최종결과보고서

KUORA

소셜 서비스를 위한 분산 스토리지

(KU + stORAge)

지도교수

유준범 교수님

9팀

컴퓨터공학과 201411277 서지원

컴퓨터공학과 201511305 홍나리

목차

프로젝트 요약

개요

흔히 접하는 instagram과 유사한 SNS 서비스에는 인기가 많은 등의 요인으로 트래픽이 몰리는 Hot data와 그렇지 않은 Cold data라는 개념이 있다. 이러한 서비스들은 많은 양의 저장소를 요구 하므로 이를 분류하여 저장하는 것은 트래픽과 저장소 가용률에 이점을 가질 수 있다 이를 위하여 Google에서 제시한 GFS의 디자인에 따라 해당 목적의 분산 파일 저장소 시스템을 구현해보고자 한다. 서비스 특성상 최근 업로드한 데이터가 hot data가 되는 경향이 많으므로, 기본적으로 시간을 기준으로 data를 나누고자 하며, 가능하다면 추가적으로 reference count를 통하여 hot/cold data간의 migration을 할 수 있도록 구성하였다. 개발의 편의성을 고려하여 golang으로 개발하였고 이런 GFS를 구현한 대표적인 구현체로는 Hadoop 의 HDFS가 있다.

목표

- hot/cold data의 분산 저장을 위한 DFS 구현
- GFS에서 필수적으로 요구하는 R/W와 Delete 기능 구현
- Replication 구현
- Re-Replication 구현

배경이론

Google이 제시한 구글파일시스템(GFS)은 크게 Client, Master Server, Chunk Server로 구성되어있다.
Master Server에서는 실제 파일 시스템처럼 Name space Tree로 metadata을 관리하고 실제 데이터에 대한 관리는 Chunk Server에서 담당한다. 여기서 데이터는 고정된 크기로 나뉜 Chunk 단위로 다뤄지며, metadata에는 파일과 Chunk의 name space, 매핑 정보, 그리고 각 Chunk 복제본들의 위치정보 등이 담겨있다.
따라서 Client가 어플리케이션을 통해 파일 이름이나 Chunk 인덱스를 Master Server에 전달하게 되면 Master Server는 해당 Chunk의 위치를 Client측에 전송하게 되고, 결론적으로 Client는 이 정보를 통해 Chunk Server에 직접 접근할 수 있게 되는 것이다.
이외에도 Master Server와 Chunk Server 사이의 Chunk instruction이나 Chunk Server의 상태를 주기적으로 전송하는 HeartBeat 등의 부가적인 기능들이 존재한다.
위에서 설명한 GFS의 대략적인 구조는 다음과 같다.

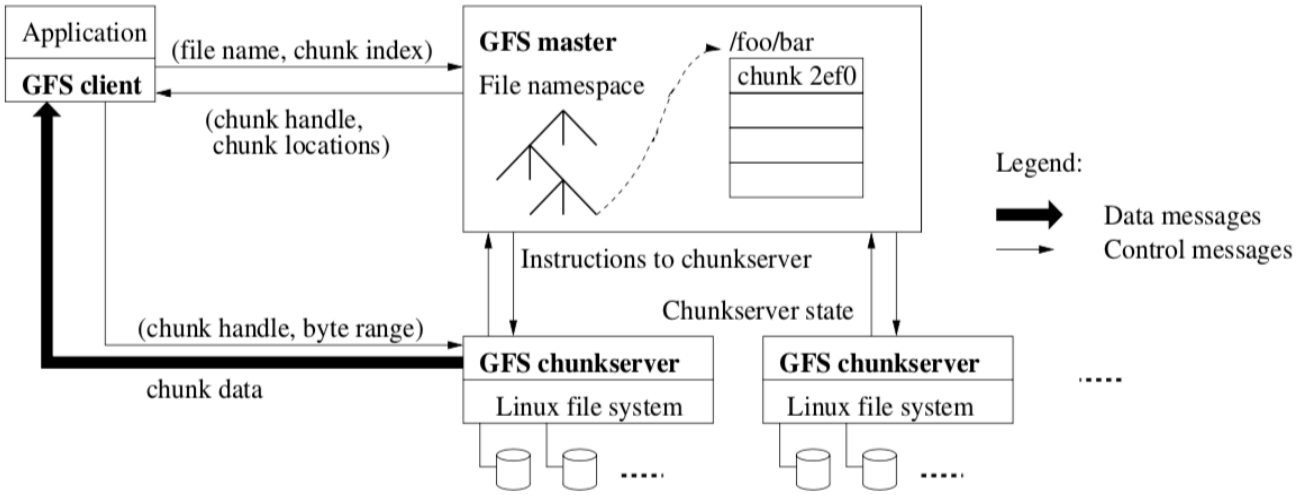


Figure 1: GFS Architecture

또한 GFS에서는 Data Server에 발생할 혹시 모를 장애에 대비해 다음과 같은 기능들을 제공한다.

1. Replication
- Master Server에 의해 Primary로 선정된 Data Server는 데이터의 Write 요청이 있을 때 가장 먼저 Write된다. 이를 성공적으로 끝마쳤을 시 Primary Data Server는 다른 Secondary Data Server에 데이터의 복제본(Replica)을 전달하여 Write하도록 한다.

2. Re-Replication
- Master Server는 Data Server로부터 Heartbeat를 통해 그들의 상태를 주기적으로 전송받는데 만약, 일정 시간동안 Data Server로부터의 Heartbeat가 전송되지 않을 시 Master Server는 해당 Data Server에 장애가 생긴 것으로 간주한다.
- 따라서 장애가 난 Data Server의 데이터를 가진 다른 Data Server는 해당 데이터를 가지고 있지 않은 다른 정상 Data Server로 데이터의 복제본(Replica)를 전달하여 Write하도록 한다.

다음은 위 프로세스를 묘사한 그림이다.

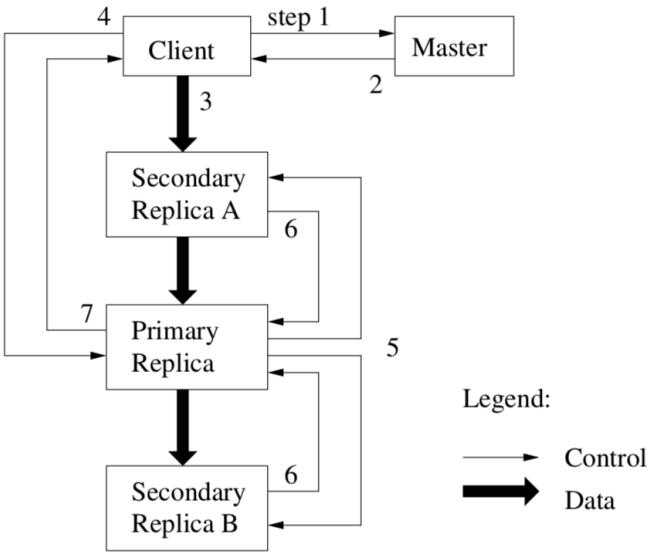


Figure 2: Write Control and Data Flow

우리는 이러한 GFS의 디자인을 유사하게 가져가되, 몇가지 차이를 두었다. 우선 Master Server에서의 관리 방식을 Name space방식이 아닌 key value 방식으로 구현하고자 하였다. 그리고 그 이유는 다음과 같다.

1. Name space의 병목현상

파일들의 write나 path에 대한 lock이 발생할 수 있다. 파일이 많아질수록, path가 깊어질수록 병목현상은 심화되므로 이에 대한 해결책으로 유니크한 key value 방식을 사용하고자 한다.

2. 중복문제

Client가 metadata를 요청하는 과정에서 Hot data를 요청하는 과정 그대로 Cold data를 요청하게 하기 위함이다. 더 자세히 설명하자면 Client가 a 라는 파일을 요구할 때, Hot storage에서 cold storage로 Migration하더라도 같은 a라는 결과를 얻고 싶은 것이다. 기존의 Name space방식의 경우, Master Server에서 a라는 파일의 정보를 계속 보유하고 있어야 하기때문에 중복문제를 다루기가 까다롭지만 각 데이터가 유니크한 Key-value를 가지게 된다면 이같은 중복문제는 해결될 것이다.

또한 우리는 한 번 Write된 데이터에 대해서 수정이 불가능하도록 하였다.

일단 GFS의 경우 수정에 대해 record append라는 한정적인 오퍼레이션을 지원하는데 이는 Chunk의 1/4사이즈 정도의 데이터 수정만을 허가한다. 하지만 우리는 SNS서비스를 타겟으로 하는만큼 모든 데이터를 미디어파일이라고 한정하고 파일에 대한 수정이 없는 것으로 간주한다.
metadata의 경우는 key value로 관리하고 Hot data는 기존 GFS대로 In-memory 방식을, Cold data는 디스크에 저장하는 방식을 따른다.

요구사항 분석

요구사항

A. FUNCTIONAL REQUIREMENTS

1. Client Side

1.1. Read File

- Client가 Master에게 Key를 통하여 File에 해당하는 Chunk의 위치를 요청한다.

- Client는 해당 정보를 통하여 Data Server로부터 파일을 읽는다.

1.2. Write File

1.2.1. Create

- Client가 Master에 파일의 Create를 요청한다.

- Master는 논리적인 File에 해당하는 Key를 만들고 Chunk의 생성을 DataServer에 요청한다.

- Master은 Key와 Chunk의 정보를 저장하고 생성된 Key를 Client에게 전달한다.

1.2.2. Write Chunk

- Client가 Key를 통해 Master에 Write를 요청한다.

- Master는 Chunk의 위치와 Primary와 Secondary 서버의 주소를 전달한다. (기본 복제본은 3개이므로 3개의 서버의 위치를 전달한다.)

- Client는 해당 정보를 통하여 Data Server의 Chunk에 정보를 write한다.

- 2.4 (Replication) 작업이 수행된다.

1.3. Delete File

- Client가 File Key를 통하여 Master에게 요청한다.

- Master는 해당 Key의 정보를 제거하고 해당 Chunk를 Garbage 리스트에 넣고 Data Server에게 전달한다.

2. System Side (Master + DataServer)

2.1. Heartbeat

- Data Server는 Master에게 자신의 서버 주소와 정상 동작 여부를 전송한다. (주기: 400 ~ 800ms)

- Master는 Heartbeat가 이루어지는 서버의 정보를 저장한다.

2.2. Expire/Migration

- 처음 저장되는 모든 File은 Hot Storage에 저장된다.

- Hot Storage의 Hot File의 Expire Time이 부여되며 이 시간이 지날 경우 Cold Storage로의 Migration이 시행된다.

2.3. Garbage Collection

- DataServer는 주기적으로 Master에서 전달된 Garbage 리스트의 Chunk에 대해서 가비지 컬렉션을 수행한다. (주기: 1 day – Configurable)

2.4. Replication

- 처음 파일이 생성되고 데이터가 쓰여질 때, Client는 Master가 지정한 Primary에 데이터를 쓴다.

- Primary로 선정된 Data Server는 그 외의 Secondary 서버들에게 데이터를 전달한다.

2.5. Re-Replication

- Master는 HeartBeat(2.1)을 통해 DataServer의 상태를 점검할 수 있다.

- DataServer가 장기간 Heartbeat를 보내지 못했을 경우, (약 10번의 Heartbeat) 해당 DataServer가 장애가 난 것으로 간주한다.

- 장애가 난 DataServer(Fault)가 가진 Chunk들의 데이터를 가진 다른 DataServer(D1)가 Chunk를 지니지 않은 다른 DataServer(D2)에게 데이터를 전달하여 복제본의 수(3)를 유지한다.

- 작업이 끝나면 Master는 DataServer(Fault)의 정보를 제외한다.

B. QUALITY REQUIREMENTS

1. Reliability

- Master는 새로운 DataServer가 실행될 경우 이를 시스템에 포함시킬 수 있다.

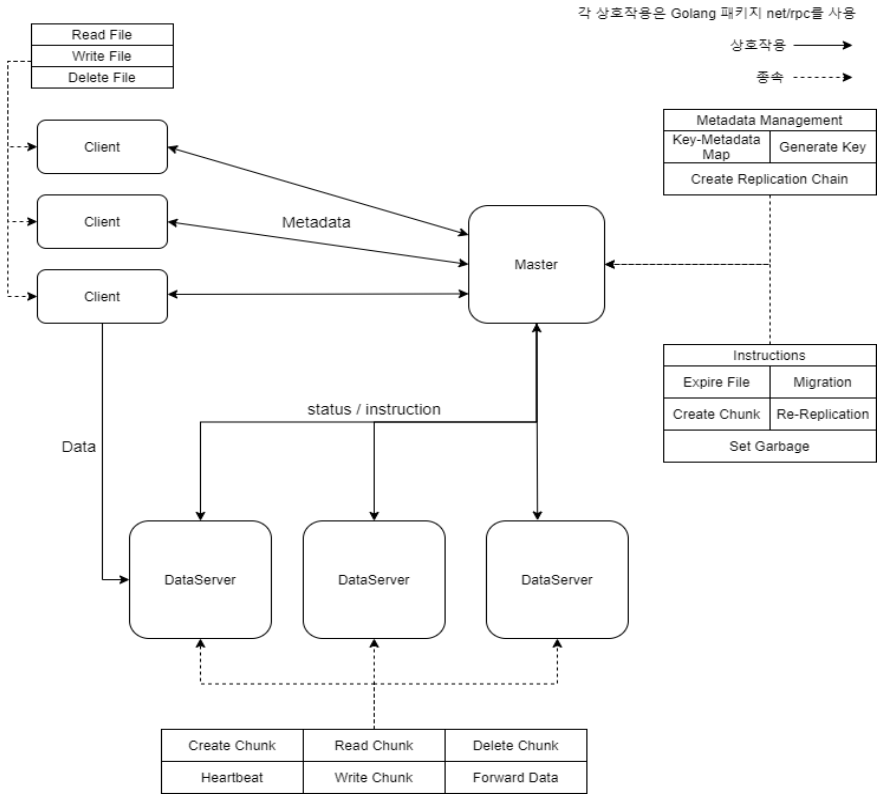
2. Atomicity

- 다수의 Client가 파일 스토리지에 접근할 수 있다.

- Master와 DataServer는 Chunk의 정보에 대하여 R/W Lock을 가진다.

- Lock을 통해서 각 Chunk에 대해서 Client의 연산에 대해서 Atomicity를 보장할 수 있도록 한다.

ARCHITECTURE DIAGRAM



A. CLIENT

3. Read/Write/Delete File

4. Client-Master 간의 Metadata 교환 후 Client-DataServer 간 직접 통신

B. MASTER

1. File Key와 Metadata간 Mapping 관리

2. DataServer의 정상 동작을 관리

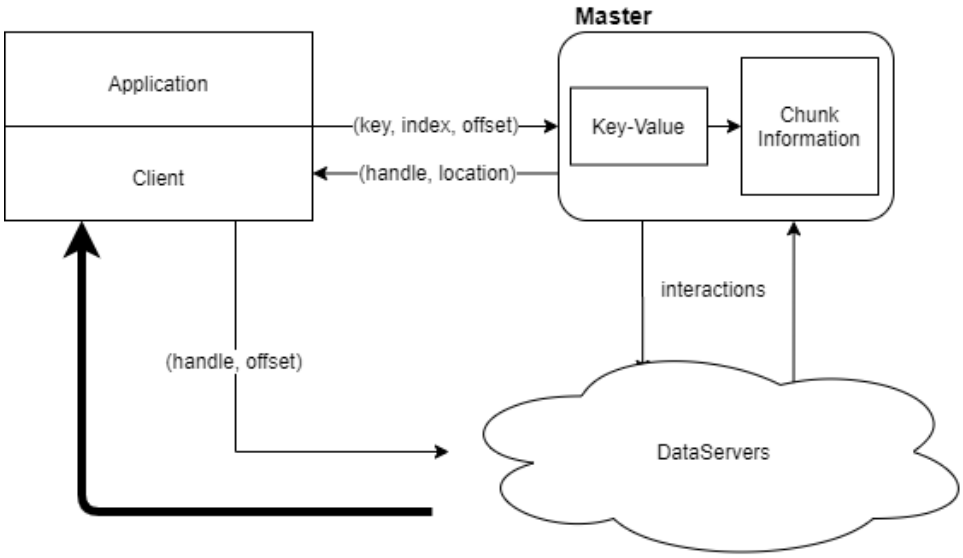
3. DataServer의 작업에 대한 지시

C. DATASERVER

1. 실질적인 파일의 데이터를 Chunk로 관리

2. 각 데이터에 대한 Create/Read/Write/Delete
3. 다른 DataServer로의 데이터의 전달

PROTOTYPE



A. CONCEPT

- Client는 Master에 metadata를 요청한 뒤 전달받은 metadata를 통해 Data Server에 직접 접근한다.
- Master는 Chunk에 대한 metadata와 key-value만을 가지고 있으며 실제 모든 데이터의 R/W는 클라이언트와 데이터서버의 통신으로 이루어진다.
- 앞서 언급했듯 파일에 대한 수정은 없는 것으로 간주한다. 또한 GFS와는 달리 key-value방식을 사용하였으며 Hot data는 메모리에, Cold data는 디스크에 저장한다.

B. SUCCESS CRITERIA

- hot/cold data storage간 data migration
- Read / Write / Delete 의 작동
- Heartbeat 포함 구현 요소의 작동
- Create 시 Replication
- DataServer 장애 발생 시 Re-Replication

C. TEST ENVIRONMENT

- 1개의 Master / 최소 3개의 DataServer / 1개 이상의 Client 필요
- Re-Replication을 테스트하기 위해서는 4개 이상의 DataServer가 필요
- 하나의 Local Machine에서 Maste
- r/DataServer/Client에 해당하는 프로세스를 각각 실행하여 테스트할 수 있다.AWS, Google Cloud Platform 등 클라우드 서비스를 이용하여 Multiple Machine의 환경에서 여러 개의 instance를 실행하여 테스트할 수 있다.

TEST CASE

No.	Name	Description
1.1	Read File	Client에서 Key를 통해 Read를 요청하였을 때 해당 데이터를 읽어오는가 여부를 확인
1.2	Write File	Client에서 새로운 파일을 만들고 Write 요청을 할 때 해당 데이터가 DataServer에 올바르게 저장되는가 여부를 확인
1.3	Delete File	Client에서 Key를 통해 Delete 요청을 처리 후 해당 데이터가 Garbage Collection 되는지 확인
2.1	Heartbeat	Master가 지속적으로 DataServer로부터 요청을 받아 DataServer가 정상임을 확인할 수 있는지 여부를 확인
2.2	Expire/Migration	Master의 지시에 따라 DataServer가 Expire에 의한 Delete 혹은 Migration을 수행할 수 있는지 여부를 확인
2.3	Garbage Collection	Configure된 주기에 따라 DataServer 내의 실제 Chunk 데이터가 삭제되는지 여부를 확인
2.4	Replication	Client의 Write 후 DataServer에 Chunk 데이터가 복제가 되었는지 확인
2.5	Re-Replication	DataServer 하나를 중지시킨 후 해당 DataServer가 가진 데이터가 다른 서버에 복제되었는지 확인
Q1	Persistent Metadata Test	Disk 내의 파일을 Serialize 하여 Metadata 정보가 동일한지 확인
Q2	Multiple Client Test	Client를 다수 실행하여 쓰기 및 읽기를 실행하였을 때 데이터의 동일성 및 Key의 유일성을 확인

TEST FLOW

A. CLIENT TEST

- Write File / Replication : Client의 요청에 따라 논리적인 File에 대한 Key를 발급하고 더미 데이터의 write와 복제되는 동작을 log를 통해 확인한다.
- Read File : Client가 Write하고 받은 key를 바탕으로 Read를 수행한다. 이 때 데이터가 쓰여진 3개의 서버 모두에서 read를 수행하여 Client가 쓴 파일의 데이터와 일치하는지 확인한다
- Delete File : Client가 Key를 통해 파일을 삭제하고 key를 Listing하여 삭제 여부를 확인한다. 또한, Garbage Collection 이후 해당 Chunk가 실제로 삭제되었는지 확인한다.

- 여러 Thread를 생성하여 Client의 기능을 수행하여 Multiple Client 테스트를 수행한다.

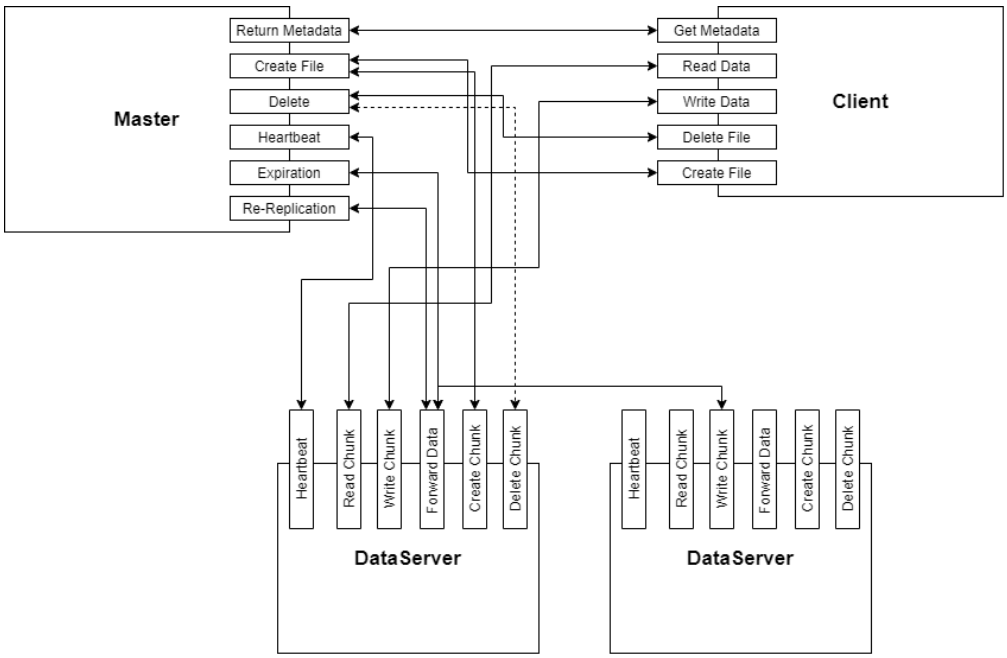
B. SYSTEM TEST

- Heartbeat : Master의 Log를 통해 확인
- Expire/Migartion/Garbage Collection : Master와 DataServer의 Log와 실제 DataServer에 접속하여 Linux 내 파일이 존재하는지 여부를 확인
- Re-Replication : DataServer 중 하나를 강제종료한 후 가지고 있던 Chunk들이 다른 DataServer로 복제되는지 확인
- Persistent Metadata : Master/DataServer 종료 후 정상적으로 disk의 metadata를 메모리에 저장할 수 있는지 확인

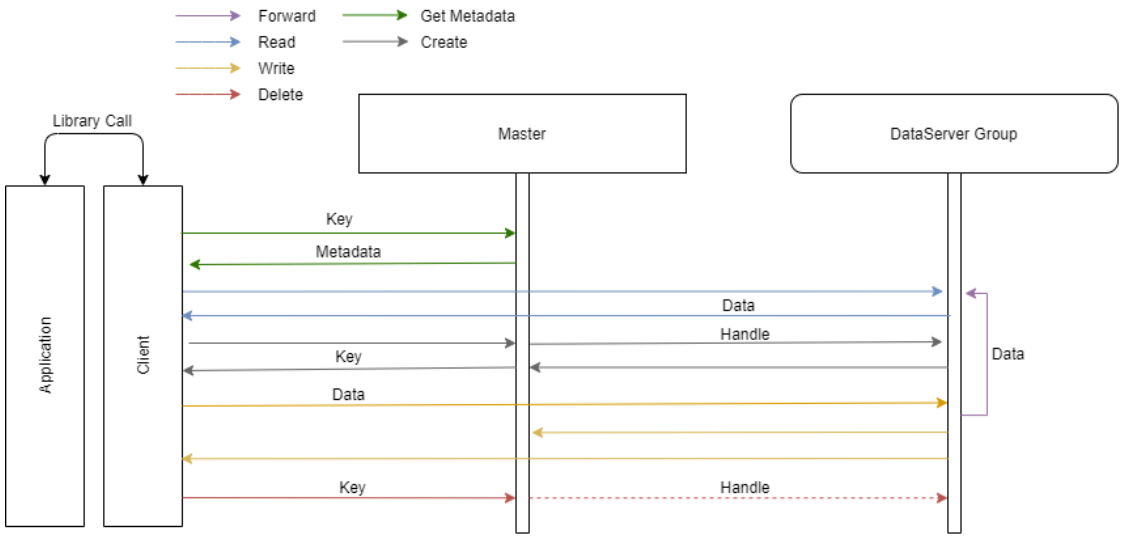
설계 내용

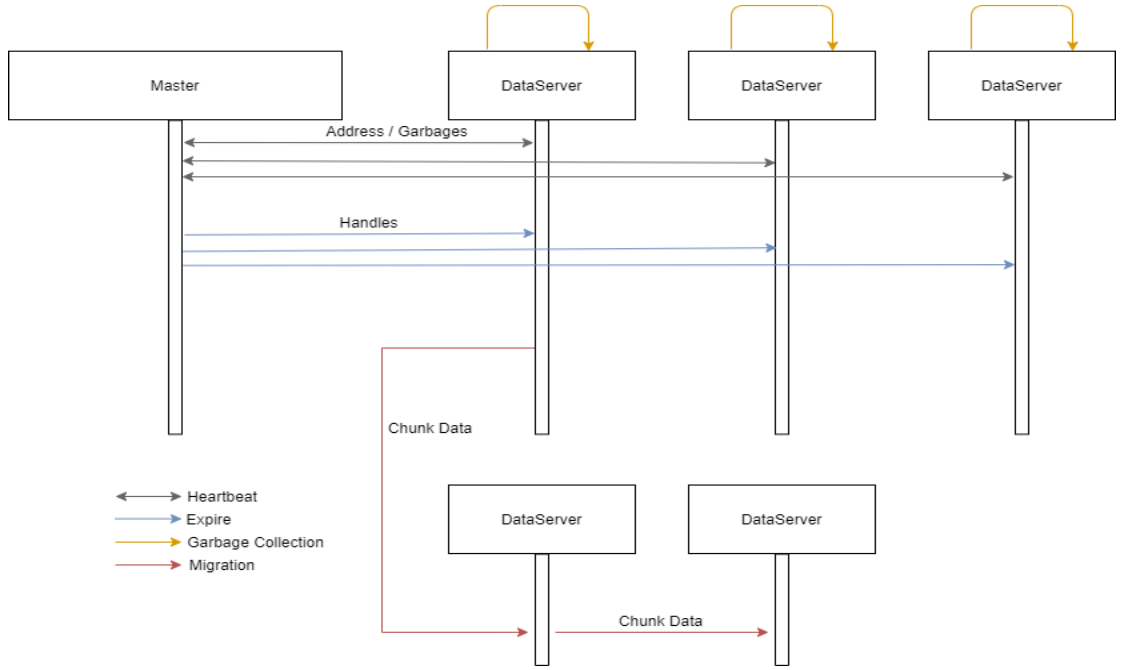
상위 디자인

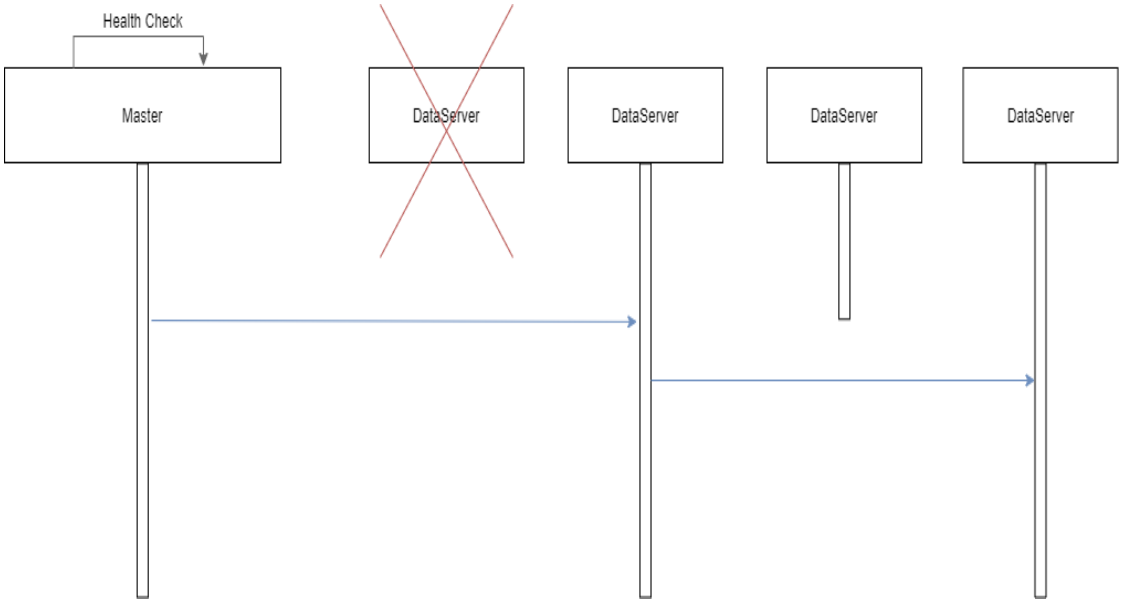
A. HIGH-LEVEL INTERFACES



B. SYSTEM SEQUENCE DIAGRAM







1. Read

   - Client와 Master 사이의 metadata 교환

   - metadata를 통해 Data Server에 직접 접근하여 데이터 read

2. Write

   - Master에 파일 Create 요청

   - key value를 통해 생성된 파일에 write

   - Replication 수행

3. Delete

- Master에 delete 요청
- Master가 Data Server에 delete 지시

4. Heartbeat

   - DataServer가 Master에게 Address 및 상태 전송

   - Master는 DataServer에 Garbage (Chunks) 전송

5. Expire/Migration

   - Master는 주기적으로 Chunk의 Expire 여부 체크

   - Expired Chunks를 Migration하도록 DataServer에 지시

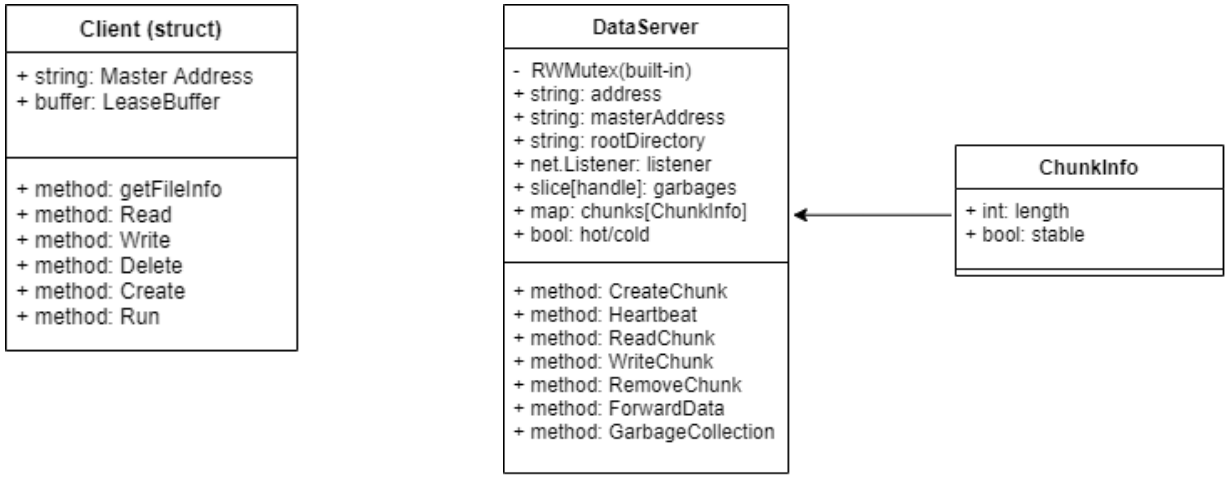
   - DataServer가 Migration 수행

6. Garbage Collection

   - DataServer가 Master에게 전송받은 Garbage chunks에 대해 주기적으로 Garbage Collection 수행

상세 디자인

A. CLASS DIAGRAM



1. Client

   1.1. getFileInfo ; Key를 통해 File의 Metadata 얻을 수 있다.

   1.2. Read : Key를 통해 Chunk를 Read할 수 있도록 요청한다.

   1.3. Write : Key를 통해 Chunk에 data Write를 요청한다.

   1.4. Delete : Key를 통해 File을 지우도록 요청한다.

   1.5. Create : Master에 File을 생성하도록 요청한다.

2. DataServer

   2.1. CreateChunk : Master의 지시를 통해 빈 Chunk를 생성할 수 있다.

   2.2. Heartbeat : Master에 주기적으로 현재 DataServer의 상태를 전송한다.

   2.3. ReadChunk : Client의 요청에 의해 Chunk의 데이터를 읽어서 전송한다.

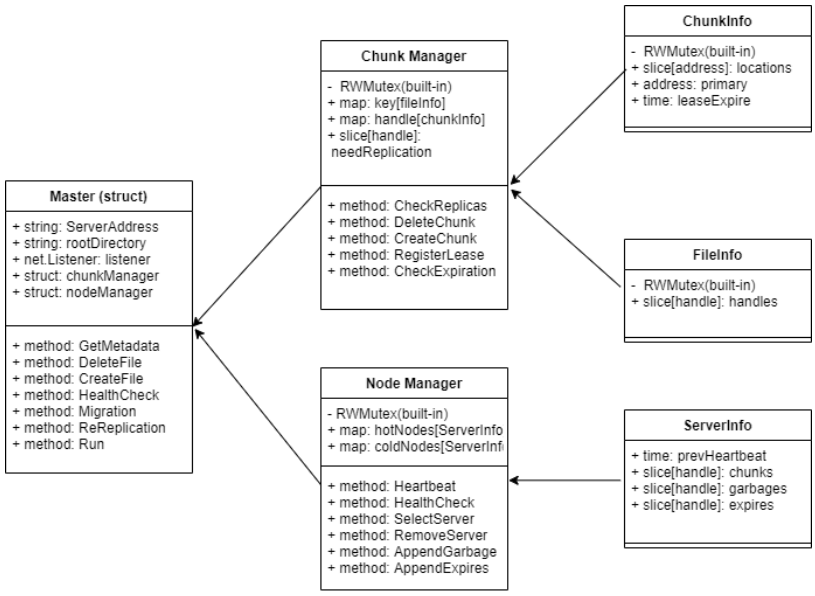
   2.4. WriteChunk : Client의 요청에 의해 Chunk에 데이터를 쓸 수 있다.

   2.5. RemoveChunk : Chunk를 삭제한다.

   2.6. ForwardData : 다른 DataServer로 Chunk data 전송한다.

   2.7. GarbageCollection : 주기적으로 garbage collection 실행하도록 한다.

3. Master



3.1. GetMetadata : Client의 요청에 따라 Metadata 제공한다.

3.2. DeleteFile : Key와 함께 논리적인 File을 삭제한 뒤, DataServer에 Garbage 전송한다.

3.3. CreateFile : Client의 요청으로 Key와 함께 논리적인 File을 생성한다.

3.4. HealthCheck : DataServer로 부터 전송받은 Heartbeat를 통하여 각 서버의 상태 체크한다.

3.5. Migration : Expired Chunks를 파악하여 Migration을 지시한다.

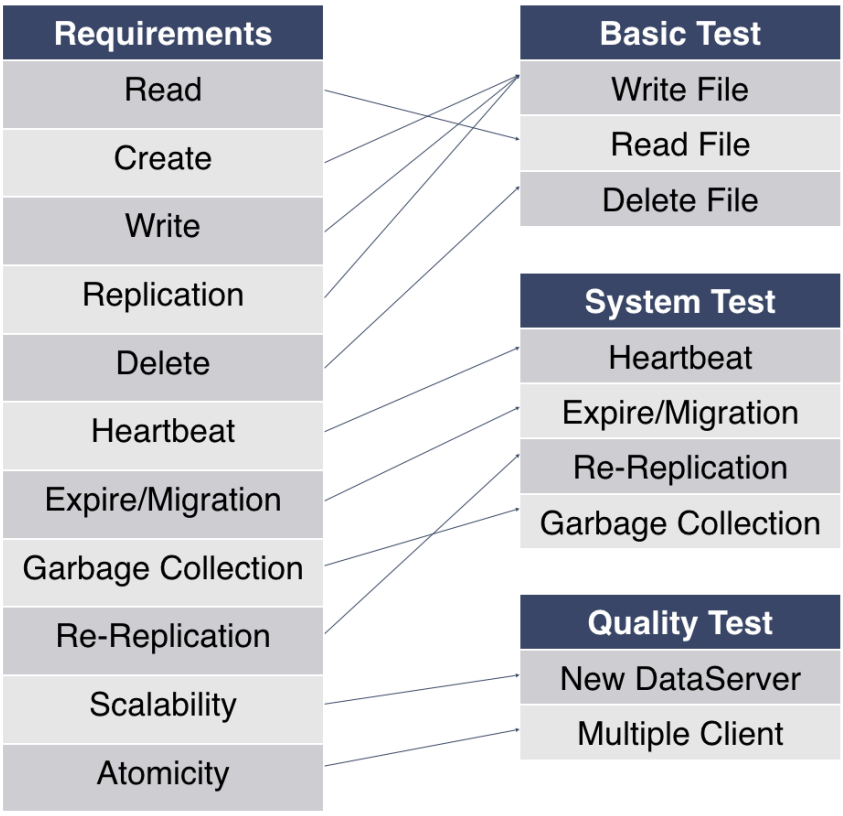
3.6. ReReplication : DataServer에 ReReplication을 지시한다.

추적성 분석표

A. BASIC TEST

- Write File : Test Code를 통하여 Client 요청으로 File(Dummy Data) 쓰기 테스트

- Read File : Client 요청으로 File을 읽어 Dummy Data와 일치하는지 확인
- Delete File : Client 요청으로 File 지우기 테스트

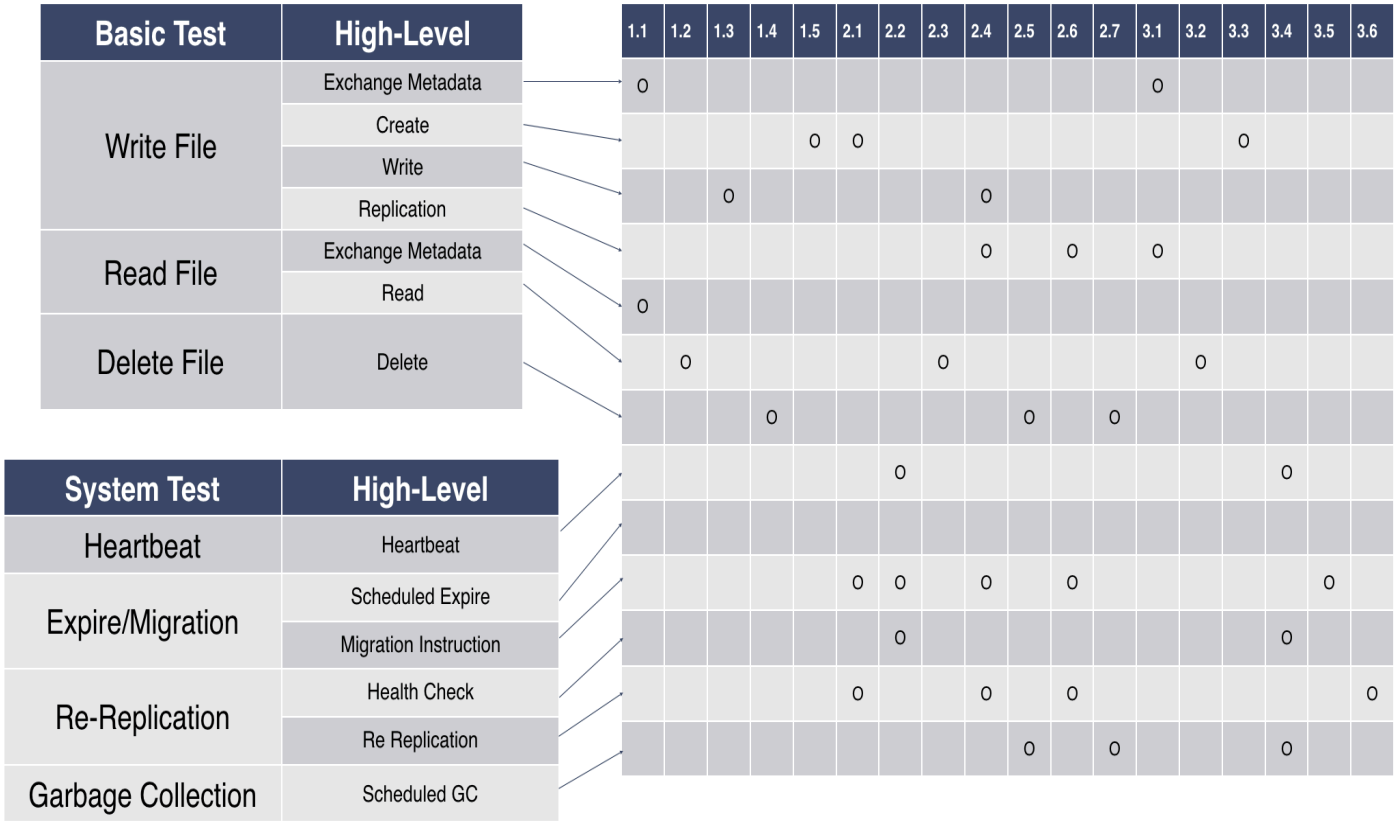


B. SYSTEM TEST

- Heartbeat : Master와 DataServer 간의 데이터 교환
- Expire/Migartion : Master가 Expire된 Chunk에 대해서 DataServer에 Migration을 지시
- Re-Replication : DataServer를 하나 Shutdown한 후 Chunk의 데이터가 복사되는지 확인
- Garbage Collection : config된 tick마다 DataServer내의 Garbage가 지워지는지 확인

C. QUALITY TEST

- New DataServer : 새로운 DataServer 추가하여 정상 작동하는지 테스트
- Multiple Client : 다수의 thread를 통해 요청하여 정상 작동하는지 테스트



Basic Test	High-Level
Write File	Exchange Metadata
	Create
	Write
	Replication
Read File	Exchange Metadata
	Read
Delete File	Delete

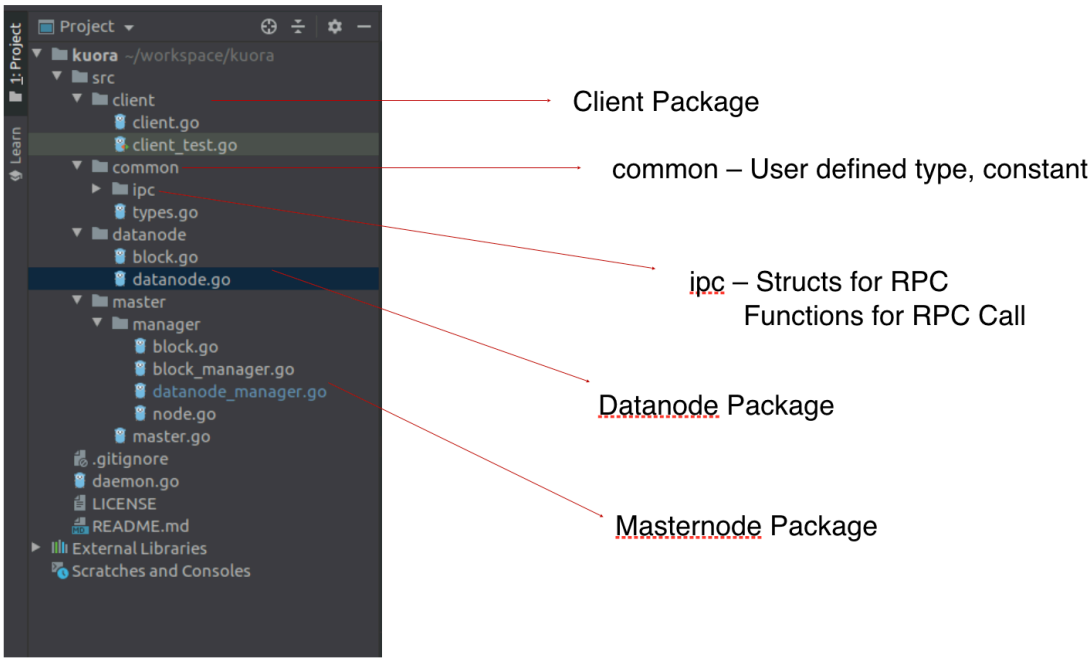
System Test	High-Level
Heartbeat	Heartbeat
Expire/Migration	Scheduled Expire
	Migration Instruction
Re-Replication	Health Check
	Re Replication
Garbage Collection	Scheduled GC

	1.1	1.2	1.3	1.4	1.5	2.1	2.2	2.3	2.4	2.5	2.6	2.7	3.1	3.2	3.3	3.4	3.5	3.6
Exchange Metadata	O												O					
Create					O	O									O			
Write			O						O									
Replication									O		O		O					
Exchange Metadata	O																	
Read		O						O						O				
Delete				O						O		O						
Heartbeat							O									O		
Scheduled Expire						O	O		O		O						O	
Migration Instruction							O									O		
Health Check						O			O		O							O
Re Replication										O		O				O		
Scheduled GC																		

구현 과정

구현과정

A. 1ST ITERATION

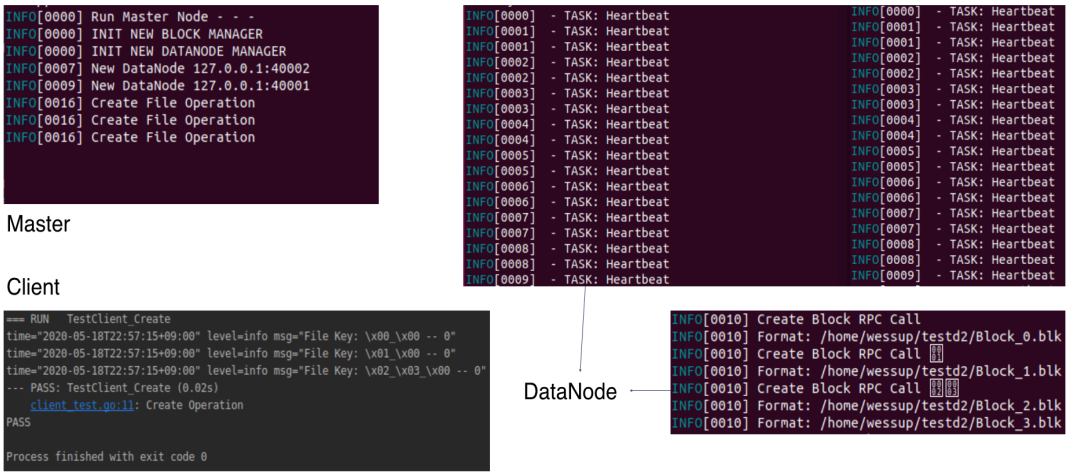


1. 프로젝트 구성

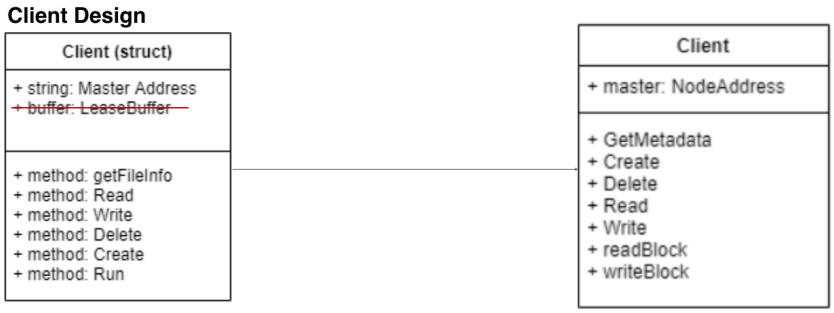
- client : client의 전반적인 기능을 담당한다. golang에서 제공하는 테스트 프레임워크를 통해 client_test는 테스트 코드로 인식되며 이들을 일괄적으로 처리한다.
- common : 사용자 지정 타입이나 공용함수를 정의하였다.
- idc : RPC를 위한 구조체와 RPC Call을 위한 기능을 포함한다.
- datanode : 데이터를 다루기 위한 기능들을 담당한다.
- masternode : master의 기능과 데이터노드 관리를 위한 기능을 담당한다.

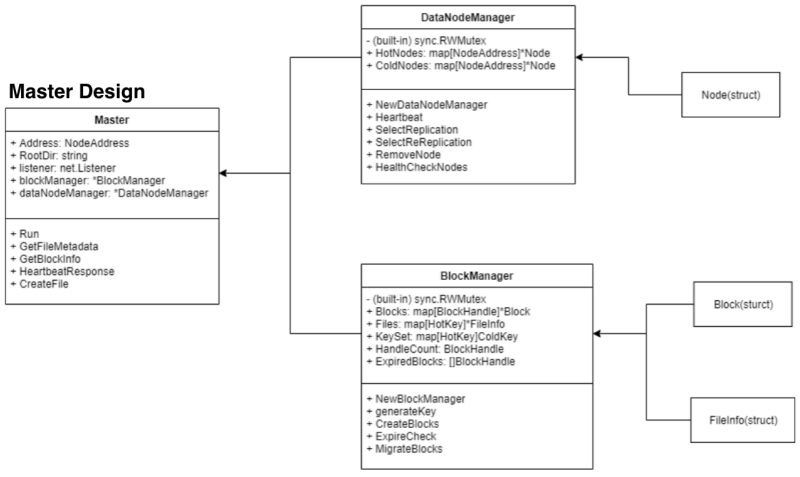
2. Test Logs

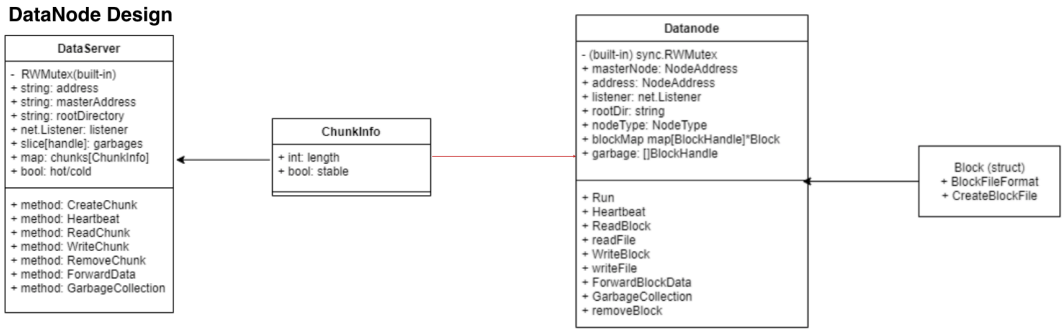
시스템이 잘 작동이 하는지를 가시적으로 확인하기 어렵기 때문에 테스트 코드를 실행하여 logging해 본 결과.



3. Implementations







4. Pass/Fail Criteria

No.	Name	Description
1.1	Read File	Client에서 Key를 통해 Read를 요청하였을 때 해당 데이터를 읽어오는가 여부를 확인
1.2	Write File	Client에서 새로운 파일을 만들고 Write 요청을 할 때 해당 데이터가 DataServer에 올바르게 저장되는가 여부를 확인
1.3	Delete File	Client에서 Key를 통해 Delete 요청을 처리 후 해당 데이터가 Garbage Collection 되는지 확인
2.1	Heartbeat	Master가 지속적으로 DataServer로부터 요청을 받아 DataServer가 정상임을 확인할 수 있는지 여부를 확인
2.2	Expire/Migration	Master의 지시에 따라 DataServer가 Expire에 의한 Delete 혹은 Migration을 수행할 수 있는지 여부를 확인
2.3	Garbage Collection	Configure된 주기에 따라 DataServer 내의 실제 Chunk 데이터가 삭제되는지 여부를 확인
2.4	Replication	Client의 Write 후 DataServer에 Chunk 데이터가 복제가 되었는지 확인
2.5	Re-Replication	DataServer 하나를 중지시킨 후 해당 DataServer가 가진 데이터가 다른 서버에 복제되었는지 확인
Q1	Scalability	Master가 새로운 DataNode를 시스템에 포함시킬 수 있다.
Q2	Multiple Client Test	Client를 다수 실행하여 쓰기 및 읽기를 실행하였을 때 데이터의 동일성 및 Key의 유일성을 확인

구현 완료

미구현

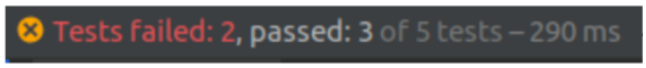
일부 구현 / 진행중

B. 2ST ITERATION

1. Test Flow

- Generate Random Bytes
- Create File Blocks
- Write Bytes to Blocks
- Read Operation

- Compare Original Bytes and Read Bytes



그러나, 사이즈가 큰 파일에 대한 Create/Read/Write 오퍼레이션을 실행하거나 랜덤한 오프셋이나 길이로 파일을 Read 하려 할 때 아래와 같은 오류가 발생한다.

따라서 최종 발표전까지 나머지 test case에 대한 구현을 추진하고 위 Client 측의 버그를 수정하는 것을 목표로 하였다.

2. Project Planning

- Read Big File Operation (Multiple Blocks)
- Re-Replication
- Block Sweep
- Expire/Migration

No.	Name	Description
1.1	Read File	Client에서 Key를 통해 Read를 요청하였을 때 해당 데이터를 읽어오는가 여부를 확인
1.2	Write File	Client에서 새로운 파일을 만들고 Write 요청을 할 때 해당 데이터가 DataServer에 올바르게 저장되는가 여부를 확인
1.3	Delete File	Client에서 Key를 통해 Delete 요청을 처리 후 해당 데이터가 Garbage Collection 되는지 확인
2.1	Heartbeat	Master가 지속적으로 DataServer로부터 요청을 받아 DataServer가 정상임을 확인할 수 있는지 여부를 확인
2.2	Expire/Migration	Master의 지시에 따라 DataServer가 Expire에 의한 Delete 혹은 Migration을 수행할 수 있는지 여부를 확인
2.3	Garbage Collection	Configure된 주기에 따라 DataServer 내의 실제 Chunk 데이터가 삭제되는지 여부를 확인
2.4	Replication	Client의 Write 후 DataServer에 Chunk 데이터가 복제가 되었는지 확인
2.5	Re-Replication	DataServer 하나를 중지시킨 후 해당 DataServer가 가진 데이터가 다른 서버에 복제되었는지 확인
Q1	Scalability	Master가 새로운 DataNode를 시스템에 포함시킬 수 있다.
Q2	Multiple Client Test	Client를 다수 실행하여 쓰기 및 읽기를 실행하였을 때 데이터의 동일성 및 Key의 유일성을 확인

구현 완료

미구현

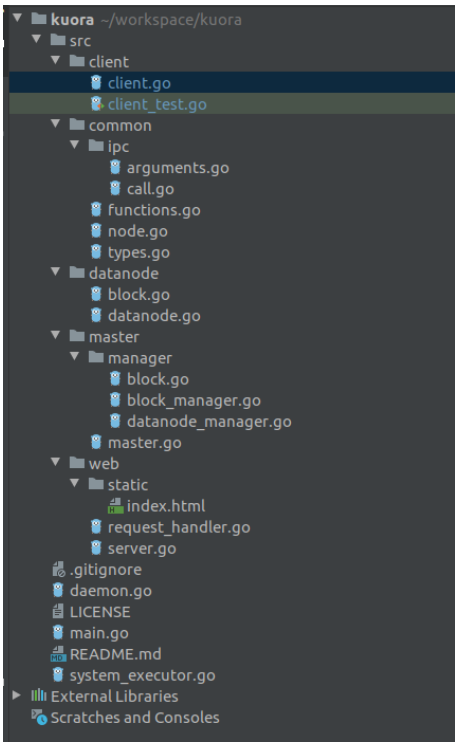
일부 구현 / 진행중

3. Pass/Fail Criteria

최종결과물

- System 동작을 보여주기 위한 시험용 Web View 추가 (web package)
- 기타 구현되지 않았던 기능들 추가 (Re-Replication 등)
- 직접 입력한 바이트, Random으로 생성한 바이트 등으로 구성된 파일들로 읽기, 쓰기, 지우기 테스트

A. 최종 패키지 구성



B. 작업 내용

1. **Client Side Bug Fix**

1. **Read Big File Operation (Multiple Blocks / Random Offset) (Completed)**

2. **System Side Implementation**

    1. **Re-Replication (Completed)**

    2. **Block Sweep (Completed)**

    3. **Expire/Migration (Not Implemented)**

3. **Web View Implementation**

    1. **Write Index.html**

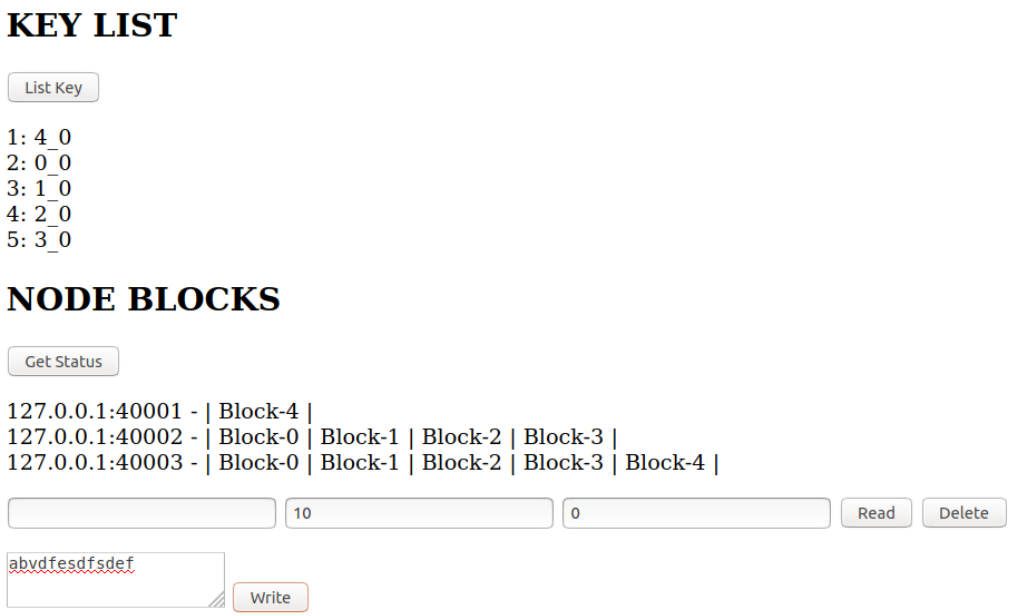
    2. **Write Web Application**

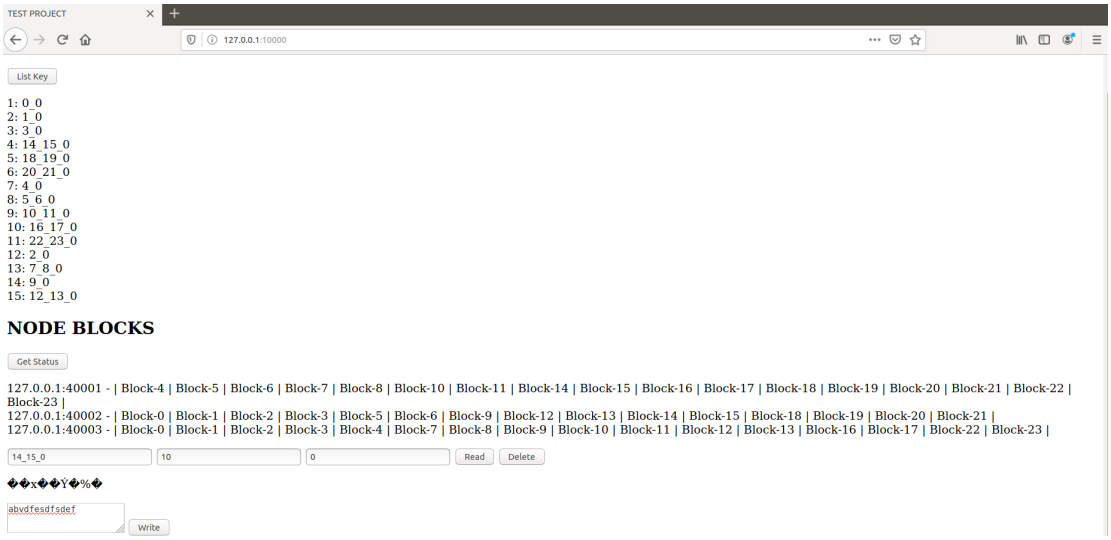
    3. **Write Web Request Handler / Scheduled Job**

C. 최종 구현물 TEST CASE

No.	Name	Description
1.1	Read File	Client에서 Key를 통해 Read를 요청하였을 때 해당 데이터를 읽어오는가 여부를 확인
1.2	Write File	Client에서 새로운 파일을 만들고 Write 요청을 할 때 해당 데이터가 DataServer에 올바르게 저장되는가 여부를 확인
1.3	Delete File	Client에서 Key를 통해 Delete 요청을 처리 후 해당 데이터가 Garbage Collection 되는지 확인
2.1	Heartbeat	Master가 지속적으로 DataServer로부터 요청을 받아 DataServer가 정상임을 확인할 수 있는지 여부를 확인
2.2	Expire/Migration	Master의 지시에 따라 DataServer가 Expire에 의한 Delete 혹은 Migration을 수행할 수 있는지 여부를 확인
2.3	Garbage Collection	Configure된 주기에 따라 DataServer 내의 실제 Chunk 데이터가 삭제되는지 여부를 확인
2.4	Replication	Client의 Write 후 DataServer에 Chunk 데이터가 복제가 되었는지 확인
2.5	Re-Replication	DataServer 하나를 중지시킨 후 해당 DataServer가 가진 데이터가 다른 서버에 복제되었는지 확인
Q1	Scalability	Master가 새로운 DataNode를 시스템에 포함시킬 수 있다.
Q2	Multiple Client Test	Client를 다수 실행하여 쓰기 및 읽기를 실행하였을 때 데이터의 동일성 및 Key의 유일성을 확인

D. WEB VIEW 화면





발전 방향

처음에 계획했던 부분은 Hot Data와 Cold Data를 구분하여 Migration까지 진행하는 것이었으나 시간상 GFS 논문에 명시된 Re-Replication까지 구현을 진행하였다. 따라서 여유가 있었다면 Migration을 통해 데이터의 사용 용도에 따라 데이터를 효과적으로 관리할 수 있도록 구현하는 것이 더 효율적인 방법일 것이다.

현재까지 구현된 모든 기능에 대해서는 동작을 확인하였으며, Web View를 통하여 한정적이지만 기능을 테스트해볼 수 있도록 구현하였다. 동시성 측면에서는 Go에서 기본적으로 제공하는 Test 패키지를 이용하여 30개의 Client Pool을 두고 각 Client가 동시에 Read/Write하는 케이스에 대해서 테스트를 진행하였다.

이번에 구현한 System 내에서 발전 방향을 크게 세 가지로 분류해볼 수 있을 것이다.

1. 구현되지 못한 기능의 구현
   - 앞서 서술하였듯 처음에 구성한 것과 다르게 Data간의 Migration을 구현하지 못하였으므로 이에 대한 구현이 필요할 것이다.
   - 이를 위해서는 Data에 대한 Expire 기간의 부여 및 이를 체크하여 배치 형식으로 Migration을 수행하는 형식이 적절하다고 여겨진다.
2. 효율적인 데이터 전송 및 처리
   - 이번 졸업작품에서 시간 관계 상 빠르게 구현하기 위하여 데이터 전송의 효율성 측면에서 미흡한 부분이 있었다고 생각한다.
   - GFS 에서 명시된 Client 측 Lease (일종의 Client 측 캐시)에 대한 구현과 DataNode 간의 데이터 전송이 이루어질 때 이를 Caching하는 Buffer가 존재한다면 더 효율적인 구성이 이루어질 수 있을 것이라고 생각한다.
   - 또한 함수 호출 시에 현재는 각 Block Handle에 대하여 반복적으로 호출하는 형식이지만 이를 일괄적으로 처리할 수 있는 프로세스를 구성한다면 더 효율적인 처리가 이루어질 것이다.
3. 데이터 수정에 대한 자율성 추가
   - 처음 프로젝트를 구성할 때 데이터에 대한 수정은 없다고 가정하였다.
   - 만약 데이터에 대한 자율성을 좀 더 추가한다면 더 범용성을 가진 System을 구성할 수 있을 것이다.
   - GFS 같은 경우는 AppendOnly를 통하여 Chunk의 1/4 사이즈에 해당하는 데이터의 추가가 가능한 것과 같이 데이터의 Write에서 더 자율성을 부여할 수 있을 것이다.

목표 대비 달성률

앞서 설명하였듯 Data의 Migration과 관련한 부분은 구현을 하지 못하였다.

그러므로 처음 Success Criteria에 언급된 Migration에 대한 부분에 대해서는 충족을 하지 못하였고, 1번에 해당하는 Read, Write, Delete 그리고 System간 Heartbeat와 Replication, Re-Replication이나 Garbage Collection 등은 원활하게 이루어지는 것을 확인하였다.

이상의 결과에 따라서 목표 대비 달성률은 절반 정도로 측정할 수 있을 것이며, 완료된 테스트에 대해서는 앞서 언급되었으므로 생략하겠다.

참고문헌

- Google File System

https://static.googleusercontent.com/media/research.google.com/ko//archive/gfs-sosp2003.pdf

- OwFS: A Distributed File System for Large-scale Internet Services (NHN)

http://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE01194876

- TidyFS: A Simple and Small Distributed File System

https://www.microsoft.com/en-us/research/publication/tidyfs-a-simple-and-small-distributed-file-system/

- Needle in a haystack: efficient storage of billions of photos

https://engineering.fb.com/core-data/needle-in-a-haystack-efficient-storage-of-billions-of-photos/