Homework 9: Distribution

Facelook has been devastated. The team at Giggle, aiming to make Giggle Plus the supreme social network, managed to sabotage our systems and delete any trace of the Facelook platform save a simple GUI. Zark Muckerberg has asked you and your partner(s) to save the day and remake Facelook. He only cares about core functionality for now: friends, friend requests, subscriptions, profiles and the news feed.

The demand was very high for Facelook, so one machine cannot store and serve all the data. Zark wants you to partition the data so that it can scale up, again assuming all the users haven't switched to Giggle Plus.

Finally, Zark is really full of himself, so he is afraid that the machine which hosts his profile will be overwhelmed with calls for his most recent status updates. To fix this, he wants you to implement a cache so that this won't be as much of a problem and the service can handle his awesomeness.

For this assignment you may work alone or with any other single project partner, i.e. teams of one or two. Your partner does not need to be in the same recitation as you, nor does your partner need to be the same as your partner(s) for hw6. Expect details soon via Piazza for how to register your team (or tell us you are working alone) so we can set up a SVN repository for you to submit your work.

Goals

- Write client-server code
- Scale the server to multiple machines
- Use caching to reduce latency and server load

Instructions

You will be using your Facelook interface from Assignment 5 and implementing a distributed back-end with partitioning and caching. Your back-end does NOT have to be persistent. If you did not complete Assignment 5, or you would rather use a "staff approved" GUI, you may use the interface we have provided.

The following are specifications for your back-end implementation:

Facelook Behavior

Your application's behavior should be identical to the behavior described in Assignment 5. For ease of access, we have included the Assignment 5 documentation in the same directory as the Assignment 9 documentation.

Partitioning

You must partition your data across multiple servers. You may assume that your servers will never fail and that the number of servers never changes, even though these assumptions are very unrealistic for real systems. How you partition the data among servers is up to you. (You may hard-code server names / IP addresses into your application; just be sure to include instructions for the TAs in a README.txt file so they can run your solution.)

Partitions should be hidden from the Facelook application. Consider the following behavior. If the client connects to server A and requests something that is not stored locally, server A will contact some server B and get the data, then present the data to the Facelook application as if it were stored locally. Essentially, the application does not differ if Facelook is run on a large cluster or Zark Muckerberg's Macbook Air.

Caching

Each server should have a cache using time-based expiration and also some replacement policy if the cache is full. You should define a "full" cache using some reasonable memory size or number of items; be sure to describe how to adjust this configuration so that the TAs can test your work, or set the cache size to be small enough such that the TAs can easily test your replacement algorithm without resizing the cache.

Design Documentation

This assignment is very open-ended, and we have intentionally left many choices up to you and your group. Using at most two pages, describe your design decisions including, but not limited to, how you choose to store data, how you choose to send it, the replacement policy in your cache, etc. We recommend that you write this documentation as you implement your solution rather than delay writing it until afterward.

Also, we require that partitioning be used for this assignment. What are the pros and cons of this method and if you were to be given complete freedom on your Facelook creation, what would you make sure to include?

Hand in Assignment 9 as an Eclipse project, and be sure to include any instructions or notes in a README.txt file at the same level as your project. The README.txt file should contain instructions on how to get your servers running and start your code, as well as anything else you believe is important for evaluating your solution.

Evaluating your homework submission

In order to achieve full credit for this homework, you should:

- Not use static functions (other than `main`)
- Not use static fields
- Handle all exceptions that might be generated by your program
- Make all class data fields private
- Document public methods with a short /** javadoc comment */ that describes what the method does.
- Use proper style and Java naming conventions.

Points for this assignment will be allocated approximately as follows:

- Compiles and behaves as expected: 10 points
- Data is partitioned among multiple machines: 40 points
- Working cache with expiration time: 30 points
- Design documentation: 10 points
- Style and design: 10 points
- Work submitted for Lab 9: 10 points

Tips

- Start early! The design is almost entirely up to you and your partner(s), it just must have the behavior outlined and use the GUI weve provided. This assignment will be a fair amount of work, but it will (hopefully) be quite fulfilling.
- The application is completely separate from the data management. Consider dividing up the work among your group.
- You can define a list of servers that the Facelook application can connect to and choose one to connect to. This can be done randomly, but in a real world situation this won't be the case.
- Please include a README.txt that will describe how to get your servers running and to start your code.