CS433/533 Homework Assignment Three: Network Servers and Server Performance Evaluation (Part 2)

This assignment gives you a chance (Part 1) to better understand HTTP design, including its protocol and the processing at the server (Part 1). Another key goal of this assignment is give you a chance (Part 2) to become more familiar with concurrent network programming, covering topics including threads, synchronization, wait/notify (monitor), asynchronous I/O, and benchmarking.

Due:

- Part 2: 5:00 PM, Sunday Oct. 28, 2018 (after October recess).
- Competition: Among all valid servers. Bonus: 25% bonus and a small prize (A Rasberry Pi 3 Kit).

Part 2a: Concurrent HTTP Servers using Threads

As we covered in class, a key approach to controlling the overhead of threads is to use a thread pool. We covered two designs: (1) shared welcome socket; and (2) a shared queue with wait and notitify. Please implement the following three threadpool servers:

- thread pool with service threads competing on welcome socket;
- thread pool with busy wait;
- thread pool with a shared queue and suspension;

For each design, your server needs to read from the configuration file the pool size:

```
ThreadPoolSize <number of threads>
```

Part 2b: Async Server: Multiplexed, Nonblocking Server (Reactive Server)

- We recommend that the software structure of your asynchronous server be based on v2 of the `EchoServer` that we discussed in class. You need to write a handler for the particular protocol. You can feel free to modify the structure if you see any way to improve it (fix error handling, etc). You need to document your changes.
- The server should have a timeout thread. Upon accepting a new connection, the accept handler should register a timeout event with the timeout thread with a callback function. The timeout value is specified by `IncompleteTimeout <timeout in seconds>`. The default timeout value is 3 seconds. If the connection does not give a complete request to the server *approximately* within timeout from the time of being accepted, the server should disconnect the connection. Note that the timeout monitoring thread should not directly close a channel that the dispatcher thread is still monitoring (why?). You need to think very carefully about the exact details of the interaction between these two threads, propose a software design, and implement it.
- Design (implementation not required) question: If your server is required to support connection keep alive, please describe briefly how your code structure may look like?
- Design (implementation not required) question: If your server is required to support HTTP/2, please describe briefly how your code structure may look like?

Part 2c: Async Server: Proactive Server

In this part, instead of using the select structure, you use AsynchornousServerSocketChannel and AsynchornousSocketChannel design, based on Future/Listener to implement the same functionality as in 2b.

Part 2d: Comparison of Designs

A great way to learn about your design is to compare with other designs. You need to read the docuemnts or code of three related frameworks: xsocket, netty, and nginx.

Part 2d(1): Comparison with xsocket

Although xsocket is no longer under active development, it provides a design alternative. Please read the source code and document of x-Socket, a high performance software library for reusable, asynchronous I/O servers. Please discuss in your report the following questions (please refer to the specific location when you refer to its document or source code:

- How many dispatchers does x-Socket allow? If multiple, how do the dispatchers share workload?
- What is the basic flow of a dispatcher thread?
- What is the calling sequence until the `onData` method of EchoHandler (see `EchoHandler, EchoServer`, and `EchoServerTest`) is invoked? Please check this link for testing code: http://sourceforge.net/p/xsocket/code/HEAD/tree/xsocket/core/trunk/src/test/java/org/xsocket/connection/
- How does x-Socket implement Idle timeout of a connection?
- Please give an example of how the library does testing (see http://sourceforge.net/p/xsocket/code/HEAD/tree/xsocket/core/trunk/src/test/java/org/xsocket/connection/EchoServerTest.java for an example). Please describe how you may test your server with idle timeout?

Part 2d(2): Comparison with Netty

Netty is another Java async IO framework used by many; see for example use cases. Please read Netty user's guide and answer the following questions:

- Netty provides multiple event loop implementations. In a typical server channel setting, two event loop groups are created, with one typically called the boss group and the second worker group. What are they? How does Netty achieve synchronization among them?
- Method calls such as bind return ChannelFuture. Please describe how one may implement the sync method of a future.
- Instead of using ByteBuffer, Netty introduces a data structure called ByteBuf. Please give one key difference between ByteBuffer and ByteBuf.
- A major novel, interesting feature of Netty which we did not cover in class is ChannelPipeline. A pipeline may consist of a list of ChannelHander. Compare HTTP Hello World Server and HTTP Snoop Server, what are the handlers that each includes?
- Please scan Netty implementation and give a high-level description of how ChannelPipeline is implemented.

Part 2d(3): Comparison with nginx

nginx is considered to be the highest performance HTTP server. Please discuss in your report the following questions (please refer to the specific location when you refer to its document or source code:

- # questions to be finalized

Part 2e: Performance Benchmarking

One important computer systems skill is to evaluate the performance of design alternatives. In this assignment, we conduct performance evaluation of the alternatives:

- To conduct the testing, you will need to setup the DocumentRoot at the server. It is highly recommended that you generate a number of files of different sizes under DocumentRoot named such as file1.html, file2.html, ..., file1000.html. If you download gen.tar, and untar it (tar -xvf gen.tar), you will see a directory named doc-root and a directory named request-patterns.

  To compare the performance with Apache, we will use the department zoo Apache server. We will use /home/httpd/html/zoo/classes/cs433/web/www-root to store testing files. Suppose we want to fetch /home/httpd/html/zoo/classes/cs433/web/www-root/html-small/doc1.html. To use the department Apache server, since the department server has set DocumentRoot as /home/httpd/html/zoo, the URL should be: http://zoo.cs.yale.edu/classes/cs433/web/www-root/html-small/doc1.html

To use your server, suppose you set the DocumentRoot as /home/httpd/html/zoo/classes/cs433/web/www-root, and you run your server on cicada.cs.yale.edu at port 9876. Then the URL is:
http://cicada.cs.yale.edu:9876/html-small/doc1.html

- For the test, you will need to generate a request file for the client. The request pattern can have a major impact on your server performance (how requests repeat). The TA will use a Pareto distribution to generate request patterns to test your server. You can write a simple Java program or script to generate the request.

- You should vary the client parallel (see Client command line above) with a reasonable increment schedule (e.g., 1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, ...). A reasonable test time is 60 to 120 seconds. You can write a simple script to automate this task.

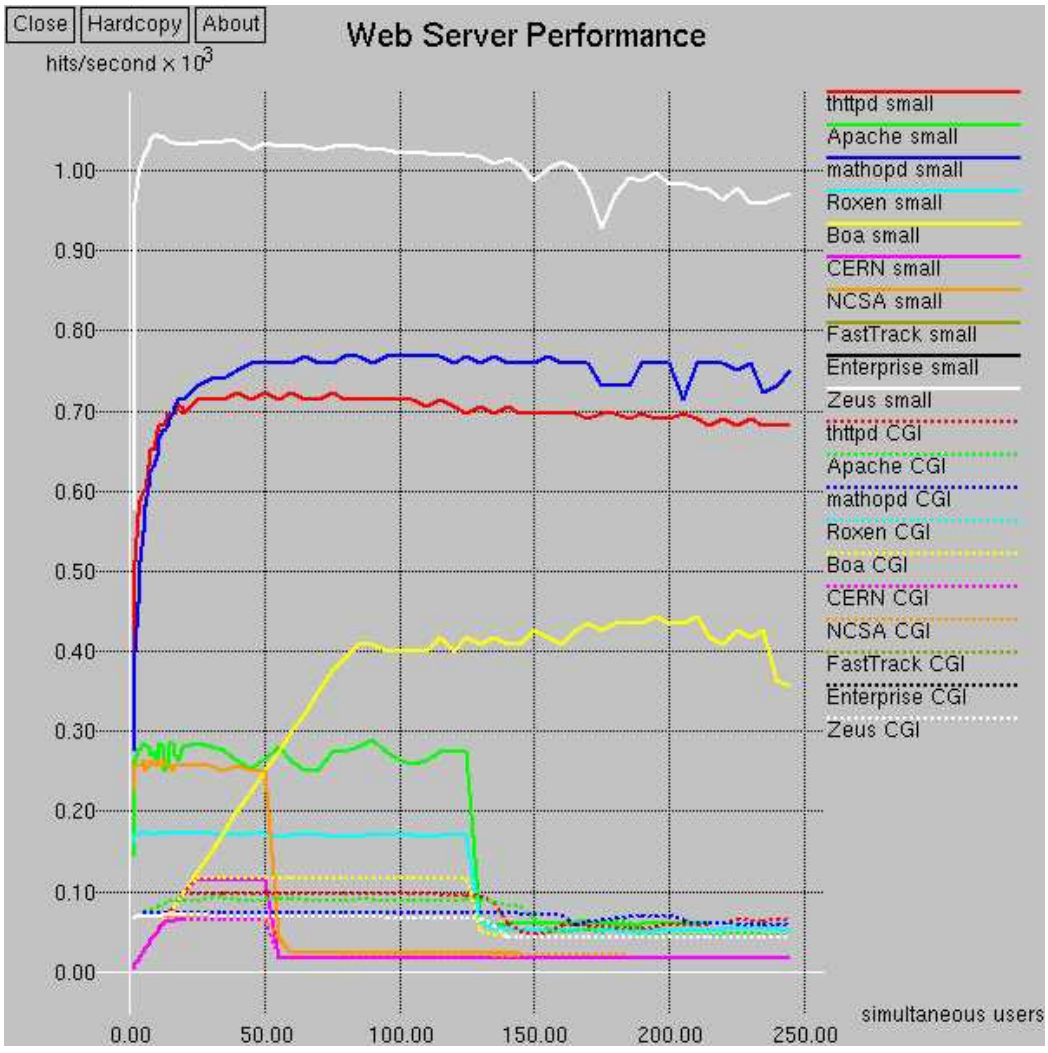
- For multi-threaded server, please try two thread pool sizes: one small and one large.

Part 2f: Report

- You should submit a report on your server design.
  - Please answer all questions we specified above.

  - Please report the measured performance of both Apache and your best server for these performance metrics: throughput and (mean) delay. You can use open office or gnuplot to generate figures. Below is an example figure showing the performance of multiple servers.

  - The TA will benchmark all servers and pick the one with the highest throughput. This server will receive a bonus of 25%.



Submission

- **Please submit using canvas. Please include README to tell the TA the directory structure, e.g., which file is the report. Please generate a single jar file containing all of your files.**

Suggestions

During your async i/o design based on select, think how you implement a finite state machine to handle each request (e.g., initial state after accepting a connection, what other states). Java async i/o does not allow you to select events on a file channel. There are can be multiple design options to handle file i/o:

- Use standard file i/o by assuming that file system is fast and will not become bottleneck;
- Try out mapped file i/o:

```
FileInputStream fin = new FileInputStream(args[0]);
FileChannel in = fin.getChannel();
ByteBuffer input = in.map(FileChannel.MapMode.READ_ONLY, 0, in.size());
```

- Try out direct transfer: See FileChannel.transferTo;
- Use standard file i/o and use a thread pool to help with reading files.

References

- Book
  - Java Network Programming (4th ed.) by Elliotte Harold is a good reference book on Java network programming. Yale library has electronic version of this book (you need Yale IP address to gain access). Please try to search Orbis and follow the link. The examples codes can be found here.
- General java information:
  - Development environments:
    - zoo has Java installed:
    - eclipse provides a nice java IDE.
  - Java book
    - An overall very good book on Java is **Thinking in Java (4th Edition)** by Bruce Eckel. The book web site is here.