CS425, Distributed Systems: Fall 2026
Machine Programming 1 – Distributed Log Querier

Released: Aug 25, 2026
Due (Hard Deadline): Sun, Sep 13, 2026 (code + report due by 11.59 PM)
Demos on Monday September 14, 2026

Please form groups of exactly 2 students for the MPs **by Monday August 31st.** Do not email us your group information, instead fill out the form (see "Assignments" page on course website). **This is a hard deadline so we can create per-group VMs that are required for all MPs. (If you don't submit, you don't get VMs, and you can't do 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 and naivest fictitious cloud startup in Silicon Valley. They went bankrupt 3 years ago, but the company made a miraculous comeback recently under a new CEO, who is also the old CEO; as in the iconic closing lyric from The Who: "meet the new boss, same as the old boss". Their business model is to fix typos in people's tweets/X's – go figure! In any case, 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 system that is fast and correct.

You will use the aforementioned CS VM Cluster for your MPs.

DO NOT use LLMs for generating your MP1 code. MP1 is simple, and you'll lose out on valuable learning of fundamentals. If we find signs of LLM usage or if you're unable to explain your code during your demo then **you will get a zero for this MP**.

First, debuggers (e.g., gdb, lldb, IDEs) work well mostly in single-threaded programs. The popular approach to debugging distributed systems is **logging**. Each VM logs status messages, error messages, and other useful debugging information into one or more local files. These logs are then queried remotely during debugging.

Second, any code that we write will have bugs. One technique used by the industry to ensure program correctness is **unit testing**. A unit test invokes a small unit of our program, typically a small module, with some input data and automatically verifies that it produces the desired outputs. More sophisticated

testing (integration, end-2-end, etc) is done only after unit testing has been satisfactorily completed. Unit tests can be short or long, but are expected to be comprehensive, e.g., by exploring most code paths. Unit tests are run without manual intervention. The total amount of unit test code typically is larger than the program being tested.

This MP has two related parts.

I. 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 (N=10 in the demo), each with a generated log file (named machine.i.log, where i is the VM number). 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: it must include the appropriate line counts for each matching file (with the file name where the match came from). If you call the grep executable directly, it is also acceptable to output the number of lines returned by grep. 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 the original grep options, especially arbitrary regexps or regular expressions via the -E flag: you should *not* have to re-implement these options).

Make your program fast. Design before you code. Think about query speed – when you have an infrequent pattern, does it make sense to fetch all the log files to the querying machine and to 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 individual VMs?

You can choose some convenient way to create the log files. You are free to use the 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 the CS VM Cluster (with your permission levels). You cannot reuse any remote querying capabilities from these external libraries.

As we get close to the demo date, we will give you a list of log files to be used during the demo. Your MP should run correctly regardless of the log 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 need to write distributed unit tests. At the minimum, a 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 are comfortable using a testing framework, you can consider some frameworks that are popular in the industry, e.g., googletest, junit, etc. Alternatively, 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. Also, while a query is in progress, the querying machine will never fail, but other machines containing logs might fail. It should be noted that any machine should be able to act as the querying machine.

You can assume that machines are fail-stop, i.e., once failed, they do not come back online. If you're looking for a challenge, you could implement node rejoins, but it's not needed for MP1.

We consider this a "bootstrap" MP for multiple reasons. For e.g., you will use the log-querying program for debugging your subsequent MPs.

You will use sockets (or RPCs if you're using Go). You will be using sockets or RPCs in future MPs so you can use MP1 to brush up your socket programming skills. 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. Approximately 10 VMs will be assigned to each MP group. You do not need to wait for the VMs to be assigned to get started on development. You can run multiple processes on your personal machine to test your early code while we wait for the VMs to get provisioned by IT. The VMs do not have persistent storage, so you are required to use git to manage your code (git is an industry standard, so here's a chance to learn another useful thing!). By Aug 28, we will post instructions on Piazza detailing how to set up your course GitHub repo for MP version control & submission. If you want to clone your repo on the VMs, you can set up a PAT (or SSH key).

Demo: Demos are scheduled on the Monday right after the MP is due. The demos will be on your CS VM Cluster machines; **you will use all your provisioned VMs during your demo**. 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. Demo details and a signup sheet will be made available closer to the demo date. All group members must attend demos in person (MCS Chicago students can dial in via Zoom—please create a Zoom link).

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: Submit 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 each of them), and the average query latency (defined as time for the last machine to respond with the last byte to the query) when 4 machines each store 60 MB log files. Plot the latency for (i) frequent, (ii) infrequent, and (iii) rare queries. Make sure you run at least 5 trials per data point, and plot both average and standard deviation. (In the past, students typically lost points in MPs 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 the page limit – for every line over the page limit you will lose 1 point!

Submission: (1) Submit your report to Gradescope BEFORE the deadline, and **TAG** your page(s) on Gradescope (there will be a penalty if you don't). Only one group member should submit – you must tag your partner.
(2) There will also be a demo of each group's MP1. Signup sheets & demo instructions will be posted on Piazza.
(3) Submit your working code via the MP1 code submission form. In your repo, please include a README explaining how to compile, how to run your unit tests, and how to run your MP1 code. Further submission instructions will be posted on Piazza.

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: 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 discuss only the MP spec and lecture concepts with other students and on the 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 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 one of the instructors.

Happy Logging; the fictitious Covfefe! Inc. is watching you!