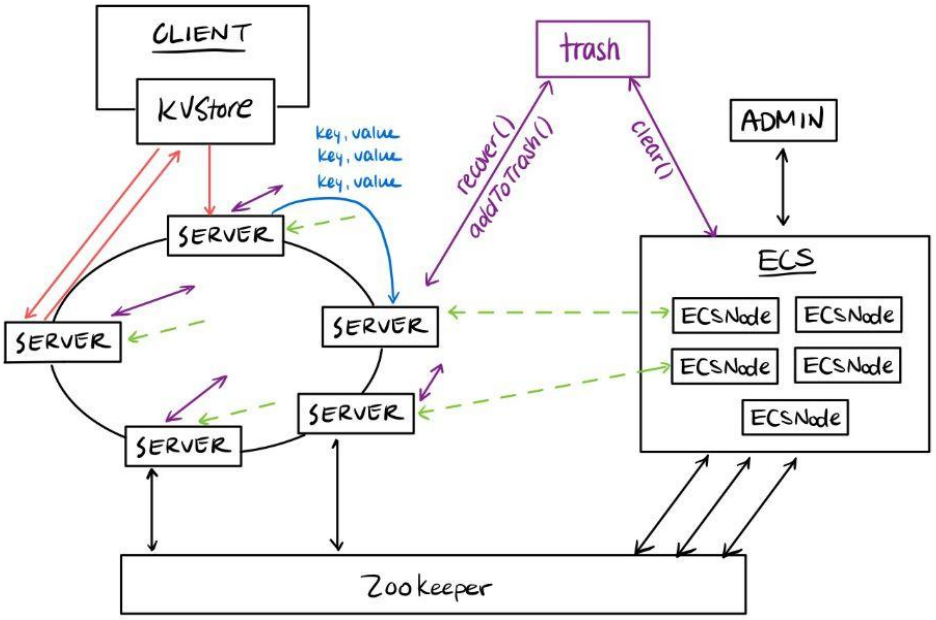
ECE419 Milestone 4 Design Document

Team 28: Akino Watanabe (1004133653), Lisa Li (1003924532), Sandra Petkovic (1004018537). Submission by: *petkov12*

The Overall Design



Design Decisions

Soft Delete Overview

For Milestone 4, our team decided to implement a soft delete feature: users have the option to recover keys after deleting them as long as they do not expire in the trash. For testing and demo purposes, our keys are permanently deleted and no longer recoverable after one minute of being deleted. However in industry, this value would be closer to, say, 7 years.

Every 30 seconds, an asynchronous cleanup job clears the trash of any expired keys. Once they have been cleared, the key-value pairs cannot be recovered.

Persistent Storage & Recovering Keys

In the `put()` function, on the deletion of a key, the key-value pair is moved to a global `trash.txt` file, with an addition of a `timestamp` key, currently set for 1 minute after we locate the key in the server's storage. Using the format of our `JSONMessage`, an example of how the key value pair appears in `trash.txt` is shown below:

```
{"status":"NO_STATUS","key": "key1", "value": "value1",
        "timestamp":"2022-04-14T15:08:26.998"}
```

Should the client `DELETE` a key, `PUT` it again, then `DELETE` it again, all before the initial key is permanently deleted, the timestamp of the key in the trash will be updated with the time relative to the most recent instance it was deleted.

Should the client `DELETE` a key, `PUT` the same key again with a different value, then try to recover the initial key, the value from the trash will overwrite the value in the server's storage. This is to preserve the uniqueness of keys and ensure that the desired value is always current.

To recover a key on the `RECOVER` command, PersistentStorage first checks if the key falls in its hash ring range. If not, it returns a `SERVER_NOT_RESPONSIBLE` status and its updated metadata to KVStore. If it does, it calls a `recover()` function in PersistentStorage, which is nearly identical to our `put()` and `get()` functions. We read `trash.txt` line by line, and as each line is read, they're deserialized back into a `JSONMessage` object, where we are able to easily extract the `key`. We then compare the key from the line to the key we are trying to recover. If the key exists in `trash.txt`, we add that line back to the server's storage file, remove it from `trash.txt`, and return a `RECOVER_SUCCESS` status. If the key does not exist in `trash.txt`, we return a `RECOVER_ERROR` status.

On a `RECOVER_SUCCESS`, the key and value are immediately put into the cache.

ECSClient

The ECSClient is in charge of running the asynchronous trash cleanout. The ECSClient was chosen over the KVServer to do so because the task only needs one machine executing it in any given run. Running it on ECSClient ensures this.

The cleanup job is initialized with the ECSClient and is scheduled to run every 30 seconds after an initial 30 second delay. The ECSClient uses the function `clearTrash()` from PersistantStorage to perform this task. `clearTrash()` also reads `trash.txt` line by line, deserializes each line back into a `JSONMessage` object. The `timestamp` value, which is the time the key-value pair should expire, is converted into a `LocalDateTime` Java object which is compared with the current time (also a `LocalDateTime` object). If the current time is past the timestamp, we remove the line from `trash.txt`, meaning it cannot be recovered by the client anymore.

KVClient

In addition to command `put <key> <value>` and `get <key>`, the client can now run `recover <key>`. The KVStore then sends a `JSONMessage` with a `RECOVER` status to the server. It expects a status of `RECOVER_SUCCESS`, `RECOVER_ERROR`, or `SERVER_NOT_RESPONSIBLE`, and acts accordingly, either printing the status to the client's console or switching servers based on the updated metadata, respectively.

Performance test

Two sets of performance tests were run: the first set computes the time duration it takes to add and remove a certain number of ECS nodes; the second set computes the latency and throughput of processing a certain number of requests (PUT, GET, and RECOVER are equally split) when having different number of servers and clients and different cache strategy and size. The detailed numbers for the performance tests are in Appendix 2 below.

The time duration of adding and removing nodes increased as the number of nodes increased, and overall, all the time duration for adding the nodes was similar or smaller than that of

Milestone 2, while the time duration for removing the nodes was similar or larger than that of Milestone 2. This is expected because this milestone incorporates creating and setting up the trash system when adding the nodes.

As the number of servers increased, the latency and throughput fluctuated, and as the number of clients increased, both latency and throughput decreased. This may be because the clients could simultaneously process multiple requests at once.

Regarding caching, FIFO and LFU's latencies and throughputs decreased for milestone 3, and the different cache size did not have much effect on the performance.

Overall, compared to milestone 2, the latency was similar or decreased and throughput decreased for milestone 4, and hence for our system, adding the recovery strategy improved the overall performance of the distributed system. Additionally, compared to milestone 3, the overall performance for the time duration, latency, and throughput were similar or better for milestone 4.

Attribution

Name	**Contributions to M4**
Akino Watanabe	KVStore, Performance test, debugging, unit tests
Lisa Li	Moving keys to trash, recovering keys from trash
Sandra Petkovic	Clear trash, asynchronous clearing

Appendix

Appendix 1: Unit tests

Test Function	**Description**
testRecover	Confirms that the key-value pair can be recovered after the user calls delete if the user calls recover within a minute
testCannotRecover	Confirms that if the user did not delete the key-value pair but the user calls recover, RECOVER_ERROR is received
testMultipleKeys	Confirms that multiple key-value pairs can be recovered after the user calls multiple deletes if the user calls those recover requests within a minute
testRecoverNonExistentKey	Confirms that RECOVER_ERROR is received when the user tries to recover something that does not exist to begin with
testRecoverOverTimeDuration	Confirms that RECOVER_ERROR is received if the user tries to call recover request after 90 seconds of deleting

	the key-value pair because the trash bin is already cleared
testRecoverOverTimeDurationMultiple	Confirms that multiple RECOVER_ERROR is received if the user tries to call multiple recover requests after 90 seconds of deleting the key-value pairs because the trash bin is already cleared
testRecoverOverwriteWithUpdte	Confirms that RECOVER_SUCCESS is received after the key-value pair is deleted and the user calls update request for the key that is deleted with a new value and then tries to recover the original key-value pair; in this case, the recover request will overwrite the existing key-value pair

Appendix 2: Performance test results

Number of nodes (no caching)	**Time duration of adding nodes (ms) (Milestone 2 values)**	**Time duration of removing nodes (ms) (Milestone 2 values)**
1	1147 (1126)	2 (2)
3	3400 (3245)	15 (16)
5	5605 (9332)	33 (19)
7	7882 (9705)	73 (58)
10	11179 (15687)	150 (87)

Number of servers	**Number of clients**	**Caching strategy**	**Cache size**	**Latency (ms) (Milestone 2 values)**	**Throughput (B/ms) (Milestone 2 values)**
1	2 (8 requests)	None	0	6258375 (6260000)	1.19839415183 6539E-3 (1.5974E-3)
2	2 (8 requests)	None	0	17501250 (6261750)	4.28540818512 96334E-4 (1.5969E-3)
3	2 (8 requests)	None	0	13761000 (13766125)	5.45018530630 0414E-4 (1.5965E-3)
4	2 (8 requests)	None	0	6261250 (13766125)	1.19784388101 41745E-3 (6.3562E-4)
2	4 (8 requests)	None	0	12509250 (27510125)	2.99778164158 5227E-4 (9.0875E-5)
2	1 (8 requests)	FIFO	8	8751375 (10637125)	5.71338789618 7742E-4 (9.4010E-4)

2	1 (8 requests)	LFU	8	8750875 (10648750)	5.71371434285 1429E-4 (9.3907E-4)
2	1 (8 requests)	LRU	8	8750875 (8750750)	4.28528575713 8572E-4 (8.5706E-4)
2	1 (8 requests)	FIFO	3	8751000 (10638750)	5.71363272768 8264E-4 (9.3996E-4)