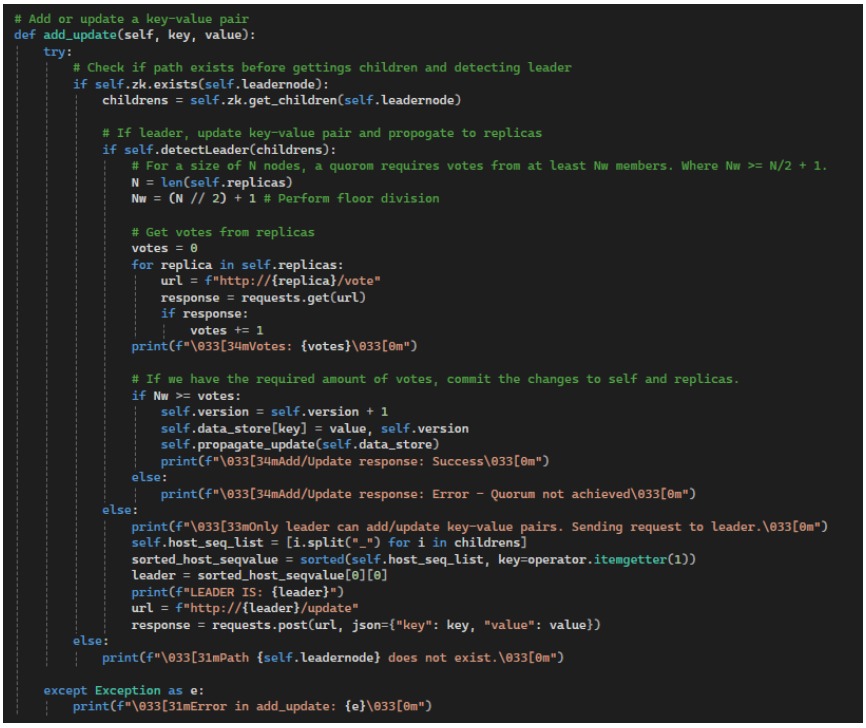
Wagonblast, Gregory
CS2510

Assignment 3: Zookeeper-Quorom

Project Description: This project is based on the project code for assignment 2. The same base logic still applies. That being, a replicated key-value data store maintained by N servers was implemented and each server maintains a copy of the data store. Zookeeper was used in order to implement leader election. All requests are routed through the leader and propagated to the replicas. That said, an add_update(key, value) method only commits changes when a quorum is achieved. Meaning, votes are needed for at least Nw members, where Nw >= N/2 + 1. Similarly, the read(key) method only sends the value to the client when a quorum is achieved. Meaning, Nr replicas are selected to share their values, where Nr is a random sample of Nw nodes (Nw = (N // 2) + 1 in this case), and Nr + Nw > N. If there is no disagreement between values and versions, the value is sent to the client. Otherwise, this process is repeated until a timeout of 30 seconds is reached.

Project Implementation:

Zookeeper_Quorom.py add_update(key, value) method. This method was modified to commit updates when a quorum is achieved. This is done by requesting votes from the replicas through HTTP requests. If Nw >= votes, the data is updated. This update includes a version number to track state consistency between the leader and replicas.



```python
# Add or update a key-value pair
def add_update(self, key, value):
    try:
        # Check if path exists before gettings children and detecting leader
        if self.zk.exists(self.leadernode):
            childrens = self.zk.get_children(self.leadernode)

            # If leader, update key-value pair and propogate to replicas
            if self.detectLeader(childrens):
                # For a size of N nodes, a quorom requires votes from at least Nw members. Where Nw >= N/2 + 1.
                N = len(self.replicas)
                Nw = (N // 2) + 1 # Perform floor division

                # Get votes from replicas
                votes = 0
                for replica in self.replicas:
                    url = f"http://{replica}/vote"
                    response = requests.get(url)
                    if response:
                        votes += 1
                print(f"\033[34mVotes: {votes}\033[0m")

                # If we have the required amount of votes, commit the changes to self and replicas.
                if Nw >= votes:
                    self.version = self.version + 1
                    self.data_store[key] = value, self.version
                    self.propagate_update(self.data_store)
                    print(f"\033[34mAdd/Update response: Success\033[0m")
                else:
                    print(f"\033[34mAdd/Update response: Error - Quorum not achieved\033[0m")
            else:
                print(f"\033[33mOnly leader can add/update key-value pairs. Sending request to leader.\033[0m")
                self.host_seq_list = [i.split("_") for i in childrens]
                sorted_host_seqvalue = sorted(self.host_seq_list, key=operator.itemgetter(1))
                leader = sorted_host_seqvalue[0][0]
                print(f"LEADER IS: {leader}")
                url = f"http://{leader}/update"
                response = requests.post(url, json={"key": key, "value": value})
        else:
            print(f"\033[31mPath {self.leadernode} does not exist.\033[0m")

    except Exception as e:
        print(f"\033[31mError in add_update: {e}\033[0m")
```

Zookeeper_Quorom.py read (key) method. This method was modified to randomly select Nr replicas and compare the data within them in order to check state consistency. If the data is consistent, a read operation is performed, otherwise there is a read conflict and the process is repeated. I implemented a 30 second timeout in the event that there are continual read conflicts.

```python
# Return the value for the key in the dictionary, otherwise return empty string
def read(self, key):
    i = 0
    timeout = 30

    # Track the start time
    start_time = time.time()

    # Repeat until replicas reach an agreement
    while True:
        # Check replicas
        print(f"\033[33mChecking replicas during read: {self.replicas}\033[0m")
        # For a size of N nodes, a quorom requires votes from at least Nw members. Where Nw >= N/2 + 1.
        N = len(self.replicas)
        Nw = min((N // 2) + 1, N) # Perform floor division
        valuesArr = []
        Nr = random.sample(self.replicas, Nw) # Randomly select Nr replicas
        count = len(Nr)
        print(f"Replica count = {count}")

        # Check if timeout has been exceeded
        elapsed_time = time.time() - start_time
        if elapsed_time > timeout:
            print("\033[31mTimeout reached! Exiting read operation.\033[0m")
            break # Break out of the loop

        for replica in Nr:
            try:
                value =  self.data_store.get(key, "")
                valuesArr.append(value)

                # Process when we iterate to the last replica
                if i == count-1:
                    values = [value[0] if value else None for value in valuesArr]
                    versions = [version[1] if version else None for version in valuesArr]
                    if len(set(values)) <= 1 and len(set(versions)) <= 1:
                        print(f"\033[33mValues/Versions: {valuesArr}\033[0m")
                        print(f"\033[33mValues: {values}\033[0m")
                        print(f"\033[33mVersions: {versions}\033[0m")
                        print(f"\033[34mQuorom reached. Updating data_store.\033[0m")
                        return self.data_store.get(key, "")
                    else:
                        print(f"\033[34mRead response: Conflict - Retrying\033[0m")
                    i += 1

            except requests.exceptions.RequestException as e:
                print(f"\033[31mError reading value for {replica}: {e}\033[0m")
```

Wagonblast, Gregory
CS2510

Project Testing:

Test Add/Read:

(a.) Three servers started in docker containers.

```
Composing Docker Environment...
time="2025-03-30T14:47:07-04:00" level=warning msg="Found orphan containers ([zk8 zk5 zk9 zk6 zk7 zk4]) for this project
. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag t
o clean it up."
 Container zoonavigator  Running
 Container zk3  Running
 Container zk1  Running
 Container zk2  Running
```

(b.) Zookeeper is used to implement the Bully algorithm in order to elect a leader.

```
Starting Server On 127.0.0.1:5000...
Created node: /election/127.0.0.1:5000_0000000005
Childrens: ['127.0.0.1:5000_0000000005']
sorted_host_seqvalue: [['127.0.0.1:5000', '0000000005']]
I am current leader: 127.0.0.1:5000
```

(c.) Votes are requested from the replicas in order to achieve a quorum.

```
Votes: 2
Replicas: ['127.0.0.1:5001', '127.0.0.1:5002']
```

(d.) A quorum is reached and the replica data stores are updated. Here we see zk1 update. This Add/Read is made through the leader. Versions are tracked for state consistency.

```
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:47:50] "GET /read?key=key2 HTTP/1.1" 200 -
Read response: {'key': 'key2', 'value': ''}
For Port: 5001
Checking replicas during read: ['127.0.0.1:5001']
Replica count = 1
Values/Versions: [['value0', 1]]
Values: ['value0']
Versions: [1]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:47:50] "GET /read?key=key0 HTTP/1.1" 200 -
Read response: {'key': 'key0', 'value': ['value0', 1]}
For Port: 5001
Checking replicas during read: ['127.0.0.1:5001']
Replica count = 1
Values/Versions: ['']
Values: [None]
Versions: [None]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:47:50] "GET /read?key=key1 HTTP/1.1" 200 -
Read response: {'key': 'key1', 'value': ''}
For Port: 5001
Checking replicas during read: ['127.0.0.1:5001']
Replica count = 1
Values/Versions: ['']
Values: [None]
Versions: [None]
```

(e.) I test the Add/Read through the leader and the replicas.

```
Testing Add and Read...
```

```
Testing Add and Read Through Replicas...
```

(f.) A quorum is reached and the replica data stores are updated. Here we see zk2 update. This Add/Read is made through a replica. Versions are tracked for state consistency.

```
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:47:50] "GET /read?key=key2 HTTP/1.1" 200 -
Read response: {'key': 'key2', 'value': ['value2', 2]}
For Port: 5002
Checking replicas during read: ['127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: [['value0', 1], ['value0', 1]]
Values: ['value0', 'value0']
Versions: [1, 1]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:47:50] "GET /read?key=key0 HTTP/1.1" 200 -
Read response: {'key': 'key0', 'value': ['value0', 1]}
For Port: 5002
Checking replicas during read: ['127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: ['', '']
Values: [None, None]
Versions: [None, None]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:47:50] "GET /read?key=key1 HTTP/1.1" 200 -
Read response: {'key': 'key1', 'value': ''}
For Port: 5002
Checking replicas during read: ['127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: [['value2', 2], ['value2', 2]]
Values: ['value2', 'value2']
Versions: [2, 2]
```

Test Kill the Leader:

(a.) Kill the leader.

```
I am the current leader to kill: 127.0.0.1:5000
127.0.0.1 - - [30/Mar/2025 14:48:30] "GET /kill HTTP/1.1" 200 -
Kill response: {'is_leader': True}
Testing Leader Election...
Killing the leader, electing a new one.
Stopping Server...
Starting Server On 127.0.0.1:5000...
Created node: /election/127.0.0.1:5000_0000000010
Childrens: ['127.0.0.1:5002_0000000009', '127.0.0.1:5001_0000000008', '127.0.0.1:5000_0000000010']
sorted_host_seqvalue: [['127.0.0.1:5001', '0000000008'], ['127.0.0.1:5002', '0000000009'], ['127.0.0.1:5000', '0000000010']]
I am a worker: 127.0.0.1:5000
```

(b.) A new leader is elected.

```
LEADER IS: 127.0.0.1:5001
Childrens: ['127.0.0.1:5002_0000000009', '127.0.0.1:5001_0000000008', '127.0.0.1:5000_0000000010']
sorted_host_seqvalue: [['127.0.0.1:5001', '0000000008'], ['127.0.0.1:5002', '0000000009'], ['127.0.0.1:5000', '0000000010']]
I am current leader: 127.0.0.1:5001
```

(c.) Heartbeat started on new leader (extra credit, explained below)

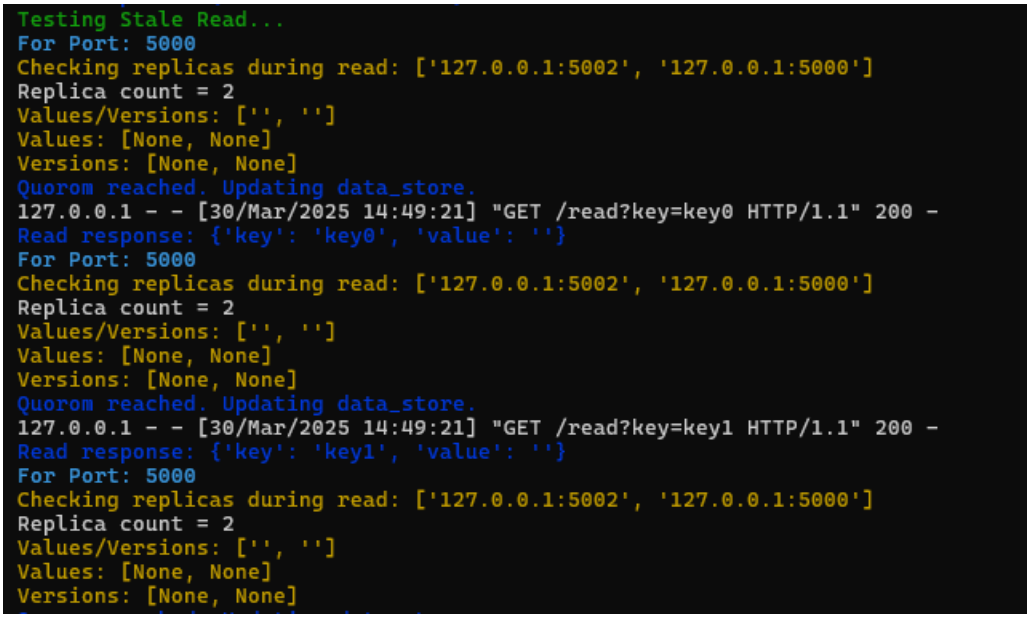
```
Heartbeat Started: False
Starting Heartbeat thread.
```

(d.)*Subsequent requests are routed through the new leader.*

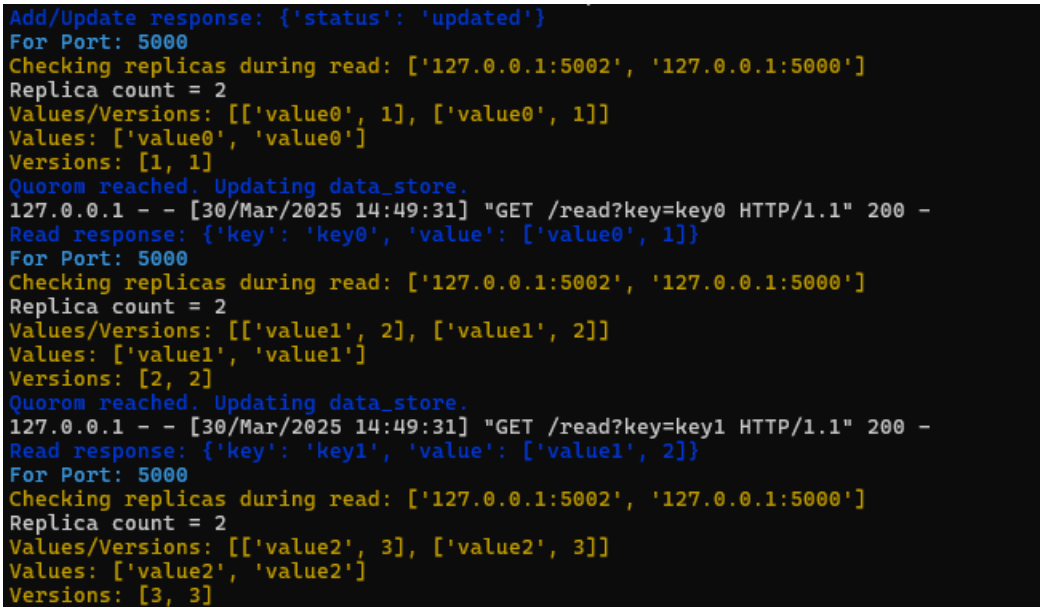
```
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:49:21] "GET /read?key=key2 HTTP/1.1" 200 -
Read response: {'key': 'key2', 'value': ['value2', 2]}
For Port: 5002
Checking replicas during read: ['127.0.0.1:5002', '127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: [['value0', 1], ['value0', 1]]
Values: ['value0', 'value0']
Versions: [1, 1]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:49:21] "GET /read?key=key0 HTTP/1.1" 200 -
Read response: {'key': 'key0', 'value': ['value0', 1]}
For Port: 5002
Checking replicas during read: ['127.0.0.1:5002', '127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: ['', '']
Values: [None, None]
Versions: [None, None]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:49:21] "GET /read?key=key1 HTTP/1.1" 200 -
Read response: {'key': 'key1', 'value': ''}
For Port: 5002
Checking replicas during read: ['127.0.0.1:5002', '127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: [['value2', 2], ['value2', 2]]
Values: ['value2', 'value2']
Versions: [2, 2]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:49:21] "GET /read?key=key2 HTTP/1.1" 200 -
Read response: {'key': 'key2', 'value': ['value2', 2]}
```

Test Stale Data:

(a.) The killed leader is back online and may have stale data.



(b.) Once key is updated, output. In this example, I update all key/value pairs.



Wagonblast, Gregory
CS2510

Test Timeout:

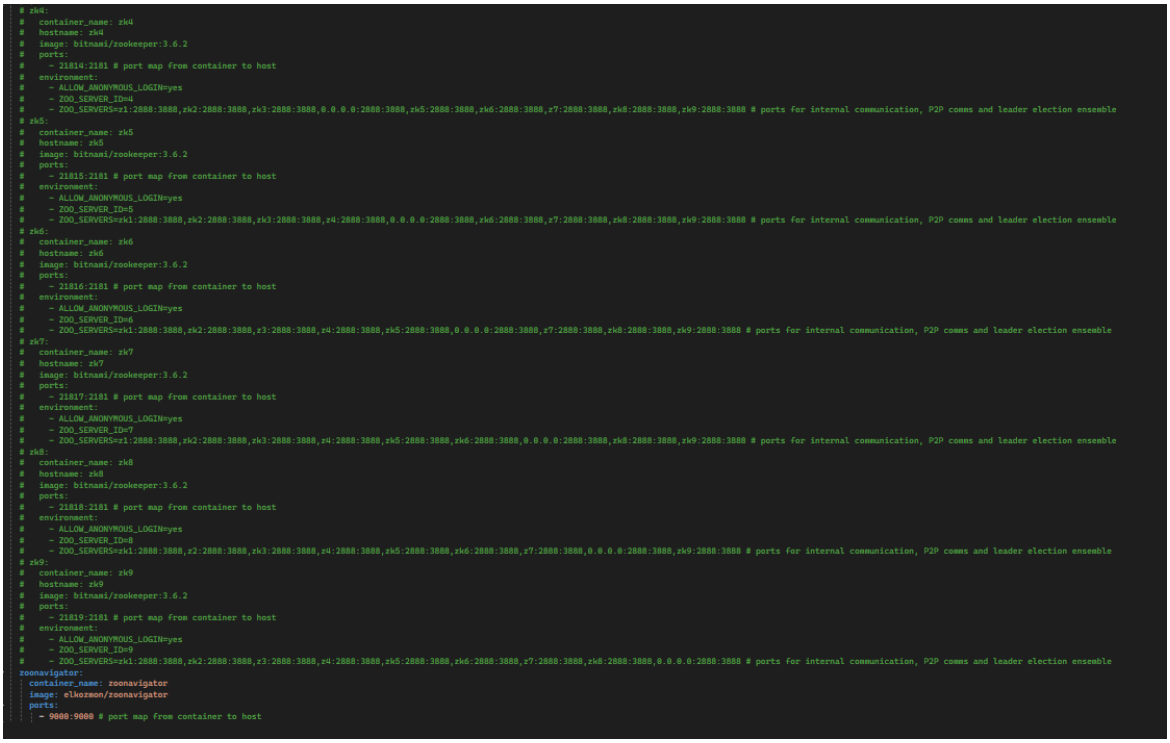
A timeout was implemented in the read(key) method. I pulled the logic out of that method and made an individualized test. The timeout works.

```python
# Timeout for testing
def timeout(self):
    timeout = 10

    # Track the start time
    start_time = time.time()

    # Repeat until replicas reach an agreement
    while True:
        # Check if timeout has been exceeded
        elapsed_time = time.time() - start_time
        if elapsed_time > timeout:
            print("\033[31mTimeout reached! Exiting read operation.\033[0m")
            break # Break out of the loop
```

```
Testing Timeout Logic...
Timeout reached! Exiting read operation.
127.0.0.1 - - [30/Mar/2025 14:49:21] "GET /timeout HTTP/1.1" 200 -
Timeout response: {'status': 'timeout complete'}
```

Stress Test:

I tested 9 zookeeper nodes, the Add/Read still functioned properly. These are commented out in the docker-compose.yml for the final submission because it prints too much to the console.



Extra Credit: ***My restore protocol is implemented by starting a heartbeat thread on the leader. This "heartbeat" monitors the status of the replicas, and ensures that the replica data stores are consistent periodically. If the leader OR any replicas crash, all nodes will be consistent with the current leader. This implementation is not efficient, but demonstrates a type of restore protocol.***

```python
# Heartbeat to check if replicas are alive
def heartbeat(self, replicas):
    while True:
        time.sleep(25)  # Send heartbeat every 25 seconds

        try:
            for replica in replicas:
                url = f"http://{replica}/heartbeat"
                response = requests.get(url, timeout=3)
                if response.status_code == 200:
                    print(f"\033[32mHeartbeat successful with replica {replica}\033[0m")
                    self.propagate_update(self.data_store)
                else:
                    print(f"\033[31mHeartbeat failed with replica {replica}\033[0m")
        except requests.exceptions.RequestException as e:
            print(f"\033[31mHeartbeat request error: {e}\033[0m")
```

```python
if not self.heartbeat_started and self.count == 1:
    # Starting heartbeat in a separate thread for leader
    print(f"\033[35mStarting Heartbeat thread.\033[0m")
    heartbeat_thread = threading.Thread(target=self.heartbeat, args=(self.replicas,), daemon=True)
    heartbeat_thread.start()
    self.heartbeat_started = True
```

Testing Extra Credit:

(a.) Begin testing by killing a replica.

```
Testing Replica Restore...
Killing a replica.
Stopping Server...
Starting Server On 127.0.0.1:5002...
```

(b.) A heartbeat thread is started on the current leader. This works after a new leader election as well.

```
Heartbeat Started: False
Starting Heartbeat thread.
```

(c.) This shows that after a replica is stopped and started it is read from with new data. After a heartbeat is triggered, the replicas data store is updated and becomes consistent with the current leader.

```
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:48:20] "GET /read?key=key0 HTTP/1.1" 200 -
Read response: {'key': 'key0', 'value': ''}
Checking replicas during read: ['127.0.0.1:5002', '127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: ['', '']
Values: [None, None]
Versions: [None, None]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:48:20] "GET /read?key=key1 HTTP/1.1" 200 -
Read response: {'key': 'key1', 'value': ''}
Checking replicas during read: ['127.0.0.1:5002', '127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: ['', '']
Values: [None, None]
Versions: [None, None]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:48:20] "GET /read?key=key2 HTTP/1.1" 200 -
Read response: {'key': 'key2', 'value': ''}
127.0.0.1 - - [30/Mar/2025 14:48:21] "GET /heartbeat HTTP/1.1" 200 -
Heartbeat successful with replica 127.0.0.1:5001
Replicas: ['127.0.0.1:5002', '127.0.0.1:5001', '127.0.0.1:5002']
Propagating....127.0.0.1 - - [30/Mar/2025 14:48:21] "POST /propagate HTTP/1.1" 200 -

Successfully propagated update to 127.0.0.1:5002
Propagating....127.0.0.1 - - [30/Mar/2025 14:48:21] "POST /propagate HTTP/1.1" 200 -

Successfully propagated update to 127.0.0.1:5001
Propagating....
127.0.0.1 - - [30/Mar/2025 14:48:21] "POST /propagate HTTP/1.1" 200 -
Successfully propagated update to 127.0.0.1:5002
Childrens: ['127.0.0.1:5002_0000000009', '127.0.0.1:5000_0000000005', '127.0.0.1:5001_0000000008']
sorted_host_seqvalue: [['127.0.0.1:5000', '0000000005'], ['127.0.0.1:5001', '0000000008'], ['127.0.0.1:5002', '0000000009']]
I am current leader: 127.0.0.1:5000
Heartbeat Started: True
Checking replicas during read: ['127.0.0.1:5002', '127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: [['value0', 1], ['value0', 1]]
Values: ['value0', 'value0']
Versions: [1, 1]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:48:30] "GET /read?key=key0 HTTP/1.1" 200 -
Read response: {'key': 'key0', 'value': ['value0', 1]}
Checking replicas during read: ['127.0.0.1:5002', '127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: ['', '']
Values: [None, None]
Versions: [None, None]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:48:30] "GET /read?key=key1 HTTP/1.1" 200 -
Read response: {'key': 'key1', 'value': ''}
Checking replicas during read: ['127.0.0.1:5002', '127.0.0.1:5001', '127.0.0.1:5002']
Replica count = 2
Values/Versions: [['value2', 2], ['value2', 2]]
Values: ['value2', 'value2']
Versions: [2, 2]
Quorom reached. Updating data_store.
127.0.0.1 - - [30/Mar/2025 14:48:30] "GET /read?key=key2 HTTP/1.1" 200 -
Read response: {'key': 'key2', 'value': ['value2', 2]}
```