Working Paper presented at the

Peer-to-Peer Financial Systems
2017 Workshop

2017

Performance Improvement of the Consortium Blockchain for Financial Business Applications

Takeshi Miyamae
Fujitsu Laboratories Ltd., Fujitsu Software Technologies Limited

Takeo Honda
Fujitsu Laboratories Ltd., Fujitsu Software Technologies Limited

Masahisa Tamura
Fujitsu Laboratories Ltd., Fujitsu Software Technologies Limited

Motoyuki Kawaba
Fujitsu Laboratories Ltd., Fujitsu Software Technologies Limited

Powered by

P2P Financial Systems

Exponential Science

Performance Improvement of the Consortium Blockchain for Financial Business Applications

Takeshi Miyamae, Takeo Honda†, Masahisa Tamura and Motoyuki Kawaba
Fujitsu Laboratories Ltd. and †Fujitsu Software Technologies Limited
{miyamae.takeshi, honda.takeo, masahisa, kawaba}@jp.fujitsu.com

Abstract

In preparation for improving consortium blockchain's performance for financial business applications, we first surveyed typical records of transaction rates for various financial systems and the performance capacity of several known consortium blockchains at this moment. The blockchain itself does not offer performance advantage, but rather sacrifices performance to achieve tamper-resistance. However, the average transaction rates for blockchain technologies are continuously improving, and have already reached around 1,000-2,000 TPS, which is equivalent to the transaction rate for Zengin System, the common name for Japanese Bank's Payment Clearing Network.

Next, we analyzed and improved transaction rate for Hyperledger Fabric, one of the open source software tools for implementing consortium blockchain. We found that inefficient message transfer between the platform's container and the application's container was the main cause of Fabric's performance bottleneck. Therefore, we have introduced more efficient API between the containers and improved the transaction rate from 725 TPS to 1,350 TPS (86% increase).

Finally, we analyzed Fabric's performance under the artificial network latency. We determine that Fabric running over a wide area network might be required to replace its consensus algorithm to improve its transaction rate, because other causes of performance deterioration, including the consensus algorithm, seem to become dominant.

1. Introduction

The records for transaction rates actually processed by financial systems have increased in the past 50 years. Although most organizations have not disclosed their performance values, we can grasp the rough trend in transaction rate from a handful of articles on the financial systems stating their actual transaction rates.

In advance, we must mention some notes about the transaction rates for the financial systems we will introduce in this paper. First, they are not capacities of the systems, but rather past records of the transaction rates actually processed by the systems. Second, they are not 'average', but 'peak' transaction rates. As systems that can handle average rate cannot necessarily handle peak rate, we will consider only peak rate from the beginning. However, as average rates are more often reported than peak rates, we simply estimate the peak rate as 5 times the average rate whenever we are only given the average rate from the article.

Figure 1 shows typical records for peak transaction rate for financial systems in the past 50 years. From the 1960s to the 1990s, Japanese banks continued to invest in and develop their own core banking systems to introduce real-time online services (1st online systems, 1965-1975, 89 TPS), started to handle general accounts and cooperate with surrounding systems such as domestic exchange systems and automatic teller machines (2nd online systems, 1975-1985, 308 TPS), and finally separated hardwares and control programs from their business logic in order to easily benefit from technological innovations in hardware (3rd online systems, 1985-1995, 536 TPS) [1]. In the securities industries the transaction rate was drastically improved after 2000. The transaction rate for the NYSE (New York Stock Exchange) grew from 109 TPS to 1,786 TPS in the first decade of the 2000s due to the increasing small divided lots of stock orders [2]. In the case of the JPX (Japan Exchange Group), the transaction rate grew from 1,736 TPS to 6,366 TPS after the first decade of the 2000s due to the increase in stock orders thanks to "Abenomics," Prime Minister Abe Shinzo's economic policies [3]. JPX's another report [4] says that several thousands to several ten thousands TPS are required for securities industry's post-trade processing. Zengin System (Japanese Bank's Payment Clearing Network), which handles Japanese domestic exchange, implies that its peak transaction rate was 1,080 TPS in 2012 [5]. Finally, VISA Inc. implies that VisaNet's peak transaction rate was 15,855 TPS in 2010 [6].

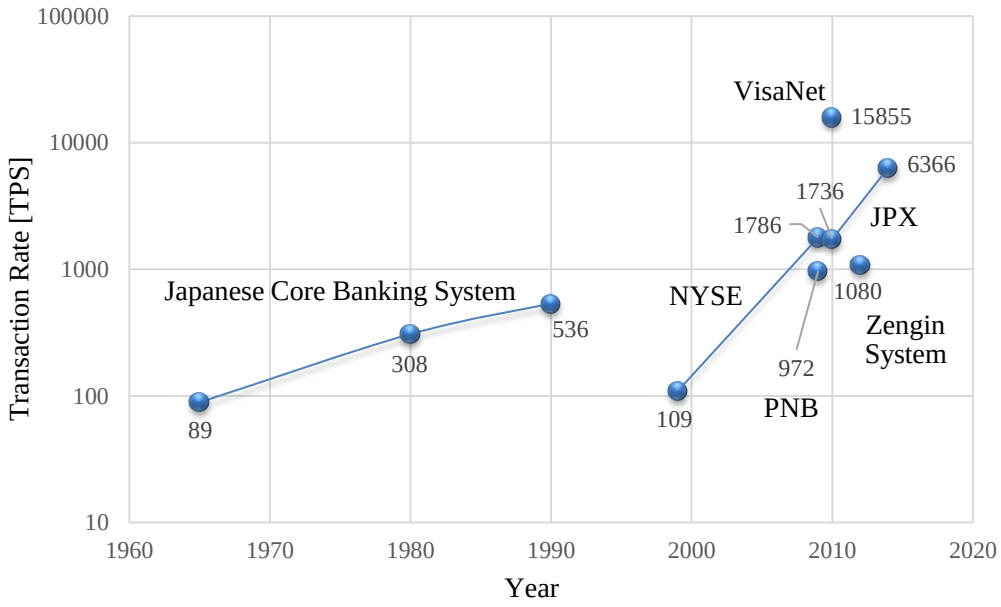

Figure 1: Typical records of transaction rates for financial systems.

On the other side, in the 2010s, a huge number of blockchain technologies have emerged and their transaction rates are continually improving day by day. 'Miyabi', a consortium blockchain developed by bitFlyer and using Paxos-like consensus algorithm "BFK2" [7], was evaluated in a report [5], which claimed that it has achieved 1,500 TPS within a single data center. Moreover, BFK2 has a good finality feature (= transaction logs are immediately recorded on its ledger) and the blocks are always decided deterministically. 'Mijin', one of the aggressive consortium blockchain projects developed by Tech Bureau, insists that it has already achieved over 3,000 TPS on three nodes [8]; however, as its source code has not been disclosed yet, it is difficult for us to know its architecture in detail. An open source consortium blockchain project called 'Fabric' [9], which is a Hyperledger Project in incubation originally developed by IBM, achieves 700 TPS at v0.6 [10]. IBM showed that it will improve Fabric's transaction rate up to 1,000 TPS at v1.0 and provide its blockchain cloud service based on Fabric v1.0 [11]. 'Iroha' [12], which is also a Hyperledger Project in incubation developed by Soramitsu, is attempting to achieve thousands of TPS due to Sumeragi architecture, UDP multicast, and C++ technology, in spite of the fact that achieving finality in 2 s was originally designed to be guaranteed [13]. One of the developers of 'Sawtooth Lake' [14], which is another Hyperledger Project driven by Intel, showed that 1,000 TPS is their current performance target by splitting off the cryptographic portions of validation into its own space that can operate faster [15]. Finally, we have to mention that Enterprise Ethereum [16], developed by Enterprise Ethereum Alliance (EEA), has a relatively low performance target (100 TPS) due to its middle-size network consisting of around ten parties [17]. From these surveys, summarized in Table 1, we found that the average performance of

the latest consortium blockchains is around 1,000-2,000 TPS at this moment.

In this paper, we will focus on Hyperledger Fabric and analyze the causes of its performance bottlenecks so that we can more deeply understand the architecture and problems of general consortium blockchain methods from the viewpoint of performance.

Table 1: Transaction rates for known consortium blockchains

blockchain	**core developer**	**region**	**the number of nodes**	**transaction rate (TPS)**	**announcement date**
Miyabi	bitFlyer	1	unknown	1,500	11/30/2016
mijin	Tech Bureau	1	3	3,085	12/20/2016
Fabric v0.6	Hyperledger Project (IBM)	1	1	700	10/27/2016
Fabric v1.0	Hyperledger Project (IBM)	unknown	unknown	1,000+ (target)	3/20/2017
Iroha	Hyperledger Project (Soramitsu)	unknown	unknown	Thousands (target)	11/2/2016
Sawtooth Lake	Hyperledger Project (Intel)	unknown	unknown	1,000 (target)	3/13/2017
Enterprise Ethereum	Enterprise Ethereum Alliance (EEA)	10	unknown	100 (target)	2/28/2017

2. Hyperledger Fabric

2.1. v0.6.1 Ledger Architecture

Chaincode is the business logic program in Fabric, which allows us to define and exchange assets using unspent transaction outputs as the inputs for subsequent transactions. The state changes of assets are recorded as transactions, each of which results in a set of key value writes on a ledger. The ledger is comprised of cryptographically chained data blocks and a state database to maintain Fabric's current state. The transactions can be submitted to the network and applied to the ledger on all peers, each of which is an uncentralized main Fabric process, is distributed among the network, and maintains a copy of the ledger. The more different organizations that manage each peer and copy of the ledger, the more tamper-resistant the ledger becomes.

2.2. Consensus Policy and v1.0 Release

In general, blockchain consensus is regarded as the achieved state when it is ensured that both the order and the result of the transactions have met certain conditions. The peers of Fabric v0.6.1 reach an agreement on the transaction order in an early stage, and all the transactions are executed serially, which is a disadvantage from the viewpoint of performance. The consensus algorithm of Fabric v0.6.1 is Practical Byzantine Fault Tolerance (PBFT).

On the other hand, the peers of Fabric v1.0 reach an agreement on the result as well, which Fabric v0.6.1 did not achieve, and have alternative consensus algorithm option called Kafka. Since Fabric v1.0 does not determine the order before running the transactions, multiple transactions are allowed to be run in parallel. When the results of the same transaction on different peers turn out to be inconsistent, Fabric v1.0 will abort and retry the transaction. Therefore, independent transactions can take advantage of Fabric v1.0's consensus policy, while interdependent transactions would rather suffer from the retry overhead.

3. Bottleneck Analysis

At first, we made sure that chaincodes were executed completely serially from the debug logging of the peer of Fabric v0.5. We also made sure that the execution time of chaincode_example02, the simplest sample chaincode for a zero-sum exchange transaction bundled in Fabric v0.5, was 2 ms. These facts indicate that the largest possible transaction rate for chaincode_example02 is around 500 TPS (= 1 s / 2 ms). Since the transaction rate we actually observed for the measurement was also around 500 TPS, we surmised that the chaincode execution was the main cause of the bottleneck at that time, and that decreasing the chaincode execution time would directly improve the transaction rate.

Moreover, we analyzed the debug logging of chaincode_example02 in micro-seconds on both the peer and the chaincode to reveal the main consumption of the execution time. We found that about 90% of the chaincode execution time was consumed by message transfers between the peer and the chaincode. Figure 2 shows the analysis of the chaincode execution time.

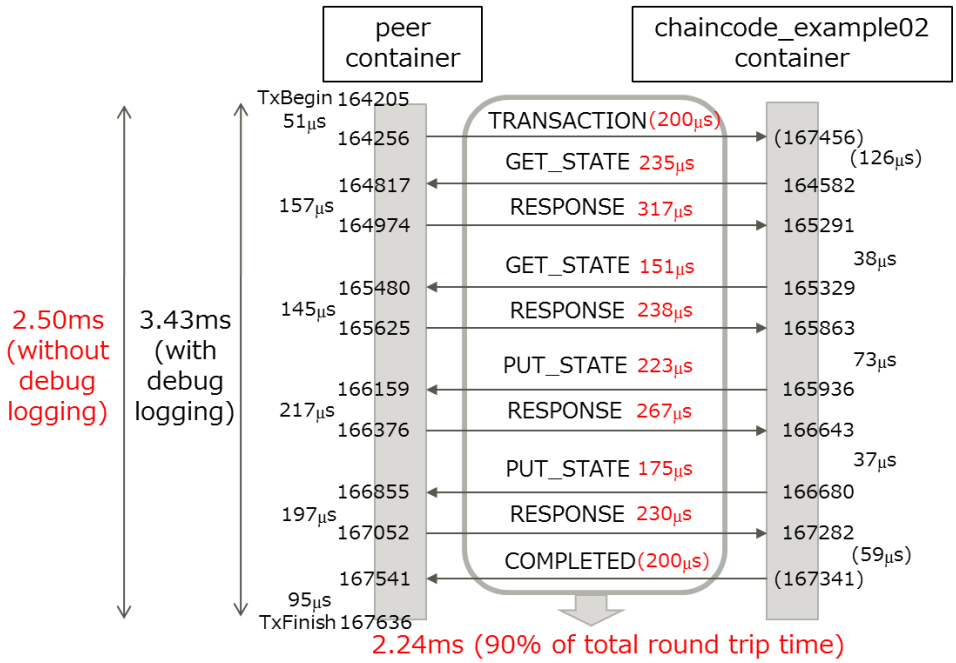

Figure 2: Analysis of chaincode execution time.

4. Our Proposed API Functions

In order to improve Fabric's transaction rate, we propose two new efficient API functions between the peer and the chaincode.

The first is the 'Differential Update State (DUS) API' or 'DiffUpdate()'. When a chaincode sends a predefined type of parameter set (key, operator, difference) to the peer, the peer calculates the updated value using the operator (e.g. A := A + X, A := A – X, A := A * X, A := A / X) and writes the value into the ledger. The advantage is that the chaincode does not have to get any state of the key in advance for updating the value. This improves by half of the number of the request messages on the existent API functions, which were required to use two messages (GetState() and PutState()). In case of chaincode_example02, only two DUS functions are sufficient because two sets of (GetState() and PutState()) can be replaced by them.

The second is the 'Compound Request (CR) API' or 'Compound()'. When a chaincode sends arbitrary set of request messages to the peer by this function, the peer execute all the requests in a specified order. CR supports multiple read functions, multiple write functions and their hybrid. The advantage is that the chaincode can reduce the number of requests to be issued. CR is especially suited for parameter read and parameter initialization. Since the peer handles the multiple requests as a transaction and issues rollback operation if the transaction fails, any rollback codes do not have to be written in each chaincode. In case of chaincode_example02, we can combine two DUS functions into one CR function and the number of messages can be decreased by half again. Considering another fixed round trip message pair (TRANSACTION / COMPLETED), the total number of the request messages are decreased from five to two. Therefore, we are expecting 2.5 times performance improvement by introducing DUS and CR.

We have implemented the prototype of DUS and CR on Fabric v0.6.1. Although the CR function itself is designed to be able to contain any kinds of function calls, current our CR implementation only supports PutState() and DUS. We are planning to support GetState() as well in the near future.

Figure 3 shows the behavior of DUS and CR. We will refer to the two new efficient API functions as 'revised API' in the remainder of this paper.

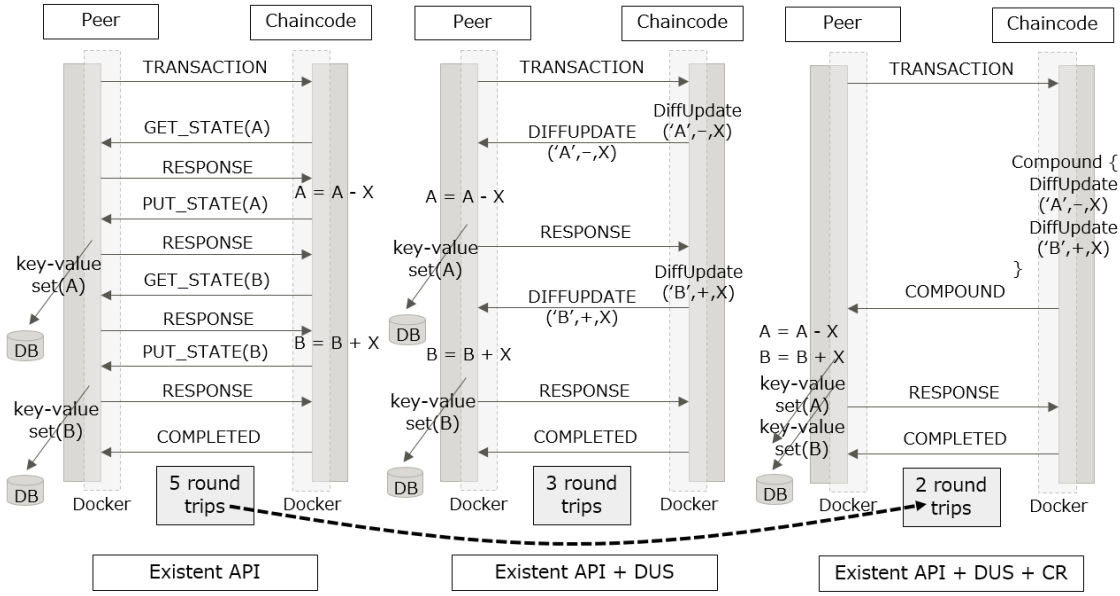

Figure 3: Behavior of two new efficient API functions.

5. Evaluation

5.1. Setup

Our Fabric environment consisted of four physical peer nodes, each configured with two 18-core Intel Xeon E5-2695v4 CPUs, 128 GB RAM, and a 1.8 TB 10k RPM hard disk drive. These nodes ran an Ubuntu Server 16.04 (Linux 4.4.0 kernel). The network environment was 1GbE.

We configured some of the TCP parameters in the Ubuntu Server, such as net.ipv4.tcp_tw_reuse = 1 and net.ipv4.tcp_fin_timeout = 5, because the number of TCP ports are generally fixed in advance and the ports have to be recycled as quickly as possible so that Fabric can receive a large amount of transaction requests in a second. If we left the parameters at the default values (net.ipv4.tcp_tw_reuse = 0, net.ipv4.tcp_fin_timeout = 60), used ports would not be released for 60 s and we could not use any heavy workload in our experiment (heavy workload would consume all the free ports in 60 s).

We used Fabric v0.6.1 for our performance evaluation. Each node was dedicated to running the Fabric peer and chaincode_example02. We did not use Fabric's Certificate Authority (CA) because we know it has its own bottleneck, which should be solved independently. We configured Fabric's parameter general.timeout.request to 86,400 s to avoid cancellation of the transactions which waited to be executed for a long while. If we left the parameter at the default value (general.timeout.request = 2s), the transactions waiting for more than 2 s since they were received by the peer could not ever be executed and we could not use any heavy workload in our experiment (heavy workload would make the transaction queue too long to be executed in 2 s).

5.2. Single Node Performance

First, we measured the Fabric's transaction rate with the existent API and our revised API on a single node. We calculate the transaction rate from the frequency of block generation, which means the transaction rate with finality. We attempted to evaluate the transaction rate under the various number of concurrent client processes. Figure 4 shows that the transaction rate reached its peak, 717 TPS, at 8 concurrent processes when we used the existent API. On the other hand, the transaction rate reached its peak, 1,417 TPS, at 16 concurrent processes when we used our revised API, which indicates that our revised API achieved 98% improvement under the condition of a single node. Since the expected transaction rate of our revised API was 2.5 times larger than that of the existent API, 98% improvement almost achieved the expectation. Moreover, we suggest that removing the containers between the peer and the chaincode might improve the transaction rate even more. However, that must be controversial because the removal of the containers would lead to a renouncement of some security features.

Just for reference, we will briefly show some important factors from the evaluation of Fabric v0.5. Although the default value of the parameter general.batchsize was 2, changing this parameter to 16 improved the transaction rate dramatically. Hyperledger community also knew it and adopted 500 as the new default value in v0.6.1. In addition to this parameter, Fabric v0.6.1 seemed to have changed something that resulted in the improvement of the transaction rate.

5.3. Four Node Performance

We also measured the transaction rate on the four physical peer nodes, distributing and leveling the concurrent client processes among the four nodes. Therefore, the x-axis for the measurement of the four nodes in Figure 4 indicates the sum of the concurrent processes among the four nodes. Figure 4 shows that the transaction rate reached its peak, 725 TPS, at 16 concurrent processes when we used the existent API. On the other hand, the transaction rate reached its peak, 1,350 TPS, at 32 concurrent processes when we used our revised API, which indicates that our revised API achieved 86% improvement under the condition of four nodes. Since the expected transaction rate of our revised API was 2.5 times larger than that of the existent API, 86% improvement almost achieved the expectation. We believe that the shift of the number of concurrent processes at which transaction rate reached its peak to a larger value was caused by the configuration with which the concurrent processes were distributed among the four nodes.

Although the blockchain's performance generally deteriorates as the number of nodes increases, the four node transaction rate achieved almost the same result as the single node, no matter which API we used. We observed over 1,000 hardware interrupts per second on each peer caused by the message transfers from the other peers during the consensus phase; however, this did not result in any performance deterioration, at least under the condition of four nodes.

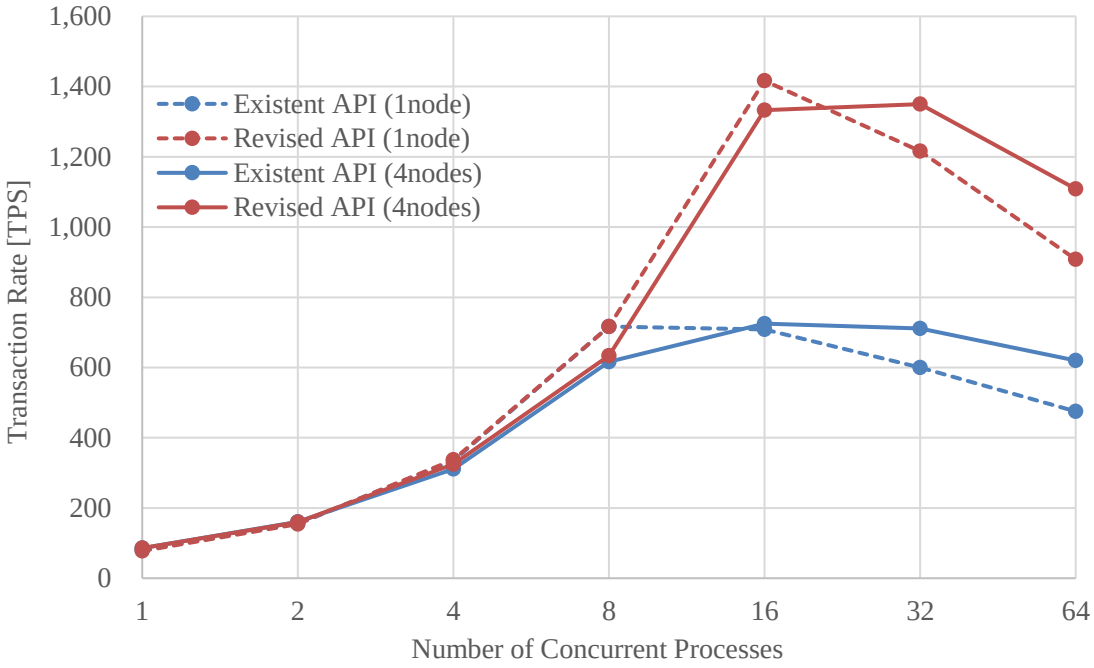

Figure 4: Performance comparison between the existent API and our revised API.

5.4. Influence of Network Latency

Finally, we evaluated the negative influence of network latency on the Fabric transaction rate when the nodes are located in various sizes of wide area network. To load each node with the network latency artificially, we used the Linux standard command named tc(8). We used the configuration in which the number of concurrent processes was fixed at 32 throughout this measurement.

The result was that the transaction rate deteriorated as the network latency increased. From this experiment, we determined some important facts about Fabric. First, until the network latency reaches 4 ms, almost equivalent to the distance between Tokyo and Shizuoka, nothing suffer from the latency. Second, until the network latency reaches 32 ms, almost equivalent to the distance between Tokyo and Hakata, the performance deterioration is relatively small and the advantage of our revised API remains. In other words, the performance bottleneck is still mainly caused by the inefficiency of the API functions between the peer and the chaincode. Finally, however, from a network latency of 64 ms and larger, almost equivalent to the distance between Tokyo and Shanghai, our revised API advantage disappears, and other causes of performance deterioration, including the consensus algorithm, seem to become dominant. Therefore, the Fabric running over a wide area network might be required to replace its consensus algorithm to improve its transaction rate. Figure 5 shows the performance comparison under the network latency.

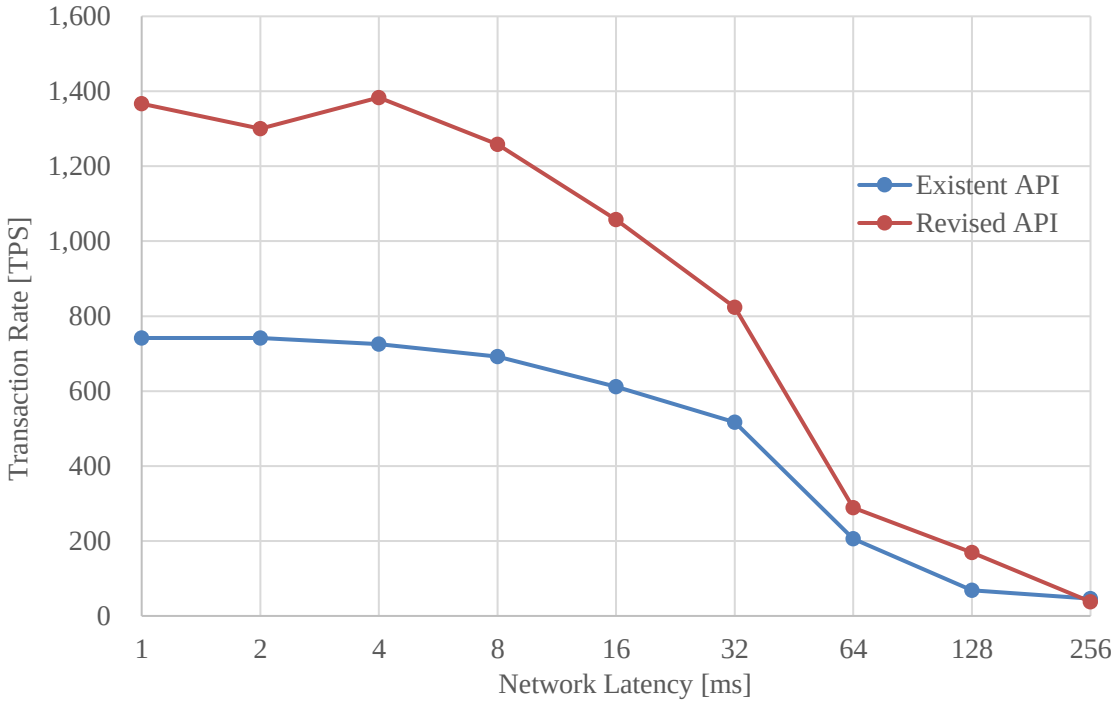

Figure 5: Performance comparison under network latency.

6. Conclusion

Blockchain itself does not offer performance advantage, but rather sacrifices performance to achieve tamper-resistance. However, the average transaction rates for blockchain technologies are continuously improving, and have already reached around 1,000-2,000 TPS, which is equivalent to the transaction rate for Zengin System, the common name for Japanese Bank's Payment Clearing Network.

In case of Fabric v0.6.1, since all the transactions are executed serially, the execution time of each transaction can often affect the transaction rate. We found that inefficient message transfer between the peer container and the chaincode container was the main cause of Fabric's performance bottleneck. In this paper, we showed that we could improve Fabric's transaction rate from 725 TPS to 1,350 TPS (86% increase) by introducing more efficient API between the containers.

From the experiment under artificial network latency, we determine that Fabric running over a wide area network might be required to replace its consensus algorithm to improve its transaction rate, because other causes of performance deterioration, including the consensus algorithm, seem to become dominant.

References

[1] Hoshino, T. 星野武史. Wakate-ga shiranai meinhureemu-to ginkoukeishisutemu-no rekishi & kisochishiki. 若手が知らないメインフレームと銀行系システムの歴史＆基礎知識. Retrieved from http://www.atmarkit.co.jp/ait/articles/1609/07/news007.html, September 12, 2016.

[2] Iga, D. 伊賀大記. Toushou Arouheddo tokushuu: 1-kai 2-yari-no kojintoushika-wa soutettaika. 東証アローヘッド特集:1 カイ 2 ヤリの個人投資家は総撤退か. Retrieved from http://jp.reuters.com/article/idJPJAPAN-13038920091221, December 21, 2009.

[3] arrowhead-no rinyuuaru-no gaiyou. “arrowhead のリニューアルの概要”. Retrieved from http://www.jpx.co.jp/files/tse/english/news/20/b7gje6000001zpm9-att/outline_of_renewal.pdf, June, 2014.

[4] Santo, A., et al. 山藤 敦史、他. Kinyuu-shijou-infra-ni taisuru bunsangata-daichogijutsu-no tekiyou-kanousei-ni tsuite. 金融市場インフラに対する分散型台帳技術の適用可能性について. Retrieved from http://www.jpx.co.jp/corporate/research-study/working-paper/tvdivq0000008q5y-att/JPX_working_paper_No15.pdf, August 30, 2016.

[5] Burokkucheen Kenkyuukai. ブロックチェーン研究会. Kokunai-no ginkoukan hurikomigyoumu-niokeru burokkucheen gijutsu-no jisshoujikken-ni kakawaru houkokusho. 国内の銀行間振込業務におけるブロックチェーン技術の実証実験に係る報告書. Retrieved from https://www2.deloitte.com/content/dam/Deloitte/jp/Documents/about-deloitte/news-releases/jp-nr-nr20161130-report.pdf, November 30, 2016.

[6] Visa Receives 2010 IBM Award for Innovation. Retrieved from https://www.visaeurope.com/newsroom/global-news/detail?id=1486724, October 25, 2010.

[7] Hoshi, A. bitFlyer-ga burokkucheen-gijyutsu Miyabi-wo happyou, shin-arugorizumu-to sumaato-contorakuto tousai. “bitFlyer がブロックチェーン技術 Miyabi を発表、新アルゴリズムとスマートコントラクト搭載”. Retrieved from http://jp.techcrunch.com/2016/12/21/bitflyer-announces-private-blockchain-technology-miyabi/, December 21, 2016.

[8] Tech Bureau, et al. テックビューロ株式会社、他. Sakura-intaanetto, Tech Beureu, Arara-ga daikibona denshimanee-kanjyou-shisutemu-heno jitsuyou-wo zenteitoshita burokkucheen tekiyou-jikken-ni seikou. さくらインターネット、テックビューロ、アララが大規模な電子マネー勘定システムへの実用を前提としたブロックチェーン適用実験に成功. Retrieved from https://www.sakura.ad.jp/press/2016/1220_blockchain/?_ga=1.217350678.1026183481.1485751049, December 20, 2016.

[9] Hyperledger Fabric. Retrieved from https://github.com/fabric/fabric.

[10] Tozawa, A., et al. 戸澤 晶彦、他. Hyperledger/Fabric no seinou-kaiseki. “Hyperledger/Fabric の性能解析”, Information Processing Society of Japan SIGPRO Workshop, October 27, 2016.

[11] IBM Launches Industry’s Most Secure Enterprise-Ready Blockchain Services for Hyperledger Fabric v1.0 on IBM Cloud (IBM Press Release). Retrieved from http://www-

03.ibm.com/press/us/en/pressrelease/51840.wss, March 20, 2017.

[12] Hyperledger Iroha. Retrieved from https://github.com/hyperledger/iroha.

[13] Soramitsu Co., Ltd. ソラミツ株式会社. Soramitsu-no kaihatsu-shita burokkucheen “Iroha” ga, “Hyperledger project” no Incubation-ni sekai-de 3-banme-ni judaku saremashita. ソラミツの開発したブロックチェーン「いろは（Iroha）」が、「Hyperledger プロジェクト」の Incubation に世界で3番目に受諾されました. Retrieved from https://prtimes.jp/main/html/rd/p/000000006.000019078.html, November 2, 2016.

[14] Hyperledger Sawtooth Lake. Retrieved from https://github.com/hyperledger/sawtooth-core.

[15] C. Gutierrez. Hyperledger’s Sawtooth Lake Aims at a Thousand Transactions per Second. Retrieved from https://www.altoros.com/blog/hyperledgers-sawtooth-lake-aims-at-a-thousand-transactions-per-second/, March 13, 2017.

[16] Ethereum wiki page : Consortium Chain Development. Retrieved from https://github.com/ethereum/wiki/wiki/Consortium-Chain-Development

[17] K. Weare. Enterprise Ethereum Alliance Releases Vision Paper. Retrieved from https://www.infoq.com/news/2017/03/Enterprise-Ethereum-Vision, Mar 12, 2017.