Benchmarking Blockchain Frameworks

Guilhem Dubois
Polytechnique Montreal
Montreal, Canada
1897198

Matyas Jesina
Polytechnique Montreal
Montreal, Canada
2165646

Chun-An Bau
Polytechnique Montreal
Montreal, Canada
2165883

***Abstract*—Storing data safely and efficiently is a crucial issue with technological advancements. Blockchain's decentralized and immutable features make it one of the most proper solutions. Blockchain is one of the most popular technologies nowadays and there are numerous blockchain techniques. Each technique has disparate features like advantages, drawbacks, and appropriate usages. We introduced two approaches, Ethereum and Hyperledger Fabric, including their unique characteristics or implementation evolution. Next, we designed a series of experiments, evaluating their difference, such as latency and throughput, in various workloads by benchmark tool BlockBench. Finally, we made a thorough analysis and discussion based on the experiment result.**

***Index Terms*—Blockchain, Ethereum, Hyperledger Fabric, BlockBench**

I. Introduction

Blockchain is catching people's eye and becoming one of the most popular recent technology. Plenty of applications based on blockchain were developed, such as Bitcoin, Namecoin, and Ripple. Its implementation feature increases the immutability and reliability as well as reduces the storage costs. As illustrated in Fig 1, the basic unit of blockchains is a block, which contains information like the transaction data, the timestamp, and the cryptographic hash of the previous block. A blockchain is essentially a growing list of blocks, which can only be extended but not edited.

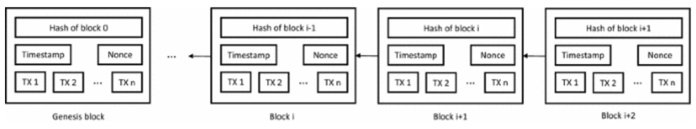


Fig. 1. Simple blockchain architecture [1]

Several characteristics made it such powerful [2] [3]:

- Decentralized: It means that there's no central controller in the system and the decisions are made by the all the participants. Decentralized is one of the most well-known features to achieve greater and fairer service.
- Immutability: As mentioned above, the information inside the blocks cannot be modified nor deleted after creation, which prevented the information from being compromised or tampered.
- Tokenization: Tokenization is the process where the value of an asset being converted into a digital token. After that, the digital tokens are shared via blockchain, and can be used to smooth the business transaction procedures.
- Reduced costs: Blockchain creates an efficient processing transaction, which significantly reduces the manual tasks such as aggregating and amending data.
- Speed: Since there are no intermediaries in blockchains, the transactions can be handled faster than those traditional approaches.

Two major types of blockchains are public and private blockchains, their difference [4] is shown in Table I.

TABLE I
Public vs. Private Blockchain

	Public	**Private**
Access	Anyone	Single organization
Authority	Decentralized	Partially decentralized
Transaction Speed	Slow	Fast
Transaction Cost	High	Low
Immutability	Full	Partial
Efficiency	Low	High

We evaluated two blockchains, Ethereum and Hyperledger Fabric, by leveraging a benchmark tool, BlockBench. We will explain the context, go over the tools setup, then show and analyse our results. Our experiments included multiple dimensions, such as the assessment of system stability and performance. Based on the results, we found that they were both feasible to use for several purposes.

II. Blockchains

Our work involved two blockchains, Ethereum and Hyperledger Fabric. In this section, we made a briefly introduction of the approaches, and compared their difference.

A. Ethereum

Ethereum is both a public and private blockchain which launched in 2015 [5]. The well-known cryptocurrency ETH is powered by Ethereum blockchain. It first used the Proof-of-Work (PoW) based consensus protocol. However, the technique was not secure enough and might suffer from the selfish mining attack, which means that an individual in a mining pool attempts to withhold a successfully validated block from being broadcast to the rest of the mining pool network [6]. As a result, the developers worked on a new clique named Proof-of-Authority (PoA). In the new approach, the blocks would be sealed by approved signers, which reduced the burden of mining. Moreover, the signers can be added or removed dynamically if the actions are approved by the majority of other signers.

B. *Hyperledger Fabric*

Hyperledger Fabric [7], a modular architecture supporting several implementations, [8] is an open source private blockchain developed by IBM and Linux Foundation. Hyperledger Fabric conquered several drawbacks of traditional blockchains, such as order-executive architecture and hard-coded consensus problems, providing better throughput and lower latency. Multiple features are supported by the current release:

- Execute arbitrary smart contracts implemented in Go
- Pluggable consensus protocol
- Certificate authorities for TLS certificates
- Persistent state using a key-value store interface
- Basic REST APIs and CLIs

Hyperledger Fabric was developed for private use in corporations to provide a secure and immutable data and transaction record. Access can be restricted to individual users to prevent information leaks. Authorized users send their transaction proposal to peers, which maintain the whole blockchain history and either approve or reject the transaction. If the transaction proposal is approved, it can then be sent to an ordering service, which adds this transaction to the next block and distributes this information across all peers. A simplified overview of this process can be seen in Fig. 2.

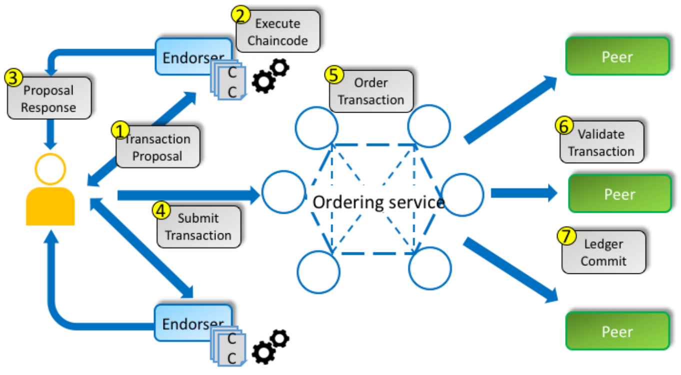


Fig. 2. Overview of a Hyperledger Fabric transaction process [13]

Since this consensus method doesn't rely on proof of work, it can achieve better speeds and generally better predictability compared to Ethereum blockchain.

C. *Comparison*

The major differences between the two blockchains [9] [10] were shown in the Table II.

TABLE II
ETHEREUM VS. HYPERLEDGER FABRIC

	Ethereum	**Hyperledger Fabric**
Purpose	B2C businesses	B2B businesses
Consensus	Mining is required	No mining is required
Cryptocurrency	Ether	None
Confidentiality	Transparent	Confidential transactions

III. METHODOLOGY

A. *Tools Used*

In order to run the blockchains locally and run the benchmarking tool, we used a virtual machine from Amazon EC2, a service to access VMs running on the cloud. This lets us have a machine for which we know the exact specifications for, and so that all members on the team can run the experiments for those same specifications instead of using our own machines.

In Amazon EC2, we used an instance called t2.large which has 2 vCPUs, 8.0 GiB of RAM and 25.0 GiB of storage [11]. The 2 vCPUs give us access to 2 processing threads, but we only use one for our benchmarking tool as we'll see later on. The 25.0 GiB of storage is important since we are running the blockchains locally, so all nodes need to be contained in the VM. We could use ssh from a terminal on our local machine to run commands on the VM.

Once we have our VM set up, we cloned the BlockBench repository and installed some dependencies that were required to run the experiments:

- Docker (and docker compose)
- nodeJS (version 10.22.0)
- python (and python3-pip, libssl-dev, libffi-dev, python3-dev, python3-venv)
- Hyperledger Fabric binaries
- Golang (version 1.17)
- c++ librairies (build-essential, libtool, autoconf, libcurl-gnutls-dev)
- restclient-cpp

To install Hyperledger Fabric, we also needed to add the necessary path variables [12]. Ethereum clique, along with some other dependencies, were available directly in the BlockBench repository. Instances from both blockchains could then be started from the BlockBench directory, running 5 miners (also called peers) responsible to create new blocks.

B. *Workload Configuration*

We start the experiment by running the benchmarking tool and specifying some options. Some important options are the target blockchain (Ethereum / Hyperledger), the workload configuration (which will be explained after) and the transaction rate (in our case we send 50 transactions per second).

Each workload configuration is defined in their respective file. In it, we set:

- Record Count: Number of total requests sent
- Operation Count: Number of operations performed per transaction
- Workload: The benchmark specification type
- Read Proportion: The proportion of requests that read data from the blockchain
- Update Proportion: The proportion of requests that writes data in the blockchain

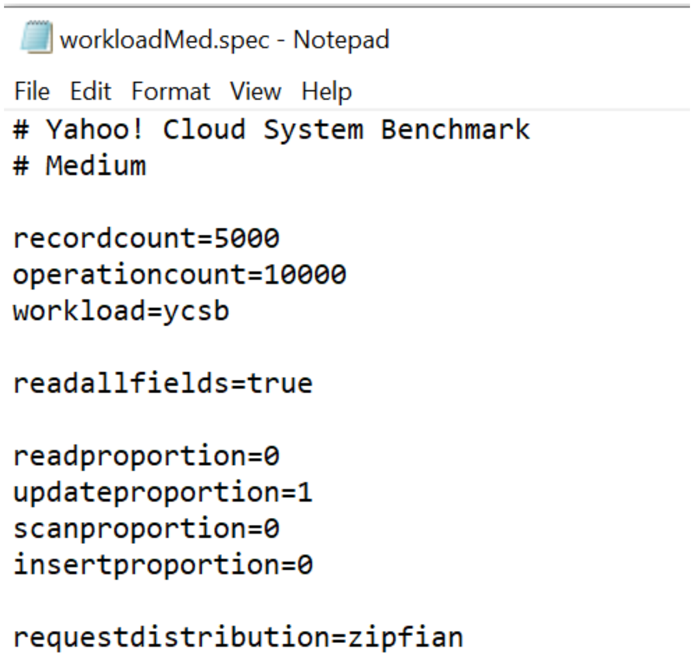
Some other properties can be left at their default value, such as the request distribution, field count, field length, etc. We tried a lot of different configurations and ended up with three workloads that gave us interesting results. We have a small, medium and large workload. The property that changes between the three is the record count which is 2,500 for the small workload, 5,000 for the medium one and 10,000 for the large one. We chose those values as the small workload ran well with both blockchains, while the large one had important performance issues with Ethereum. We then chose the medium one as a value in between the two others (twice more requests than the small one, but half of the large one).

The other properties were the same between all workloads. We have 10,000 operations per transaction. The workload type was Yahoo! Cloud Serving Benchmark (YCSB), which is a macro-benchmark specification to test transactions as storage of key-value pairs.

Macro-benchmarks test the blockchain as a whole by focusing on the application layer, which is similar to real-case uses of the blockchains.

Micro-benchmarks would be for testing a specific lower layer, such as the consensus, data or execution layer by itself. There are also other macro-benchmark specifications available in BlockBench, such as SmallBank for transactions based on money transfer.

We also used a read proportion of 0 and an update proportion of 1, which is equivalent to only writing in the blockchain. We do this so that we are testing the performance of the consensus protocol, which is the interesting part about those blockchains, as the read operations are similar to traditional databases. The complete configuration used for the medium workload can be seen in Fig. 3.



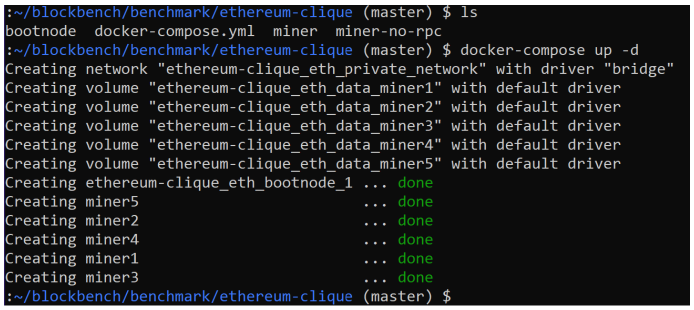
```
# Yahoo! Cloud System Benchmark
# Medium

recordcount=5000
operationcount=10000
workload=ycsb

readallfields=true

readproportion=0
updateproportion=1
scanproportion=0
insertproportion=0

requestdistribution=zipfian
```

Fig. 3. Medium workload configuration

For each of the three workloads, and for each of the two blockchains, we run BlockBench and get results as a text file.

C. Running BlockBench

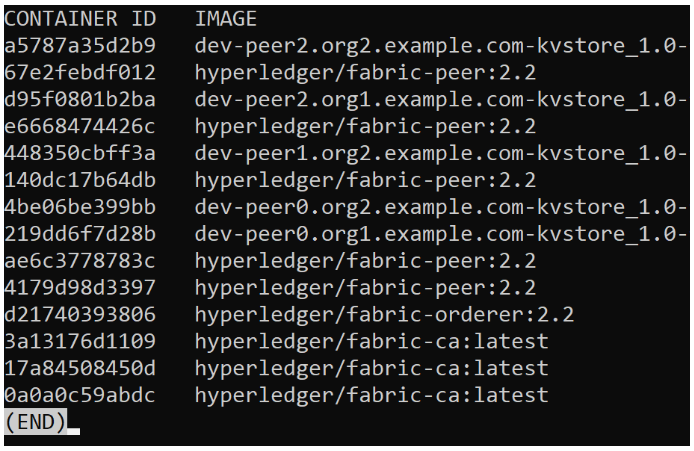
Once all the tools and workloads are set up, we run the local blockchain to be tested. For Ethereum, the BlockBench repository contains a file called ”docker-compose.yaml” which we can run in the background using the command ”docker-compose up -d”. The output is shown in Fig. 4, where we can see it created the network and 5 active miners to create blocks.



```
:~/blockbench/benchmark/ethereum-clique (master) $ ls
bootnode  docker-compose.yml  miner  miner-no-rpc
:~/blockbench/benchmark/ethereum-clique (master) $ docker-compose up -d
Creating network "ethereum-clique_eth_private_network" with driver "bridge"
Creating volume "ethereum-clique_eth_data_miner1" with default driver
Creating volume "ethereum-clique_eth_data_miner2" with default driver
Creating volume "ethereum-clique_eth_data_miner3" with default driver
Creating volume "ethereum-clique_eth_data_miner4" with default driver
Creating volume "ethereum-clique_eth_data_miner5" with default driver
Creating ethereum-clique_eth_bootnode_1 ... done
Creating miner5                        ... done
Creating miner2                        ... done
Creating miner4                        ... done
Creating miner1                        ... done
Creating miner3                        ... done
:~/blockbench/benchmark/ethereum-clique (master) $
```

Fig. 4. Output after starting Ethereum network locally

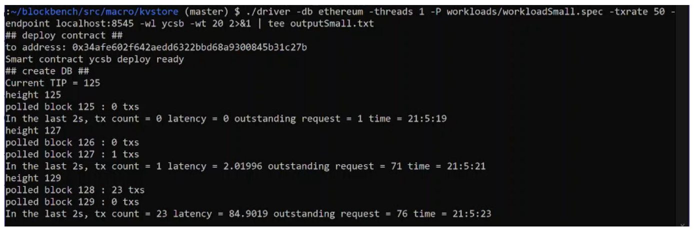
For Hyperledger Fabric, we run a script that automatically starts the network, creates the peers and waits for incoming transactions. It uses docker in the background to run the peers and the orderer (which puts transactions into blocks and sends them to the peers). We can therefore see those peers using ”docker ps” as seen in Fig. 5.



```
CONTAINER ID   IMAGE
a5787a35d2b9   dev-peer2.org2.example.com-kvstore_1.0-
67e2febdf012   hyperledger/fabric-peer:2.2
d95f0801b2ba   dev-peer2.org1.example.com-kvstore_1.0-
e6668474426c   hyperledger/fabric-peer:2.2
448350cbff3a   dev-peer1.org2.example.com-kvstore_1.0-
140dc17b64db   hyperledger/fabric-peer:2.2
4be06be399bb   dev-peer0.org2.example.com-kvstore_1.0-
219dd6f7d28b   dev-peer0.org1.example.com-kvstore_1.0-
ae6c3778783c   hyperledger/fabric-peer:2.2
4179d98d3397   hyperledger/fabric-peer:2.2
d21740393806   hyperledger/fabric-orderer:2.2
3a13176d1109   hyperledger/fabric-ca:latest
17a84508450d   hyperledger/fabric-ca:latest
0a0a0c59abdc   hyperledger/fabric-ca:latest
(END)
```

Fig. 5. Docker images running after starting Hyperledger Fabric network locally

Once the blockchain we want to test is running, we can start the benchmark using the ”driver” command from BlockBench, where we specify the number of threads (1), the transaction rate (50 per seconds), the workload (small, medium or large) and the blockchain used (Ethereum or Hyperledger). It then starts sending transactions and querying blocks as we will describe in the results section. We can see the command used when benchmarking Ethereum and its output in Fig. 6.



```
:~/blockbench/src/macro/kvstore (master) $ ./driver -db ethereum -threads 1 -P workloads/workloadSmall.spec -txrate 50 -
endpoint localhost:8545 -wl ycsb -wt 20 2>&1 | tee outputSmall.txt
## deploy contract ##
to address: 0x34afe602f642aedd6322bbd68a9300845b31c27b
Smart contract ycsb deploy ready
## create DB ##
Current TIP = 125
height 125
polled block 125 : 0 txs
In the last 2s, tx count = 0 latency = 0 outstanding request = 1 time = 21:5:19
height 127
polled block 126 : 0 txs
polled block 127 : 1 txs
In the last 2s, tx count = 1 latency = 2.01996 outstanding request = 71 time = 21:5:21
height 129
polled block 128 : 23 txs
polled block 129 : 0 txs
In the last 2s, tx count = 23 latency = 84.9019 outstanding request = 76 time = 21:5:23
```

Fig. 6. Running BlockBench on Ethereum

IV. Results

A. Results Data

There are two main metrics that we'll use to understand the performance of the blockchains:

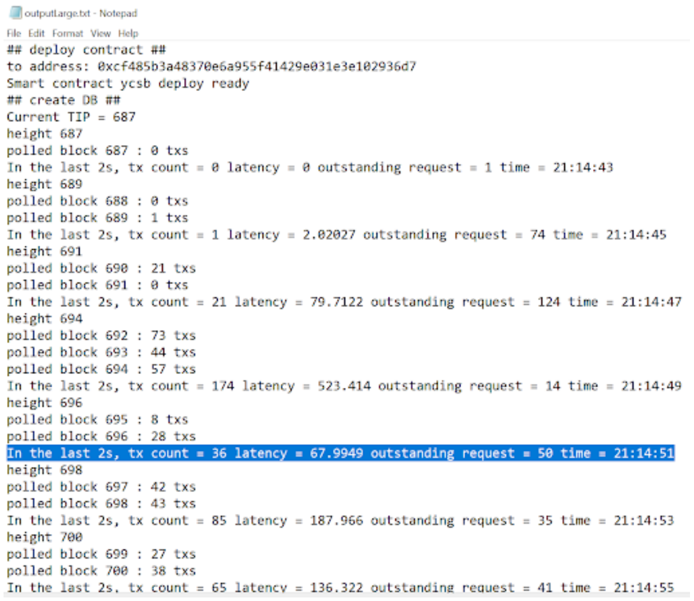
- Latency : The latency represents the response time per transaction. More precisely, it's the time between when the transaction was received by the blockchain and when it was done handling it.
- Throughput : The throughput is the number of successful transactions per second. Instead of representing how much time a transaction takes, it represents how many transactions can be done in an amount of time.

When BlockBench is running, it will send transactions continuously until the number of requests from the workload configuration is reached. At the same time, it will query the new blocks created and output the number of transactions handled by those blocks in a text file. Every two seconds, it will then output some more information:

- tx count : How many transactions have been completed in those past two seconds
- latency : The total latency of those transactions
- outstanding requests : How many outstanding requests are in the queue (pending requests that are sent but waiting to be handled)
- time: The time for each output line (after every 2 seconds)

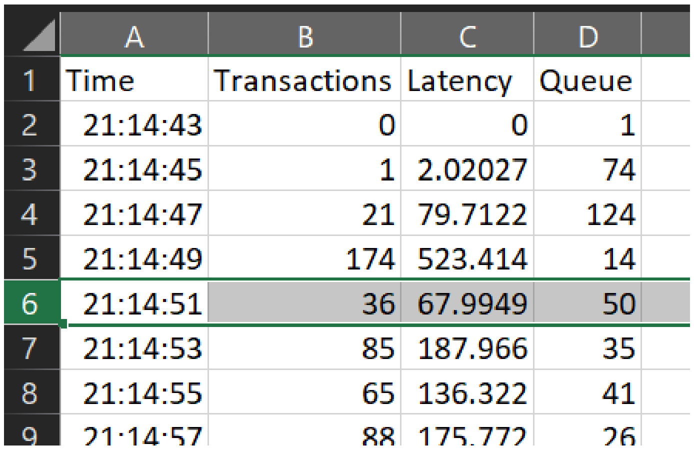
We can see this raw output as a text file in Fig. 7.

Once the experiment is completed, we can use a tool included in the BlockBench repository to format the text file into a comma-separated values (csv) file, as can be seen in Fig. 8.



```
## deploy contract ##
to address: 0xcf485b3a48370e6a955f41429e031e3e102936d7
Smart contract ycsb deploy ready
## create DB ##
Current TIP = 687
height 687
polled block 687 : 0 txs
In the last 2s, tx count = 0 latency = 0 outstanding request = 1 time = 21:14:43
height 689
polled block 688 : 0 txs
polled block 689 : 1 txs
In the last 2s, tx count = 1 latency = 2.02027 outstanding request = 74 time = 21:14:45
height 691
polled block 690 : 21 txs
polled block 691 : 0 txs
In the last 2s, tx count = 21 latency = 79.7122 outstanding request = 124 time = 21:14:47
height 694
polled block 692 : 73 txs
polled block 693 : 44 txs
polled block 694 : 57 txs
In the last 2s, tx count = 174 latency = 523.414 outstanding request = 14 time = 21:14:49
height 696
polled block 695 : 8 txs
polled block 696 : 28 txs
In the last 2s, tx count = 36 latency = 67.9949 outstanding request = 50 time = 21:14:51
height 698
polled block 697 : 42 txs
polled block 698 : 43 txs
In the last 2s, tx count = 85 latency = 187.966 outstanding request = 35 time = 21:14:53
height 700
polled block 699 : 27 txs
polled block 700 : 38 txs
In the last 2s, tx count = 65 latency = 136.322 outstanding request = 41 time = 21:14:55
```

Fig. 7. Raw output from BlockBench

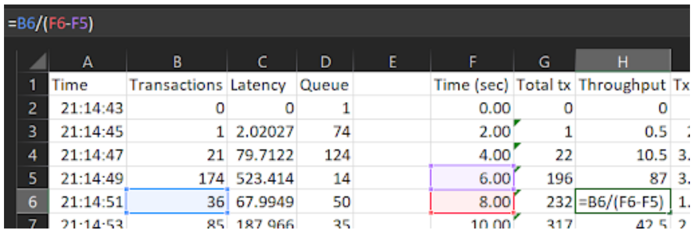


	A	B	C	D
1	Time	Transactions	Latency	Queue
2	21:14:43	0	0	1
3	21:14:45	1	2.02027	74
4	21:14:47	21	79.7122	124
5	21:14:49	174	523.414	14
6	21:14:51	36	67.9949	50
7	21:14:53	85	187.966	35
8	21:14:55	65	136.322	41
9	21:14:57	88	175.772	26

Fig. 8. Formatted output from BlockBench

Using the csv file, we did some data aggregation by adding four fields:

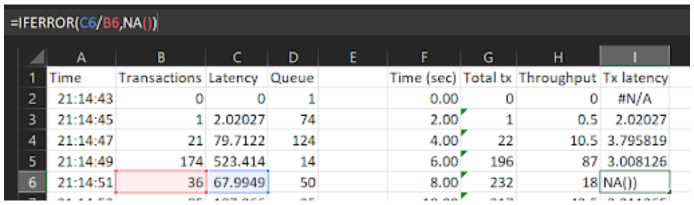
- Time in seconds : Seconds passed since the start of the experiment
- Total transactions completed : Sum of the the transactions completed since the start of the experiment
- Throughput : Number of transactions completed divided by the time passed since the previous query (Fig. 9)
- Latency: Total latency divided by the number of transactions completed (Fig. 10)



=B6/(F6-F5)

	A	B	C	D	E	F	G	H	
1	Time	Transactions	Latency	Queue		Time (sec)	Total tx	Throughput	Tx
2	21:14:43	0	0	1		0.00	0	0	
3	21:14:45	1	2.02027	74		2.00	1	0.5	
4	21:14:47	21	79.7122	124		4.00	22	10.5	3.
5	21:14:49	174	523.414	14		6.00	196	87	3.
6	21:14:51	36	67.9949	50		8.00	232	=B6/(F6-F5)	1.
7	21:14:53	85	187.966	35		10.00	317	42.5	2

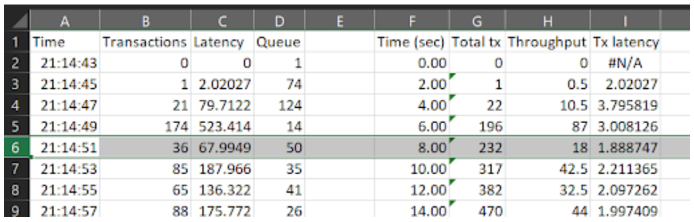
Fig. 9. Calculation of the throughput using other fields



=IFERROR(C6/B6,NA())

	A	B	C	D	E	F	G	H	I
1	Time	Transactions	Latency	Queue		Time (sec)	Total tx	Throughput	Tx latency
2	21:14:43	0	0	1		0.00	0	0	#N/A
3	21:14:45	1	2.02027	74		2.00	1	0.5	2.02027
4	21:14:47	21	79.7122	124		4.00	22	10.5	3.795819
5	21:14:49	174	523.414	14		6.00	196	87	3.008126
6	21:14:51	36	67.9949	50		8.00	232	18	NA())

Fig. 10. Calculation of the latency using other fields



	A	B	C	D	E	F	G	H	I
1	Time	Transactions	Latency	Queue		Time (sec)	Total tx	Throughput	Tx latency
2	21:14:43	0	0	1		0.00	0	0	#N/A
3	21:14:45	1	2.02027	74		2.00	1	0.5	2.02027
4	21:14:47	21	79.7122	124		4.00	22	10.5	3.795819
5	21:14:49	174	523.414	14		6.00	196	87	3.008126
6	21:14:51	36	67.9949	50		8.00	232	18	1.888747
7	21:14:53	85	187.966	35		10.00	317	42.5	2.211365
8	21:14:55	65	136.322	41		12.00	382	32.5	2.097262
9	21:14:57	88	175.772	26		14.00	470	44	1.997409

Fig. 11. Result of the data aggregation for each query

Figure 11 shows our new data after doing the data aggregation. Adding those fields now gives us all the information we need to make graphs comparing the two blockchains. The selected row is the same in Fig. 7, 8 and 11 to show how the data changed after the formatting, followed by the data aggregation.

B. Results Visualization

The figures 12, 13, 14 and 15 are representing our results for Ethereum. In each graph, the workloads are separated by colors, as indicated in the legends. In the figures 12 and 14, we show values using their minimum, average and maximum reached during the experiment for each workload. In the figures, 13 and 15, we show how the values changed over time during the experiment.

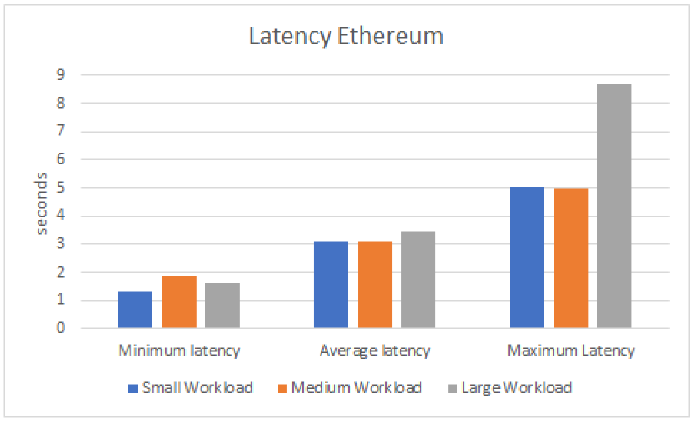


Fig. 12. Minimum, average and maximum latency for each workload (Ethereum).

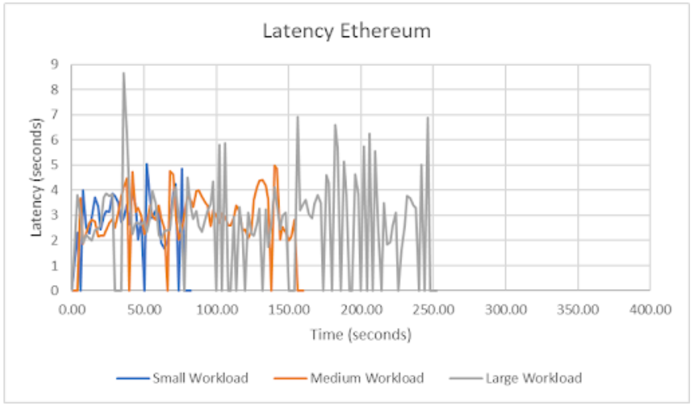


Fig. 13. Latency over time for each workload (Ethereum).

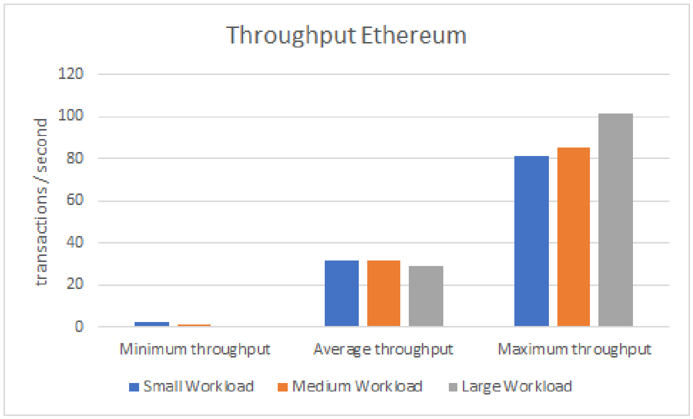


Fig. 14. Minimum, average and maximum throughput for each workload (Ethereum).

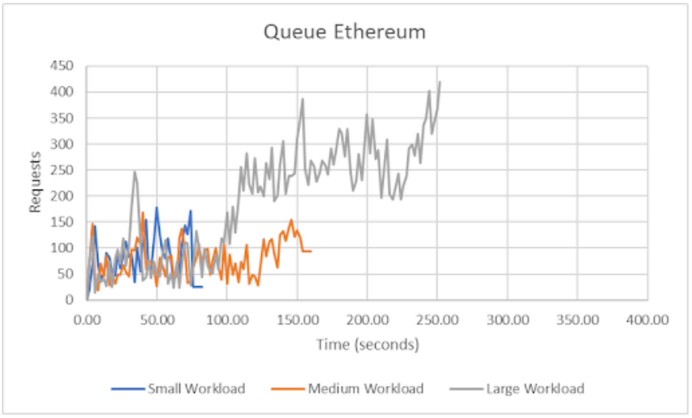


Fig. 15. Number of requests in the queue over time (Ethereum).

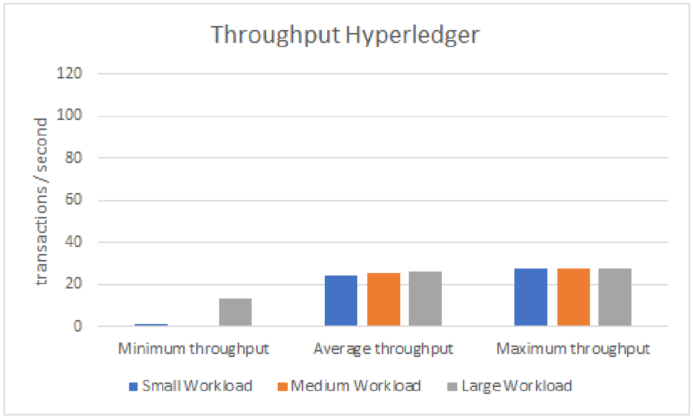


Fig. 18. Minimum, average and maximum throughput for each workload (Hyperledger).

The same graphs format shown previously is used to represent our results for Hyperledger Fabric in the figures 16, 17, 18 and 19. Once again, the figures 16 and 18 show the minimum, average and maximum values while the figures 17 and 19 represent the values over time.

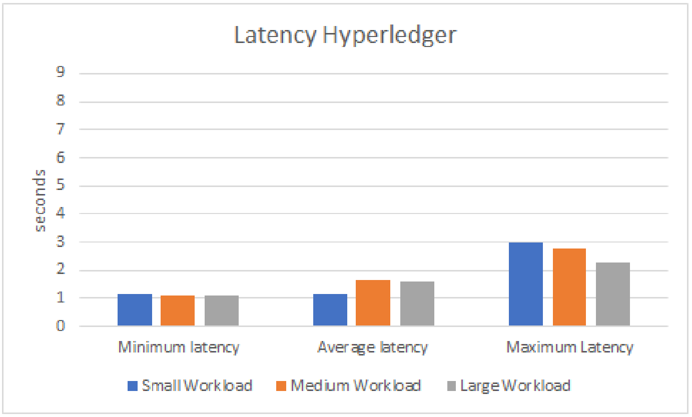


Fig. 16. Minimum, average and maximum latency for each workload (Hyperledger).

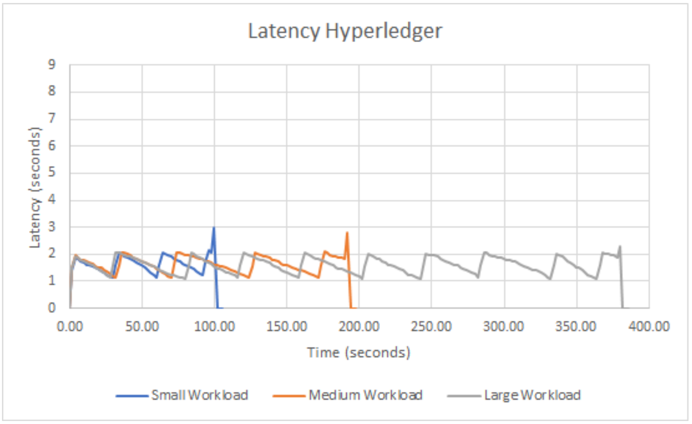


Fig. 17. Latency over time for each workload (Hyperledger).

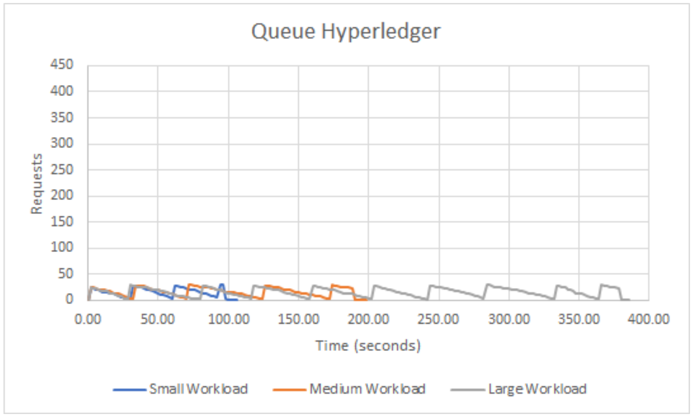


Fig. 19. Number of requests in the queue over time (Hyperledger).

C. Results Analysis

Both blockchain based technologies were measured accordingly and we were able to compare their performance on different data sets. Hyperledger was generally very stable in all parts of the experiment, which is reflected on its individual graphs. Ethereum was marginally faster in processing an equal workload.

Reported latency was relatively low for both technologies, as displayed in Fig. 20. While Hyperledger achieved latencies below 2 seconds throughout the whole experiment, Ethereum's latency was unstable with several spikes up to 5 seconds. We also see those results in the individual latency graphs for each blockchain (Fig. 12 and Fig. 16). The latency for Hyperledger is stable (small difference between minimum and maximum on all workloads) and smaller than the one for Ethereum, which reached almost 9 seconds in the large workload. In both cases, the latency didn't seem that affected by the number of transactions change. We can see in Fig. 13 and Fig. 17 that the latency for the small workload (in blue) stays at similar values as the one for the large workload (in grey).

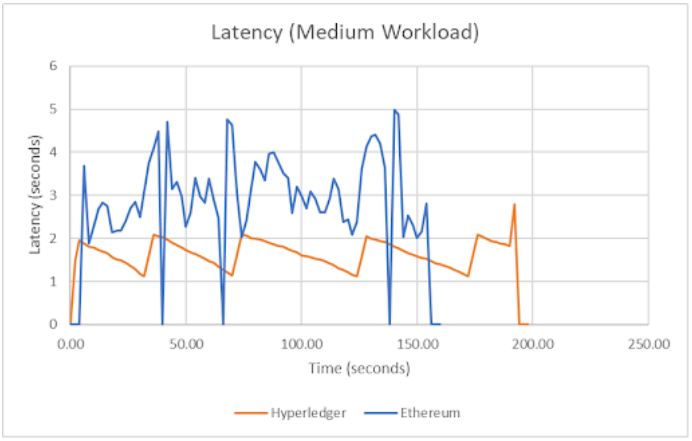


Fig. 20. Comparison of measured latency between Hyperledger (orange) and Ethereum (blue).

Similar situation has been observed during throughput measurements. Ethereum's throughput, although higher on average, was reported as very unstable and unpredictable. This can be seen in Fig. 21. Again, we can see how stable Hyperledger is by looking at the throughput graphs Fig. 18 where the average and maximum throughput is similar for all three workloads in Hyperledger. Compared to Hyperledger, Ethereum had a much bigger difference between the minimum, average and maximum values, with the maximum throughput being about three times the average throughput, as we can see in Fig. 14.

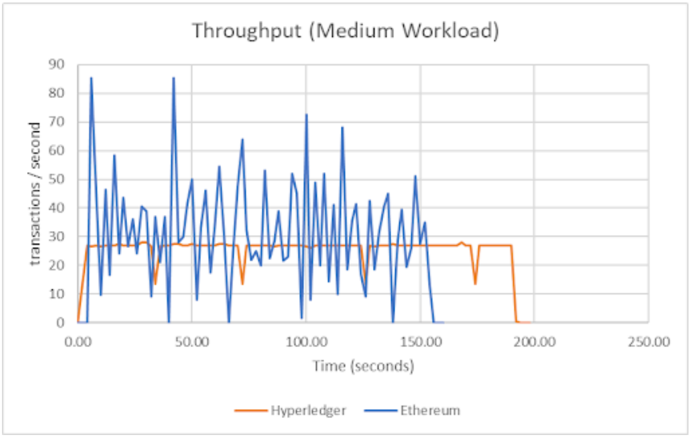


Fig. 21. Comparison of throughput between Hyperledger and Ethereum.

Ethereum's instability is also reflected in the amount of queued transactions. Despite more computing resources being available, Ethereum's queue of unprocessed transactions was growing for medium and large workloads. After all transactions were submitted, we have measured that many of them were not being processed. This can be seen in Fig. 22. After a few minutes, the number of transactions in the queue was not changing while no transactions were being handled and we had to terminate the process manually. We can notice how major that instability became for heavier workloads in Fig. 15 where the queue for the large workload (in grey) kept increasing. After a few minutes, the transactions sent by BlockBench kept coming, but were not being handled by the blockchain, making the queue increasingly bigger over time. For Hyperledger, all the transactions were always handled and we never noticed stuck transactions at the end. We can notice in Fig. 22 how it seems to work by batches where it accumulates some transactions and then handle them repeatedly, compared to the unpredictable behaviour of Ethereum.

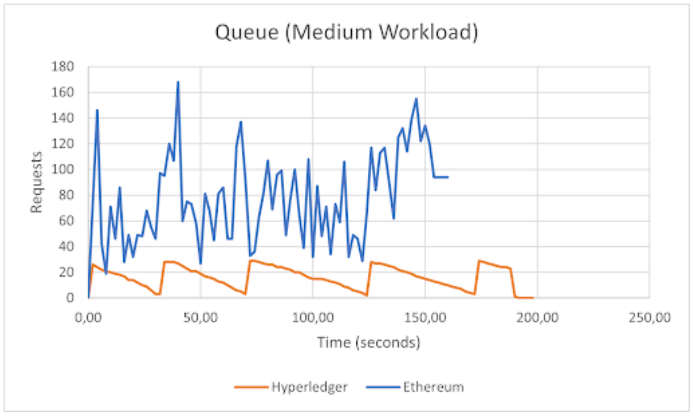


Fig. 22. Comparison of queue size between Hyperledger and Ethereum.

D. *Potential Errors*

During our testing, we have encountered possible issues with Ethereum. According to reported values, not all transactions were being processed and reported metrics showed possible instability. Further configuration and troubleshooting was attempted but we were unable to figure out the exact cause. This issue was more important on higher workloads and less noticeable on lower workloads. We think it could be that some transactions do not get picked up by miners when they are overloaded with transactions, leaving those few stuck and not getting picked up as long as they're not sent again. In fact, we noticed that the CPU was used at over 70% during the large workload (Fig. 23), which shows that the large workload we used really started pushing the virtual machine to its limits and therefore it is less surprising to find errors such as those stuck transactions.

It's also important to mention that our benchmark tests were focused on testing performance in a controlled environment, but are not necessarily representative of real use cases of those blockchains. For example, we were running the blockchain and the peers locally on one machine, while it's often used on a network, and sometimes with a lot more miners/peers. We also focused on repetitive write operations since that helps us test the performance of the consensus protocol, but both write and read operations would probably be used in a real environment.

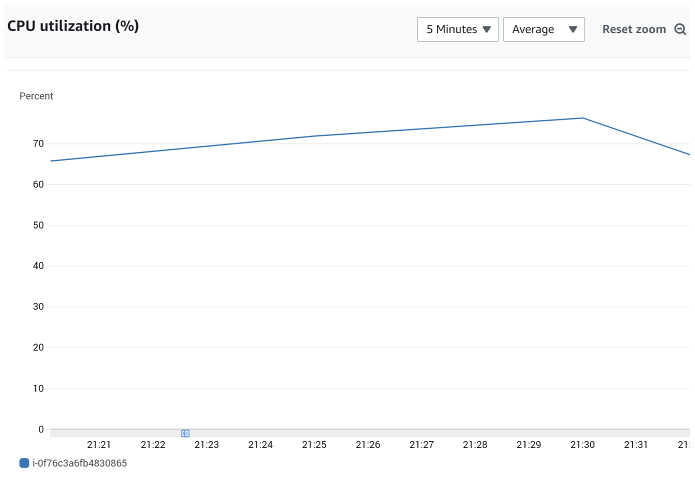


Fig. 23. CPU usage over time on the virtual machine.

V. Conclusion

Both tools achieved similar results in all parts of the experiment. It was clear Hyperledger Fabric was more stable, but it also had lower throughput. On lower workloads, the difference was less noticeable. However, once reaching large number of transactions being handled, Ethereum had a hard time keeping up and some transactions were getting stuck. Hyperledger stayed stable in all workloads, even if it took longer to finish handling all the transactions received. Due to their different purpose, both of them are viable candidates for implementing blockchain-based systems, depending on the needs.

It could be interesting to do more experiments on different workload variables. We focused on the record count, but depending on the blockchain, it could confirm or give us different conclusions when analysing workloads based on the transaction rate (number of requests sent per second) for example. In a future experiment, we could also look into comparing blockchains with traditional databases, since they're both useful alternatives in some contexts.

References

[1] A. Pieroni, N. Scarpato, and L. Felli, *Blockchain and IoT Convergence—A Systematic Survey on Technologies, Protocols and Security*, Applied Sciences, vol. 10, no. 19, p. 6749, Sep. 2020 [Online]. Available: http://dx.doi.org/10.3390/app10196749

[2] M. Nofer et al., *Blockchain*, Bus Inf Syst Eng 59, 183–187 (2017). https://doi.org/10.1007/s12599-017-0467-3

[3] M. Pratt, *Top 10 benefits of blockchain technology for business*, Accessed on: Dec 16, 2021. [Online] Available: https://searchcio.techtarget.com/feature/Top-10-benefits-of-blockchain-technology-for-business

[4] G. Iredale, *Public Vs Private Blockchain: How Do They Differ?*, 101 Blockchains, Accessed on: Dec. 15, 2021. [Online]. Available: https://101blockchains.com/public-vs-private-blockchain/

[5] J. Polge et al., *Permissioned blockchain frameworks in the industry: A comparison*, ICT Express, Volume 7, Issue 2, 2021, Pages 229-233, Available: https://doi.org/10.1016/j.icte.2020.09.002.

[6] D. Sewell, *Selfish mining attack*, Accessed on Dec 16, 2021. [Online] Available: https://golden.com/wiki/Selfish_mining_attack

[7] Cachin et al., *"Architecture of the hyperledger blockchain fabric."*, Workshop on distributed cryptocurrencies and consensus ledgers. Vol. 310. No. 4. 2016.

[8] N. Sehgal, *Hyperledger & Hyperledger Fabric*, Accessed on: Dec. 14, 2021. [Online]. Available: https://medium.com/coinmonks/hyperledger-hyperledger-fabric-53f510a006d

[9] Prerna, *Hyperledger vs Ethereum – Which Blockchain Platform Will Benefit Your Business?*, Accessed on: Dec. 15, 2021. [Online]. Available: https://www.edureka.co/blog/hyperledger-vs-ethereum/

[10] N. Srivastava, *ETHEREUM VS. HYPERLEDGER: A COMPREHENSIVE GUIDE*, Blockchain Council, Accessed on: Dec 14, 2021. [Online]. Available: https://www.blockchain-council.org/ethereum/ethereum-vs-hyperledger-a-comprehensive-guide/

[11] AWS, *Instances T2 Amazon EC2* , AWS, Accessed on: Dec. 15, 2021. [Online]. Available: https://aws.amazon.com/fr/ec2/instance-types/t2/

[12] Hyperledger, *Install Samples, Binaries, and Docker Images* , Hyperledger, Accessed on: Dec. 15, 2021. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/install.html

[13] ResearchGate, *Service Discovery for Hyperledger Fabric - Scientific Figure*, Accessed on: Dec. 15, 2021. [Online]. Available: https://www.researchgate.net/figure/Hyperledger-Fabric-high-level-transaction-flow_fig02_325009023