Final Project

Distributed Systems

CT30A3401

Vili Raunola

#######

12.04.2022

Contents

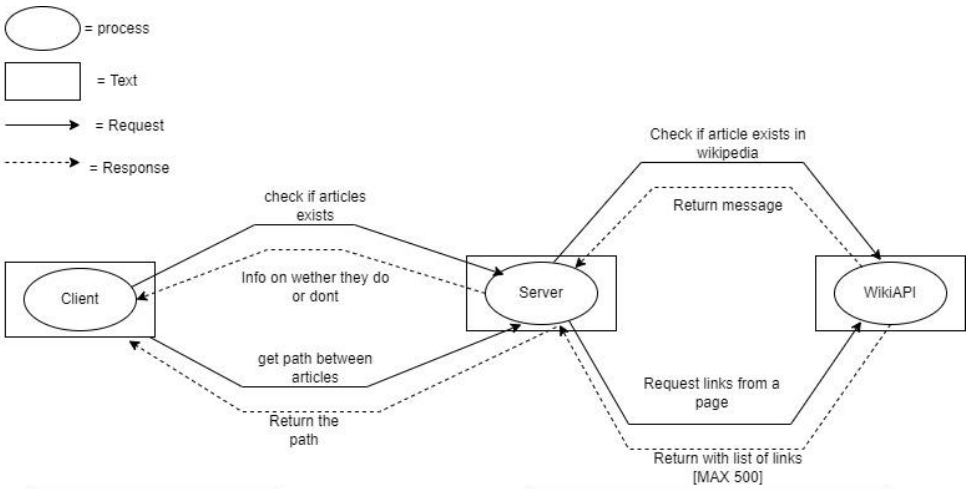
1. Introduction

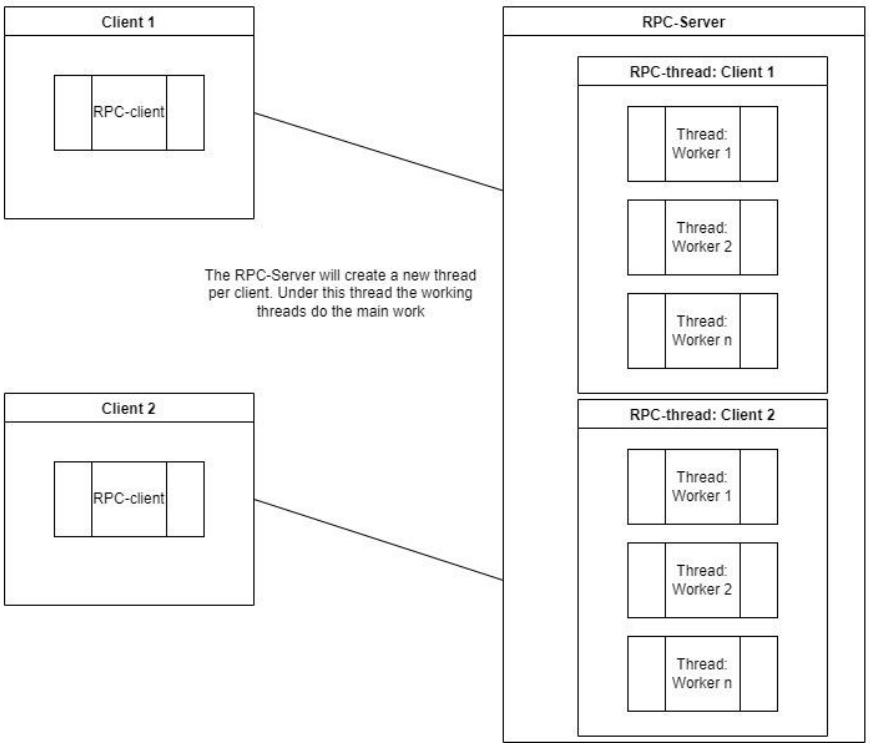
The main idea in this project was to implement a general search tree. The tree is built by working threads. Once the article that is being searched is found, building the tree is stopped. After this we have the last leaf that is the article that we were looking for. Now we can backtrack the path through the tree thus giving us the links from start to finish.

The way I thought Remote Procedure Calls (RPC) is that all the heavy lifting is done behind the RPC call. Due to this thought process, I implemented both the API calls to the Wikipedia and creating the tree on the server side. The clients only job is to process the response messages from the server and to display the menu to the user. The client also provokes the RPC calls. Each client gets their own thread on the server so concurrent execution is enabled.

What I assumed through the implementation was that the Rapid Access Memory on the server would be infinite. This might have been a bad call since when finding the path between articles that have more that 4 layers meaning that there might be millions and millions of links to go through. When there are millions of links they all will be inserted to the search tree which lives in RAM. A better approach would have been to store the tree to a file.

2. System Overview





The system overview shows all the possible communications between the processes. The client can ask from the server if the end and start article exist and it can also ask to start the search process. The server will respond accordingly to the client either with the result or why the task has failed.

The server can on the other hand ask from the Wikipedia API to get an article. From this response the server will deduct if the article exists in Wikipedia. The server can also request links from a given page through the API. The limit for the links per response is 500. If the page has more than 500 links the return message has information on this and a new request will be made to the Wikipedia API until all of the links are received.

3. Requirements Implemented

Functional requirements:

- All of the required requirements are taken into account.
- Has a client and RPC-server
- Client can start the search between two Wikipedia articles
- Server will handle the actual shortest path and give the results back to client
- Server will get the links form a Wikipedia page through the Wikipedia API
- Server will check that both articles are real before initiating the search
- When receiving the links from a page, they are accepted if they are real pages aka. the namespace is the main namespace
- You can allow the maximum depth of the search tree by changing a Boolean to True or False. You can also specify the maximum depth by changing an integer in the global variables

Non-Functional requirements:

- The performance is considered. You can change the amount of working threads but using too many will cause the Wikipedia API to limit us thus reducing our performance. The execution time is between 2 and 300 seconds depending on the path that is being searched. This is due to the amount of links on each page and the distance between two links. If the links are 5 steps apart from each other there will be millions of links. If there would be around 500 links per page then there would be 500^(5+1) nodes which to search from which is around 1.6 E16 nodes. Due to the uncertainty of the links per page the search might take from few seconds to days or months. Most of the time the search is in under two minutes. The main limiting factor is the time that takes to get the links from the Wikipedia API. If there wouldn't be any limitation on the amount of requests from the server the search would be significantly faster.
- The server is accessible to other users but now only one user can use one service at once.
- Fault tolerance is taken into account in the case when Wikipedia API blocks us. We put the leaf back to queue if the Wikipedia API limited our requests on that leaf.
- Is also scalable meaning that other servers could be deployed and clients could access them the same way as they do the current one
- Is usable for the user. Easy to understand UI in the command line.

- Uses multiple working threads to build the search tree. They communicate through a main thread that has a queue that keeps track of the leaves from which the links need to be searched.
- Error check on the client. No wrong argument will crash the service

4. Development Approach

The language used in the project was Python. It was selected since it provides easy to learn syntax, RPC-server/client functionality and it was used on the previous assignments so it is in fresh memory. The API is called with get request from the official Wikipedia API. It was selected because there is plenty of documentation on using it. Sometimes it does return links that aren't on the actual Wikipedia article which will affect the results. The server uses simple threaded XML RPC server. It was selected since it creates an own thread per connection so each user can use the functionalities concurrently. Queue is used to distribute the work between the threads. It was selected since it provides a thread safe way to handle the information flow between the threads.

Due to threading the path isn't always the same. The path is always the shortest based on the level. This is due to the fact that each link is put on a queue from which threads take it to analysis. They retrieve links from that page and put these links back to the queue. This method creates the three one layer at a time which is like breadth first search. But due to threading some links are processed before others. This is due to the fact that each thread gets some time on the processor which we can't affect. This will result on different paths on same input if there are other possible paths to find.

If the API tires to limit our requests we get an error and the and no information was gained on that link. We can't just discard that link but we must try again. This means that other threads will get ahead of us thus finding another path.

5. Distributed system properties

- Heterogeneity
    - The components can operate without many limitations. The only limitation is that the client and server machine must be able to run Python for the programs to run. They both must also have an internet connection or be connected to the same network. The server must be connected to the internet so it can access the Wikipedia API. The programming language used in the Wikipedia API doesn't matter in our case. The only thing that we are concerned about is that the API is online and can be reached through HTTP GET.
- Openness
    - The server and client can share and does share resources. Adding new functions to allow other resource sharing is also made possible. Only thing needed is to create a new RPC-method for the client to call.
- Security

    - The check for depth of the search tree is implemented so this service can't be used to attack Wikipedia API. In theory this could be used to create a DDOS attack against the API if there isn't links connecting the articles so the search tree would be built for ever meaning every thread would make calls to the API.
- Scalability
    - The server can take multiple clients and handles each client in its own thread. Adding a better CPU and more RAM would make allow the server to handle much more clients at once. It would be possible to deploy multiple servers but then there should be a coordinator that handles the incoming clients and distributes them to all the servers evenly.
- Failure handling
    - The first check is to make sure that the pages for which the path is searched for exists on Wikipedia. On the client side there are checks to make sure that the client can't give any inputs that would result in a crash. Server also checks for the API calls to make sure they are received properly. Also, the client crashing doesn't affect the other systems execution.
- Transparency
    - The transparency is implemented so that the client has no idea that the server does the searching and communicates with the Wikipedia API and returns the results. The whole process seems like it is happening on the client's machine.

6. Notes

Only copy-pasted code is from the Wikipedia API and RPC server. Sources used in the client and server programs (these can also be found on the source code on the places to which they refer to):

https://stackoverflow.com/questions/53621682/multi-threaded-xml-rpc-python3-7-1

https://stackoverflow.com/questions/5033222/is-simplexmlrpcserver-single-threaded

https://www.youtube.com/watch?v=4r_XR9fUPhQ

https://www.mediawiki.org/wiki/API:Links

https://stackoverflow.com/questions/40579942/what-do-the-parameters-in-wikipedia-api-response-mean

https://stackoverflow.com/questions/55529319/how-to-create-multiple-threads-dynamically-in-python

https://stackoverflow.com/questions/66425508/what-is-the-meaning-of-for-in-range

https://stackoverflow.com/questions/6517953/clear-all-items-from-the-queue

https://docs.python.org/3/library/xmlrpc.client.html

https://stackoverflow.com/questions/3144898/python-question-about-time-spent