CS425, Distributed Systems: Fall 2015
Machine Programming 1 – Distributed Log Querier

Released Date: August 25, 2015
Due Date (Hard Deadline): Sunday, September 13, 2015 (Code due at 11.59 PM)

(Anything in red font below are updates posted after the MP1 spec release. All such updates are posted with the date of posting. In almost all cases, they are replicas of answers already given on Piazza posts.)

You'reFired! Inc. is the newest, naivest and most fictitious cloud startup in the Valley. Their business model is to fire people – 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 must work in **groups of two** for this MP. Please form your groups by the **5 pm Thursday 8/27. Please follow instructions on the course website to let us know of your group.**

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 into which are accumulated 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 rarely run in production code. 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 produces output on your terminal (with the appropriate lines to designate which log entries come from). While this MP only requires you to implement the grep functionality (along with all its options, especially arbitrary regexps or regular expressions), you can imagine how this could be extended in the future.

Make your program fast. 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 an infrequent pattern?

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 (without the permissions that you have) – you cannot reuse any remote querying capabilities from these external libraries.

DO NOT use Mapreduce or Mapreduce-like approaches to solve this MP. That's an overkill, and if you do it, You'reFired! Inc. will fire you 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 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 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 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 might fail!).

(9/8) 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.

Further details (added after 9/8, replicated from Piazza):

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.

While you're free to use any logs you like, there are some very interesting web proxy traces available here: http://www.web-caching.com/traces-logs.html. You can use any or part of these datasets as the log files in your MP1 (split up the files among multiple machines, don't replicate them).

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 https://gitlab-beta.engr.illinois.edu/
   Login with your netid and password in LDAP tab.
2. Create a project repo, make it private and add your team member including TAs and Professor Indy to the project member.
3. Login to each of your VMs and "git clone" the repo you just created by using the url.
4. 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 C/C++/Java. 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 your unit tests (you don't need to describe all your tests), and the average query latency for 100 MB logs on 4 machines.

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. 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: 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). You'reFired Inc. is watching!

Happy Logging (from us and the fictitious You'reFired Inc.! They're watching you!)!