Scientific Research Publishing

Advances in Internet of Things, 2017, 7, 121-138
http://www.scirp.org/journal/ait
ISSN Online: 2161-6825
ISSN Print: 2161-6817

Analysis of Impactful Factors on Performance in Combining Architectural Elements of IoT

Shinji Kikuchi[1], Shodai Watanabe[1], Takahiro Kenmotsu[1], Daishi Yoshino[1], Akihito Nakamura[1], Takafumi Hayashi[2]

[1]Research Center for Advanced Information Science and Technology, University of Aizu, Aizuwakamatsu, Japan
[2]Faculty of Engineering, Niigata University, Niigata City, Japan
Email: shinji-k@u-aizu.ac.jp, takafumi@ie.niigata-u.ac.jp

How to cite this paper: Kikuchi, S., Watanabe, S., Kenmotsu, T., Yoshino, D., Nakamura, A. and Hayashi, T. (2017) Analysis of Impactful Factors on Performance in Combining Architectural Elements of IoT. *Advances in Internet of Things*, **7**, 121-138.
https://doi.org/10.4236/ait.2017.74009

Received: August 15, 2017
Accepted: October 16, 2017
Published: October 19, 2017

Abstract

We implemented a generalized infrastructure for Internet of Things (IoT infrastructure) to be applicable in various areas such as Smart Grid. That IoT infrastructure has two methods to store sensor data. They commonly have the features of double overlay structure, virtualization of sensors, composite services as federation using publisher/subscriber. And they are implemented as synthesizing the elemental architectures. The two methods majorly have the common architectural elements, however there are differences in how to compose and utilize them. But we observed the non-negligible differences in their achieved performance by the actual implementations due to operational items beyond these architectural elements. In this paper, we present the results of our analysis about the factors of the revealed differences based on the measured performance. In particular, it is clarified that a negative side effect due to combining independent elemental micro solutions naively could be amplified, if maximizing the level of loose coupling is applied as the most prioritized design and operational policy. Primarily, these combinations should be evaluated and verified during the basic design phase. However, the variation of how to synthesize them tends to be a blind spot when adopting the multiple independent architectural elements commonly. As a practical suggestion from this case, the emphasized importance in carrying out a new synthetization with multiple architectures is to make a balance naturally among architectural elements, or solutions based on them, and there is a certain demand to establish a methodology for architectural synthetization, including verification.

Keywords

Sensor Data Management, Loose Coupling, Overlay Architecture, Performance Analysis

1. Introduction

We implemented a generalized IoT infrastructure containing the functionalities such as message routing, message mediation, data storing and analyzing for Big Data. The main purpose of developing this IoT infrastructure was to use this for multiple evaluative experiments in various areas such as Smart Grid. Therefore, it was greatly expected as the maximum requirement that it should easily execute data aggregation with the various data used or generated by the numerous applications without a trouble. The following items are basic architectural requirements:

1) Realizing the maximum level of loose coupling and service oriented for various sensors, devices and utilities whenever connecting them are required. [BAR-1]

2) Applying an overlay structure, in order to make various sensor networks be connectable. [BAR-2]

3) Realizing the wide distributed computational environment in order to handle the huge amount and various types of data. [BAR-3]

4) Implementing the functionalities of the message routing, the data mediation to transforming among various messages' types, furthermore services that provide the abstract information models and meta data to support the above three items. [BAR-4]

In particular, [BAR-1], [BAR-2] and [BAR-4] are demanded as basic architectural requirements for realizing appropriate flexibilities needed in executing the numerous experiments. And the meaning of the maximum level of loose coupling in [BAR-1] corresponds to not only architectural aspects but also operational items such as procedures to exclude the any obstacles and constraints in practical operations. Furthermore, the overlay structure mentioned in [BAR-2] is quite general, and required as well in other platforms explained in the later section.

In order to fulfill the above set of the requirements, various elemental functionalities including several experimental mechanisms are applied. The first is, for instance, to virtualize sensors and to abstract the structures embedding these sensors. Then they are mapped into a generalized information model [1]. As a solution to the above [BAR-1], easiness of embedding sensors into a structure and flexibility for utilizing them is gained by using services with a repository to access these information models. The second is to adopt the double overlay structure consisting of three functional layers of the follows; the first layer is Communication Agent which wraps the various sensor networks and applications commonly, that could be also seen in other infrastructures. The second layer is that for the messaging routing, and the last layer is the set of storages and persistent services titled as IEEE1888 FIAP storages. Here, these storages and persistent services are treated as virtualized and independent elements. Based on the above elements, the data access services are provided as composition and federation using the publisher/subscriber based on utilizing the mentioned sto-

rage, persistent services and repository services as required in [BAR-3].

However, there are two actual implementations for storing data by changing the configuration consisting of the above three elemental functionalities commonly in order to respond to various demands on the sensors' variation. In other words, the double overlay structure, the virtualization of sensors, the composite services as federation using the publisher/subscriber are adopted commonly, but there is a difference in how to compose and utilize them. However, we observed the non-negligible differences in their achieved performance by the actual implementations due to operational items beyond features of these architectural elements. Thus, we analyzed the factors of these differences in performance. In particular, we clarified that a negative side effect due to combining independent elemental micro solutions unsophisticatedly could be amplified, if maximizing the level of loose coupling is applied as the most prioritized design and operational policy. Generally, these combinations should be evaluated and verified during the basic design phase. However, the variation of how to synthesize them tends to be a blind spot when adopting the multiple independent architectural elements as common ones. As a practical suggestion from this case, the emphasized importance in carrying out a new synthetization with multiple architectures is to make a balance naturally among architectural elements, or solutions based on them, and there is a certain demand to establish a methodology for architectural synthetization, including verification.

The remainder of this paper is organized as follows: We briefly explain about the related works in Section 2. Then, we will provide an overview of the basic architecture in Section 3. And, we mention our two equivalent system models in Section 4. This is an essential model on the viewpoint of traffic conditions, and generally it is required in performance measurement and evaluation to identify the scope of the measurement through the model of system configurations by using the equivalent model. In Section 5, we will present the results of performance measurement, especially the average throughput and average staying periods in system. In Section 6 we will demonstrate the analysis of the results by showing steps of the procedures and evaluating them. Furthermore, we will give explanations about the operational factors to cause the degraded performance. Then, we will conclude in the last section.

2. Related Works

As this research can be regarded as an integration consisting of multiple disciplines, the related works should be identified across many fields. Furthermore, there are some difficulties in evaluating this study only with the novelty against various approaches in the existing works, because the main points of the evaluation derive from comparison of the implementations between our two internal methods. Thus, we will limit our explanation about the storages for sensor data as our major area after touching on the general trends.

As for the general trends, there is a symbolic study about comparing IoT plat-

form architectures [2]. In their survey, an essential IoT reference Architecture model is defined in order to evaluate various existing reference architectures such as that by Cisco [3] for instance. Based on their model, the comparison among OpenMTC [4], FIWARE [5], Site Where [6] and Amazon Web Service IoT [7] was done. However, a storage for sensor data is regarded as an application and is not given fair consideration in [2]. One of the central roles in their survey is the part named as "IoT Integration MiddleWare". This serves as a supplier function of the overlay structure to handle the various devices when implementing them into the system. Whereas, Azure IoT Reference Architecture is a more particular instance of the IoT platform architecture including storages for sensor data explicitly [8]. In their explanation, the data model is mentioned positively as taking a largely neutral stance, however the concrete definition is not touched on anymore.

As for the area of the storages for sensor data, it could be roughly categorized into the following two generations, although the area itself is currently being updated. The first generation was to develop the functionality of persistence to store the stream data, and corresponds to the studies such as [9] and [10] in this generation. On the other hand, the second generation might be the set of studies related to Cloud Computing and Big Data. They have been related to developing the frameworks about the life cycle management of sensor data which includes generating mass data, storing them and reusing them for analysis. And this also means the context about how to deal with sensor data during the taking shape of the conceptual Big Data. In this context, the storages are regarded as more abstracted components. And one of the main topics here is how to embed the database management system such as SQL, and NoSQL. [11] [12] and [13] are examples in this generation. Our study should be identified as one of the second generation from the point of view of loose coupling and service oriented. However, [11] and [13] remain at the studies around the storages only, whereas our focus is just on the upper layer of the storages. Therefore, there is a difference of viewpoints between both.

Our study might have the same stance with web service benchmark such as [14]. However, our study does not remain at a discussion of performance merely, because our architectural design has certain dependency of how to combine the independent elemental functionalities and the result of the final design will influence the operational conditions. In this sense, our study also has another aspect of a new issue in regard to architectural synthesis. We have predicted that this new technical issue would arise more in future, as the concrete implementation in IoT matures.

3. Overview of the Basic Architecture

Figure 1 depicts the overview of the basic architecture of our IoT infrastructure. "Communication HUB towards the Internet" drawn at the center of the Figure is the information Hub charged as an important role for defining the double overlay structure. This has the functionality of a gateway connecting between the

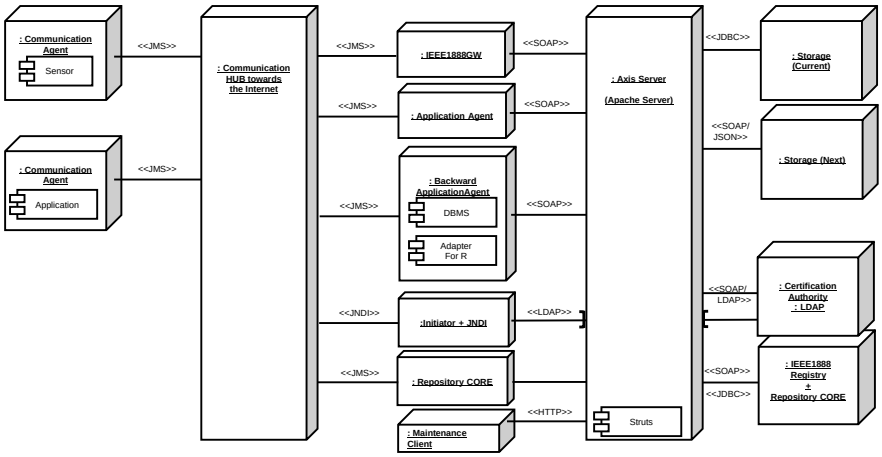

Figure 1. Overview of the entire architecture of our IoT infrastructure.

internal domains and the Internet with a firewall, and that of the messaging router. This corresponds to the previous "IoT Integration MiddleWare" in [2]. This applies the publisher/subscriber on JMS (Java Messaging Service) in order to bear the high traffic conditions. However, JMS is a typical difference from other major architectures such as AWS IoT [7], that tend to apply MQTT (Message Queuing Telemetry Transport) as the first priority. The reasons why we use JMS were due to the several given constraints from actual projects. "Communication Agent" on the left of the Figure is literally an agent of various sensor networks and applications to connect to the Communication HUB, and embedded into the environment with using JMS library. This substantially corresponds to "Gateway" in [2]. There are many kinds of standards such as IEEE-1451 in sensors area and sensor networks [15]. Furthermore, there are various legacy implementations using RS-232-C serial interfaces, various protocols and interfaces. These Communication Agents treat these differences among the above variation within a single approach, and have roles as one layer in the double overlay structure in response to the previous [BAR-1] and [BAR-2].

Set of "Storage" plus several services "IEEE1888 Registry" on the right of the Figure is a backward service to be designed for storing data from the Communication Agents in distributed deployment. They also are virtualized by using Web service to realize the previous [BAR-1] and [BAR-3]. SOAP Messaging defined in IEEE1888 can handle these variations [16]. The reason why we adopted IEEE1888 was also due to considering several constraints from actual running projects. Due to the virtualization through the double overlay structure, individual actual use case should be implemented as a composite service. Accordingly, IEEE1888GW and Application Agent as a set of subscribers are deployed at the middle of the structure, and Communication HUB will handle the various messages as input events to these subscribers. Furthermore, Repository CORE which manages the information model of the abstract sensors and provides a part of message routing functionality, is also deployed at the same position. They correspond to "Application" in [2], and the Repository CORE deals with the

matter of the previous [BAR-4]. Our IoT infrastructure has also a function of Complex Event Processing (CEP) for executing a stream processing and a mediator to transform messages; however, a detailed explanation of them is omitted here due to constraints of space.

Figure 2 shows the flow of procedures in a data flow diagram expressing another aspect from that in previous Figure 1. In order to deal with the various demands of sensors, there are two implemented methods to store the sensor data. Respectively, one corresponds to the part of "Data Conversion" at the upper-left of the Figure, and another corresponds to the part of "XML Message" at the lower-left. There are three types of data management as a characteristic of this IoT infrastructure as follows;

1) RawData(x) (x:1~n): This manages a set of the raw data from the sensors. These data sent from Communication Agents wrapping various sensors and sensor networks, are regarded as events with timestamps. And they are stored into the storages or processed as CEP events. In the case of storing, the certain data will be only inserted without any updates and deletions due to their own temporality. Accordingly, there is a demand to implement vast capability of these storages. It is substantially impossible to provide them only by local implementation, and it may also be halfway mandatory to apply the services of storages. The management systems are not always RDBMS and in many cases NoSQL or Cloud storage services applying secret sharing are also available. Thus, a Data Access Component (DAC), especially as the abstract functionality to maintain the connection resources to the data persistence, is implemented to fulfill the above requirements. For instance, in the case of using storages managed locally, a serialized object of the connection to these storages will be created through accessing Java Naming and Directory Interface (JNDI) when being demanded. This is essentially different from the Connection Pooling in order to improve the performance, and sacrifices the requirement of the performance first. On the other hand, in the case of accessing by Web services such as

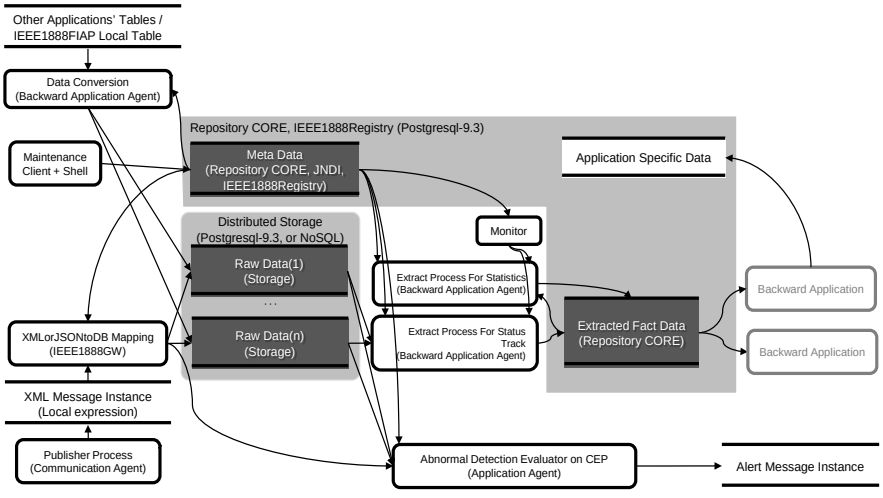

Figure 2. Overview of the entire architecture expressed in a data flow diagram.

stateless RESTful, the above instantiating process can be skipped. Furthermore, as the data will be stored over multiple storages spread and deployed widely due to their features of temporality with the growth of themselves, a part of the information for routing in querying are managed in the Repository CORE. When a demand of query arises, this routing information will be utilized.

2) Extracted Fact Data: This manages the extracted data of features of structures embedding sensors through aggregating the raw data with timestamps which are yielded by the sensors and stored in the above RawData (x). In order to extract these fact data, it is strongly demanded to access the Repository CORE frequently. As previously mentioned, the Repository CORE manages the information model about the abstract sensors and Ontology on them. Currently, it is implemented on RDBMS with allocating the specialized processors titled as two "Extract Processes" inside of the Repository CORE.

3) MetaData: This corresponds to meta data such as the information model about the abstract sensors and the Ontology. This is managed inside of the Repository CORE. Due to complicity of the model itself, these data are implemented by using RDBMS.

Figure 3 depicts the architectural configuration of the aforementioned two methods to store the sensor data. Method #1 in Figure 3 is the standardized way depicted in previous Figure 1 and corresponds to the part of "XML Message" at the lower-left in Figure 2. In this method, Communication HUB applying with JMS corresponds to the mid area between the Communication Agent and the IEEE1888GW. In order to realize the maximum level of loose coupling, a query process to extract the metadata such as data type and attributes of sensors from the Repository CORE, is included in this method. Conversely, method #2 corresponds to the part of "Data Conversion" at the upper-left in Figure 2. This method serves as fulfilling the demands for processing a data stream with sacrificing the strong demands on the loose coupling. In this way, another library of IEEE1888 processing are applied which is simplified more in spite of titling the same IEEE1888. So that, advanced capability to become executable with relatively higher performance is realized. That library is the FIAP storage developed

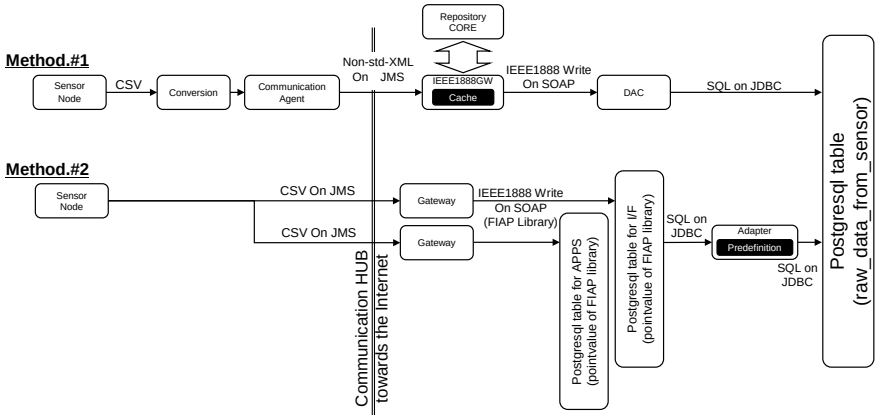

Figure 3. Architectural configuration of two methods.

by the Green University of Tokyo project [17]. The raw data yielded by sensors are once stored into the specified tables of that FIAP storage. Then, these data are transported and stored into RawData(x) in **Figure 2** by using a JDBC driver that is not regarded as loose coupling. This method #2 has certain constraints in realizing the maximum level of loose coupling in [BAR-1]. This is because of prioritizing the demands for processing a data stream. Therefore, it becomes more difficult to handle the various types of sensors, when the number of sensor types increases more as a mandate.

IEEE1888GW as a subscriber depicted in **Figure 1** has the embedded sub-functionalities of flow control and caching as its features shown in **Figure 4**. In the usual case, IEEE1888GW executes the set of normal procedures titled as step. 1, 2 and 3 in order. However, once there is a shortage of connection resources due to any reasons, for instance, high loads by other independent tasks or wasting heap areas inside of the providing Axis server are detected; an error of Out Of Memory is handled. Then, an exceptional procedure will run accordingly. In this case, the simplified handling of the error without any following up will cause another error soon in the same way, when the following other requests as step.1 arise. Therefore, the request for a temporal suppression is invoked in order to wait for release of the consumed and dominated resources by garbage collectors. In this case, when detecting a shortage of connection resources at storages, an error in IEEE1888 XML format will be informed, then the temporal suppression will be carried out at IEEE1888GW. In detail, the temporal pause in getting new requests delivered over Communication HUB will be taken by IEEE1888GW for easing storages under high loads. This execution at IEEE1888GW is due to low risks about data losing, because the handled data over Communication HUB can be temporally stored inside of internal spool at the HUB, whenever a failure in getting requests at a subscriber occurs. Therefore, it is naturally possible to make a delay in subscribing the requests. The functionality of caching inside of IEEE1888GW manages the results of queries, which are extracted from the Repository CORE with the mentioned information model about the abstract

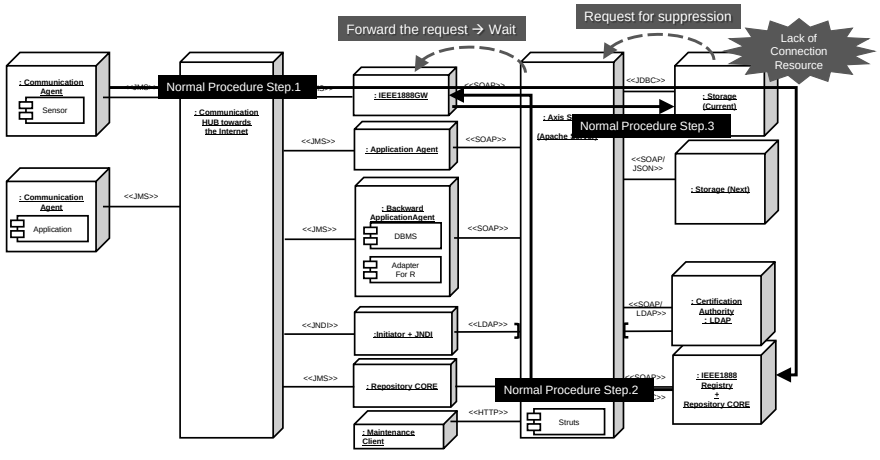

Figure 4. Outline of procedure of method #1 with flow control.

sensors and Ontology on them by doing the normal procedures in step.2. Whenever the cache results are under management after querying, the normal procedures in step.2 will be omitted with a part of step.1, as far as there are no new demands.

4. Defining Equivalent System Models

4.1. Method #1

Figure 5 shows the equivalent system model of method #1. Due to asynchronous data handling, in the equivalent system models, several parasitic queues should be expressed according to their natures. In the method #1, two natural parasitic queues arise and a substantial equivalent parasitic queue is assumed. The first natural parasitic queue takes place at the procedure where raw data from multiple sensor nodes are gathered and mapped into Communication Agent, because of making synchronicity in the procedure. The second queue arises at Communication HUB using JMS because of asynchronous data transporting with temporal storing. On the other hand, the communicational procedure in IEEE1888GW to Repository CORE and others is the SOAP messaging to Axis Servers. Therefore, a receiver thread is invoked soon without any queuing, whenever a SOAP message reaches. However, we regard this point as one with the substantial equivalent parasitic queue because of assuming hidden exclusive locking inside of the procedures. IEEE1888GW is invoked as a single thread according to "Singleton" pattern in order to simplify its internal structure and procedure. And the functionality of caching works inside it.

4.2. Method #2

Figure 6 depicts the equivalent system model of method #2. In method #2, three natural parasitic queues arise. In this case, any natural parasitic queue is not modeled at the procedure where raw data from multiple sensor nodes are mapped into Communication Agent, because the multiple sensor nodes send their own raw data independently without synchronicity. However, a natural parasitic queue arises between scope #1 and scope #2 in the Figure, because

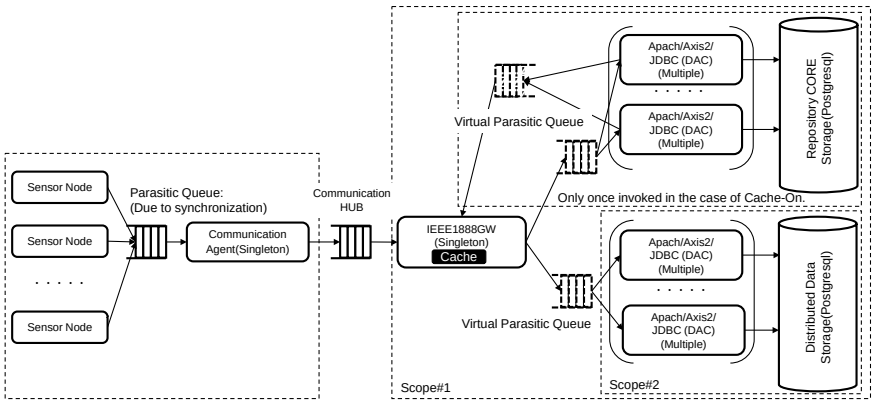

Figure 5. Equivalent system model of method #1.

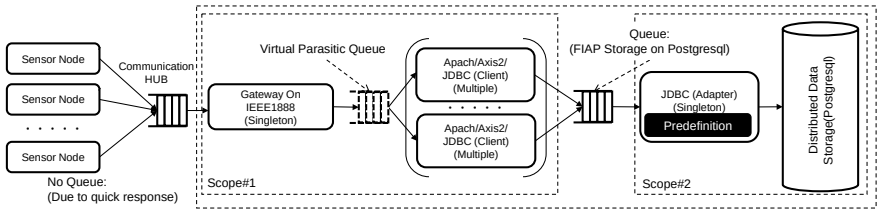

Figure 6. Equivalent system model of method #2.

FIAP Storage works as a temporal data store. Adapter is also invoked as a single thread according to "Singleton" pattern, because of retaining it as a simplified architecture from complicity caused by concurrency control. The part of the scope #2 including this adapter is executed periodically. The interval time for this invocation is specified by a parameter titled as "Wait_Period", and this will be specified explicitly in the following section. A predefined ontology expressed in XML form is preloaded at the adapter before its execution. There are some similarities between the previous caching in method #1 and this predefined ontology on the point of view of putting them on memory for excluding the needless messaging. However, this predefined ontology has an obvious weakness in regard to the maximum level of loose coupling, as it is impossible to change the contents dynamically during its running, conversely the caching can flexibly respond on demands.

5. Results of Performance Measurement

In this section, despite depicting the results, we will omit the detailed specification of the machinery environment because of space constraints for listing the set of multiple machines. In any case, we already carried out an evaluation and measurement of both methods by using almost the same or equivalent machinery environment. Our major concern here is how the performance would be influenced by variations of combination with common system components. Therefore, even a relative comparison between them without a detailed list of the machinery environment could be sufficiently capable as our evaluation.

5.1. Method #1

Figure 7 and Figure 8 shows the typical behaviors of the average throughput to respond to the dynamic state changes to confirm the effects of the flow control and the caching. In this case the average throughput is measured by counting the increase of rows of the elemental table inside RawData(x) per a second. Therefore, the targeted scope of this average throughput corresponds to the whole of the equivalent system model in Figure 5 regardless of sites with suppression. Under the initial state without a precision tuning of the whole system, the average throughput remains at 10 rows per second as maximum. The caching condition of Figure 7 is available and IEEE1888GW must query to get the necessary meta data from the Repository CORE, whenever a new type of sensor appears at the initial stage. In this case, the value of the average throughput increases more,

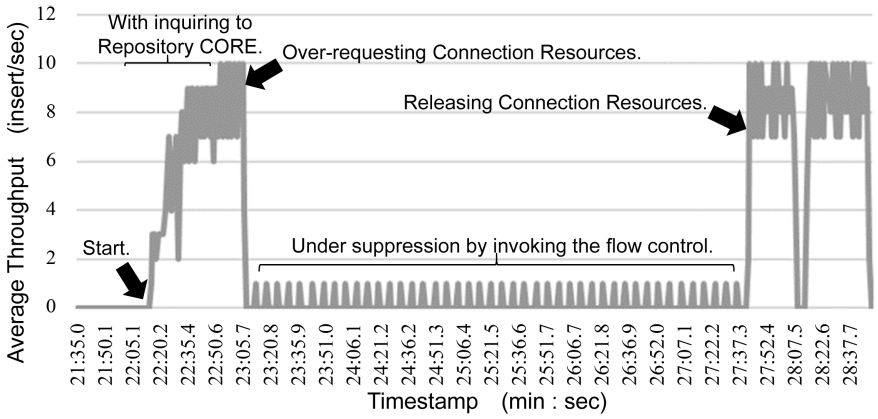

Figure 7. Average throughput in method #1 when detecting overloads at Storages.

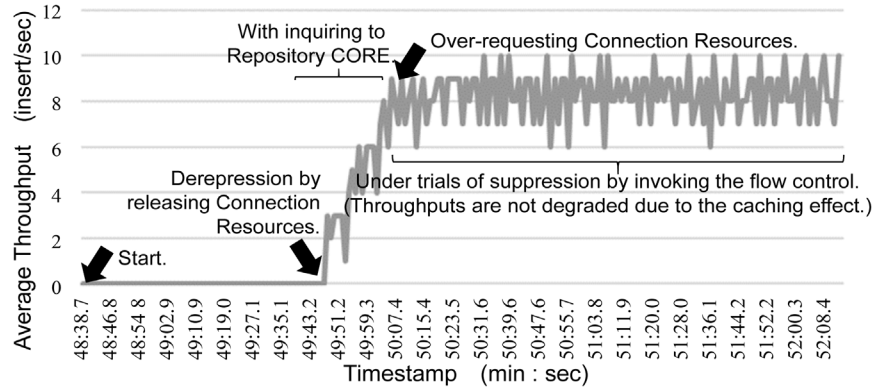

Figure 8. Average throughput in method #1 when detecting overloads at Repository CORE.

as the number of query decreases due to the caching effect. However, once an over consumption of connection resources against the constraint takes place, the request for a temporal suppression is invoked and the state is accordingly changed into waiting for the release of the resources. Thus, the average throughput temporally falls down as a degradation. After passing sufficiently beyond the occurrence of releasing the resources, a temporal pausing is broken and the average throughput regains the original performance. However, we can observe a different behavior when the procedure for a temporal suppression is applied to Repository CORE instead of the storages shown in **Figure 7**. As depicted in **Figure 8**, the inactive state continues and the average throughput also remains in low values, if starting its procedure under suppression at Repository CORE.

However, once the pausing state due to the suppression is broken, the defined procedures are carried out at a stretch, and the average throughput can recover the loss in performance. Then, the caching literally works as what it was intended for. The average throughput is maintained with its original performance because there are almost no queries to Repository CORE even under a shortage of connection resources at Repository CORE.

Figure 9 shows the result of the average staying period in systems against the number of rows inside RawData(x). This value can be captured by taking differences between following two timestamps on the stored raw data; the first one is the timestamp recorded at a sensor node and the second is the timestamp captured at the storages of RawData(x), when the raw data is inserted as a row. Both of them have the equivalent degree of precision that Network Time Protocol (NTP) can accommodate. By calculating the difference between them, we can grasp the staying period in systems from the yielding point to the end regardless of the intermediate routes. Therefore, this value should be maintained in small as much as possible. Under the initial state without a precision tuning of the whole system, this value remains around 15 seconds. Thus, there is certain room to improve the capability of the reaction of the systems. The analysis about the causes of this poor performance is mentioned in Section 6.

5.2. Method #2

Figure 10 depicts the average throughput against the number of rows inside RawData(x). And there are almost no influences from increasing rows in the shown scope, and remains at the constant level. The sudden spike of the average throughput shown around 4M rows is caused by restarting its procedures with mass messages in the spool which were stored after the tentative stopping due to an internal error of the adapter. This means that the adapter proves its sufficient capability to clear the mass requests. So, it is surely predictable that the shown constant value of the Figure might be derived from insufficient number of the raw data yielded by sensor nodes, rather than the capability of the hosting servers in performance. Therefore, the actual limitation in performance could be much higher than the observed results in this Figure. In fact, the usual average rate of CPU usage at the server, on which the adapter is deployed, remained under less than 3% and enough capability is still prepared with freed resources.

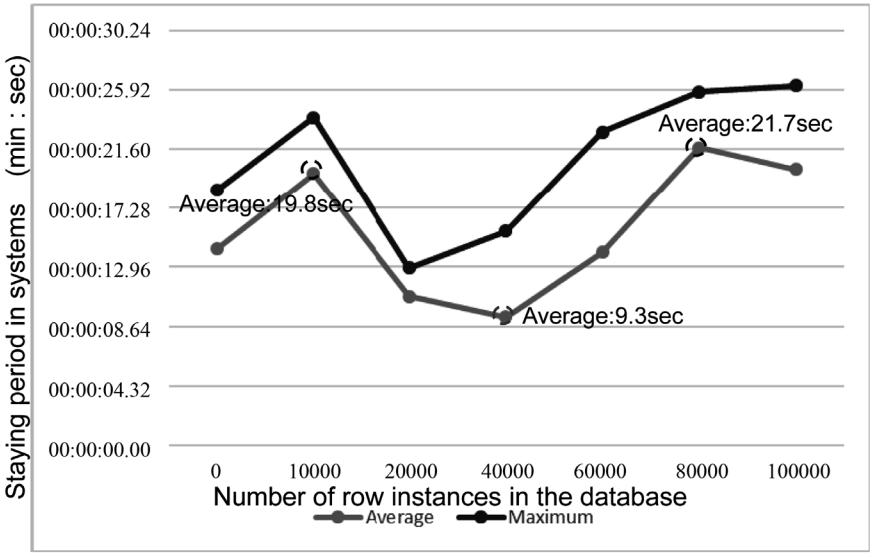

Figure 9. Staying period in systems of method #1.

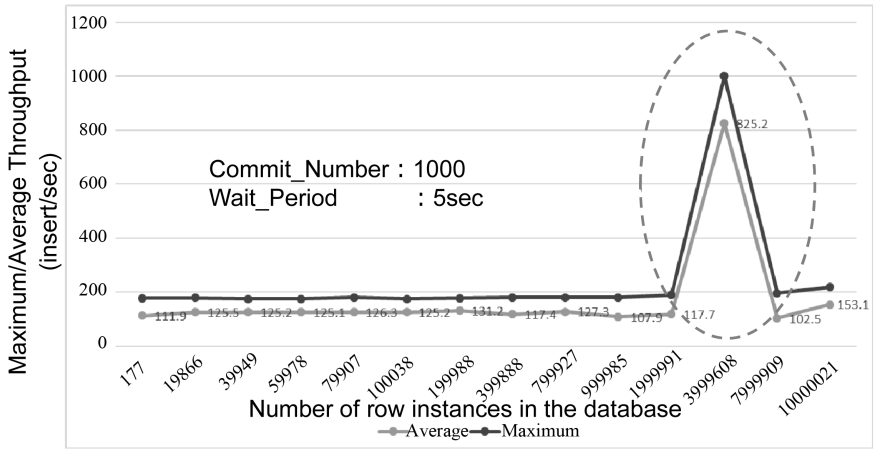

Figure 10. Average throughput of method #2.

Based on the current results, that is 100 to 150 rows per second under a usual case. So that, we could estimate the actual performance of this method #2 might reach 2K to 3K rows per second, if we can use the CPU of the server with 60% as a rate of the average usage of CPU. This means that method #2 could have its performance capability beyond 100 times of that of method #1 shown in **Figure 7**. The pursued causes in our analysis will be explained in the following Section 6. **Figure 11** shows the result of the average staying period in systems against the number of rows inside RawData(x). Individual series corresponds to the condition about Commitment Unit of a transaction. For instance, the case titled with "100 rows" means that a commitment is executed for every set of 100 rows inserted into the database. And this condition does not affect the staying periods. According to this Figure, the values of the average staying period become worse as the number of the rows increases, however they remain within several seconds. This suggests more potential performance in this method.

6. Consideration in Regards to the Factors of the Performance Gaps

As mentioned previously, the double overlay structure, the virtualization of sensors, the composite services as federation using publisher/subscriber are adopted as common elements, but there is a difference in how to compose and utilize them. In the actual performances, we have significant gaps between both methods due to some other factors, such as the operational conditions rather than just the ways in combining the above common elements. **Figure 12** individually shows the sequences of the procedures of both methods with specifying the assigned components of the individual procedure. Grayed procedures in this Figure are commonly invoked regardless of the methods. One of the primary characters of method #1 is to query to the Repository CORE at first to extract the required metadata and Ontology. However, once they are extracted and the cache manages the query results, those procedures are skipped, as far as there are no new demands. Therefore, the minimum set of the overhead in the procedures

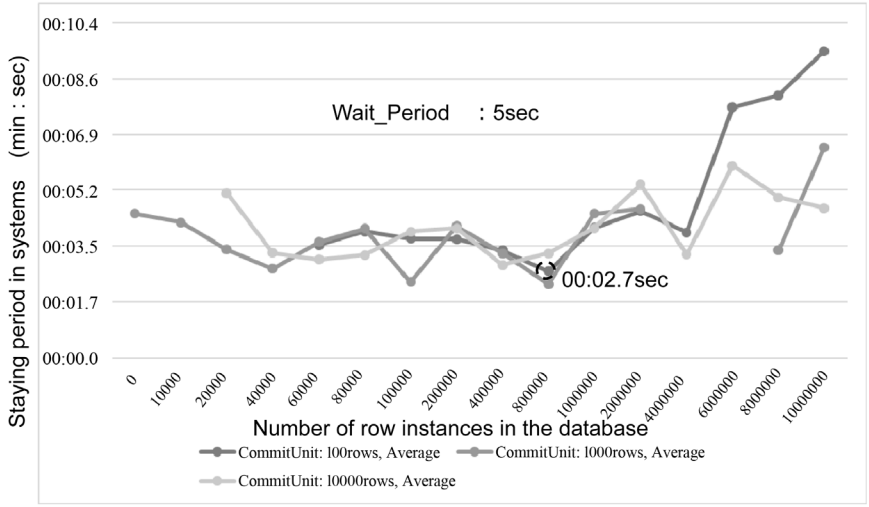

Figure 11. Staying period in systems of method #2.

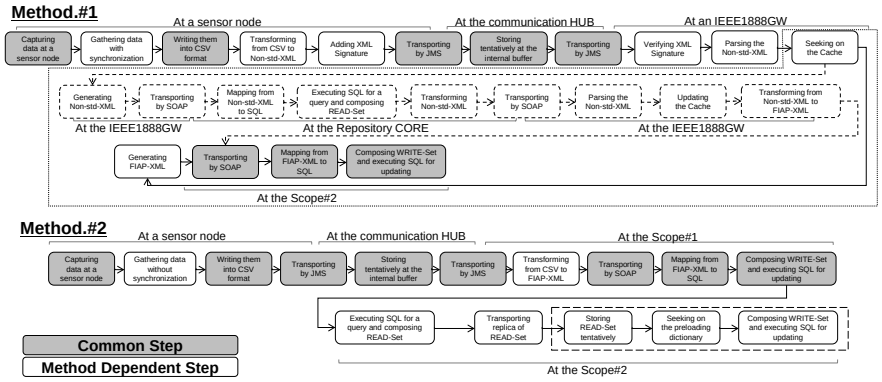

Figure 12. Steps of procedures in both methods.

are mainly identified as "seeking the data on the cache at IEEE1888GW" and "adding and verifying the XML signature" to identify the sensor nodes as whether they are permitted. Furthermore "executing the flow control" that is not explicitly described, is also included. Conversely, method #2 has twice the procedures of storing data into the storages in scope #1 and #2. In the case of applying the meta data in the cache instead of querying in method #1, the number of common procedures between both methods relatively increases. According to this, it is probably difficult to regard that minimum set of the above overhead in the procedures as a crucial factor to make both previous metrics worse.

Of course, there are other unidentified items only in comparison with Figure 12. For example, they could be a network latency of the Internet, or a possibility that a part of the facilities becomes a bottleneck. Further instance could be the simplicity of the XML formats in the communication. However, through another analysis we have already identified the most dominant factor. That is to amplify the negative effects derived from each independent elemental micro solution which is adopted for implementing the maximum level of loose coupling, through combining them. This could be identifiable as the issue in designing

solutions. In this sense, we should realize the importance of balanced design among these solutions, whenever there are multiple applicable architectural elements.

Method #1 includes several negative factors. The first factor is to increase the frequency of querying the metadata. In this method, the raw data from multiple sensor nodes are gathered, mapped their CSV forms into the XML format at Communication Agent with synchronizing, then sent to IEEE1888GW. Accordingly, IEEE1888GW would receive the set of multiple raw data from the multiple sensor nodes every time. However, the IEEE1888GW is actually implemented on the assumption that there is no preliminary information about these sensor nodes, for instance, sensor type and frequency of uploading. Thus, the query to extract these meta data is carried out every time. In this case, each query is executed for individual sensor due to prioritizing the requirement of the loose coupling in which every required access should be initialized and invoked at the demanded time. This policy invites the increase of frequency in querying. The second factor is to create a connection resource through accessing JNDI in order to make an advanced adoption of the loose coupling. However, the negative side effects by applying the loose couple are not limited only to the above. The constraints of querying individually for each sensor can dominate the following procedure to maintain the consistency. This can become another constraint about the unit of execution in data storing. As a result of these constraints, the serialization in storing the data from the sensor nodes may be caused as the third factor in spite of receiving multiple data at a time at IEEE1888GW. In particular, when the previous "Singleton" pattern is applied there, a negative influence could be given to the performance more. However, if applying the simplified multi-threads without sufficient verification in their implementation, an issue in regard to isolation at the service level could take a shape because of no transactional management at the service level. Due to combining the above three factors each other, performance degradation could be invited.

On the other hand, this issue about amplifying the negative effect by the independent elemental micro solutions, does not arise in method #2. As mentioned previously, in this method #2, the predefined ontology implemented in XML form is preloaded at the adapter before its running. In order to support this, the following is assumed; sacrificing the priority of the maximum level of loose coupling could be accepted because any sensor node is identifiable before running. Furthermore, it is not required to synchronize occurrences of raw data over multiple sensor nodes prior to storing them into the FIAP storage in the scope #1 as the front side. The following data transportation into RawData(x) in scope #2 is just less influenced. This is because there are completed correspondences between data instances at the FIAP storage of scope #1 and those at the RawData(x) of scope #2, and no room to implement any specialized procedures to map them according to data semantics. These procedures generally tend to bring a negative side effect to the operational conditions. Additionally, as a ge-

neralized batch program, it is possible to execute the commitment over the huge amount of multiple worked instances at a time. Consequently, amplifying the negative effect by the independent elemental micro solutions, does not take place anymore.

Accordingly, there should be naturally some attentions in the designing phase, for instance, performance estimation and tuning in the design, and making a delicate balance and a tradeoff among the several solutions when applying the multiple elements. It is further desired to establish these as a concrete methodology for synthesizing the multiple architectural elements. However, method #1 should not be regarded in a negative sense. In the actual operations with receiving data from a huge amount of sensor nodes, the uploaded data could be irregularly received and regarded as receiving from substantially unidentifiable sensor nodes preliminarily anytime for the backward processes, even though these nodes would be identifiable. Therefore, it is definitely required to adopt certain solutions to realize the maximum level of loose coupling as seen in the method #1. As one of our conclusions, both methods #1 and #2 should be selectable based on the features of the individual applied cases. For instance, partitioning under the shared nothing by individual unit of the sensor node, and scale out seem to be reasonable, as the method #1 is obviously difficult to be tuned any more than the its reasonable level.

7. Conclusion

We presented the outline of our IoT infrastructure having two implemented methods to store the sensor data, those methods majorly have common architectural elements in spite of the differences in how to compose and utilize them. Then, we analyzed the factors causing the differences in their achieved performance of the actual implementations. Furthermore, we pointed out that these differences are derived from the policy; whether the maximum level of loose coupling was fully pursued, or was defused with a sacrifice for maintaining performance. In particular, we also mentioned the negative side effect that is to amplify the effects negatively due to the independent elemental micro solutions which are adopted for the maximum level of loose coupling, through combining them. Primarily, these combinations should be evaluated and verified during the basic design phase. However, the variation of how to synthesize them tends to be a blind spot when adopting the multiple independent architectural elements commonly. As a practical suggestion from this case, the emphasized importance in carrying out a new synthetization with multiple architectures is to make a balance naturally among architectural elements, or solutions based on them, and there is a certain demand to establish a methodology for architectural synthetization, including verification. It is obvious that there is certain dependency on the use cases in identifying advantages and disadvantages of various architectural synthetization. However, with the above methodology for architectural synthetization including verification, the differences in measured performance shown

in this paper might be more avoidable.

References

[1] Kikuchi, S., Nakamura, A. and Yoshino, D. (2016) Evaluation on Information Model about Sensors Featured by Relationships to Measured Structural Objects. *Advances in Internet of Things*, **6**, 31-53.

[2] Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F. and Reinfurt, L. (2016) Comparison of IoT Platform Architectures: A Field Study Based on a Reference Architecture. *Proceedings in* 2016 *Cloudification of the Internet of Things* (*CIoT*).

[3] Cisco (2014) The Internet of Things Reference Model. http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

[4] Fraunhofer FOKUS (2016) OpenMTC Platform Architecture. http://www.open-mtc.org/index.html#MainFeatures

[5] FIWARE (2016) FIWARE Wiki. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Main_Page

[6] SiteWhere LLC (2016) SiteWhere System Architecture. http://documentation.sitewhere.org/architecture.html

[7] Amazon Web Services (2016) AWS IoT Documentation. https://aws.amazon.com/de/documentation/iot/

[8] MicroSoft (2016) Microsoft Azure IoT Reference Architecture. http://download.microsoft.com/download/A/4/D/A4DAD253-BC21-41D3-B9D9-87D2AE6F0719/Microsoft_Azure_IoT_Reference_Architecture.pdf

[9] Nakamura, T., Kashiwagi, K., Arakawa, Y. and Nakamura, M. (2011) Design and Implementation of New uTupleSpace Enabling Storage and Retrieval of Large Amount of Schema-Less Sensor Data. *Proceedings of* 2011 *IEEE/IPSJ International Symposium on Applications and the Internet*.

[10] Botan, I., Alonso, G., Fischer, P.M., Kossmann, D. and Tatbul, N. (2009) Flexible and Scalable Storage Management for Data-Intensive Stream Processing. *Proceedings of* 12*th International Conference on Extending Database Technology*. https://doi.org/10.1145/1516360.1516467

[11] Van der Veen, J.S., van der Waaij, B. and Meijer, R.J. (2012) Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual. *Proceedings of* 2012 *IEEE* 5*th International Conference on Cloud Computing*, 24-29 June 2012. https://doi.org/10.1109/CLOUD.2012.18

[12] Song, X., Wang, C. and Chen, Y. (2013) An Integrated Framework for Managing Massive and Heterogeneous Sensor Data using Cloud Computing. *Proceedings of* 2013 *International Conference on Mechatronic Sciences*, *Electric Engineering and Computer*, 20-22 December 2013. https://doi.org/10.1109/MEC.2013.6885113

[13] Aydin, G., Hallac, I.R. and Karakus, B. (2015) Architecture and Implementation of a Scalable Sensor Data Storage and Analysis System using Cloud Computing and Big Data Technologies. *Journal of Sensors*, **2015**, Article ID: 834217. https://doi.org/10.1155/2015/834217

[14] Juse, K.S., Kounev, S. and Buchmann, A. (2003) PetStore-WS: Measuring the Performance Implications of Web Services. 29*th International Conference of the Computer Measurement Group* (*CMG*) *on Resource Management and Performance Evaluation of Enterprise Computing Systems*.

[15] Mark, J. and Hufnagel, P. What Is 1451.4, What Are Its Uses and How Does It

Work? http://standards.ieee.org/develop/regauth/tut/1451d4.pdf

[16] IEEE Standard (2014) IEEE Standard for Ubiquitous Green Community Control Network Protocol (IEEE Std 1888-2014).

[17] Green University of Tokyo project. http://www.gutp.jp/index.html