CS425, Distributed Systems: Fall 2023
Machine Programming 3 – Simple Distributed File System

Released Date: October 10, 2023
Due Date (Hard Deadline): Sunday, November 5, 2023 (Code+Report due at 11.59 PM)
Demos on Monday November 6, 2023

The is an intense and time-consuming MP! So please start early! Start now!

FakeNews Inc. (MP2) just got acquired in a hostile and reluctant way by the fictitious company ExSpace Inc. (whose business model is to do couples therapy…but in space. They are based on a tweet/X by someone on social media that lack of gravity increases chances of reconciliation. And as we all know, all tweets/X's are highly scientific.). Anyway, ExSpace loved your previous work from MP2, so they've re-commissioned you to build a Simple Distributed File System (**SDFS**) for them. SDFS will run on 10 machines in their ExSpace's spaceship as it orbits the earth. SDFS is a simplified version of HDFS (Hadoop Distributed File System) – you can look at HDFS docs and code, but you cannot reuse any code from there (we will check using Moss). Clients of the file system can be any one of the 10 VMs.

You must work in groups of two for this MP. Please stick with the groups you formed for MP2. (see end of document for expectations from group members.) Remember – Move Fast and Break Things! (i.e., build some pieces of code that run, and incrementally grow them. Don't write big pieces of code and then compile them all and run them all!)

This MP requires you to use code from both MP1 and MP2.

SDFS is intended to be scalable as the number of servers increases. Data stored in SDFS is tolerant up to **three simultaneous machine failures** (as usual, like MP2, after the system converges, more failures are again allowed to occur, up to 3 failures simultaneously). After failure(s), you must ensure that data is re-replicated quickly so that another (set of) failures that happens soon after is tolerated (you can assume enough time elapses for the system to quiesce before the next failure(s) occur(s)). **Store the minimum number of replicas of each file needed to meet this feature. Don't over-replicate.**

SDFS is a flat file system, i.e., it has no concept of directories, although filenames are allowed to contain slashes.

Thus, the allowed file ops include: 1) `put localfilename sdfsfilename` (from local dir), 2) `get sdfsfilename localfilename` (fetches to local dir),

3) `delete sdfsfilename`. `Put` is used to both insert the original file and update it (updates are comprehensive, i.e., they send the entire file, not just a block of it).

For demo purposes you will need to add two more operations: 4) `ls sdfsfilename`: list all machine (VM) addresses where this file is currently being stored; 5) `store`: At any machine, list all files currently being stored at this machine. Here `sdfsfilename` is an arbitrary string while `localfilename` is the Unix-style local file system name. All commands should be executable at any of the VMs, i.e., any VM should be able to act as both client and server.

Mandatory: SDFS allows at most one machine to write a file at a time, but allows *up to 2 machines to read a file* simultaneously. It does not allow a writer and a reader simultaneously. For conflict operations (e.g., write-write or write-read), one operation must finish completely before the next operation starts. A goal of your design is to make your system *starvation-free*, i.e., no request must be made to wait forever – one scenario is where readers keep arriving, and a writer is made to wait forever until reads finish. In particular, ensure that no writer waits for more than 4 consecutive reads to finish before it gets write access. Similarly, ensure that no reader waits for more than 4 consecutive writes before it gets access.

Handle failure scenarios carefully. If a node fails and rejoins, ensure that it wipes all file blocks/replicas it is storing before it rejoins. Think about all failure scenarios carefully and ensure your system does not hang. For instance, what if a node sends a write and then fails before the confirmation or after receiving the confirmation notice but before responding? Work out these failure scenarios and ensure you handle them all.

Other parts of the design are open, and you are free to choose. Keep your design the simplest possible to accomplish the goals, but also make it fast. Here is a first cut way to structure your design. One of the servers should be the leader server. The leader is responsible for deciding which files get stored where. The leader can also queue read and write requests. All queries and operations can go through the leader. If the leader fails, a new leader should be re-elected quickly. While there is no leader, the system should not lose data or crash – however, it is ok for file operations to not be possible while the leader is down. Is this design enough to tolerate three simultaneous machine failures? No! Modify it so that it satisfies all the requirements!

Design first, then implement. Keep your design (and implementation) as simple as possible. Use the adage "KISS: Keep It Simple Si…". Otherwise, ExSpace may involuntarily send you to space without a partner or any counseling.

Think about design possibilities: should you replicate an entire file, or shard (split) it and replicate each shard separately? How do you do election? How does replication work – is it active replication or passive replication? How are reads processed? Can you make reads/queries efficient by using caching? How exactly does your protocol leverage MP2's membership list? What does a quorum mean and how do you select it? Optimize for speed and correctness, but also keep it simple. Don't go overboard, and **please keep it simple.**

Create logs at each machine. You can make your logs as verbose as you want them, but at the least you must log each time a file operation is processed locally. We will request to see the log entries at demo time, perhaps via the MP1's querier.

Use your MP2 code to maintain membership lists across machines.

You should also use your MP1 solution for debugging MP3 (and mention how useful this was in the report).

We also recommend (but don't require) writing unit tests for the basic file operations. At the least, ensure that these actually work for a long series of file operations.

Machines: We will be using the CS VM Cluster machines. You will be using 7-10 VMs for the demo. The VMs do not have persistent storage, so you are required to use git to manage your code. To access git from the VMs, use the same instructions as MP1.

Demo: Demos are usually scheduled on the Monday right after the MP is due. The demos will be on the CS VM Cluster machines. You must use the max of all the VMs you have for your demo (details will be posted on Piazza closer to the demo date). Please make sure your code runs on the CS VM Cluster machines, especially if you've used your own machines/laptops to do most of your coding. Please make sure that any third party code you use is installable on CS VM Cluster. Further demo details and a signup sheet will be made available closer to the date.

Language: Choose your favorite language! We recommend C/C++/Java/Go/Rust/Python. We will release "Recommended MPs" from the class in these languages only, as long as we have enough submissions in the respective language (so you can use them in subsequent MPs).

Report: Write a report of less than 3 pages (12 pt font). Briefly describe the following (we recommend your report contain headings with the bold keywords): a) **Design**: (no more than a page) algorithm and design used, and how you met the requirements – focus especially on (a.1) your replication level, as well as (a.2) how you avoided starvation for both reads and writes, b) **Past MP Use**: (very briefly, 1-2 sentences) how MP2's membership protocol was used in MP3 and how useful MP1 was for debugging MP3, and c) **Measurements** (measure real numbers, do not calculate by hand or calculator!): see questions below. For each plot/data, please use the terms in brackets, so your report is easy to read.

(i) (Overheads) Re-replication time and bandwidth upon a failure (you can measure for different filesizes ranging up to 100 MB size, at least 5 different sizes).

(ii) (Op times) Times to insert, read, and update, file of size 25 MB, 500 MB (6 total data points), under no failure.

(iii) (Large dataset) Time to store the entire English Wikipedia corpus into SDFS with 4 machines and 8 machines (not counting the leader): use the Wikipedia English (raw text) link at: http://www.cs.upc.edu/~nlp/wikicorpus/

(iv) (Read-Wait) Pick a large file that takes at least 30 s to read/write. Specify the file size, and the typical read time. A reader just starts reading. When a further m readers arrive simultaneously (just as the read starts), how long does the last reader take to finish reading? Plot this against the number of waiting readers m (=1, 2, 3, 4, 5). In particular, also plot the *overhead* due to SDFS, i.e., if read time were 30 s (say), then with m waiting readers the lowest time is (m+2)*30 s. How much % higher than that ideal value is your actual read finish time? You should have two lines (or two plots) – one for actual finish time, one for overhead (if you use one plot, make sure the overhead plot axes are visible – in some cases logarithmic y axes might help!) What trends do you observe? Discuss briefly!

(v) (Write-Read) Pick a large file that takes at least 30 s to read/write. Specify the file size, and the typical time to read and time to write. 1 writer just started writing. A further m writers arrive simultaneously, along with a single reader. How long does the last writer/reader take to finish (whoever finishes last)? Plot this against the number of waiting writers m (=1, 2, 3, 4). Again, also plot the overhead of SDFS. What trends do you observe? Discuss briefly!

For each data point, run at least 5 trials and give the average and standard deviation bars. Discuss your plots, don't just put them on paper, i.e., discuss trends, and whether they are what you expect or not (why or why not). (Measurement numbers don't lie, but we need to make sense of them!). Stay within page limits, otherwise there will be a penalty from ExSpace.

Please make the text within your plots/tables is readable!

Submission: Same as past MPs! Follow the same workflow and instructions on Piazza. There will be a demo of each group's project code. On Gradescope submit your report by the deadline (11.59 PM Sunday). In the submission link, submit by the deadline your report as well as working code; please include a README explaining how to compile and run your code. (Don't forget to give access to your git!). Other submission instructions are similar to previous MPs (submit the Google form with your git link, submit the report on Gradescope, and signup for demo, and attend demo).

When should I start? Start NOW. This MP involves a significant amount of planning, design, and implementation/debugging/experimentation work. **Do not** leave all the work for the days before the deadline – if you do, you won't be able to finish the MP, and there will be no extensions. (This is an exercise for the software engineering process that you encounter at companies, where there is a release date. In our case, the release date does not get moved!)

Evaluation Break-up: Demo [50%], Report (including design and plots) [35%], Code readability and comments [15%].

Academic Integrity: You cannot look at others' solutions, whether from this year or past years. We will run Moss to check for copying within and outside this class – first offense results in a zero grade on the MP, and second offense results in an F in the course. There are past examples of students penalized in both those ways, so just don't cheat. You can only discuss the MP spec and lecture concepts with the class students and forum, but not solutions, ideas, or code (if we see you posting code on the forum, that's a zero on the MP). ExSpace is watching!

We recommend you stick with the same group from one MP to the next (this helps keep the VM mapping sane on IT's end), except for exceptional circumstances. We expect all group members to contribute about equivalently to the overall effort. If you believe your group members are not, please have "the talk" with them first, give them a second chance. If that doesn't work either, please approach Indy.

Happy Filing (from us and the fictitious ExSpace)!