID transactions, SQL language and relational logics. It was designed from the beginning to enable cloud based scalability and resilience with no single point of failure [4].

With this variety of modern databases models comes a question: what architecture gives better results in terms of performance, scalability and what the tradeoff associated with these new designs for different workloads is? Which DB type works better for loading big data, is it a NoSQL or a NewSQL? And, how these data stores that implement a different data model actually compare when it comes to social networking?

Although there is still no particular common way to evaluate performance, estimate actual capabilities or compare those different data stores, there are many different benchmark utilities that can be used for data base evaluation.

Probably most widely known utility that was used for DB evaluation is TPC-C [11]. However, this benchmarking system was initially designed to estimate database performance for transaction processing, which makes it less valuable for other environments evaluation.

In 2010 a couple of researchers from Yahoo! came with an idea of Yahoo Cloud Serving Benchmark (YCSB) project. The goal of YCSB was to develop a framework and common set of workloads for evaluating the performance of different "key-value" and "cloud" serving stores [12]. The Figure 1 describes the high-level design of YCSB benchmark system.

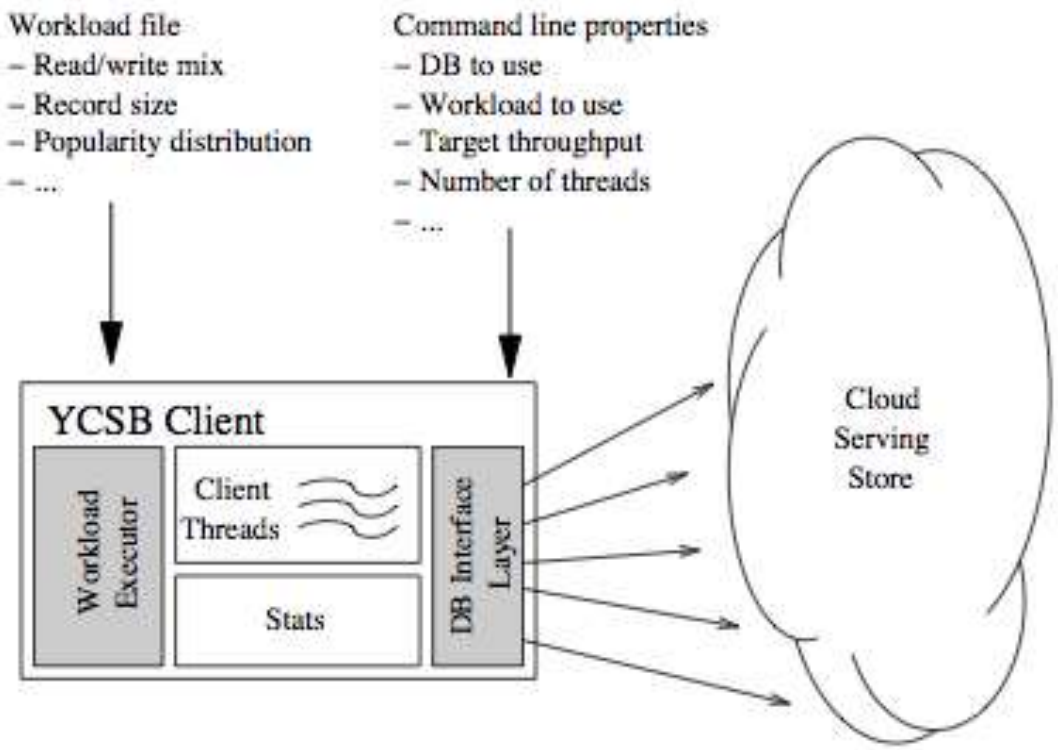

Figure 1: YCSB high-level design

In this project, we have evaluated Cassandra and NuoDB using a social networking benchmark named BG [7, 8, and 9]. BG has been inspired by prior benchmarks that evaluate cloud services such as YCSB [12] and YCSB++ [13], e-commerce sites, and object-oriented [10] and transaction processing systems [11]. We selected BG because of the conceptual data model (see Figure 2) which is more complex than that of the YCSB and YCSB++. Besides the schema, BG main contributions are two folds. First, it emphasizes interactive social actions that retrieve a small amount of data. Second, it promotes the amount of unpredictable data produced by a solution as a first class metric for comparing different data stores with one another [9].

BG populates data base with a social graph consisting of a fixed number of members, friends per member, and resources per member. It consists of social networking actions such as extend a friendship invitation to a member and view a member profile. See

Table 1 for a list of the BG actions considered in this study. We use BG to establish two important metrics:

1. Database population time, i.e. measure the time to fully load BG database including 10000 members with 100 friends per user and 100 resources per user. Load is done using 10 loading threads.
2. Social Action Rating (SoAR): This ratings compute the number of concurrent actions performed by a system when a fixed percentage of requests (say 98%) observe a latency equal to or lower than a pre-specified threshold (say 100 msec) with the amount of unpredictable data less than a fixed threshold (say 0.01%) for some fixed duration of time (say 10 minutes). The values in the parenthesis are inputs to BG. BG's output is the SoAR rating of its target data store [7]

To the best of our knowledge, this is the first study to evaluate Cassandra and NuoDB for processing interactive social networking actions. An interesting outcome of this investigation is the observation that NuoDB works much faster during the initial Social graph data load (data population), while Cassandra has shown a better performance during an experiment emulating interactive social actions.

Data Model

Figure 2 shows the conceptual design of BG's social graph used for this evaluation. See [8, 9, and 10] for a comprehensive description of BG schema. The Members entity set contains those users with a registered profile. It consists of a unique identifier and a fixed number of string attributes. One may configure BG to create a social graph with or without images. In this paper, we consider both possibilities. With images, all experimental results are obtained using a social graph configured with a 2 KB thumbnail image and a 12 KB profile image. Thumbnail images are displayed when listing friends of a member and the higher resolution profile image is displayed when a member visits a profile. A member may extend a friend invitation to another member or be friends with a member, represented using "Invite" and "Friend" relationship sets, respectively [8, 9]. A resource may pertain to an image, a posted question, a technical manuscript, etc. These entities are captured in one set named "Resources". In order for a resource to exist, a member must "Own". A member may post a resource, say an image, on the profile of another member, represented as a "Posted on" relationship between two members and a resource. A member may comment on a resource. This is implemented using the "Manipulation" relationship set.

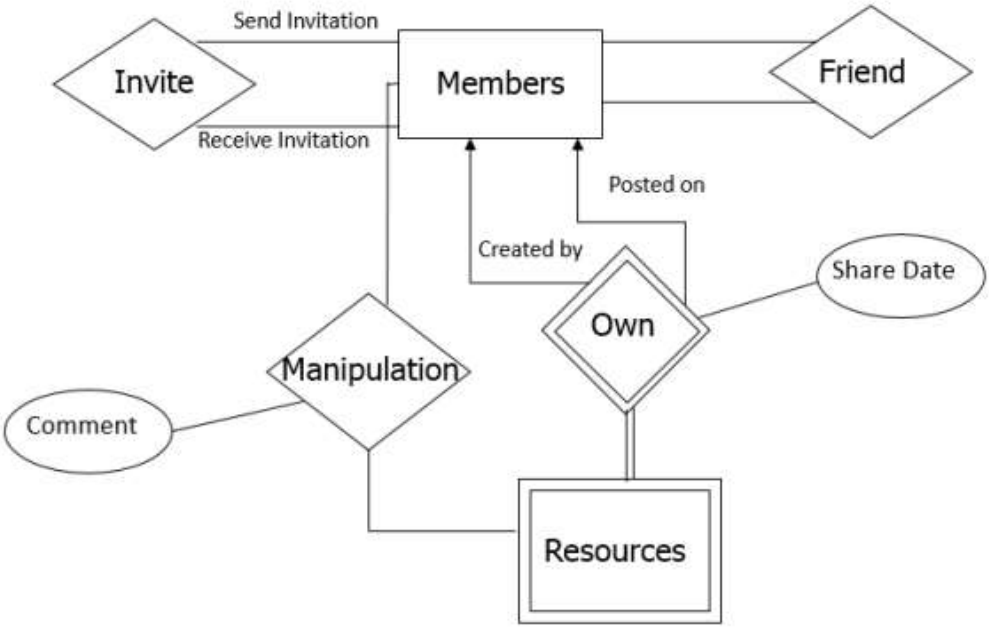

Figure 2: Conceptual data model of BG database

In the following chapters we describe our implementation of BG data model schema in Cassandra and NuoDB data stores.

Cassandra Schema Description

- Data is stored in the following 3 Column Families:
    - Members
    - Resource
    - Manipulation
- Profile and thumbnail images for a user are stored as Blobs (Binary Large OBjectS) columns within Members column family.

- PengingRequests and ConfirmedFriendship are stored as a collection-type columns within Members column family. Those columns contain sets of user ids of the corresponding users.
- We decided to implement Manipulation as a separate Column Family and not as a collection-type column within Resources, because of the following 2 reasons:
    - The length of the collection-type field is limited (CQL3 limitation is 64K)
    - In Cassandra 1.2 they didn't support non primitive type as a member of collection column. According to Datastax: it might be possible to support such functionality in the future if that is deemed useful, but it will require additional work [20]
- Primary Keys
    - "Members" primary key is userid
    - "Manipulation" primary key is manipulationid (mid)
    - "Resource" has a compound primary key of (walluserid, rid)
- Indexes
    - The "Resource" column family is also indexed on creatorid
    - The "Manipulation" column family is also indexed on resourceid (rid)

NuoDB Schema Description

- Data is stored in the following tables:
    - users
    - friendship
    - resources
    - manipulation
- Friendship is stored in one table, "friendship", as two records for every confirmed friendship.
- Aggregates are calculated by issuing aggregate queries.
- Profile and thumbnail images for a user are stored in the file system.
- Indexes
    - The "users" table is indexed on userid.
    - The "friendship" table is index on friend1 and friend2.
    - The "resources" table is indexed on resourceid and walluserid (the userid for the wall, where the resource is posted on).
    - The "manipulation" table is indexed on manipulationid and resourceid.

Actions

In this section we provide the list description of Social Actions defied in the BG system [9].

Actions defined in BG can be easily mapped into particular actions in most popular Social Networking Systems, see Table 1.

View Profile (VP)	✓	✓	✓	✓	✓
List Friends (LF)	✓	✓	✓	✓	✓
View Friend Requests (VFR)	✓	⊖	⊖	✓	⊖
Invite Friend (IF)	✓	Add to Circle	Follow	✓	Subscribe
Accept Friend Request (AFR)	✓	⊖	⊖	✓	⊖
Reject Friend Request (RFR)	✓	⊖	⊖	✓	⊖
Thaw Friendship (TF)	✓	Remove from Circle	Unfollow	✓	Unsubscribe
View Top-K Resources (VTR)	✓	✓	✓	✓	✓
View Comments on a Resource (VCR)	✓	✓	✓	✓	✓
Post Comment on a Resource (PCR)	✓	✓	Reply to a Tweet	Send a Recommendation	✓
Delete Comment on a Resource (DCR)	✓	✓	Delete Reply to a Tweet	Withdraw Recommendation	✓

Table 1: Social Actions

Here we present the implementation details of all the required BG social actions for Cassandra and NuoDB (please see Appendix C for an example of an implementation code of one of the Social Actions). The summary of actions and the number of implementation steps for every social action can be found in Table 2 below.

View Profile

The VP action emulates a member (socialite) visiting the profile of either himself or another member. Its input include the socialite's id and the id of the referenced member, B. A socialite's id may equal B, emulating a socialite referencing his own profile. The output of VP is the profile information of B, including B's attribute values and the following two aggregate information: B's number of friends and number of resources. If the socialite is referencing his own profile (socialite id equals B's id) then VP retrieves a third aggregate, B's number of pending friend invitations [9].

CQL Cassandra client implements View Profile as following:

First we get the value of the column representing ConfirmedFriends and PendingRequests for the referenced member B. Those are collection–type columns, thus by getting the size of those collections we will receive the number of confirmed friends and the number of pending friend's requests for the member correspondingly. It's worth to note that the number of pending friend requests will be returned to the calling function only if the socialite is referencing his own profile (socialite id equals B's id).

Second we obtain the number of rows in column family Resources, which belong to the referenced user (according to walluserid). This aggregation operation is enabled in

CQL3, because we defined walluserid column to be a part of PK (primary key) on the given column family (CF).

Third we get all the values of the referenced member from Members CF and push them to the result hashmap.

NuoDB client implements VP by performing the following queries using NuoDB JDBC clients:

- An SQL query with an exact match selection predicate referencing each attribute of inviterID or inviteeID of the Friendship table, where status is “ConfirmedFriendship”. In NuoDB schema Friendship is represented as one row, that’s why we use “or” clause.
- An SQL query with an exact-match selection predicate referencing inviteeID attribute of the Friendship table, where status is equal “PendingRequest status”.
  This last query is issued only when the socialite id equals B’s id, because only if owner viewing his own profile, she can view his pending friend requests.
- An aggregate count SQL query with an exact match selection predicate referencing the wallUserID of the Resource table for getting the number of resources.
  - An SQL query with an exact match selection predicate referencing the userid of the User table to retrieve the member attribute values to populate the required fields of the referenced user.

List Friend

The LF action emulates a socialite A viewing member B's list of friends. Similar to the discussion of VP, B may equal A to emulate the socialite viewing his own list of friends. LF retrieves the profile information of each friend including their thumbnail image and excluding their profile image.

Using Cassandra CQL we implement LF in two steps.

First we retrieve the value of the column representing ConfirmedFriends for the referenced member B and use that value to populate a collection of type set of strings. Second, we iterate over the set of friends we just obtained and for every "userID" we get all the values of the referenced member from Members CF and push them to the result hashmap.

NuoDB client implements LP by performing just one single SQL query which joins 2 relationships: Members and Friendship and retrieves all the tuples from Members that have ConfirmedFriendship status with the profileOwnerID in the Friendship table.

View Friend Request

The VFR action emulates a Socialite A retrieving his pending friend invitations. This retrieves the profile of each member extending the profile information with his thumbnail (if configured with images).

Cassandra CQL: the implementation of the View Friend Requests is very similar to List Friends.

The only difference is that in first step we retrieve the value on PendingRequests columns. Further steps are the same as above.

NuoDB JDBC client implements VFR by performing just one single SQL query which joins Members and Friendship and retrieves all the tuples from Members that have PendingRequest status with the profileOwnerID in the Friendship table.

Invite Friend

The IF action emulates a Socialite A sending an invitation to another member of the social network to become his “friend”.

Cassandra client implementation just updates the PendingRequests column of the corresponding inviteeID to include inviterID in it. This column has been defined as a collection-type column of type set, therefore we would not have duplicate IDs within that field.

NuoDB JDBC implementation of IF consists of an insert statement where a new record is added into Friendship relation with values of inviteeID, inviterID and status equals to PendingRequest.

Accept Friend Request

When a Socialite A accepts the pending friend invitation of another member B.

Cassandra CQL implementation contains of 2 steps:

First we remove the corresponding entry to B (inviter) from the PendingRequests column object of the user A (invitee).

Next, we invoke 2 updates: add user A to the ConfirmedFriendship column of B and symmetrically add user B to the ConfirmedFriendship column of A.

NuoDB client implementation of AF consists of a single update statement, where the status of corresponding record within Friendship relation is updated from PendingRequest to be ConfirmedFrienship.

Reject Friend request

When Socialite A rejects friend invitation from B.

Cassandra CQL implementation just removes the corresponding entry to B (inviter) from the PendingRequests column object of the user A (invitee).

NuoDB implementation consists of a single deletion: we remove from Friendship the entry, where inviter is A and invitee is B and the status is PendingRequest.

Thaw Friendship

This action emulates member A removes friendship with member B.

Cassandra CQL implementation just updates the ConfirmedFriends columns of both members A and B and removes from those sets B and A correspondingly.

NuoDB implementation of this action consists of just single delete SQL statement: we remove both entries from Friendship, where inviter is A and invitee is B and vice versa and the status is ConfirmedFriendship.

View Top-K Resource

This action enables socialite to view top K resources that were posted on his wall. The definition of “top” and the value of K are both configurable.

Cassandra client implementation of VTR retrieves all the rows from the Resource column family where walluserid is the ID of the required user. We fetch this data ordered by resource ID, it’s possible in this case using CQL, because the corresponding field “rid” has been defined as a 2nd field in the primary key of Resource CF (otherwise CQL wouldn’t allow using of “ORDER BY” clause).

We also limit the result set by number of K, while due to current limitation of the CQL3 [16] we couldn't use prepared statement and it was done simply by building query string (concatenation). The final CQL query syntax for this action looks very similar to SQL. NuoDB client implements this action by a single select * query with an exact-match selection predicate referencing profileOwnerID. To get just the "top" K values we use NuoDB-specific SQL syntax: "FETCH FIRST".

View Comment on a Resource

Using this action member A display comments posted on an resource with a unique rid R.

When user A invokes VCR on a resource with rid R, Cassandra client retrieves all the rows from the Manipulation CF where rid equals R. This query is legitimate in CQL3, since we have defined resourceIndex – a secondary index on the Manipulation CF.

NuoDB implementation of this action contains of a single SQL with an exact-match selection predicate referencing rid attribute of the Manipulation table.

Post Comment on Resource

This action emulates member A posting the comment on a resource with specific resource id (rid).

Cassandra client implements this action by insertions of a new row into Manipulation column family. The row is identified by given rid and manipulation id.

NuoDB implementation of this action contains of a single SQL insert statement where a new record is added into Manipulation relation with values of rid and manipulation id.

Delete Comment on Resource

This action emulates member A deleting the comment posted on a resource with specific resource id (rid).

Cassandra client implements this action by simple deletion of the row from Manipulation column family. The row is identified by given rid and manipulation id.

NuoDB client implementation also contains single delete statement: record with given resource id and manipulation id is to be deleted.

Syntactically SQL queries corresponding to the recent two actions are almost identical to the CQL statements for those two actions respectively.

	Cassandra Client		NuoDB Client	
Social Actions	**Implementation Description**	**Number of steps**	**Implementation Description**	**Number of steps**
View Profile (VP)	Get ConfirmedFriends and PendingRequests collections Get count of Resources Get member from Members CF	3	Get ConfirmedFriendship from Friendship get PendingRequest from Friendship Get count of Resource Get User attributes	4
List Friends (LF)	Get ConfirmedFriends collection Iterate over the collection and for every entry get values of member from Members CF	2	Join Members and Frienship and retrieve tuples with ConfirmedFriendship and corresponding ID	1
View Friend Requests (VFR)	Get PendingRequests collection Iterate over the collection and for every entry get values of member from Members CF	3	Join Members and Frienship and retrieve tuples with PendingRequest and corresponding ID	1
Invite Friend (IF)	Update PendingRequests to include inviterID	1	Insert into Friendship with values of inviteeID, inviterID, PendingRequest.	1
Accept Friend Request (AFR)	Remove B from the PendingRequests of A Add user A to the ConfirmedFriendship column of B add user B to the ConfirmedFriendship column of A	3	Update status in Friendship from PendingRequest to ConfirmedFrienship.	1
Reject Friend Request (RFR)	Remove B from the PendingRequests of A	1	Delete from Friendship where inviter is A and invitee is B and status is PendingRequest.	1
Thaw Friendship (TF)	Remove A from ConfirmedFriends of B remove B from ConfirmedFriends of A	2	delete from Friendship, where inviter is A and invitee is B and status ConfirmedFriendship	1
View Top-K Resources (VTR)	Retrieve all from Resource where walluserid is the given ID	1	Get all from Resource for a given profileOwnerID	1
View Comments on a Resource (VCR)	Retrieve all from Manipulation where rid equals R	1	Get all from Manipulation for a given profileOwnerID	1
Post Comment on a Resource (PCR)	Add entry into Manipulation with given rid and manipulation id	1	Insert into Manipulation with values of rid and manipulation id	1
Delete Comment on a Resource (DCR)	Remove of from Manipulation by given rid and manipulation id	1	Delete of from Manipulation by given rid and manipulation id	1

Table 2: Actions Implementation Summary

Experiment description

AWS configuration

We have used Amazon Web Services [17] as a platform for conducting the experiments.

All the machines were running Linux Ubuntu Linux 12.10

Cassandra Setup

For Cassandra server we were using a cluster of 3 M3 instances that were running DataStax AMI for AWS.

NuoDB setup

We tried the NuoDB AMI available in AWS store, however it was not useful as it had some certificate expiration issue which prevented us from running nuodb-agent on that instance.

Therefore NuoDB server has been installed manually on M3 instance.

Client setup

Client has been executed on a separate instance.

BG set up consists of the following components:

- BG client including DB store proprietary client libraries
- BG Coordinator
- BG Listener

Experiment flow

- Make sure that the security group applied allows access to Cassandra broker port, NuoDB agent port and the range of ports that will be used by Listener processes.

- Prepare a configuration of Listeners and run each of them in a separate window.
- Prepare a configuration of the Coordinator, the following parameters have to be set-up:
- Start the coordinator
- Monitor logs
    - Final Round indicates the completion of experiment.
    - Please see Appendix C for actual log samples.

Configuration and data loading

Cassandra

For Cassandra experiment we used the following configuration:

- 3-nodes cluster with Cassandra Release Version: 2.1.2
- Cassandra Cluster Nodes: AWS m3.medium instances with the following characteristics:

Model	vCPU	Mem (GiB)	SSD Storage (GB)
m3.medium	1	3.75	1 x 4

Table 3: AWS medium instance

- DB Client: Datastax Cassandra Client (CQL3) version 2.1.1
- Client was installed on AWS m3.2xlarge instance with the following characteristics:

Model	vCPU	Mem (GiB)	SSD Storage (GB)
m3.2xlarge	8	30	2 x 80

Table 4: AWS extra-large instance

The data load has been conducted on the key space with the following configuration

- Replication Factor = 1 and durable writes = false
    - 10,000 members, 100 friends per user, 100 resources per user, no user images: 18 Minutes

    - 10,000 members, 100 friends per user, 100 resources per user, 12KB profile images and 2KB thumbnail images: 19 Minutes
- Replication Factor = 3 and durable writes = true
    - 10,000 members, 100 friends per user, 100 resources per user, no user images: 33 Minutes
    - 10,000 members, 100 friends per user, 100 resources per user, 12KB profile images and 2KB thumbnail images: 36 Minutes

NuoDB

Both NuoDB server (version 2.1.1) and the machine running the client were installed on AWS m3.medium instance (Please see Table 3).

The data load has been conducted on the data base with the following configuration

- 10,000 members, 100 friends per user, 100 resources per user, no user images: 5.5 Minutes
- 10,000 members, 100 friends per user, 100 resources per user, 12KB profile images and 2KB thumbnail images: 6 Minutes

Figure 3 displays data base loading time for the configurations described above.

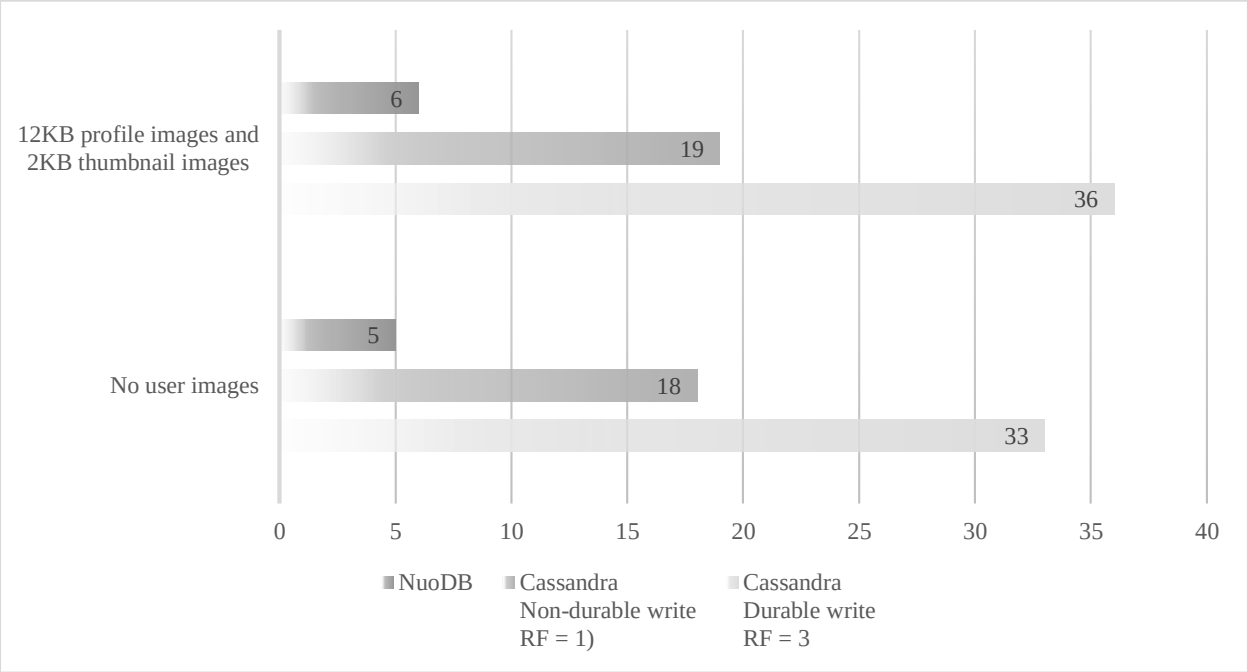

Figure 3: Database loading

Experiment results

In this section we present the SoAR rating obtained for Cassandra (NoSQL) and NuoDB (NewSQL).

During the experiment we used different mixes of Social Actions: Very Low Update, Low Update and High Update workloads.

BG Social Type Actions	Type	Very Low (0.1%) Write	Low (1%) Write	High (10%) Write
View Profile, VP	Read	40%	40%	35%
List Friends, LF	Read	5%	5%	5%
View Friend Requests, VFR	Read	5%	5%	5%
Invite Friend, IF	Write	0.04%	0.40%	4%
Accept Friend Request, AFR	Write	0.02%	0.20%	2%
Reject Friend Request, RFR	Write	0.02%	0.20%	2%
Thaw Friendship, TF	Write	0.02%	0.20%	2%
View Top-K Resources, VTR	Read	40%	40%	35%
View Comments on a Resource, VCR	Read	9.90%	9%	1%

Table 5: three mixes of social actions

To compare database loading time has been loaded with 10000 members: Figure 3 shows results of database loading time, while Figure 4 displays SoAR values obtained during experiments.

The definition of SoAR (“Social Actions Rating”) was given in the “Introduction” section, and here we just repeat that SoAR is the highest throughput (actions per second) of a data store that satisfies the pre-specified SLA (service level agreement) [9].

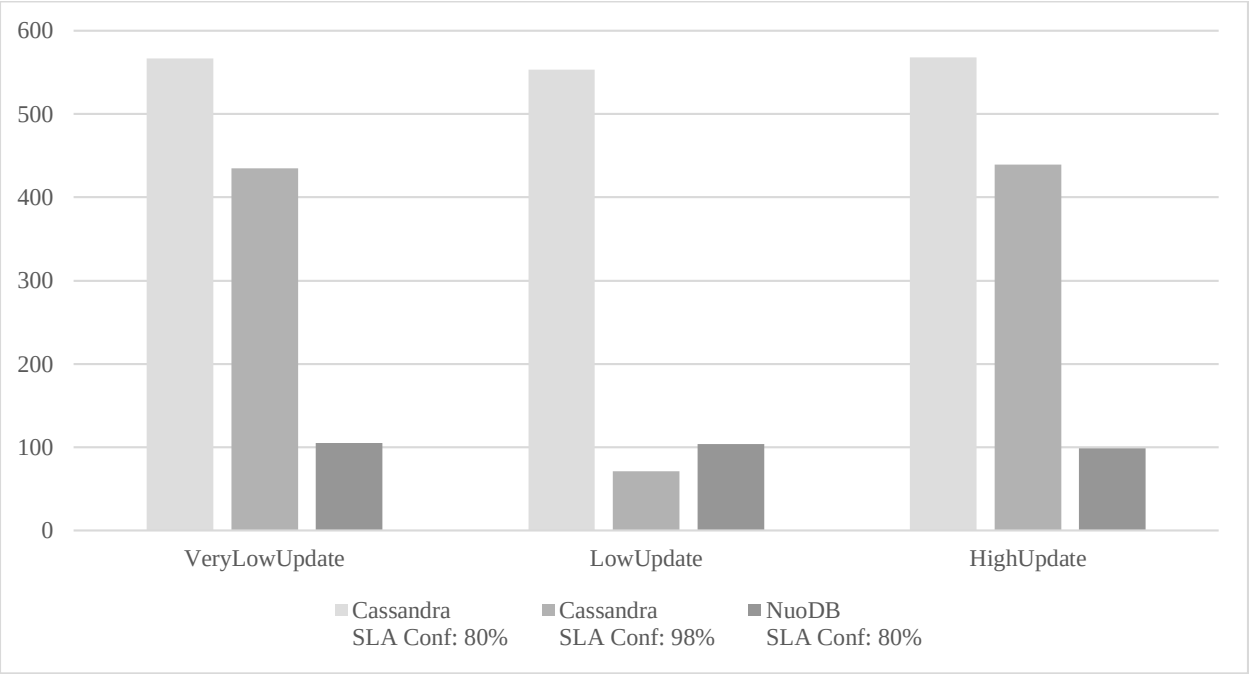

Figure 4. SoAR rating of databases using different workloads

Conclusion

This project evaluates the performance of Cassandra and NuoDB data stores by comparing them using the BG benchmark. We observed that NewSQL data store NuoDB showed better result during the population of the BG database, see Figure 3. However, Cassandra seems to be more efficient while running experiments using all kinds of BG social actions workload, e.g. VeryLowUpdate, LowUpdate and HighUpdate load, see Table 5. We have also noticed that the following factors have had a significant impact on Cassandra data load results: durability of writes and Replication Factor. With reduced RF (replication factor) and durability write turned off we obtained a 2 times increase in database population performance. Cassandra worked fine with both SLA Confidence level of 80 and 98 percent, while NuoDB, with the same amount of threads, couldn't handle 98% SLA (the SoAR for 98 was calculated as 0). Thus, we report NuoDB results only for SLA Confidence level of 80 percent. Removal of SLA confidence level with experiments on Cassandra has shown certain increase (25-39%) in the SoAR rating measures (see Figure 4).

Our future research plans are to prepare a comparative survey of NoSQL and NewSQL systems using the common taxonomy. We're going to compare data model, performance and use-cases for each type of data stores. The practical results of the current project will be used for database technology comparison and use-case determination. This work will be done as a part of the "Final Paper".

References

[1] Source: META Group. "3D Data Management: Controlling Data Volume, Velocity, and Variety." February 2001.

[2] Cassandra http://en.wikipedia.org/wiki/Apache_Cassandra

[3] NuoDB http://en.wikipedia.org/wiki/NuoDB

[4] NuoDB Technical Whitepaper http://go.nuodb.com/white-paper.html

[5] NewSQL http://en.wikipedia.org/wiki/NewSQL

[6] VoltDB Technical Overview and Whitepapers www.voldb.com/resources/datasheets

[7] BG Benchmark. http://bgbenchmark.org/

[8] S. Barahmand. Benchmarking Interactive Social Networking Actions, Ph.D. thesis, Computer Science Department, USC, 2014.

[9] S. Barahmand and S. Ghandeharizadeh. BG: A Benchmark to Evaluate Interactive Social Networking Actions. Proceedings of 2013 CIDR, January 2013.

[10] S. Barahmand and S. Ghandeharizadeh. Extensions of BG for Testing and Benchmarking Alternative Implementations of Feed Following. ACM SIGMOD Workshop on Reliable Data Services and Systems (RDSS), June 2014.

[11] Transaction Processing Performance Council, "TPC benchmark C standard spec. 5.11," Feb 2010, http://www.tpc.org/tpcc/spec/tpc-c_v5-11.pdf.

[12] Brian F. Cooper , Adam Silberstein , Erwin Tam , Raghu Ramakrishnan ,Russell Sears, Benchmarking cloud serving systems with YCSB, Proceedings of the 1st ACM symposium on Cloud computing, June 10-11, 2010, Indianapolis, Indiana, USA

[13] Swapnil Patil , Milo Polte , Kai Ren , Wittawat Tantisiriroj , Lin Xiao , Julio López , Garth Gibson , Adam Fuchs , Billie Rinaldi, YCSB++: benchmarking and performance debugging advanced features in scalable table stores, Proceedings of the 2nd ACM Symposium on Cloud Computing, p.1-14, October 26-28, 2011, Cascais, Portugal

[14] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux. OLTP-Bench: An extensible testbed for benchmarking relational databases. Proceedings of the VLDB Endowment, 7(4), 2013.

[15] Timothy G. Armstrong , Vamsi Ponnekanti , Dhruba Borthakur , Mark Callaghan, LinkBench: a database benchmark based on the Facebook social graph, Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, June 22-27, 2013, New York, New York, USA

[16] https://datastax-oss.atlassian.net/browse/JAVA-54

[17] Amazon Web Services http://aws.amazon.com/

[18] A.Lakshman, P. Malik., Cassandra: A Decentralized Structured Storage System, in ACM SIGOPS Operating Systems Review, Volume 44 Issue 2, April 2010

[19] Michael Stonebraker: SQL databases v. NoSQL databases. Commun. ACM 53(4): 10-11 (2010)

[20] DataStax http://www.datastax.com/documentation

Appendices

Appendix A:

Report for the seminar “Traditional SQL, NoSQL or NewSQL?” has been prepared within research-seminar course “Database Systems and Data Mining” (22953 – 2014a).

Appendix B:

Our Cassandra CQL client implementation has been kindly reviewed by Dr. Sumita Barahmand (the author of BG benchmark) and published the official BG website in the University of South California.

http://www.bgbenchmark.org/BG/CassandraClient.html

Appendix C:

Implementation of Accept Friend Request (AFR) Action.

Cassandra:

```
@Override
      public int acceptFriend(int inviterID, int inviteeID) {

            int retVal = SUCCESS;

            if (inviterID < 0 || inviteeID < 0)
                  return ERROR;

            int res1 = deletePendingRequest(inviterID, inviteeID);
            int res2 = CreateFriendship(inviterID, inviteeID);

            if ((ERROR == res1) || (ERROR == res2))
                  return ERROR;

            return retVal;

      }

      public int deletePendingRequest(int inviterID, int inviteeID) {

            int retVal = SUCCESS;

            PreparedStatement statement = null;
            BoundStatement boundStatement = null;

            if (inviterID < 0 || inviteeID < 0)
                  return ERROR;

            statement = getSession()
                        .prepare(
                                    "UPDATE Members SET pendingRequests =
pendingRequests - ? WHERE userid = ?");
            boundStatement = new BoundStatement(statement);

            Set<Integer> inviterIDset = new HashSet<Integer>(
                        Arrays.asList(inviterID));
            getSession().execute(boundStatement.bind(inviterIDset,
inviteeID));

            return retVal;
      }

@Override
      public int CreateFriendship(int friendid1, int friendid2) {

            int retVal = SUCCESS;
```

```
            PreparedStatement statement = null;
            BoundStatement boundStatement = null;

            if (friendid1 < 0 || friendid2 < 0)
                  return ERROR;

            statement = getSession()
                        .prepare(
                                    "UPDATE Members SET confirmedFriends =
confirmedFriends + ? WHERE userid = ?");
            boundStatement = new BoundStatement(statement);

            Set<Integer> friendid2set = new HashSet<Integer>(
                        Arrays.asList(friendid2));
            getSession().execute(boundStatement.bind(friendid2set,
friendid1));

            statement = getSession()
                        .prepare(
                                    "UPDATE Members SET confirmedFriends =
confirmedFriends + ? WHERE userid = ?");
            boundStatement = new BoundStatement(statement);

            Set<Integer> friendid1set = new HashSet<Integer>(
                        Arrays.asList(friendid1));
            getSession().execute(boundStatement.bind(friendid1set,
friendid2));

            return retVal;
      }
```

NuoDB Implementation:

```
@Override
      public int acceptFriend(int inviterID, int inviteeID) {

            int retVal = SUCCESS;
            if (inviterID < 0 || inviteeID < 0)
                  return -1;
            String query;
            query = "UPDATE bg_db.friendship SET status = 2 WHERE
inviterid=? and inviteeid= ? ";
            try {
                  // preparedStatement = conn.prepareStatement(query);
                  if ((preparedStatement =
newCachedStatements.get(ACCREQ_STMT)) == null) {
                        preparedStatement =
createAndCacheStatement(ACCREQ_STMT, query);
                  }
                  preparedStatement.setInt(1, inviterID);
                  preparedStatement.setInt(2, inviteeID);
```

```
            preparedStatement.executeUpdate();
        } catch (SQLException sx) {
            retVal = -2;
            sx.printStackTrace(System.out);
        } finally {
            try {
                if (preparedStatement != null)
                    preparedStatement.clearParameters();
                // preparedStatement.close();
            } catch (SQLException e) {
                retVal = -2;
                e.printStackTrace(System.out);
            }
        }
        return retVal;
}
```

Appendix D:

BG experiments results examples

Cassandra (SoAR – VeryLowUpdate)

2015/01/26 02:28:17 , BGconfigFile, /home/ubuntu/_rating/cassandra_config_veryLowUpdate_SoAR.txt
2015/01/26 02:28:17 ,time ,Objective, SLA Latency, SLA Confidence, numClients, ThreadCount, Throughput, ActThroughput, totalStaleness, TotalClientsSucceeded
2015/01/26 02:46:23 ,806596,InitialSOAR,0.1,98,4,10,435.6078201593565,435.4084016401304,0.0,4.0
2015/01/26 03:02:45 ,799194,InitialSOAR,0.1,98,4,20,532.2373436643167,531.9472136461238,0.0,0.0
2015/01/26 03:19:19
,811835,InitialSOAR,0.1,98,4,10,435.80527709034203,435.63080193767917,0.0,4.0
2015/01/26 03:35:20 ,790604,InitialSOAR,0.1,98,4,15,485.4964603107468,485.2475194961413,0.0,0.0
2015/01/26 03:51:51
,820998,InitialSOAR,0.1,98,4,12,467.17504123558365,466.86758677544844,0.0,1.0
2015/01/26 04:08:07 ,796754,InitialSOAR,0.1,98,4,11,443.5076762795701,443.2833147150421,0.0,4.0
2015/01/26 04:40:32 ,LastRound,InitialSOAR,0.1,98,4,10,423.6436235014151,
423.49406072660656,0.0,4.0
2015/01/26 04:57:05
,LastRound,InitialSOAR,0.1,98,4,10,436.22457660034627, **435.991838930856**,0.0,4.0

Cassandra (SoAR – LowUpdate)

2015/01/26 05:09:09 , BGconfigFile, /home/ubuntu/_rating/cassandra_config_lowUpdate_SoAR.txt
2015/01/26 05:09:09 ,time ,Objective, SLA Latency, SLA Confidence, numClients, ThreadCount, Throughput, ActThroughput, totalStaleness, TotalClientsSucceeded
2015/01/26 05:26:16
,807122,InitialSOAR,0.1,98,4,10,441.68140055923357,438.9390170117668,0.0,4.0
2015/01/26 05:40:31 ,673243,InitialSOAR,0.1,98,4,20,481.2976392062589,478.6104806063462,0.0,0.0
2015/01/26 05:56:31 ,779172,InitialSOAR,0.1,98,4,10,432.8229868580662,430.587484980192,0.0,4.0
2015/01/26 06:12:26 ,774559,InitialSOAR,0.1,98,4,5,332.0420580487388,329.938961641644,0.0,4.0
2015/01/26 06:28:49
,812931,InitialSOAR,0.1,98,4,3,167.86231270748553,166.83835656965118,0.0,4.0
2015/01/26 06:46:42
,861808,InitialSOAR,0.1,98,4,2,117.16019471271017,116.58576288991594,0.0,4.0
2015/01/26 07:03:49
,LastRound,InitialSOAR,0.1,98,4,1,71.2430704689607, **70.86017746258469**,0.0,4.0

Cassandra (SoAR – HighUpdate)

2015/01/26 00:10:34 , BGconfigFile, /home/ubuntu/_rating/cassandra_config.txt
2015/01/26 00:10:34 ,time ,Objective, SLA Latency, SLA Confidence, numClients, ThreadCount, Throughput, ActThroughput, totalStaleness, TotalClientsSucceeded
2015/01/26 00:26:58
,803640,InitialSOAR,0.1,98,4,10,471.41344136025026,432.20241219758407,0.0,4.0
2015/01/26 00:43:26 ,808444,InitialSOAR,0.1,98,4,20,576.0035276902364,528.262076663242,0.0,0.0
2015/01/26 00:59:41 ,805406,InitialSOAR,0.1,98,4,10,473.6172375234472,434.836819413201,0.0,4.0
2015/01/26 01:16:08
,805251,InitialSOAR,0.1,98,4,15,530.5757356931908,487.42657471626933,0.0,0.0
2015/01/26 01:32:17
,798421,InitialSOAR,0.1,98,4,12,499.00449296096133,457.48938381768914,0.0,0.0

2015/01/26 01:48:33
,805861,InitialSOAR,0.1,98,4,11,484.9957285229022,444.96133165012316,0.0,3.0
2015/01/26 02:04:59
,LastRound,InitialSOAR,0.1,98,4,10,476.42641734007304, **438.5070727326181**,0.0,4.0

Cassandra (SoAR - LowUpdate, SLA Conf = 80)

2015/01/29 03:22:00 , BGconfigFile, /home/ubuntu/_rating/4_cassandra_config.txt
2015/01/29 03:22:00 ,time ,Objective, SLA Latency, SLA Confidence, numClients, ThreadCount, Throughput, ActThroughput, totalStaleness, TotalClientsSucceeded
2015/01/29 03:38:39
,818949,InitialSOAR,0.1,80,10,20,566.5721087513589,519.4980012446155,0.0,4.0
2015/01/29 03:55:03 ,803751,InitialSOAR,0.1,80,10,40,695.0903152573969,638.3580824141,0.0,0.0
2015/01/29 04:11:34 ,810302,InitialSOAR,0.1,80,10,20,574.206602038895,525.8820230172375,0.0,4.0
2015/01/29 04:28:02 ,807195,InitialSOAR,0.1,80,10,30,631.719959980808,578.6630399191625,0.0,3.0
2015/01/29 04:44:10
,786229,InitialSOAR,0.1,80,10,25,632.8283999089073,580.9140088757281,0.0,4.0
2015/01/29 05:00:46 ,815796,InitialSOAR,0.1,80,10,27,626.1378276975224,574.227329947599,0.0,4.0
2015/01/29 05:17:20
,822973,InitialSOAR,0.1,80,10,26,610.1784347314197,560.0112814415513,0.0,4.0
2015/01/29 05:48:27
,LastRound,InitialSOAR,0.1,80,10,24,603.8154671079427, **553.5205178978515**,0.0,4.0

NuoDB (database population, no image)

300 Seconds: Load is in progress
310 Seconds: Load is in progress
320 Seconds: Load is in progress
Done loading resources
Done loading manipulation
Done creating indexes and closing db connection
330 Seconds: Load is in progress
Done doing load sanity check
LoadTime (msec)=**327432**
Load configuration parameters (those missing from the list have default values):

useroffset: 0
confperc: 1
nuodb.passwd: dba
resourcecountperuser: 100
nuodb.server: 172.31.54.157
threadcount: 10
usercount: 10000
friendshipworkload: edu.usc.bg.workloads.FriendshipWorkload
requestmean: 0.27
resourceworkload: edu.usc.bg.workloads.ResourceWorkload
requestdistribution: dzipfian
friendcountperuser: 100
userworkload: edu.usc.bg.workloads.UserWorkload
nuodb.user: dba
db: NuoDB.NuoDBClient
insertimage: false

Stats queried from the data store:
MemberCount=10000

```
    ResourceCountPerUser=100
    FriendCountPerUser=100
    PendingCountPerUser=0
340 Seconds:  Load is in progress
state thread came out of while
load state thread exited
Loading datastore completed.
EXECUTIONDONE
```