DD2471: Modern Database Systems and Their Applications
Distributed data management using Apache Cassandra

Frej Connolly, Erik Ranby, and Alexander Roghult

KTH CSC
The School of Computer Science and Communication
100 44, Stockholm, Sweden
{connolly,roghult,ranby}@kth.se
https://www.kth.se/csc

Abstract. The aim of this project was to get a hands on experience with a distributed database system by developing an application that uses one of these. Apache Cassandra is a distributed database management system with the aim of handling large amounts of data without a single point of failure. It was initially developed at Facebook, but it is now a top level project at Apache.

The application should use Cassandra as data store and distribute the data across 3 nodes. The application will be used to query Cassandra on one node while more data is inserted on another. A staged failure of one or more nodes will be done and the result from the application measured. We are also interested in examining how the data is distributed and replicated over the different nodes. The data used are Wikipedia dumps stored in large XML files.

Keywords: Apache Cassandra, Distributed Databases

1 Introduction

This project was conducted for the course DD2471 Modern Database Systems and Their Applications. With distributed database management systems becoming all the more common when handling large sets of data, general knowledge of these seem relevant for students studying Computer Science when applying for future jobs. As the project members had never used a distributed database management system before this course, it was decided that the project should in some way address this field. The project should provide the members with a hands-on experience using a distributed database management system and examine how data is distributed across the cluster as well as how sensitive the system is to node failure.

2 Project description

In the paper Solving Big Data Challenges for Enterprise Application Performance Management[1] the authors perform a comprehensive performance evaluation of six modern database management systems. Their conclusion was that Apache Cassandra was the clear winner in regards of scalability, achieving the highest throughput for the maximum number of nodes in their tests. For this reason the group members decided create a simple web application, using Apache Cassandra for data storage. The goal of the project is to allow the project members to get first hand experience with Apache Cassandra, give them insight in how it can be configured and setup using a multi node cluster as well as how it handles node failures.

3 Implementation

The implementation phase of the project mainly consisted of three parts. First the database system itself had to be installed, setup and configured on all nodes. Also, tools for monitoring the status and data distribution on the nodes were configured. Next, a program for loading data from the chosen data source into the database was constructed. Finally, an application directed to the final user was written. For this project the database cluster was spread out over three nodes.

The data chosen to be used in the project were article dumps from the English Wikipedia stored in XML files. The dumps contained information about changes made to Wikipedia pages. The choice to use data from Wikipedia was made on the basis that Wikipedia has a lot of data that is easy to access.

3.1 Setup of database systems

First of all Apache Cassandra was downloaded and installed on all three nodes. To setup the cluster, each node needed to configure its cassandra.yaml configuration file, specifying the IP-addresses of the machines involved in the cluster. After this simple configuration Cassandra was started and the three different instances found each other and set up a connection. From this point the CQL (Cassandra Query Language) interactive terminal could be started to send queries to the database.

The first thing needed to be done before inserting data into the database was to create a keyspace. A keyspace is a structure that holds together several tables, much like a database on relational database systems. When creating the keyspace, a replication factor had be set. The replication factor specifies how many copies of every row that should be created. The copies are always stored on different nodes to protect the complete system from failing if one or a few nodes goes offline. For this keyspace, where three nodes were supposed to be used, a replication factor of two was chosen. This leads to that the system will

function as long as at least two of three nodes are up and running. Furthermore, each node will hold $^2/_3$ of the total amount of data.

Apart from setting up the database itself, a graphic monitoring tool called OpsCenter[2] from DataStax was installed to get a clear and simple overview of the database cluster. For this tool to work properly, agents that gather and send data to OpsCenter had to be installed on all nodes.

When choosing a data model it is important that one does not follow the rules of relational databases. For one thing, there are no joins in CQL. The philosophy behind Cassandra data modelling is that disk space is cheap[3]. Therefore one should not be afraid of duplicating data if it will increase the performance of a query. And while writes aren't necessarily free, they are very cheap, while reads are expensive. We therefore chose to create two tables, named pages and revisions, in our keyspace that were tailored around our most common queries; fetch all pages, fetch a certain page, fetch all revisions of a certain page and fetch a page with a given name. In order to be able to fetch a page for a certain page name an index is placed on the column title.

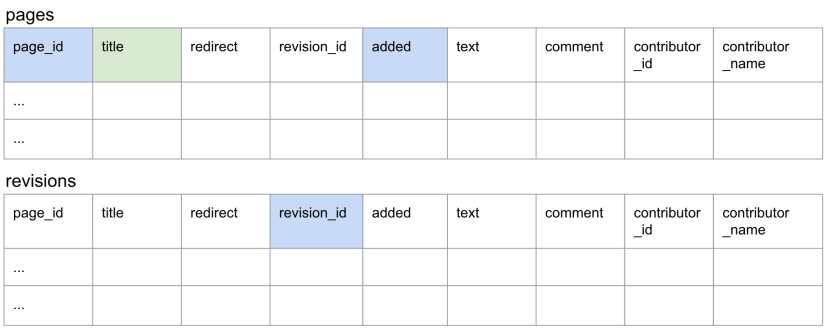

Fig. 1. Database tables.

3.2 Load tool

For creating, dropping and altering tables as well as inserting data, queries can be executed directly from the shell included in the product. But to ease testing of different table configurations and inserting custom amounts of data in a simpler way a data load tool was constructed. It was written in Python and resulted in a quite simple terminal program that had three purposes; creating tables, dropping tables and inserting a chosen amount of data entities. How the tables were to be created and dropped could easily be configured by modifying a .cql file. To communicate with Cassandra, the Python Cassandra Driver[4] from DataStax was used.

3.3 User application

To make the project result in some kind of utilizable application a web application was created. It was also written in Python using the Flask Framework[5]. The purpose of the web app was to provide an overview of the data in the database. In the web app all available articles are listed as a clickable title. When a title is clicked the complete content of that articles latest revision is shown. From here, it is also possible to show all the earlier revisions of the current article.

4 Result

Setting up Apache Cassandra was fairly straight forward. A cluster can be spanning over multiple data centers and regions, but we only tried using all nodes in the same data center. Installing Cassandra on a new node, setting it up and connecting to an existing cluster took less than five minutes after a couple times learning.

Our first approach when creating the database schema for storing the data from Wikipedia was to think relation database and normalize tables. After we learned the basics about Cassandras data model it was hard to re-learn old patterns and habits. Getting into the thought of duplicating data all over the place took some time to get comfortable with.

Since Cassandra does not handle joins this has to be done in the Application layer. This put a lot of responsibility on the developer. The Apache Cassandra Python Driver was easy to use and understand and simplified handling complex queries and data management in the application. It also helped securing the application from injecting malicious statements.

Inserting data into the cluster was done from one node and was then immediately duplicated on all other nodes. Monitoring the duplication process was easy with the DataStax OpsCenter. In our three node cluster with two seeds at least two nodes needs to be up and connected to be able to use the cluster at all. That means one of the two nodes will be a seed. If the third node disconnects and then comes back the cluster will automatically do node balancing and update the third node with new data. With the OpsCenter it was visible that each node hold around two third of the total data. Two nodes are enough to access all data.

Staging a failure of one node from our three node cluster does not affect it. It will still be able to continue to work to both read and write. But staging failure of two nodes means it will no longer continue to work at all. This is due to the replication factor of two that was configured in our cluster. At least two nodes needs to be up and running in order for the cluster to work.

5 Conclusion

The evaluation of Apache Cassandra was simple and easy to perform. This distributed database works well with handling failing nodes, scaling up the number

of nodes and reading performance. The method of building and using an application for evaluating Cassandra can seem superfluous. But it is needed because of the data model since it does not handle joins and foreign keys.

One of the hardest part with Cassandra is to learn the data model and how to utilize it when developing applications. Scaling it up and adding more nodes and machines is somewhat easy. It would make an excellent choice to trust it for both small as well as big project.

Further evaluation that could be done is measuring updating the same data at the same time from different nodes. Which of the nodes will it decide to save and how will this data be replicated over the cluster? Investigating using multiple data centers and multiple clusters at the same time is also another interesting challenge.

References

1. Tilmann Rabl, Sergio Gómez-Villamor, Mohammad Sadoghi, Victor Muntés-Mulero, Hans-Arno Jacobsen, and Serge Mankovskii. 2012. Solving big data challenges for enterprise application performance management. Proc. VLDB Endow. 5, 12 (August 2012)
2. DataStax OpsCenter, http://www.datastax.com/products/datastax-enterprise-visual-admin
3. Hobbs, Tyler: Basic Rules of Cassandra Data Modeling, http://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling
4. Python Cassandra Driver, https://datastax.github.io/python-driver
5. Flask, http://flask.pocoo.org