IPSJ SIG Technical Report

Vol.2021-MBL-100 No.17
Vol.2021-UBI-71 No.17
Vol.2021-CDS-32 No.17
Vol.2021-ASD-21 No.17
2021/9/3

SOXCollaborator: A Bridge System to Realize Open-Data Commerce by Collaborating with SOXFire and Blockchain Marketplace

Takafumi Kawasaki[1,2,a] Georgios Palaiokrassas[6,5,b] Akira Tsuge[1,2,4,c] Tadashi Okoshi[1,3,d]
Antonis Litke[6,5,e] Jin Nakazawa[1,3,f]

Abstract: SOXFire is the IoT platform based on pub/sub-model. This platform can provide easy methods for treating sensors because it has a feature by extended XMPP. In addition, this platform is implemented by XMPP, so this platform can spread data for many clients. We focused on this platform; we thought the extend the system to realize an open-data marketplace. For this reason, data will have a huge value better than now, for example, such as monetary value in the future world. Then, we focus on blockchain as an effective technology. In addition, the IoT marketplace using blockchain is implemented now. We focus on realizing open-data commerce by collaborating with SOXFire and the blockchain marketplace. In this paper, we developed a system to integrate those. Last, we tried to do preliminary experiments to evaluate feasibility by measuring loads of a machine resource and delay time.

Keywords: IoT Platform, Data Delivery, Ubiquitous Computing, System Integration

1. Introduction

The number of sensors and smart devices all over the world is growing day by day, and their data can be gotten easily by developing information technology. Those are used to improve the QoL (quality of life) of all people and that's useful in various situations. As the result, IoT (Internet of Things) is very general, and it will spread better than now. People can think more flexibility how to use these sensor data by this. For example, up until now, engineers can use the data of individual sensors, but they can collaborate many sensors now, so engineers can create more applications to the convenient for the environment, people, and industry. Then, we need to think about how to easily send the data from too many sensors and devices with many clients. The pub/sub (publish/subscribe) messaging model[1] is focused to resolve this problem. However, the general pub/sub model platform is difficult to treat complex sensor device for clients.

Then, Yonezawa et al. was developed SOXFire[2][8][12] can resolve this problem. SOXFire realized the easy topic management and multi data sending smoothly, so clients can develop some sensor application by using it.

Besides, when it will have realized to share many data among clients, we will need to think next step. That is data security and data commerce. Data security means reliability and impartiality and more because data will have the important value better than current. However, SOXFire does not have the function to satisfy these requirements. Next, we focused on the technique of blockchain.[3][6][11] The blockchain can provide trade-guaranteed trust and impartiality. In fact, the marketplace is created by this technology. This will be important for data trade in the future.

It means one data share within the owner organization by a pub/sub model platform; besides, a part of data can sell to external clients by using blockchain marketplace. However, blockchain marketplace is not suitable to spread data for many clients immediately because it creates data by chaining among data. Therefore, we think collaboration with SOXFire and blockchain marketplace. It can realize the strong data exchanging system because the system can use good features from each model.

From the above, we implemented a bridge system to collaborate with SOXFire and blockchain marketplace. The system calls "SOXCollaborator". SOXCollaborator can provide smooth data delivery from the pub/sub IoT platform to the blockchain platform. In addition, we think the problem with using this bridge system is because it must generate a new cost, for instance, loads of machine resources and sending delay time via our system. We evaluated feasibility by measuring them.

At this time, we use the SOXFire such as a pub/sub model

[1] Keio University, 5322, Endo, Fujisawa, Kanagawa 252-0882, Japan
[2] Graduate School of Media and Governance
[3] Information and Environment
[4] YRP R&D Promotion Committee
[5] School of Electrical and Computer Engineering
[6] National Technical University of Athens
[a] drgnman@sfc.keio.ac.jp
[b] geopal@mail.ntua.gr
[c] tsuge@sfc.keio.ac.jp
[d] slash@sfc.keio.ac.jp
[e] litke@mail.ntua.gr
[f] jin@sfc.keio.ac.jp

platform, so we implemented the integration system to realize to connect SOXFire to the general blockchain platform.

In this paper, the 2nd section is written the related works, the 3rd section is about SOXCollaborator, the 4th section is the evaluation of it, and the last section is described the future work and conclusion.

2. Related Works

2.1 Publish/Subscribe Messaging Model

First, we describe about pub/sub messing model because we choose this model to collaborate platform in this paper. This messaging model is suitable to send data to many clients. The best feature of the pub/sub messaging model is that a subscriber (data receiver) and publisher (data sender) does not communicate directly. The pub/sub model realizes two data publish methods that topic-based and contents-based.

On the topic-based pub/sub model, when publishers send data to clients, they do not send it to clients directly, they send data to the topic on the broker server.

On the other hand, subscribers subscribe to the topic that they want to get data in advance. When a subscriber subscribes topic, a connection is created between the subscriber to the broker server. Then, the publisher sends data to a broker server, the broker server sends data to a subscriber to whom he subscribes the sending targeted topic. Currently, the broker sends data to all subscribers who subscribe to the target topic. Therefore, if the publisher and subscriber do not know each other, they can send and receive data.

Besides, at filtering-based pub/sub model too, publishers and subscribers do not communicate directly. In this model, the broker server has a filtering function by the value of data instead does not have the function to manage topics. The broker server sends data to subscribers by using this filtering, so subscribers can get the data without subscribing to specific topics. This model is effective when the subscriber cannot choose a specific topic. For instance, a client does not find the topic that should subscribe, or a client wants to get data that are sent to multiple topics, and a client wants to choose data to get by using filtering value.

Currently, it is used a topic-based pub/sub model generally, so in this research, we implement the integration system for the IoT platform of the topic-based.

However, pub/sub model does not consider that exchanging data for data commerce because data sends one way a sender to a subscriber. The purpose of this model is data multi-cast for many clients on the same time. Therefore, only this model cannot realize the data commerce for users.

2.2 SOXFire

In this paper, we use SOXFire[2][8][12] as an IoT platform. SOXFire is used by the topic-based pub/sub model, and this system are implemented by an extending Openfire [7] platform. The platform is implemented by the XMPP (Extensible Messaging and Presence Protocol). [10]. Therefore, SOXFire also uses XMPP. XMPP is developed for instant messenger applications, so this protocol can treat text data and provide certain multicast clients. Therefore, Openfire can treat the large data of texts, so it can publish image data such as converted base64.

SOXFire extends Openfire for the treating sensor data simply. In this system, all sensors are structured two topics. They are “meta” topics (meta node) and “data” topics (data node). Meta node manages the sensor information. The management items are sensor name, category, and transducers. At XMPP, data is structured in XML format. Therefore, generally, XMPP platforms manage text data as only one tag.

However, a complex sensor has multi values of variables, in this case, it is difficult to manage at one tag data. SOXFire manages these values as each transducer. This is realized by granting a tag for each value. Therefore, SOXFire can provide simple methods what management complex sensors, and subscribers can get all sensor data by integrating data format. This feature is satisfying to collaborate with other platforms. This reason is what it can send the data structure clearly. For the above, this system is very useful for IoT, so we focused on this system to collaborate with the other systems.

In this research, targeting this system, all sensor data collaborate with the other systems.

2.3 Blockchain and Its Marketplace

Recently, there is a big research interest for blockchain based systems as data sending platforms [3][4][8]. The underlying blockchain technology of such platforms has big differences comparing to the previous general network models for data exchanging. The previous most widely used general model includes the existing clients and the server. Almost all data available for exchanging are stored on the server, so communication and access to the server is necessary, if a client wishes to acquire data for business or other purposes. In this case, when the client accesses the server, the server replies to the client with the requested data, which will be then used by the client according to its needs. This means that all data is exchanged via the server. Therefore, if the server is broken, clients will not be able to access it. Besides, this server has a large load because it has a very important role, so during the periodical regular server’s maintenance clients will also not be able to access it until the maintenance work is finished. It is evident that, when the server stops, the effect spreads to all networks where its clients are connected.

However, blockchain has very high reliability, availability, and durability because a downtime almost never happens. The main reason for these advantages is the features and the technologies of its structure. It is based on p2p connections, so users can directly connect. The blockchain network is formed by all participating users and in the case of a member of the network being down or unavailable, this doesn’t make the whole network being down. As the result, blockchain can be used to construct very robust networks.

In addition, the blockchain has another very strong point, namely integrity, since all data are chained into series of blocks with timestamps forming a distributed ledger of transactions. This means that in the case of an attempt by a malicious user to modify part of the data, this altered information will not match to the existing information of all other nodes, which are part of the blockchain network. In this case, the participating nodes will not include the modified data to the chain of blocks and as a result the malicious attempt will fail. As a result, blockchain allows security and trust among members of a decentralized network, realizing high levels of integrity of the data.

From the above, blockchain's privacy and security features, as well as smart contracts running on it can be used to construct a strong network, where users are able to trade data safely and securely among each other. This provides very attractive perspectives as data have an increasing value in the IoT world.

To this direction, IoT Marketplace was developed based on blockchain technology for sharing IoT sensors' data[4]. By combining blockchain technology with IoT, it leverages smart contracts that are executed on the Ethereum blockchain, creating a platform that allows users to exchange sensor data. This decentralized application (Dapp) is designed based on Sensing-as-a-Service (S2aaS) business model, originally proposed for Bitcoin [5]. S2aaS brings new data monetization opportunities to IoT sensor operators, by enabling them to easily sell sensor data while it makes IoT sensor data more accessible to customers, by enabling them to easily access those data. It consists of web applications, front-end user interfaces, enabling user interactions and Node-Red [9] flows allowing the connection with other APIs, data sources, deployed smart contracts and databases. Different users, namely owners registering their sensors and buyers purchasing available datasets, are able to exchange data and value using the dedicated created cryptocurrency, in a secure and user-friendly manner.

3. SOXCollaborator

In this section, we describe implemented system, "SOXCollaborator". SOXCollaborator performs by connecting SOXFire, and this system helps to collaborate with SOXFire and blockchain marketplace.

In addition, when users want to connect SOXFire and blockchain marketplace, users need to think about the information to integrate them. In this paper, we defined the format that this necessary information. This format will become the reference when users realize it.

3.1 Define The Data Format to Collaborate Blockchain Marketplace

We presuppose open-data market by using collaborated with SOXFire and blockchain marketplace. The sent data from the SOXFire includes the following information. It is publisher name, topic name (that means sensor name or device name), the value of that, and publish timestamp. However, the case of users publish data to the blockchain marketplace, does not satisfy only this information. We thought the following additional data. It is login information about the blockchain marketplace, monetary value, and callback URL. Callback URL is the most important in those. A Publisher does not connect the marketplace directly, when sends the data via the SOXFire. A Publisher cannot note when it is done action oneself data on the marketplace. Therefore, clients need to prepare the API to receive the notification from the marketplace. The blockchain marketplace can notify the client by using a callback URL when actions client's data. For instance, the time is when it sold the data. Table.1 shows the all necessary data to collaborate the blockchain marketplace.

Table 1 Necessary parameters for collaborating the blockchain marketplace

Eelements	Description
Publisher	Publisher name
Topic	Sensor or device or information name
Category	Information's category
Value	Value or contents
Timestamp	Publish Timestamp
Username (login info)	Username of the external services
Password (login info)	Password of the external services
CallbackURL	Endpoint to receive notification

3.2 System Architecture

Fig.1 shows the system architecture of the SOXCollaborator. SOXCollaborator uses two protocols as shown in Fig.1. First, XMPP is used to communicate between SOXFire. This connection follows the methods of the SOXFire connection.

Second, HTTP is used to communicate between SOXCollaborator and client. The client orders various control by using an HTTP request. These controls include starting to subscribe and topic index. These controls do not need to sustain connection until subscribing because our system has the function what storing data. Therefore, responses for needing clients are only topic searching results or signals to confirm starting subscribe, so we chose HTTP.

SOXCollaborator has the multi-functions below to collaborate with other platforms.

Functions:
- Sustainable Subscribing to SOXFire
- Receiving data from SOXFire to store Database
- Subscribing from multi servers
- Searching topic and data from SOXFire and SOXCollaborator

Each detail is described below in subsections.

3.2.1 Sustainable Subscribing to SOXFire

If the platform has this function, it must sustain a connection to SOXFire. However, the web application uses HTTP connects only when it is called API by clients. Therefore, connection to SOXFire is a burden for web applications because it occurs in other packets sustainably.

SOXCollaborator performs data subscriber as instead of web applications. This system can subscribe to multiple topics.

Therefore, we implemented multi-endpoint to realize flexible designation.

designation method of subscribing topic:
- All Topics
- Specific Topic
- Prefix Match Topics

First, all designation methods have common parameters. It is a connection destination and user information. These parameters are saved database of SOXCollaborator that uses it to connect SOXFire. All parameters are described in Table.2.

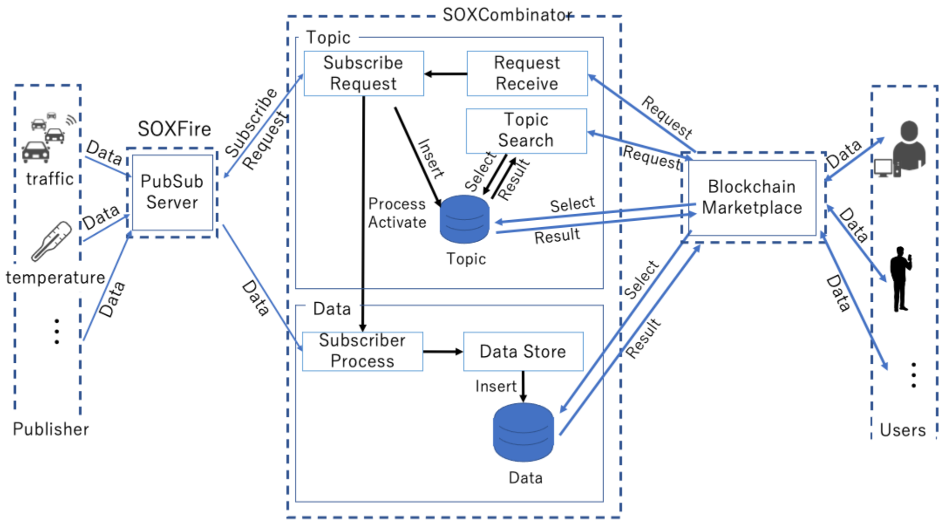

Fig. 1 System Architecture

Table 2 Request Parameters for subscribing

Name	Description
sox_server	This name means connection designation.
user_name	The user name on SOXFire
user_pass	The Password of using user on SOXFire
node_name	Topic name. If prefix_match's flag is on, this name uses to search topic prefix match.
prefix_match	This is flag to judge the search method.

If a user wants to subscribe to all topics on SOXFire, the user sends a parameter that is shown above only. However, if the user wants to subscribe specifically, the user sends add to the parameter below.

Specific Topics & Prefix Match Topics

In case of user wants to subscribe to a specific topic, the user needs to send the topic name. Users can send this request by list format, so users can specify multi topics at one request. Besides, a user wants multiple topics, but too many targets, in that case, the user can specify prefix match. The user can request easily because the user only adds one parameter to the user's request.

3.2.2 Receiving Data From SOXFire to Store Database

General pub/sub IoT platform does not have a store function for publishing past data. SOXFire also does not have it, it has only cache last publish data, so if another platform wants to use past data, it must implement the function to store data. SOXCollaborator has stored function. At SOXCollaborator, it can store all data from subscribing topics. The table structure of storing data is simple, it's shown in Table.3. Many transducers are managed by topic. In addition, this table has two types of timestamps. This means stored time and published time. SOXFire has a function to cache the latest published data, so starting to subscribe time is not the same publish time necessarily. Therefore, SOXCollaborator needs to store publishing time to treat exact time.

3.2.3 Subscribing From Multi Servers

SOXFire may be installed in multiple locations. This reason is mainly the location and load distribution. At the First, location, IoT platforms often treat real-world data. Therefore, these are installed in each region, and these are managed by the region's municipality or company. Next, load distribution, IoT platforms can manage various and many sensors. Those sensors are different performance. For example, the temperature sensor publishes value high frequency. On the other hand, the weather sensor publishes a value every hour, so this frequency is low. Like this, sensors publish frequency and data size are different completely. If one IoT platform manages all sensors, the load is too high. Therefore, it needs to manage these sensors to be distributed to some IoT platforms.

Table 3 Data table on the store database

Name	Description
TOPIC_ID	Unique topic name
TRANSDUCER_ID	Each transducer name
VALUE	Value
PUB_TIMESTAMP	Timestamp when published from publisher
TIMESTAMP	Timestamp when stored on the database

From the above, SOXFire may be installed in multiple locations. By then, if clients want to use data from several SOXFire, clients have to manage connections and subscriptions from all SOXFire. This is difficult for clients, and development is hard.

Therefore, SOXCollaborator can manage some connection SOXFire. This system stores each server's information to realize, and all data stores the same database, so clients can use data easily.

3.2.4 Searching Topic and Data From SOXFire and SOX-Collaborator

SOXFire does not searching function for topics. It's not usability, so we implement search topics on the SOXFire. Clients want two pieces of information about a topic. Those are the topic name, and meta-information about the topic. First, we implemented the topic name getting function, if a client calls this function, he can get all topic names on SOXFire. Second, if the client gets a topic name, the client doesn't know the detail about the topic. We implemented the function to get the topic's detail, client calls it, he can get the detailed information on the topic. In addition, we implemented the function to check the database.

If clients subscribed to several topics, they want to check the subscribed topics. Therefore, SOXCollaborator provides the function that clients can check the subscribed topic name on the database.

From the above, SOXFire can help clients to subscribe and collaborate data on SOXFire to other systems.

4. Evaluation

We evaluated the performance of SOXCollaborator by measuring the computation cost and delay time occurring via this system.

4.1 Experimental Environments

We constructed two environments to measure performance correctly. SOXFire is constructed on the host machine, and SOXCollaborator is constructed on the virtual machine. This reason is that we do not work many processes in one environment. The host machine's spec is a Table.4 and, the virtual machine's spec is Table.5. Publisher performs on the same virtual machine because to measure delay correctly via SOXFire and SOXCollaborator. In this case, we give a bit many machine resources for the virtual machine because this machine activates two processes and the publisher works many threads.

Table 4 Host machine's spec

Name	Description
OS	macOS Big Sur(11.4)
CPU	2.4GHz 8 core Intel Core i9
MEMORY	64GB 2667 MHz DDR4
LANGUAGE	java(openjdk 14.0.2)
ACTIVATED PROCESS	SOXFire

Table 5 Virtual machine's spec

Name	Description
OS	ubunt 18.04
CPU	8 processer
MEMORY	16GB
LANGUAGE	java(openjdk 14.0.2)
ACTIVATED PROCESS	SOXCollaborator, publisher(multi)

4.2 Measure Load of Machine Resources

First, we evaluated the measure load of machine resources. SOXCollaborator saves the subscribing topic to the database, and it subscribes to all topics stored in the database when receiving the subscribing request. And, our system doesn't have a data cache, it inserts data into the database immediately when receives it. Therefore, the amount of using memory depends on the number of topics. When our system subscribes to 300 topics, the percentage of the memory is almost 2.2%. This value shows almost 352MB. In addition, we could not confirm changing this percentage when it received much data and inserted it into the database. Therefore, SOXCollaborator doesn't have a problem when users use some situations. However, if the user wants to subscribe to too many sensors, the system must waste much memory.

On the other hand, wasting the CPU processor depends on the amount of receiving data. Therefore, we evaluated the wasting it each the number of publishers. In this case, subscribing topics are fixed at 300 because the publisher and SOXFire connections are broken when 500 publishers. This reason is the conflict connections among publishers.

We evaluated wasting the CPU resource in two situations. First, publishers publish data every 0.5 seconds. This publish frequency is high, so the number of the publisher is increasing, loads of CPU also is increasing. We defined this situation as a high load situation. We show the result on Table.6. At Table.6, when publishers are more than 50%, the max load is very high. In addition, the max loads exceed 100%, when there are 300 publishers. Therefore, if the user's machine can use only one CPU processor, SOXCollaborator has limits subscribing topics. This value likes less than 100 topics. However, this situation is very rare because all sensors send very high frequency.

However, the average percentage is less than 30%. This average is not high, so if publish rate is a little down, SOXCollaborator can perform enough.

Table 6 Wasting the CPU Resource (every 0.5sec(high load situation))

PUBLISHERS	AVERAGE(%)	MAX(%)	MODE(%)
1	0.160	1	0
10	8.493	18.8	3
50	17.368	70	4
100	22.861	74	21
300	23.262	114	14

Next, we evaluated the low load situation. Publishers publish data every 10 seconds, and publishers are activated every 3 seconds because these spreads publish timing. We defined this situation as a low load situation. Table.7 shows the result.

At Table.7, all max loads are down less than the high load situation. Especially, the mode and average are very low. Fig.2 shows the of max and average of the CPU loads.

From this result, the load of CPU resources depends on published rates. In addition, from Fig.2, if too many publishers publish data at the same time, the system wastes too many CPU resources. However, that case is rare, normally our system wastes the resource less than 30% on the high load situation. This average percentage is never high, we judge SOXCollaborator has usefulness.

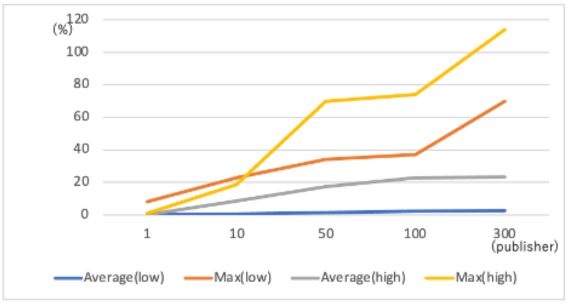

Fig. 2 Wasting the CPU Resource (all situations)

4.3 Measure Delay Time via SOXCollaborator

Finally, we evaluated delay time via our system. If the user uses this system, the user's application gets data via SOXCollaborator and SOXFire from the publisher. The user gets

Table 7 Wasting the CPU Resource (every 5sec(low load situation))

PUBLISHERS	AVERAGE(%)	MAX(%)	MODE(%)
1	0.226	8.0	0
10	0.771	22.7	0
50	1.536	34.0	1
100	2.262	37.0	1
300	2.670	70.0	2

Table 8 Delay time (the high load situation)

PUBLISHERS	AVERAGE(ms)	MAX(ms)	MODE(ms)
1	4	22	3
10	19	64	13
50	5	30	2
100	7	40	2
300	7	52	2

Table 9 Delay time (the low load situation)

PUBLISHERS	AVERAGE(ms)	MAX(ms)	MODE(ms)
1	4	17	3
10	2	14	2
50	3	27	2
100	2	41	2
300	2	22	2

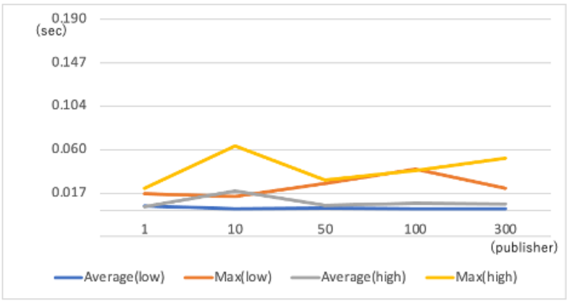

Fig. 3 Delay time (all situations)

data from the database that it connects to our system. In addition, we describe in subsection 4.1, our system can subscribe to many topics at the same time. Therefore, we measured the delay time. In this case, like measuring CPU resources, we evaluated two situations, and each the number of publishers.

First, we show the result when the high load situation on Table.8 The unit of time in milliseconds, the delay time is very little from Table.8. Especially, average time and mode time are less than 20 milliseconds.

On the other hand, we show the result of the low load situation on Table.9 and, Fig.3 shows the max and average of the delay time.

At Table.9 and Fig3, we could not confirm the large difference between high and low load situations. Therefore, we judged the no effect by the number of topics and publishers.

In addition, the delay time is very short. The most max time is only 64 milliseconds from Table.8and Table.9. Almost mode time is two milliseconds, and average times are less than 20 milliseconds.

From those results, we confirm the no problem via SOXCollaborator because the delay time is very short.

5. Future Work

In this paper, we described the system to collaborate SOXFire and blockchain marketplace. From the evaluation, our system has usefulness enough to satisfy operating. Therefore, we are planning the fieldwork using SOXCollaborator at the next step. At the fieldwork, we use SOXFire and implement a client application for a smart city. Users collect data in the city by using the client application and upload the data to blockchain marketplace via SOXFire and SOXCollaborator. Therefore, we will evaluate the usefulness of our system in the real world, and we will search for other problems.

6. Conclusion

Currently, the number of sensors and smart devices all over the world is growing day by day, and their data can be gotten easily by developing information technology. As the result, IoT is very general, and it will spread better than now.

There are too many sensors and data from those now, we need to prepare the platform to send and receive data from the various senders. We focus on SOXFire as a IoT platform because that platform provides simple data treating and managing.

Besides, when it will have realized to share many data among clients, we need to think next step. It is the open-data commerce among users. In a near future, we think the various data will have a monetary value. Then, if users must upload the buyer application by managing a private company, data commerce cannot realize seamlessly. Therefore, we must construct a market where users can do data buying/selling trust and impartiality. Blockchain marketplace is the best method to realize that market because it can provide trade-guaranteed trust and impartiality.

Therefore, we suggested and implemented the system “SOX-Collaborator” to collaborate with SOXFire and blockchain marketplace. The system helps to collaborate those. In addition, we evaluate the performance of our system. From the result, we confirm no problem that the load of machine resources and the delay time by using the system, so our system has usefulness for integration the systems. We are planning the fieldwork using SOXCollaborator at the next step. We will verify and search the other theme through the fieldwork.

Acknowledgments This work was supported (in part) by National Institute of Information and Communications Technology (NICT), Japan.

References

[1] Bedi, Bharat, Marc Carter, and Andrew Stanford-Clark. “Control of publish/subscribe messaging.” U.S. Patent Application No. 11/209,445.
[2] Bhatia, G., Rowe, A., Berges, M., and Spirakis, C. (2011). Sensor-over-XMPP.
[3] Gupta, S. S. (2017). Blockchain. IBM Onlone (http://www. IBM. COM).
[4] JS Foundation, ”Node-RED,” JS Foundation, [Online]. Available: https://nodered.org/.
[5] K. Noyen, D. Volland, D. Wörner and E. Fleisch, ”When Money Learns to Fly: Towards Sensing as a Service Applications Using Bitcoin,” 20 9 2014. [Online]. Available: https://arxiv.org/abs/1409.5841.
[6] Nofer, M., Gomber, P., Hinz, O., & Schiereck, D. (2017). Blockchain. Business & Information Systems Engineering, 59(3), 183-187.
[7] https://www.igniterealtime.org/projects/openfire/
[8] Rowe, A., Berges, M. E., Bhatia, G., Goldman, E., Rajkumar, R., Garrett, J. H., ... and Soibelman, L. (2011). “Sensor Andrew: Large-scale campus-wide sensing and actuation.” IBM Journal of Research and Development, 55(1.2), 6-1.
[9] Papadodimas, G., Palaiokrasas, G., Litke, A., Varvarigou, T. (2018, November). Implementation of smart contracts for blockchain based

IoT applications. In 2018 9th International Conference on the Network of the Future (NOF) (pp. 60-67). IEEE.
[10] Saint-Andre, Peter. “Extensible messaging and presence protocol (XMPP): Core.” (2004).
[11] Yaga, D., Mell, P., Roby, N., & Scarfone, K. (2019). Blockchain technology overview. arXiv preprint arXiv:1906.11078.
[12] Yonezawa, Takuro, et al. “Soxfire: A universal sensor network system for sharing social big sensor data in smart cities.” Proceedings of the 2nd International Workshop on Smart. 2016.