Computer Science 3003: Distributed Systems
University of Arkansas – Fort Smith
Spring 2025

Final Project – Distributed Sort and Aggregation System

200 points

Version 1.0

due Monday, 5 May 2025, by 11:00 PM CT

Review the following assignment in its entirety prior to beginning. All submissions will be managed from within the course website.

Application Development

Your task is to develop a distributed system that combines the various topics presented in this course. You will need to develop a containerized master server along with a containerized utility server in a team environment that demonstrates solid object-oriented design principles.
The following tasks are required for implementation in this project:

1. Implementation of a client to connect to the master server
2. Implementation of a master server that can accept multiple concurrent connections
3. Implementation of a utility server that can accept multiple concurrent connections
4. Deployment of master and utility servers in separate containerized environments

The following algorithms need to be designed and implemented by your team:

1. Design of the broadcast protocol as indicated in the illustrations that follow in a multi-master socket environment using threads.

2. Implementation of a master server that can receive request from a client application and a utility server that can process the task. For this project, the client application needs to be able to connect to the master server and transmit a file of line-delimited numbers to be sorted. The master server should be be able to divide this into chunks of size 10,000,000 (10 million numbers) and send a single chunk to a utility server. For example, if you had 3 utility servers and 40,000,000 numbers, then three utility servers would receive 10,000,000 numbers and the first server to finish would receive the remaining 10,000,000 number chunk. Each utility server must be assigned a command-line argument that tells it which sorting algorithm to run out of the following options: { Bubble Sort, Insertion Sort, Merge Sort }. When the utility server runs, it should sort the file chunk and return the total sum of the first and last 5 numbers in sorted ascending order (return: first 5 nums + last 5 nums). The master server should aggregate the results from each of the utility servers and produce a single summation for all chunks, and every connected utility server should append the current timestamp and the result to a text file named `log.txt` to verify the results. You may not assume that the file will be evenly divisible by 10,000,000, so one system may receive a smaller chunk, but it will contain at least 10 numbers in it.

3. Implementation of the heartbeat protocol as indicated in the illustrations that follow in a multi-master socket environment using threads. The heartbeat protocol should be invoked each time the master server receives a request from the client.

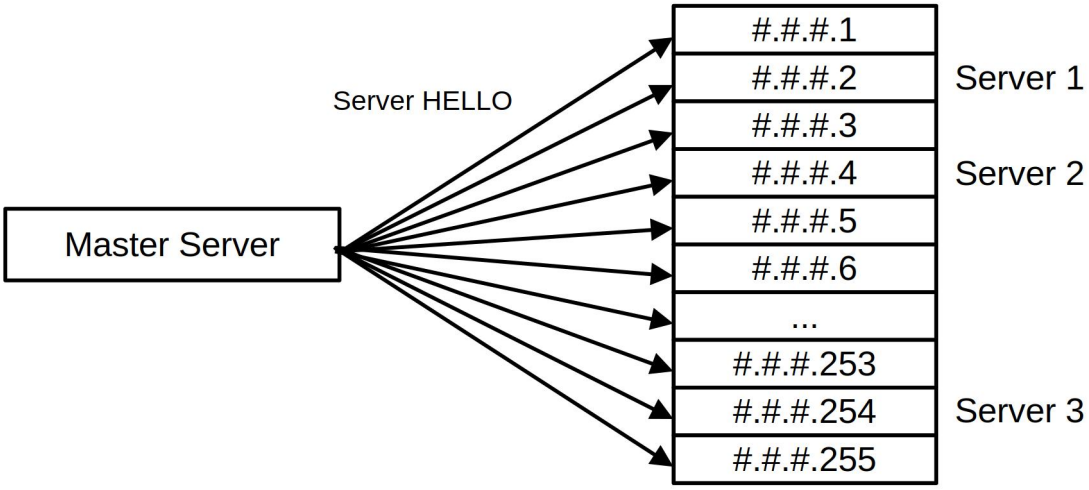

Figure 0.1: The master server initiates the protocol by broadcasting a request for all IPs on the same host network.

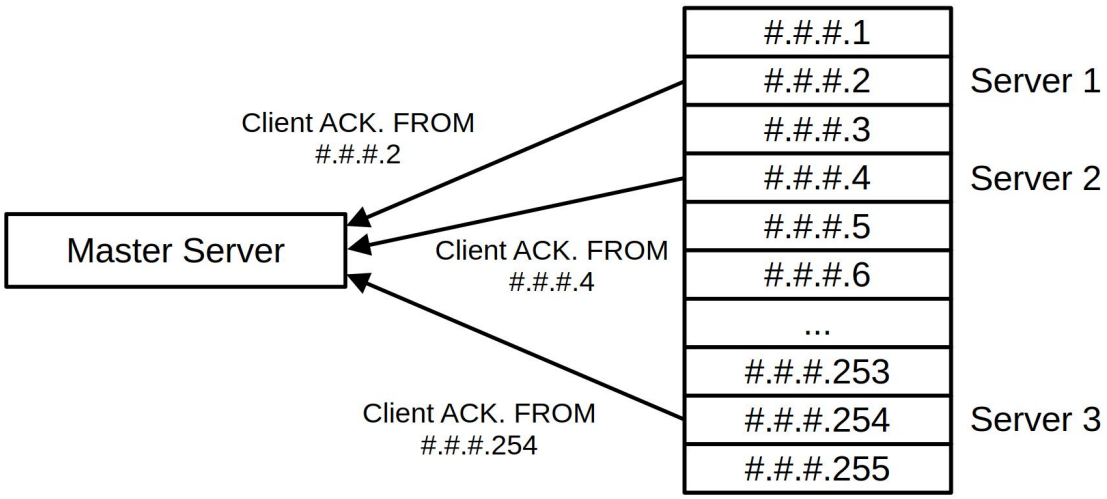

Figure 0.2: Each utility server on the same host network receives the broadcast message from the master server and responds with an acknowledgment message that contains relevant information about the utility server (i.e. IP address, etc.).

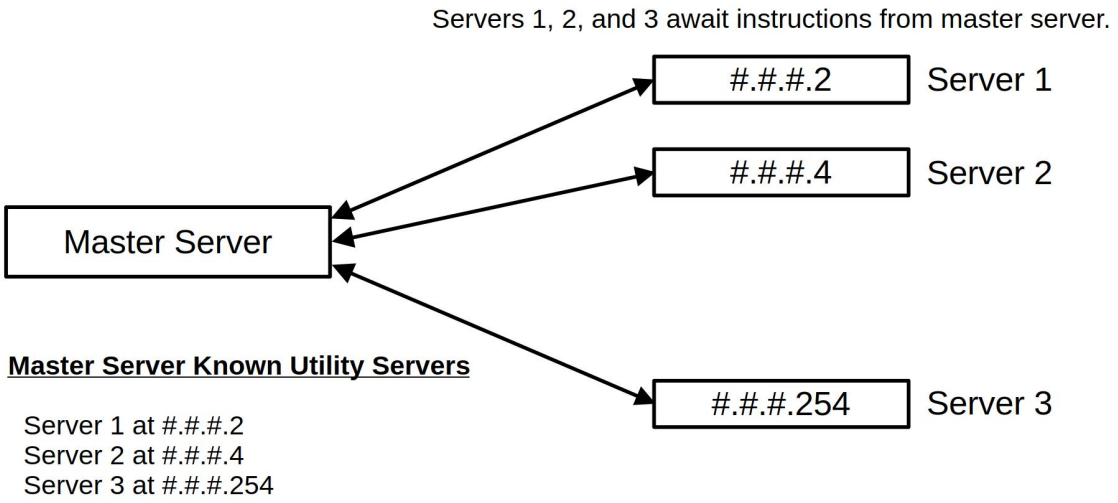

Figure 0.3: After the master server receives the acknowledgment from each client, the master server retains a collection of valid servers along with their current locked state (`LOCKED` or `AVAILABLE`), IP address, and other relevant information. Each utility server remains in an `AVAILABLE` state until it receives a task from the master server.

Utility Server Locking

Each utility server is required to maintain two possible states:

1. `AVAILABLE` – the server is available to accept a task from the master server.
2. `LOCKED` – the server is currently processing a task and the master server should not initiate any further requests until the utility server completes the task and becomes available.

When a master server sends a task, the utility server should go into a locked state until the task is complete. The master server should not send any further requests to the utility server until it is complete. If the server fails to complete its task, then the master server should send the request to a separate server and deactivate the utility server that failed.

When the utility server receives a request, the thread of computation on the utility server should sleep for 15 seconds before it completes the task to add an artificial delay for testing and evaluation purposes (and log the activity).

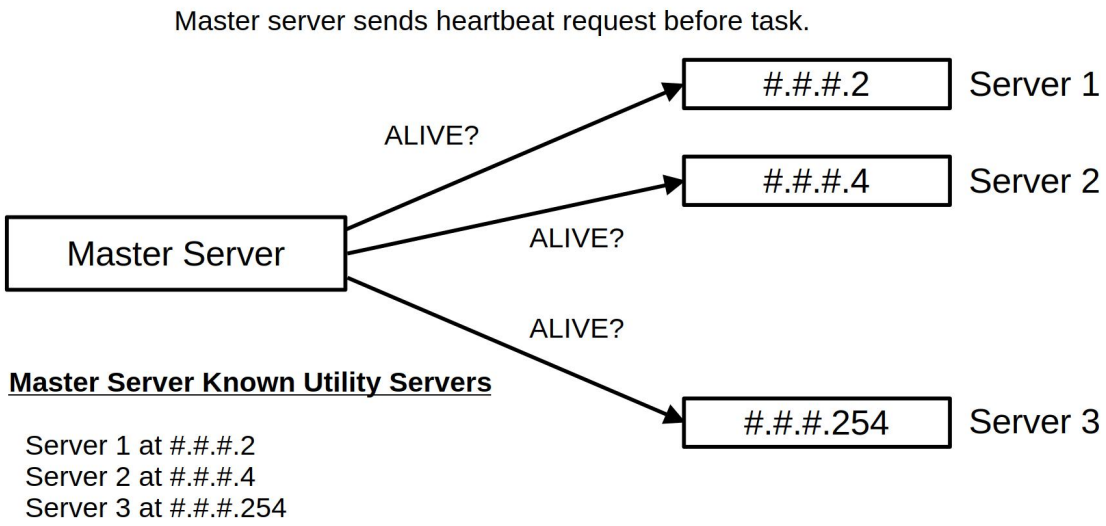

Figure 0.4: Once the master server receives a task that it is required to complete, it will first issue a *heartbeat request*" to verify the availability of all utility servers. This is to confirm whether or not a server is still available. For simplicity, this will only take place when the master server receives a task rather than on fixed time intervals (i.e. every 10 minutes).

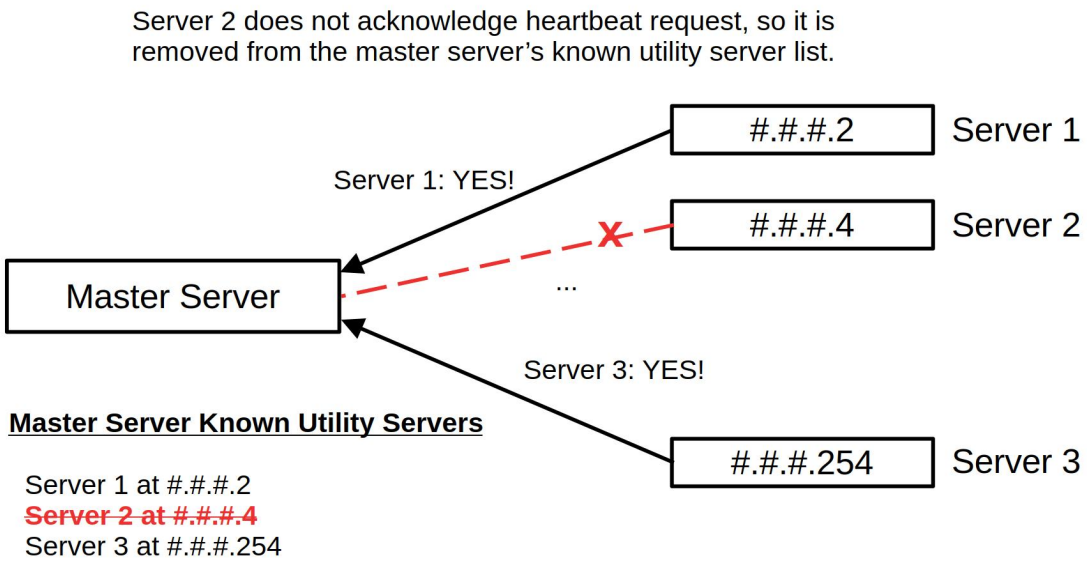

Figure 0.5: The master server will wait for a specified time, which should be a command line argument which we will refer to as the `heartbeatTimeoutPeriod` and is measured in milliseconds. If the master server does not here from a client within the heartbeat timeout period, then the master server will drop the server from its collection of available servers. The server will not be readded until the master server is restarted or it invokes the broadcast message once again.

The following information should facilitate the development of this project:

1. A log message should be output to `log.txt` for every operation that takes place for verification purposes. This should be verbose for debugging purposes. For example, the master and utility servers should log every action with a timestamp. Failure to sufficiently demonstrate the use of logging in a way that is useful for evaluation purposes will result in a point penalty. The log output should also be well-formatted (hint: use `System.out.printf()`).

2. Your code should be self-documenting. Break-up complex logic using methods.

3. Each team will need to draft a formal report the details each algorithm and provides the pseudocode (not actual code) of your implementation for the three algorithms named above. This should also include information about the use of threads, synchronization, and other relevant information about your implementation.

Evaluation

Each team will be required to demonstrate and present a successful implementation to receive credit. Partial credit will only be awarded at the instructor's discretion. Each team will be evaluated on the following criteria:

Evaluation Criteria	**Points**	**Score**
Development of master server with concurrency	20	
Development of utility server with concurrency	20	
Locking algorithm and implementation	10	
Discovery protocol implementation	20	
Logging and object-oriented design	5	
Containerized application deployment	15	
Successful end-to-end demonstration of project	10	
Total	100	

Deliverables

You will be responsible for delivering the following items:

1. Latex Documents – your latex code should be used to generate a PDF which will then be submitted. Be sure that all documents submitted list your name, problem set information, date and class.

2. Application Code – be sure to submit your source code as indicated above to the course website and through the `code` server. All submitted code must have your 1) name 2) username (code server) 3) problem set number and 4) due date as a comment at the top of each class. Create a folder on the code server in the `problemsets` directory named `ps#` (e.g. `ps2`).

```
/*********************************
Name:           Israel Cuevas
Username:       ua12345
Problem Set:    PS#
Due Date:       Month day, YEAR
*********************************/
```