RabbitMQ implementation as Message Broker in Distributed Application with REST web services based

Vyorbigger B. Oppier
Information System Magister
Satya Wacana Christian University
Salatiga, Indonesia
email: vyor.cs@gmail.com

Danny Manongga
Information Techology Faculty
Satya Wacana Christian University
Salatiga, Indonesia
email: dmanongga@gmail.com

Irwan Sembiring
Information Techology Faculty
Satya Wacana Christian University
Salatiga, Indonesia
email: irwan@staff.uksw.edu

Abstract*— REST web service is a technology in which it is applied commonly in the distributed enterprise application model. The high number of requests and data complexity that are received by REST web services simultaneously become a determining factor of the REST performance itself. The bigger data size that is sent then response time which is produced by REST web service also becomes high as the effect of processing that takes place in the data source. Database is one of the data sources that is generally used in distributed enterprise application which is based on REST web services. However, database implementation with data processing mechanism application according to the arrival sequence still has limitation. Technically, query consumption resulted to meet the mechanism becomes more complex. Besides that, resources that are needed are also getting higher along with the increasing of requests and data. RabbitMQ is one of the data sources and a light message broker and it adopts FIFO (First In First Out) concept in processing the data. This research also conducts implementation and evaluation of Rabbit MQ on REST web services. In addition, comparison on each of REST web services which uses single database and RabbitMQ as data storage is also conducted. It gives the output in the form of engineering on the data flow that is received by REST web services by locating RabbitMQ between the REST web services and database. This engineering is based on the performance evaluation that is resulted by each of the data source.

***Keywords:* REST web services, Distributed Application, RabbitMQ, Message Broker**

I. Preface

The high number of requests is one important part that cannot be separated in the distributed enterprise application that is based on REST web services. Request that is accepted from various locations can simultaneously give influence on the performance and response time of REST web services. It is because the data contained in the request which is sent has different sizes to be stored or to be processed again. The bigger data size that is sent and the high number of requests will give additional time in data storing and processing so that the produced response time also becomes high. At this current time, the common system used in data processing is database. However, database has some limitations in its application. Database performance may decrease when high number of requests is accepted for query process implementation. The decreasing database performance is caused by high load resources and in certain cases database does not have notification mechanism to filter the old and the new data. Database needs mechanism to determine data sequence according to its arrival so that the data received by the database meets FIFO (First In First Out) principle. This mechanism is often handled in *handcode* application by adding a new column to differentiate the old and the new data and also the data which is being processed. The process brings effect for the number of query that is written and run to process and to meet the notification mechanism. By the time REST web service continues more than one data at the same time to the database, the database will do query on each request that is sent and marks that column. The higher number of request then forces the database to do query repetition to process the new data and to update it to meet notification mechanism and data marking. Besides that, the query process and time consumption depend on the data size that is received. If query for one data needs quite long time, it will influence on the database workload that escalates significantly and it influences another data which is also received at the same time. This is generally happened in the case of big size data transfer and high number of request. In addition, database application with FIFO concept on the data will be affected on query complexity that should be implemented. High database workload will influence system scalability from the resources side or hardware. Resources that are owned will increase and addition will go along with the increasing need of high request. These limitations cause the database cannot be precise if the application serves as queue system with FIFO concept on the distributed system that is based on REST web services. Message broker is used to transfer the message between the source and the target server. With message broker, the data that is transferred from one location to another location can be faster and more precise [11], [13]. Message broker ensures that the message that is stored is success without interfering locking transaction such as needed by the database in executing query. This causes relatively smaller resource consumption than the database, but it gives better coverage. Database is good to store more structured data, but message broker use is better when it is compared with database in processing high request simultaneously. Therefore, the goal of this research is *message broker* implementation on REST web services in which the product that is used is RabbitMQ. This research brings evaluation on REST web services which use single database and *message broker* as *data-storage*. Besides that, this research conducts engineering on the data that is received by REST web services by locating RabbitMQ between REST web service and database based on the evaluation that is resulted by each data source.

II. Literature studies

There have been many researches about web services performances that use various technology, architecture, method or different scope. There are two kinds of web services that are used; they are REST and SOAP (Simple Object Access Protocol). Commonly, discussion and implementation and also web service development is influenced by some variable that are used. Those variables can be categorized into data size, kinds or data type and *response time* that is received. Data size is implemented by making some functions that receives data in form of text parameter with certain size on the function of web services. Besides, the kinds or data type is also varied; they are text, byte, or numeric [3]. From these three data types, numeric data type has smaller *response time* than text type or byte type for each of REST or SOAP web services. The smaller response time resulted then performance that is produced by web services is better. In another side, architecture that is owned by each of REST and SOAP is different in the context of handling the produced *request* and *response.* Commonly, these two kinds of web services have four main components; those are *Http Listener*, *Request Handler*, *Parse Module* and *Web Servlet* [4]. Web services performance can also be influenced by the used method. Several methods that can be used to develop web services performance in its implementation are such as compression, partial *request* and *cache* [5]. REST or SOAP web services use on the data upload implementation with the kind of data image on *mobile device* in which the data also becomes reference for performance evaluation of these two types of web services. Besides that, development and implementation of each of the web services is conducted in some variations by modify API use (Application Programming Interface), *protocol,* or *cloud system* [6]-[8]. From the research that has been done for comparison performance of REST and SOAP, it can be concluded in common that REST web services give better comparison result than SOAP web services [9]. Nevertheless, SOAP web services use can be used for more specific condition, such as for a client who needs object that is formed beside the server in real time and focuses on that object security [10].

Better comparison performance on REST web services is addressed in several sides; they are technology, framework, data size or methods that are used. Therefore, REST web service has been used much in the implementation of distributed web application at this moment than SOAP with consideration of better performance that is possessed by REST web services. This becomes the basic for this research in taking REST web service as its research object in which it is supported with other systems whether it will keep providing better performance or the reverse. That system is focused for the storage media with some variables that are classified into text data size, *response time* and data integrity. Other researches related with *message-broker* are done to provide description on its use on data processing on the distributed system, even on *cloud* technology [2], [13]. *RabbitMQ* is one technology of *message-broker* that can be used in conducting evaluation on *high-availability* and *fault-tolerant* on the *middleware* clustering [1]. The main difference of this research with the previous researches is addressed to the request flow engineering that is sent based on the evaluation of REST web services performance which uses SQLServer and RabbitMQ.

Identify applicable sponsor/s here. *(sponsors)*

A. RabbitMQ

RabbitMQ is a queue system and *message-broker* that is based on open source that is use much in processing big data amount [14]. This system provides easiness in data distribution on the communication among different system and it uses AMQP (Advance Message Queueing Protocol) as the protocol to communicate between the producer and consumer [15]. Producer on RabbitMQ is a node that conducts request in form of message that consists of the data that will be sent. The data that has been received will be continued to the customer who has a knot to RabbitMQ [16], [17]. Consumer is simpler because it only receives all RabbitMQ message by identifying payload and label which are on the message. Figure 1 shows the flow that takes place on RabbitMQ.

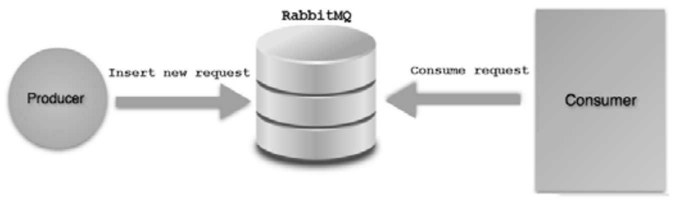


Fig 1. Workflow of RabbitMQ

III. Methodology and System Design

This research divides the system into 3 layers; they are client, REST web services, and data source. The first layer is client scope that accesses and uses service of REST web services. On this layer, every request that is sent to REST web services will be responded by REST web services with the result that has been processed. In this part, client does not know in detail how the message is processed and stored. The second layer is implementation of REST web services where on this layer it manages how data *request* is received, processed, and continued to the next layer. This layer will be built by using Microsoft MVC .NET v4 technology. The third layer is the data source in which it uses Microsoft SQLServer and RabbitMQ product. This layer is used to receive data *request* from the second layer and then process it to be stored. This research implements data flow engineering that is happened between REST web services layer and data source layer. This engineering will be based on the performance of each data source that is accessed by REST web services. It is expected that result of this engineering may bring good performance on the layer of REST web services.

Besides that, this research uses a PC server with the following specification: (1) Processor Intel Xeon CPU 2.60GHz and Memory 32GB, (2) Microsoft Windows Server Operation system 2012R2, (3). Microsoft Internet Messaging Service v8.5.9, (4) Microsoft ASP MVC NET 4 application. (4). Microsoft Sql Server 2012 R2 v12.0.4887.0. (5). RabbitMQ v3.6.6. and client PC with the following the specification: (1) Processor Intel Core i5 2.3GHz and Memory 12GB, (2) Microsoft Windows 8 Operational system. Furthermore, this research uses VPN (Virtual Private Network) to communicate between the client and server.

IV. Implementation and Discussion

Implementation and test on this research is started by building REST web services that is divided into two stages, that is by using each of SQLServer and RabbitMQ as data source on the REST web service and implementing request flow engineering that is based on the result which is obtained from the first stage. The number of request that will be tested on the first stage are 200 data requests in parallel and the scheme that will be used in this implementation uses 3 layers concept. In this concept, REST web services will be located on the second layer. This part will receive the request and continue it to the third layer of SQLServer. Figure 2 explains implementation of the first stage system.

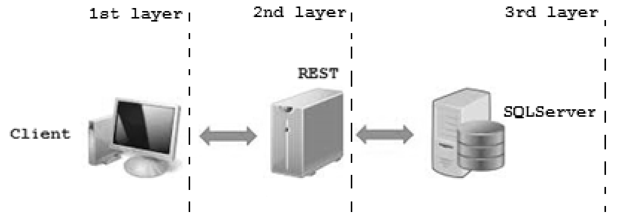
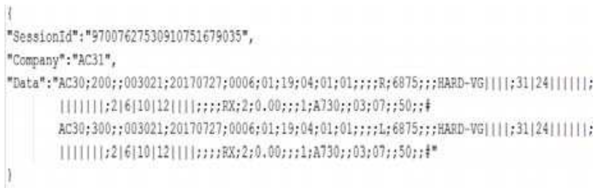


Fig 2. Implementation of the first stage system

Request that is sent by the first layer will be received by REST web services and then it is continued to the SQLServer. Response from the SQLServer will be returned to REST web service and then it will be returned to the client. From request flow until response that is produced, it gives a picture that client will receive the response which is very dependent on the second and third layer. When delay response occurs on one of those layers then it will influence to the client side as well.

Test data that is used on the request has a structure that consists of 3 properties; those are sessionid, companyid, and data. Sessionid on the structure is a property that is used to differentiate request that is sent based on the session and companyid is used to determine entity that is used, and data property shows the data that is contained in defining that data. Figure 3 shows example of data string that is sent to REST web services.



```
{
"SessionId":"970076275309107516790035",
"Company":"AC31",
"Data":"AC30;200;;003021;20170727;0006;01;19;04;01;01;;;;R;6875;;;HARD-VG||||;31|24||||||;
        |||||||;2|6|10|12||||;;;;RX;2;0.00;;;1;A730;;03;07;;50;;#
        AC30;300;;003021;20170727;0006;01;19;04;01;01;;;;L;6875;;;HARD-VG||||;31|24||||||;
        |||||||;2|6|10|12||||;;;;RX;2;0.00;;;1;A730;;03;07;;50;;#"
}
```

Fig 3. Data text or string that is sent to REST web services.

Data type that is used on the request which is sent is text or string and the format which is used is in the form of JSON (Java Script Object Notation) in which it is separated using comma (,) for every property that is possessed. Besides that, semicolon (;) is used to separate the sub-data into certain parts that will be mapped to be columns in the table. Hashtag (#) sign is used in the data property as a sign of a line that will be stored in the table. The more complex the content of the data property then the size of data text becomes bigger. Data size that is sent by client to REST web services is different, they are 100Kb, 250Kb and 500Kb and data request that is fail and succeed will be counted to see how big the implementation of REST web service is if it is used on *enterprise application.* Table I and Figure 4 shows results of data request that are succeed and fail to be received by REST web services which uses SQLServer.

TABLE I. Result of REST web services with SQLServer

Message size	**Result of REST web services**			
	Failed	***Succeed***	***Failed (%)***	***Succeed (%)***
100Kb	110	90	55	45
250Kb	156	44	78	22
500Kb	195	5	97.5	2.5

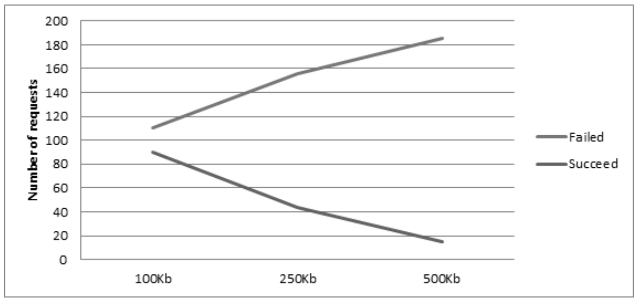


Fig 4. Result of REST web service with SQLServer

Result of the first stage explains that the number of data that works is smaller being compared to the data that is able to be received. The bigger data size is directly proportional with the failure possibility in the data sending. REST web service faces significant performance decrease on each of data size with the request that is sent in parallel for 200 data requests in certain period. Therefore, it is better that the first stage modeling usage is used for the application with average number of request that is sent is under 200 requests and data size that is sent is under 100Kb.

In order to solve the problem at the first stage, then a new layer is added between the second and the third layer. This layer is filled with the placement of *message-broker* in which it will receive all requests that are sent by the client. The choosing of RabbitMQ as *message-broker* is based on the better performance of RabbitMQ in handling bigger data and high number of request. Besides that, RabbitMQ use on the implementation of the second stage is based on the better result which is showed by RabbitMQ with similar test condition that is conducted on the first stage. Result of RabbitMQ testing can be seen on Table II and Figure 5.

TABLE II. RESULT OF REST WEB SERVICES WITH RABBITMQ

Message size	**Result of REST web services**			
	Failed	***Succeed***	***Failed (%)***	***Succeed (%)***
100Kb	3	197	1.5	98.5
250Kb	7	193	3.5	96.5
500Kb	10	190	5	95

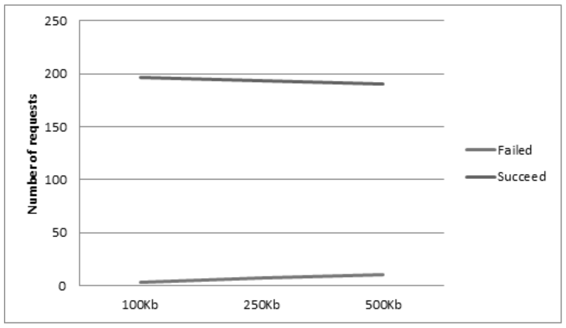


Fig 5. Result of REST web service with RabbitMQ

This result describes that the number of data that is received on REST web services which uses RabbitMQ is higher and it tends to be more stable than the number of the fail data. The number succeeded request increases although data size that is sent is bigger. Therefore, data integrity on REST web service which uses RabbitMQ is better than REST web service that uses SQLServer.

Aside from data integrity, *response time* also a factor that is used to rate the performance on REST web services [3], [4], [6]. That test is conducted by building REST web services with the use of two different data sources and those two data sources are not connected to each other. Temporary scheme of REST web services implementation to test this thing can be seen on Figure 6.

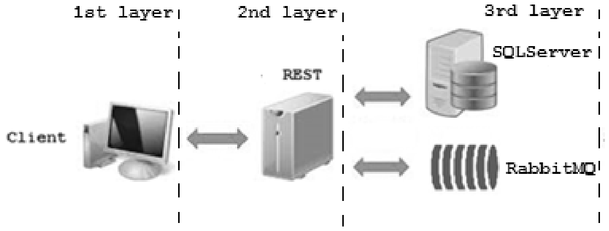


Fig 6. Implementation Scheme of REST web service with SQLServer and RabbitMQ.

Types and data size that are used is the same and appropriate with the first stage, that is 100Kb, 250Kb, and 500Kb on each of REST web services with the number of *request* for 10, 100, 1000, 1500 and 2000 data *request* sequentially. Average *response-time* will be counted for each group of that REST web services. Test result for 10 data *requests* shows that REST web service which uses RabbitMQ has smaller *response-time* on average than REST web services that uses SQLServer. Average response time on both datasource can be seen at Table III and Figure 7.

TABLE III. RESULT OF REST WEB SERVICES RESPONSE TIME (10 REQUESTS)

Average response time (ms) for various message size (100,250,500)Kb		
Client	***SQLServer***	***RabbitMQ***
Client 1	2547	1252.67
Client 2	2274	1299.33
Client 3	2000.67	1416.33
Client 4	2156.67	1075.33
Client 5	2132	1126.67
Client 6	2259.67	1026
Client 7	2088.33	1109
Client 8	2026.33	1075.33
Client 9	2317.67	1291.67
Client 10	2110.67	2214

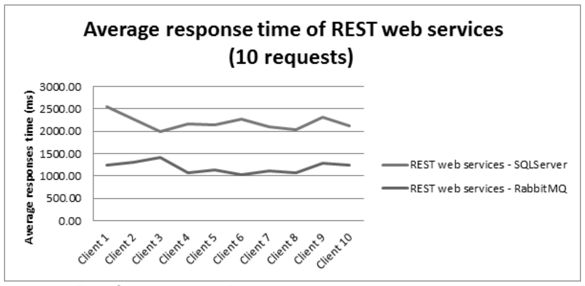


Fig 7. Result of REST web services response time (10 requests)

Besides that, the test on the group of 100, 1000, 1500 and 2000 data request also show good result on RabbitMQ. The result that is obtained on this test shows that REST web services which uses RabbitMQ still has smaller *response-time* than REST web services that uses SQLServer.

TABLE IV. Result of REST web services response time (10 requests)

Average response time (ms) for various message size (100,250,500)Kb		
Client group	***SQLServer***	***RabbitMQ***
Group 100	5552.94	1816.26
Group 1000	6468.12	1860.36

Group 1500	7452.63	1921.34
Group 2000	10413.5	1963.29

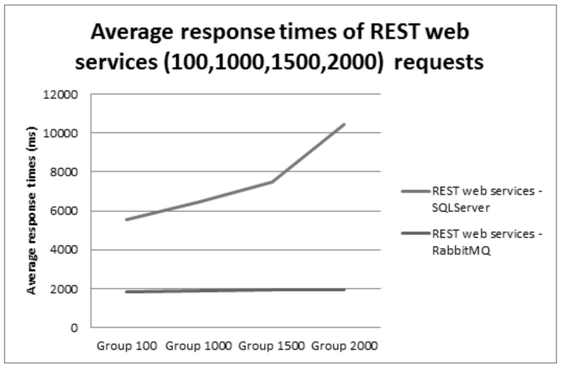


Fig 8. Result of REST web services response time (100,1000,1500,2000 requests)

Generally, the test on RabbitMQ usage on REST web service has better result than REST web service which uses SQLServer. This result covers data integrity and better *response time.* Therefore, changing or modification implementation of data request flow will be done on the second stage. This flow engineering will be based on the result of the previous test by locating RabbitMQ system between the layer of REST web service and SQLServer. Figure 9 explains flow engineering on the implementation of the second stage.

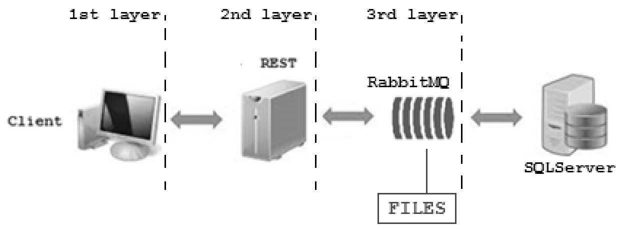


Fig 9. Layer implementation scheme on the second stage.

When client sends request to REST web services, data request will be continued to RabbitMQ system to be stored first. The data will be processed and stored in the data structure of RabbitMQ. Besides that, RabbitMQ system will conduct data backup processing in the form of flat file and this file type is locked by RabbitMQ system so that it cannot be opened directly by other systems except it goes through the protocol and mechanism of RabbitMQ itself. Backup process on RabbitMQ gives preventive step on data loss that is caused by other factors; one of them is *crash-system.* The next step from data flow which has been stored is by continuing that data to SQLServer. The response that will be generated by SQLServer will be returned to RabbitMQ and then it will be continued to REST web services to be responded to the client. Therefore, implementation of RabbitMQ on the second stage scheme is used as a bridge that connects REST web services with SQLServer. Result of RabbitMQ system performance creates positive impact on REST web services but it also does not lose momentum in the structured data processing that is owned by SQLServer. High number of request receiving and high data integrity also good validity influences performance of REST web services on the second stage implementation. Ideally, the number of request that can be received by REST web services is between 1500 until 2000 at the same time with the data on the request being queued on RabbitMQ system so that *response* process that is returned to the client side becomes better. Furthermore, in order to provide higher and better performance, clustering concept and technique can be applied on RabbitMQ. That concept may bring performance improvement on RabbitMQ system with smaller tolerant fault value [1].

V. Conclusion and Recommendation

This research concludes that use of RabbitMQ on REST web services can bring positive impact for the performance of REST web service itself. RabbitMQ system is placed as a bridge to connect REST web services with SQLServer. It is because RabbitMQ system has better performance than SQLServer system in receiving high number of requests. *Response time* that is produced by RabbitMQ has smaller value than SQLServer with data text size that is sent is varied among 100Kb, 250Kb, and 500Kb. Besides that, RabbitMQ system performs better data integrity compared to SQLServer on the clarification of high number of requests. The value of small *response time* and high data integrity provide a reference on data flow engineering on the third layer in the early stage. It is done by inserting a new layer in form of Rabbit MQ system between REST web services layer and SQLServer layer. With this engineering, RabbitMQ can bridge over data flow from REST web service to SQLServer by keep maintaining the good performance of REST web services when there is acceptance on high request and big size data. In another side, this engineering creates impact which is not too good for the implementation of the second stage. Disadvantage on this engineering makes the use of processor server becomes big. The more data that is handled by RabbitMQ is directly proportional with the high use of the processor and it needs clustering concept use on RabbitMQ. These two impacts and other *message-broker* technologies use besides RabbitMQ can be the reference for the discussion topic for further researches.

Acknowledgment

This work was supported by Information System Magister, Technology Information Faculty, Satya Wacana Christian University. I would like to thank Danny Manongga and Irwan Sembiring who support me on this collecting, preparing and writing this paper. Another thank to HOYA Vision Care, Mr. Danang Suryonugroho and all HOYA's team for great opportunities and assistances during development the system and implemented it in the wide aspect of real case enterprise distributed business application.

REFERENCES

[1] M. Rostanski, K. Grochla, and A. Seman, “Evaluation of Highly Available and Fault-Tolerant Middleware Clustered Architectures using RabbitMQ” *Proceedings of 2014 Federated Conference on Computer Science and Information Systems*, pp. 1-4, 2014.

[2] B. Meena. M, “A Distributed File Transfer using Message Broker in Cloud”, *Indian Journal of Science and Technology,* vol. 9 (48), pp. 1-4, 2016

[3] A. S. Johal, B. Singh, “Performance Analysis of Web Services for Android Based Devices”, *International Journal of Computers Applications),* vol. 92 (11), pp. 43-46, 2014.

[4] H. Hamad, M. Saad, and R. Abed, “Performance Evaluation of RESTful Web Services for Mobile Devices”, *International Arab Journal of e-Technology,* vol. 1 (3), pp. 72-78, 2010.

[5] S. Mumbaikar, P. Padiya, “Web Services Based on SOAP and REST Principles”, *International Journal of Scientific and Research Publications*, vo. 3 (5), pp. 1-4, 2013.

[6] K. Wagh, R. Thool, “A Comparative Study of SOAP vs REST Web Services Provisioning Techniques for Mobile Host”, *Journal of Information Engineering and Application,* vol. 2 (5), pp. 12-16, 2012.

[7] G. M. Tere, R. R. Mudholkar, and B. T. Jadhav, “Improving Performance of RESTful Web Services”, *International Conference Advances in Engineering and Technolog,* vol. 1 (2), pp. 12-16, 2014.

[8] K. Elgazzar, P. Martin, and H. Hassanein, “Mobile Web Services: State of The Art and Challenges”, *International Journal of Computer Science and Application,* vol. 5 (3), pp. 173-188, 2014

[9] R. Sinha, M. Khatkar, and S. C. Gupta, “Design and Development of a REST Based Web Services Platform for Applications Integration on Cloud”, *International Journal of Innovative Science, Engineering and Technology,* vol. 1 (7), pp. 385-389, 2014

[10] M. Choi, Y. Jeong, J. H. Park, “Improving Performance through REST Open API Grouping for Wireless Sensor Network”, *International Journal of Distributed Sensor Networks,* vol. 9 (11), pp. 1-13, 2013.

[11] D. Rahod, “Performance Evaluation of RESTful and SOAP/WSDL Web Services”, *International Journal of Advanced Research in Computer,* vol. 7 (7), pp. 415-420, 2017

[12] V. Kumari, “Web Services Protocol: SOAP vs REST”, *International Journal of Advanced Research in Computer Engineering and Technology*, vol. 4 (5), pp. 2467-2469, 2015.

[13] L. Magnoni, “Modern Messaging of Distributed Systems”, *Journal of Physics: Conference Series 608 012038*. pp 1-8, 2015.

[14] RabbitMQ website [online], http://www.rabbitmq.com/, accessed 30.11.2017

[15] AMQP website [online], https://www.cloudamqp.com/, accessed 30.11.2017

[16] M. Toshev, *Learning RabbitMQ* (Birmingham, UK: Packt Publishing, Ltd, 2015).

[17] A. Videla, J. W. Williams, *RabbitMQ in Action. Distributed Messaging for Everyone* (Shelter Island, NY: Manning Publications Co, 2012)

AUTHORS PROFILE

Vyorbigger B. Oppier completed his bachelor degree in Information Technology from Satya Wacana Christian University, Indonesia in 2008. He is nearly close to complete his magister degree that start in 2016 in the same university of his undergraduate program. While completing his magister degree, he also active in software engineering especially Enterprise Resources Planning and other enterprises system since 2009 until now. He already built a various enterprise system on multinasional company and also have technical certification of Microsoft Dynamics Ax 2012R3. Email: vyor.c@gmail.com

Danny Manongga finished his bachelor degree, Electronics program, in Satya Wacana Christian University, Indonesia, achieved MSc degree in Information Technology from Queen Mary College, London, and Phd degree in Information Sciences from University of East Anglia, Norwich-England. His research interests are Information Systems and Business Intelligence. Email: dmanongga@gmail.com

Irwan Sembiring, Completed his undergraduate program in UPN “Veteran” Yogyakarta, majoring in Information Technology in 2001, pursued higher degree in School of Computer Science and Electronics Gadjah Mada University, Yogyakarta, Indonesia and received Master Computer in 2004. Doctor in Computer Sciences from Gadjah Mada University ,Yogyakarta, Indonesia ,Now he is a lecturer at faculty of information technology Satya Wacana Christian University, Salatiga Indonesia. His research interests include Network Security and Digital Forensic. Email