SOSP 2023 Artifact evaluation

#107 QuePaxa: Escaping the tyranny of timeouts in consensus

In this document, we describe the artefact evaluation instructions for our SOSP paper: "QuePaxa: Escaping the tyranny of timeouts in consensus."

Repositories

- QuePaxa: https://github.com/dedis/quepaxa contains the QuePaxa go-lang implementation, automated scripts to run in AWS, and the compiled binaries of Rabia, EPaxos, Paxos, and Raft we use for the accepted paper.
- EPaxos and Multi-Paxos: https://github.com/dedis/ePaxos-open-loop is an updated version of the go-lang code used in the EPaxos paper (https://github.com/efficient/epaxos). We added two modifications to this code: (1) configurable replica side batching and pipelining and (2) open loop Poisson distribution based client. Main drawbacks of this implementation (and the original code base https://github.com/efficient/epaxos) is; it does not correctly implement failure case scenarios and configurable view timeouts.
- Multi-Paxos and Raft: https://github.com/dedis/paxos-and-raft is our new implementation of Multi-Paxos and Raft that supports replica failure scenarios and configurable view change timeouts and is therefore more directly comparable to our QuePaxa implementation
- Rabia: https://github.com/dedis/rabia-open-loop is a copy of the go-lang code of Rabia (https://github.com/haochenpan/rabia). The only change that we included is the support for Poisson loop based open loop client implementation.

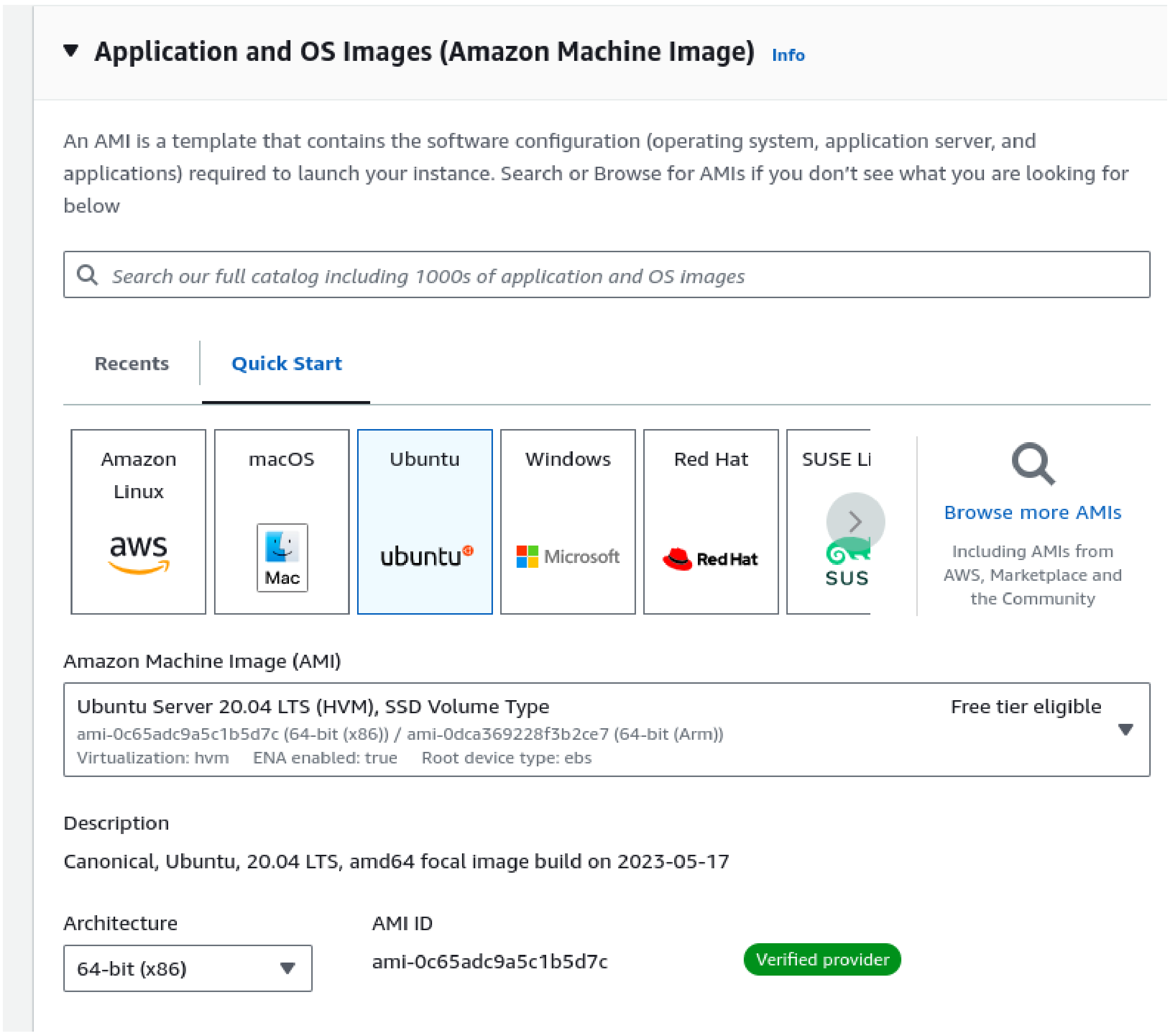
Setup

We use AWS EC2 for replicas and submitters, and a local Ubuntu laptop as the controller. Our automated scripts can be used in any Ubuntu based deployment, however, the patterns and the results may vary due to differences in VM resources, available network bandwidth, and network delay. In this document, we only explain the AWS setup we used, but the explanation is valid for any set of Ubuntu VMs.

VM Setup

Since the specification of VMs we used for each 4 experiments are different from each other, we will describe the machine specifications for each experiment, separately. In the following section, we will describe the requirements that should be satisfied by "all" the VMs used in this evaluation.

Step 1: All the VMs should have 64 bit x86 Canonical, Ubuntu, 20.04 LTS as the operating system. You can choose this image when spawning a new instance in AWS as follows.


▼ Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Amazon Machine Image (AMI)

Ubuntu Server 20.04 LTS (HVM), SSD Volume Type — Free tier eligible
ami-0c65adc9a5c1b5d7c (64-bit (x86)) / ami-0dca369228f3b2ce7 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

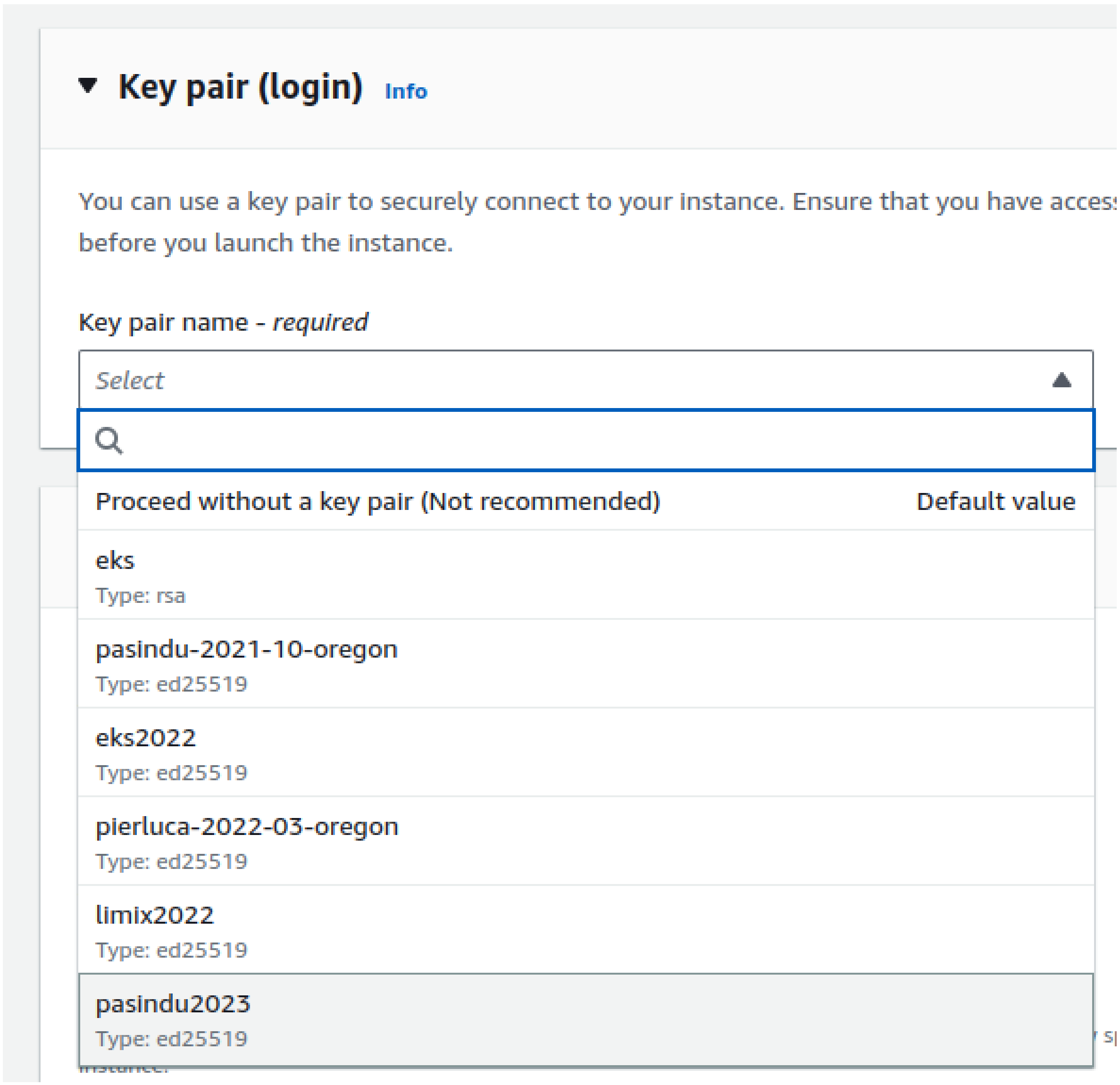
Canonical, Ubuntu, 20.04 LTS, amd64 focal image build on 2023-05-17

Architecture: 64-bit (x86)

AMI ID: ami-0c65adc9a5c1b5d7c

Verified provider

Step 2: All the machines should be configured with the same ssh key. For this, first generate a key pair using ssh-keygen, and then use the private key when instantiating a new VM in AWS. In our scripts we assume the name of the key pair to be pasindu2023, with pasindu2023.pem as the private key.



▼ Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have acces
before you launch the instance.

Key pair name - *required*

Select

Proceed without a key pair (Not recommended) Default value

eks
Type: rsa

pasindu-2021-10-oregon
Type: ed25519

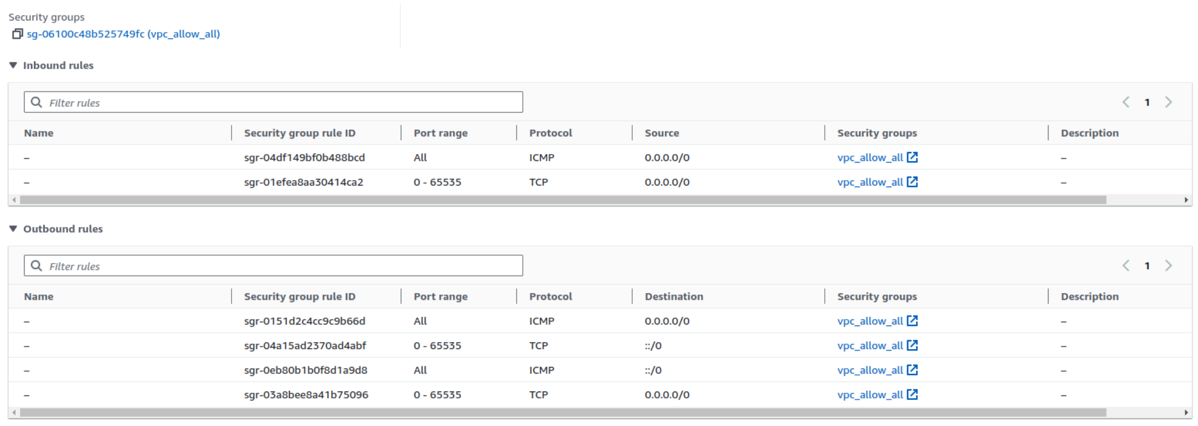
eks2022
Type: ed25519

pierluca-2022-03-oregon
Type: ed25519

limix2022
Type: ed25519

pasindu2023
Type: ed25519

Step 3: We assume that all the replicas accept TCP traffic from all other replicas. In AWS, you can configure this by changing the Security group settings in the security tab. A correctly configured VM's security tab should show the inbound and outbound rules as follows. Allowing TCP traffic from all other replicas and clients in the experiment is important for the correctness of the tests, because we assume an all-to-all connected network.


Security groups
sg-06100c48b525749fc (vpc_allow_all)

Inbound rules

Name	Security group rule ID	Port range	Protocol	Source	Security groups	Description
–	sgr-04df149bf0b488bcd	All	ICMP	0.0.0.0/0	vpc_allow_all	–
–	sgr-01efea8aa30414ca2	0 - 65535	TCP	0.0.0.0/0	vpc_allow_all	–

Outbound rules

Name	Security group rule ID	Port range	Protocol	Destination	Security groups	Description
–	sgr-0151d2c4cc9c9b66d	All	ICMP	0.0.0.0/0	vpc_allow_all	–
–	sgr-04a15ad2370ad4abf	0 - 65535	TCP	::/0	vpc_allow_all	–
–	sgr-0eb80b1b0f8d1a9d8	All	ICMP	::/0	vpc_allow_all	–
–	sgr-03a8bee8a41b75096	0 - 65535	TCP	0.0.0.0/0	vpc_allow_all	–

Setup Controller

Controller is the local laptop that orchestrates each experiment. Controller directs the AWS VMs to run replicas and submitters, collect summary files and calculate stats and graphs.

As the controller of the experiments, we need a separate machine with at least 4 cores, 8GB memory and 20GB HDD. The controller machine should have Ubuntu 20.04 LTS x86 64bit with sudo access.

Step 1: install python pip3, matplotlib, sshpass and go 1.19.5

```
sudo apt update
sudo apt install python3-pip
pip3 install matplotlib
rm -rf /usr/local/go
wget https://go.dev/dl/go1.19.5.linux-amd64.tar.gz
sudo tar -C /usr/local -xzf go1.19.5.linux-amd64.tar.gz
export PATH=$PATH:/usr/local/go/bin
sudo apt install sshpass
```

Step 2: clone the Quepaxa repository, build and run integration tests

```
git clone https://github.com/dedis/quepaxa
cd quepaxa
/bin/bash build.sh
python3 integration-test/python/integration-automation.py
```

The log files in the logs/ directory contain the output of each integration test (ReadMe.md of https://github.com/dedis/quepaxa contains detailed explanations)

Step 3: Copy the private key "pasindu2023.pem" that can access all the VMs to a new directory named "private_key_aws" inside quepaxa/experiments directory of the controller.

```
mkdir -p quepaxa/experiments/private_key_aws
```

```
cp pasindu2023.pem quepaxa/experiments/private_key_aws/pasindu2023.pem
chmod 400 experiments/private_key_aws/pasindu2023.pem
```

Step 4: Double check that the controller can reach all the VMs in each experiment by running ssh on the controller to all VMs.

```
ssh -i pasindu2023.pem ubuntu@vm-dns.com # run for each VM
```

AWS Cost

We have automated the scripts in quepaxa/experiments directory, and each experiment can be run without any manual intervention. We have found the total cost of AWS for the evaluation to be about 1500 CHF (swiss francs), and the total test time is more than 48h, when run in the sequential order. Hence, for each experiment, we first explain how to run the automated script. Then, for each experiment, we describe how to manually run only for specific data points of interest, so that the reader can verify the correctness of our evaluation only for the data points of interest.

Experiment 1

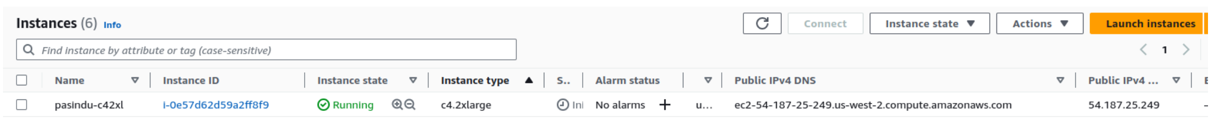
This experiment evaluates the performance of QuePaxa under normal-case network scenarios, under a local-area-network (LAN) and a wide-area-network (WAN).

LAN: Figure 6 (a) and (b)

Setup

VM Specification: We need five c4.4xlarge instances as replicas and five c4.2xlarge instances as clients in this experiment, which are located in the North Virginia AWS site.

Update lines 6,9,12,15,18,21,24,27,30,33, and 40-50 of experiments/setup-5/ip.sh by specifying the public DNS and ip address of each replica and submitter. You can find the public DNS and ip of each replica in the AWS console.



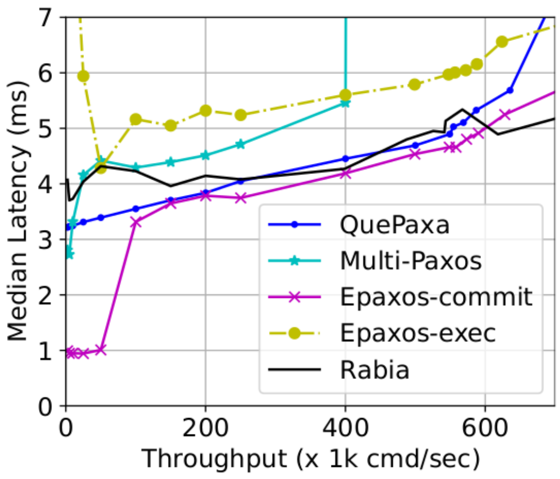
Name	Instance ID	Instance state	Instance type	S..	Alarm status		Public IPv4 DNS	Public IPv4 ...
pasindu-c42xl	i-0e57d62d59a2ff8f9	Running	c4.2xlarge	Ini	No alarms	u...	ec2-54-187-25-249.us-west-2.compute.amazonaws.com	54.187.25.249

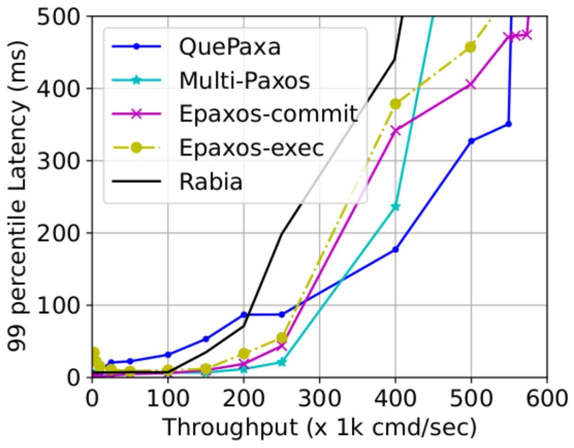
Automated Run

Run the automated script in the Quepaxa directory: /bin/bash experiments/best-case/best-case.sh LAN

Test Time: 18 hours

Test Output: throughput_medianLAN.pdf and throughput_tailLAN.pdf in experiments/best-case/logs/ directory





Manual Run

If you would like to confirm the reproducibility of results for a "selected set of data points", then you can run the bash scripts individually for each arrival rate.

In the experiments/best-case/test-automation.py lines 37-78, you can find the examples of how to run each bash script for a specific arrival rate.

To run QuePaxa for a specific arrival rate x, run the following command

```
/bin/bash experiments/best-case/quepaxa.sh x 2000 4000 LAN 1 1 0
```

You can find the client output logs in experiments/best-case/logs/ folder. Refer the client logs 21-25.log to check the median latency, throughput, 99% latency and error rate statistics.

The following table 1 summarises QuePaxa results for different arrival rates.

arrivalRate	throughput cmd/sec	median µs	99% µs
2500	2515	3216	6266
5000	5035	3233	5835
10000	10047	3244	10449
25000	25105	3312	20836
50000	50127	3392	22469
100000	100117	3550	31532
150000	150095	3705	53312
200000	200134	3839	86987
250000	250058	4049	87156
400000	400100	4453	177118
500000	500058	4693	327252
550000	549159	4896	350761

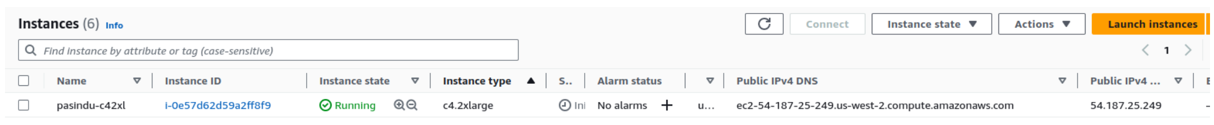
Table 1: QuePaxa LAN performance

WAN: Figure 6 (c) and (d)

Setup

VM Specification: We need five c4.4xlarge instances as replicas and five c4.2xlarge instances as submitters in this experiment, which are located in the AWS regions Tokyo, Mumbai, Singapore, Ireland, and São Paulo. For each AWS site there is one c4.4xlarge VM as the replica and one c4.2xlarge VM as the client.

Update lines 6,9,12,15,18,21,24,27,30,33, and 40-50 of experiments/setup-5/ip.sh by specifying the public DNS and ip address of each replica and submitter. You can find the public DNS and ip of each replica in the AWS console.

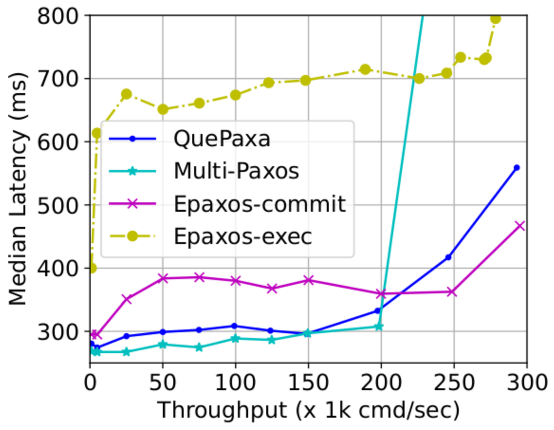


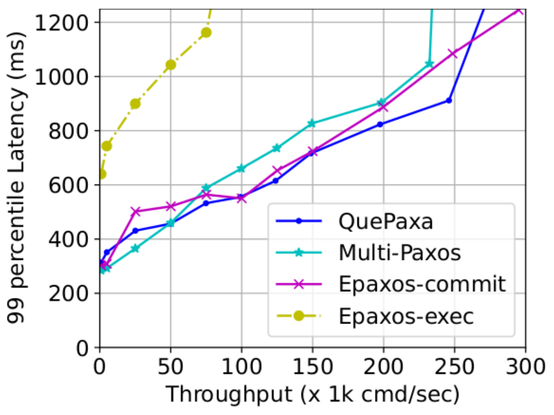
Name	Instance ID	Instance state	Instance type	S..	Alarm status		Public IPv4 DNS	Public IPv4 ...
pasindu-c42xl	i-0e57d62d59a2ff8f9	Running	c4.2xlarge	Ini	No alarms	u...	ec2-54-187-25-249.us-west-2.compute.amazonaws.com	54.187.25.249

Automated Run

Run the automated script in the quepaxa directory: /bin/bash experiments/best-case/best-case.sh WAN

Test Time: 18 hours
Test Output: throughput_medianWAN.pdf and throughput_tailWAN.pdf in the experiments/best-case/logs/ directory





Manual Run

If you would like to confirm the reproducibility of results for a “selected set of data points”, then you can run the bash scripts individually for each arrival rate.

In the experiments/best-case/test-automation.py lines 37-78, you can find the examples of how to run each bash script for a specified arrival rate.

To run QuePaxa for a specific arrival rate x, run the following command

```
/bin/bash experiments/best-case/quepaxa.sh x 3000 5000 WAN 1 0 0
```

You can find the client output logs in experiments/best-case/logs/ folder. Refer the client logs 21-25.log to check the median latency, throughput, 99% latency and error rate statistics.

The following table 2 summarises QuePaxa results for different arrival rates.

arrivalRate	**throughput cmd/sec**	**median µs**	**99% µs**
1000	1010	280611	313340
5000	5035	274311	351531
25000	25087	292343	430956
50000	50028	298959	456972
75000	74831	302219	532357
100000	99069	308509	725493
125000	123860	301071	715070
150000	148761	296083	736288

200000	197454	332487	823242

Table 2: QuePaxa WAN performance

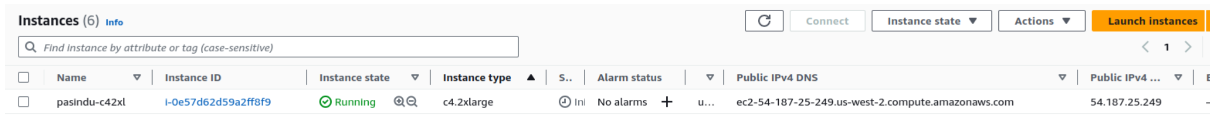
Experiment 2

This experiment evaluates the impact of network asynchrony for QuePaxa performance under a WAN deployment.

Setup

VM Specification: We need five c4.4xlarge instances as replicas and five c4.2xlarge instances as clients in this experiment, which are located in the AWS regions Tokyo, Mumbai, Singapore, Ireland, and São Paulo. Each AWS site should contain one c4.4xlarge instance and one c4.2xlarge instance.

Update lines 6,9,12,15,18,21,24,27,30,33, and 40-50 of experiments/setup-5/ip.sh by specifying the public DNS and ip address of each replica and submitter. You can find the public DNS and ip of each replica in the AWS console.



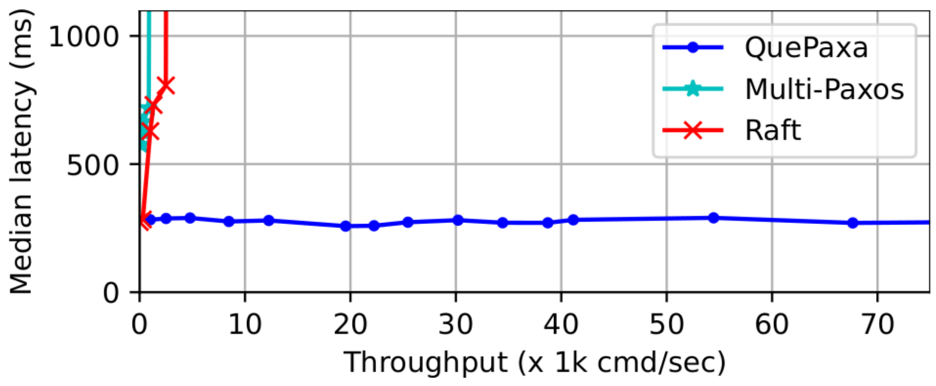
	Name	Instance ID	Instance state	Instance type	S..	Alarm status		Public IPv4 DNS	Public IPv4 ...
	pasindu-c42xl	i-0e57d62d59a2ff8f9	Running	c4.2xlarge	Ini	No alarms	u...	ec2-54-187-25-249.us-west-2.compute.amazonaws.com	54.187.25.249

Automated Run

Run the automated script in the quepaxa directory: /bin/bash experiments/asynchrony/asynchrony.sh

Test Time: 5 hours

Test Output: asychrony.pdf in experiments/asynchrony/logs/ directory



Manual Run

If you would like to confirm the reproducibility of results for a "selected set of data points", then you can run the bash scripts individually for each arrival rate.

In the experiments/asynchrony/test_automation.py lines 18-25, you can find the examples of how to run each bash script for a specific arrival rate.

To run QuePaxa for a specified arrival rate x, run the following command

```
/bin/bash experiments/asychrony/quepaxa.sh x 1
```

You can find the client output logs in experiments/asynchrony/logs/ folder. Refer the client logs 21-25.log to check the median latency, throughput, 99% latency and error rate statistics.

The following table 3 summarises QuePaxa results for different arrival rates.

arrivalRate	**throughput cmd/sec**	**median latency µs**
1000	1010	282727
2500	2508	287623
5000	4789	289597
10000	8470	275936
15000	12254	279911
25000	19542	257743
30000	22236	259339
35000	25453	273002
40000	30210	281089
45000	34437	270971
50000	38725	270736
60000	41143	282348
75000	54452	290138

Table 3: QuePaxa asynchronous performance

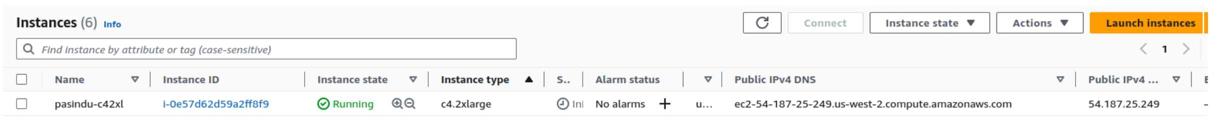
Experiment 3

This experiment evaluates the impact of hedging-delay for QuePaxa performance and recovery time in a WAN deployment.

Setup

VM Specification: We need five c4.4xlarge instances as replicas and five c4.2xlarge instances as clients in this experiment, which are located in the AWS regions Tokyo, Mumbai, Singapore, Ireland, and São Paulo. Each AWS site should contain one c4.4xlarge instance and one c4.2xlarge instance.

Update lines 6,9,12,15,18,21,24,27,30,33, and 40-50 of experiments/setup-5/ip.sh by specifying the public DNS and ip address of each replica and submitter. You can find the public DNS and ip of each replica in the AWS console.



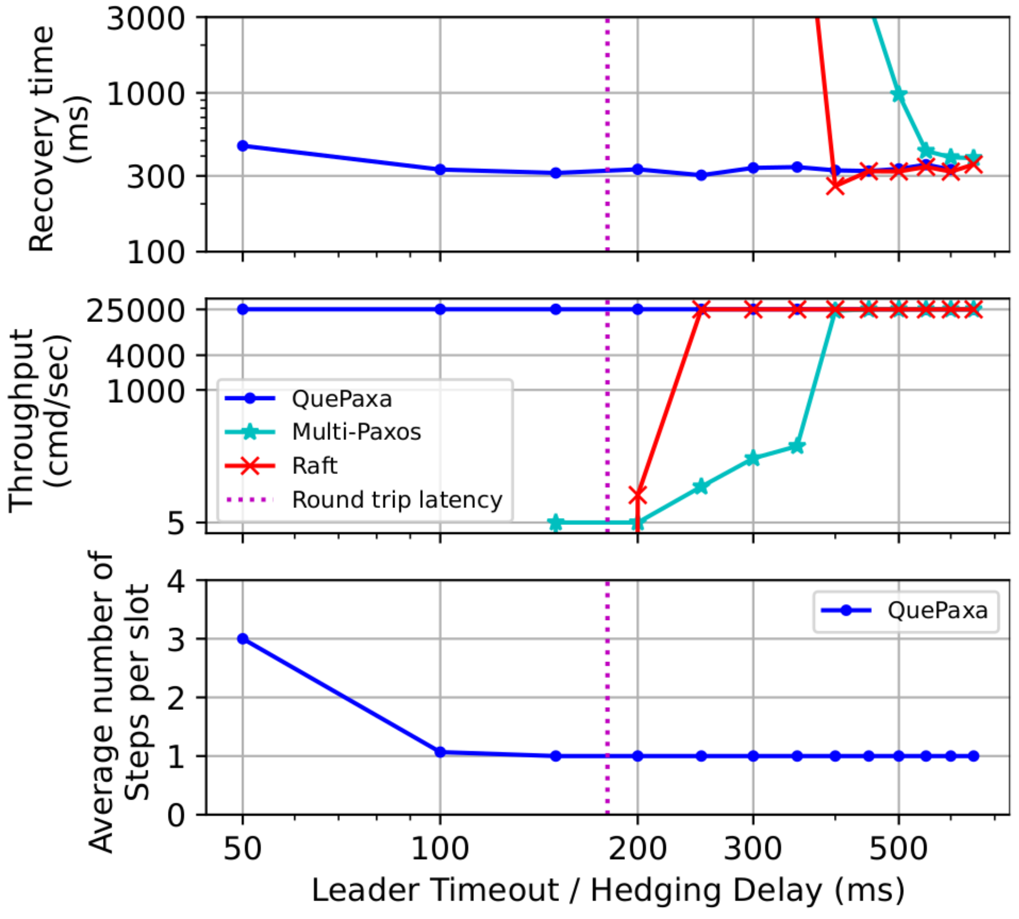
Name	Instance ID	Instance state	Instance type	S..	Alarm status		Public IPv4 DNS	Public IPv4 ...
pasindu-c42xl	i-0e57d62d59a2ff8f9	Running	c4.2xlarge	Ini	No alarms	u...	ec2-54-187-25-249.us-west-2.compute.amazonaws.com	54.187.25.249

Automated Run

Run the automated script in the quepaxa directory:
/bin/bash experiments/leader-timeout/leader-timeout.sh

Test Time: 15 hours for AWS, and 4-6 hours for generating the graphs in the controller (computationally heavy)

Test Output: steps.pdf, timeout_performance.pdf, and timeout_recovery.pdf in experiments/leader-timeout/logs/ directory.



Notes: To reproduce the results in our paper; it is important that you run this experiment in AWS, in the exact AWS locations we have mentioned. If you run the experiment in a different setup with different network delays, then the graphs will vary significantly, although the patterns should be the same.

Manual Run

If you would like to confirm the reproducibility of results for a "selected set of data points", then you can run the bash scripts individually for each arrival rate. For information about how to run for selected data points, please refer to experiments/leader-timeout/test_automation.py lines 20-30.

The statistics calculation for this experiment in experiments/leader-timeout/performance-summary.py , experiments/leader-timeout/recovery-summary.py , and experiments/leader-timeout/slot-summary.py are quite complex and need major revisions and simplifications to run for individual arrival rates. Hence we recommend running the entire experiment as outlined above.

The following table 4 summarises QuePaxa results for different hedging-delays.

Timeout µs	**Throughput cmd/sec**	**Recovery Time ms**	**Average number of steps per slot**
50000	25104	464	3.01
100000	25098	329	1.02
150000	25095	313	1
200000	25101	330	1
250000	25104	303	1
300000	25096	337	1
350000	25097	341	1
400000	25095	225	1
450000	25099	322	1
500000	25100	332	1
550000	25100	453	1
600000	25095	429	1
650000	25104	430	1

Table 4: QuePaxa hedging-performance

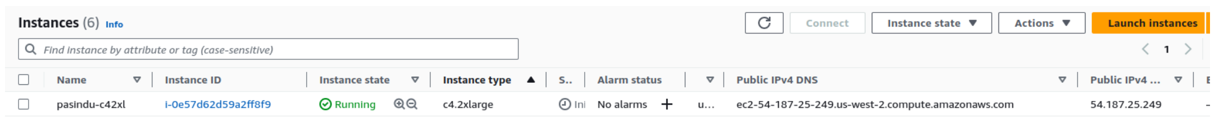
Experiment 4

This experiment evaluates QuePaxa's auto-tuning mechanism to converge on the best hedging schedule.

Setup

VM Specification: This experiment uses five replicas in a single AWS region (Oregon), on five heterogeneous EC2 machines (t2.large, t2.2xlarge, c4.large,c4.xlarge, and c4.4xlarge). For the submitter we use a single c4.2xlarge VM in the same AWS region.

Update lines 6,9,12,15,18,21,24,27,30,33, and 40-50 of experiments/setup-5/ip.sh by specifying the public DNS and ip address of each replica and submitter. You can find the public DNS and ip of each replica in the AWS console. Since this experiment has only 1 client, use the same client DNS and IP for all clients 1-5 in ip.sh



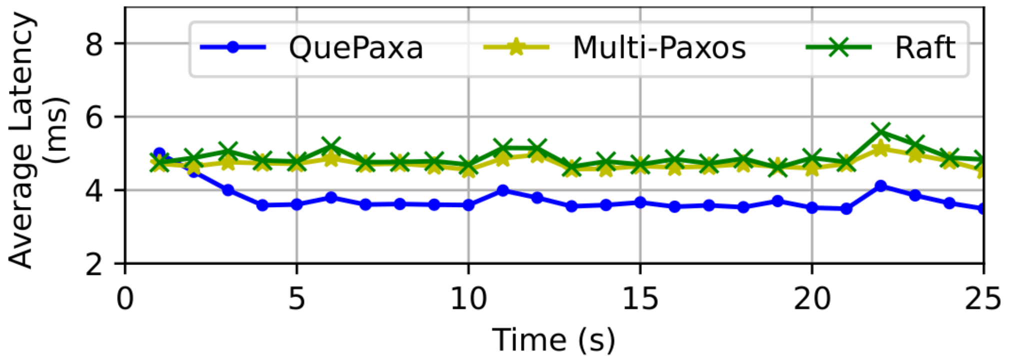
Name	Instance ID	Instance state	Instance type	S..	Alarm status		Public IPv4 DNS	Public IPv4 ...
pasindu-c42xl	i-0e57d62d59a2ff8f9	Running	c4.2xlarge	Ini	No alarms	u...	ec2-54-187-25-249.us-west-2.compute.amazonaws.com	54.187.25.249

Automated Run

Run the automated script in the quepaxa directory:
/bin/bash experiments/tuning/tuning.sh

Test Time: 2 hours

Test Output: time_latency.pdf in experiments/tuning/logs/ directory



Notes: To reproduce the results in our paper; it is important that you run this experiment in AWS, with the exact AWS VMs we have mentioned. If you run the experiment in a different setup with VM specs, then the graphs will vary significantly, although the patterns should be the same.

Possible Problems

The QuePaxa protocol and the implementation (and also Rabia, Paxos, Raft, and EPaxos) assume that the links are reliable, for the entire run of the protocol. However, due to transient errors in the ssh based setup, it is possible that some experiments fail due to some network level error / ssh error / similar random problems. In that case, the automated scripts will abort in the controller in the graph generation phase (but after running all the experiments on AWS), and print to the console the file in which an error was detected.

In the presence of such an error, you can either rerun the entire experiment from scratch (which will double the cost), or manually run the failed experiment just once with the same set of parameters.