Distributed Sorting System

Advanced Programming

Team Green

Project Management

- Workflows
- Statistics
- Milestones

Design

- Remarkables
- Modifications

Experiments

- On Docker Containers
- On Real-World

Retrospective

- Minjae Gwon
- Tahyeok Ha
- Jiwon Lee

POSTECH

Project Management

Workflows and Statistics

POSTECH

Workflow

Project Management

- Convention
  - Adhered to coding, commit message, and documentation conventions.
- Communication
  - Online: Discord
  - Offline: Weekly Iteration Meetings
- Project Management
  - GitHub Project (https://github.com/users/betarixm/projects/2)
  - Tickets created and assigned during iteration meetings

Workflow

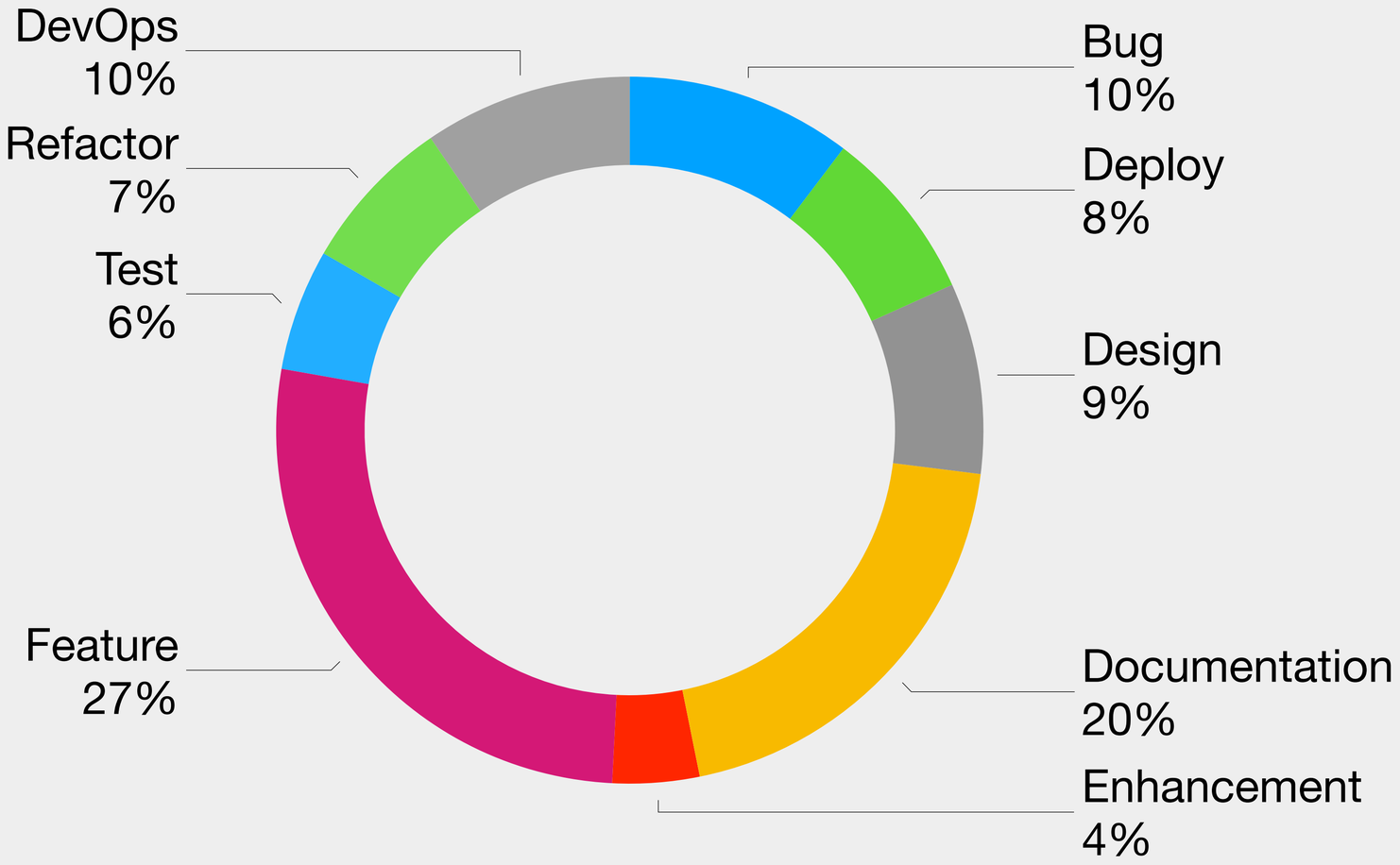
Project Management

- Git
  - Branch Rules; Merging with PRs and code reviews.
- Issue Tracking
  - GitHub Issues (https://github.com/betarixm/434project/issues)
  - GitHub Project Tickets converted to GitHub Issues
- CI/CD
  - GitHub Actions (https://github.com/betarixm/434project/actions)
  - Automated linting and testing on every push
  - Automated release triggered on a specific branch push

Statistics

Project Management

- 7 Iterations
- 65 Tickets
- 295 Commits
- 50 Issues
- 65 Pull Requests



Milestones

Details and Statistics

POSTECH

Data Abstraction

Milestones

11/9

100% complete 0 open 17 closed

- Key should be constructible from bytes.
- Record should be constructible from bytes.
- Record should be constructible from bytes.
- Block should be constructible from bytes or files.
- Block should be writable to a file.

Data Abstraction

Milestones

Data Abstraction 11

16	Implement `Blocks` #50	betarixm	Small
17	Block e2e test #53	hataehyeok	Medium
18	Implement `Record` #30	betarixm	Small
19	Define `Key` #29	betarixm	Small
20	Implement File Binding over `Block` #35	betarixm	Large
21	Define Type Aliases #31	betarixm	Tiny
22	Implement `WorkerMetadata` #32	leejiwon1125	Small
23	Implement `MasterMetadata` #33	leejiwon1125	Small
24	Define `KeyRange` class #48	betarixm	Small
25	Make `Partition` into tuple of `KeyRange` and `Block` #46	betarixm	Small
26	Refactor `block.partition` with `groupBy` #47	betarixm	Medium

+ Cannot add items when grouped by milestone

Sampling

Milestones

11/16

100% complete 0 open 1 closed

- Blocks should be able to be sampled.

Sampling

Milestones

Sampling 1			
32	Implement Sort and Sample over `Block` #34	hataehyeok	Large
+	Cannot add items when grouped by milestone		

Sorting

Milestones

11/16

100% complete 0 open 1 closed

- Keys should be comparable.
- Records should be comparable.
- Block should be sortable.

Sorting

Milestones

Sorting 1			
33	Implement Partition and Merge over `Block` #36	leejiwon1125	Large
+	Cannot add items when grouped by milestone		

Coordinating

Milestones

12/7

100% complete 0 open 19 closed

- Workers should be able to make partitions and merge them.
- Each client should be able to communicate with server.
- Workers should be able to exchange their partitions.
- Worker should be able to process master’s requests.

Coordinating

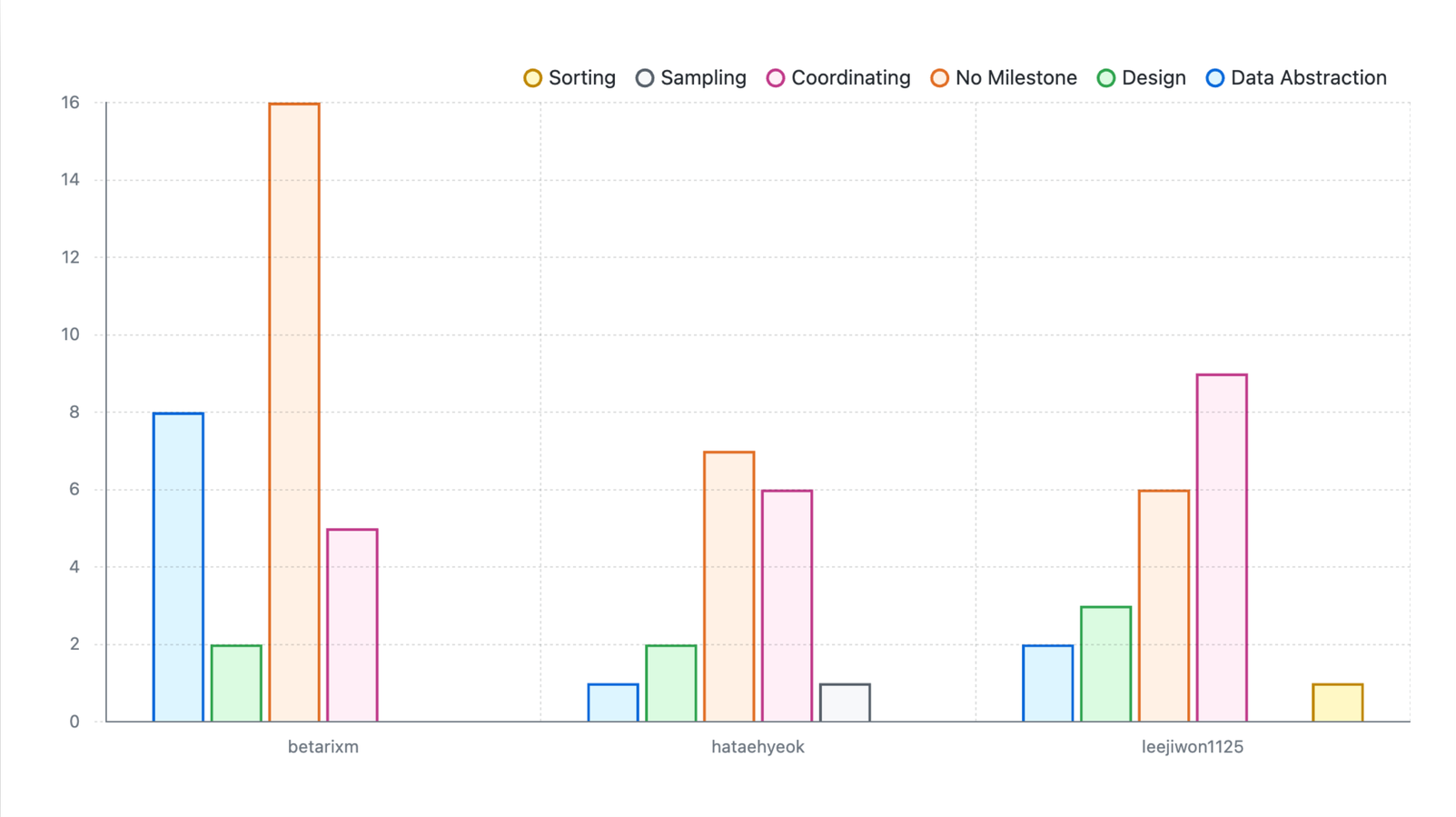
Milestones

Coordinating 15

1	Implement Master Server #65	leejiwon1125	Small
2	Implement Master RPC Service #55	leejiwon1125	Small
3	Implement Worker Server #63	hataehyeok	Large
4	Implement Exchange RPC Service #54	hataehyeok	Medium
5	Write gRPC `proto` #52	leejiwon1125	Large
6	Setup ScalaPB #51	betarixm	Small
7	Refactor Servers and Clients #75	betarixm	Large
8	Extract Key Range from Worker #76	leejiwon1125	Small
9	E2E Test (Small) #77	betarixm, hataehyeok, and leejiwon1125	Large
10	Exchange Server and Client #78	leejiwon1125	Medium
11	Grpc Test #80	hataehyeok	Medium
12	Logging #81	betarixm	Medium
13	E2E Test (Real World) #79	betarixm, hataehyeok, and leejiwon1125	X-Large
14	Implement Exchange Server #64	leejiwon1125	Medium
15	Set max size of partition #93	hataehyeok and leejiwon1125	

Statistics

Milestones



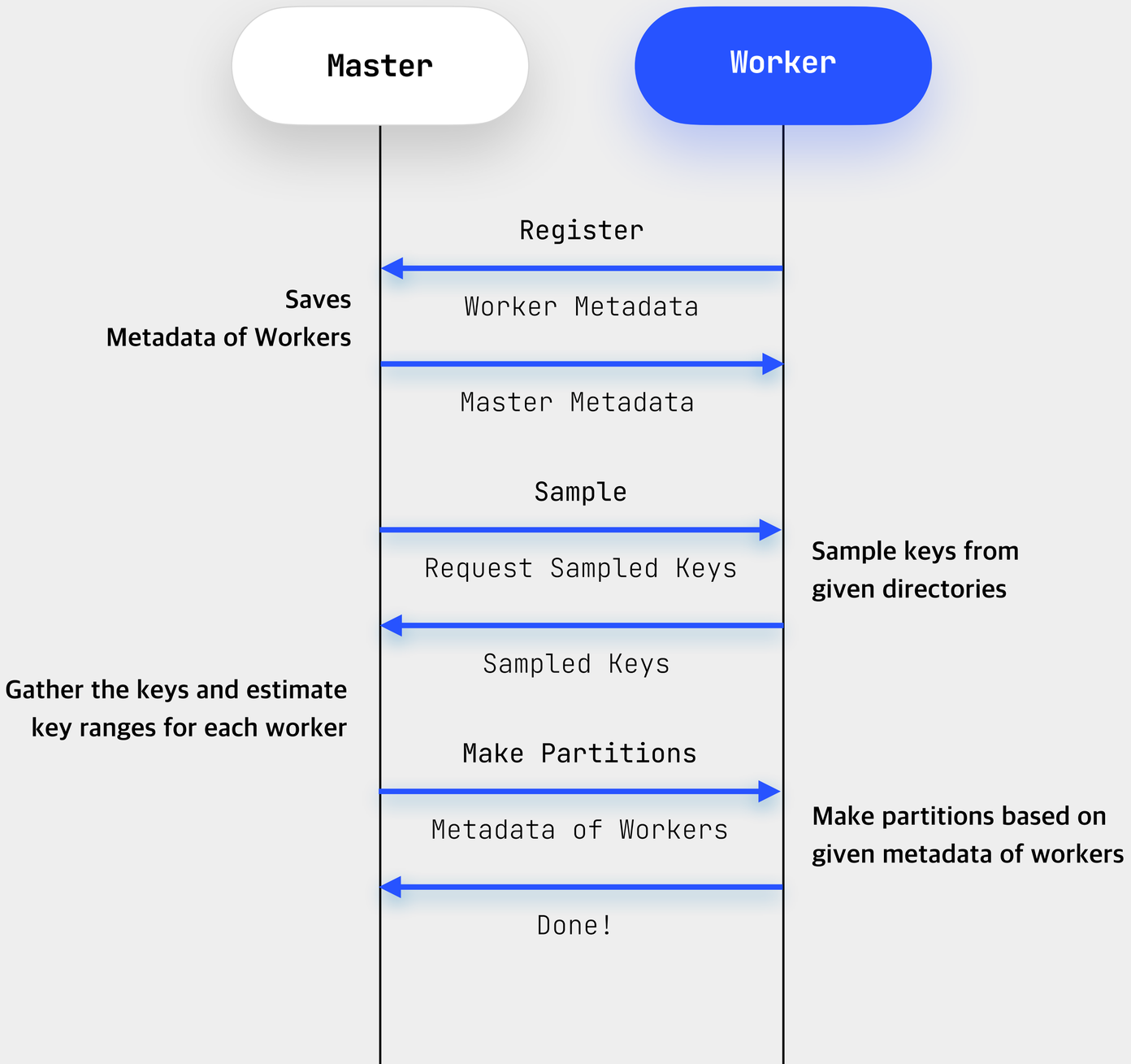
Design

Remarkables and Modifications

POSTECH

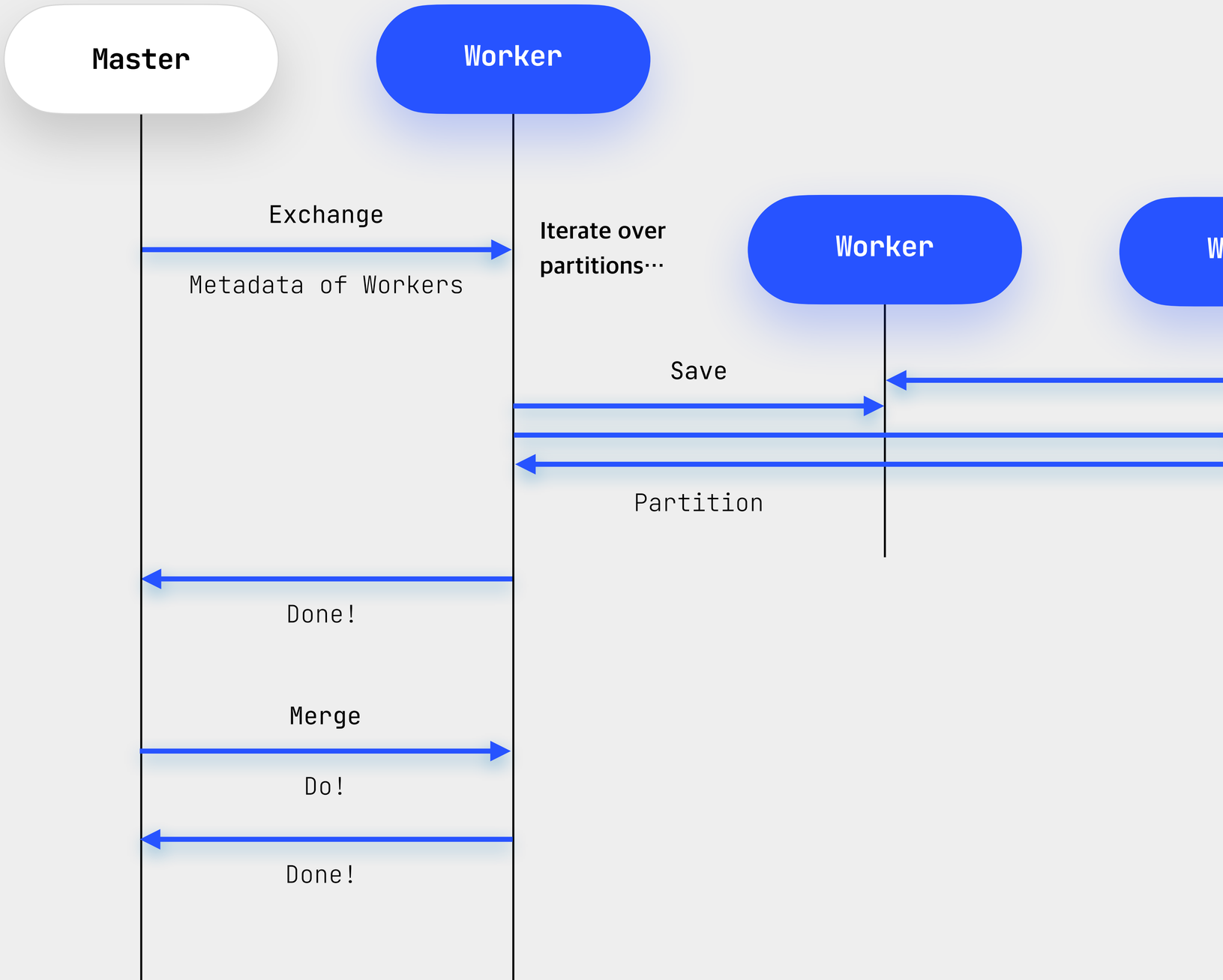
Overview

Design



Overview

Design



Remarkables

Design

- Dependency Injection
  - Separating gRPC Server and Services
- Hooks
  - Set of functions that depend on outer states
- Mutable State
  - Wrapper for Managing Mutable States
- Automated E2E Testing
  - Powered by Scalatest

Remarkable – Dependency Injection

Design

- Separate gRPC Server and Service Implementation
- Class: GrpcServer
  - Responsible for managing the server
- Class: MasterService, WorkerService
  - Responsible for handling requests

Remarkable – Dependency Injection

Design

```
class GrpcServer[T <: ServerBuilder[T]](
    service: ServerServiceDefinition,
    port: Int
) {
  private val server: Server =
    ServerBuilder
      .forPort(port)
      .addService(service)
      .build()
}
```

Remarkable – Dependency Injection

Design

```
class MasterService {
    // ...
}

class Master {
    def run() = {
        val server =
            new GrpcServer(
                MasterService(),
                port
            )
    }
}
```

```
class WorkerService {
    // ...
}

class Worker {
    def run() = {
        val server =
            new GrpcServer(
                WorkerService(),
                port
            )
    }
}
```

Remarkable – Hooks

Design

- Set of functions that depend on outer states
  - In other words, functions whose result value changes with each call.
  - Grouped apart from pure functions to be separated.

```
object Hooks {
  def useLocalHostAddress: String = // ...

  def useAvailablePort: Int = // ...

  def useTemporaryDirectory: Directory = // ...
}
```

Remarkable – Mutable State

Design

- Wrapper for Managing Mutable States

```
class MutableState[A](var underlying: A) { self ⇒
  def update(f: A ⇒ A): MutableState[A] = {
    synchronized {
      underlying = f(underlying)
    }
    self
  }

  def get: A = underlying
}
```

Remarkable – Mutable State

Design

```
class MasterService(
    mutableWorkers: MutableState[List[WorkerMetadata]]
) {
  override def register(request: RegisterRequest): Future[RegisterReply] = {
    // ...
    mutableWorkers.update(_ :+ worker)
    // ...
  }
}
```

Remarkable – Automated E2E Testing

Design

- Implemented end-to-end (E2E) Testing using a small dataset
- Powered by Scalatest
- Runs on Each Push within the CI/CD Pipeline

Run tests 48s

```
[info] compiling 1 Scala source to /home/runner/work/434project/434project/e2e/target/scala-2.13/test-classes ...
[info] done compiling
172.17.0.1:36207
172.17.0.1, 172.17.0.1, 172.17.0.1
[info] SystemSpec:
[info] System
[info] - should be working
[info] Run completed in 8 seconds, 917 milliseconds.
[info] Total number of tests run: 1
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 1, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 11 s, completed Dec 12, 2023, 2:42:22 PM
```

Modifications

Design

- Sampling
  - Previously: Sampling at intervals
  - Now: Take records from the head
- Data Exchange
  - Previously: Sending one Block
  - Now: Sending one Block with multiple chunks

Experiments

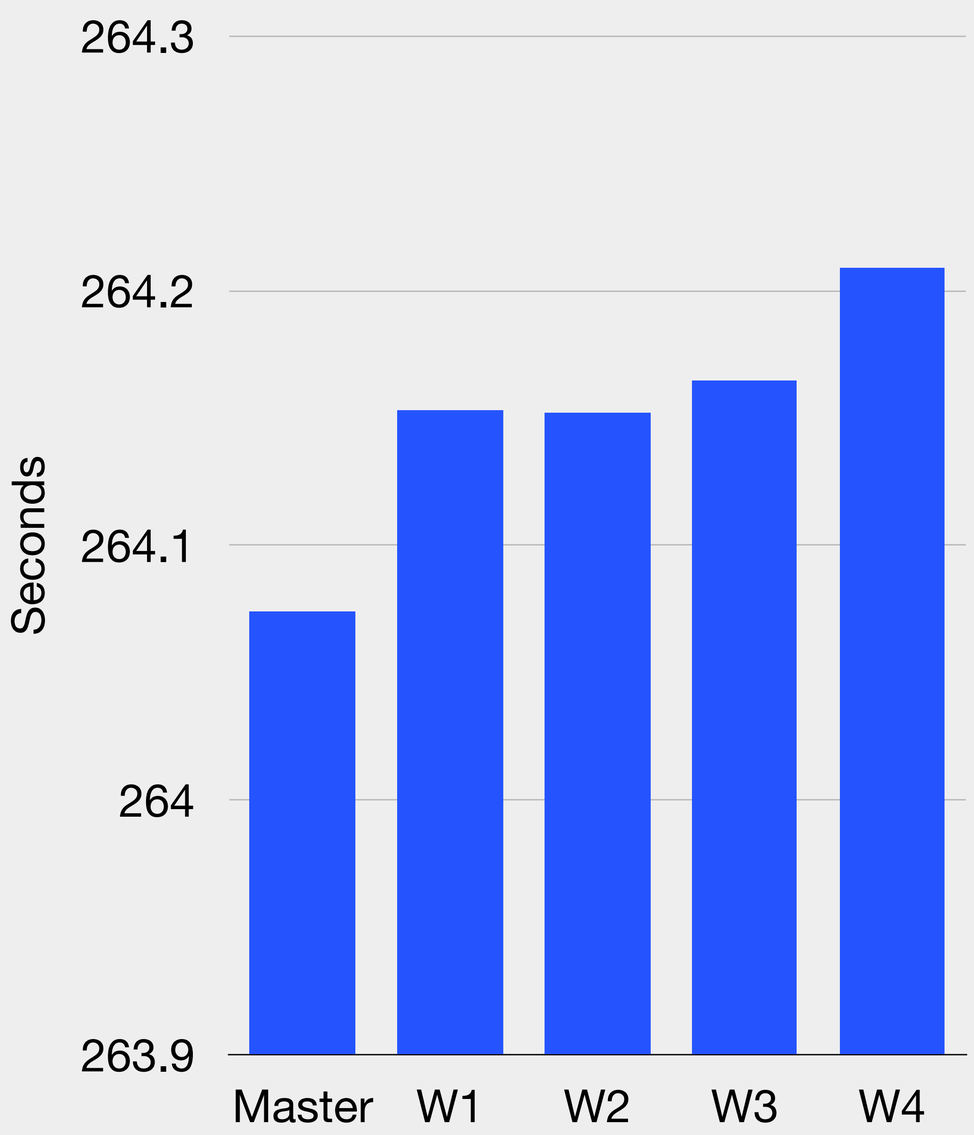
Simulation and Real World

POSTECH

On Docker Containers

Experiment

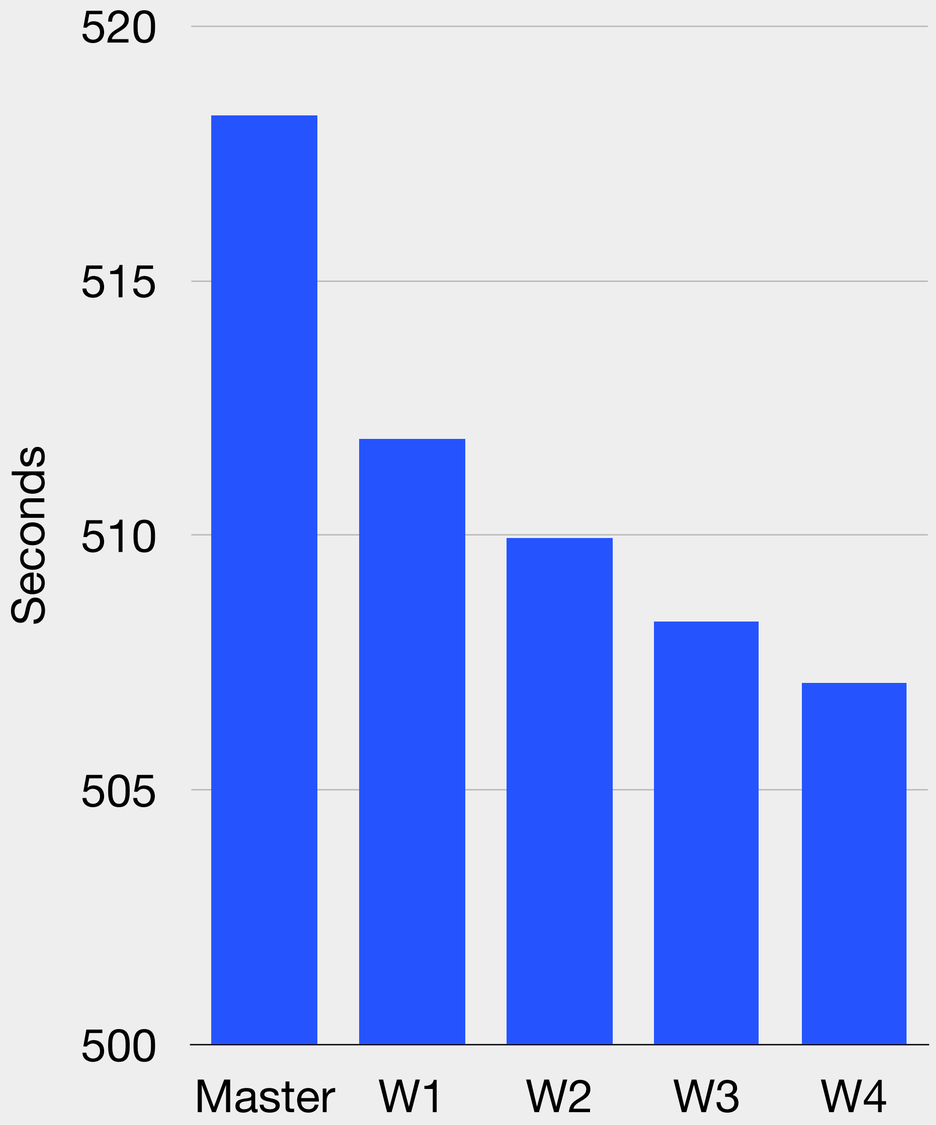
- Simulated Cluster with docker-compose
  - To check locally before running on the real-world
- Scenario
  - 4 Workers
  - 2 Input Directories
  - 2 Files in Each Directory
  - 340,000 Records in Each File



On Real-World

Experiment

- An Actual Cluster
  - Including Seconds within Manual Execution
- Scenario
  - 4 Workers
  - 1 Input Directories
  - 2 Files in Each Directory
  - 300,000 Records in Each File



Retrospective

What We Learned

POSTECH

Minjae Gwon

Retrospective

- Challenges in Ticket Sizing
  - Utilized tickets as the primary unit of work measurement.
  - Frequently encountered inaccuracies in estimating ticket sizes.
  - Resulted in an uneven distribution of workload among team members.
- Recognizing the Limitations of Code Reviews
  - Merged codes based on reviewed Pull Requests (PRs).
  - Observed the occurrence of bugs even after thorough code reviews.

Minjae Gwon

Retrospective

- The Power of Data-Oriented Thinking and Data Abstraction
  - Started from abstracting data structures.
  - Other implementations followed seamlessly.
- Uncomfortable Branching Strategy
  - Initially employed the main-develop branch strategy, which proved unsuitable for current situation.
  - Emphasized the importance of establishing appropriate branch rules for improving Developer Experience (DX).

Taehyeok Ha

Retrospective

- Was Able to Work by Lowering Communication Costs
  - Wrote design documents and continuously following code changes through code reviews.
  - 디자인 문서를 작성하고 코드에 관한 변경 사항을 코드 리뷰로 지속적으로 따라가는 등 communication cost 를 낮추며 작업을 해볼 수 있었다.
- Was Able to Contemplate Aspects such as Cost and Efficiency
  - Had the opportunity to experience the entire process of handling a project from scratch to packaging.
  - Allowed me to gain insights into the workflows of many real-world projects.
  - 하나의 프로젝트를 처음부터 패키징까지 다뤄보며 많은 현업 프로젝트들이 진행되는 과정을 경험할 수 있었고 그 과정에서 cost, efficient 등에 대해 생각해볼 수 있었다.

Taehyeok Ha

Retrospective

- Understands How to Fix Bugs Efficiently
  - Was able to debug by tracing the logs I placed in the code.
  - Understood how to formulate hypotheses and trace program execution.
  - 디버깅을 할때 코드에 넣은 로그를 추적하며 고칠 수 있었고 어떻게 가설을 세우고 프로그램을 추적하며 에러를 고칠 수 있는지를 알 수 있었다.

Jiwon Lee

Retrospective

- Understands How to Debug Easily
  - Made debugging became more manageable by utilizing assert statements and keeping memos for each experiment.
  - Assert문을 활용하고, 각 실험에 대한 기록을 남기며 진행하니 디버깅이 생각보다 수월했다.
- Recognized Challenges in Achieving Mutual Understanding
  - Realized that there were differences in understanding between each other during the design process, making it difficult to align perspectives.
  - 초기 디자인 과정에서 문제 상황에 대한 서로의 이해를 서로가 이해하는 것이 결코 쉬운과정이 아님을 깨달았다.

Distributed Sorting System

Advanced Programming, 2023.

Gwon Minjae, Dept. of Computer Science & Engineering, POSTECH.
Lee Jiwon, Dept. of Computer Science & Engineering, POSTECH.
Ha Taehyeok, Dept. of Computer Science & Engineering, POSTECH.

POSTECH