6.824 - Spring 2018

6.824 Project

Proposals due:	March 23 23:59
Code and write-up due:	May 11 23:59
Presentations:	May 17

Introduction

You can either do a final project based on your own ideas, or Lab 4.

If you want to do a project, you must get our approval for your idea in advance. You must form a group of 2 or 3 6.824 students to collaborate on the project. You'll turn in your code and a short write-up describing the design and implementation of your project, and make a short in-class presentation about your work. We will post your write-up on the course web site unless you talk to us about why you want to keep it confidential.

Your project should be something interesting and challenging that's closely related to 6.824 core topics, such as fault tolerance. The project must involve at least as much effort as Lab 4. Below you'll find some half-baked ideas that we think could turn into interesting projects, but we haven't given them too much thought.

Deliverables

There are four concrete steps to the final project, as follows:

1. Form a group and decide on the project you would like to work on. Feel free to use Piazza to find group members and discuss ideas. Course staff will be happy to discuss project ideas via e-mail or in person.
2. Flesh out the exact problem you will be addressing and how you will go about solving it. By the proposal deadline, you must submit a proposal (less than a page) describing: your **group members** list, **the problem** you want to address, **how you plan to address it**, and what are you proposing to **specifically design and implement**. Submit your proposal to https://6824.scripts.mit.edu/2018/handin.py/. We'll tell you whether we approve, or not, and give you feedback.
3. Execute your project: design and build something neat!
4. Write a document describing the design and implementation of your project, and turn it in along with your project's code by the final deadline. The document should be about 3 pages of text that helps us understand what problem you solved, and what your code

does. You can either send the code to the staff list or provide a link to an repository (e.g., on GitHub) in your writeup. The code and writeups will be posted online after the end of the semester.
5. Prepare a short in-class presentation about the work that you have done for your final project. We will provide a projector that you can use to demonstrate your project.

Half-baked project ideas

Here's a list of ideas to get you started thinking -- but you should feel free to propose your own ideas.

- Re-implement any of the systems described in the papers that 6.824 lectured on.
- Build a distributed, decentralized, fault-tolerant Reddit.
- Make the state synchronization protocol (DDP) in Meteor more efficient (e.g., send fewer bytes between server and client) and more fault-tolerant (e.g., a client should be able to tolerate server failures, as long as enough servers remain live).
- Build a system for making Node.js applications fault-tolerant, perhaps using some form of replicated execution.
- Improve the Roost Javascript Zephyr client by replicating the backend to make it fault-tolerant.
- Add cross-shard atomic transactions to Lab 4, using two-phase commit and/or snapshots.
- Build a system with asynchronous replication (like Dynamo or Ficus or Bayou). Perhaps add stronger consistency (as in COPS or Walter or Lynx).
- Build a file synchronizer (like Unison or Tra).
- Build a coherent caching system for use by web sites (a bit like memcached), perhaps along the lines of TxCache.
- Build a distributed cooperative web cache, perhaps along the lines of Firecoral or Maygh.
- Build a collaborative editor like EtherPad, using eventually-consistent or CRDT primitives.
- Build something useful around a blockchain.
- Add one of the following features to "Distributary", an eventually-consistent research data-flow system (with some similarities to Naiad):
    1. Transactions with two-phase commit across different base tables.
    2. Client-side data-flow operators and stateful caches.
    3. Replication for individual data-flow operators or the whole data-flow.
    4. Rollback recovery from failures similar to this proposal.