Project presentation

Team orange : Bastian Preisel, Edwige Weber

Final plan schedule

Mid-way vs final implementation

Results

Difficulties

Time management

Final plan

	Date	Bastian	Edwige
	Week 45	• Learn about *Protobuf* • Design for connecting and messaging	• Learn about *Protobuf* • Design merging and partitioning
	Week 46	• Design connection and messaging	• Design merging and partitioning
Milestone #1	Week 47	• Implement slaves and master	• Generate data • Implement local sorting
	Week 48	• Implement sampling • Perform progress report	• Implement partitioning • Perform progress report
Milestone #2	Week 49	• Implement sampling • Shuffle	• Implement partitioning • Merge
Milestone #3	Week 50	• Test	• Test

Mid-way implementation plan

1. Server starts up with known amount of clients.
2. Initiate contact with greet stub and grant ID to workers (unary)
3. Clients sends the server its key ranges to master, and gets a dummy response. (unary)
4. Clients ask server every one second if it's done partitioning. (unary)
5. Server partitions, and tells clients it's ready (unary)
6. Clients requests their ranges, and the server provides it. (server-side streaming)
7. Clients splits up their data according to the partitioning done by the server
8. Clients sends their unwanted data to the server, and gets a dummy response (client-side streaming)
9. Server stores the data according to the partitioning.
10. Clients ask server every one second if it's done receiving data. (unary)
11. Server tells clients it's done shuffling. (unary)
12. Clients requests their data, and the server provides it. (server-side streaming)
13. Clients sort their data locally, and the server knows what order the clients are sorted in.

Final implementation plan

1. Server starts up with known amount of clients.
2. Initiate contact with greet stub and grant ID to workers (unary)
3. Clients sends the server its key ranges to master, and gets a dummy response. (unary)
4. Clients ask server every one second if it's done partitioning. (unary)
5. Server partitions, and tells clients it's ready (unary)
6. Clients requests their ranges, and the server provides it. (server-side streaming)
7. Clients splits up their data according to the partitioning done by the server
8. Clients sends their unwanted data to the appropriate client, and gets a dummy response
9. ~~Server stores the data according to the partitioning.~~
10. ~~Clients ask server every one second if it's done receiving data. (unary)~~
11. ~~Server tells clients it's done shuffling. (unary)~~
12. Clients receives their data
13. Clients sort their data locally, and the server knows what order the clients are sorted in.

Results

/input10/

Success

All files somewhat equally distributed

(3.16 mill, 3.16 mill, 3.26 mill)

/input100/

Failure

Initial sorting took too much memory in the heap.

java.lang.OutOfMemoryError: GC overhead limit exceeded

Memory leak

/input300/

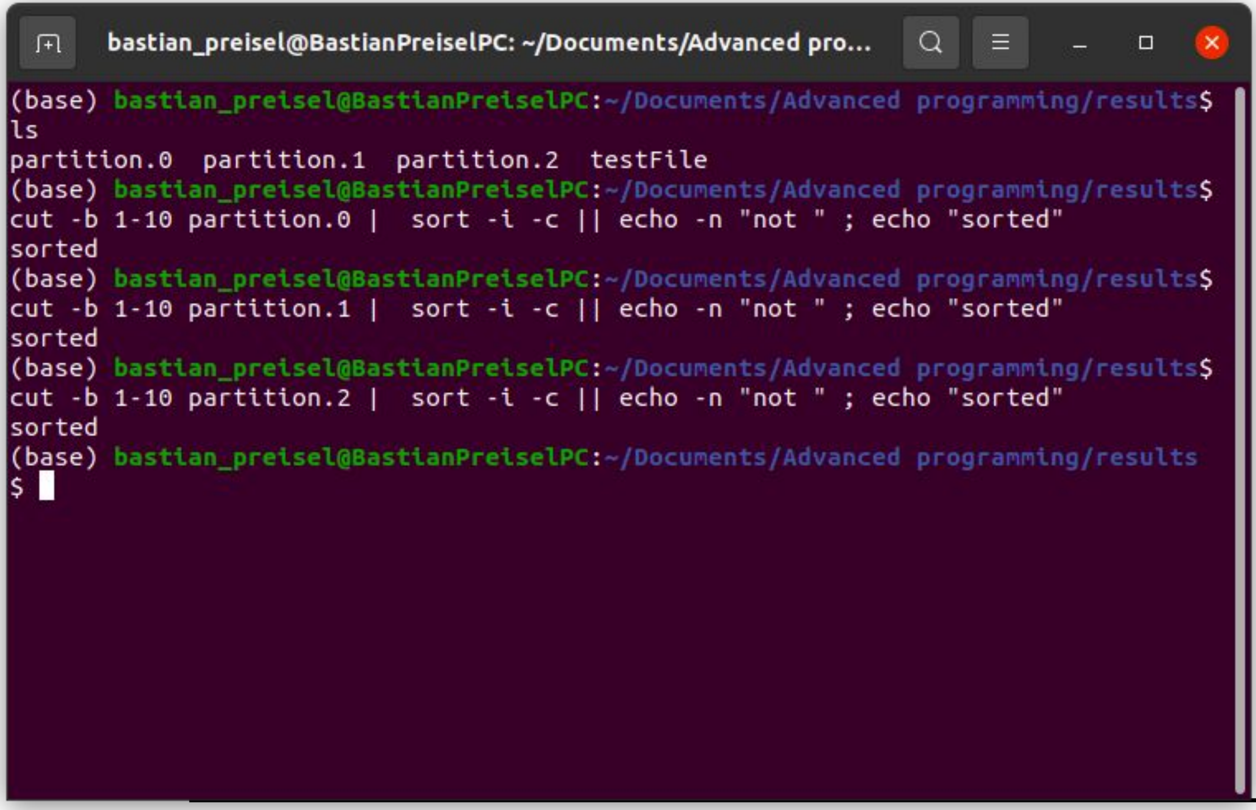
Failure

Initial sorting took too much memory in the heap.

java.lang.OutOfMemoryError: GC overhead limit exceeded

Memory leak

Validating result



bastian_preisel@BastianPreiselPC: ~/Documents/Advanced pro...

```
(base) bastian_preisel@BastianPreiselPC:~/Documents/Advanced programming/results$
ls
partition.0  partition.1  partition.2  testFile
(base) bastian_preisel@BastianPreiselPC:~/Documents/Advanced programming/results$
cut -b 1-10 partition.0 |  sort -i -c || echo -n "not " ; echo "sorted"
sorted
(base) bastian_preisel@BastianPreiselPC:~/Documents/Advanced programming/results$
cut -b 1-10 partition.1 |  sort -i -c || echo -n "not " ; echo "sorted"
sorted
(base) bastian_preisel@BastianPreiselPC:~/Documents/Advanced programming/results$
cut -b 1-10 partition.2 |  sort -i -c || echo -n "not " ; echo "sorted"
sorted
(base) bastian_preisel@BastianPreiselPC:~/Documents/Advanced programming/results
$ 
```

Difficulties

- Linux/windows environment
- Communication
- Testing

Linux/Windows environment

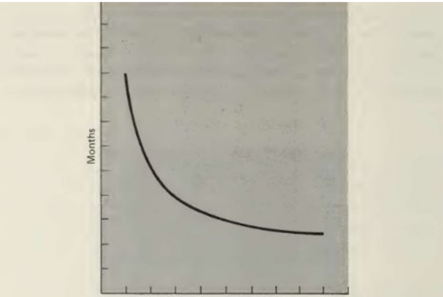


Fig. 2.3 Time versus number of workers—partitionable task requiring communication

- Windows could not run the full code
  - Had to work partially with the code
  - SBT was inconsistent
- Had to ‘guess’ if the code would work with the full code
  - In-person meetings to check if the code worked on the full code
- Not a big problem, since our work was partitionable
  - We had a deal on the save locations and file naming conventions

Communication

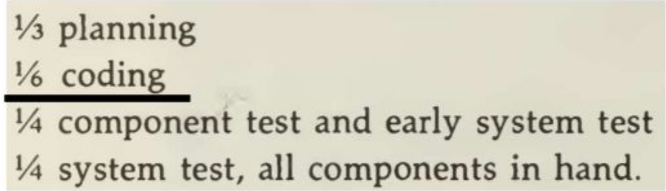
- Quarantine due to close-contact
  - Zoom meetings not as good quality as in-person
- Didn’t explain much how our code worked, because of our agreed upon plan
- Relatively small number of people
- Communication overhead was minimum

Testing

- Not a lot of time for testing (test-case)
  - Bad idea to put it in the end
  - Could have saved time
- Instead, intimate knowledge of the code
- Not tested for large amount of data
  - Memory leakage

Time management

- ⅓ planning
- ⅙ component testing
- ⅓ coding
- ⅙ system test all components



⅓ planning

⅙ coding

¼ component test and early system test

¼ system test, all components in hand.

What went wrong?

- Plan for implementation had to change the more we learned
- Choosing correct library (ZIO vs scalaPB)
- Didn’t prioritize testing

Thank you for listening

Do you have any questions ?