SOSP21 Artifact Evaluation Instructions

#617 Rabia: Simplifying State-Machine Replication Through Randomization

1.1 Summary

- We have four evaluations reported in the paper:
  - Performance without Batching: Table 1
  - Throughput vs. Latency: Figure 4
  - Varying Data Size: No table/figure (text only)
  - Integration with Redis: Figure 5

  For these evaluations, we provide either the binaries (for competitors) or compile scripts.

- We have also provided all the necessary proof scripts for formally verifying our algorithm. See https://github.com/haochenpan/rabia/tree/main/proofs

- Rabia consists of x server VMs and y client VMs. In our evaluations, we have tested x = 3 or 5, and y = 1 or 3.

- Key parameters of Rabia that need to be configured are located in two files:
  - "profileX.sh" configurations -- adjust the parameters below in deployment/profile/profile0.sh
    - NServers -- the number of servers (n)
    - NFaulty -- the number of faulty servers (f)
    - NClients -- the number of clients
    - ClientTimeout -- the runtime of each client
    - ClientBatchSize -- the number of KV-store commands packed in a single client request
    - ProxyBatchSize -- the maximum number of client-batched requests packed in a single consensus object and hence stored in a single consensus slot
    - Other parameters that are not essential for the artifact evaluation: see comments in deployment/profile/profile0.sh

  - "config.go" configurations -- adjust the parameters below in internal/config/config.go
    - "c.KeyLen" and "c.ValLen" at line 162-163 need to be changed for the "Varying Data Size" test. These two parameters specify the lengths of keys and values of the KV-store, respectively
    - "c.StorageMode" at line 171 needs to be changed to "2" for the Rabia-Redis test. 2 stands for the MSET & MGET mode.

- ○ Note: If the evaluation instruction asks you to change a file (i.e., config.go or profile.sh), please change that file on ALL VMs

- In the first few sets of experiments, we compare Rabia against EPaxos and Paxos using the EPaxos repo[1]. We mainly edited two parts: (i) we fixed the way Go imports modules, and (ii) we modified the original codebase so that EPaxos and Paxos support the non-pipelining execution (to produce Table 1).

- EPaxos and Paxos do not provide an easy way to adjust parameters; hence, we create several branches, one for each combination of the parameters that we use. We will specifically mention which branch to use[2].

- In the last set of experiments, we compare Rabia against our implementation of synchronous replication on Redis and RedisRaft.

- Our paper reported data collected on Google Cloud Platform (GCP). Here, we explain how to replicate it on CloudLab. We expect the data to be a little bit off, because computation and communication resources, such as CPU specifications, memory frequencies, network bandwidths, and network stability, are different in two platforms. Among these, network stability affects the numbers the most. For example, CloudLab has average RTT roughly 0.13 ms, whereas GCP has average RTT roughly 0.25 ms. Due to the property of Rabia, the performance improvement due to the lower RTT is not as great as Paxos and EPaxos.

- One set of evaluations (Figure 4c in the paper) might be difficult to achieve on CloudLab, because it does not provide multiple availability zones. We originally conducted the experiments on Google Cloud Platform (GCP), see steps to run at section 2.2.

- For the reviewers' convenience, we have a video walkthrough of section 2.1: https://youtu.be/LPjyibkhgYc

1.2 Setup on CloudLab: General

- Instantiate the following profile, which consists of three server VMs and three client VMs with these VMs being connected with each other on the experiment network.

1.3 Setup on CloudLab: Rabia

- Install Rabia by entering the commands (blue texts) below on EACH VM
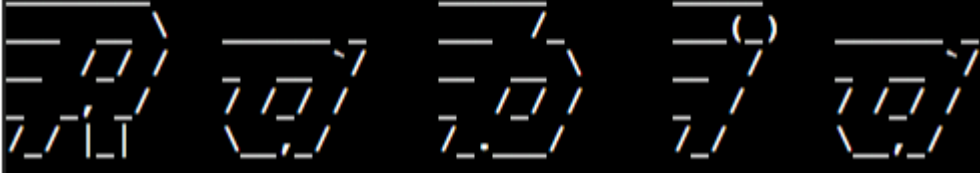
  - ○ # 0.use root

---

[1] https://github.com/efficient/epaxos
[2] The modified EPaxos and Paxos can be found at https://github.com/zhouaea/epaxos-single for non-pipelined version, and https://github.com/zhouaea/epaxos for pipelined version.

- ○ sudo su

- ○ # 1. download the project to the default path (see section 1.3)
- ○ mkdir -p ~/go/src && cd ~/go/src
- ○ git clone https://github.com/haochenpan/rabia.git

- ○ # 2. install Rabia and its dependencies
- ○ cd ./rabia/deployment
- ○ . ./install/install.sh
- ○ # note: this may take a while (3~5 mins), as the script is getting all the dependencies, including Redis and RedisRaft

- ○ # 3. check the installation:
- ○ cd ./run
- ○ # the default parameter runs a Rabia cluster on a single machine for 20 seconds (see the note below)
- ○ . single.sh
- ○ # inspect the performance statistics; one should see a few lines of file output
- ○ cat ../../result.txt
- ○ # remove logs, result.txt, and the built binary
- ○ . clear.sh

● Note: if Rabia starts successfully, you should see the following:

```
root@ms1222:~/go/src/rabia/deployment/run# . single.sh
network =  1 successfully connected to all servers
network =  0 successfully connected to all servers
network =  2 successfully connected to all servers
all servers and clients are connected
```

1.4 Setup on CloudLab: Non-Pipelined Paxos and EPaxos (Table 1)

- ● Note: You only need to perform these steps if you're evaluating Table 1. For other EPaxos/Paxos evaluation, we provide binaries in the Rabia repository.

- ● Make sure Rabia is properly installed. Follow the instructions in Section 1.3. This step is critical as it provides the go binary and python3.8 needed for testing.

- ● Install Paxos/EPaxos (NP) by entering the commands below on EACH VM

- # 0. use root
- sudo su
-
- # 1. download the project to the default path (see section 1.3) and switch to the Paxos (NP) or EPaxos (NP) branch. Assuming you already have ~/go/src
- cd ~/go/src
- git clone https://github.com/zhouaea/epaxos-single.git && cd epaxos-single
- git checkout paxos-no-pipelining-no-batching OR git checkout epaxos-no-pipelining-no-batching

- # 2. compile Paxos or EPaxos (on each VM) (This will not output anything if successful.)
- cd ~/go/src/epaxos-single
- . compilePaxos.sh OR . compileEPaxos.sh

1.5 Setup on CloudLab: Paxos and EPaxos

- Note: You only need to perform these steps if you're evaluating Table 1. For other EPaxos/Paxos evaluation, we provide binaries.

- Install Paxos/EPaxos by entering the commands below on EACH VM. This will only be used for Table 1 experiments.
    - # 0. use root
    - sudo su
    -
    - # 1. download the project to the default path (see section 1.3) and switch to the Paxos (NP) or EPaxos (NP) branch. Assuming you already have ~/go/src
    - cd ~/go/src
    - git clone https://github.com/zhouaea/epaxos.git && cd epaxos
    - git checkout paxos-no-batching OR git checkout epaxos-no-batching
    -
    - # 2. compile Paxos or EPaxos (on each VM) (This will not output anything if successful.)
    - cd ~/go/src/epaxos
    - . compile.sh

- For all experiments besides those done for Table 1, please follow the instructions below to launch the exact binary of EPaxos and Paxos for each evaluation below.

1.6 Setup on Cloudlab: Disable Cores:

- We used 2-core server VMs to collect our data, where each core ran 2 threads (i.e., 4 vCPU). Hence, it is necessary to disable 12 cores on server VMs to provide a comparable specification, since the smallest raw machine has 16 cpus on CloudLab (8 cores, 2 threads per core).

- Follow the steps below to disable cores on <u>server VMs after installing Rabia</u>:
  - Run as super user
    - sudo su
  - To check CPU status
    - lscpu
  - Disable necessary cpu cores
    - . ~/go/src/rabia/deployment/env/disable_cpu.sh
  - In case that you accidentally use the disable_cpu script on the client VMs, you can use the following script to re-enable all cpu cores.
    Note: in all the tests, we <u>do not need</u> to disable cores on client VMs.
    - . ~/go/src/rabia/deployment/env/reenable_cpu.sh
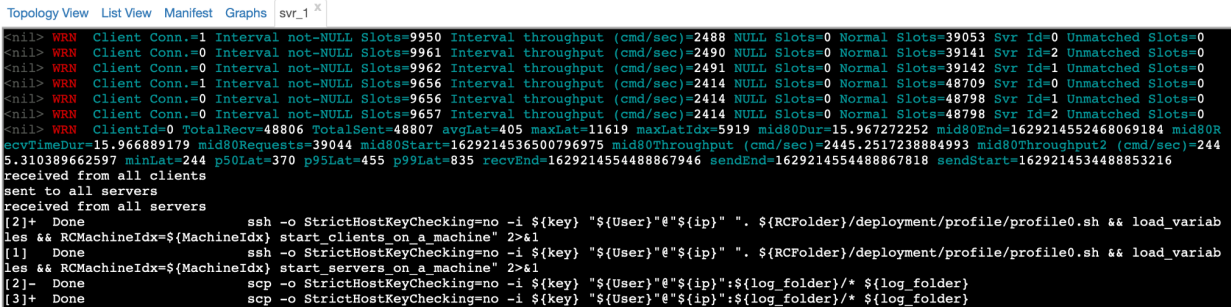
2.1 Performance without Batching (Table 1)

2.1.1 Rabia

- Configure VMs
  - System setup: three Rabia server VMs and one client VM. This experiment uses only one client VM; our profile provides three so that it's easier to run other experiments.
  - Make sure all the steps in "Setup on CloudLab: General" and "Setup on CloudLab: Rabia" are completed on ALL VMs
  - Make sure you're in the folder of ~/go/src/rabia
  - Configure the following parameters by editing deployment/profile/profile0.sh <u>on ALL VMs</u>
    - Find the experiment network ip address of all three server VMs and one client VM that you chose to work on.
      - Note: these IPs usually appear as 10.10.1.x in CloudLab (see section below titled *Note: Find "experiment network IP" on Cloudlab*)
    - Fill ServerIps with 3 server VMs' internal IPs. e.g.,
      - ServerIps=(10.10.1.1 10.10.1.2 10.10.1.3)
      - Note: white space between each pair of IPs
      - Fill ClientIps with the client VM' internal IPs. e.g.,
      - ClientIps=(10.10.1.1)
    - Let Controller be the first server VM's IP:some unused port, e.g.,
      - Controller=10.10.1.1:8070
    - Replace the "run_once" function call at the bottom of deployment/run/multiple.sh to the followings (the result looks like the figure below):
      - run_once ${RCFolder}/deployment/profile/profile_sosp_table1.sh

```
source ../profile/profile0.sh # required, load the variables and functions
run_once  ${RCFolder}/deployment/profile/profile_sosp_table1.sh
```

- Run Rabia
  - At the first server VM (which will act as the <u>controller</u> of the experiment):
    - cd ~/go/src/rabia/deployment/run
    - . multiple.sh
    - After 120-150 seconds, the run should be over. An example output is shown below.

    - Check result.txt in ~/go/src/rabia of the <u>controller VM</u> to retrieve performance statistics. You may see a bunch of numbers separated by commas in the file.
    - You can copy them to a Google Sheet, select the first column, and click "Data" -> "Split text to columns" -> "Separator: Comma" to get a human-readable presentation.
- Note: the other evaluation for Rabia basically follows the same recipe. We will explicitly specify the parameters that need to be changed and the profile to be used.

Note: Find "experiment network IP" on CloudLab

- Find the experiment network ip address of the desired node in your rspec manifest.
- First, find the description of your machine by looking for a <node> tag with the attribute client_id=<your_machine_name>.
- Inside the node tag, look for <interface>, and inside it you should see an <ip/> tag. The attribute address will give you the machine's experiment network ip. 2. The address likely has the format 10.10.1.x.

2.1.2 Paxos (NP) and EPaxos (NP) -- NP stands for Non-Pipelining version

- You need to run the following steps twice. One for Paxos (NP) and the other for EPaxos (NP).
- System setup: three paxos/epaxos (NP) server VMs and one client VM. You can use the same VMs used in the Rabia test.
- Follow Section 1.4 to manually create binaries and switch to the correct directory and branch (paxos-no-pipelining-no-batching or epaxos-no-pipelining-no-batching)
- System setup: three server VMs and one client VM
- Configure Master
  - Choose one of your server VMs to be the master server. The other two will be replicas.
  - Edit runMasterServer.sh in ~/go/src/epaxos-single
    - Set MASTER_SERVER_IP equal to the server machine's experiment network IP address (see section above titled *Note: Find "experiment network IP" on Cloudlab*).
- Configure Replicas
  - You need to perform the following on the other two replicas
  - Find its experiment network ip address.
  - Edit runServer.sh in ~/go/src/epaxos-single
    - Set MASTER_SERVER_IP equal to the master server VM's experiment network IP address.
    - Set REPLICA_SERVER_IP to the current server VM's experiment network IP address.
- Configure the Client
  - Edit runClient.sh in ~/go/src/epaxos-single
    - Set MASTER_SERVER_IP equal to the master server VM's experiment network ip address.
- Run Paxos (NP) or EPaxos (NP)
  - In your master server VM, run . runMasterServer.sh in ~/go/src/epaxos-single
  - In every replica server VM, run . runServer.sh in ~/go/src/epaxos-single
  - Once all servers are connected, you should see messages from rpc.Register and the message
        Replica id: x. Done connecting to peers
  - Alternatively, you may see the error below. If this is the case, you did not input the network ip addresses correctly in one or more machines. Type . kill.sh in each server VM to terminate server processes and retry the configuration steps above.

    [signal SIGSEGV: segmentation violation code=0x1 addr=0x18 pc=0x560ac5]
    goroutine 56 [running]:
    genericsmr.(*Replica).waitForPeerConnections(0xc00027c000, 0xc000096480)
          /root/go/src/epaxos-single/src/genericsmr/genericsmr.go:196 +0x105
    created by genericsmr.(*Replica).ConnectToPeers

- ○ Finally, run . runClient.sh > run.txt in the terminal of your client VM.
  - ■ After a couple of seconds, press the enter key a few times on your keyboard.
  - ■ You should see a bunch of DONE messages, equal to the number of NClients specified.
  - ■ You can also use the ps command to check for pending clients.
  - ■ Verify the contents of run.txt. It should contain a list of start and end times.
  - ■ You'll know runClient.sh failed if any client in NClients outputs Error connecting to replica 0 after a timeout period of around 1 minute.
- Analysis
  - ○ Dependencies:
    - ■ We use python3.8 for our analysis scripts, which should have been downloaded when you installed Rabia.
    - ■ Note: our latency analysis script requires the numpy package, which should have been downloaded when you installed Rabia. If it somehow did not get installed, we have provided a shell script in the repo that will install pip and numpy for python3.8 that can be run with . download_numpy.py
  - ○ Latency and throughput analysis:
    - ■ In your client VM, run . calculate_throughput_latency.sh. This will run both latency_analysis.py and throughput_analysis.py
    - ■ You may get an error from calculate_throughput_latency.sh if you run it before runClient.sh has terminated, or if runClient.sh failed.
  - ○ Note:
    - ■ If you see SyntaxError: Non-ASCII character '\xc2' in file latency_analysis.py on line 21, but no encoding declared; see http://python.org/dev/peps/pep-0263/ for details. It's likely because you're using python 2 instead of python 3.

2.1.3 Paxos and Epaxos

- Note: the workflow for EPaxos/Paxos is different from the one for EPaxos (NP) and Paxos (NP)
- You need to run the following steps twice. One for Paxos and the other for EPaxos.
- System setup: three paxos/epaxos server VMs and one client VM. You can use the same VMs used in the Rabia test.
- Follow Section 1.5 to manually create binaries and switch to the correct directory and branch (paxos-no-batching or epaxos-no-batching)
- Configure Each VM:
  - ○ Choose one of your server VMs to be the master server.
  - ○ **On each VM**, configure the following parameters by editing base-profile.sh:
    - ■ Fill ServerIps with 3 server VMs' IPs. e.g.,

- - - ServerIps=(10.10.1.1 10.10.1.2 10.10.1.3)
      - Note: white space between each pair of IPs
      - Note: IPs should be experiment network ip addresses, which usually appears as 10.10.1.x in CloudLab (see section above titled *Note: Find "experiment network IP" on Cloudlab*)
    - Fill ClientIps with the client VMs' IPs. e.g.,
      - ClientIps=(10.10.1.4)
    - Let MasterIp be the first VM's IP, e.g.,
      - MasterIp=10.10.1.1
- Run Paxos or EPaxos
  - Finally, run . runPaxos.sh or . runEPaxos.sh (depending on the branch) on your master VM only. You should see a relatively constant flow of messages in your terminal.
  - If all works correctly, there will be n client logs inside the /logs directory in your master machine.
    - If something doesn't work correctly, run . kill.sh and try to execute again.
    - Alternatively, you can manually search and kill processes
      - Use ps to find active processes
      - Use kill -9 <pid> to kill a process
- Analysis
  - Dependencies:
    - This script uses python3.8, which should have been downloaded when you installed Rabia.
  - For throughput/latency analysis, run python3.8 analysis.py ./logs
  - If you get the error below, you didn't supply ./logs as an argument
    - Traceback (most recent call last):
    - File "analysis.py", line 164, in <module>
    - infos, disconnects = analysis_epaxos_logs()
    - File "analysis.py", line 100, in analysis_epaxos_logs
    - params = get_experiments(argv[1])
    - IndexError: list index out of range
  - If successful, this will output a variety of different statistics. You can find the relevant statistics as clientp50Latency: x, clientp99Latency: x, throughput: x.

2.2 Throughput vs. Latency (Figure 4)

2.2.1 Rabia

- Figure 4a, 4b
  - System setup: three server VMs and three client VMs
  - Following the steps in "Performance without Batching," run Rabia on CloudLab with profile profile_sosp_4abc.sh:
    - i. Make sure ServerIps, ClientIps, Controller are adjusted in the way described in "Performance without Batching"
    - ii. Replace the "run_once" function call at the bottom of deployment/run/multiple.sh to the followings:
      - 1. run_once ${RCFolder}/deployment/profile/profile_sosp_4abc.sh
  - Adjust the number of clients (parameter NClients) manually after a run to produce a series of data points (the exact numbers are in the profile, which range from 20 to 500).
  - Copy the content of result.txt to a Google Sheet, check the columns of median latency and 99-percentile latency.
    Note: You may need to split your pasted data into columns by clicking Data > Split text to columns

- Figure 4c (this experiment requires multiple availability zones, which unfortunately is not supported by CloudLab but available on GCP)
  - System setup: three server VMs and three client VMs
    - i. 3 server VMs one in us-east-1-b, one in us-east-1-c, and the last in us-east-1-d
    - ii. 3 server VMs: e2-highmem-4 instances (Intel Xeon 2.8GHz, 4 vCPUs, 32 GB RAM)
    - iii. 3 client VMs in us-east-1-b, each with 30 vCPUs and 120 GB RAM
    - iv. OS for all VMs: Ubuntu-1604
  - Following the steps in "Performance without Batching," run Rabia on GCP with profile profile_sosp_4abc.sh (see the note at the end of this file).
    - i. Make sure ServerIps, ClientIps, Controller are adjusted in the way described in "Performance without Batching"
    - ii. Replace the "run_once" function call at the bottom of deployment/run/multiple.sh to the followings:
      - 1. run_once ${RCFolder}/deployment/profile/profile_sosp_4abc.sh
  - Adjust the number of clients (parameter NClients) manually after a run to produce a series of data points (the exact numbers are in the profile, which range from 20 to 500).
  - Copy the content of result.txt to a Google Sheet, check the columns of median latency and 99-percentile latency.
    Note: You may need to split your pasted data into columns by clicking Data > Split text to columns

- Figure 4d
  - System setup: five server VMs and three client VMs
  - Instantiate the provided profile

- ○ Following the steps in "Performance without Batching," run Rabia on CloudLab with profile profile_sosp_4d.sh:
    - i. Make sure ServerIps (previously 3 entries, now 5 entries), ClientIps, Controller are adjusted in the way described in "Performance without Batching"
    - ii. Replace the "run_once" function call at the bottom of deployment/run/multiple.sh to the followings:
        1. run_once ${RCFolder}/deployment/profile/profile_sosp_4d.sh
- ○ Adjust the number of clients (parameter NClients) manually after a run to produce a series of data points (the exact numbers are in the profile, which range from 20 to 500).
- ○ Copy the content of result.txt to a Google Sheet, check the columns of median latency and 99-percentile latency.
  Note: You may need to split your pasted data into columns by clicking Data > Split text to columns

2.2.2 EPaxos, Paxos

- ● Figure 4a, 4b
    - ○ System setup: three server VMs and three client VMs
    - ○ On each VM, open the paxos-batching or epaxos-batching folder with cd ~/go/src/rabia/epaxos/figure-4/paxos-batching or ~/go/src/rabia/epaxos/figure-4/epaxos-batching, respectively.
    - ○ Configure Each VM:
        - i. Choose one of your server VMs to be the master server.
        - ii. **On each VM**, configure the following parameters by editing gre:
            1. Fill ServerIps with 3 server VMs' IPs. e.g.,
                a. ServerIps=(10.10.1.1 10.10.1.2 10.10.1.3)
                b. Note: white space between each pair of IPs
                c. Note: IPs should be experiment network ip addresses, which usually appears as 10.10.1.x in CloudLab (see section above titled *Note: Find "experiment network IP" on Cloudlab*)
            2. Fill ClientIps with the client VMs' IPs. e.g.,
                a. ClientIps=(10.10.1.4, 10.10.1.5 10.10.1.6)
            3. Let MasterIp be the first VM's IP, e.g.,
                a. MasterIp=10.10.1.1
        - iii. In our paper, we run the Paxos/EPaxos configuration 9 times with varying clients (ranging from 20 to 500). profile0.sh contains code to run with 20 clients, while profile8.sh will run with 500 clients. To choose which profile is being executed, edit the second line of runBatchedPaxos or runBatchedEPaxos in each machine.

- ○ Run Paxos or EPaxos
    - i. **On the master VM**, run . runBatchedPaxos or . runBatchedEPaxos. You should see a relatively constant flow of messages in your terminal.
    - ii. If all works correctly, there will be n client logs inside the /logs directory in your master machine.
        1. If something doesn't work correctly, run . kill.sh and try to execute again.
        2. Alternatively, you can manually search and kill processes
            a. Use ps to find active processes
            b. Use kill -9 <pid> to kill a process
- ○ Analysis
    - i. Dependencies:
        1. This script uses python3.8, which should have been downloaded when you installed Rabia.
    - ii. For throughput/latency analysis, run python3.8 analysis.py ./logs
    - iii. If you get the error below, you didn't supply ./logs as an argument
        1. Traceback (most recent call last):
        2. File "analysis.py", line 164, in <module>
        3. infos, disconnects = analysis_epaxos_logs()
        4. File "analysis.py", line 100, in analysis_epaxos_logs
        5. params = get_experiments(argv[1])
        6. IndexError: list index out of range
    - iv. If successful, this will output a variety of different statistics. You can find the relevant statistics as clientp50Latency: x, clientp99Latency: x, throughput: x.

- ● Figure 4c (this experiment requires multiple availability zones, which unfortunately is not supported by CloudLab but is available on GCP)
    - ○ System setup: three server VMs and three client VMs
        - i. 3 server VMs one in us-east-1-b, one in us-east-1-c, and the last in us-east-1-d
        - ii. 3 server VMs: e2-highmem-4 instances (Intel Xeon 2.8GHz, 4 vCPUs, 32 GB RAM)
        - iii. 3 client VMs in us-east-1-b, each with 30 vCPUs and 120 GB RAM
        - iv. OS for all VMs: Ubuntu-1604
    - ○ With Rabia installed on machines in GCP, follow the instructions for "Figure 4a, 4b." The only difference is that instead of using experiment network ips, you will use GCP's internal network IPs.

- ● Figure 4d
    - ○ System setup: five server VMs and three client VMs
    - ○ With Rabia installed on machines in Cloudlab, follow the instructions for Figure 4a, 4b. The only difference is that instead of configuring 3 server ips, you need to configure 5.

2.3 Varying Data Size (No Figure or Table)

2.3.1 Rabia

- System setup: three server VMs and three client VMs
- The base configuration for comparison (the setup of that produces Figure 4a) uses 16B key-value pairs. For this section, change the size to 256B: at each VM, in internal/config/config.go, at line 162-163, set
    - c.KeyLen = 128, c.ValLen = 128
- Following the steps in "Performance without Batching," run Rabia on CloudLab with profile profile_sosp_vd.sh:
    - Make sure ServerIps, ClientIps, Controller are adjusted in the way described in "Performance without Batching"
    - Replace the "run_once" function call at the bottom of deployment/run/multiple.sh to the followings:
        - run_once ${RCFolder}/deployment/profile/profile_sosp_vd.sh
- Run Rabia according to the "Run Rabia" steps in "Performance without Batching."
- Copy the content of result.txt to a Google Sheet, check the first column for throughput

2.3.2 EPaxos, Paxos

- System setup: three server VMs and three client VMs
- **On each VM**, open the paxos-batching-data-size-256B or epaxos-batching-data-size-256B folder with cd ~/go/src/rabia/epaxos/varying-data-size/paxos-batching-data-size-256B or ~/go/src/rabia/epaxos/varying-data-size/epaxos-batching-data-size-256B, respectively.
- Configure **each VM**
    - Choose one of your server VMs to be the master server.
    - In each machine, configure the following parameters by editing base-profile.sh:
        - i. Fill ServerIps with 3 server VMs' IPs. e.g.,
            1. ServerIps=(10.10.1.1 10.10.1.2 10.10.1.3)
            2. Note: white space between each pair of IPs
            3. Note: IPs should be experiment network ip addresses, which usually appears as 10.10.1.x in CloudLab (see section above titled *Note: Find "experiment network IP" on Cloudlab*)
        - ii. Fill ClientIps with the client VMs' IPs. e.g.,
            1. ClientIps=(10.10.1.4, 10.10.1.5 10.10.1.6)
        - iii. Let MasterIp be the first VM's IP, e.g.,
            1. MasterIp=10.10.1.1
    - In our paper, we run the Paxos/EPaxos configuration 9 times with varying clients (ranging from 20 to 500). profile0.sh contains code to run with 20 clients, while profile8.sh will run with 500 clients. To choose which profile is being executed, edit the second line of runBatchedPaxos or runBatchedEPaxos in each machine.

- Run Paxos or EPaxos
  - On the **master VM**, run . runBatchedPaxos.sh or . runBatchedEPaxos.sh. You should see a relatively constant flow of messages in your terminal.
  - If all works correctly, there will be n client logs inside the /logs directory in your master machine.
    i. If something didn't work correctly, run . kill.sh and try to execute again.
    ii. Alternatively, you can manually search and kill processes
      1. Use ps to find active processes
      2. Use kill -9 <pid> to kill a process
- Analysis
  - Dependencies:
    i. This script uses python3.8, which should have been downloaded when you installed Rabia.
  - For throughput/latency analysis, run python3.8 analysis.py ./logs
  - If you get the error below, you didn't supply ./logs as an argument
    i. Traceback (most recent call last):
    ii. File "analysis.py", line 164, in <module>
    iii. infos, disconnects = analysis_epaxos_logs()
    iv. File "analysis.py", line 100, in analysis_epaxos_logs
    v. params = get_experiments(argv[1])
    vi. IndexError: list index out of range
  - If successful, this will output a variety of different statistics. You can find the relevant statistics as clientp50Latency: x, clientp99Latency: x, throughput: x.

2.4 Integration with Redis (Figure 5)

2.4.1 Sync-Rep (Synchronous replication using standalone Redis)

- There are two configurations:
  - (i) Sync-Rep (1): two VMs, one as the Redis leader and the other as a follower; and
  - (ii) Sync-Rep (2): three VMs, one as the Redis leader and the other two as followers.
  - Note: We run clients on follower VMs to force the system to have one RTT so that it's compatible with SMR-based approach.
- Configuration
  - On each VM, complete the following steps:
    - Follow the steps in "Setup on CloudLab: Rabia" to clone Rabia and install necessary dependencies, including Redis. Following the steps in "Performance without Batching"
    - # Clone code for Sync-Rep
    - cd ~/go/src

- - - git clone https://github.com/YichengShen/redis-sync-rep.git
    - cd redis-sync-rep
  - Configure IP of master VM in config.yaml
  - Start Redis server: You could configure the current VM either as a master or a replica.
    - configure as a master
      . ./deployment/startRedis/startServer.sh
    - configure as a replica
      . ./deployment/startRedis/startServer.sh replica
    - Note: if run successfully, you should see
      OK
      PONG
- Run Sync-Rep
  - Change the following parameters in config.yaml to produce the lower bar
    - NClients=1 and ClientBatchSize=1
  - On one of the follower VMs, run the main program
    - go run main.go
  - Change the following parameters in config.yaml to produce the taller bar
    - NClients=15 and ClientBatchSize=20
  - On one of the follower VMs, run the main program
    - go run main.go

2.4.2 Redis-Rabia

- System setup: three server VMs and three client VMs
- For each of the three Rabia **server VMs**, start a stand-alone Redis instance on the VM (n server requires n Redis instance), and start a client to monitor the Redis:
  - # on server-0
  - src/redis-server --port 6379 --appendonly no --save "" --daemonize yes
  - src/redis-cli -p 6379
  - # on server-1
  - src/redis-server --port 6380 --appendonly no --save "" --daemonize yes
  - src/redis-cli -p 6380
  - # on server-2
  - src/redis-server --port 6381 --appendonly no --save "" --daemonize yes
  - src/redis-cli -p 6381
- Adjusting the StorageMode in config.go on all VMs
  - In config.go, adjust the `c.StorageMode` in `loadRedisVars()` to 2: use Redis' MGET and MSET commands only
    Note: path to the file: internal/config/config.go

- Following the steps in "Performance without Batching," run Rabia on CloudLab with profile profile_sosp_5a.sh to generate the low bar:
  - Make sure ServerIps, ClientIps, Controller are adjusted in the way described in "Performance without Batching"
  - Replace the "run_once" function call at the bottom of deployment/run/multiple.sh to the followings:
    - run_once ${RCFolder}/deployment/profile/profile_sosp_5a.sh
- After a run, you may want to type `keys *` in each redis-cli's terminal to see whether keys are written to the DB.
  Note: this operation takes a significantly long time if there are many keys.
- Finally, run `flushall` to truncate the DB to prepare for the next clean run.
- Following the steps in "Performance without Batching," run Rabia on CloudLab with profile profile_sosp_5b.sh to generate the high bar:
  - Make sure ServerIps, ClientIps, Controller are adjusted in the way described in "Performance without Batching"
  - Replace the "run_once" function call at the bottom of deployment/run/multiple.sh to the followings:
    - run_once ${RCFolder}/deployment/profile/profile_sosp_5b.sh
- After a run, you may want to type `keys *` in each redis-cli's terminal to see whether keys are written to the DB.
  Note: this operation takes a significantly long time if there are many keys.
- Finally, run `flushall` to truncate the DB to prepare for the next clean run.
- **Note: If you need to re-run the previous evaluation, reset StorageMode in config.go to 0 on all VMs after these runs.**

2.4.3 Redis-Raft

- **Install Redis-Raft**
  Note: These instructions assume you have the Rabia project cloned on the standard Cloudlab profile (deployment/env/cloudlab)
  - sudo su && cd
  - # clone the project to root
  - git clone https://github.com/RedisLabs/redisraft.git
  -
  - # install the project's build deps
  - cd ~/go/src/rabia/redis-raft && . install.sh
  -
  - # build the project to generate redisraft.so binary.
  - . build.sh
- **Run Redis-Raft**
  - System setup: three VMs -- Redis-Raft supports clients running on server VMs, so we did not use separate client VMs.

- ○ This script assumes port 5001 is open and unused. You may use any free port you like.
- ○ HOST (1..n) represents the internal IP. (See section above titled Note: Find "experiment network IP" on Cloudlab)
- ○ RedisRaft recommends an odd number of nodes in a cluster. We have tested on 3.
- ○ **Steps**
- ○ On the first VM (this will act as the leader), type:
  - ■ redis-server --bind <HOST 1> --port 5001 --dbfilename raft1.rdb --loadmodule ~/redisraft/redisraft.so raft-log-filename raftlog1.db addr <HOST 1>:5001 &
  - ■ redis-cli -h <HOST 1> -p 5001 RAFT.CLUSTER INIT
  - ■ redis-cli -h <HOST 1> -p 5001 RAFT.CONFIG SET raft-log-fsync no
- ○ In subsequent two VMs, type:
  - ■ redis-server --bind <HOST n> --port 5001 --dbfilename raft<n>.rdb --loadmodule ~/redisraft/redisraft.so raft-log-filename raftlog<n>.db addr <HOST n>:5001 &
  - ■ redis-cli -h <HOST n> -p 5001 RAFT.CLUSTER JOIN <HOST 1>:5001
  - ■ redis-cli -h <HOST n> -p 5001 RAFT.CONFIG SET raft-log-fsync no
- ● **To view the performance, in a new terminal on a leader VM, type**
  - ○ redis-benchmark -h <HOST 1> -p 5001 -t set,get -c 500 -n 1000 -d 16 -P 100 -q
- ● **To kill the cluster and wipe the rdb files associated with the nodes:**
  - ○ cd ~/go/src/rabia/redis-raft/multiple && . multiple_kill.sh
- ● **Known issue:**
- ● As mentioned in the paper, Redis-Raft is still in the experimental mode. So you might observe the issue of constant leader election. An example output is as follows: