University of St.Gallen

Institute of Computer Science

Distributed Systems, Autumn Semester 2023
Jérémy Lemée, Sanjiv Jha, Andrei Ciortea
{jeremy.lemee, sanjiv.jha, andrei.ciortea}@unisg.ch

Assignment 5: Fault-Tolerant Key/Value Service (7pt)

Deadline: Dec. 12, 2023; 23:59 CET

In this assignment, you will get practical experience with distributed consensus and Raft by implementing a *fault-tolerant key/value service*, which is a service a client can use to retrieve and store values using a unique key. This assignment is structured as follows:

- In the first task, you will explore how the Raft algorithm works using an interactive visualization.
- In the second task, you will implement the fault-tolerant key/value service in Python.
- In the third task, you will simulate failures to asses the fault tolerance of your implementation.
- In the fourth task, a few conceptual questions will guide you to dive deeper into Raft.

The project uses the Python *PySyncObj* package[1], which implements the Raft algorithm.

(1) Discovering Raft (1pt)

The website https://raft.github.io/ includes an interactive visualization of how the Raft algorithm works in a group of five nodes. In this task, you will use this interactive visualization to perform several exercises.

Hint: Before you start working on the exercises, we also recommend going through the following visual illustration of how Raft works: https://thesecretlivesofdata.com/raft/.

1.) (0.2pt) What happens when Raft starts? Explain the process of electing a leader in the first term.

2.) (0.4pt) Once the leader has been elected, send a request to the leader and wait until the operation is committed by all the nodes (*note:* in the interactive visualization, a committed operation is represented with a solid border in replicated logs). Then, perform the following steps:

- Pause the simulation, perform a new request on the leader, and take a screenshot (task1_q2_1.PNG). Stop the leader and then resume the simulation for a new election.
- Once, there is a new leader, perform a new request, and then resume the leader from the previous term. Once the new request is committed by all servers, pause the simulation, and then take another screenshot (task1_q2_2.PNG).

How did the logs of the nodes evolve during the above exercise? Explain what happened.

3.) (0.4pt) Using the same visualization, stop the current leader and two additional servers. Only two nodes should now be running in your simulation. Then, perform the following steps:

- After a few term increments, pause the simulation to take a screenshot (task1_q3_1.PNG).

[1] https://pysyncobj.readthedocs.io/en/latest/

- Resume all nodes and resume the simulation. After the new leader election, pause the simulation and take a screenshot (task1_q3_2.PNG).

How do you explain the behavior of the system during the above exercise?

(2) Implementing a Fault-Tolerant Key/Value Service with Raft (2.5pts)

In this task, you will implement a key/value service that supports three types of operations:

- `Get(Key)` returns the value associated with the key. The value is a list of strings.
- `Put(Key, Value)` sets the value (a list) for the given key.
- `Append(Key, Value)` adds the value (a string) to the list of values for the given key. This operation acts like `Put(Key, [Value])` if the current value for the key is empty or undefined.

To enable fault tolerance, the service can use Raft to replicate the key/value store across a group of servers. We provide you with a Python project template based on PySyncObj[2], an open-source implementation of Raft in Python. To implement this task, you will then have to update the following files (see also project README):

1.) `kvstorage.py`, which implements an instance of the key/value service itself, and

2.) `kvstorage_http.py`, which implements an HTTP server that is used to perform queries on the key/value service.

Initialize three Raft servers using your key/value service implementation by using the provided initialization scripts (see README). Then, perform HTTP queries to all the three servers.

For example, to initialize the "a" key with a `Put` request, one could do the request:

POST http://localhost:8080/keys/a HTTP/1.1
Content-Type: application/json

{"type": "PUT",
"value": ["cat", "dog"]
}

Then, one can perform an `Append` request with:

POST http://localhost:8080/keys/a HTTP/1.1
Content-Type: application/json

{"type": "APPEND",
"value": ["mouse"]
}

The final value can then be read with a `Get` request:

GET http://localhost:8080/keys/a HTTP/1.1

The status for node 0 can be read with:

GET http://localhost:8080/admin/status HTTP/1.1

To test your implementation:

- Perform a `Put` operation on server 0 to test your implementation.

[2] https://pysyncobj.readthedocs.io/en/latest/

- You can use the `Get` operation on server 1 with the same key to verify that the put operation was replicated and that server 1 reflects the changes.

In your `Report.md` file, indicate the replies you get from the `/admin/status` HTTP endpoint of each server. Then, answer the following questions:

1.) Which server is the leader? Can there be multiple leaders?

2.) Perform a `Put` request for the key "a" on the leader. What is the new status? What changes occurred and why (if any)?

3.) Perform an `Append` request for the key "a" on the leader. What is the new status? What changes occurred and why (if any)?

4.) Perform a `Get` request for the key "a" on the leader. What is the new status? What change (if any) happened and why?

(3) Evaluation of the Key/Value Service (2pts)

In this task, you will simulate several failures to test the behavior of your implementation from **Task 2**:

1.) Shut down the server that acts as a leader. Report the status that you get from the servers that remain active after shutting down the leader.

2.) Perform a `Put` request for the key "a". Then, restart the server from the previous point, and indicate the new status for the three servers. Indicate the result of a `Get` request for the key "a" to the previous leader.

3.) Has the `Put` request been replicated? Indicate which steps lead to a new election and which ones do not. Justify your answer using the statuses returned by the servers.

4.) Shut down two servers, including the leader — starting with the server that is not the leader. Report the status of the remaining servers and explain what happened.

5.) Can you perform `Get`, `Put`, or `Append` requests in this system state? Justify your answer.

6.) Restart the servers and note down the new status. Describe what happened.

(4) Questions on Raft (1.5pt)

1.) (0.5pt) What is a consensus algorithm? What are they used for in the context of replicated state machines?

2.) (0.5pt) What are the main features of the Raft algorithm? How does Raft enable fault tolerance?

3.) (0.5pt) What are Byzantine failures? Can Raft handle them?

Hand-in Instructions By the deadline, you should hand in a single **zip** file via Canvas upload. The name of this file should start with `a5` and contain the last names of all team members separated by underscores (e.g., `a5_lemee_jha_ciortea.zip`). It should contain the following files:

- All answers to the assignment questions in the given `REPORT.md`; if you wish to submit your solution code via GitHub, please include a link to your GitHub repository as well

- Your code as an archive if you do not provide a GitHub link in your Report.
- The different screenshots that you have to take. (i.e., task1_q2_1.PNG, task1_q2_2.PNG, task1_q3_1.PNG, and task1_q3_2.PNG)

Across all tasks in this and the other assignments in this course, you are **required to declare** any support that you received from others and, within reasonable bounds,[3] any support tools that you were using while solving the assignment.

[3]It is not required that you declare that you were using a text-editing software with orthographic correction; it is however required to declare if you were using any non-standard tools such as generative machine learning models (e.g., GPT).