Final Project

Distributed Systems

CT30A3401

Jani Heinikoski

0541122

22.04.2022

Contents

1. Introduction

The project’s end-goal was to create a distributed system that finds the shortest path between two Wikipedia articles purely through links that are found on subsequent pages. Furthermore, a simple CLI-based client program to use the system was developed. I chose a simple client-server architecture. However, the server program is intended for one client at a time. This is purely because Wikipedia’s API can’t handle many concurrent requests from a single IP. I assumed that performance for a single client is preferred in this case (was not mentioned in the instructions) however you could limit the number of workers per request to 1-2 to handle ~10 concurrent clients but each client will have extremely poor performance. In conclusion, note that you can add more cores and concurrent workers, but you will most likely get an IP ban after ~16 concurrent workers with my program. Finally, I assumed that finding the complete step-by-step route between the two articles is not necessary since this course is about distributed systems and not data structures.

2. System Overview

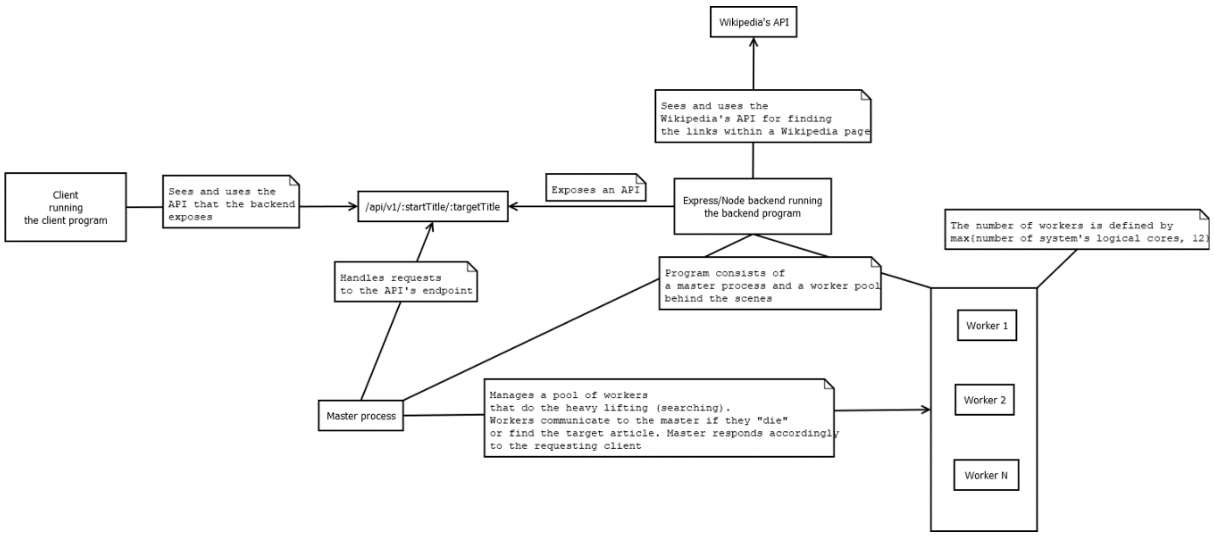
Wikipedia's API
Sees and uses the Wikipedia's API for finding the links within a Wikipedia page
Client running the client program
Sees and uses the API that the backend exposes
/api/v1/:startTitle/:targetTitle
Exposes an API
Express/Node backend running the backend program
The number of workers is defined by max(number of system's logical cores, 12)
Handles requests to the API's endpoint
Program consists of a master process and a worker pool behind the scenes
Worker 1
Worker 2
Worker N
Master process
Manages a pool of workers that do the heavy lifting (searching). Workers communicate to the master if they "die" or find the target article. Master responds accordingly to the requesting client

Figure 1: High level depiction of the system’s architecture

3. Requirements Implemented

Functional requirements

- End user has a client CLI-program which uses a separate distributed server application.
- The performance is appropriate considering the complexity and 3rd party limitations. Simply put, the system works fast (under a minute) when the articles are one level apart (i.e., the length of the shortest path between the articles is two).
- The system is reliable.
- End user gets the final step of the path (what was the last article before finding the target).

- End user gets a time estimation on the performance of the search. The values can be used to find out relative path lengths between different paths.

Non-functional requirements

- Invalid requests from the client are dropped, otherwise cyber security is not considered.
- The system finds the shortest path as a breadth first search of a cyclic graph by dividing the load between multiple child processes called workers.
- The system is somewhat vertically scalable, although strongly limited by the 3rd party Wikipedia API used.
- The system is horizontally scalable (backend program can be run on multiple servers and a load balancer can be implemented between them to handle more clients).
- The system is fairly fault tolerant; all exceptions are handled accordingly, the system has a thirty-minute timeout (i.e., if the search lasts over 30 minutes, it stops the search and tells the client no path was found), if a single worker or the client “dies” the whole system does not crash.
- Performance is achieved by evenly distributing the load between the child workers.
- The system is interoperable as the server offers a uniform REST-API which can be used by other software as well.
- The system works on most modern operating systems, Node JS, NPM and few NPM packages (express, node-fetch, express-validator, are required).

4. Development Approach

I used NodeJS for creating both the client and the server-side applications. It suits the project requirements very well because Node can create HTTP servers very quickly. Node also works in an asynchronous non-blocking manner which enables having more workers than logical cores (workers that wait for an asynchronous I/O-operation to finish give control flow back to the caller and thus enable doing other stuff while some workers wait for I/O). NodeJS also supports forking and thus true concurrent programming through a built-in module called cluster. Express was also used to create a simple REST-API for the client program.

5. Distributed system properties

The backend which can be considered a distributed system conforms to many common distributed system properties. First, the system can be considered distributed because it consists of multiple coherent components, that is workers. They work together to achieve the same goal. The workers are controlled by a single master process which also handles the communication between the client and the server. Furthermore, a third-party API is utilized.

The system is transparent. Even though the system is distributed, it appears as a single system which can be accessed through the REST-API. In other words, the client does not actually know or is able to find out that the system works concurrently unless they have access to the source code. Concurrency can also be considered a property of a distributed system.

The system can be considered fault tolerant. If a single worker dies, the whole system does not stop working. Furthermore, loss of Internet connection between the client and the server does not affect the whole system.

The system is open, that is the system provides a uniform, well-defined, standard REST API. This can be used by other pieces of software (if for example someone wants to create their own client).

The system is vertically and horizontally **somewhat** scalable. Vertical scalability is only limited by the third part Wikipedia API, but horizontal scaling is technically infinite, however not feasible. Basically, you would have to have a separate server for each client if you do not want to limit performance. Furthermore, some kind of a load balancer should be implemented if horizontal scaling is desired.

6. Notes

All the sources that I have used are mentioned at the top of the source code files. I have also left comments indicating in which functions I have used the sources. The application has been tested with Node version 17.6.0 on a x86-64 Windows 10 machine. The system overview picture can be found as a separate picture on the code repository as well. Installation instructions are included in the code repository.