<u>CMSC 621</u>

Fall 2023

Project

The Mechanics

You will do this project in groups. The group shall have two members.

- On or before **October 2**, you will submit (electronically) a document which provides preliminary system specs and design, assumptions you have made, and a plan of execution (including a proposed timeline). It should also include some preliminary idea of what experiments you will conduct to show that your system works as described.
- On **October 16 and November 6**, you will submit brief reports (approx. 2 pages) of your progress to date, changes in the design if any, as well as an updated timeline. You should describe what has been accomplished, what is still not done, if you are on the schedule you anticipated and if not, what mitigating actions you are taking etc. The second report should also have a well-developed set of proposed experiments you will include to prove that your system works.
- The project report detailing your work (the system design, experimental studies etc.) as well as submission of your code, will due on **December 3.** Please note that pdf is the only format we will accept. The paper should be written like a conference paper. It should have an abstract, an introduction, some background and related work, your own approach described, a results and evaluation section. Here is a great primer that tells you what your submitted paper should look like (http://www.cs.columbia.edu/~hgs/etc/writing-style.html) from Schulzrinne. Using the IEEE or ACM conference submission format (https://www.ieee.org/conferences/publishing/templates.html) , submit an (upto) 8 page paper.
- You will arrange to demonstrate your project to the instructional team between **December 4** and **December 17**.
- You will submit, by **December 6**, a short video of your team "presenting" the paper. Imagine if you were presenting to the conference. We will ask everyone to "grade" their peers' presentations – more on this later.

You can discuss the project across groups. However, you are *not* allowed to share solutions. You may read papers and textbooks in this area as well -- some pointers are provided in this document. However, you should cite the sources you have consulted. You may do this project using any language you want. You MAY NOT (re) use code of an existing similar system. We already know where similar systems are on GitHub, and will use automated tools to verify the originality of your code. You *may not* in general use ChatGPT, CoPilot or similar tools, except to write routine parts of your code. Where you have used these tools, please document in your design and report. Your demonstration/experiments should use several virtual machines – 4-5 at least.

The Bare bones Task (65 points)

In this project, you will design a simple distributed file system with replication and disconnection management features. The system will consist of several file servers, and clients which interact with them. Normal operations related to file systems, such as creating, deleting, opening, closing, reading, writing, seeking etc. should be allowed. These requests should be processed by the fileserver that the client is connected to. When a file is created by a client, it resides on the fileserver which the client is connected to, and is *owned* by this client. In other words, the primary copy of the file is kept on this server. When some fileserver receives a request for operations on a file, it should discover and contact the server which owns the file and create a replica locally. All operations should then proceed on the replica. For the basic task, we will create a pessimistic replication scheme. In other words, any operations (such as write) which could throw the replicas out of sync will be permitted on one replica only. However, any number of file servers can freely create replicas and perform operations which do not alter the file. Your system should allow a fileserver wanting to create a replica to tell the owner filesystem the mode in which it wants a file. If the mode is one which will change the file, then the requester should also specify a *lease period*. The lessee fileserver promises to return any changes to the file to the lessor (owner) within the lease period. If changes are returned in time, then the owner fileserver should "commit" them, and update all other replicas appropriately. If the lessee does not return the changes before lease expiry, then the lessor should invalidate any changes that the lessee many have made (and send it a message to this effect). Any request for a lease on a file that has already been leased out should be blocked. The owner should lease the file out in a FIFO process and unblock the corresponding process.

Resilience in Presence of Node Failures (20 points)

Any server may fail randomly or get disconnected. This is especially true if nodes are mobile or have poor connectivity. You should guard against this by providing replication of the file across servers. This replication can be done dynamically as a part of the basic system described above. However, compared to the basic case, you need to make sure that if the primary of a file dies, someone else is elected/selected as primary. You may assume that failure and disconnection is graceful with respect to this issue – namely a node has enough time to identify some other replica as primary if it fails and inform the other nodes that have replicas.

Optimistic Replication (15 points)

The idea here is to extend the bare bones system to permit multiple servers to simultaneously operate on replicas in a manner which will change them (e.g. concurrent writes on replicas). There are several published approaches in literature to achieve this, and you can chose to implement (some subset) of any one of them. Good starting points include the CODA and SPRITE papers on the course web site, besides the URLs provided below. When the changes are returned and do not conflict, they should be merged as needed and the file updated. If the changes are inconsistent, your system should simply flag an error and chose one of the changes to commit. Your design documents should specify exactly what you would implement.

Experimental Evaluation

A part of your project is to design and carry out an experimental validation to convince us that your system works. This means your experiments should show the basic system working, show the system working in presence of disconnection, and show examples of conflicting and non conflicting changes to files. You should also analyze your system for scalability -- both in the number of clients and the number of servers. **Graphs** are very important in this respect, they help give us a clear understanding of your system under varying types of requests, sizes, loads etc. Comment on the results of your experiments and tests. You should experiment with different workload mixtures. Comment on the results you get -- try to identify what assumptions you made or implementation mechanisms of your system cause particular scalability patterns. Try and identify both the strengths and weaknesses of your system. If you use your experimental results to refine your design, make sure you bring this out in your report. Log your progress, and capture intermediate results, so that you can show your work even if your demo doesn't go smoothly. We will use your experimental results to evaluate your project.

Some general suggestions

As should be evident to most of you, it is imperative for a project of this complexity and involving teams that you *design your system before you code*! In your design, you will need to make assumptions as you flesh in the details of the system. Please make sure that you state them in your design document. Make a timeline for your work, and try and stick to it. Where you divide tasks, make sure you clearly define points of articulation and interfaces between modules. As you form groups, please make sure that you can find a common time to meet. This is especially true for those who are part time students and hold jobs which will restrict your schedule. Please comment your code well -- it will help both you and us. You in figuring out code your partners have written, us in grading it. Also, use some form of revision control on your source tree. You may want to create a (private) repository on Github. This will help if lightning strikes, UPC fails and machines/disks crash, making your recent changes disappear! Please do create makefiles as well.

References

We have discussed some distributed Filesystems in class (CODA, GFS etc.) in class. Here are some other references.

1. The Coda System (http://www.cs.cmu.edu/afs/cs/project/coda/Web/coda.html)
2. The FICUS System (http://ficus-www.cs.ucla.edu/ficus/)
3. The Sprite System (http://www.cs.berkeley.edu/projects/sprite/sprite.html)
4. The NOW/xFS System (http://now.cs.berkeley.edu/Xfs/xfs.html)