Group 43 — Report for Assignment 3

Rodrigues, Diogo	Silva, Tiago	Martins, João Lucas
MTK 03770446	MTK 03770544	MTK 03770536
diogo.rodrigues@tum.de	tiago.silva@tum.de	joao.martins@tum.de

TUM – Cloud-Based Data Processing 2022/23
30th November 2022

Description of the implementation

Our implementation is simple:

- The coordinator opens a socket to which workers can connect to, and waits until it can accept a connection.
- Once the coordinator accepts a connection, it adds the new connection to a data structure to store the list of available workers. This connection is persistent and kept alive during the whole lifetime of the worker.
- When a worker starts, it communicates with the coordinator endpoint to report it is alive.
- When the coordinator has work to do, it chooses one of the available workers that is not processing anything, and sends it the chunk URL that the worker must download and process.
- When the worker finishes processing a chunk, it replies to the coordinator with the result, as such reporting as well that it is now free and can accept more work.

Design questions

- How does the leader process know the number of alive workers?
  A: Workers notify the leader when they are ready to work, by establishing an individual TCP connection for each worker.
- How can we distribute work "fairly" among workers?
  A: Each worker asks for more work when they are finished. After a worker finishes processing a chunk, it yields the CPU so that other workers on the same machine also have a fair change of running.
- With what messages do leader - worker communicate?
  A: Worker establishes a connection to the coordinator. Then the coordinator can send a `MessageWork` message to the specified worker that contains the URL of the chunk to be retrieved and processed. After this, when the worker finishes processing the file, it sends back the result to the coordinator in a `MessageWork` message with type `RESPONSE`, thereby also informing the coordinator that it is ready to receive more work.
- How can we detect failed / crashed workers?
  A: The coordinator assumes that a worker has crashed when its TCP connection to the coordinator has closed unexpectedly.
- How do we recover when a worker fails?
  A: If the worker was working on a chunk, the chunk URL is given to another, functioning worker that will process the chunk that the failed worker was processing at the time it failed.

Scalability questions

What is the limit in scaling? (network, CPU-bound)

Number of workers	Attempt 1	Attempt 2	Attempt 3	Avg.
1	42.069	44.476	43.869	43.471
2	34.910	28.980	27.052	30.314
4	20.987	20.291	21.242	20.840
8	20.106	19.350	23.384	20.947
16	23.247	16.992	21.578	20.606

Figure 1: Execution time (in seconds) of three attempts for 1, 2, 4, 8 and 16 workers.

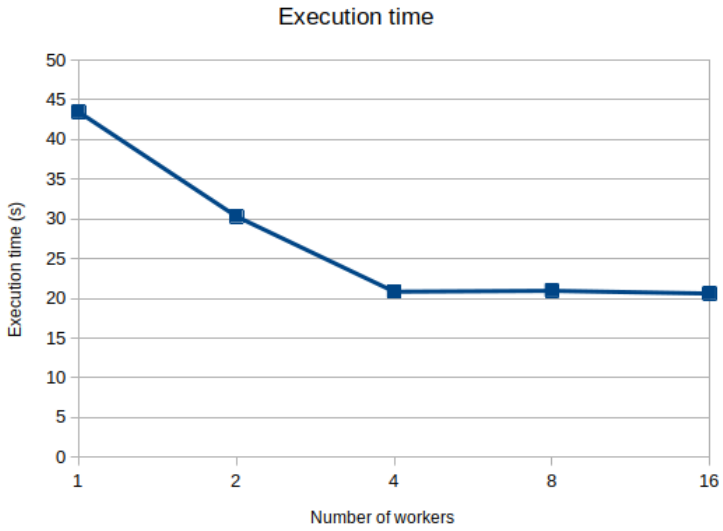

Figure 2: Execution time versus number of workers.

The computer where we ran these tests has 4 cores, 8 threads, so the fact the execution time flat-lined at 4 workers means that this system is not CPU-bound. We would therefore assume that the system is network-bound.

Measure each worker's load – how is load balancing affected by scale?

Number of workers	1	2	4	8	16
W1	100	50	26	12	6
W2		50	21	13	6
W3			24	14	7
W4			29	13	5
W5				12	5
W6				11	8
W7				12	5
W8				13	7
W9					5
W10					7
W11					7
W12					7
W13					5
W14					5
W15					7
W16					8
Avg.	100	50	25	12.5	6.25
StdDev.	±0.000	±0.000	±2.915	±0.866	±1.090

Figure 3: Execution time (in seconds) of three attempts for 1, 2, 4, 8 and 16 workers.

Load balancing is relatively unaffected by scale, since the standard deviation always stays below 3, and usually around 1.

Could you think of a case when the coordinator would become a bottleneck in such a system?

Such a scenario would occur if the system had a very large number of workers, each with a high bandwidth, but the coordinator is running in a relatively slow machine, in which case the coordinator cannot dispatch work messages fast enough to keep all workers busy. Thus, the coordinator becomes a bottleneck for the system.