ECON381 Fall 2024
Homework Assignment 1

1-Try to model the load balancer's server selection process. Sketch the data structure for the problem.
2. Is there a sorting-related problem here?
3. Which type of algorithm would be better suited for this problem? Why?
4.Suppose ABC Bank decided to implement a "high priority customer" program. In this program these high priority customers' transactions are processed earlier than those of the ordinary customers. How would you handle this change in your data structure and your algorithm?
5. Suppose banking regulations now enforce all banks to prioritize transactions that have been waiting the longest duration. How would you handle this change in your data structure and your algorithm?
6. How would you classify the current design and your proposed designs in terms of sorting stability?

1. Modeling the Load Balancer's Server Selection Process
The server selection process works as follows:
Servers are stored in a `PriorityQueue` data structure, which sorts servers based on the number of active tasks.
Whenever a new task request arrives, the least busy server is selected from the queue (using the `poll()` method). After the task is assigned, the server's task count is incremented and it is re-added to the queue.

Data Structure Design:
```
PriorityQueue<Server> servers
```

Server:
Each server contains an ID and the number of active tasks. Servers are sorted according to the task count using the `Comparable` interface.

2. Is There a Problem with Sorting?

Yes, servers need to be kept in a sorted manner. Therefore, using a `PriorityQueue` is the correct choice.
When a new task is assigned, the queue is re-sorted based on the active task count.
The sorting issue becomes more prominent as the number of servers and task load increases.

3. Suitable Algorithm Type for This Problem

For this type of problem, the most suitable algorithm type is one based on data structures that enable selection based on priority:

- **Min-Heap** or **PriorityQueue**: These have O(log N) time complexities for insertion and removal, making them suitable since the state of the servers changes dynamically.
- Alternatively, in large-scale systems, a distributed system model could be considered, where server states are updated in parallel.

4. Priority Customer Program

Priority customer tasks should be processed faster than regular customer tasks. For this change:

- **Data Structure Change:**
  Instead of using a `highPriorityQueue` in the `PriorityLoadBalancer` class, a `PriorityQueue` can be used.
  Priority customers are added to the queue with a high priority value.
- **Algorithm Change:**
  If the `highPriorityQueue` is not empty, tasks are taken from this queue.
  If it is empty, the regular task queue (`servers`) is used.

5. Prioritization Based on Wait Time

If tasks are prioritized based on wait time:

- **Data Structure Change:**
  Each task is stored with a timestamp to track wait time. This can be tracked inside a `PriorityQueue<Transaction>` or a list within the servers.
- **Algorithm Change:**
  Tasks are ordered in servers based on wait time. Each new task is assigned starting from the task with the longest wait time.

6. Classification of the Design in Terms of Sorting

- **Current Design:**
  It sorts based on the active task count. This can be classified as a load balancing sorting algorithm.
- **Priority Changes:**
  Prioritization and wait time-based sorting fall under dynamic sorting types.
- **Performance:**
  Sorting algorithms like `PriorityQueue` are suitable for scalability, but optimization is needed as the task load increases.

Barış PALA
20232810024