ResilientDB Visualization Framework

Yu-Hsuan Lin, Yi-Hsiang Chiu, Tzu-Hsin Yang, Pin-Ting Liu, Yu-Cheng Hwang
Department of Computer Science
University of California, Davis
{zyhlin,yihchiu,zixyang,ptliu,timhwang}@ucdavis.edu

December 11, 2021

***Abstract*—Nowadays, Cryptocurrency, Bitcoins, Ethereum, Dogecoin, and so on, become more and more popular as investments and transactions. The blockchain technology behind these cryptocurrencies has also entered people's field of sight. Nonetheless, the mechanisms of blockchain are complex and confusing because it is difficult to look into the details of the mechanisms. Therefore, visualizing the view of blockchain is critical work. It can assist people to grab more ideas behind the blockchain. Also, blockchain researchers can make use of visualization systems to monitor the status of blockchain. As we know, blockchain is a distributed system, and each block works on a different machine independently. In this project, our goal is to visualize the status of those individual machines. We will use ResilientDB as the example of blockchain, and monitor the situation of each machine (replica), and then build a web user interface (UI) to show the information such as CPU utilization, throughput, and so on during a transaction.**

I. Introduction

In order to adopt blockchain technology in a wider range of fields, we need to explore and analyze transaction data to better understand user behaviors and mechanisms in the blockchain system. Therefore, visualization tools can support humans in analyzing transactions and deriving hypotheses and models used in the blockchain.

There are many previous work developing visualization systems. [1]–[6]. However, most of them only provide descriptive statistics or transaction details. In order to help users and new bitcoin learners can understand clearly about the mechanism of bitcoin. We would like to visualize the communication process between replicas.

There were many types of visualization tools proposed for above purposes. Among them, time series data were the main visualization data type for blockchains data [7]–[13]. It is very straightforward to monitor the health condition of blockchains by observing the time series data because blockchains is a complex time dependent system. The metrics to monitor can be CPU, memory, network bandwidth usage, system load and disk R/W data, etc.

II. Related Work

A. Logging

The purpose of collecting loggings is to gather the real time data from the blockchain. Most of the data that we want to fetch rely on loggings to export them. Therefore, without a doubt, loggings are the most essential and important part for our visualization framework. Yet, the original loggings in ResilientDB are more complicated than we need. To collect those data, we carefully review the code we need on how ResilientDB logs those data.

B. Parameters Decision

The parameter decision will impact the way and efficiency of manipulating the dataset which we use to save the data exported from the blockchain. The precision of timestamp, the format of timestamp, the compressing mechanism, and the tag for each type of message will directly affect the chart in the frontend. Although high precision and less compression log can bring us meticulous charts, it consumes more storage and CPU power, or larger

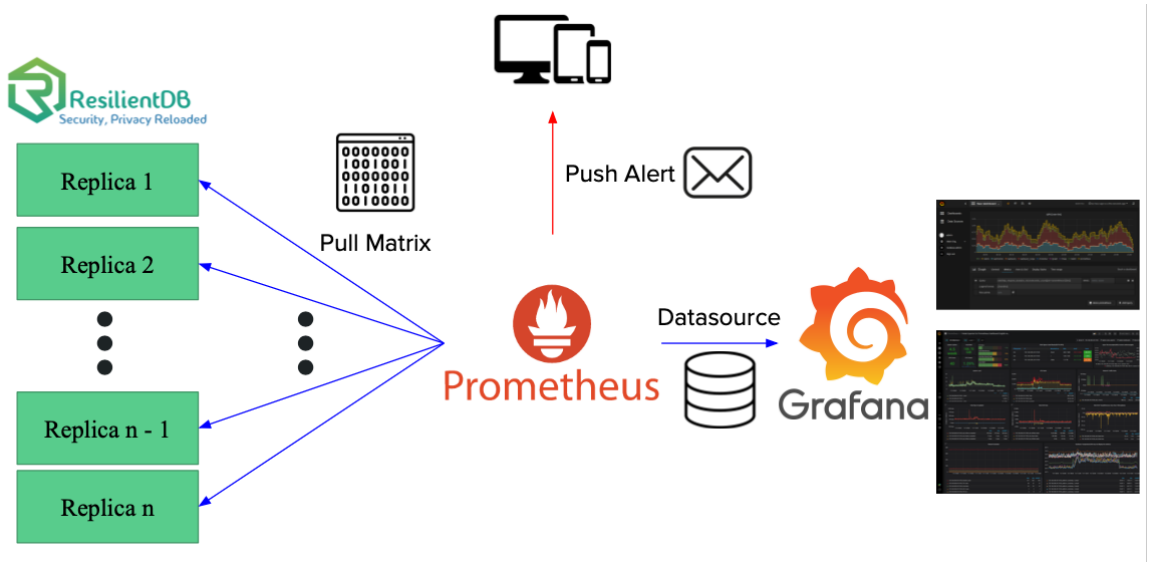

Fig. 1. Flowchart

latency, while rendering the page. Therefore, it will be a trade-off between the user view and server reliability.

C. Infrastructure Integration

Integrating our library and infrastructure setting to the original codebase is a foreseeable hard work. We will ingrain an exporter on each node, which sends their log to the central unit. The central unit acts as a data collector and a temporary storage. Moreover, we will also set up a web visualization interface as the monitor allowing users to track the status of each component of the blockchain. Both the central unit and the web visualization interface are expected to be built over Docker. As a result, because the intense infrastructure development is involved in this project, the part of bringing up the whole system will be an endeavor.

D. Original ResilientDB Dashboard

In ResilientDB, we know that it has its own simple visualization implementation, a simple dashboard. After further looking into how the original dashboard works, we realize the current dashboard does not work as we expect. It does not really include the visualization part in the process of installing the ResilientDB. That means if users want to use a dashboard, they have to install a central unit along with a visualization interface on their own. It requests lots of external work for users to do so. Therefore, in the project, we will build up a solid way for the users who work on ResilientDB in the future to easily install a visualization framework with our project.

E. ResilientDB Backend

Even though the dashboard would not work right now, we still trace the code to its backend to see how it fetches data from the replica which is an instance in the blockchain. We realized that in the backend of the dashboard it uses some Shell script to get the data it wants. It is not the usual means of backend. Nonetheless, we still take some time to trace down the code and make sure we got some initial data that we needed for our visualization.

F. Replica IP table

Since we try to fetch data from the replica, it is essential to understand where the data is from. Then, we can ensure the data we get is from the corresponding replica. Yet, the IP table that we get is nothing but a list of IP addresses. Even though we can check the addresses in the container, without any identifier, we can not correspond to the replica, and monitor the status right away. Therefore, while constructing our central unit, we can also connect

the replica through the setting in the central unit, then we don't have to worry about the confusion of the original IP table.

III. Our Approach

In our approach, we leverage the benefits of Docker [14]. Docker technology has been highly used nowadays. We concatenate the replicas, the central unit, and web visualization interface in the docker network by their domain name. Since the domain name settings are more readable compared to IP settings, using Docker can also address the current IP table problem.

Next, to be able to build new images that integrate all applications we need for ResilientDB, we dive into the source code of ResilentDB, and decide which metrics of replicas are important and necessary for visualization. Then, we capture these features by logging them in the output file. On the other hand, we introduce Prometheus [15] as our central unit, which is an open-source systems monitoring and alerting toolkit. We use the Prometheus server to retrieve the logs from the output file and visualize the statistics of replicas on Grafana [16] Web UI. Also, we configure the alert manager to monitor the status of replicas and notify the administrator by email when errors occur. The Fig. 1 is the flow chart of our system.

A. ResilientDB

S. Gupta et al. presented the Geo-Scale Byzantine Fault-Tolerant consensus protocol (GeoBFT) [17] to address the problem of geo-scale deployments where many replicas spread across a large area participate in consensus. The main challenge of geo-scale blockchains is to distinguish between local and global communication. Only by distinguishing between these two communication types, can global communication be minimized. GeoBFT performs a topological-aware grouping of replicas into local clusters so that it can minimize global communication. On the other hand, the paper introduces a novel global sharing protocol that optimistically performs minimal inter-cluster communication, still being able to reliably detect communication failures. The experiment results show that GeoBFT achieves up-to-six times more throughput than existing BFT protocols.

ResilientDB has originally provided a visualization platform. However, we found that the way the dashboard works is not available now. We then found that in the backend of the dashboard, it applies some shell scripts to get the data it needs as shown in Fig. 3. Moreover, the dashboard acquires a pre-established intra-logging server in the internet. Nevertheless, the log server establishing scripts is not existed in the original codebase. Therefore, we find that an error occurs when people uses logging system. Another problem is that the original logging function posts logging data to the log server, InfluxDB, using different shell scripts. The dependency leads dashboard malfunctioning. To be more specific, this approach highly couples with the original log output format. Once the original code has been changed, the log scripts need to be adjusted accordingly. If we implement new function in ResilientDB and want to see the logs of that function, we have to add several log collectors and post functions. It is an unusual way of backend engineering for monitoring. It humongously hampers the future implementation, prolongs the development cycle, and hinders the research on ResilientDB. Therefore, we aim to improve this system by building a visualization tool that makes the service more visible and easily accessible.

B. Prometheus

Our Monitoring architecture aims to achieve low dependency, high integration, and open for future expansion. Because the monitoring is related to time series databases, log collectors, and visualization, we take several existing open third party time series databases into considering, including InfluxDB [18], Prometheus, and Elasticsearch [19]. The Prometheus have robust community, which gives the database long-term support and comprehensive document. Therefore, we choose Prometheus as our central unit which can provide more scalability for future implementation. Also, Prometheus is developed by Golang, which is highly

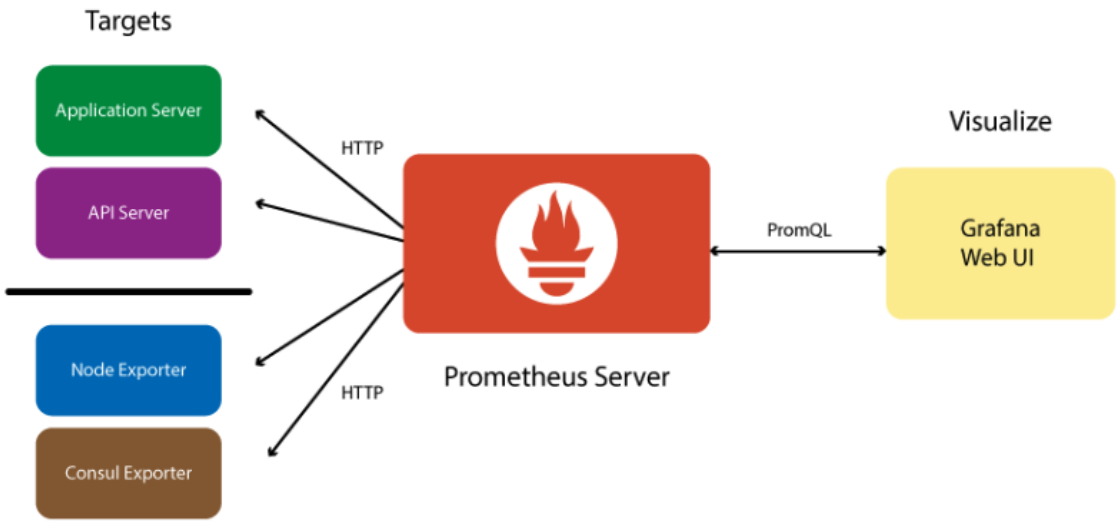

Fig. 2. Framework

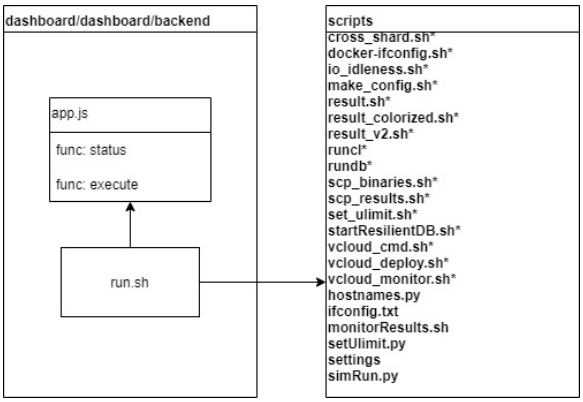

Fig. 3. Component Dependency

efficient language and able to achieve low dependency. More, Prometheus allows huge amount of third party libraries to interact with it. While it is compatible with many UIs, it comes with a default UI which we can first test whether our Prometheus server is working or not. In conclusion, Prometheus is an ideal choice for us as a third-party library to be the central unit. Prometheus is highly scalable as a node exporter and inverts the current dependency of visualization implementation in ResilientDB.

Prometheus is an open source monitoring system developed by engineers at SoundCloud in 2012. In 2016, Prometheus was accepted into the Cloud Native Computing Foundation. The Prometheus monitoring system includes a rich, multidimensional data model, a concise and powerful query language called PromQL, an efficient embedded timeseries database, and over 150 integrations with third-party systems.

With the help of Prometheus, one can quickly locate the current application's problems, thus handling solutions immediately. The benefit of Prometheus is that we can integrate with any system and extract the part we want to see. For example, when implementing GeoBFT, one can easily observe the network latency with specific metrics data or discover the network throughput of the system. Besides, if there exists an idle node in the network, it would be found out in seconds by observing the inbound/outbound rate of the instance.

C. *Grafana*

We use Grafana as our web visualization interface. Grafana was written in Golan and first released in 2014 by Torkel Ödegaard at Orbitz. Grafana is an interactive visualization web application. It demonstrates diagrams, charts, curves and graphs on the web when it connects to data sources. In our work, we use Prometheus as a monitoring solution for storing time series data like metrics, and use Grafana to visualize the data stored in Prometheus, including all the statistics regarding the status of the replicas in ResilientDB.

Grafana provides a handy dashboard to let users organize the metrics data pulled from Prometheus

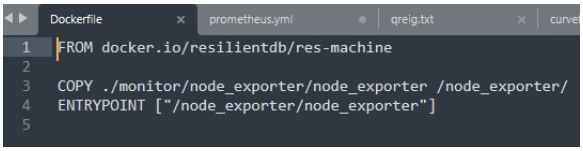

Fig. 4. Dockerfile Configuration

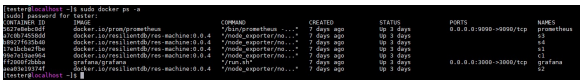

Fig. 5. Docker Status

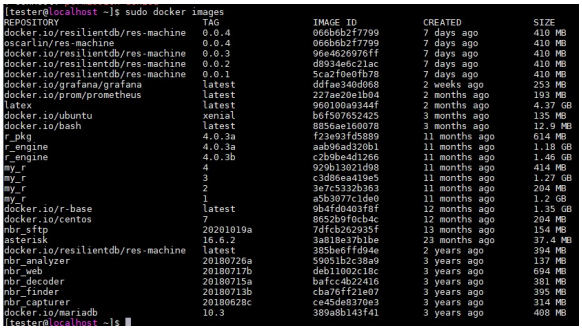

Fig. 6. Docker Image

with the highly customizing interface. For example, we can compare the network throughput of each replica to observe the data exchange between them.

IV. Results

A. Environment Settings

The new architecture involves three parts, Docker, Prometheus, and Grafana. For the integration part, we create Dockerfile, docker-compose, and modify the original ResilientDB-docker script. In Fig. 4, we ingrain the node_exporter with original ResilientDB images and modify the Dockerfile. The new image is named docker.io/resilientdb/res-machine with the version number of 0.0.4, as shown in Fig. 6, allows Prometheus to scrape data from each node. We bring up all of the services by using docker-compose, as shown in Fig. 7. We established the ResilientDB essential services: replicas, clients, Prometheus, and Grafana. All services join the same docker network, resilientdb_default. The network runs in bridge mode and allows services to expose their ports. We can access the docker containers via those exposing ports. In Fig. 8 , we adjust the ResilientDB script to prevent it from not creating a new docker-compose file while starting the simulation. Besides, the monotonous docker-compose uses the domain name of the containers, improving readability and simplifying the configuration.

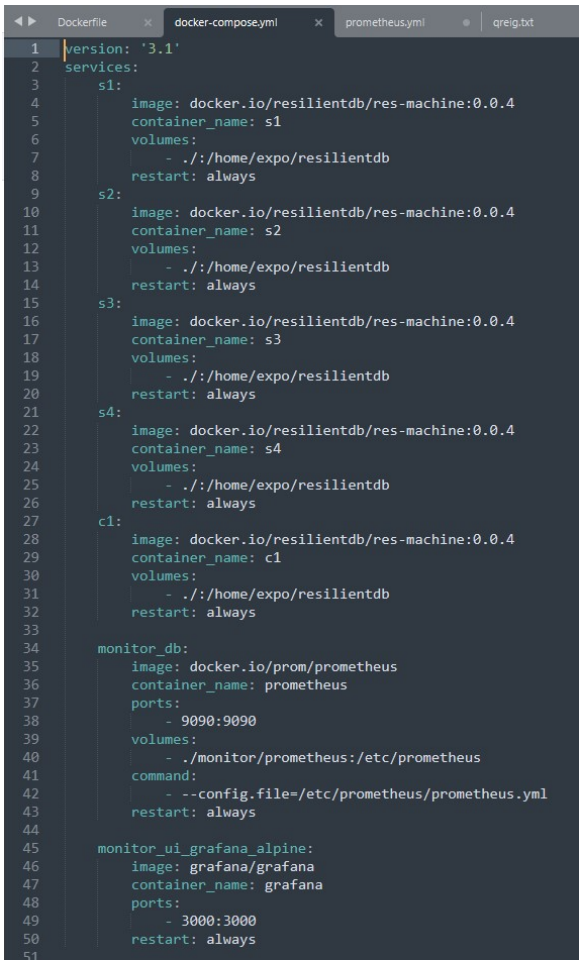

Fig. 7. Docker Compose

B. Prometheus

In Figure 9 we set the basic information of replicas in the YAML file for Prometheus to connect to the instances. And the status of the Prometheus after executing the YAML is shown as Fig. 10. We have already pulled the metrics data from each replica, and we only take several data as examples in our project demo. The data we retrieve is enough for monitoring the health status of the whole system. The Prometheus provides a basic UI dashboard as shown in Fig. 11, but later we will further integrate with Grafana with a more user-friendly interface.

Fig. 8. ResilientDB Shell Script

Fig. 9. Prometheus Configuration

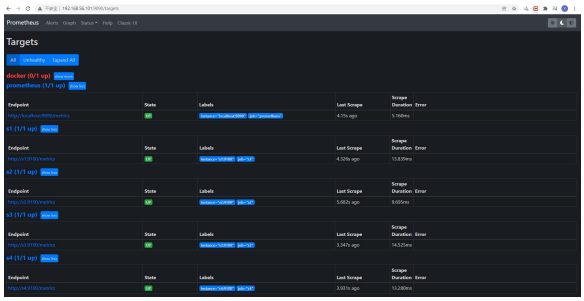

Fig. 10. Prometheus UI

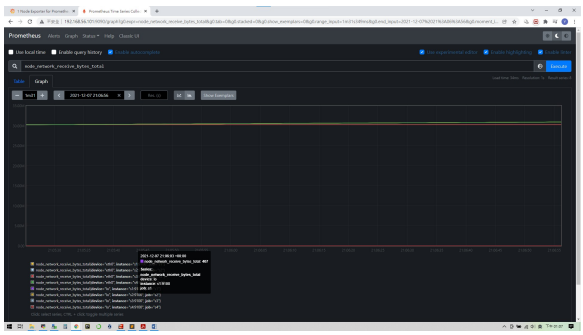

Fig. 11. Prometheus UI Dashboard

C. *Grafana*

On Grafana UI web, we can choose which data source we would like to connect. In our case, obviously, we choose Prometheus server as our data sources as shown Fig. 15. We can show the status of all replicas on the Grafana web UI as shown in Fig. 12, including CPU, memory, partition usage. Also, we can see the transmit and receive status among replicas. If we are interested in one specific replica, we can pull out its status, and monitor its statistics in detail as shown in Fig. 13 and Fig. 14. From these figures, we can see the time series values of all monitored metrics. This feature is significant especially when the network is unusual, and we want to find which replica is abnormal.

V. Future Work

The current progress of this project has already implemented the basic structure of the system, such as retrieving metrics data from replicas and the customizable visualized dashboard. Prometheus provides more than just extracting the data.

Besides, in the next step, we will implement an alert manager to notify users if the system

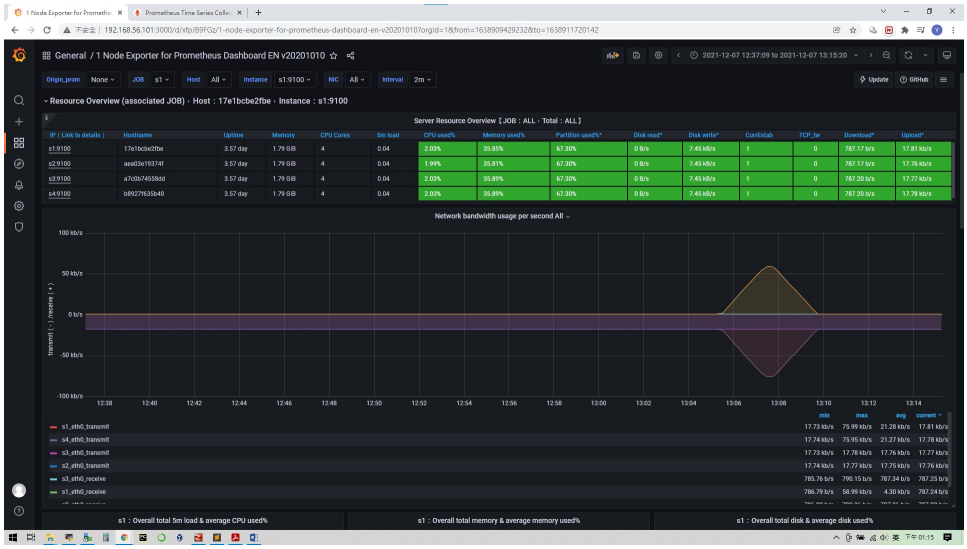

Fig. 12. Grafana UI

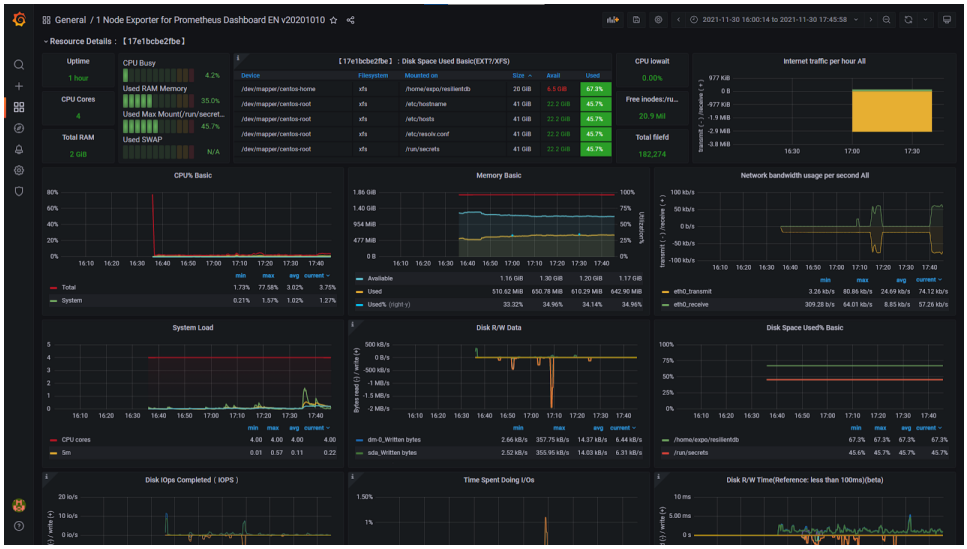

Fig. 13. Grafana UI

encounters some issue in the future, thus resolving the problems in a short time. Also, Grafana can integrate with other related data sources with the whole blockchain networks, such as the data from different chains.

On the other hand, there is more work we can do on ResilientDB. Now, the logs we extract from ResilientDB are only information of each replicas such as CPU and memory usage. In the next step, we expect to store more information about the state of the entire system, especially at which stage it is in, including pre-commit, commit, and view changes. If we can show the stages of ResilientDB, it can help users learn more about the process and health condition of each transaction.

Moreover, in our work, we manually write the docker compose file and Prometheus YAML file. However, it is not ideal for system integration. Therefore, in the future, we would like to design a shell script which is able to automatically generate docker compose file and Prometheus YAML file.

VI. Conclusions

In this work, we build a visualization system based on ResilentDB, which is the Geo-Scale Byzantine Fault-Tolerant consensus proto-

Fig. 14. Grafana UI

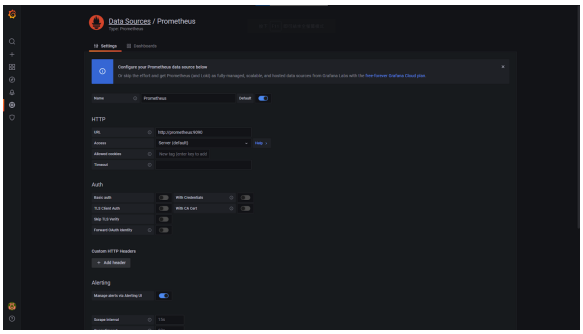

Fig. 15. Data Source

col (GeoBFT) aiming to address the problem of geo-scale deployments where many replicas spread across a large area participate in consensus. Since the original visualization platform of ResilientDB has issues of components dependency, we construct a new visualization system to provide users and learners to understand the mechanisms and status of replicas more easily. We built the Prometheus server to pull all the statistics of replicas in ResilientDB. Then, we build the Grafana user interface, and let users decide which metrics they would like to monitor. When users query the metrics, the Grafana will pull the specific metrics from the Prometheus server, and demonstrate the charts and curves on the web UI. This work is very valuable for providing an independent service to monitor the communication process within a protocol. Moreover, our work has high scalability. For those users and developers on ResilientDB, it is costly-less for integrating our project into the current ResilientDB. Our work can not only monitor ResilientDB, but it can be used on a variety of protocols.

REFERENCES

[1] "Bitbonkers.com," https://bitbonkers.com/.
[2] "Bitcoin transaction visualization.com," http://bitcoin.interaqt.nl/.
[3] "Bitnodes.earn.com," https://bitnodes.earn.com/.
[4] "Realtime bitcoin globe," https://blocks.wizb.it/.
[5] "Blockchain.info," https://blockchain.info/tree/114688199.
[6] "Elliptic.co," https://www.elliptic.co/.
[7] A. Bogner, "Seeing is understanding: Anomaly detection in blockchains with visualized features," p. 5–8, 2017.
[8] C. Kinkeldey, J. Fekete, T. Blascheck, and P. Isenberg, "Visualizing and analyzing entity activity on the bitcoin network," *CoRR*, vol. abs/1912.08101, 2019.
[9] H. Kuzuno and C. Karam, "Blockchain explorer: An analytical process and investigation environment for bitcoin," pp. 9–16, 2017.
[10] D. McGinn, D. Birch, D. Akroyd, M. Molina-Solana, Y. Guo, and W. Knottenbelt, "Visualizing dynamic bitcoin transaction patterns," *Big Data*, vol. 4, pp. 109–119, 06 2016.
[11] D. McGinn, D. McIlwraith, and Y. Guo, "Toward open data blockchain analytics: A bitcoin perspective," *CoRR*, vol. abs/1802.07523, 2018.
[12] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: Characterizing payments among men with no names," *Commun. ACM*, vol. 59, no. 4, p. 86–93, mar 2016.
[13] L. Norbutas, "Offline constraints in online drug marketplaces: An exploratory analysis of a cryptomarket trade network." *The International journal on drug policy*, vol. 56, pp. 92–100, 2018.

[14] “Docker,” https://www.docker.com/.
[15] “Prometheus,” https://prometheus.io.
[16] “Grafana,” https://github.com/grafana/grafana.
[17] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, “Resilientdb: Global scale resilient blockchain fabric,” *CoRR*, vol. abs/2002.00160, 2020.
[18] “Influxdb,” https://www.influxdata.com/.
[19] “Elasticsearch,” https://www.elastic.co/.