Measurements, Analysis and Modeling of Hyperledger Fabric

Abstract—Bitcoin network

Index Terms—Bitcoin Network, Mining Pools, Malthusian Trap, Incentive Mechanism

1 Introduction

Bitcoin [1] is a decentralized peer to peer (P2P) cryptocurrency that was first proposed by Satoshi Nakamoto in 2008. Without resorting to any trusted third party, Bitcoin adapts a cryptographic proof mechanism that enables anonymous peers to complete transactions through the P2P network. Blockchain is the core mechanism of the Bitcoin system. It not only records historical transactions from Bitcoin clients, but also prevents the Bitcoin network from double spending attacks [2]. The Bitcoin network participants, who maintain and update the ongoing chain of blocks, are called miners. These miners compete in a mining race driven by an incentive mechanism [3], [4], where the one who first solves the Bitcoin cryptographic puzzle [5] has the right to collect unconfirmed transactions into a new block, append the new block to the main chain, i.e., the longest chain of blocks, and gain some BTCs [6] as a mining reward.

2 Preliminary

2.1 Cluster Configuration

This is about cluster configuration

2.2 MSP

Certificates

2.3 TLS

TLS secure protocol descrition

3 Environment

Hyperledger Fabric version 1.4
Fabric SDK v1.0.0
Jmeter

4 BlockTime Solo Orderer Mode

4.1 Model Definition

4.2 Model of Orderer's Configuration

The configuration file, i.e., configtx.yaml, configures the orderer service as follows.

TABLE 1
Configuration Notations

Notations	Descriptions
N_{msg}	MaxMessageCount
S_{max}	AbsoluteMaxBytes
S_{pref}	PreferredMaxBytes
T_{batch}	Time to generate a block when MaxMessageCount is not satisfied

TABLE 2
Customized Notations

Notations	Descriptions
T_σ	Time to generate a block when MaxMessageCount is satisfied
TAR	Transaction arrival rate
$BlockTime$	Time it takes to generate a block
$TxDelay$	Time it takes to issue a transaction
$S_{payload}$	Size of transaction payload
$Throughput$	Throughput of transactions

```
BatchTimeout: 2s
BatchSize:
    MaxMessageCount: 10
    AbsoluteMaxBytes: 98 MB
    PreferredMaxBytes: 512 KB
```

Case Study 1 without considering block size, we have the following model, Given a TAR of which $\Delta t = 1$, the time it costs is given as follows, the average block time is,

$$BlkTime = \begin{cases} \infty & TAR = 0 \\ T_{batch} & 0 < TAR \leq \frac{N_{msg}}{T_{batch}} \\ \frac{N_{msg}}{TAR} & \frac{N_{msg}}{T_{batch}} < TAR \leq \frac{N_{msg}}{T_\sigma} \\ \left\lfloor \frac{TAR}{N_{msg}} - 1 \right\rfloor \cdot T_\sigma & \frac{N_{msg}}{T_\sigma} < TAR, TAR \mid N_m \\ \left\lfloor \frac{TAR}{N_{msg}} - 1 \right\rfloor \cdot T_\sigma + T_{batch} & otherwise \end{cases} \quad (1)$$

If $TAR = 0$, then $BlockTime = \infty$. It means that if there are no transactions, there are no blocks.

If $0 < TAR \leq \frac{MaxMessageCount}{BatchTimeout}$. It means that the number of transactions are less than MaxMessageCount given a BatchTimeout. Therefore, blocks are created for each BatchTimeout.

If $\frac{MaxMessageCount}{BatchTimeout} < TAR$. It means that the number of transactions are larger than MaxMessageCount in each BatchTimeout. Therefore, blocks are created as soon as possible and σ is a small value.

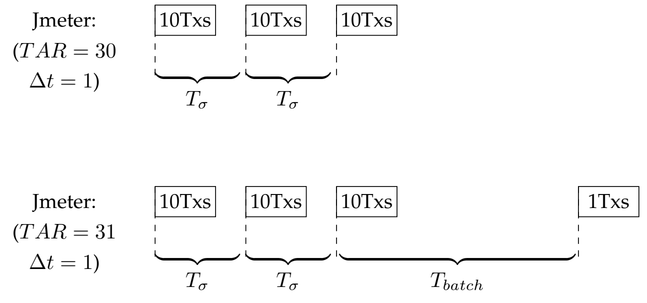

Fig. 1. The TAR Configurations

4.2.1 Experiment 1: How TAR affects T_{sigma} and Throughput

In this section, we focus on how TAR affects the block time when MaxMessageCount is satisfied.

Experiment 1 – Setup: for normal TAR we can use workload generators "Thread Group" on Ubuntu 01; while for small TAR we can use other workload generators "Ultimate Thread Group" on Ubuntu 01.

Experiment 1 – Setup: for experiment analysis, we can check with logs in a peer. We have a program to analyze the log data, see $readdata.py$ at Github benchmark module at the $q3v14$ folder.

Experiment 1 – Observation 1: when TAR achieves a high value (e.g., 80 tps, 90 tps), then a block cannot be fully make used (e.g., 8 transactions per block + 2 transactions per block, instead of 10 transactions per block).

Experiment 1 – Observation 2: when TAR achieve a very high value (e.g., 200 tps), then a peer cannot handle such a high frequent event, and the SDK client will be rejected by the peer. See more error about TAR=200 per peer, Error "peer0.org2.example.com:7051, PKIid:6830efc127d4818973edd435ee7df6fa6fdf3e23286479b1e34dd8229266559b5 isn't responsive: EOF". – To solve this problem, we need more distributed machines.

TO DO – why 70 tps to 80 tps, much not-full used blocks are created?

Experiment 2 – Observation 1: for small TAR (e.g., 20 tps, 30 tps, 40 tps, 60 tps), setting of TAR (e.g., 11 tps, 21 tps, 31 tps, 51 tps) highly affect the T_{sigma}.

TODO –: for large TAR(e.g., 80 tps, 90 tps, 140 tps), why experiment 2 is much lower than experiement 1?

TO DO –: why TAR=91 results in fewer null-fully used blocks than TAR=90?

4.2.2 Endorsement Policy (OR) on Transaction Delay

Experiment 3 - Setup: Endorsement Policy (OR) the source code is at Ubuntu 01 - nodeTest invoke3.js.

Experiment 3 - Setup: we can calculate with timestamps at invoke3.js.

Experiment 3 - Setup: we save the timestamp results with Jmeter in a res.xml file.

Experiment 3 - Setup: we can summary the timestamp results with readxml.py at nodeTest in ubuntutu 01.

Experiment 3 - Setup: get proposal approval from localhost, see invoke3.js nodeTest in ubuntu 01.

Experiment 3: Set up: transaction payload size 2 bytes

Experiment 3 - Observation 1: when TAR are higher than 10 transactions per second, the transaction delay will increase up, since the length of the waiting line is longer.

Experiment 3 - Observation 2: when Tx Orderer Delay are larger than 3 seconds, the transaction proposal will be rejected.

Experiment 3 - Observation 3: transaction proposal delay increases linearly with transaction arrival rate, while orderer transaction delay is not.

Experiment 3 - Observation 4: why these 14 transactions are rejected? It is nothing about orderer. It is limited to the capacity of ubuntu01 local peer. While the next experiment 4, the ubuntu00 remote peer has more powerful computing capacity.

Optimal Solution to this problem: try to let different peers to help to send transaction proposals at the same time.

4.2.3 Endorsement Policy (OR) on Transaction Delay

Experiment 4 - Setup: Endorsement Policy (OR) the source code is at Ubuntu 01 - nodeTest invoke3a.js.

Experiment 4 - Setup: we can calculate with timestamps at invoke3a.js.

Experiment 4 - Setup: we save the timestamp results with Jmeter in a res.xml file.

Experiment 4 - Setup: we can summary the timestamp results with readxml.py at nodeTest in ubuntutu 01.

Experiment 4: Set up: get proposal approval from remote peer, see invoke3a.js nodeTest in ubuntu 01.

Experiment 4: Set up: transaction payload size 2 bytes

Experiment 4 - Observation 1: we can not find any performance difference between local proposers and remote proposers.

4.2.4 Endorsement Policy (AND) on Transaction Delay

Experiment 5 - Setup: Endorsement Policy (AND) see Github benchmark module the config7.md and the config8.md files.

Experiment 5 - Setup: Endorsement Policy (AND) see invoke5.js at nodeTest folder in Ubuntu01 machine.

Experiment 5 - Observation 1: endorsement policy (AND) will have relatively small larger in transaction delay than endorsement policy (OR), when there are only two peers.

Experiment 5 - Observation 2: if the endorsement policy (AND), we will have more responses say two times, and the peer ubuntu00 will have double burdens, resulting in peer channel problem.

Experiment 5 - Next Steps: if we have more peers say 16 peers using the endorsement policy (AND), then how about the transaction delay?

Experiment 5: Next Steps: what is the technical solutions to the error problem with the eventhub? the eventhub shutdown problem see $https://stackoverflow.com/questions/52773529/hyperledger-fabric-nodejs-sdk-eventhub-has-been-shutdown$. Then try to build up your own queue on the peer nodes.

TABLE 3
Experiment 1: TAR affects T_{sigma}, $number of transactions of a block$ and $number of transactions rejected$

TAR	20	30	40	50	60	70	80	90	100	150
$BlkTime$.AVG	0.338	0.3828	0.4151	0.4035	0.5730	0.4866	**1.1803**	**1.1792**	**1.0968**	**1.0285**
$BlkTime$.MAX	0.325	0.4040	0.5727	0.5260	0.6412	0.5290	**1.5029**	**1.5668**	**1.3033**	**1.4904**
$BlkTime$.MIN	0.35	0.3715	0.3337	0.3362	0.4634	0.4492	0.6994	0.4832	0.7277	0.7244

TABLE 4
Experiment 2: how batch of TAR affects T_{sigma}

TAR	11	21	31	41	51	61	71	81	91	141
$BlkTime$.AVG	2.2673	1.2507	1.0408	0.9152	0.7564	0.7073	0.7243	**0.9656**	**0.7010**	**nll**
$BlkTime$.MAX	2.4900	1.2920	1.3040	1.1757	0.8044	0.8395	0.9021	**1.3219**	**0.8963**	**nll**
$BlkTime$.MIN	2.0030	1.1800	0.8203	0.7715	0.7222	0.6288	0.5626	0.5490	0.5187	nll

TABLE 5
Experiment 3: Endorsement Policy "OR", with local proposer

TAR	Phase 1: Tx Proposal Delay		Phase 2: Tx Orderer Delay		Total Tx Delay	Rounds	Accepted Txs	Rejected Txs
	Delay (s)	Proportion (%)	Delay (s)	Proportion (%)				
5	0.75	24.67	2.29	75.33	3.04	5	25	0
10	1.52	71.69	0.60	28.31	2.12	5	50	0
30	4.58	82.67	0.96	17.33	5.54	5	150	0
50	7.88	85.65	1.32	14.35	9.2	5	250	0
70	11.19	88.11	1.51	11.89	12.70	5	350	0
90	**14.39**	**89.10**	**1.76**	**10.90**	**16.15**	**5**	**436**	**14**

TABLE 6
Experiment 4: Endorsement Policy "OR", with remote proposer

TAR	Phase 1: Tx Proposal Delay		Phase 2: Tx Orderer Delay		Total Tx Delay	Rounds	Accepted Txs	Rejected Txs
	Delay (s)	Proportion (%)	Delay (s)	Proportion (%)				
5	0.80	26.05	2.27	73.95	3.07	5	25	0
10	1.52	73.07	0.56	26.93	2.08	5	50	0
30	4.51	83.82	0.87	16.18	5.38	5	150	0
50	7.60	86.75	1.16	13.25	8.76	5	250	0
70	10.88	88.62	1.41	11.48	12.29	5	350	0
90	14.33	89.67	1.65	10.33	15.98	5	450	0

TABLE 7
Experiment 5: Endorsement Policy "AND", with remote proposer

TAR	Phase 1: Tx Proposal Delay		Phase 2: Tx Orderer Delay		Total Tx Delay	Rounds	Accepted Txs	Rejected Txs
	Delay (s)	Proportion (%)	Delay (s)	Proportion (%)				
5	0.74	24.74	2.25	75.25	2.99	5	25	0
10	1.56	71.88	0.61	28.12	2.17	5	50	0
30	4.65	82.88	0.96	17.12	5.61	5	150	0
50	7.82	86.12	1.26	13.88	9.08	5	250	0
70	11.09	87.59	1.57	12.41	12.66	5	347	3
90	14.34	88.62	1.84	11.37	16.18	5	438	12

Solutions to the eventhub problem: build up your own queue $https : //stackoverflow.com/questions/52773529/hyperledger-fabric-nodejs-sdk-eventhub-has-been-shutdown$. See Event hub with read-write conflict transactions $https : //ieeexplore.ieee.org/stamp/stamp.jsp?arnumber = 8751453$.

4.2.5 Transaction Size on Transaction Delay

Experiment 6 - Setup: Transaction size 1 KB

Experiment 6 - Setup: invoke3.js in "nodeTest" folder in ubuntu 01.

5 Kafka Orderer Mode

5.0.1 Endorsement Policy (OR) on Transaction Delay

Experiment 6 - Setup: Endorsement Policy (OR) the source code is at Ubuntu 01 - nodeTest invoke3.js.

Experiment 6 - Setup: we can calculate with timestamps at invoke3.js.

Experiment 6 - Setup: we save the timestamp results with Jmeter in a res.xml file.

Experiment 6 - Setup: we can summary the timestamp results with readxml.py at nodeTest in ubuntutu 01.

Experiment 6 - Setup: get proposal approval from localhost, see invoke3.js nodeTest in ubuntu 01.

Experiment 6: Set up: transaction payload size 2 bytes

Experiment 6: setup python analysis file readxml.py is in nodeTest/bench_result. Usage of the readxml.py: each time there is a new experiment data, people should run python readxml.py to analyze the res.xml data file. After finishing it, people should delete the res.xml file.

Observation 1: the experiment 6 should have a comparison with the experiment 3, from which we know the difference between Hyperledger Solo orderer and Hyperledger Kafka orderer.

Observation 2: the experiment 6 shows almost the same results with experiment 3. When the transaction arrival rate are going high e.g., 90 transactions per second, then the performance bottleneck will be in peer. Specifically, it will be when peer sends proposals.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun, "Misbehavior in bitcoin: A study of double-spending and accountability," *ACM Transactions on Information and System Security (TISSEC)*, vol. 18, no. 1, p. 2, 2015.

[3] Y. Lewenberg, Y. Bachrach, Y. Sompolinsky, A. Zohar, and J. S. Rosenschein, "Bitcoin mining pools: A cooperative game theoretic analysis," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 919–927.

[4] O. Schrijvers, J. Bonneau, D. Boneh, and T. Roughgarden, "Incentive compatibility of bitcoin mining pool reward functions," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 477–498.

[5] I. Giechaskiel, C. Cremers, and K. B. Rasmussen, "On bitcoin security in the presence of broken cryptographic primitives," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 201–222.

[6] BTC. [Online]. Available: https://en.bitcoin.it/wiki/Bitcoin, Accessed on 31 January 2019.

TABLE 8
Experiment 6: Transaction Size on Transaction Delay 1 KB

TAR	Phase 1: Tx Proposal Delay	Phase 2: Tx Orderer Delay	Total Tx Delay	Rounds	Accepted Txs	Rejected Txs
5	0.78	2.33	3.11	5	25	0
10	1.54	0.68	2.22	5	50	0
30	4.67	0.94	5.61	5	150	0
50	7.83	1.25	9.08	5	250	0
70	11.14	1.48	12.62	5	350	0
90	14.54	1.86	16.4	5	449	1

TABLE 9
Experiment 6: Endorsement Policy ”OR”, with local proposer

TAR	Phase 1: Tx Proposal Delay		Phase 2: Tx Orderer Delay		Total Tx Delay	Rounds	Accepted Txs	Rejected Txs
	Delay (s)	Proportion (%)	Delay (s)	Proportion (%)				
5	0.74	24.18	2.32	75.82	3.06	5	25	0
10	1.52	71.36	0.61	28.64	2.13	5	50	0
30	4.62	81.76	1.03	18.24	5.65	5	150	0
50	7.92	85.81	1.31	14.19	9.23	5	250	0
70	11.15	87.72	1.56	12.28	12.71	5	350	0
90	14.07	88.21	1.88	11.79	15.95	5	433	17