Uroš Stojković, Lazar Minić

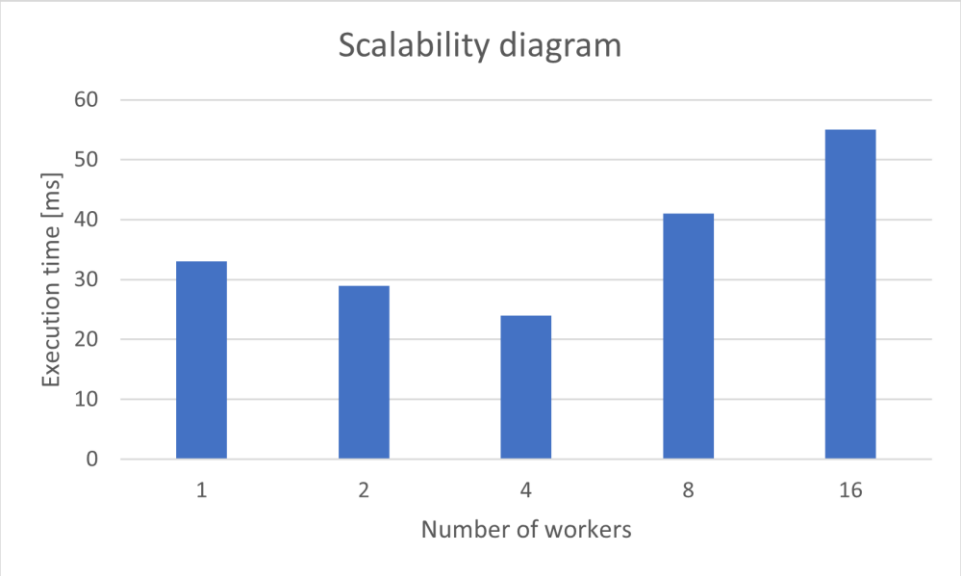
Assignment 3

Design questions

1. **How does the leader process know the number of alive workers?**
   Leader process maintains a hash map where keys are socket descriptors associated with TCP connection with each worker. When a new connection with a worker process is established, leader adds the associated socket descriptor to the hash map. Value of the hash entry is a structure which includes name of the file commissioned to the worker as well as a timestamp indicating the time of the last communication.
2. **How can we distribute work "fairly" among workers?**
   It is somewhat vague what "fairly" means. In the proposed solution, tasks (files) are given to whichever worker requests them. Worker program consists of a loop where, after performing the given task and sending the result back to leader, it waits for the next task by blocking on the `recv()` call. On the other hand, leader polls its connections and once it notices that some worker is ready to receive another task, it sends one of the still unassigned files to it. Thus, the system distributes the work fairly in the sense that it doesn't discriminate against any worker as long this worker is ready to contribute.
3. **With what messages do leader and workers communicate?**
   Their communication is very simple. Leader sends a worker name of the file to be processed, while the worker responds with an integer. By using sockets and C standard libraries, processes involved in the connection can also detect when the other side is ready to receive data or when it hung up (closed its end of the connection).
4. **How can we detect failed / crashed workers?**
   Leader process can detect the crash of some worker by polling the connection and detecting that connection has been terminated. In addition, if the leader attempts to send a task to the worker and gets a response from the OS indicating some error occurred during communication, it assumes there's something wrong with the worker and closes the corresponding socket. Furthermore, for each worker, leader periodically checks how much time has passed since their last "contact". This is achieved by storing the timestamp which is updated whenever it successfully sends or receives some data from the worker.
5. **How do we recover when a worker fails?**
   When a worker fails, the file that has been assigned to him is returned to the list of unassigned files so that it can be assigned again to some other worker. Furthermore, an entry in the hash map indexed by a socket descriptor associated with that worker is removed. Also, the same socket descriptor is cleared from the list of sockets that are being polled.

Scalability questions

1. **Configure the test scripts to spawn 1,2,4,8,16 workers - draw a diagram visualizing how execution time is affected by adding workers.**



2. **What is the limit in scaling? (Network, CPU-bound)**
   If the files to be processed have a small number of URLs (as in the test case where there are only 2 URLs per file), the communication over the network eats up most of the time. Hence, to make the most out of distributed processing, URLs should be divided in larger chunks.
3. **Measure each worker's load - how is load balancing affected by scale?**
   On a single machine, it looks like it becomes more probable that distribution is skewed.
4. **Could you think of a case when the coordinator would become a bottleneck in such a system?**
   The coordinator could become a bottleneck if there are a lot of workers who are executing their tasks very fast. In that case, workers would spend most of the time waiting for the overwhelmed coordinator to give them new files to process.