6.824 - Spring 2020

6.824 Project

Proposals due:	April 2 23:59
Code and write-up due:	May 14 23:59
Presentations:	May 20 in class

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. At the end of the term 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'll post the project write-ups and code on the course web site.

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/2021/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 presentation about the work that you have done for your final project, and deliver it during the last class meeting.

Half-baked project ideas

You should feel free to propose your own project idea. If you'd like some starting points, here are some topics that might (or might not) be worth thinking about.

- Re-implement any of the systems described in the papers that 6.824 lectured on.
- Build a very high-performance Raft implementation, changing the design as needed.
- Build a distributed, decentralized, fault-tolerant Reddit.

- Build a system for making Node.js applications fault-tolerant, perhaps using some form of replicated execution.
- Add cross-shard atomic transactions to Lab 4, using two-phase commit and/or snapshots.
- Build a data-flow processing system in the style of Google FlumeJava or Spark or Naiad.
- 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.
- Use a block-chain to build something other than a crypto-currency.
- Build a fault-tolerant and/or sharded file service.
- Build a distributed shared memory (DSM) system, to make it possible to run existing parallel code intended for a single multi-core machine, but on a cluster of machines.
- Build a distributed block store in the style of Amazon EBS or FAB. Maybe you can get standard operating systems to talk to you network virtual disk using iSCSI or Linux's NBD (network block device).
- Build a geo-replicated storage system, like Dynamo or COPS, perhaps providing something useful and/or efficient in the the way of transactions or consistency.
- Use modern high-speed NIC features (e.g. RDMA or DPDK) to build a very high-speed service, perhaps with replication or transactions.
- Use modern fast non-volatile storage (e.g. Intel Optane) to simplify the design of a fault-tolerant system.
- Build a fault-tolerance framework that's easier than Raft to layer service code on top of.
- Figure how to say something useful about whether applications really need strictly consistent storage, or what the cost in application complexity is of having to use storage with weak consistency.
- Build a data-processing system that is good at both big data (like MapReduce and Spark) and on-line processing (like a key/value store or SQL database).