Cloud Computing (INGI2145)
Assignment 2: Scaling a Micro-blogging Cloud Application

Due 17th December 2015 at 11:59pm

Ribbit is a micro-blogging Cloud application. Since its launch, the application has been growing steadily in users and traffic. This is good news but also bad news: the original application was not built to scale to the numbers that are projected for the near future. You are a new team at ribbit and your role is to improve the application scalability.

This assignment focuses on the challenges to scale ribbit. You will tackle three aspects where scalability is crucial. The first is the application server and its ability to handle concurrent requests. The second is the message feeds, also called timeline. The third aspect where a scalable solution is needed is a streaming pipeline for data analytics.

Ribbit provides a Twitter-like web application. From the point of view of user functionality, as you would expect, users can create accounts and log in; they can then (1) post messages (max 140 characters), (2) follow other users, (3) see the messages posted by followed users. Another piece of functionality consists of a feature that reports the top 10 hashtags in the last 10 minutes, updated every minute.

The original application was created using modern technologies. Ribbit used AngularJS/Node.js/Express for the frontend. It uses MongoDB for the backend. MongoDB is a schemaless NoSQL database that uses a document-oriented data model. In contrast to typical relational databases, MongoDB opts to keep the schema flexible and mostly structureless. While MongoDB was convenient to use until this point, its performance is not suitable for ribbit. Ribbit has decided to adopt Cassandra as the storage technology to power the application.

To simplify your task, some engineers have already produced a version of the application where the unscalable parts have been stripped off. Note: this skeleton of the original application merely represents a convenient starting point for the assignment, but you are free to choose different technologies, provided that you justify your choice and result in a more scalable application. The only restriction is that you need to use Cassandra as the data store.

The following describes in more details the areas of the application that need improvement.

Frontend scaling [0.5pt]

The initial version of the application uses a single-threaded Node.js instance. This is a problem because all of our servers are multi-core. You should devise a solution and provide a working implementation to make use of multiple cores and machines. You are expected to demonstrate that the solution works with multiple cores. Regarding multiple machines, you are expected to discuss a solution that can work with multiple machines even if your application runs inside a single machine.

Timeline scaling [1pt]

Why is it hard to scale the timeline? Consider Twitter: “Twitter now has 150M world wide active users, handles 300K QPS to generate timelines, and a firehose that churns out 22 MB/sec. 400 million tweets a day flow through the system and it can take up to 5 minutes for a tweet to flow from Lady Gaga’s fingers to her 31 million followers” [1].
The basic problem is that when a user requests its timeline, this should comprise of all the recent messages posted by all the users followed by this user. But people are creating content on Twitter all the time. So, the job of Twitter is to figure out how to send it to each user’s followers. What makes this task challenging in that there is also the requirement to be nearly real-time. That is, the goal is to have a message flow to a user in just a few seconds (say 5).
There are generally two solutions to the above problem: fanout on read and fanout on write. They are also referred to as pull on demand or push on change, respectively. These approaches reflect the differences regarding at what point in time most of the work is done: during the write time or the read time. Use references [2-4] to get a better sense of the problem.

It turns out that for the Twitter workload, the fanout on write model is a more scalable solution. Our original application was using the fanout on read model and MongoDB as the data store, which has poor write performance. For this reason, we have decided to adopt Cassandra as the new data store of the application. Cassandra is a distributed data store designed for high write throughput. The design of Cassandra is heavily inspired by the Dynamo key-value store, which we covered in class. As with Dynamo, Cassandra only allows to retrieve a data element inserted in the store via a key. This is an important consideration because the way the data is queried determines the way the data is stored. To better understand the implications of this, make sure you become comfortable with the basic rules of Cassandra data modeling by reading the article in [5].

Design and implement an appropriate way of using Cassandra for storing the tweet data. As the fanout on write model causes most work to be done at write, you should consider how to efficiently implement in a scalable fashion this functionality. While you can write from within the Node.js application itself to Cassandra, this might not be a great solution. What if the write time is very long? What if the Node.js instance crashes?

[1]
http://highscalability.com/blog/2013/7/8/the-architecture-twitter-uses-to-deal-with-150m-active-users.html

[2]
http://highscalability.com/blog/2009/10/13/why-are-facebook-digg-and-twitter-so-hard-to-scale.html
[3] http://www.slideshare.net/nkallen/q-con-3770885
[4] http://qr.ae/RwdxdT
[5] http://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling

Analytics scaling [1pt]

Ribbit has a feature to present **the top 10 hashtags in the last 10 minutes, updated every minute**. For this feature, the original application was using an inefficient implementation: For every user request of the page **/top10**, the application would run a query on the database to read the relevant tweets and aggregate the statistics in the results. This causes many redundant computations and too much load on the database, as the tweets are read every time. Therefore a more scalable solution could be to have a background analytics service that periodically computes the aggregate results from the database and then caches the new statistics into some caching service (e.g., memcache). While this would solve the problem of redundant computations, it still requires costly reads from the database. A better solution would avoid reading from the data store.

Design and implement an appropriate way to perform this analytics task on the incoming tweet data. Decide where to store the analytics results and ensure that these are shown when the user accesses the **/top10** page of the application (our skeleton does not include this page).

Discussion

Before you begin working on the code, we ask that you sketch a plan for your solutions. You should discuss your plan with the course instructors. We recommend that you do this so that you avoid going in the wrong direction. We will have office hours on **Wed 25/11 from 6pm to 7pm** and **Fri 27/11 from 9am to 10:30am**. Plan an appointment in advance, if possible. The office hours are in a.043.

Notes

- In the project skeleton, look for "HINT" to find some punctual suggestions for your implementation.
- The provided skeleton includes two sample datasets of users (users.json) and tweets (sample.json). **We require that you use this sample data**. You will need to create a script to load the data inside Cassandra based on your design. We provide a skeleton for this script called `loader.js`.
- We have updated the puppet config file to install and configure Cassandra on the INGI2145-vm. After your `git pull`, run `vagrant up --provision` to reprovision your VM.

- We have reduced the maximum heap size of Cassandra based on the constraint of 2 GB of RAM for the VM. This should be enough for our sample datasets. But keep an eye for out of memory errors and eventually increase the RAM of the VM.
- You can read how ribbit was originally created by your colleagues last year in this document: http://perso.uclouvain.be/marco.canini/ingi2145/hw2-2014.pdf.

Report [0.5pt]

We ask that you submit a short report in addition to your code. This report should be **no longer than 2 pages** and should answer the following questions:

- Explain the high-level design of your solutions to the three scalability challenges of the application.
- A description of your database design and data model. In particular, how did you decide to store user relations and tweets.
- Describe the challenges you encountered, if any (it's fine not to have anything to say, don't make up something).
- Describe how to run your application (i.e., what commands to execute so that all the components are running). We will test your code in the INGI2145-vm.

Extra Credit

The following extra credit item will be available for this assignment:

- [0.5pt] Deploy your solution on multiple machines and demonstrate that it works. If you use AWS EC2 for this, please use micro or small instances only.

These points will *only* be awarded if the main portions of the assignment work correctly.

Grading

This assignment carries 3pt of credit out of 20. The following base policy will be applied to determine your grade:

- [0.5pt] based on your report.
- [0.5pt] for devising a solution to scaling the application server.
- [1pt] for devising a solution to scaling the timelines.
- [1pt] for devising a solution to scaling the analytics.

To assess the solutions, we will organize office hours where you will explain your solutions and demonstrate that they work.

If you only submit a partial solution, then we will award up to [1pt] for the effort depending on the quality of the code, documentation and design (as described in the report).
You should document your code properly. A small credit will be awarded for good documentation, and **no credit** will be given for undocumented code.
There will be **no extensions** to the assignment deadline. An (automatic) lateness penalty applies for homework turned in late: 10% is deducted on the assignment grade for each day late or fraction thereof. If an assignment is 1 minute late, it is one day late.

Submission

You should submit your solution as a single .tgz file via INGInious (https://inginious.info.ucl.ac.be/course/INGI2145) before the deadline. Access to INGInious requires a valid INGI or UCL account.

Submission checklist

- ❏ You have written a report with all the information requested.
- ❏ Your code contains a reasonable amount of documentation.
- ❏ Your .gz file is smaller than 500 KB. **Do not** submit your database contents, nor the node_modules directory or JAR files.