Learning Experiences on Blockchain Related Projects

Xuheng Duan

Computer Science Department

Boston College

Supervisor

Lewis Tseng

In partial ful llment of the requirements for the degree of

Bachelor of Science in Computer Science

Abstract

Today, blockchain technology becomes a hot topic and attracts people's attention worldwide. Initially, blockchain was introduced in 2008 by Satoshi Nakamoto as a secure system for bitcoin. Today, there are public blockchain projects like Ethereum and Bitcoin that enables anyone to process secure peer-to-peer transactions. Meanwhile, there are also private blockchain projects like Hyperledger Fabric[1], which restricts the participants and realizes secure communications between corporations.

Throughout the past year, I studied the Hyperledger blockchain system and encountered numerous di culties. Moreover, witnessing so many di erent sorts of blockchain projects, one would like to understand and compare di erent blockchain performance at scale. Thus, I also studied three di erent benchmark tools, namely Blockbench[2], Caliper++[3] and Hyperledger Caliper[4], and focused on the latter two. At last, we used Mininet[5] to emulate a virtual network and tested the performance of Ethereum. Overcoming the struggles, I document my learning outcomes and share my experience on these open-source projects with those who would like to engage the eld of blockchain.

Contents

1. Introduction

The thesis includes four di erent major parts, corresponding to four di erent systems: Hyperledger Fabric, Hyperledger Caliper, Caliper++, and Mininet/Ethereum. In the introduction part, I will brie y talk about the basic concepts and structures of the four systems. Then, in the major sections, I included detailed discussion, some related work, and working notes. Lastly, there is a short summary at the end of the thesis.

1.1 Hyperledger Fabric

Admittedly, traditional public blockchain systems have proven their utility. However, many enterprise use cases require a private environment in which the permissionless blockchain project could not provide. For instance, in nance transactions, users need to know each other's identities in order to avoid money laundering or nancial fraud. Although many developers adapted earlier public blockchains for business scenarios, the projects still have some inherent problems. As a solution, Hyperledger Fabric[6] was designed for enterprise use, combining novel and unique philosophies. It aims to build a secure network where all participants must be identi ed. Older blockchain projects like bitcoin have issues of low transaction throughput and high latency of transactions because it relies on miners to package and verify the data. However, Hyperledger Fabric designs orderers to arrange the transactions, which brings the transaction speed to a new level. Using modular architecture and smart contracts, Hyperledger Fabric also makes it easy for participants to customize the network.

1.2 Hyperledger Caliper

Hyperledger Caliper[7] is one of the official benchmark frameworks. Treating the blockchain system as a whole, Caliper can target the SUT(System Under Test) by user-defined chaincodes and integrate the system responses into a meaningful report. Caliper consists of two important configuration files: benchmark files and network files. In a benchmark file, users can define different parameters they want, such as transaction numbers and transactions per second. Furthermore, users are able to customize their own chaincode and plug it in the Caliper, which would be called later during benchmark phase. On the other hand, Caliper uses a network configuration file to contact the SUT. The network file usually defines the network topology, nodes' endpoints, and smart contracts Caliper should deploy or interact with.

1.3 Caliper++

Although Caliper version 0.2 is powerful, it still has some disadvantages. For instance, it could only perform on local, predefined networks. It is also hard to run on large-scale and distributed networks. In addition, Caliper has a benchmark client closely connected to Fabric transaction workflow, so the client would need to wait for endorsing peers before they validate the responses, which would potentially influence the benchmark results. Thus, another development team designed Caliper++ that extended the functions of the original Caliper. The redesigned Caliper++ supports additional functionalities for benchmark Kafka-based[8] Fabric network at scale. To be more specific, it contains scripts that could generate configuration files for Fabric network topology and brings up a large scale of network across any number of cluster nodes. It successfully solved the problem that hindered Caliper and brought new ideas to design a distributed benchmark framework.

1.4 Mininet and Ethereum

Mininet[9] is a battle-tested software that is widely used in prototyping and evaluating Software-De ned Networks(SDNs). It has the capability to illustrate realistic simulation of physical network topology, and it also supports many extensible con- gurations. Another world-famous Blockchain project, Ethereum, supports customized applications. We deployed Go-Ethereum(Geth) network on the mininet and raised some meaningful results.

2. Hyperledger Fabric

2.1 Structure and concepts

A typical Hyperledger Fabric transaction will undergo the following steps: the of Hyperledger Fabric. The peers usually are assigned names in the format of peerX.orgX.example.com, where X is a number. Similar to miners in public blockchain systems, Hyperledger Fabric uses peer units to accomplish validating transactions and to commit data to the ledger. Meanwhile, the peers also run chaincodes(smart contracts), which contain user-de ned codes, to manage the assets.

2.1.2 Orderer and Channel

Di erent from public blockchain systems, Fabric uses orderers to complete consensus work. After peer units have completed validating the transactions, orderers would receive the endorsed blocks from peers, and then start to reach a consensus. Eventually, orderer nodes would add the blocks to the ledger, nishing the entire process. Peers could update their own data from the world state at other times. On top of orderers, a network needs multiple channels to operate. Channels give Hyperledger Fabric the ability to build private communications between desired organizations while not interfering with the other organizations. As a result, a channel creates a black box and private ledger for consortiums that are invited to the channel.

2.1.3 Certi cate Authority(CA) and Membership Service Provider (MSP)

CA and MSP are used to verify one's identity in Hyperledger Fabric. The CA would issue cryptographic materials for the network components, like orderers, organizations, and peers. These materials, usually a pair of public and private keys, would be later used to enroll components' admin, as well as peers. After the network generates cryptographic materials, it also needs MSPs to identify which certi cates are needed and to issue valid identities for their members. Using the metaphor I learned before, certi cates are similar to the credit cards that can tell whether or not an individual is able to pay(and this is created by the bank, which is parallel to CA in Hyperledger Fabric), and MSPs are the list of accepted credit cards that individual can use in a store.

2.2 Related Work and Suggestions

To successfully launch a functional Hyperledger Fabric network, one needs to address several components and set up the running environment correctly: pre-requisites, crypto materials, channel con gurations, and chaincode. After we successfully install the chaincode on the Fabric network, we could query against our ledgers and retrieve the data.

Figure 2.4 Create and Join Channel for Hyperledger Fabric ver2.0

In gure 2.6, we move 10 values from the account a to b, and then we asked the ledger to see if the nal result is correct.

Figure 2.5 Install Chaincode for Hyperledger Fabric ver2.0

Figure 2.6 Install Chaincode for Hyperledger Fabric ver2.0

Figure 2.7 Invoke Chaincode for Hyperledger Fabric ver2.0

Following the o cial guidance is the most straightforward way to start with Hyperledger Fabric. The o cial BYFN[10] tutorial explains most of the concepts one needs to know about setting up a functional Hyperledger Fabric network. However, completing the tutorial does not provide one enough knowledge nor experience to construct a customized network. Hereby, I will include some di culties

3. Hyperledger Caliper and Caliper++

3.1 Structure and concepts

Hyperledger Caliper has three major parts, Workload Module, Benchmark con gu- ration le, and network con guration le. Caliper itself does not have any tangible benchmark implementation. Alternatively, workload modules are responsible for generating transactions and submits them against the System Under Test (SUT). These Workload Modules are Node.js codes that utilize outside APIs. Thus, be careful with the Node.js version when one uses Caliper. The current stable version should be either Node.js 8 or Node.js 10. Benchmark con guration le consists of detailed information about the benchmarks, like how it should be executed and what's the desired transaction rate. By default, Hyperledger Caliper contains two types of benchmark scenarios. One of which mimics the bank environment and the other one contains various simple and straightforward tests again SUT, like plain query and open functions. Lastly, network con guration les de ne the SUT network topology. Since Caliper itself is a benchmark workframe, which does not generate the SUT itself, it is users' responsibility to construct a functional SUT with accessible endpoint addresses. Similarly, Caliper itself utilized les from BYFN.

When I touched Hyperledger Caliper, it was still on version 0.2 and has little documentation on distributed benchmarks. Thus, another development team de- signed Caliper++[3]. Their contributions include the followings: they extended Hyperledger Caliper by adding support for distributed benchmarking. Further- more, the development team designed scripts that can start Fabric with varying sizes and con gurations. With Caliper++ the team successfully showed that endorsing peers in the Hyperledger Fabric network with Kafka ordering service is the bottleneck.

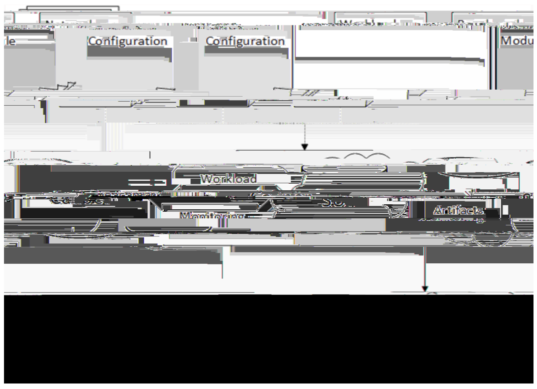

Figure 3.1 The architecture of Caliper

3.2 Related Work and Notes

While reproducing the results in *Understanding the scalability of Hyperledger Fabric*[3 and thus Docker Swarm was not a viable option to me. But, there could be multiple ways to build up a cluster for the Hyperledger Fabric network. To establish a cluster, I chose between Google Cloud Platform and virtual machines as nodes. After setting up the machines, I need to connect these nodes together, making sure they could communicate with each other. In my case, I set up two virtual machines in VirtualBox, with Ubuntu 16.04 LTS. [12] and [13] guided me complete this task. Meanwhile, users need to emulate the network so that virtual machines could connect to the internet and receive an external IP address. [14] showed how to set up network cards. After ensuring that both machines could talk with each other by Secure Socket Shell(ssh), I used example con guration les from Hyperledger Fabric o cial site to construct the network. Everything was smooth and neat until when I tried to adjust the YAML les into a more complex network topology. Although articles like [15] and [16] explained how to con gure one's own Hyperledger Fabric network, unfortunately, I could not resolve the problems I encountered later.

Hyperledger Fabric network needs certi cates on every host. Therefore, before using Caliper[4] or Caliper++, users need to assign the Hyperledger Fabric artifacts, including certi cates, crypto information, and other binaries to every host and under correct paths. Manually allocating these materials is unrealistic, and I did not gure out an easy way that could help me to do that. On the other hand, making sure hosts in the distributed system could contact each other properly is challenging. Most of the time, connection requests from one host will be denied by another one due to diverse problems. There is no easy way to solve them, except by carefully examining the bug and getting help from others.

Figure 3.2 illustrates a typical report generated by Hyperledger Caliper. The report would express the SUT on the right side of the page, under \System Under Test" section. In the gure, I targeted Caliper against Hyperledger Caliper version 1.4.1 and set up the network topology with 2 organizations, one peer each. The orderer type was solo, and the network was on a single host. Finally, the world state was maintained by GoLevel Database. On the other hand, the report section

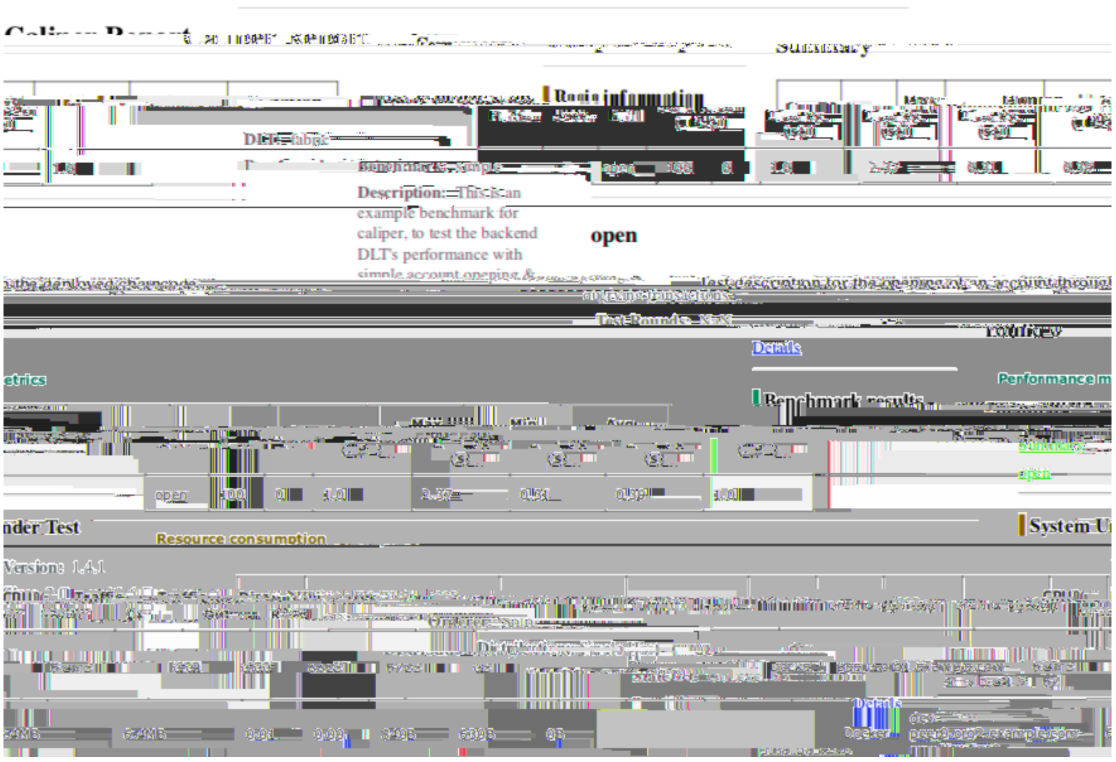

Figure 3.2 Hyperledger Caliper report

4. Ethereum and Mininet

Finally, we did some work on Ethereum and Mininet[9 which are bandwidth, delay, jitter, and network loss.

Here the throughput is measured as the number of blocks that are successfully appended to the main (canonical) chain per second. Latency is measured as the duration between (i) the time a miner reports it has mined a potential block; and (ii) the time the miner reports its block has reached the canonical chain. In this set of experiments, we use a single switch topology.

Figure 4.4 describes the impact of added latency(link latency) on network latency.

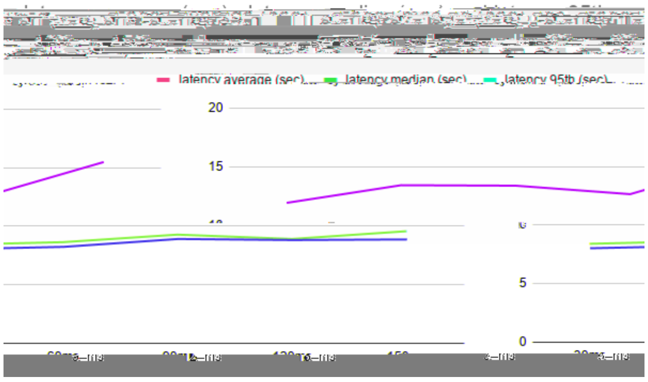

Figure 4.4 Geth Latency vs. Link Latency

Meanwhile, gure 4.5 presents the throughput under added latency (inside Mininet) from 30ms to 150ms. The latency between the host and the switch is called the \added latency" that we use Mininet to simulate. We can see that the throughput is quite stable with moderate latency. Then we enlarged the added latency to 500ms and 1000ms. Check gure 4.6 for more details.

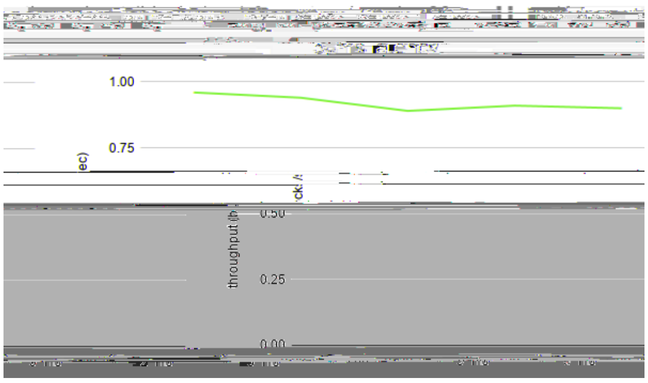

Figure 4.5 Geth Throughput vs. Link Latency

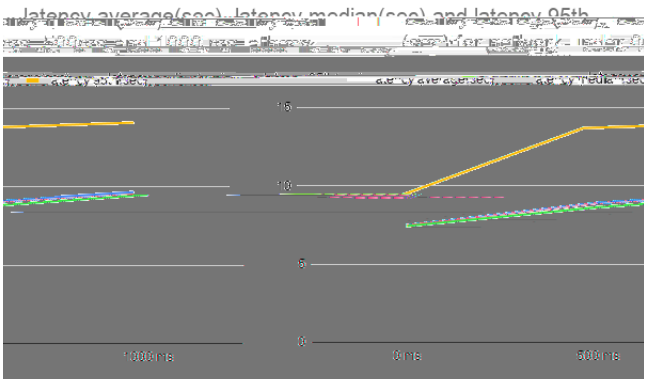

Figure 4.6 Geth Latency vs. Large Link Latency

4.4.2 Di erent Topology and Network Failures

Other than normal network delay, we also tested the system with various hardware-level topologies and di erent failure patterns.

Table 4.1 presents Geth's latency and throughput numbers in di erent topology with 2% and 4% packet loss rate. Each topology has 5 miners with each link having 500ms delay and 10ms jitter. The FatTree topology has three layers (core, aggregate, and edge) and one can adjust the number of switches in each layer and the number of hosts under each edge switch. For links, one can tune the network delay, jitter, loss, and bandwidth constraint, and for hosts, one can con gure the CPU constraint and the maximum number of physical cores to use. Our result indicate that fat-tree, while su ering slightly higher latency, is more robust to network loss. We have not observed similar study and analysis before.

Table 4.1: Geth performance vs. Network Failures

	Fat-Tree 2%loss	Fat-Tree 4%loss	Linear 2%loss	Linear 4%loss
throughput (blocks/sec)	0.82	0.8	0.82	0.76
avg. latency (sec)	9.83	9.69	9.53	9.67
med. latency (sec)	9.29	9.17	8.86	9.17
95th latency (sec)	15.14	14.93	15.24	14.48

4.4.3 Heterogeneous Machines

Table 4.2 presents Geth's latency and throughput numbers with heterogeneous hosts, i.e., each host has a di erent computation power. In this particular evaluation, we have a Fat-tree topology with 500ms added latency per link, no jitter, and no package loss. The relative computation power of each host is listed in the table. The evaluation runs for 3153.27s, and the whole system generates 2023 valid blocks.

Table 4.2: Geth's performance with heterogeneous machines

	Overall	host1	host2	host3	host4	host5
relative comp. power	1	0.1	0.1	0.27	0.27	0.26
throughput (blocks/sec)	0.64	0.02	0.01	0.27	0.01	0.33
avg. latency (sec)	24.23	268.31	330.15	13.35	17.27	10.27
med. latency (sec)	11.16	270.8	332	12.46	18.36	9.66
95th latency (sec)	28.78	424.79	511.6	22.71	24.03	16.41

4.4.4 Large-scale Test

Finally, we extend our framework on a virtual instance with 96 vCPUs and 360 GB memory. The network adopts a Fat-Tree topology with 30 hosts, 500 ms added latency per link, no jitter, and no package loss. Table below presents the result. As expected, the throughput is similar regardless of the number of hosts due to the design of PoW protocol with a xed di culty. Table 4.3 sh(wit)1(h)-327(a)-326(xed)-327(

5. Other Errors

Here I include some common errors, as well as some general tips for successors:

Docker Socket: Sometimes one will get permission denied from Docker daemon socket. This might be caused by various reasons, but highly likely because of the low level of permission.[20] explained some potential solutions and sudo chmod 666 /var/run/docker.sock solved my issue.

Node.js Version: Hyperledger Fabric and Hyperledger Caliper have a restricted requirement for Node.js version. They do not support the latest version of Node (currently 14.0.0), but only version 8.x.x or 10.x.x. Thus, check the Node.js version before you start running them.

Cluster: As mentioned before, Caliper and Caliper++ used Docker Swarm to set up the cluster. Thus, I would recommend to learn and use swarm in the rst place. Otherwise, Kubernetes is also a viable option.

Bash Script: Hyperledger Fabric includes a heavy amount of scripts. Thus, learning how to read script les is also critical. [21] is the bash cheat sheet I used.

Research and choose the right tools. I did not do enough study before deciding which tool to use, and thus wasted a good amount of time switching from plan to plan. Thus, make a good plan of action is the most important part of a project.

Don't hesitate to ask. It would save me much time if I asked the author of the paper about the cluster and network earlier. Furthermore, the tech forum and community provided me a lot of assistance as well.

6. Summary

In general, this paper concentrates on the Hyperledger Fabric, Hyperledger Caliper, Caliper++, as well as Ethereum and mininet. Each part consists of the basic concepts of the system, and the related work. Summarizing the errors I made and giving out some advices, I hope my thesis provides enough information to those who wish to enter the eld of Blockchain, specially Hyperledger Fabric. Meanwhile, realizing my inadequacies in decision making, I also write some general suggestions to help others to improve.

Foremost, I would like to express my sincere gratitude to my advisor, Prof. Lewis Tseng, for the support and guide, for his patience, motivation, and encouragement. Nevertheless, I would like to thank my fellow, Haochen Pan, and Yingjian Wu, for their consistent help and assistance.

References

[1] T. L. Foundation, *Hyperledger fabric*, 2020. [Online]. Available: https:// www.hyperledger.org/projects/fabric .

[2] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, \Block-bench: A framework for analyzing privateblockchains", 2017. doi : http: //dx.doi.org/10.1145/3035918.3064033 .

[3] N. Q. Minh, D. Loghin, and T. T. A. Dinh, \Understanding the scalability of hyperledger fabric", 2019. [Online]. Available: https://bcdl.comp.nus. edu.sg/papers/understanding_the_scalability_of_hyperledger_ fabric.pdf .

[4] Hyperledger, *Measuring blockchain performance with hyperledger caliper*, Mar. 2018. [Online]. Available: https://www.hyperledger.org/blog/ 2018/03/19/measuring-blockchain-performance-with-hyperledger-caliper .

[5] *Mininet, http://mininet.org/*. [Online]. Available: http://mininet.org/ .

[6] *Hyperledger fabric: A blockchain platform for the enterprise*, Available at https://hyperledger-fabric.readthedocs.io/en/latest/index.html , Lastest Version: 2.1.

[7] *Caliper*, Mar. 2020. [Online]. Available: https://github.com/hyperledger/ caliper .

[8] J. Kreps, N. Narkhede, J. Rao, *et al.*, \Kafka: A distributed messaging system for log processing", 2011, NetDB, Vol. 11, pp. 1{7.

[9] M. Team. (). Mininet, an instant virtual network[Online].
Available1[(:)]TJ0 g 0 G/F42 11.9552 Tf54.6241 0 Td [(http://mininet.org/)]TJ0 g 0 G/F
.

[illegible]

[14] B. Linkletter, How to emulate a network using virtualbox, Jan. 2017. [Online]. Available: https://www.brianlinkletter.com/how-to-use-