Visualization of consensus mechanisms in PoS based blockchain protocols

Daniil Baldouski
University of Primorska
Koper, Slovenia
d.baldovskiy@mail.ru

Aleksandar Tošić
University of Primorska
Koper, Slovenia
Innorenew CoE
Izola, Slovenia
aleksandar.tosic@upr.si

ABSTRACT

In the past decade, decentralized systems have been increasingly gaining more attention. Much of the attention arguably comes from both financial, and sociological acceptance, and adoption of blockchain technology. One of the frontiers has been the design of new consensus protocols, topology optimisation in these peer to peer(P2P) networks, and gossip protocol design. Analogue to agent based systems, transitioning from the design to implementation is a difficult task. This is due to the inherent nature of such systems, where nodes or actors within the system only have a local view of the system with very little guarantees on availability of data. Additionally, such systems often offer no guarantees of a system wide time synchronisation. This research offers insight into the importance of visualisation techniques in the implementation phase of vote based consensus algorithms, and P2P overlay network topology. We present our custom visualisations, and note their usefulness in debugging, and identifying potential issues in decentralized networks. Our use case is an implementation of a blockchain protocol.

KEYWORDS

Grafana, visualisation, consensus mechanism, blockchain protocols, P2P, overlay network

1 INTRODUCTION

Distributed systems are notoriously difficult to inspect and their problems difficult to identify. The difficulty stems from the fact that predominant issues can be stochastic and difficult to reproduce, and from the inability to easily observe, compare, and test multiple programs running on separate machines at the same time. Another important aspect in distributed systems is that they inherently make heavy use of the network. The use of various network protocols imposes additional complexity, which increases the search space in identifying bugs. In recent years, distributed systems have been gaining more attention both in academia and private sector. This increasing interest can be largely attributed to the rapid development of distributed ledger technology, and blockchain. In recent years, many new consensus mechanisms, blockchain protocols, network protocols, improvement in gossip protocols have been proposed. Many of them are transitioning from a theoretical framework to a practical implementation. However, public distributed ledgers (or distributed ledger technology or DLT) and blockchains secure their consensus mechanisms and provide spam resistance through the use of tokens representing value. The use of digital value within the protocol enables the protocol to enforce a level of security through economic incentives, and game theoretical aspects that make most attack vectors economically infeasible or impractical for the attacker. A good example of this is the Proof of Stake (PoS) consensus mechanism, where nodes in the decentralized protocol secure the consensus mechanism by requiring nodes to stake and lock up a considerable amount of value, which can be deducted (usually refereed to as slashing) by the protocol in case the node misbehaves. The economic aspect of public blockchains poses a very high security risk. With such strong economic incentives to identify and exploit potential bugs, and system faults, it is of upmost importance for the developers to thoroughly test and examine potential problems. However, the aforementioned difficulties in debugging distributed and decentralized protocols require developers to be equipped with tools that supports their efforts.

In this study, we review the state of the art approaches in testing and debugging voting based consensus mechanisms and decentralized networks. We develop a visualisation specifically designed for researchers and developers to test such networks and compare real-time observed data with the expected. We conclude that visualisation techniques can be complementary to traditional log based debugging, and testing techniques. Moreover, we provide our tools as open source software as plugins for popular visualisation platform Grafana. Both tools make no assumptions on the data storage implementation. The plugins can be configured via Grafana plugin configuration interface to fit the specifics of the protocol implementation. We validate our tools by applying them to a custom developed blockchain, and then explain how successful they turned out to be in identifying anomalies and bugs in the protocols.

2 THE ROLE OF VISUALIZATIONS IN DEBUGGING COMPLEX DISTRIBUTED SYSTEMS

Distributed and decentralized systems are difficult to debug as developers are working on the third layer. Which includes L1 (code level bugs), issues with concurrency on L2 (individual runtime), and finally the third dimension for potential bugs arising from the message exchange between nodes. In general, it is often hard to capture the state in a distributed system as debuggers cannot be attached to all nodes' run-times. Additionally, it is often difficult to reproduce errors when they are inherently stochastic. We consider several methods, such as *Logging, Remote debugging, Simulations* and *Visualisations*.

- Logging is the most common debugging method for all three layers. However, in distributed systems it is important to aggregate logs, and analyze them as a time series.

Information Society 2022, 10–14 October 2022, Ljubljana, Slovenia

Additionally, aggregating distributed logs assumes the system has some method of clock synchronization protocol. Log collection has been proven to be effective in detecting performance issues for systems such as *Hadoop* [12] and *Darkstar* [13]. The aggregation can be done with specific tools for log collection such as *InfluxDB* [8], *Logstash* [10], etc. Aggregated logs then can be viewed in a form of a dashboard using tools like Grafana (see Figure 1).

Figure 1: Part of the Grafana dashboard used by developers to gain insight into a running PoS based blockchain network.

- Remote debugging is a technique where a locally running debugger is connected to a remote node in the distributed system. This allows developers to use the same features as if they were debugging locally. However, it is difficult to determine which remote node should be debugged. Additionally, in case of Byzantine behaviour due to network faults connecting the debugger could fail.
- Distributed deterministic simulation and replay is a technique that attempts to address the issues of reproducibility in distributed systems. Tools like *Friday* [5] and *liblog* [6] can be used to record the specific state of the network to use and analyze it later. The technique suggests implementing an additional layer that abstracts the underlying hardware and the network interfaces to allow for an exact replay of all the state changes and messages exchanged between nodes. Tools such as FoundationDB or even custom systems are built on containerisation software.
- Visualisation and time series analysis attempts at capturing the state of the system, and all the nodes by visualising the collected logs. Tools like Prometheus [11] and Grafana [2] are used extensively. Tools like *Theia* [4] and *Artemis* [3] are designed for monitoring and analyzing performance problems in distributed systems and support built-in visualization tools for data exploration. However, such tools provide logs aggregated based summaries of the distributed systems and are not capable of observing underlying low-level network properties, e.g. monitoring network communication, especially in real-time while the system is running. *ShiViz* [1] on the other hand displays distributed system executions as an interactive timespace diagram. With this tool all the necessary events and interactions can be viewed in an orderly manner and inspected in detail. ShiViz visualization is based on logical ordering, meaning that unlike our tools, it is not capable of running in real-time, together with the considered distributed network. ShiViz also works with aggregated logs about various types of events of the distributed system and unlike our tools does not support direct database connections. ShiViz is generalized and works with all kind of distributed systems, while our tools are created specifically for monitoring PoS voting based consensus mechanisms and underlying network topology of the distributed system.

3 RESEARCH OBJECTIVES

The main goal of this research is to build visualisation tools that offer more insight into a running distributed system using the time series log collection data. The targeted system is a custom proof of stake based blockchain. Such tools should visualize if nodes contributing to the consensus learned about their correct roles, and if they perform their roles accordingly. In the consensus algorithms this is done by sending messages, so the tools should visualise messages exchanged between nodes.

In the structured P2P networks information spreads using gossip protocols and network topology changes every time slot. Our tools should visualize such changes in the network topology by drawing nodes and their cluster representatives, while at the same time indicating the consensus roles for each node.

In our implementation time series data comes from InfluxDB, but we want our tools to have no assumption on the data storage implementation and there are other popular databases, such as kdb+ and Prometheus, that work well with time series data. Because of that we choose Grafana as a platform for visualizations, which supports all of the aforementioned databases and many more at the time of writing.

In this work we implement two Grafana plugins built to visualize PoS based blockchains, and decentralized network topology. Our tools are designed with generality in mind, and are hence applicable to other PoS voting based blockchains and other distributed ledger implementations. We evaluate our tools by applying it to the custom developed blockchain and note their usefulness in debugging and identifying potential issues in decentralized networks.

4 GRAFANA PLUGINS FOR VISUALISING VOTE BASED CONSENSUS MECHANISMS AND P2P OVERLAY NETWORKS

We have developed two plugins that extend the functionality of Grafana. Figure 2 outlines the architecture used in production. A server running a database instance (preferably time series i.e. InfluxDB), and the Grafana platform. Depending on the underlying blockchain implementation, nodes can insert their telemetry directly to the database, or if possible have an archive node gather telemetry from nodes, and report them. In this example, a cluster was used to run multiple nodes. A coordinating node is responsible for maintaining an overlay network and serving the nodes within the overlay with a DHCP, DNS, and routing. Nodes are packed within docker containers and submitted to the coordinator, which uses built in load balancing and distributes them to other cluster nodes.

The telemetry inserted is timestamped to create a time series stream of data that is consumed by Grafana. Figure 1 shows a small part of the dashboard created within Grafana using the built-in plugins for typical visualisations. These visualisations are time series data of a running blockchain showing telemetry reported by the nodes. However, rendering telemetry from hundreds of nodes as factors is hardly informative.

Both plugins were developed as React components, using a well-known D3.js JavaScript library for animations and life-cycle of the plugins is managed by Grafana

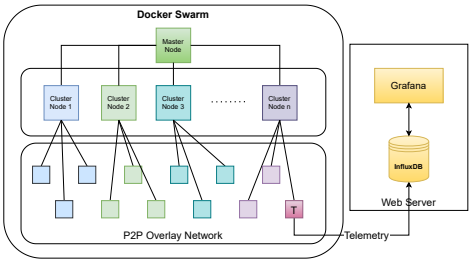

Figure 2: System architecture.

4.1 Network Plugin

P2P networks propagate information using gossip protocols. There are many variations of the general and implementation specifics but in general the family of protocols aims at gossiping the fact that new information is available in the network. Should a node hear about the gossip, and require the information it will contact neighbouring nodes asking for the data. In general, gossip protocols make no assumptions about the topology of the overlay network. However, with structured networks, the information exchange can be made much more efficient. The observed blockchain implementation utilized a semi structured network topology for propagating consensus based information. This is made possible by using a seed string shared between nodes that is used for pseudo-random role election every block. Using the seed, nodes self-elect into roles without the need to communicate. However, when performing roles, committee members must attest to the candidate block produced by the block producer. The seeded random is therefore also used to cluster the network using a k-means algorithm. The clustering is again performed by each node locally. The shared seed guarantees that nodes will produce the same topology, which is then used to efficiently propagate attestations to the block producer.

The network topology hence changes every slot. The plugin aims to visualize the changes in the network topology by drawing nodes, and their cluster representatives. Additionally, the consensus roles for each node are indicated with the vertex color. Figure 3 shows the network plugin rendering a test network of 30 nodes in real-time. The node in the center coloured green is the elected block producer for the current slot, nodes surrounded by the red stroke are cluster representatives, the rest of the nodes are coloured based on their role in the current slot.

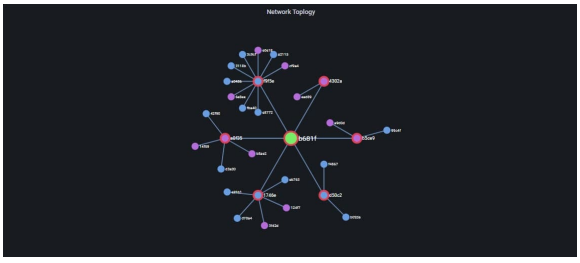

Figure 3: Network topology plugin visualising a test network of 30 nodes in real time.

4.2 Consensus Plugin

The aim of visualising the consensus mechanism is to quickly evaluate if nodes contributing to the consensus learned about their correct roles, and if they perform their roles accordingly. In order to have a scalable visualisation, nodes are placed around a circle, and scaled according to the size of the network. Roles are visualized with a color map. Each slot, nodes change their roles, and execute the protocol accordingly. To visualise the execution, the plugin visualises messages exchanged between nodes in a form of animated lines flying from an origin node to the destination node. The animations are time synchronous, and transfer times, and latencies are taken into account. Additionally, every message is logged with a type, indicating the sub protocol within which it was created. As an example, messages being sent from committee members to the block producer are attestations for the current block. The animated lines are coloured indicating the message type.

The thickness of the animated lines indicates the size of the payload transferred between nodes. Figure 4 shows the consensus plugin running live visualising a test network of 30 nodes. The green coloured node indicates the block producer role for the current slot, nodes coloured violet are part of the committee, and blue nodes are validators.

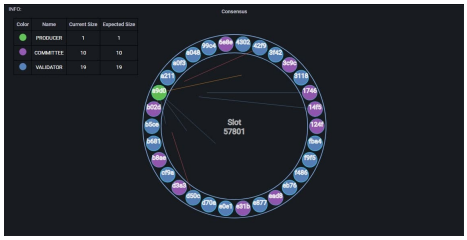

Figure 4: Consensus plugin (with legend) visualising a test network of 30 nodes in real time.

4.3 Generality

In order to use the above plugins, users have to provide certain data to the Grafana dashboard and this can be done through Grafana GUI. For the plugins to work all of the data should follow a specific naming policy. For example, for the Consensus plugin there is one necessary query to visualize data about the nodes of the network. It can be provided using SQL or Grafana GUI:

```
SELECT "slot", "node", "duty" FROM "<table-name>"
WHERE $timeFilter
```

Both plugins can be customized from the Grafana options menu. For example, users can add new roles, name and color them. Figure 5 shows the consensus plugin options menu, where users can additionally turn on or off display of messages, nodes or containers labels and so on. For both plugins, users have to manually provide the *slot time* of the network in the plugins options menus.

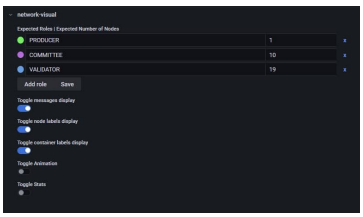

Figure 5: Consensus plugin options menu.

By using our tools we can visualize other protocols. For example with the consensus plugin we can visualize the famous Paxos algorithm, first introduced in [7] by Leslie Lamport. For that, we should provide the plugin with the *Nodes* and *Messages* queries. For the *Nodes* query, parameters *slot*, *node* and *duty* should be provided, which represent the slot number, node id and the role of the node respectively. From the point of nodes and slots, for this visualization Paxos works in the same way as the example of the PoS based consensus we mentioned before. For the *duty* parameter, nodes can have one of the three roles: *proposer*, *acceptor* or *learner*. That is why in the options menu of the plugin we should create 3 roles and name them according to the names from the data table.

We should specify *slot time* (in seconds) in the plugin options menu and at this point we can set the Grafana dashboard refresh time and see the results, since all the necessary conditions are fulfilled. But in order to gain more information from the plugin, we should add the *Messages* query. For the data we should have the following parameters: *id*, *source*, *target* and *endpoint*, which represent the message id, node id that sends the messages, node id that receives the message and the type of the message. For the additional information we can specify parameters *delay* (in seconds) and *size* of the message.

If we know the expected amount of nodes for some role, we can put it in the in plugin options menu to see this information in the plugin legend. In a similar way we should be able to visualize other consensus protocols, for example 2PC or Raft [9].

Source code for both plugins is open source, licensed under the MIT license and available on GitLab, where users can find the installation procedure of the plugins:

- Network plugin - https://gitlab.com/rentalker/topology-visualization-plugin,
- Consensus plugin - https://gitlab.com/rentalker/consensus-visualization-plugin.

5 CONCLUSION

We developed two Grafana plugins for visualising PoS based blockchains, and the underlying overlay network topology. The plugins were used to identify critical bugs, and faults in the protocol. With the help of visualisations, we were able to detect two problems when running test-nets.

- **Network congestion:** for every slot, validators must report their statistics to the block producer. Prompt delivery is desired but not critical. However, as the network grew in size, reporting statistics to a single node (block producer) became increasingly latent as all nodes attempted to propagate messages in tandem, and even more importantly, the network topology required a lot of routing for messages to arrive to the block producer. The network plugin helped us identify what the problem was by looking at the topology.
- **State synchronisation:** at random, nodes failed to perform their roles. This resulted in missing votes even on small test-nets, and sometimes a chain halt where no blocks were produced for the slot. We observed the likelihood of this happening grows in correlation with network size. However, it was infeasible to debug the state of all nodes in a large network. Visualising the state of nodes at a given slot we observed that states were not always synchronized and hence, some nodes did not learn about their consensus role.

We conclude that visualisation is an important tool in design and implementation of decentralized, and distributed systems. The methods serve a complementary role to existing debugging methods, and are very powerful at observing unexpected behaviour of the system as a whole. Visualisation techniques are specifically important in detecting stochastic faults that are non-trivial to reproduce. Our tools are open-source and available for researchers and engineers to use. They are suitable for testing any kind of voting-based consensus protocol with little effort.

For future work we would like to further develop our tools to accommodate other consensus protocols and help developers visualize and debug other types of issues related to distributed systems. Also, we would like to explore other types of visualizations and other existing tools that can help developers as well. Since Grafana is rapidly evolving, our developed plugins can be updated and new technologies can be integrated with our tools to improve their performance.

6 ACKNOWLEDGMENTS

The authors gratefully acknowledge the European Commission for funding the InnoRenew CoE project (H2020 Grant Agreement #739574) and the Republic of Slovenia (Investment funding of the Republic of Slovenia and the European Union of the European Regional Development Fund) as well as the Slovenian Research Agency (ARRS) for supporting the project number J2-2504 (C).

REFERENCES

[1] Beschastnikh, I., Wang, P., Brun, Y., and Ernst, M. D. Debugging distributed systems. *Commun. ACM 59*, 8 (jul 2016), 32–37.
[2] Chakraborty, M., and Kundan, A. P. Grafana. In *Monitoring Cloud-Native Applications*. Springer, 2021, pp. 187–240.
[3] Creţu-Ciocârlie, G. F., Budiu, M., and Goldszmidt, M. Hunting for problems with artemis. In *Proceedings of the First USENIX Conference on Analysis of System Logs* (USA, 2008), WASL'08, USENIX Association, p. 2.
[4] Garduno, E., Kavulya, S. P., Tan, J., Gandhi, R., and Narasimhan, P. Theia: Visual signatures for problem diagnosis in large hadoop clusters. In *Proceedings of the 26th International Conference on Large Installation System Administration: Strategies, Tools, and Techniques* (USA, 2012), lisa'12, USENIX Association, p. 33–42.
[5] Geels, D., Altekar, G., Maniatis, P., Roscoe, T., and Stoica, I. Friday: Global comprehension for distributed replay. vol. 7.
[6] Geels, D., Altekar, G., Shenker, S., and Stoica, I. Replay debugging for distributed applications. In *2006 USENIX Annual Technical Conference (USENIX ATC 06)* (Boston, MA, May 2006), USENIX Association.
[7] Lamport, L. The part-time parliament. *ACM Transactions on Computer Systems 16, 2 (May 1998), 133-169. Also appeared as SRC Research Report 49. This paper was first submitted in 1990, setting a personal record for publication delay that has since been broken by [60].* (May 1998). ACM SIGOPS Hall of Fame Award in 2012.
[8] Naqvi, S. N. Z., Yfantidou, S., and Zimányi, E. Time series databases and influxdb. *Studienarbeit, Université Libre de Bruxelles 12* (2017).
[9] Ongaro, D., and Ousterhout, J. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (USA, 2014), USENIX ATC'14, USENIX Association, p. 305–320.
[10] Sanjappa, S., and Ahmed, M. Analysis of logs by using logstash. In *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications* (Singapore, 2017), S. C. Satapathy, V. Bhateja, S. K. Udgata, and P. K. Pattnaik, Eds., Springer Singapore, pp. 579–585.
[11] Turnbull, J. *Monitoring with Prometheus*. Turnbull Press, 2018.
[12] Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. Online system problem detection by mining patterns of console logs. In *2009 Ninth IEEE International Conference on Data Mining* (2009), pp. 588–597.
[13] Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. I. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), SOSP '09, Association for Computing Machinery, p. 117–132.