

SD progress presentation

team white

Contents

1. how we manage the project?
2. master-worker connection
3. sorting algorithm
4. project class structure - class diagram
5. weekly progress & milestone

1. Management

The screenshot shows a dark-themed interface of a project management or documentation tool. On the left, there's a sidebar with the following sections:

- Skills & Tech Stack**
 - 1. 프로젝트 관리 : Git bash / Git Kraken (version control)
 - 2. 커뮤니케이션 : Discord, Kakaotalk, 대면 미팅
- Packages**
 - TBD
- Git**
 - 1. Commit / PR 컨벤션
 - Feat : 새로운 기능 추가
 - Fix : 버그 수정
 - Design : 디자인 및 UI 수정
 - Docs : 문서 (README, 메뉴얼 등)
 - Test : 테스트 코드
 - Refactor : 코드 리팩토링 (기능 변화 없이 성능 개선)
 - Style : 코드 의미에 영향을 주지 않는 변경 사항
 - Chore : 빌드, 설정 파일
 - Comment : 주석 추가
 - 2. Git 브랜치
 - main : 기본 브랜치
 - develop : 개발된 기능(feature)을 통합하는 브랜치
 - docs : 문서작업 브랜치
 - feat/[issue_num]-[function_name] : 각 기능별 개발을 진행하는 브랜치
 - bug/[issue_num]-[bug name] : 버그 해결용 브랜치
 - test/[name] : 테스트용 브랜치
- Authors**
 - 하승재, 전재영, 채승현

git convention

github: version control system

+ gitkraken, fork

discord: save file, online meeting

kakaotalk: simple communication

google drive: make ppt file, report

boardmix: make UML

2.a. sampling

```
j Jong@DESKTOP-0300964: /test$ gensort -a 1000 test_gensort
j Jong@DESKTOP-0300964: /test$ ls
test_gensort
j Jong@DESKTOP-0300964: /test$ cat test_gensort
AsFGHMsom 00000000000000000000000000000000 0000222200002222000022220000222200002222000000001111
-sHd0jdv6X 000000000000000000000000000000001 77779999444488885555C0CC777755555555B0BB666644446666
u1^EYm8s|= 000000000000000000000000000000002 CCCCCCFF77799995555FFF111122229998884444DDFFFF
Q)JN)R9z-L 000000000000000000000000000000003 FFFF111100000000000066668888BBB33333333AAA1111CCCC
o4FoBkqERn 000000000000000000000000000000004 7777AAABBBBBBBB222244444444499995555BBB111188880DDD
*]-Wz1;TD- 000000000000000000000000000000005 AAAA88883333BBBB888888844447772227779999000222
0fsxx)-{ob 000000000000000000000000000000006 FFFF99997774444AAA777EEEEDDDAAAAAAA99998888BBBB
mz4VCN@#f" 000000000000000000000000000000007 DDDDBBBB1111FFF2222D0DDFFFB BBBFF666644447778888
my+-r57(N| 000000000000000000000000000000008 22226666CCCC66662222FFF0000EEE111888844455559999
SHA\z%t(% 000000000000000000000000000000009 0000AAA8888FFF000088800000002225551111FFFESEE
.PkXQ+&cc 00000000000000000000000000000000A 66667777FFFFFFF2222FFF3333FFF22224444DDDD77777777
GLSlnlm0** 00000000000000000000000000000000B 5555EEE111BBBB55555555AAAABBB33335555BBB11114444
swBzQ#t' K< 00000000000000000000000000000000C DDDDEEE111BBBB55555555EEE111BBBBEEECCCCEEE44466665555
9SCz G(10 00000000000000000000000000000000D BBBB1115555333DD1115555EEE111BBBBDDDD3333AAA
07-drsiz'L 00000000000000000000000000000000E BBBB BBBFFF1118884444333388886666DDDCCCAAA3333
X-%q4)YDS/ 00000000000000000000000000000000F 111BBBB111BBBBAAFAFFAAA666622223339999FFFF0000
gjm%556*d2 000000000000000000000000000000010 7777EEEEEE2222222333BBBB888866665555DDDBBBB1111
~&2ch[E*nf 0000000000000000000000000000000011 6666DDDDFFFFFFF4444EEE1116666BBB77776666BBB6666
]az'~,CNd 0000000000000000000000000000000012 66661117773333333CCCC999933322200033330000FFF
```

j Jong@DESKTOP-0300964: /test\$ valsort test_valsort
Records: 1000
Checksum: 1f5fdb3631a
Duplicate keys: 0
SUCCESS - all records are in order

```
[white@vm-1-master valsort]$ ls
valsort
[white@vm-1-master valsort]$ chmod +x valsort
[white@vm-1-master valsort]$ ls
valsort
[white@vm-1-master valsort]$ |
```

gensort

valsort

2.b. Build Libraries Setting

OS: window 11

+ **WSL**(CentOS 7 or Ubuntu)

Java: jdk 8 (JDK 8u202)

Scala: scala 3.3.7

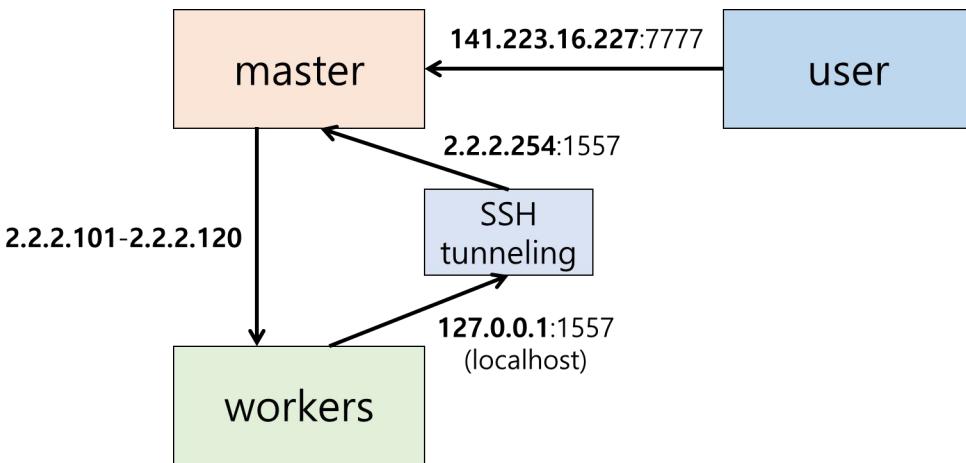
Libraries: Scalab, netty, protobuf,
gRPC

```
white@vm-1-master:~$ cat /etc/os-release
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
```

```
white@vm01:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="18.04.3 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.3 LTS"
```

master uses CentOS, **worker** uses Ubuntu

2.c. server connection test



could be automated with JSch

The screenshot displays three terminal windows:

- white@vm-1-master~**: Shows the command `java -cp 332project-assembly-0.1.0-SNAPSHOT.jar com.example.GreeterServer` running on port 1557.
- white@vm01~**: Shows the command `java -cp 332project-assembly-0.1.0-SNAPSHOT.jar com.example.GreeterClient` connecting to the master at port 1557, receiving a response "Hello from Master, Worker No. 1" from the master.
- white@vm-1-master~**: Shows the command `ssh -L 1557:localhost:1557 -p 7777 white@vm01` being run to establish an SSH tunnel from port 1557 on the master to port 1557 on the client.

connection succeeded

3.a. sample sort

step	to do
sampling	for each worker, sample random K data (ex. first 1000 of data).
partitioning	master collects and sorts all samples, then finds the pivots ($N-1$) to divide them into ranges corresponding to the number of workers (N). → informs this pivot information back to all workers.
shuffle	each worker sorts its data and compares this against the pivots provided. then send them to each worker.
merge	each worker bother the sended data and combine them by k-way merge.

in short, **quick sort with multiple pivots!**

3.a. sample sort - **if worker is dead**

if worker is dead, worker is going to resurrect soon.

but **input data will be still remained.**

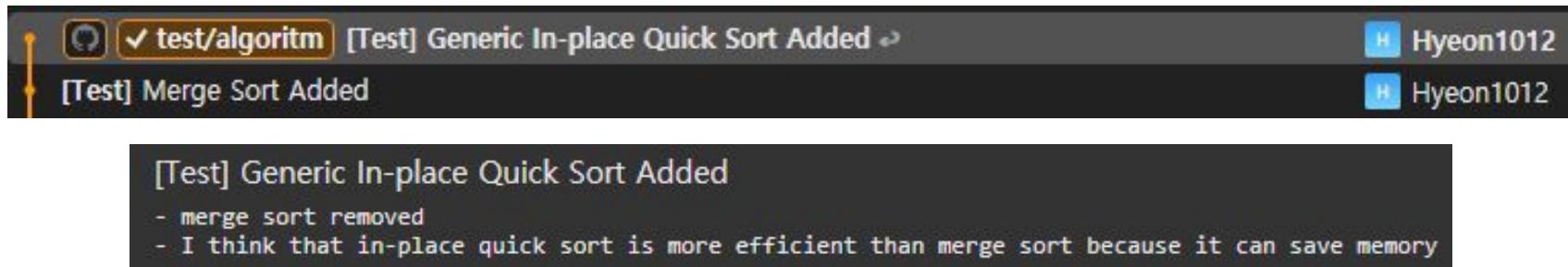
a resurrected worker gets the overall progress of sample sort at the master.

→ **do something** (for step 1 ~~~, for step 2 ~~~ , for step 3/4 ~~~)

key idea : worker creates and saves temporary files before send data to other worker

→ **even if one worker died, we can easily resume the shuffle / merge.**

3.b. k-way merge sort



```
[Test] Generic In-place Quick Sort Added
- merge sort removed
- I think that in-place quick sort is more efficient than merge sort because it can save memory
```

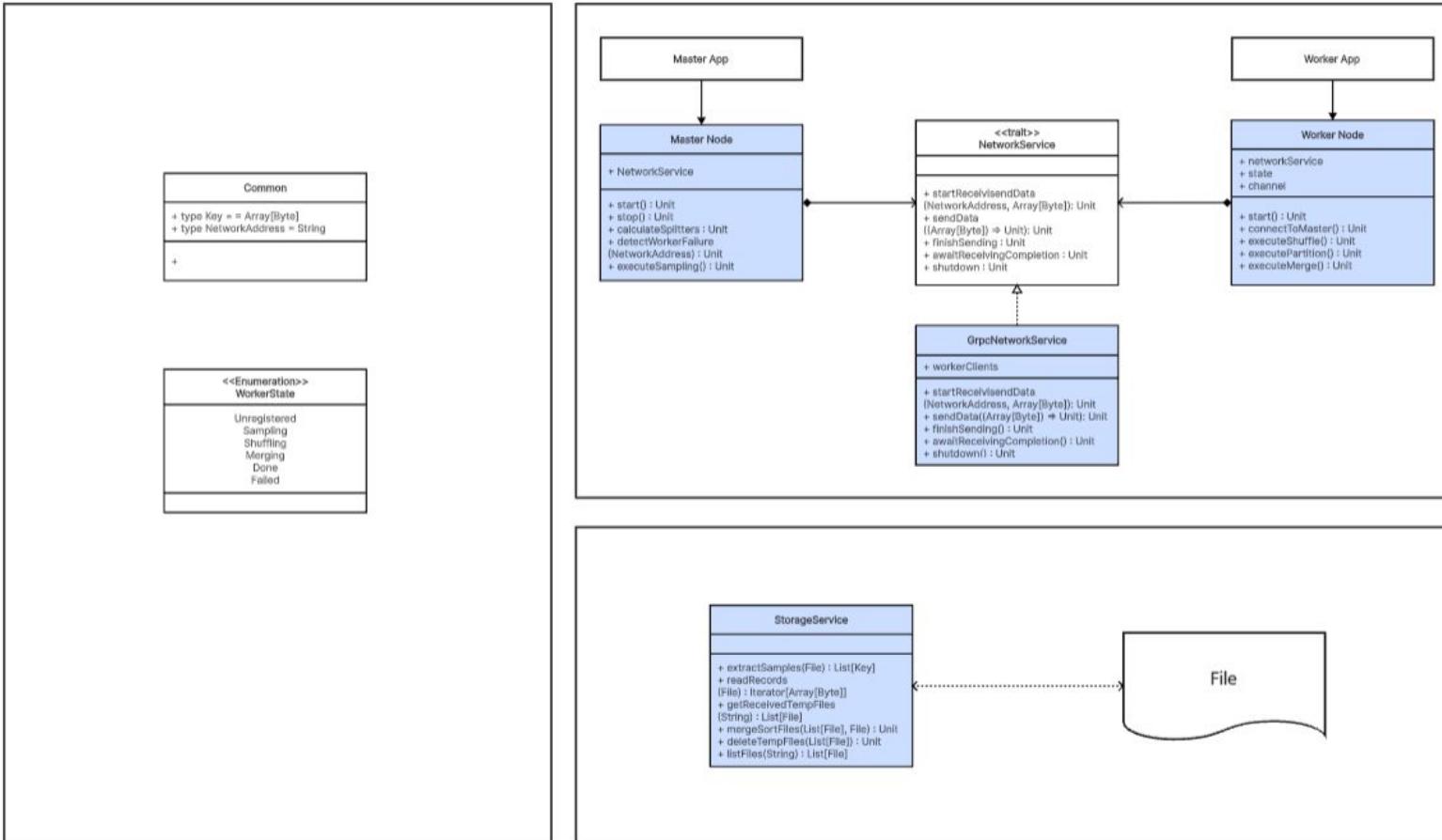
in fact, **quick sort is more efficient/faster than merge sort for smaller datasets**

so, I implemented **generic in-place quick sort**

but I noticed that I can use [java.util.Arrays.sort\(\)](#)

I have to implement k-way merge by using this

Class Diagram



Review of Weekly Progress

Week 1 Setup & Planning: Project initialization, GitHub repository setup, and establishing code conventions and meeting schedules.

Week 2 System Architecture: Designed the high-level system architecture and discussed core concepts for distributed sorting and tech stack.

Week 3 Environment & Tooling: Configured the development environment, secured VPN/IP access, and set up essential tooling (UML, IntelliJ).

Week 4 Core Prototyping: Began implementation with gRPC/protobuf (using ScalaPB), created Master/Worker prototypes, and generated sample data.

Week 5 Sorting Logic & Integration: Drafted class diagrams, implemented core sorting logic (QuickSort), and prepared for code integration and testing.

MileStone

Milestones

Open 5 Closed 0 Sort

<u>Test running program on multiple machines</u> ⚠ Overdue by 3 day(s) • Due by November 14, 2025 Due by December 5, 2025	0% complete 0 open 0 closed
<u>Implement distributed Sorting algorithm</u> ⚠ Overdue by 3 day(s) • Due by November 14, 2025 • 0/1 issues closed	0% complete 1 open 0 closed
<u>System test, all components in hands</u> Due by December 5, 2025	0% complete 0 open 0 closed
<u>Component test and early system test</u> Due by November 21, 2025	0% complete 0 open 0 closed
<u>Project Deadline</u> Due by December 7, 2025	0% complete 0 open 0 closed

New milestone

Milestones

Open 5 Closed 0 Sort

<u>Implement distributed Sorting algorithm</u> Due by November 25, 2025 • 0/1 issues closed	0% complete 1 open 0 closed
<u>Test running program on multiple machines</u> Due by November 25, 2025	0% complete 0 open 0 closed
<u>Component test and early system test</u> Due by November 28, 2025	0% complete 0 open 0 closed
<u>System test, all components in hands</u> Due by December 5, 2025	0% complete 0 open 0 closed
<u>Project Deadline</u> Due by December 7, 2025	0% complete 0 open 0 closed

New milestone