Cloud Computing Concepts (C3), Part 2

Programming Assignment (Machine Programming or MP) for C3 Part 2 (also for use in CS425 Coursera section)

Name: Key-Value Store

2020

Important Dates

Released: Week 1 of C3 Part 2
Due: End of Week 5 of C3 Part 2

1. What is this MP about?

In this MP, you will be building a fault-tolerant key-value store. We are providing you with the same template provided for C3 Part 1 Programming Assignment (Membership Protocol), along with an almost-complete implementation of the key-value store, and a set of tests (which don't pass on the released code). This means first you need to use your working version of Membership protocol from C3 Part 1 Programming Assignment and integrate it with this assignment. (You will have the option to be graded against our hidden solution for the Part 1 assignment when you submit your work for Part 2, but for best compatibility and for the sake of local testing and debugging, you should try to use your own solution.) Then you need to fill in some key methods to complete the implementation of the fault-tolerant key-value store and pass all the tests.

If you did not complete the MP from C3 Part 1, please do so before you continue with the rest of this document and MP.

First, make sure that you have a working version of C3 Part 1 Programming Assignment. Use the MP1Node.cpp and MP1Node.h from that working version and replace the MP1Node.cpp and MP1Node.h provided in the template to you. Please note that this MUST be your own code. Now that you have figured out how to integrate the Membership Protocol, back to the description of the Fault-Tolerant Key-Value Store.

Concretely, you will be implementing the following functionalities:

- A key-value store supporting **CRUD operations** (Create, Read, Update, Delete).
- **Load-balancing** (via a consistent hashing ring to hash both servers and keys).
- **Fault-tolerance** up to 2 failures (by replicating each key 3 times to 3 successive nodes in the ring, starting from the first node at or to the clockwise of the hashed key).
- **Quorum consistency level** for both reads and writes (at least 2 replicas).
- **Stabilization** after failure (recreate 3 replicas after failure).

This programming assignment uses C++. It is one of the more commonly used languages in industry for writing systems code. For this, you will need at least C++11 (gcc version 4.7 and onwards).

The only files you can change are MP1Node.{cpp,h} and MP2Node.{cpp,h}.

On Course Week 13 (CS 425 version; position in the Cloud Computing Concepts course may vary), you'll find two pages related to MP2. One item is where you can generate "submission tokens" and view your grading results. The other page allows you to launch an online programming environment. These pages also have attached an extra zip file containing the starter template code, in case you want to try working offline instead.

We will be providing you with the autograder scripts (unit tests) that you can use to test that your program passes all requirements.

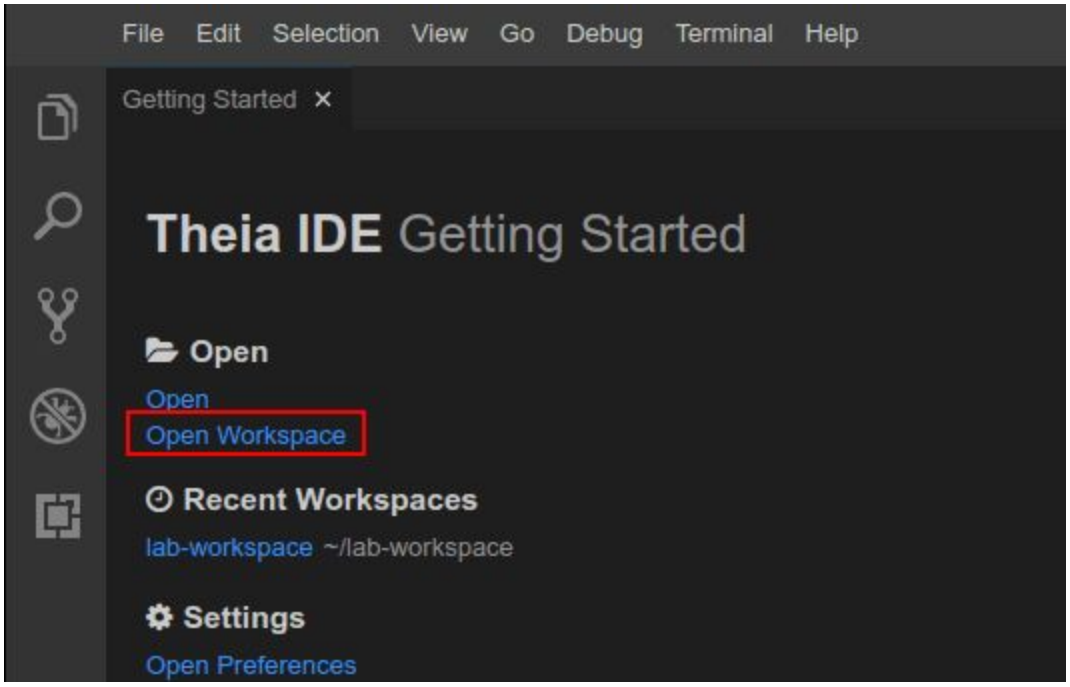
1.1 Theia IDE Workspace

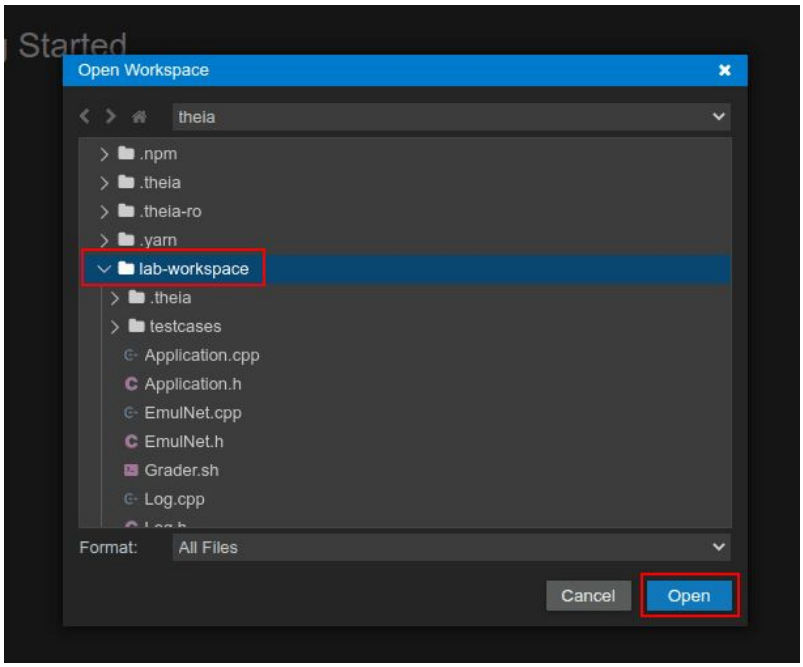
(Beginning Fall 2020) An embedded workspace is provided on the Coursera site for those taking a Coursera section of the course. This workspace provides an IDE similar to (an open-source variant of) VS Code. The starter files and submission script are already present in the environment when you open it, but if the workspace loads in a state with no panels or files visible, you may need to manually reload the workspace and open the file editor pane.

A separate, illustrated document introducing Theia will also be attached to the assignment item on Coursera. Here is a brief overview. Some programming assignments will be split into two items on Coursera: one where you can copy a "submission token" key for use in a later step, as well as view your submission results; and a separate "Ungraded Lab" item, which will allow you to launch the Theia IDE workspace and actually work on the project. Although the Lab may be labeled "Ungraded," there is indeed a way to submit your work for a grade there. We have

provided a script, submit.py, which will ask for your email address and the temporary submission token, in order to submit the assignment instantly.

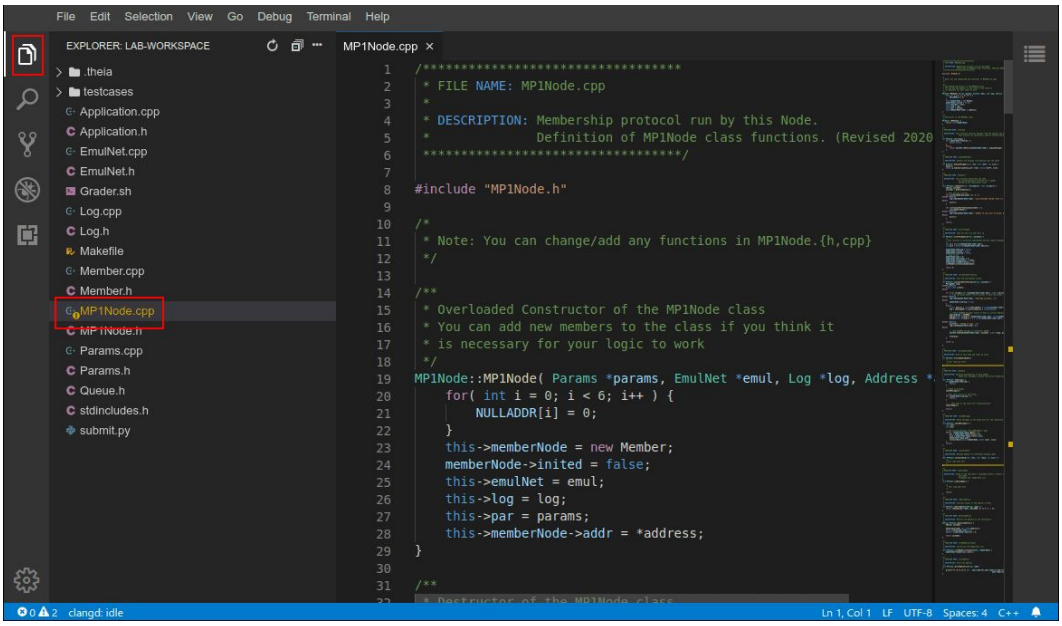
When you first open Theia, it may appear empty. Just click “Open Workspace”. This will open another panel where you can highlight “lab-workspace” and then click “Open.”



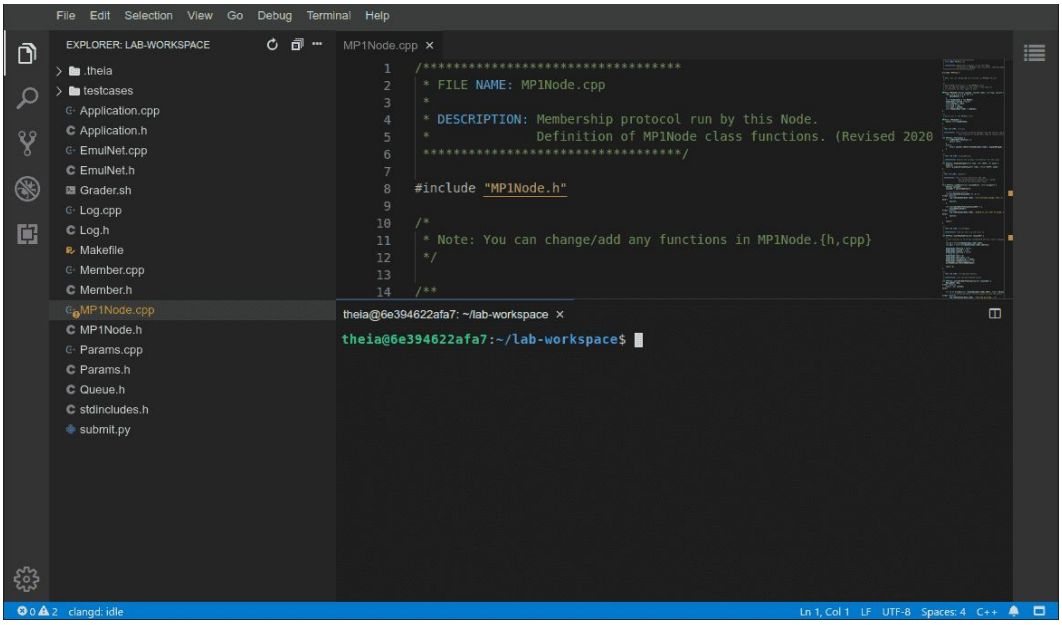


It's **very important** that you use the lab-workspace folder to do your work. This folder will be specially backed up and preserved in the cloud in between your sessions. If you try to use a different folder in the filesystem to work on your project, you may lose your work.

Once you've opened "lab-workspace" as your workspace, then at this point, the starter files have loaded in the background. However, you need to open some viewing panes to see them. Click the file explorer icon in the upper left corner to view a file list, then click a filename to open it for editing. You may need to expand a folder such as "mp2" to see the rest of the files.



You can also open a Bash terminal using Terminal > New Terminal. At this point you'll be ready to work. You may need to *cd* into a subdirectory such as ~/lab-workspace/mp2/ (this may vary).



1.2 Backing Up Your Work

As we mentioned above, it's important for you to work on your MP under the **~/lab-workspace** directory to make sure your files are automatically saved online between sessions. (Note that "~/lab-workspace" is a shorthand notation for the directory "/home/theia/lab-workspace" in the filesystem.) Your files will be saved to the cloud every few minutes, and you should be able to resume from where you left off by returning to the lab. If you worry you might lose your files, or you want to take them offline, then you can download them from the workspace. First, open the Theia terminal, and then type:

```
cd ~/lab-workspace
make clean
zip -r backup.zip *
```

This will clean up the temporary workspace files and bundle everything into a zip file, which you will see in the file list. Then, you can right-click on the backup.zip file and choose Download.

Instructions for testing and submitting your code are given later in this document.

1.3 Academic Integrity

All work in this MP must be individual, i.e., no groups. You can talk with others about the MP specification and concepts surrounding the MP, but you can neither discuss solutions or code, nor share code. Please refer to the academic integrity description on the course website. The usual academic integrity rules of Coursera apply here as well.

2. What the code does, and What to Implement?

Similar to C3 Part 1 MP, we are providing you with a three-layer implementation framework that will allow you to run multiple copies of peers within one process running a single-threaded simulation engine. The three layers are 1) the lower EmulNet (network), 2) middle layer including: MP1Node (membership protocol) and the MP2Node (the key-value store), and 3) the application layer.

We are providing you with an implementation of an emulated network layer (EmulNet). The *Key-Value store* implementation will sit above EmulNet in a peer- to-peer (P2P) layer, but below an App layer as shown in Figure 1. Think of this like a 3 layer protocol stack with App, P2P, and

EmulNet as the three layers (from top to bottom). Each node in the P2P layer is logically divided in two components: *MP1Node* and *MP2Node. MP1Node* runs a membership protocol that you implemented in C3 Part1 MP. *MP2Node,* which you will also implement should support all the KV Store functionalities. At each node, the key-value store talks to the membership protocol and receives from it the membership list. It then uses this to maintain its view of the virtual ring. Periodically, each node engages in the membership protocol to try to bring its membership list up to date.

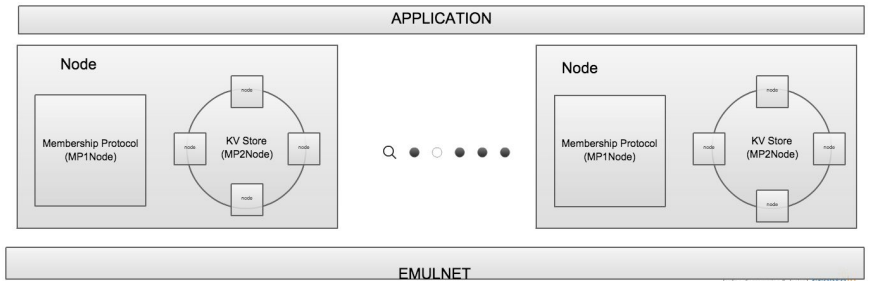


Figure 1. The three layers

Please note that the membership list may be stale at nodes! This models the reality in distributed systems. So, code that you write must be aware of this. Also, when you react to a failure (e.g., by re-replicating a key whose replica failed), make sure that there is no contention among the would-be replicas. Do not over-replicate keys!

Each MP2Node should implement both the client-side as well as the server-side APIs for all the CRUD operations. The application layer chooses a non-faulty node randomly as the client. The same node can be considered as the coordinator. You can assume that the coordinator never crashes. Your Key Value store should accept std::string as key and value.

Some of the important classes in MP2 are:

- HashTable: A class that wraps C++11 std::map. It supports keys and values which are std::string. This has already been implemented and provided to you. You can either use this or have your own implementation of a hash table inside MP2Node.
- Message: This class can be used for message passing among nodes. This has already been implemented and provided to you. You can either use this or have your own implementation in MP2Node.
- Entry: This class can be used to store the value in the key value store. This has already been implemented and provided to you. You can either use this or have your own implementation in MP2Node.

- Node: This class wraps each node's Address and the hash code obtained by consistently hashing the Address. The upcall to MP1Node returns the membership list as a std::vector<Node>.
- MP2Node: This class must implement all the functionalities of a key-value store which include the following:
    - Ring implementation including initial setup and updates based on the membership list obtained from MP1Node
    - Provide interfaces to the key value store
    - Stabilizing the key value store whenever there is a change in membership
    - Client-side CRUD APIs
    - Server-side CRUD APIs

The only file you should change is MP1Node.{cpp,h} and MP2Node.{cpp,h}. Like C3 Part 1 MP, when running a test, your code will be generating a log file (dbg.log) that will be then automatically checked for correctness. Please use the log messages provided in the code to generate the log file.

3. What should I change?

The methods in MP2Node which you should implement are:

- **MP2Node::updateRing**: This function should set up/update the virtual ring after it gets the updated membership list from MP1Node via an upcall. For more information look at the function prologue in MP2Node.cpp.
- **MP2Node::clientCreate, MP2Node::clientRead, MP2Node::clientUpdate, MP2Node::clientDelete** : These function should implement the client-side CRUD interfaces. For more information look at the respective function prologue in MP2Node.cpp.
- **MP2Node::createKeyValue, MP2Node::readKey, MP2Node::updateKeyValue, MP2Node::deletekey**: These functions should implement the server-side CRUD interfaces. For more information look at the respective function prologue in MP2Node.cpp.
- **MP2Node::stabilizationProtocol()**: This function should implement the stabilization protocol that ensures that there are always three replicas of every key in the key value store.

While making your changes the only files you can add changes to are MP2Node.cpp, MP2Node.h, MP1Node.cpp and MP1Node.h. If you delete or modify anything already in these two files, do so at your own risk. Do not change or add to any other file. **However, if**

you do not want to use the classes provided by us and want to add your own classes, add the classes to MP2Node.{cpp,h}.

Note that MP2 makes use of the MP1Node.h and MP1Node.cpp files again. The actual MP1 files distributed with MP2 are just the old MP1 starter files. You may reuse your files from your MP1 solution, or otherwise the submission script will ask if you'd like to have a reference solution for MP1 used to grade MP2 on the remote server. **We'll mention this again below in the submission instructions.**

4. Logging

Use the following functions provided to you in Log.h to log either successful or failed CRUD operations (Remember that this is how the grader will check for correctness of your implementation):

- Log::logCreateSuccess, Log::logReadSuccess, Log::logUpdateSuccess, Log::logDeleteSuccess: Use these functions to log successful CRUD operations.
- Log::logCreateFail, Log::logReadFail, Log::logUpdateFail, Log::logDeleteFail: Use these functions to log failed CRUD operations.

What all nodes should log the messages ?

- All replicas (non-faulty only) should log a success or a fail message for all the CRUD operations
  AND
- If the coordinator gets quorum number of successful replies then it should log a successful message, else it should log a failure message

5. Test cases

The tests include:

- Basic CRUD tests that test if 3 replicas respond
- Single failure followed immediately by operations which should succeed (as quorum can still be reached with 1 failure)
- Multiple failures followed immediately by operations which should fail as quorum cannot be reached
- Failures followed by a time for the system to re-stabilize, followed by operations that should succeed because the key has been re-replicated again at 3 nodes.

For more information about the test look at the comments in Application.cpp starting from Line 232.

6. Submitting

Submit your solution to coursera. Check the due date at the top of this document. Please follow the detailed instructions given below:

- Thoroughly test your solution locally for all the CRUD operations using KVStoreTester.sh provided to you.
- Once you are confident that your solution works then submit your solution to coursera
    - **How do I submit to coursera ?**
        - You will see a python script, 'submit.py'
        - Run the script from the terminal
          `$ python3 submit.py`
        - The script will ask for login details. This includes the email address that you use with Coursera, as well as the submission token that you can generate on the assignment instructions page on the Coursera site. These tokens are single-use and must be generated each time you wish to resubmit.
        - **About reusing your MP1 solution (or not):** MP2 makes use of the solution from MP1. You can either use your own solution from MP1, or otherwise, the submission script will give you the option to have a reference solution for MP1 added in the autograding step on the server, so that any weaknesses in your MP1 solution will not adversely affect your MP2 grade. However, this substitution will only take place on the grading server; you won't directly receive a copy of an MP1 solution.
        - You can check the score of your submission. Please note that it takes a while (sometimes a few hours) for your submission to be graded. Please be patient!

7. If you finish your MP early…

The MP has been designed for use of porting to a real distributed system. If you finish the MP early and all your tests are passing, you may want to look into making your MP code run on a truly distributed system. Start by changing the Emulnet layer, and then perhaps using multithreading. You can also change any of the underlying classes and implementations if you think that will make the code more efficient. If you can get a version working across at least 3 machines, then you have a fully working key-value store! (There will not be extra credit for this portion, but it's incredibly useful for you to tell your friends, relatives, and interviewers that you've built a real working key-value store!)

All the best !