CS425, Distributed Systems: Fall 2023
Machine Programming 1 – Distributed Log Querier

Released Date: Aug 23, 2023
Due Date (Hard Deadline): Sunday September 10, 2023 (Code+Report due at 11.59 PM)
Demos on Monday September 11, 2023

Please form groups of exactly 2 students (no more, no less) for the MPs and **by Thursday August 31st, 2023,** please fill out the form (see "Assignments" page on course website) and send us an email with the group information.
This is a hard deadline as we will be creating VMs right after it. VMs are required for all the MPs in this course. (if you don't submit, you don't get VMs, and you can't do the rest of the MPs).

Note that these MPs are only for non-Coursera (non-MCS Online-Coursera program) students who are enrolled for 4 credits. MCS Coursera students should attempt the MPs on Coursera.

Covfefe! Inc. is the newest, naivest and most fictitious cloud startup in the Valley. Their business model is to fix typos in people's tweets – go figure! Anyways, the company just discovered that distributed systems are hard to debug. Since you knew that already (from CS425), they've hired you to build a solution that they can use. They want a system that is fast and correct.

You will be using the CS VM Cluster for all MPs in this course. This group information above will be used to create VMs for your group.

First, debuggers (e.g., gdb, IDEs) work well mostly in single-threaded programs. In industry, the most popular approach to debugging distributed systems is **logging**. This means that each machine creates one or more local files for logging. These local files accumulate status messages, error messages, and in general anything that you want to log. These logs can then be queried remotely.

Second, any code that we write will have bugs. The industry standard for making sure that your program accomplishes what you desire is **unit testing**. A unit test is a piece of code that calls a small unit, typically a small module, and automatically verifies that it produces the desired outputs for appropriate inputs. Unit tests are run before the code goes into production, to test whether the code satisfies basic functionalities. Unit tests can be short or long, but they are expected to be as comprehensive as possible, e.g., by exploring most code paths. Unit tests

are run without manual intervention. Often, the amount of test code exceeds the amount of production code.

This MP has two related parts.

I. First, you will write a program that allows you to query distributed log files on multiple machines, from any one of those machines. The scenario is that you have N (>5) machines, each generating a log file (named machine.i.log). You open a terminal on any of those N machines. You should now be able to execute a grep command that runs on all the log files across all machines, and prints output on your terminal (with the appropriate line counts, i.e., number of matching lines, and file names to designate where each log entry line came from). You do not need to implement grep from scratch, instead you can call a library implementation of grep or similar program (please ensure you are able to call all its options, especially arbitrary regexps or regular expressions).

Make your program fast. Think of the design before you code. Think about query speed – when you have an infrequent pattern, would it be feasible to fetch all the log files to the querying machine and then run grep? What about when you have a frequent pattern? Does it even make sense fetching files to the querier, or should you always be doing the grep in a local, parallel way at the workers?

For creating the logs, choose the most appropriate way. You are free to use cout/print capabilities of your language. Alternately, you can use Apache Common's logging libraries (see Quick start guide: http://commons.apache.org/proper/commons-logging/guide.html#Quick%20Start), but please make sure that anything you use is installable on CS VM Cluster (with the permissions that you have) – you cannot reuse any remote querying capabilities from these external libraries.

Closer to the demo date, we will be giving you a list of log files that you will use during the demo. Your program should run correctly regardless of the file size, but the demo will test your program on log files with ~300,000 lines.

DO NOT use Mapreduce or Mapreduce-like approaches to solve this MP. That's an overkill, and if you do it, Covfefe! Inc. will say, "You're Fired!" for this mistake.

II. How do you know your program works? This is the MP's second part – you will write unit tests. While unit tests typically run locally, for this MP, you will think more broadly and write distributed unit tests. At the minimum, one unit test that we want you to write is one that generates log files at every machine, with some known lines and other random lines. The log-querying program then runs multiple greps and verifies automatically that the results are what you expect. You should use query patterns that are rare, frequent, and somewhat frequent, and

patterns that occur in one/some/all logs. Tests don't need to be fast or short; they need to be as comprehensive as possible.

If you feel comfortable using a testing framework, you may want to look at those that are popular in industry, e.g., googletest, junit, or others. Alternately, if you find it easier to just write the tests in a raw manner (function calls), that's fine too.

Your log-querying program must be fault-tolerant, i.e., it should fetch answers from all machines that have not failed. It is ok to initialize your instances with hard state, e.g., machine names or ip addresses. It is also ok to assume that the querying machine will not fail (but other machines containing logs might fail!).

You can assume that machines are fail-stop, i.e., once failed, they do not come back online. If you want, you could implement node rejoins, but you don't need to for MP1 (that's part of MP2!).

This is a bootstrap MP for multiple reasons, including that you will use the log-querying program for debugging your subsequent MPs in this course.

While you are free to use whatever you want (sockets, bash, ssh), you probably want to treat this MP as a practice run towards using sockets. MP2 onwards will rely a lot on sockets (or RPCs if you're using Go). Unless you're very familiar with sockets programming already, you should take MP1 as a way of brushing up your socket skills. But once again, MP1 can be done using any paradigms you want. Of course, don't just use something that exists on the web that solves the problem directly -- if you do that, you won't learn (and it probably counts as plagiarism). You can use other libraries that are components in your code.

Machines: You will be using the CS VM Cluster machines for everything, including the demo. About 5-10 VMs will be assigned to you. The VMs do not have persistent storage, so you are required to use git to manage your code (git is industry standard, so here's your chance to learn another useful thing!). To access git from the VMs, do the following:

1. Go to GitLab: https://gitlab.engr.illinois.edu
   Log in with your NetID and password in the UOFI tab.
2. Create a blank private project; this will be your MP1 repository.
3. Under Project Information > Members, add your partner to enable project collaboration.
4. Additionally, reference the staff list https://courses.engr.illinois.edu/cs425/fa2023/staff.html to add all of the TAs and the professor as members with at least "Reporter" level access.
5. Login to each of your VMs and "git clone" the repo you just created by using the url.

6. If you make changes to the project repo, don't forget to sync them with the repos in the VMs by using "git pull ".

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 at least 5 VMs 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 one of C++/Java/C/Go/Rust (but you are free to go beyond this list!). We will release "Best MPs" from the class in these languages only (so you can use them in subsequent MPs).

Report: Write a report of less than 1 page (12 pt font, typed only - no handwritten reports please!). Briefly describe your design (algorithm used), very briefly describe your unit tests (you don't need to detail all your tests), and the average query latency when 4 machines each store 60 MB log files. Make sure you run at least 5 trials per data point, and plot both average and standard deviation. (Most students lose points in MP1 because they didn't plot standard deviations. Standard deviations should always be plotted as "error bars" alongside average. Do not plot SD as a separate plot from average – it is hard to reconcile the two.)

Draw plots where you can – this is more visual than a table or numbers inside text. 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 limit – for every line over the page limit you will lose 1 point! You can submit your report on Gradescope.

Submission: There will be a demo of each group's project code. Submit your report (softcopy) as well as working code. Please include a README explaining how to compile and run your code. Default submission is via gitlab sharing. Submission instructions will be posted on Piazza. If (and only if) the instructions ask you to submit via email, use the subject line "CS425 MP1 Submission" – emails not matching exactly this subject line may be discarded by the email filters.

When should I start? We strongly recommend that you start early on this MP. Each 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 – there will be no extensions.

Evaluation Break-up (If MP is not optional): Demo [40%], Report (including design and performance) [40%], Code readability and comments [20%].

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). If you do, Covfefe! Inc. is watching and will be Sad!

Contributions by Group Members: 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 Logging (from us and the fictitious Covfefe! Inc. They're watching you!)!