Project presentation

Team RED

김대희
박정훈
홍재영

Design Summary

gRPC communication(M: master / W: worker)

- Connection (M ⇔ W)
- Sampling (M ⇔ W)
- Range (M ⇔ W)
- SortPartition (M ⇔ W)
- Shuffle
  - Shuffle ready (M ⇔ W)
  - Shuffle (W⇔W)
  - Shuffle complete (M ⇔ W)
- Merge (M ⇔ W)

Design Details: Worker, Master, gRPC

Worker design

- Instantiate client for communicating with master server
- Execute each phase with data according to replies from Master
- Instantiate server & client for shuffle phase
- Manage input/output data files

Master design

- Instantiate server for communicating with client workers
- Shutdown after all phases are done

gRPC

- Usually use unary communication between Master and Worker
- Client Stream from Worker at sampling, shuffle phases.

Design Details: Phases

Connection

- Master instantiate server object and wait for connection request
- Workers send connection request and get reply back from Master

Sampling

- Workers get Samples from input blocks with 1% ratio using client stream
- Master save samples to make key ranges for each workers

Range

- Workers send request to get keyrange after every worker receive reply
- Master send reply with arranged key range lists

Design Details: Phases

Sort/Partition

- Workers send request once sort & partition done and get reply back.

Shuffle

- Every workers instantiate client & server for shuffling
- Once shuffle server run, send shuffle ready to Master waiting for other workers
- Shuffle partitions in order and send shuffle complete to Master

Merge

- Each workers merge their partition files with sorting
- Set max output file capacity as about 20MB

Experiment (4 workers * 32MB *2 blocks)

1. Does the master start? & 2. Does each worker connect to the master?

```
red@vm03:~/test/daehuikim/332project$ sbt "run master 4"
[info] welcome to sbt 1.8.0 (Private Build Java 1.8.0_352)
[info] loading settings for project root-332project-build from plugins.sbt ...
[info] loading project definition from /home/red/test/daehuikim/332project/project
[info] loading settings for project root from build.sbt ...
[info] loading settings for project master from build.sbt ...
[info] loading settings for project worker from build.sbt ...
[info] loading settings for project common from build.sbt ...
[info] set current project to gensort (in build file:/home/red/test/daehuikim/332project/)
[info] running gensort.Main master 4
2.2.2.103:50051
Dec 12, 2022 5:54:48 PM network.NetworkServer start
INFO: Server started, listening on 50051
Dec 12, 2022 5:55:00 PM network.NetworkServer$NetworkImpl connection
INFO: [Connection] Request from 2.2.2.104:9000 arrived
Dec 12, 2022 5:55:00 PM network.NetworkServer$NetworkImpl connection
INFO: [Connection] Request from 2.2.2.105:9000 arrived
Dec 12, 2022 5:55:01 PM network.NetworkServer$NetworkImpl connection
INFO: [Connection] Request from 2.2.2.107:9000 arrived
Dec 12, 2022 5:55:01 PM network.NetworkServer$NetworkImpl connection
INFO: [Connection] Request from 2.2.2.106:9000 arrived
Dec 12, 2022 5:55:01 PM network.NetworkServer$NetworkImpl connection
INFO: [Connection] Input file reply to 2.2.2.106:9000 completed
```

Experiment (4 workers * 32MB *2 blocks)

3. Does the master collect sample data? &
4. Does the master return distribution keys back to workers?

```
Dec 12, 2022 5:55:02 PM network.NetworkServer$NetworkImpl sampling
INFO: [sample]: Worker tries to send sample
Dec 12, 2022 5:55:02 PM network.NetworkServer$NetworkImpl sampling
INFO: [sample]: Worker tries to send sample
Dec 12, 2022 5:55:02 PM network.NetworkServer$NetworkImpl sampling
INFO: [sample]: Worker tries to send sample
Dec 12, 2022 5:55:02 PM network.NetworkServer$NetworkImpl sampling
INFO: [sample]: Worker tries to send sample
Dec 12, 2022 5:55:06 PM network.NetworkServer$NetworkImpl$$anon$1 onCompleted
INFO: [sample]: Worker done sending sample
Dec 12, 2022 5:55:06 PM network.NetworkServer$NetworkImpl$$anon$1 onCompleted
INFO: [sample]: Worker done sending sample
Dec 12, 2022 5:55:07 PM network.NetworkServer$NetworkImpl$$anon$1 onCompleted
INFO: [sample]: Worker done sending sample
Dec 12, 2022 5:55:07 PM network.NetworkServer$NetworkImpl$$anon$1 onCompleted
INFO: [sample]: Worker done sending sample
Dec 12, 2022 5:55:07 PM network.NetworkServer$NetworkImpl range
INFO: [Range] Try to broadcast range
Dec 12, 2022 5:55:07 PM network.NetworkServer$NetworkImpl range
INFO: [Range] Try to broadcast range
Dec 12, 2022 5:55:07 PM network.NetworkServer$NetworkImpl range
INFO: [Range] Try to broadcast range
Dec 12, 2022 5:55:07 PM network.NetworkServer$NetworkImpl range
INFO: [Range] Try to broadcast range
```

Experiment (4 workers * 32MB *2 blocks)

5. Do workers pass intermediate data between each other (during shuffling)?

1. Send request to other worker

```
INFO: [Shuffle] Try to send partition from2.2.2.105to4
Dec 12, 2022 7:43:37 PM shufflenetwork.FileClient sendPartition
```

2. Reply messages per each partition file

```
INFO: [ShuffleServer]: Worker done sending partition
Dec 12, 2022 7:43:37 PM shufflenetwork.FileClient sendPartition
```

3. After shuffling complete, send request to Master

```
INFO: [Shuffle] Done sending partition
Dec 12, 2022 7:43:40 PM network.NetworkClient checkShuffleComplete
INFO: [Shuffle] Try to send Master shuffle complete
Dec 12, 2022 7:43:41 PM network.NetworkClient checkShuffleComplete
INFO: [Shuffle] complete arrange every partitions at2.2.2.105
```

Experiment (4 workers * 32MB *2 blocks)

6. Does the master print a sequence of workers?

```
2.2.2.105:9000, 2.2.2.106:9000, 2.2.2.104:9000, 2.2.2.107:9000,
```

7. Is the output sorted in each worker?

```
'InK<4:97+  00000000000000000000000000039766  EEEE4444AAAA7777CCCCFFFFDDDD444488883333DDDDAAAABBBB
'InN=^kI9>  0000000000000000000000000B4BAE  4444BBBB33339999DDDDDDDDCCCC0000DDDDDDDD555588883333
'InN=^kI9>  0000000000000000000000000B4BAE  4444BBBB33339999DDDDDDDDCCCC0000DDDDDDDD555588883333
'Ip6LG&zFo  0000000000000000000000000C7813  00009999666677775555333322224444888866665555000CCCC
'Ip6LG&zFo  0000000000000000000000000C7813  00009999666677775555333322224444888866665555000CCCC
'Ipjz&E.Y4  0000000000000000000000000E73A4  99998888CCCCAAAA3333CCCCDDDD4444000011111111666DDDD
'Ipjz&E.Y4  0000000000000000000000000E73A4  99998888CCCCAAAA3333CCCCDDDD4444000011111111666DDDD
'Iq#\+gdF!  0000000000000000000000000A8EB0  DDDDAAAAEEEEAAAA2222EEEEEEEECCCCCCCC0000CCCC99991111
'Iq#\+gdF!  0000000000000000000000000A8EB0  DDDDAAAAEEEEAAAA2222EEEEEEEECCCCCCCC0000CCCC99991111
'IqL,cwoYr  00000000000000000000000004CC33  FFFF5555CCCCFFFF99993333111188888888AAAA2222EEEECCCC
"partition_1" 200001L, 19800099C
```

Experiment (4 workers * 32MB *2 blocks)

8. # of records in the input == # of records in the output?

```
red@vm04:~/test/daehuikim/332project/data/output$ ls
partition_1  partition_2  partition_3  partition_4
"partition_1" 200001L, 19800099C
"partition_2" 200001L, 19800099C
"partition_3" 200001L, 19800099C
"partition_4" 35495L, 3514005C
```

Line : 635,498

```
red@vm06:~/test/daehuikim/332project/data/output$ ls
partition_1  partition_2  partition_3  partition_4
"partition_1" [noeol] 200001L, 19800098C
"partition_2" [noeol] 200001L, 19800098C
"partition_3" [noeol] 200001L, 19800098C
"partition_4" 54113L, 5357187C
```

Line : 654,116

```
red@vm05:~/test/daehuikim/332project/data/output$ ls
partition_1  partition_2  partition_3  partition_4
"partition_1" 200001L, 19800099C
"partition_2" [noeol] 200001L, 19800098C
"partition_3" [noeol] 200001L, 19800098C
"partition_4" 34717L, 3436983C
```

Line : 634,720

```
red@vm07:~/test/daehuikim/332project/data/output$ ls
partition_1  partition_2  partition_3  partition_4
"partition_1" [noeol] 200001L, 19800098C
"partition_2" [noeol] 200001L, 19800098C
"partition_3" [noeol] 200001L, 19800098C
"partition_4" 35663L, 3530637C
```

Line : 635,666

Total Lines : 2,560,000 => Same as the input !!!

Experiment Report

Environment specification

- Tested on VM cluster.
  - Master: red@vm03
  - Worker: red@vm04, red@vm05, red@vm06, red@vm07

Input specification

- **32MB block * 2 per each worker (total 2,560,000 lines)**

Output specification

- **total 2,560,000 lines sorted from vm04 to vm07**

Executing time

- **6541 seconds(1h 49m 01s)**

```
[success] Total time: 6541 s (01:49:01), completed Dec 12, 2022 7:43:46 PM
*** shutting down gRPC server since JVM is shutting down
*** server shut down
red@vm03:~/test/daehuikim/332project$
```

Project management

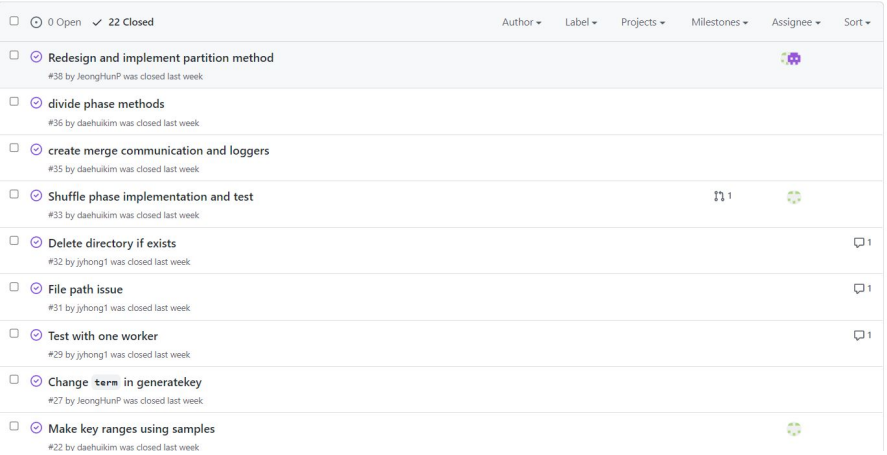
Feedback from progress presentation

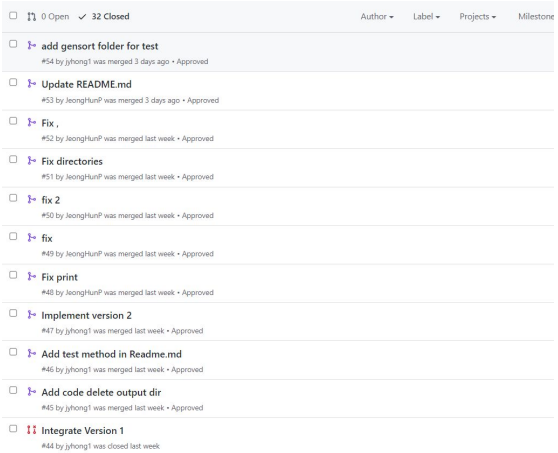
1. 주도적으로 개발을 리드할 사람이 정해지면 개발 속도에 더욱 탄력이 붙을 것 같다.
2. 실제 개발과 관련된 코드를 커밋해보면서 코딩하면서 하루빨리 발생할 문제에 대해 해결해보길 바란다.
3. docs 남길 때 개발에 직접적으로 도움되는 문서 위주로 남기는 것이 더욱 도움 될 것 같다.
4. 4주차치고 아직 진행도가 늦은 상태로 직접 개발에 시작해보는 것이 중요하고 마일스톤도 구체적이면 좋겠다.

Together, democracy (before mid-feedback)
→ Leader, distribute works (after mid-feedback)

Use github issue/PR system for each works

- **Leader sets each issues to members and every codes are reviewed by PR**





Project management (Milestones)

- General setup (input generation - jaeyoung Hong, grpc/project setup - jeonghun Park)
- Connection phase (jeonghun Park)
- Sampling phase (daehui Kim)
- Sort/partitioning phase (jeonghun Park / jaeyoung Hong)
- Shuffling phase (daehui Kim)
- Merge phase (jeonghun Park)
- Refactor (Everyone)
- Test and Verify (jaeyoung Hong)

Project Review

Good

- Anyway we made success on experiment and tests!
- Communications
- Select leader to control project
- Use github issue/PR systems for distributing works and reviewing codes

→ Progress-feedback was really helpful.

Bad

- Performance (e.g. partitioning algorithm)
- Too late to start implementation
- Lack of test codes/invariants
- More efficient design?

Project Review

1. Don’t be optimistic

2. Spending a lot of time on design is not a waste (that’s why implementation time is ⅙)

3. Leader-based system is much better than democracy

4. Unit tests and invariants are really powerful

Q/A