Project Details

Computer Systems Performance, Spring 2021

Contact: Zsolt István, zsis@itu.dk

The goal of the project is *1)* to implement a multi-threaded application and benchmark it, *2)* to analyze the measurements and build a model of the system and *3)* to determine the bottleneck of the system in terms of performance.

The project is based on a simple Word Count application that has the goal of turning an input document into a series of tuples in the form <word, count>, corresponding to each distinct word in the document and its number of occurrences. In large machine learning pipelines, this operation is often used to determine the similarity of two documents or to classify them into different categories based on their content. In our project, however, we study this application without plugging it into a larger pipeline.

You will be given a skeleton of the project, with a simplistic, single-threaded implementation of a Word Count server and clients that connect to it over TCP sockets (run on two virtual machines in the cloud). Your task will be to extend this application with multi-threaded functionality and to instrument the code to provide more insight into where time is spent during processing.

The project is carried out individually. It is important to note that the absolute performance of the system you implement plays a smaller role in the final grade than your understanding of its behavior and the quality of the report and slides.

Deadlines

You will have to **submit the final project report, as well as your code (as a single compressed archive), by 31st of May 2021, 14:00** (Copenhagen time). This is the material that will be taken into account for grading.

It is **obligatory to take part in the project progress presentations on the 27th of April 2021**. Even though your presentation will not be graded, we strongly encourage you to treat it as the project deadline and to email us (or show us during Office Hours) your presentation slides in advance before the 26th of April 2021 for feedback. **You will have to submit the slides in LearnIt before 14:00 on 27th of April 2021.** We recommend that you start writing your report immediately after this presentation.

Deliverables

Submitting all deliverables is done through LearnIT. If you have questions, email us or as ask us on Discord!

The report is to be based on the outline provided on the seminar website. You can use any text editor and plotting program of your choice, but the font size shall not be smaller than 11 points and the total length of the document cannot exceed 8 pages. The submission format is PDF.

To submit the code of the project, please put all relevant files into a zip archive and attach them to the email. In this zip, please include a folder that holds the experimental data that you used to generate the plots in the report. If the resulting file size is larger than what you can attach to an email, keep the most important files in the zip and omit the others (but don't delete them from your computer in case we need to review them together).

The slides should follow the template posted on the seminar website (don't add or remove slides).

Project Skeleton

You will be provided with a working starting point of the project that consists of simple benchmarking clients and a single-threaded server implementation.

Clients

The clients are single-threaded. They generate their own test document by reading from two possible input files (provided with the skeleton) starting at a random offset. The length of a document is a start-up parameter. Once the document has been generated, the client will start sending work to the server in a closed loop (the name and port of the server is also a parameter). It will send the document N times (provided as a runtime parameter), then exit.

While running, the clients will print statistics once every second, followed by a final statistics when they finish. The final statistics includes a listing of the percentiles of the response times measured by the client.

Server

The Word Count application is implemented as a single server that uses Java NIO to accept TCP connections and to read data from many the clients. Its implementation consists of a main loop that reads data from the sockets as it becomes available, and enqueues it in client-specific buffers until the text of an entire document has been received. The server uses a special separator character to recognize the end of the document (the client will ensure that the separator is only present at the end of the document).

Once there is a document in the buffer of a client, the server will convert it to lower case and remove any non 'a'-'z' characters and spaces. It then performs a word count on the resulting "stripped down" document. The resulting table of <word,count> pairs is serialized and sent back to the client.

Communication protocol

The client and the server communicate over TCP sockets, using a very simple protocol:

1. The server opens a port it will listen on and accepts incoming connections.
2. The client sends a document to the server. The document is encoded as a string (new line characters are removed), with a special separator character at the end (this is defined in code, but by default is '$').
3. The server processes the document once it has been received in its entirety, and sends back the <word, count> pairs to the client in a simple text-based format (word1,count1,word2,count2,…). The end of the response is signaled using a newline character ('\n').
4. The client only parses the answer if it is running in debug mode, otherwise it waits until the end of line character has been received and discards it immediately. This ensures that processing in the client is not a performance bottleneck.

Features to Implement

You have to extend the system with the following features:

1. **Cleaning of document from special words**: Our input will be (fragments of) HTML documents that contain both tags (text between a < and >) and the actual words we are interested in. Your task is to extend the word count logic of the skeleton to disregard tags (even if it spans multiple words, e.g., <a href="abcd" style="xyz"/>) and only count the remaining words (e.g. only keep "foo" from "<p><b>foo</p></b">).
   There is one exception to this rule: the text inside a *title* attribute of an anchor has to be kept (e.g., <a href="xyz" title="FooBar">foobar</a> → FooBar foobar)
2. **Add fine-grained statistics gathering inside the server** and print out the aggregate statistics once all clients have disconnected (that is, the experiment is finished). The instrumentation of the code should measure:
   - time spent until the entire document has been received
   - time spent cleaning the document (removing tags)
   - time spent performing the word count
   - time spent serializing the results

In terms of final aggregate printout, the average and the percentiles (0-100 in 1% steps) are needed. It is not necessary to separate statistics per client, but you might consider using data structures dedicated to each client before merging the results to avoid synchronization overheads.

3. **Extend the server into a multi-threaded implementation** where there are W worker threads performing the word count (inter-client parallelism).

Command Line Interface and Building

When extending the project skeleton, please leave the command line arguments of both the client and the server unchanged, and make sure that these arguments change the behavior of your code as expected.

Do not add dependencies to you project (no external JAR files). You may use any program, scripting language, etc., to process logs however – just make sure to explain the steps you took in the report.

For building and running the programs, please use the ANT build script provided (feel free to add more targets but keep the original ones as well). This will allow us to quickly test your application and eventually replicate the experiments you carried out.