2020 FALL

EE 324, Programming Assignment #4

Implementing Distributed Computing Framework with gRPC

1. Overview

In this assignment, you need to design and implement a distributed computing framework that translates source keywords to target values in parallel. By leveraging multiple machines in a distributed environment, better performance can be provided. When you implement a distributed computing framework, your framework should provide two core concepts, **Parallelism** and **Remote Procedure Call (RPC)**. Parallel processing saves as much time as the many resources you work with. RPC hides all of the network code into stub functions, so application programs don't have to worry about socket-level details.

Your framework follows 'master-slave architecture', which has two types of nodes; 1) super node and 2) child node. Super nodes accept the client's request and split a large file into smaller files. And then, they send them to child nodes. Child nodes receive small files and translates them with the key-value information communicated with the remote database (DB) server. Super nodes can communicate each other, but child nodes only communicate with connected super node. Figure 1 shows the topology of the framework.

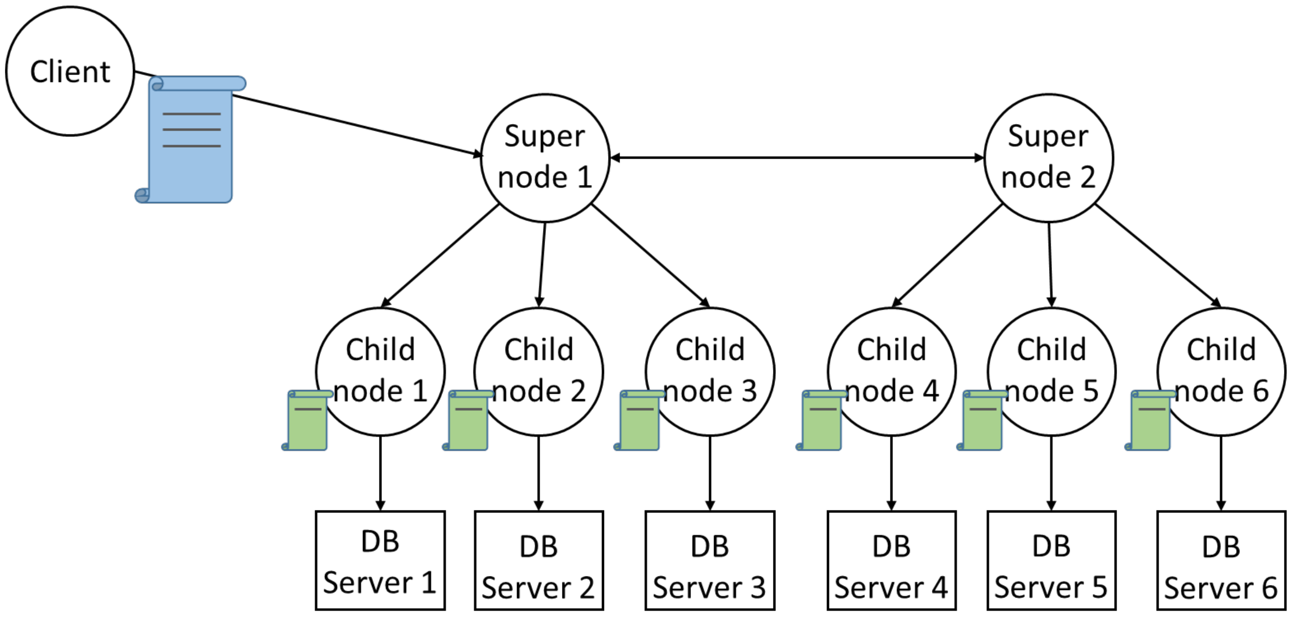

Figure 1. Topology of distributed computing framework

2. Design requirements

There are four design requirements in this assignment; 1) Parallelism, 2) gRPC, 3) DB miss handling, 4) Cache management.

1) Parallelism

- When a super node sends small files to child nodes, they have to process the files in parallel, not in sequential manner.
- If only one child node is working at a time, it makes no sense to split large files into smaller files and do distributed computing.
- You can achieve parallelism with one thread asynchronous I/O or multi-thread synchronous I/O. There is no specific design requirement for this. It depends on your design choice.

2) gRPC

- gRPC is an open source remote procedure call (RPC) system initially developed at Google in 2015.
- In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.
- gRPC uses protocol buffers as the Interface Definition Language (IDL) for describing both the service interface and the structure of the payload messages.
- Through the IDL, gRPC provides simple service definition and works across languages.
- Requirements
  1. Communication between nodes (Super node <-> Super node and Super node <-> Child node) should be implemented by gRPC, not socket-programming.
  2. Your child node should communicate with the provided python DB according to the given gRPC IDL.

3) DB miss handling

- Six database (DB) servers are provided on eelab machines.
- Each DB server has their own database and each database does not overlap anything.

- A child node communicates with only one DB server.
- DB hit and DB miss
  - 1-1. If a key-value mapping information for a keyword to be translated by a child node is in the communicating DB, a DB hit occurs. The child node continues translation.
  - 1-2. If a key-value mapping information for a keyword to be translated by a child node is not in the communicating DB, a DB miss occurs. In this case, the child node requests the value of the keyword to the super node.
- DB miss handling
  1. The requested super node broadcasts requests for the value of the keyword to other connected child nodes, first.

  1-1. If any child node finds the value of the keyword in its connected DB, it passes it to the requested child node through the super node.

  2. If it is not found in the connected child nodes, then it asks to another super node.

  2-1. If any child node that is connected to another super node finds the value of the keyword in its connected DB, it passes the value to the requested child node through the two super nodes.

4) Cache management

- It is inefficient to repeatedly ask the DB for the same keyword.
- A super node and a child node should provide key-value mapping cache for requested keywords.
- Child node cache
  1. Child node stores both DB hit and DB miss keywords mapping in its own cache.
  2. Child node cache size is smaller than super node cache.
- Super node cache
  1. A super node stores the keywords mapping from the DB miss in its own cache.
- Due to the size limit of cache, you should design cache replacement policy for your cache such as FIFO (First In First Out), LRU (Least Recently Used), etc.

3. Elements specification

There are four elements in this assignment; 1) Client, 2) Super node, 3) Child node, 4) DB server. You need to implement and submit element 1, 2, 3. Element 4 is provided.

1) Client

- A client sends a source keywords file to any super node.
- Your client runs with the input file, super node's ip address and port (i.e. **"./client input_file eelab5.kaist.ac.kr 12345"**)
- Your client should send up to **4MB** file properly.
- **Source keywords file specification**
  - Each source keyword consists of only alphanumeric characters.
  - Keywords in the source file must be separated by non-alphanumeric characters (i.e. punctuation marks, white space, newline, tab, etc.).
  - You should only change the keywords to the target values. Non-alphanumeric characters should not be changed (e.g. "AAA, CCCC!" → "EE324, network!").
  - We uploaded example DBs, example test cases and the translated results, so please refer to the examples carefully.

2) Super node

- A super node receives a client's request.
- It sends half of the file to another super node.
- It splits the remaining file into the number of child nodes and sends them to the child nodes.
- In your distributed computing framework, there will be **"two"** super nodes.
- Your first super node runs with its port number for client, a port number for gRPC and its child nodes' ip and port. (i.e. **"./super 12345 [gRPC port] [child1's ip_address]:[child1's port] [child2's ip_address]:[child2's port] [child3's ip_address]:[child3's port] …"**)
- The number of child nodes to be connected are determined by how many child node information is written in the argument.
- Your second super node runs with its port number for client, a port number for gRPC, another super node's ip, port with '-s' flag, and its child nodes' ip, port. (i.e. **"./super 12346 [gRPC port] -s [another super node's ip_address]:[another super node's port] [child4's ip_address]:[child4's port] [child5's ip_address]:[child5's port] [child6's ip_address]:[child6's port] …"**)
- Through the '-s' flag, the second super node communicates with the first super node.
- The maximum number of child nodes is five.
- Communication between client and super node is socket programming. You can

design anyway you want.

- Communication between super nodes and between super node and child node must be done through gRPC.
- A super node has limited cache size, **30KB**.

3) Child node

- A child node receives a small file from a super node.
- It translates the file through the information communicated with the DB.
- A child node communicates with one DB server.
- Communication between a child node and python DB is done according to the given gRPC IDL.
- If there is a DB miss, it asks about the translated keyword to connected super node.
- A child node has limited cache size, **10KB**.
- Your child node runs with its port number, super node's ip and port for gRPC, DB server's ip and port. (i.e. **"./child 50051 [super node's ip_address]:[super node's gRPC port] [DB server's ip_address]:[DB server's port]"**)

4) DB server

- A DB server code is provided (**python_DB.py**)
- A gRPC IDL is also provided to communicate with the python DB (**assign4.proto**).
- Before running the python DB, you need to run a script to create gRPC related modules (**python_build.sh**).
- You can test your framework by changing the database information of the DB server.
- You can run the DB server with the command **"python python_DB.py"**
- Also, you can test your framework with provided six DB servers.
- Three DB servers' ip is **"eelab5.kaist.ac.kr"** and the other three servers' ip is **"eelab6.kaist.ac.kr"**.
- Each server's port number is **"50061", "50062", "50063"**.

4. Tips for gRPC

We could not write all the information about gRPC in this document. gRPC is very popular library and there is well-written document in their homepage.

Here are some useful links

gRPC homepage link: https://grpc.io/

Introduction to gRPC: https://grpc.io/docs/what-is-grpc/introduction/

Cpp gRPC examples: https://grpc.io/docs/languages/cpp/

NOTE: We strongly recommend you to follow all of the tutorials above before starting programming. Tutorials will tell you about how to use gRPC library correctly and how to write and use CMakeLists.txt.

NOTE: gRPC only supports object-oriented languages. So, there is no C library in gRPC. If you are only familiar with C language, you can do C-style programming inside the C++ file (i.e. *.cc, *.cpp).

5. Instruction for Submission

- You should upload your code on GitLab. (You don't need to submit any files on KLMS.) https://gitlab.ee324.kaist.ac.kr/assignments/assignment-4
- TAs will download your project when the due is over.
- **Fork the GitLab repository to your group and clone the forked repository to your local machine before you start.**
  - You need to make a group named after your student id (e.g., 20201234) before forking.
  - Your forked repository should be in the form of (if your ID is 20201234):
  - https://gitlab.ee324.kaist.ac.kr/20201234/assignment-4
  - Make sure you are not directly cloning from the assignment repository.
- In the GitLab, there are seven files. **You have to add two more IDL files for gRPC requirements.** You may add more files as you wish. However, the total size of the submission must not exceed 10MB. Please do not add unnecessary files to your commit (e.g., log files) (NOTE: You should write client, super, child programs with cpp).
  - README.md
  - CMakeLists.txt
  - client.cc, super.cc, child.cc
  - assign4.proto

- python_DB.py

- Make sure that your CMakeLists.txt correctly generates Makefile for your source code. The CMakeLists.txt must reside in the root of the repository.

- In the README.md:
  - You should put your name, student ID.
  - Write your name and date on the ethics oath.
  - In addition, briefly describe how you designed and implemented your programs.
  - Describe whatever help (if any) you received from others while doing the assignment, and write the names of any individuals with whom you collaborated, as prescribed by the course Policy web page.

- Write concise comments in the source code to understand your program.

IMPORTANT 1: Please strictly follow the above structure and name policy; if not, TAs will not evaluate your program.

IMPORTANT 2: Please make sure that there is no compile and execution error. TAs will not give you any score in case of the problems.

6. Grading (Total 120 points)

1) Functionality – 100 points

I. Does distributed computing framework provide parallelism? – 20 points

II. Do elements communicate via gRPC? – 40 points

  i. Communication between super nodes – 10 points

  ii. Communication between super node and child node – 20 points

  iii. Communication between child node and python DB server – 10 points

III. Does distributed computing framework handle DB miss properly? – 30 points

IV. Does super node and child node provide cache management? – 10 points

2) Performance – 20 points (Max for the best performance)

I. We will send maximum 4MB file to your distributed computing framework with our provided DB servers. We will measure the time it takes for the request to complete from the client.

II. The grades for this performance part are scored through a relative evaluation manner. Students with better performance will get higher scores than those who do not. (The policy will be finalized after grading, based on statistics)

III. Metric: “second” (the time it takes to complete the request measured by the client).

7. Test Environment

- Language: C/C++
- **NOTE:** We will not consider your compilation and execution problems due to the different OS versions. (defaults: eelab machine, testing machine)

8. Due Date

- **11:59 PM, Dec. 21, 2020 (Mon)**
- **There is no late submission in this assignment. Above due date is the hard deadline.**
- We will not receive additional submissions after the hard deadline.
- Exceptions: documented medical/personal emergency.
- Please DO NOT e-mail Prof. Han or TAs to submit your assignments.

9. Plagiarism

- You can discuss with your colleagues, but you should turn in your own programs
  - ✓ Copy and Paste
    - Will run plagiarism detection on source code.
    - “Copy and paste” codes will get severely penalized
    - If detected, 0 points for all assignments (both providers and consumers)
    - But you will have a chance to defend yourself

10. Questions?

- Please use PIAZZA Q&A board to ask any questions.
- Or you can ask to TAs at official office hour

- Please carefully read specification described at PA #4 document.