Testing and Comparative Analysis of the F-BFT-based DLT Solution

Alexander Bogdanov[1], Nadezhda Shchegoleva[1(✉)], Vladimir Korkhov[1,4], Valery Khvatov[2], Nodir Zaynalov[3], Jasur Kiyamov[1], Aleksandr Dik[1], and Anar Faradzhov[1]

[1] Saint Petersburg State University, St. Petersburg, Russia
{a.v.bogdanov,n.shchegoleva,v.korkhov}@spbu.ru,
{st087383,st087383,st069744}@student.spbu.ru
[2] DGT Technologies AG, Toronto, Canada
[3] Samarkand branch Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Samarkand, Uzbekistan
nodirz@mail.ru
[4] Plekhanov Russian University of Economics, Moscow, Russia

Abstract. Consensus algorithm is a crucial part of a blockchain system. In particular, in any blockchain-based Distributed Ledger Technology (DLT) solution the consensus algorithm plays the key role in maintaining consistency of databases. The choice of consensus type during design of the distributed system will inevitably and significantly affect such characteristics as throughput and network load what in turn can impose serious restrictions on functionality or applicability of the whole system. In this research paper we overview the most commonly used algorithms and approaches, the highest attention is paid to those based on the solution of the Byzantine Fault Tolerance problem (BFT). One of the best-known algorithms of that type is the Practical Byzantine Fault Tolerance (PBFT) which has become the basis and the benchmark for a plenty of different BFT algorithms. The focus of this paper is sharpened on a particular PBFT modification known as the Federalized BFT. The aim of this study is to investigate a particular F-BFT-based DLT solution, to compare it with the other systems of that type and to determine areas of its applicability.

Keywords: Blockchain · Distributed ledger technology · Smart contracts

1 Introduction

1.1 Background

The modern business trends are directed towards integration and further complication of the IT environments. In these circumstances the need in universal solutions which could be compliment with higher security requirements and, at the same time, could be universal and flexible has risen as much as never before.

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12952, pp. 31–41, 2021.
https://doi.org/10.1007/978-3-030-86973-1_3

The situation is aggravated by the fact that today businesses have a tendency to move their infrastructure into clouds. It is understandable, despite all the complications that migration to cloud environment entails, it helps to optimize operational expenses, deployment costs, and costs of exploitation quite effectively [1,2]. Fast spreading of cloud-based services has inspired interest to highly-scalable blockchain systems, their possibilities and applicability in different business environments.

Unfortunately, there are still a lot of obstacles for blockchain deployment in the cloud. The most notable one is performance instability of blockchain based solutions in dynamic cloud environments which is also worsened by complications of influence factors identification.

A key role in controlling this instability plays the choice of consensus algorithm that would be most appropriate for the purpose in consideration.

Nowadays there are plenty of different approaches to designing consensus algorithms. For instance, in Bitcoin and Ethereum Proof-of-Work [4] protocol is implemented. The main idea of this algorithm is based on the idea that only those nodes who have done some computational work are allowed to create a new block in the blockchain. Another well-known specimen is Proof-of-Stake [5] protocol implemented in Nxt [6] and many other cryptocurrencies. The idea of this one is in giving the right for creating a new block to the stakeholders of a currency only.

Even though the diversity of different consensus algorithms is high, they all impose different expenditures and restrictions on participants. One of well researched approaches to construction consensus protocols is based on Byzantine Fault Tolerance problem [9] solution. The BFT-based algorithms family includes: DBFT [10], FBFT [11], RBFT [12], PBFT [13] and many others. This study is focused on comparison of two DLT-products based on different BFT algorithms: Hyperledger Sawtooth [17] and BGX/DGT [16], which are highly similar, but use different BFT-based consensus protocols.

1.2 BGX/DGT

BGX/DGT is the DLT solution based on Hyperledger Sawtooth framework. It inherits most of the architectural and technological concepts applied in Sawtooth of version 1.1. Nevertheless, BGX/DGT differs from the original Sawtooth in the most crucial parts. Unlike Sawtooth, BGX/DGT is not a p2p network, it is based on the federated structure of clusters—separated p2p networks, and the ring of arbiters—the second layer which connects clusters with each other. Therefore, p2p based consensus protocols are incompatible with such a network structure, that is one of the reasons why BGX/DGT uses two-layered F-BFT consensus [8] to keep its database consistent (see Fig. 1).

As in Sawtooth, the node in BGX/DGT consists of several components that communicate with each other using TCP protocol. The central component of any node is the validator, which functions are: providing interoperability for other modules on the particular node, communicating with other nodes, authenticating and identity checking for nodes in the network.

Another important component is the REST-API module— it provides an interface for users to interact with the system. REST-API allows users to access the validator using simple HTTP queries.

The consensus engine in BGX/DGT is also presented as the separated module.

Besides these three components, this product uses different transaction processors to set up and sustain the environment as well as to implement user-defined business-logic.

Several distinguishing features should be emphasized [7]:

- The storage system in BGX/DGT has a structure of Directed Acyclic Graph what makes it similar with IOTA, Hedera Hashgraph, Orumesh, DagCoin, Byteball, Nano. But unlike aforementioned platforms, BGX/DGT is designed in the way that allows voting in federalized networks with dynamic topology.
- There are several solutions that use similar federative approach e.g. Ripple, Stellar. The combination of DAG and federative voting methods together with leaders rotation and dynamic topologies makes BGX/DGT notably more horizontally scalable, but not less secure.
- Consortium based consensus helps to preserve flexibility along with not losing interoperability. In comparison with, for instance, Hyperledger Fabric which is designed to be used in private networks only and ICON which have to use a special Loopchain Fault Tolerance mechanism to interact with networks on the side, usage of dynamic topologies together with DAG-based storage in BGX/DGT provides a significant asynchronicity. This fact ensures a significant level of interoperability (Table 1).

1.3 F-BFT Algorithm Description

When the system is up, it should choose the higher rank nodes - Leaders and Arbiters. In each cluster there should be exactly one Leader (see Fig. 2, I).

Once the transaction is initiated by any node, the initiator's node publishes this transaction to the whole network, unless the transaction is incorrect (Fig. 2, II). When initiator's cluster nodes receive the transaction, they verify it and, in case everything is correct, they commit it using PBFT consensus algorithm. Meanwhile the rest of the nodes should receive that transaction too, but they have to verify it only, keeping committing pending until they receive the permission from arbiters. The next phase of the algorithm comes up when the leader of the initiator's cluster receives $\frac{2n+1}{3}$ confirmation messages, where n is the number of nodes in initiator's cluster. At that moment the leader sends a special message to arbiters (Fig. 2.2, III) who in turn, in case the corresponding transaction have been verified successfully, send permission for committing to other nodes(Fig. 2, IV).

Table 1. Blockchain platforms' functionality comparison

Parameter	Ethereum	Stellar	Hyperledger fabric	BGX/DGT
Purpose	Distributed processing, cryptocurrencies	Payment system	Data exchange within corporate environment	Real time data integration, ecosystems
Networking	P2P, public	Federated, public	P2P, public	Federated, consortium based
Cryptocurrency supported	Yes	Yes	No	Optional
Consensus	PoW	FBA	PBFT	F-BFT
Data storage	Blocks	Blocks	Blocks	DAG
Tokenization	Yes	No	No	Yes
Smart contract	Yes	Yes	Yes	Yes
Encryption	ECDSA	Asymmetric, Ed25519	PKCS11, pluggable	Asymmetric, ECDSA curve secp256k1
Programming language	GO	Java	Java, JavaScript (Node.js), GO	C++, Python

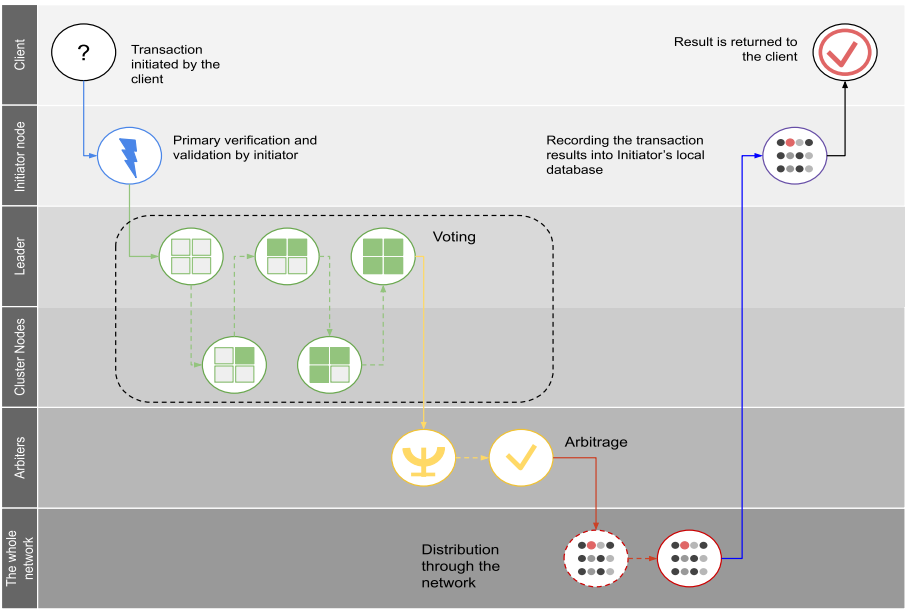

Fig. 1. Transaction committing in BGX/DGT

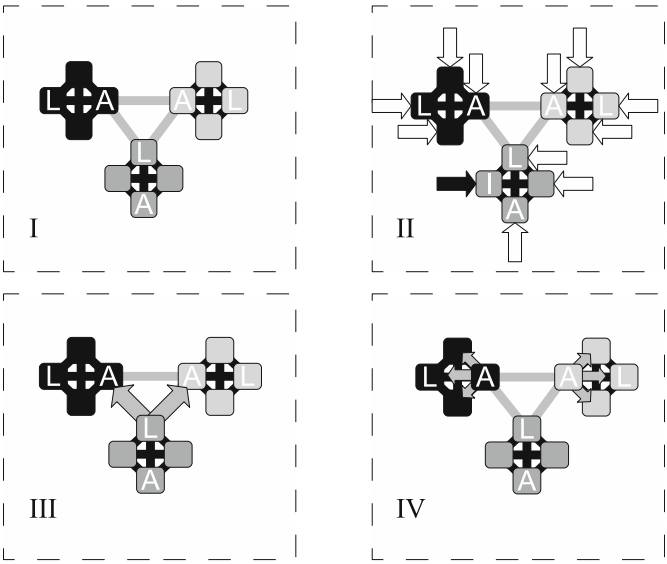

Fig. 2. F-BFT consensus illustration

2 Testing

The following section contains the testing results for several scenarios which will inevitably occur during exploitation of the system. The testing was conducted to examine weaknesses of the current version of BGX/DGT in cases unspecific for different applicability areas, as well to assess competitiveness of it in the current stage of development.

2.1 Testing Environment

The environment configuration was used for testing is listed below:

- OS: Fedora 28
- CPU: Intel(R) Xeon(R) CPU E5-2690 v4, 2.60 GHz
- RAM: 255 GB

Tests were conducted using Docker [14] and Docker-compose [15] for creating test networks. All the measurements were taken using Docker tools, vnStat and embedded tools of Hyperledger Sawtooth.

Tested version of the product was Kawartha [16].

A typical node in BGX/DGT network consists of several components:

- transactions processor
- validator module
- shell module
- consensus module

– settings module
– topology processor
– REST API module

Each component was running in a separated docker container in order to remain isolated from the others. Hyperledger Sawtooth [17] was chosen as the benchmark to compare with. This choice was dictated by several reasons. First, BGX/DGT is the modification of Hyperledger Sawtooth, hence it inherits almost all technologies and concepts used in Hyperledger Sawtooth except consensus algorithm and transaction storage. And second, Hyperledger Sawtooth has pluggable PBFT [18] consensus algorithm which is the nearest relative for F-BFT consensus developed for BGX/DGT.

2.2 Testing Method

Tests. A bunch of tests for this study describe four different situations that can occur during system exploitation and does not include situations of intended attacks on cryptographic protection nor taking control over one or several nodes of the network by intruder:

1. Single transactions committing. During this test transactions were made one at a time from each node. The next transaction was not sent until the previous had been committed.
2. Asynchronous streams of transactions committing. Transactions were sent in bunches of 100 from different nodes in different clusters (through one node in cluster at the same time).
3. Simultaneous transactions committing test I. As in the second, transactions were sent in bunches of 100 from different nodes in different clusters, but this time streams were synchronized – transactions were sent simultaneously.
4. Simultaneous transactions committing test II. A bunch of 100 transactions was sent simultaneously from every node.

Tests were taken in five different configurations:

1. One cluster of 6 nodes,
2. Two clusters of 6 nodes,
3. Two clusters of 6 nodes and one cluster of 3,
4. Two clusters of 6 nodes and two clusters of 3,
5. Two clusters of 6 nodes and three clusters of 3.

It should be kept in mind that the first and the second clusters had 6 nodes, while the third, fourth and fifth consisted of 3 only—this configuration was static and had never been changed during testing.

The explanation of why only those five configurations were chosen to conduct tests is due to several technical issues of BGX/DGT which impose restrictions on scalability of the network. The second reason is the fact that such a size of the network and its scaling dynamics was enough to see differences between

BGX/DGT and the benchmark, and to establish the most crucial shortcomings of the current version (Kawartha) of the product in consideration.

In order to unify terminology we assume that the benchmark also has the federated structure, therefore in new terms the first six nodes will belong to the first cluster, the second six to the second cluster, 13, 14, 15 nodes—to the third, and the last 6 nodes will compose the fourth and the fifth clusters correspondingly.

Metrics. The benchmark and BGX/DGT were compared using two main metrics: committed transactions per second (CTPS) and the average volume of network traffic on validator per transaction (VONT):

$$\text{CTPS} = \frac{\text{committed transactions}}{\text{time elapsed}},$$

$$\text{VONT} = \frac{\text{average input volume} + \text{average output volume}}{\text{committed transactions}}.$$

Committed transactions are transactions included in databases in ALL nodes.

Elapsed time is the time elapsed since sending the first transaction until all transactions are committed.

Input/output volume was measured within the same period as the number of committed transactions.

In order to make results more precise we took two additional metrics: average resting state volume (ARSV) – an average volume of network traffic for sustaining network (without committing transactions), and committing rate (CR):

$$\text{ARSV} = \frac{\text{60 sec. average input} + \text{60 sec. average output}}{\text{60 sec.}}$$

$$\text{CR} = \frac{\text{transactions committed}}{\text{transactions sent}},$$

Here, 60 s time interval was chosen as it ensures that no unique message will remain unaccounted, at the same time, it is short enough for the test to be conducted several times.

As BGX/DGT is not a p2p network, ARSV was calculated separately for two kinds of nodes—leader nodes and ordinary nodes. On the contrary, Sawtooth PBFT builds up a p2p network, hence there is no need in calculation of two different ARSV values.

3 Results

Testing results are shown in the following tables (see Tables 2, 3).

The results of Test 1 show that the speed of committing standalone transactions for BGX/DGT is considerably higher then for Sawtooth PBFT, moreover, the dynamics of the traffic volume ingested by a single node for BGX/DGT is much flatter than for Sawtooth PBFT.

Table 2. Testing results

Num. of nodes	Sawtooth PBFT			**BGX/DGT**		
Test 1. Single transactions committing.						
	CTPS, tps	VONT, kb/tr	CR	CTPS, tps	VONT, kb/tr	CR
6	0.85	81.2	1	0.62	226	1
12	0.8	168	1	1.2	162.7	1
15	0.83	212.8	1	1.07	207	1
18	0.62	293.1	1	1.09	234.9	1
21	0.87	340	1	1.18	206	1
Test 2. Asynchronous streams of transactions committing						
6	0.74	66	1	1.23	180.8	1
12	1.85	109.2	1	1.49	185	0.57
15	2.43	121.6	1	1.72	142.6	0.49
18	4.54	108.8	1	1.09	209	0.36
21	5.55	115	1	0.81	232.7	0.28
Test 3. Simultaneous transactions commiting test I.						
12	0.8	109.2	1	–	–	0
15	3.13	110	1	–	–	0
18	3.8	123.4	1	–	–	0
21	4.76	112.6	1	–	–	0
Test 4. Simultaneous transactions commiting test II.						
6	0.87	42.5	1	–	–	0
12	1.18	101.6	1	–	–	0
15	2.78	75.8	1	–	–	0
18	3.45	103.8	1	–	–	0
21	3.85	129.9	0.95	–	–	0

Table 3. Testing results

Num. of nodes	Sawtooth PBFT	BGX/DGT	
	ARSV, kb/sec.	ARSV Lead, kb/sec.	ARSV Ord., kb/sec.
6	1.59	32.42	10.08
12	1.59	25.26	8.53
15	1.43	26.28	15
18	1.59	24.9	17.06
21	1.36	21.16	17.07

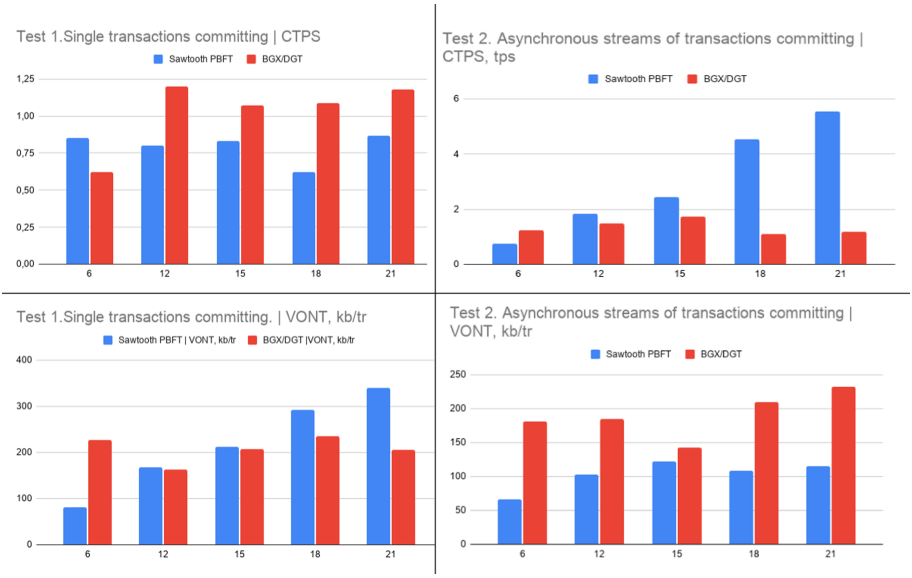

Fig. 3. Results of tests 1 and 2

In higher loads, with the presence of several parallel transaction streams, results shown by BGX/DGT are much more ambiguous. CTPS shows that the throughput of the system does not rise when the load increases, while Sawtooth shows quite a steady rising dynamics. Also, Sawtooth works much more efficient in terms of VONT when the load increases. It might be happening due to different storage models that BGX/DGT and Sawtooth use to store transactions. While Sawtooth unites simultaneous transactions in blocks and stores blocks, BGX/DGT stores standalone transactions in its databases so it is unable to process more then one transaction at the same time. This hypothesis is supported also by the dynamics of CR in Test 2: as streaming density increases, more transaction collisions happen and more of them left unprocessed. Results of Tests 3 and 4 speak in favor of this hypothesis too.

4 Conclusion and Future Work

In this paper, we presented results of implementing several testing scenarios of BGX/DGT and Hyperledger Sawtooth DLT platforms with comparable consensus algorithms used. Regarding the results, it is possible to surmise the main areas of applicability for BGX/DGT Kawartha. For now it might be applied to highly distributed but low loaded systems. For systems of higher reliability Sawtooth PBFT is more preferable.

Even though the current version of BGX/DGT performs poorly in dealing with parallel workloads, it remains effective in terms of horizontal scalability, what is a promising result in the current development stage.

Implemented testing scenarios do not model the whole plenty of possible situations distributed system may encounter during its lifetime. Both systems' behavior in situations of higher loads and cases of faulty transactions and faulty nodes are yet to be analysed.

References

1. Jeffery, K., et al.: Challenges emerging from future cloud application scenarios. Procedia Comput. Sci. **68**, 227–237 (2015). https://doi.org/10.1016/j.procs.2015.09.238
2. Koulouzis, S., Martin, P., Zhou, H., et al.: Time-critical data management in clouds: challenges and a dynamic real-time infrastructure planner (DRIP) solution. Concurrency Comput. Pract. Exper. **32**(16), e5269 (2019). https://doi.org/10.1002/cpe.5269
3. Ayed, A.: A conceptual secure blockchain-based electronic voting system. Int. J. Netw. Secur. Appl. (IJNSA) **9**(3), 1–9 (2017). https://doi.org/10.5121/ijnsa.2017.9301
4. Jakobsson, M., Juels, A.: Proofs of work and bread pudding protocols (extended abstract). In: Preneel, B. (ed.) Secure Information Networks. ITIFIP, vol. 23, pp. 258–272. Springer, Boston (1999). https://doi.org/10.1007/978-0-387-35568-9_18
5. Li, W., Andreina, S., Bohli, J.-M., Karame, G.: Securing proof-of-stake blockchain protocols. In: Garcia-Alfaro, J., Navarro-Arribas, G., Hartenstein, H., Herrera-Joancomartí, J. (eds.) ESORICS/DPM/CBT -2017. LNCS, vol. 10436, pp. 297–315. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67816-0_17
6. Nxt technical documentation: https://nxtdocs.jelurida.com Accessed 11 May 2021
7. Bogdanov, A., Degtyarev, A., Uteshev, A., Shchegoleva, N., Khvatov, V., Zvyagintsev, M.: A DLT based innovative investment platform. In: Gervasi, O., et al. (eds.) ICCSA 2020. LNCS, vol. 12251, pp. 72–86. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58808-3_7
8. Bogdanov, A., Uteshev, A., Khvatov, V.: Error detection in the decentralized voting protocol. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11620, pp. 485–494. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24296-1_38
9. Lamport, L., Shostak, R., Pease, M.: The Byzantine general problem. ACM Trans. Programm. Lang. Syst. **4**, 382–401 (1982)
10. Wang, Q., Yu, J., Peng, Z., Bui, V.C., Chen, S., Ding, Y., Xiang, Y.: Security analysis on dBFT protocol of NEO. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 20–31. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_2
11. Malkhi, D., Nayak, K., Ren, L.: Flexible Byzantine fault tolerance. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1041–1053 (2019). https://doi.org/10.1145/3319535.3354225
12. Aublin, P., Mokhtar, S., Quéma, V.: RBFT: redundant byzantine fault tolerance. In: Proceedings - International Conference on Distributed Computing Systems, pp. 297–306 (2013). https://doi.org/10.1109/ICDCS.2013.53
13. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: Proceedings of the third symposium on Operating systems design and implementation (OSDI 1999), pp. 173–186. USENIX Association, USA (1999)
14. Overview of Docker Engine. https://docs.docker.com/engine/ Accessed 15 Mar 2021

15. Overview of Docker Compose. https://docs.docker.com/compose/ Accessed 15 Mar 2021
16. BGX/DGT repository. https://github.com/DGT-Network/DGT-Kawartha Accessed 17 Mar 2021
17. Hyperledger Sawtooth repository. https://github.com/hyperledger/sawtooth-core Accessed 15 Mar 2021
18. Introduction to Sawtooth PBFT. https://sawtooth.hyperledger.org/docs/pbft/releases/latest/introduction-to-sawtooth-pbft.html Accessed 15 Mar 2021