Patricia Marukot 1002157499
Jing Yi Li 1002346730
Seongju Yun 1002275290

M2 Design Documentation

Message formats:

The structure of Message class (src/shared/messages/Message.java) for communication between KVClient and KVServer:

```
Public class Message {
	Private String key;
	Private String value;
	Private StatusType status;
	Private Metadata metadata;
}
```

KVAdminStatusType

```
Public enum KVAdminStatusType {
	SUCCESS,
	ERROR
}
```

KVServer and ECS

```
Public class KVAdminMessage extends Metadata {
	Private KVAdminStatusType status;
	Private int cacheSize;
	Private String replacementStrategy;
}
```

Metadata:

```
Public class Metadata {
	Protected String name; // name in ecs.config
	Protected String host; // ip
	Protected ServerStatusType serverStatusType;

	/* Inclusive */
	Protected Pair<long,long> hashRange;

	Protected HashRing hashRing;
}
```

ECSClient

Listed below is the pseudo code used in implementing the ECSClient application

/* This is the constructor for ECSClient which is called at startup. The main functionality is to parse the ECS config file and start up the specified servers using the bash script script run_kvserver.sh. The server statuses are initialized to IDLE indicating that they are not accepting client requests */

ECSClient(ecs_config):

Create empty

For each KVServer in ecs_config:

Deserialize KVServer

Set name, IP, port #

Set KVServer state to IDLE

Put into Server-Repo

/* Below is a list of all of the commands supported by ECSClient (these commands directly correspond to the API functions specified in the M2 document.

ECSClient_handleCmd(cmd):

If cmd is “ADD_NODES <num_nodes> <cache_size> <replacement_strategy>”:

From Server-Repo, choose num_nodes KVServers which are IDLE/SHUTDOWN

For each KVServer selected:67

Change KVServer state: IDLE/SHUTDOWN→ STOPPED

Start KVServer process with IP, port: exec bash

Send INIT_KVSERVER request to KVServer, args=[Metadata, cache_size, replacement_strategy]

If cmd is “ADD_NODE <cache_size> <replacement_strategy>”:

Choose random KVServer from Server-Repo with state {IDLE || SHUTDOWN}

KVServer.state = {IDLE || SHUTDOWN} → STOPPED

Hash = MD5({KVServer_ip, KVServer_key})

Put KVServer on the Hash Ring based on Hash

exec(bash, KVServer_ip, KVServer_port) // start KVServer

Send INIT_KVSERVER request to KVServer, args=[Metadata, cache_size, replacement_strategy]

If cmd is “START”:

For each KVServer in Server-Repo with state STOPPED:

KVServer.state = STOPPED → STARTED

KVServer.start()

If cmd is “STOP”:

From Server-Repo, choose KVServers with state STARTED

For each KVServer selected:

Change KVServer state: STARTED → STOPPED

Send UPDATE request to KVServer, args=[Metadata]

If cmd is “SHUTDOWN”:

From Server-Repo, choose KVServers with state STARTED/STOPPED

For each KVServer selected:

Send SHUTDOWN request to KVServer

If cmd is “QUIT”:

Call shutdown() to shutdown all servers

Exit ECSClient application;

KVServer:

/* Approach to handle client requests - note only the new scenarios introduced in M2 are displayed, the PUT/GET request logic remains the same */

KVServer_recvKVClientRequest(<k,v>):

If server state is SERVER_STOP:

Reply SERVER_STOP to requesting client

If server state is SERVER_WRITE_LOCK:

Reply SERVER_WRITE_LOCK

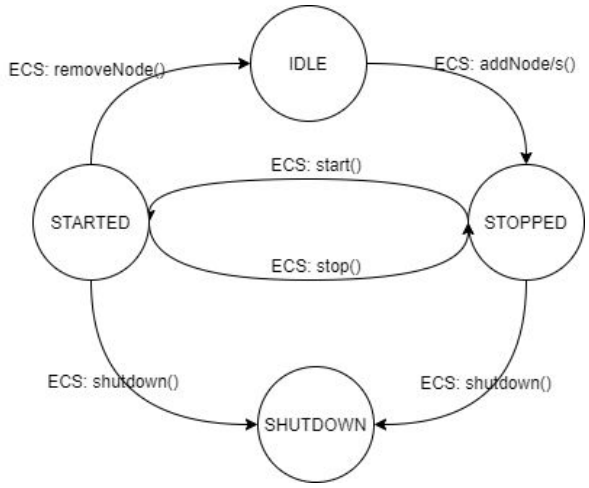
Hash = compute hash from k

If Hash is not in range of this server:

Reply NOT_RESPONSIBLE + complete metadata to requesting client

Else:

Server request (same as m1) and reply to requesting client



KVServer: state transition diagram

Client library (KVStore) & application (KVClient)

Initial State:

Client has no metadata.

KVClient_doRequest(<k,v>):

If client doesn't have metadata:

Send <k,v> to default KVServer

Else :

Hash = compute has from k

Determine which server to send based on where hash falls in ring

Send <k,v> to corresponding KVServer

Wait for server response

If response type is NOT_RESPONSIBLE:

Update client metadata

doRequest(message) // keep trying until succeeds

If response type is SERVER_WRITE_LOCK:

Throw end-user error

If response type is SERVER_STOP:

Throw end-user error

If no response after TIMEOUT:

Throw end-user error

// response is successful

Display response to user (just like m1)

Test Cases

Interaction	Expected behavior	PASS/FAIL
Client GET/PUT request - no servers in START state	1. Client connection to server is successful 2. PUT/GET request fails because server was not set to START state	
Client GET/PUT request	If server has data: return data If server doesn't have data: Server checks HashRing to see if this key is "owned" by another server If key hashes to another server: Reply NOT_RESPONSIBLE + updated HashRing If key hashes to this server (but no data in this server): Reply GET_ERROR	
ECS: adds server to HashRing	Add NEW_NODE to ring Recalculate range in HashRing Init NEW_NODE Set write lock on successor node Transfer subset of successor to NEW_NODE Broadcast metadata update (hashRing)	
Client *connect* - all servers SHUTDOWN (after call to ECSClient>shutdown)	KVClient connection fails for all servers specified in ecs.config file	
ECSClient > stop request - client issues GET/PUT requests	GET/PUT requests should fail	

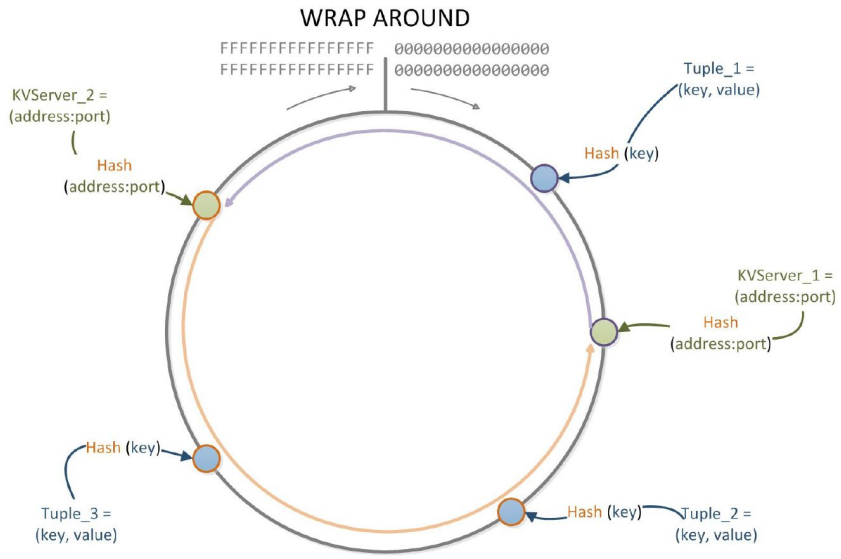
Server Name	Host:Port	MD5 Hash
server-0	localhost:50000	2b786438d2c6425dc30de0077ea6494d
server-1	localhost:50001	0221f85727f09bb279fa843d25c48052
server-2	localhost:50002	05eaa8ab2a10954744c21574cd83e7f7

Key	Value	MD5 hash	Server
jimmy	KOBE	c2fe677a63ffd5b7ffd8facbf327dad0	1
patricia	BRYANT	823fec7a2632ea7b498c1d0d11c11377	1
ann	IS	7e0d7f8a5d96c24ffcc840f31bce72b2	1
amy	THE	7771fbb20af6ef10827c593daa3aff7b	1
richard	GREATEST	6ae199a93c381bf6d5de27491139d3f9	1
lemon	PLAYER	3f24e567591e9cbab2a7d2f1f748a1d4	1
apple	IN	1f3870be274f6c49b3e31a0c6728957f	0
peach	THE	889560d93572d538078ce1578567b91a	1
iphone	NBA	0b3f45b266a97d7029dde7c2ba372093	0

Consistent Hashing - Hash Ring requirements

1) All storage servers (KVServers) arranged CW (increasing) in ring, position determined by value of hash
2) Use 128-bit hash: MD5 (java.security.MessageDigest)

Server_Hash = **MD5**("${server_ip}:${server_port}")
Data_Hash = **MD5**(Key)



Client library (KVStore) & application (KVClient)

Initial State:
Client has no metadata.

KVClient_doRequest(<k,v>):

If client doesn't have metadata:
Send <k,v> to default KVServer
Else :
Hash = compute has from k
Determine which server to send based on where hash falls in ring
Send <k,v> to corresponding KVServer

Wait for server response
If response type is NOT_RESPONSIBLE:
Update client metadata
doRequest(message) // keep trying until succeeds
If response type is SERVER_WRITE_LOCK:
Throw end-user error
If response type is SERVER_STOP:
Throw end-user error
If no response after TIMEOUT:
Throw end-user error

// response is successful

Display response to user (just like m1)

KVServer:

KVServer_recvKVClientRequest(<k,v>):

If server state is SERVER_STOP:

Reply SERVER_STOP to requesting client

If server state is SERVER_WRITE_LOCK:

Reply SERVER_WRITE_LOCK

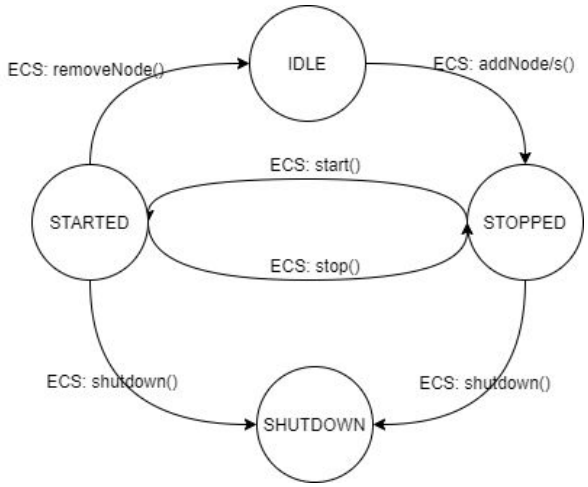
Hash = compute hash from k

If Hash is not in range of this server:

Reply NOT_RESPONSIBLE + complete metadata to requesting client

Else:

Server request (same as m1) and reply to requesting client



KVServer: state transition diagram

ECSClient

ECSClient(ecs_config):

Create empty

For each KVServer in ecs_config:

Deserialize KVServer

Set name, IP, port #

Set KVServer state to IDLE

Put into Server-Repo

ECSClient_handleCmd(cmd):

If cmd is “ADD_NODES <num_nodes> <cache_size> <replacement_strategy>”:

From Server-Repo, choose num_nodes KVServers which are IDLE/SHUTDOWN

For each KVServer selected:67
Change KVServer state: IDLE/SHUTDOWN→ STOPPED
Start KVServer process with IP, port: exec bash
Send INIT_KVSERVER request to KVServer, args=[Metadata, cache_size, replacement_strategy]

If cmd is “ADD_NODE <cache_size> <replacement_strategy>”:
Choose random KVServer from Server-Repo with state {IDLE || SHUTDOWN}
KVServer.state = {IDLE || SHUTDOWN} → STOPPED
Hash = MD5({KVServer_ip, KVServer_key})
Put KVServer on the Hash Ring based on Hash
exec(bash, KVServer_ip, KVServer_port) // start KVServer
Send INIT_KVSERVER request to KVServer, args=[Metadata, cache_size, replacement_strategy]

If cmd is “START”:
For each KVServer in Server-Repo with state STOPPED:
KVServer.state = STOPPED → STARTED
KVServer.start()

If cmd is “STOP”:
From Server-Repo, choose KVServers with state STARTED
For each KVServer selected:
Change KVServer state: STARTED → STOPPED
Send UPDATE request to KVServer, args=[Metadata]

If cmd is “SHUTDOWN”:
From Server-Repo, choose KVServers with state STARTED/STOPPED
For each KVServer selected:
Send SHUTDOWN request to KVServer

Message formats:

The structure of Message class (src/shared/messages/Message.java) for communication between KVClient and KVServer:

```
Public class Message {
      Private String key;
      Private String value;
      Private StatusType status;
      Private Metadata metadata;
}
```

KVAdminStatusType

```
Public enum KVAdminStatusType {
      ACK,
      ERROR
}
```

KVServer and ECS

```
Public class KVAdminMessage extends Metadata {
      Private KVAdminStatusType status;
      Private int cacheSize;
      Private String replacementStrategy;
}
```

Metadata:

```
Public class Metadata {
      Protected String name; // name in ecs.config
      Protected String host; // ip
      Protected ServerStatusType serverStatusType;

      /* Inclusive */
      Protected Pair<long,long> hashRange;

      Protected HashRing hashRing;
}
```

Tasks:

1) ~~KVClient: update request handling~~
2) KVServer:
    a) initKVServer
    b) Start, Stop, update, shutDown
    c) lockWrite, unlockWrite, isWriterLocked
    d) moveData
    e) getServerState, getMetadata
3) ~~Update message format for KVClient-KVServer, KVServer-ECSClient (see above)~~
4) ECS:
    a) Data structures: Metadata, ECSNode, HashRing
    b) CLI
    c) addNodes, addNode, removeNode
    d) Start, Stop, shutDown, getServer
5) Zookeeper

Patricia
Ann
Jimmy

Unified data structure (ECS + Client + Server) - request/response

```
UnifiedRequestResponse {
        StatusType status,              // status of every request/response
        KVServerMetadata metadata,      // hashring, etc.
        KVDataSet dataset,              // for data transfer between servers
        String key,                     // client request (PUT/GET request)
        String value,                   // client request (PUT/GET request)
}
```

Test cases: