COMPARATIVE ANALYSIS AND APPLICABILITY DETERMINATION FOR SEVERAL DLT SOLUTIONS

A. Bogdanov[1], V. Korkhov[1,3], N. Shchegoleva[1], V. Khvatov[4], N. Zaynalov[2], J. Kiyamov[1], A. Dik[1], A. Faradzhov[1,a]

[1] *St Peterburg university, Russia*

[2] *Samarkand branch Tashkent university of information technology, Uzbekistan*

[3] *Plekhanov Russia university of economics, Russia*

[4] *DGT Technologies AG, http://dgt.world/*

E-mail: [a] st069744@student.spbu.ru

Abstract: Potential benefits of implementation of distributed ledger technology are widely discussed among different business actors and governmental structures. Within the last decade, with growing popularity of blockchain-based payment systems and cryptocurrencies, these discussions considerably sharpened. Therefore, an extensive body of research has emerged on this soil. The goal of this study is to attempt to make a comparative analysis of several existing blockchain-based distributed ledger platforms. Besides that, authors overview the most commonly used consensus algorithms and design approaches, as for any blockchain product, consensus algorithm is a crucial part which determines the performance of the overall system. Choosing the right algorithm would ensure high reliability and throughput, while the wrong choice could cause fatal malfunctions for the application. A suitable algorithm usually should be chosen according to the task in consideration, e.g. Nakamoto-style protocols could be considered better for public networks, while multiround voting protocols are more suitable for private and secure systems. The highest attention is paid to consensus algorithms based on the solution of the Byzantine Fault Tolerance problem (BFT).

Keywords: Byzantine Fault Tolerance problem, distributed ledger technology, multiround voting protocols

1. Introduction

Current business trends are aimed at integrating and further complicating the IT environment. In these conditions, the need for universal solutions that can meet higher security requirements and at the same time can be versatile and flexible has grown like never before.

To make matters worse, businesses today tend to migrate their infrastructure to the cloud. This is understandable, despite all the difficulties that migration to the cloud entails, it helps to effectively optimize operational costs, deployment costs and operating costs [1,2]. The proliferation of cloud services has sparked interest in highly scalable blockchain systems, their capabilities and applicability in various business environments.

Unfortunately, there are still many obstacles to deploying blockchain in the cloud. The most notable of these is the performance instability of blockchain-based solutions in dynamic cloud environments, which is also exacerbated by the difficulty of identifying enablers.

The key role in managing this instability is played by the choice of the consensus algorithm that is most suitable for the considered purpose.

While the variety of different consensus algorithms is great, they all impose different costs and constraints on the participants. One well-studied approach to building consensus protocols is based on the solution to the Byzantine fault tolerance problem. This study focuses on comparing two DLT products based on different BFT algorithms: Hyperledger Sawtooth and BGX/DGT, which are very similar but use different BFT-based consensus protocols.

The special interest of authors is centered around novel Federated Byzantine fault tolerance protocol(F-BFT) which, as it is claimed, should outperform classical BFT algorithms, such as PBFT, in terms of horizontal scalability.

Within the framework of the current research we tried to estimate capabilities of the latest version of BGX/DGT's consensus module based on FBFT. In order to do that we conducted several tests and compared BGX/DGT performance with the benchmark.

The results of conducted tests would help to reveal strengths and weaknesses of the particular F-BFT-based DLT and, therefore, to determine its applicability in the real systems.

2. Considered platforms

Hyperledger Sawtooth [3] is built to be an open source distributed ledger for the modern enterprise. Unlike many popular blockchains, Sawtooth is not built for cryptocurrency, but instead for business supply chain management. The transaction flow begins with the client placing all transactions into a block, and then signing the batch and sending it to a validator. The validator uses its transaction processor to ensure the integrity of the batch, and then commits it. Sawtooth executes transactions in parallel, instead of in serial, when possible through a REST API to improve performance. It also contains the novel feature of being modular, which includes consensus algorithms, rule sets, coding language, and smart contracts. This allows it to efficiently change depending on the business need. Programmers can use Python, JavaScript, Go, C++, Java, and Rust to build and interact with the Sawtooth blockchain. Currently, four different consensus algorithms are supported by Sawtooth. These are Dev_mode, PoET, RAFT and PBFT. Dev_Mode is a random generator algorithm used purely for developer testing. Proof of Elapsed Time (PoET) is built specifically for Sawtooth and does not follow byzantine fault tolerance (BFT), allowing it to reach higher throughput than other models.

BFT is a type of system's behavior in case of a malicious actor. It can be defined as the maximal fraction of faulty nodes the distributed system can tolerate. The basic theory behind this is explained by Lamport et al. [4].

RAFT is an election-style leader-based algorithm where each node can become a leader candidate if it does not hear back from a current leader after a certain amount of time. Candidate then requests votes from other nodes, and if it gets more than half of the votes, it becomes a new leader. The leader also has the job of replicating the new log to all other nodes to maintain consistency.

The last consensus algorithm implemented in Sawtooth is PBFT which was proposed by Castro and Liskov [5]. This is a classical four-stage election-style Byzantine fault tolerant algorithm.

BGX/DGT [6] is a DLT solution based on the Hyperledger Sawtooth platform. It inherits most of the architectural and technological concepts used in Sawtooth version 1.1. However, BGX/DGT differs from the original Sawtooth in the most crucial ways. BGX/DGT doesn't imply consensus module plugability. It can use Federated Byzantine fault tolerance (F-BFT) algorithm only. F-BFT is the implementation of PBFT algorithm in networks with federated structure (a set of nearly-independent clusters).

3. Testing

3.1 Configuration

The following section presents test results for several scenarios that will inevitably occur during system normal operation. The testing was carried out in order to investigate the weak points of the current version of BGX / DGT in cases that are not specific to various fields of application, as well as to assess its competitiveness at the current stage of development.

The environment configuration was used for testing is listed below:

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

Tests were conducted using Docker and Docker-compose. All metrics were measured using Docker tools, vnStat and embedded tools of Hyperledger Sawtooth.

Tested version of the product was Kawartha. 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 run in a separate docker container to stay isolated from the others. Hyperledger Sawtooth was chosen as a benchmark for comparison. This choice was dictated by several reasons. Firstly, BGX / DGT is a modification of Hyperledger Sawtooth, so it inherits almost all technologies and concepts used in Hyperledger Sawtooth, with the exception of the consensus algorithm and transaction store. Secondly, Hyperledger Sawtooth has a pluggable PBFT consensus algorithm, which is the closest relative of the F-BFT consensus developed for BGX / DGT.

3.2 Scenarios

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:

- 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.
- 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).
- 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.
- Simultaneous transactions committing test II. A bunch of 100 transactions were sent simultaneously from every node.

Tests were taken in five different configurations:

- One cluster of 6 nodes,
- Two clusters of 6 nodes,
- Two clusters of 6 nodes and one cluster of 3,
- Two clusters of 6 nodes and two clusters of 3,
- 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.

3.3 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).

3.4 Results

As It may be seen on diagram (Figure 1), in Test 1 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.

In case of 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 steady rising dynamics. Also, Sawtooth works much more efficiently 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 them, BGX/DGT stores standalone transactions in its databases so it is unable to process more than one transaction at the same time.

Test 3 and Test 4 were completely failed by BGX/DGT - there were no committed transactions, therefore, results are not shown on the diagram.

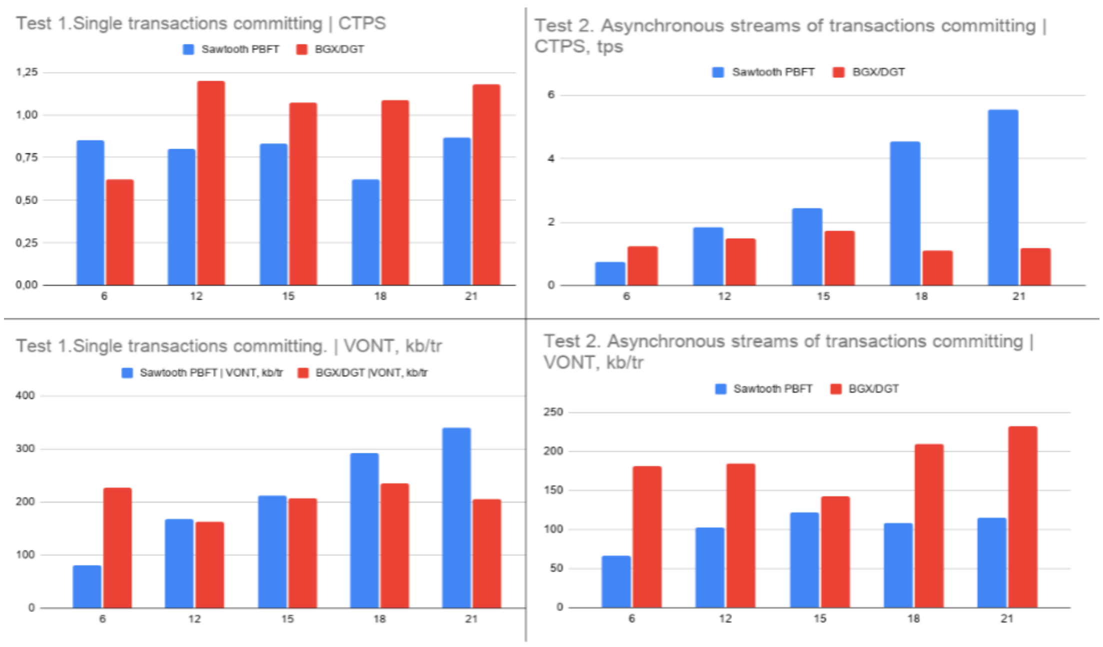

Figure 1. Testing results

4. Conclusion

In this article, we have presented the results of implementing multiple test scenarios on BGX / DGT and Hyperledger Sawtooth. As far as the results are concerned, the main areas of applicability of BGX / DGT Kawartha can be assumed. For now, this can be applied to highly distributed but lightly loaded systems. For higher reliability systems, Sawtooth PBFT is preferred.

Should be mentioned that BGX/DGT is in the active development currently, and weaknesses revealed in this research would be eliminated in the future.

References

[1] Jeffery, K., Kousiouris, G., Kyriazis, D., Altmann, J., Ciuffoletti, A., Maglo-giannis, I., Nesi, P., Suzic, B., Zhao, Z.: Challenges Emerging from Future Cloud Application Scenarios. Procedia Computer Science. 68. 227–237 (2015). DOI: 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 Computat Pract Exper. 2020; 32:e5269.(2019). DOI: 32.10.1002/cpe.5269

[3] Hyperledger Sawtooth repository https://github.com/hyperledger/sawtooth-core (Last accessed 15.06.2021)

[4] Lamport, L., Shostak, R., Pease, M.: The Byzantine general problem. ACM Trans. Programm. Lang. Syst.. 4. 382-401. (1982)

[5] Castro, M., Liskov, B.: Practical Byzantine Fault Tolerance. In Proceedings of the third symposium on Operating systems design and implementation (OSDI ’99).USENIX Association, USA, 173–186. (1999)

[6] BGX/DGT repository https://github.com/DGT-Network/DGT-Kawartha (Last accessed 17.06.2021)