Inversion of Web Service Invocation using Publish/Subscribe Push-Based Architecture

Thanisa Numnonda[1] and Rattakorn Poonsuph[2]

[1] School of Applied Statistics, National Institute of Development Administration, Bangkok, 10240, Thailand

[2] School of Applied Statistics, National Institute of Development Administration, Bangkok, 10240, Thailand

Abstract

Among enterprise application integration solutions, Web services technologies are promising technologies to achieve the interoperability in heterogeneous environments. However, traditional Web service invocation may lead to unnecessary network traffic, long response time, and bottleneck problems at service providers. While a publish/subscribe model provides an advantage of prompt notification which can eliminate unnecessary network traffic, its achievement in interoperability is limited. By integrating Web services technologies with a publish/subscribe model, a pull-based architecture and a push-based architecture are mentioned in this paper. The pull-based architecture uses the integrated solution based on traditional Web service invocation, still the bottleneck problems at service providers are likely to occur. Therefore, we propose an alternative, the push-based architecture which presents an innovative approach of using inversion of Web service invocation. Instead of letting service clients invoke services at service providers as usual, the service clients simply wait for updated information from the service providers. Experimental results showed that the response time was significantly minimized and the bottleneck problems at service providers were eliminated in the push-based architecture. Thus, service providers can be very small and thin in ubiquitous computing such as sensor or mobile devices.

Keywords: *Traditional Web Service Invocation, Inversion of Web Service Invocation, Publish/Subscribe, Pull-Based Architecture, Push-Based Architecture.*

1. Introduction

Sharing information between applications among or within enterprises is a major business strategy. There are several solutions and concepts for sharing information to streamline the business workflow and to apply in enterprise application integration (EAI). In recent years, service-oriented architecture (SOA) concept is significantly getting industry attention by using Web service technologies which are promising technologies to achieve interoperability in heterogeneous environments. However, SOA may also create mesh connections to multiple applications within an enterprise which is difficult to maintain. In addition, traditional Web service invocation (traditional-WSI) may lead to unnecessary network traffic and long response time when service clients are required to periodically poll for updated information. Moreover, bottleneck problems at service providers may occur when service providers confront with numerous active polling from service clients simultaneously. An overview of the polling architecture using traditional-WSI is shown in Fig. 1.

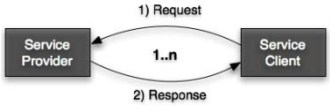

Fig. 1 An overview of the polling architecture using traditional-WSI.

Instead of using periodically polling mechanism, message-oriented middleware (MOM) and a publish/subscribe (pub/sub) model can be used for prompt notification which can eliminate unnecessary network traffic. After service providers and service clients register as publishers and subscribers respectively, all registered service clients can be notified of updated information when available. By integrating Web services technologies based on traditional-WSI with a pub/sub model, a pull-based architecture is used in the way that after service clients receive notification message, they have to send requests to service providers in order to get updated information. Therefore, when there are numerous requests simultaneously, the bottleneck problems at service providers are still likely to occur. An overview of the pull-based architecture using traditional-WSI with a pub/sub model is shown in Fig. 2.

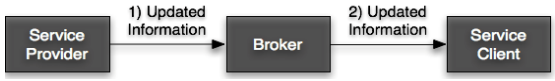

Fig. 2 An overview of the pull-based architecture using traditional-WSI with a pub/sub model.

Finally, we propose an alternative, a push-based architecture which is a complete solution for resolving shortcomings of EAI. The push-based architecture presents an innovative approach of using inversion of Web service invocation (inversion-WSI) with a pub/sub model. After service providers push Web service messages to the broker, those messages are propagated to all registered service clients. Therefore, instead of letting service clients invoke services at service providers as usual, service clients simply wait for updated information from service providers. Apparently, the response time can be reduced since service clients are able to receive updated information once available without sending any requests. The bottleneck problems at service providers are eliminated as the broker, designed to acquire high performance, is responsible for handling all services instead. Thus, service providers can be very small and thin in ubiquitous computing such as sensor or mobile devices. An overview of the push-based architecture using inversion-WSI with a pub/sub model is shown in Fig. 3.

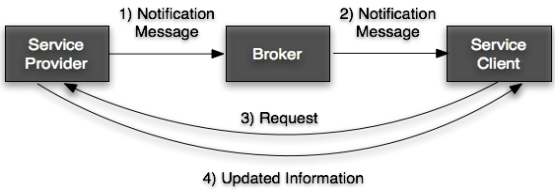

Fig. 3 An overview of the push-based architecture using inversion-WSI with a pub/sub model.

The rest of this paper is organized as follows. Section 2 provides some backgrounds on a pub/sub model and Web services technologies. Section 3 describes related works. Section 4 explains conceptual models of inversion-WSI, pull-based architecture, and push-based architecture. Section 5 clarifies research methodology in two approaches: mathematical models for total response time of pull-based and push-based architectures including comparison between them and implementation of the push-based architecture in details. Section 6 shows performance comparison results between pull-based and push-based architectures. Finally, section 7 discusses conclusion and future work.

2. Background

To share information or process among or within enterprises, EAI of heterogeneous systems is inevitable. Heterogeneous systems are normally developed by using several computer programming languages, different technologies, and deployed on various platforms. Therefore, integrating them is non-trivial. In the mid-1990s, evolution of EAI was started as enterprises tried to integrate the systems by using point-to-point connections between their applications [1]. It was successful in that era since there were only limited applications. However, the complexity of linkages between applications and difficulty of maintenance integration portions turned into problems when there were many applications. Additionally, data transformation and code conversion increased difficulty to implement. Therefore, several systematic approaches have been introduced to improve efficiency with minimal maintenance.

In the late 1990s, MOM became a very popular methodology used in EAI. The middleware concept is to allow applications to pass messages to others with single connection to MOM and more maintainable. MOM supports two types of communication: queue and topic. The queue in MOM can send a message to one consumer at a time whereas the topic in MOM with a pub/sub model is a better model and can send a message to multiple consumers concurrently [2]. This model usually consists of three basic elements: publisher, subscriber, and broker. The publisher is any application that wants to produce a message. The subscriber is any application registered to receive a copy of the message. The broker is the intermediary between publishers and subscribers. An application can be both a publisher and a subscriber at the same time. In EAI life cycle, number of publishers and subscribers can grow and shrink over time. Updated information can be either pulled by subscribers or pushed by publishers. Publishers can multicast a message of a topic to subscribers who subscribed on the topic via a broker [3]. Although publishers and subscribers are loosely coupled and transparent to each other, they are required to operate on the middleware infrastructure.

Web services technologies are promising technologies to achieve the interoperability in heterogeneous environments. An application often communicates with other applications using XML to encapsulate data and context. Using XML makes Web services platform, language, and vendor independent. As a result, Web services are ideal to be candidates for EAI solutions. Two main cores of first-generation Web services standard are SOAP (originally defined as Simple Object Access

Protocol) [4], a simple XML-based protocol and WSDL (Web Service Description Language) [5], an XML-based language to describe Web services. For traditional-WSI, any application wants to be a service provider must provide WSDL as a Web service interface. Service clients can then invoke services through stubs generated from the WSDL. An overview of traditional-WSI is shown in Fig. 4 Second-generation of Web services can form complex Web service applications. WS-* deals with aspects such as security, transactions, messaging, and notification [6].

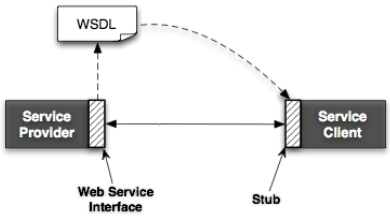

Fig. 4 An overview of traditional-WSI.

Enterprise service bus (ESB) [7], another approach to EAI, allows applications to communicate via a bus. The bus acts as a broker which basically supports multiple protocols such as being in the form of a pub/sub broker between message-based services. ESB is considered the next generation of MOM and it also extends functionality of MOM. However ESB normally requires extra level of translation which can decease performance. As a result, we decide to use the integration of Web services technologies with a pub/sub model in this research due to its interoperability and performance for EAI.

3. Related Work

Some existing specifications and works already used the integration of Web services technologies as traditional-WSI with event notifications. Two competing specifications, Web Services Notification (WS-Notification) [8] and Web Services Eventing (WS-Eventing) [9], are crucial for asynchronous Web service-based event notifications. WS-Messenger supports both WS-Notification and WS-Eventing along with mediation between them. We briefly summarize WS-Notification, WS-Eventing, and WS-Messenger including some other related works in this section.

3.1 WS-Notification

WS-Notification endorsed by OASIS, is to standardize the message protocols for topic-based or content-based pub/sub mechanisms based on Web services. There is a family of related three specifications: WS-BaseNotification (mechanisms for basic notification), WS-BrokeredNotification (intermediary brokering capability), and WS-Topics (means to categorize notifications). WS-Notification implementation and extension are described in [10-12]. WS-BrokeredNotification is very similar to the pull-based architecture of this research.

3.2 WS-Eventing

WS-Eventing is a new version and much simpler than WS-Notification. WS-Eventing basically relies upon WS-Addressing [13] for endpoint addresses. However, it only defines key pub/sub related functions such as subscribe, unsubscribe, and renew. Y.Huang et al. [14] compares WS-Notification and WS-Eventing in almost all aspects such as the delivery mode, message structure, and filter.

3.3 WS-Messenger

WS-Messenger is a project from Indiana University [15]. It aims to support both WS-Notification and WS-Eventing specifications and conveys mediation between them by using Normalization-Processing-Customization (NPC) model. WS-Messenger can reduce some overheads in SOAP message processing since it processes SOAP messages directly at the XML message level without creating data binding between XML elements and Java objects. R. Jayasinghe et al. [16] presents few approaches motivated by WS-Messenger to improve message delivery of pub/sub system at the broker.

3.4 Other Work

X. Feng et al. [17] used message-driven pub/sub system to help servers push recommended Web Services to customers based on subscribed conditions. The architecture for push-based Web service wrappers is focused by L. Brenna and D. Johansen [18], but it still uses the wrapper to regularly pull Web services. Therefore, some pull requests may return unchanged data which cause unnecessary network traffic and run down server resources. To the best of our knowledge, however, all of the related works we mentioned do not use the concept of inversion-WSI. Thus, they cannot eliminate the bottleneck problems at service providers whereas our push-based architecture can.

4. Conceptual Models

Since we use the concept of inversion-WSI in our push-based architecture, an overview of inversion-WSI is demonstrated in this section. To understand the response time comparison in the next section, sequence diagrams of

pull-based and push-based architectures are also provided here.

4.1 Inversion of Web Service Invocation (Inversion-WSI)

Instead of letting service clients invoke services at service providers as usual, service clients simply wait for updated information from service providers. This is called inversion-WSI which is an opposite of traditional-WSI. The broker is responsible for defining the canonical message comprising of a name and WSDL of a topic. A service provider must generate a stub from the WSDL so that it can invoke a service at the broker through the stub. Meanwhile, service clients must also provide Web service interfaces of the same WSDL for the broker to invoke services. Therefore, to make inversion-WSI possible, all service providers (as publishers) and service clients (as subscribers) of the same topic must use the same canonical message. An overview of inversion-WSI is shown in Fig. 5.

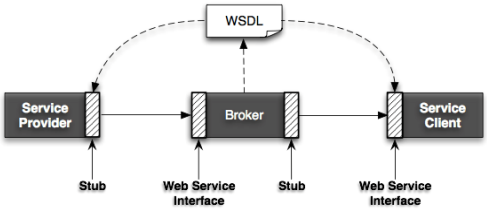

Fig. 5 An overview of inversion-WSI.

4.2 Pull-based Architecture

This architecture is based on traditional-WSI. When a predefined event occurs at a publisher, the publisher sends a notification message to a broker. The broker will then propagate that notification message to all registered subscribers. After that, subscribers have to send requests to the publisher in order to get updated information. Finally, acknowledgements must be sent from subscribers to the broker so that the broker can keep track of successful or failed transmissions. A sequence diagram of the pull-based architecture is shown in Fig. 6.

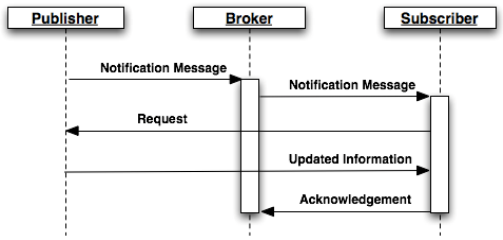

Fig. 6 A sequence diagram for the pull-based architecture.

However, this architecture has two shortcomings. First, the workload on publishers can be very high when they face numerous requests from subscribers simultaneously. Second, the response time is likely to be time-consuming since this architecture requires at least four one-way communications for a subscriber to get updated information.

4.3 Push-based Architecture

This architecture is based on inversion-WSI. The transfer of updated information is triggered by a predefined event at a publisher. The publisher first pushes the updated information to a broker, then the broker multicasts that information to all corresponding subscribers. As a result, subscriber can receive updated information without sending any requests. Acknowledgements of subscribers must be sent to the broker in order to keep track of successful or failed transmissions. Fig. 7 shows a sequence diagram to describe this architecture.

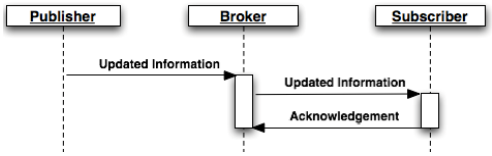

Fig. 7 A sequence diagram for the push-based architecture.

The push-based architecture is good for wide area distributed systems since publishers have no necessity to process numerous requests from subscribers. For this reason, publishers can be very small and thin. Besides, the response time for a subscriber to receive updated information is minimized to merely two one-way communications. We can briefly compare performance between three architectures shown in the Table 1.

Table 1: Performance comparison between three architectures

	Polling Architecture (traditional-WSI)	***Pull-based Architecture (traditional-WSI + pub/sub)***	***Push-based Architecture (inversion-WSI + pub/sub)***
Unnecessary Network Traffic	occur	is eliminated	is eliminated
Response Time	is long	is shorter	is shortest
Bottleneck Problems	occur	still occur	are eliminated

5. Research Methodology

To clarify our research methodology, mathematical models for total response time of pull-based and push-based architectures are provided and compared. Implementation details in two phases of the push-based architecture; pre-installation phase and runtime phase are also described in this section.

5.1 Mathematical Models

There are five steps of the pull-based architecture and only three steps of the push-based architecture to calculate the total response time. We approximate that processing time at a publisher and at a broker of sending a notification message are the same and equal to p_n. All important symbols and their meanings are listed in Table 2. Mathematical models of the pull-based and push-based architectures are shown in Fig. 8 and Fig. 9 respectively.

Table 2: Symbols and meanings

Symbol	*Meaning*
p_n	Processing time of sending a notification message
p_{pi}	Processing time at a publisher of sending updated information
p_{bi}	Processing time at a broker of sending updated information
t_n	Sending time of a notification message
t_r	Sending time of a request message
t_i	Sending time of updated information
t_a	Sending time of an acknowledgement message
p_s	Processing time at a subscriber

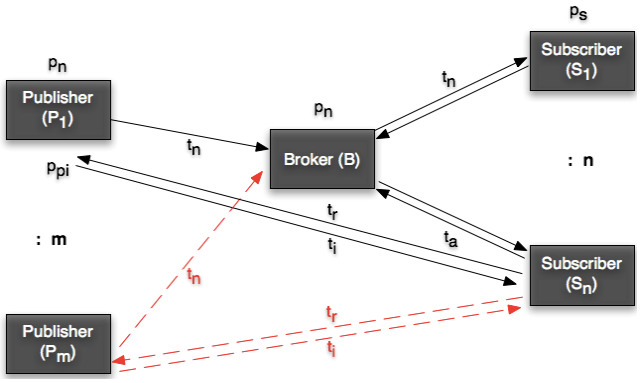

Fig. 8 A mathematical model of pull-based architecture.

Step 1: When a predefined event occurs at a publisher, the publisher sends a notification message to a broker. The summation of processing time at the publisher and sending time of a notification message from the publisher to the broker (P to B) is defined as

$$p_n + t_n$$

Step 2: After the broker receives the notification message from the publisher, it forwards that message to all registered subscribers. The summation of processing time at the broker and sending time of a notification message from the broker to all registered subscribers (B to $S_1 \ldots S_n$) is defined as

$$(n * p_n) + (n * t_n)$$

Step 3: In order to get updated information, a subscriber has to send a request message to the publisher. In this research, we assume that all registered subscribers send requests to the publisher. Therefore, the summation of processing time at the subscriber and sending time of a request message from all registered subscribers to the publisher ($S_1 \ldots S_n$ to P) is defined as

$$(n * p_s) + (n * t_r)$$

Step 4: After the publisher gets a request, it will process the request and send updated information as a response. Sending time of updated information depends on size of updated information (s_i), therefore t_i is equal to the time constant (τ) multiplied by s_i. The summation of processing time at a publisher and sending time of updated information from the publisher to all registered subscribers (P to $S_1 \ldots S_n$) is defined as

$$(n * p_{pi}) + (n*t_i) \quad \text{where } t_i = \tau * s_i \qquad (1)$$

Step 5: After receiving updated information from the publisher, a subscriber must return an acknowlegement message to the broker. The summation of processing time at the subscriber and sending time of an acknowledgement message from all registered subscribers to the broker ($S_1 \ldots S_n$ to B) is defined as

$$n * t_a$$

By summarizing all the above five steps, the formula of the total response time for m publishers of the pull-based architecture (t_{pull}) is defined as

$$t_{pull} = \Sigma_m (\text{step 1} + \text{step 2} + \text{step 3} + \text{step 4} + \text{step 5})$$

$$t_{pull} = m * ((p_n + t_n) + ((n * p_n) + (n * t_n)) + ((n * p_s) + (n * t_r)) + ((n * p_{pi}) + (n*t_i)) + (n * t_a))$$

$$t_{pull} = m * (((n+1) * (p_n + t_n)) + (n * p_s + t_r + p_{pi} + t_i + t_a))) \quad (2)$$

To simplify the formula (2) when n is large (n + 1 ~ n), the summation of the step 1 (from P to B) can be ignored. Therefore, the formula (2) can be rewritten into the formula (3) as

$$t_{pull} \sim m * n * (p_n + t_n + p_s + t_r + p_{pi} + t_i + t_a) \quad (3)$$

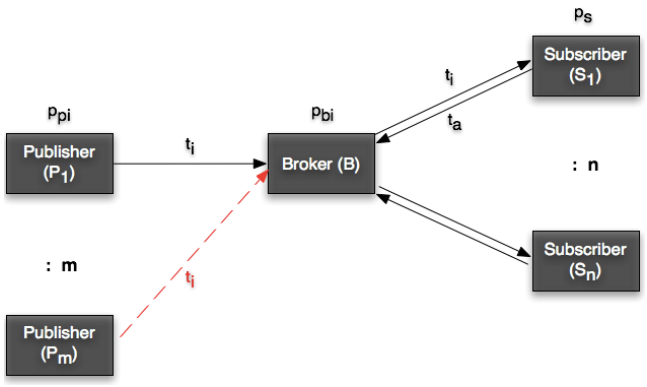

Fig. 9 A mathematical model of push-based architecture.

Step 1: When a predefined event occurs at a publisher, the publisher will send updated information to a broker. The summation of processing time at the publisher and sending time of updated information from the publisher to the broker (P to B) is defined as

$$p_{pi} + t_i$$

Step 2: After the broker receives updated information from the publisher, the broker will retrieve all endpoint addresses of registered subscribers and forward that updated information to them. Sending time of updated information depends on size of updated information (s_i), therefore t_i is equal to the time constant (τ) multiplied by s_i. The summation of processing time at the broker and sending time of updated information from the broker to all registered subscribers (B to S_1... S_n) is defined as

$$(n * p_{bi}) + (n * t_i) \quad \text{where } t_i = \tau * s_i \quad (4)$$

Step 3: All registered subscribers can receive updated information from the publisher via the broker without sending any request. After receiving updated information, a subscriber should return an acknowledgement to the broker. The summation of processing time at the subscriber and sending time of an acknowledgement message from all registered subscribers to the broker (S_1... S_n to B) is defined as

$$(n * p_s) + (n * t_a)$$

By summarizing all the above three steps, the formula of the total response time for m publishers of the push-based architecture (t_{push}) is defined as

$$t_{push} = \Sigma_m (\text{step 1} + \text{step 2} + \text{step 3})$$

$$t_{push} = m * ((p_{pi} + t_i) + ((n * p_{bi}) + (n * t_i)) + ((n * p_s) + (n * t_a))) \quad (5)$$

To simplify the formula (5) when n is large (n + 1 ~ n), the summation of the step 1 (from P to B) can be ignored. Therefore, the formula (5) can be rewritten into the formula (6) as

$$t_{push} \sim m * (p_{pi} + (n * (p_{bi} + t_i + p_s + t_a))) \quad (6)$$

Comparing the difference total response time between t_{pull} from (3) and t_{push} from (6)

$$t_{pull} \sim m * n * (p_n + t_n + p_s + t_r + p_{pi} + t_i + t_a)$$

$$t_{push} \sim m * (p_{pi} + (n * (p_{bi} + t_i + p_s + t_a)))$$

$$t_{pull} - t_{push} \sim m * ((n * (p_{pi} - p_{bi})) + (n * (p_n + t_n + t_r + t_i))) \quad (7)$$

Let δ be the difference processing time between at a publisher and at a broker of sending updated information, $\delta = p_{pi} - p_{bi}$, the formula (7) can be rewritten into the formula (8) as

$$t_{pull} - t_{push} \sim m * n * (\delta + p_n + t_n + t_r) \quad (8)$$

If the publisher and broker have the same specifications (δ=0), the difference total response time between the pull-based and push-based architectures will depend mainly on ($p_n + t_n + t_r$) as shown in the formula (9). However, the processing time at the broker is normally much less than that of the publisher and if δ is much greater than ($p_n + t_n + t_r$), the difference time between the pull-based and push-based architectures is mainly depended on δ as shown in the formula (10).

$$t_{pull} - t_{push} \sim m * n * (p_n + t_n + t_r) \quad \text{when } \delta=0 \quad (9)$$

$$t_{pull} - t_{push} \sim m * n * (\delta) \quad \text{when } \delta >> (p_n + t_n + t_r) \quad (10)$$

5.2 Implementation of the Push-based Architecture

Implementation processes of the push-based architecture are set up and carried out based on the following conditions:

- There is only one operation in unique WSDL per topic.
- There can be multiple publishers per topic and a publisher can also be a subscriber of the same topic. It means that one or more publishers can simultaneously publish similar messages to subscribers of the same topic.
- When a publisher, at the same time acts as a subscriber of the topic, sends a message to the broker, the broker will handle this situation by not sending that message back to the publisher.

There are two phases of the push-based architecture needed to be explained: pre-installation phase and runtime phase. For the pre-installation phase, to be able to make inversion-WSI possible, any application interested to be a source of updated information must register as a publisher of a topic at a broker. After that, it can obtain WSDL of that topic from the broker. Any application interested to receive the updated information must get a related file to the WSDL of the topic from the broker and can implement in its desired way. Finally, interested subscribers must register and provide their endpoint addresses to the broker. An overview of the pre-installation phase of push-based architecture is shown in Fig. 10.

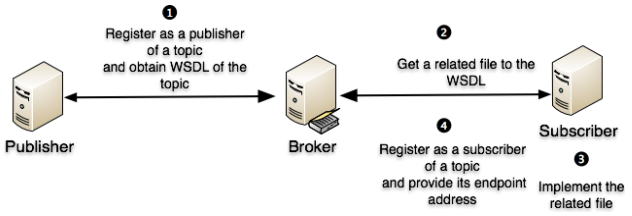

Fig. 10 An overview of the pre-installation phase of push-based architecture.

For the runtime phase, when a predefined event occurs at a publisher, the publisher invokes Web service at a broker and the broker then invokes Web services of all registered subscribers. Each subscriber must return an acknowledgement back to the broker so that the broker can keep track of successful or failed transmissions. An overview of the runtime phase of push-based architecture is shown in Fig. 11.

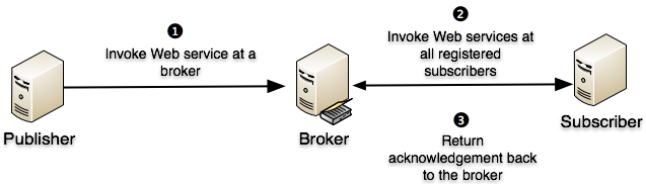

Fig. 11 An overview of the runtime phase of push-based architecture.

6. Experimental Results

In this section, experimental results are presented for performance comparison between pull-based and push-based architectures. The simulation of both architectures was set up within the same running environment such as the same processor speed and the same network bandwidth. We used twelve identically configured machines: Pentium 2.4 GHz and 2GB of RAM on Window 7 for both architectures. Ten machines were used for subscribers with up to 10 simulated subscribers on each machine. Each subscriber on the same machine was operated on a separated but identical server. A broker and a publisher each acquired own machine.

We started to measure the total response time after the broker received a notification message or updated information from the publisher. The summation time from the publisher to the broker was ignored as explained from the formula (2) to (3) of the pull-based architecture and from the formula (5) to (6) of the push-based architecture. The total response time would end after the broker received acknowledgements from all subscribers. We experimented in 20 times of each total response time and calculated the average of them.

Three scenarios were experimented to find the average total response time with following factors:

1. The number of subscribers was increased from 10 to 100 with a step of 10 by fixing the size of updated information to 1 Kbyte and using only 1 publisher.
2. The size of updated information was increased from 4Kbyte to 40Kbytes with a step of 4 by fixing the number of subscribers to 40 and using only 1 publisher.
3. The number of publishers was increased from 1 to 10 with a step of 1 by fixing the number of subscribers to 20 and the size of updated information to 1 Kbyte. Some publishers may also be subscribers of the same topic.

Experimental results of the first scenario are shown in Fig. 12. When the number of subscribers was increased from 10 to 20 and continuously into 100, we found that average total response time of the pull-based architecture was rising higher than that of the push-based architecture. Since the same specification of machine was used for the broker and the publisher (δ=0) and the number of publisher was fixed to 1 (m=1), ($p_n + t_n + t_r$) in the formula (9) could be approximated to 10 ms. Therefore, the number of subscribers (n) is influential to difference total response time between the pull-based architecture and the push-based architecture by around n * 10 ms.

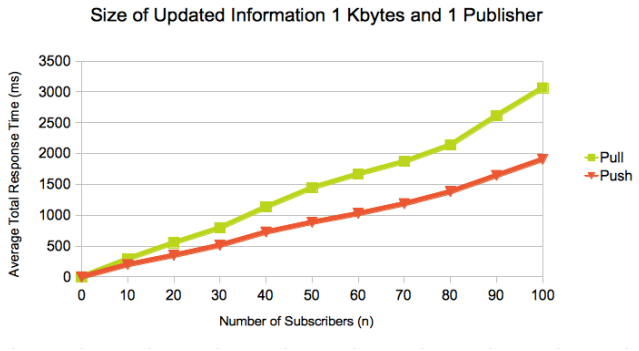

Fig. 12 Average total response time when the number of subscriber was increased.

Experimental results of the second scenario are shown in Fig. 13. When the size of updated information was increased from 4Kbytes to 8Kbytes and continuously into 40Kbytes, we found that difference average total response time between the pull-based architecture and the push-based architecture remained nearly the same. From the formula (3) and (6), both architectures need to send updated information ($t_i = \tau * s_i$ in the formula (1) and (4)), thus the difference time does not rely on the size of updated information. In this scenario, we set up 4 subscribers per machine for 10 machines to be the total of 40 subscribers.

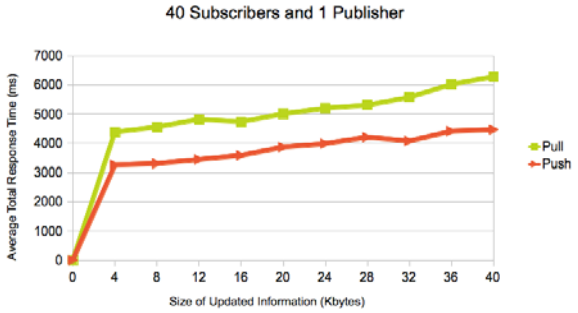

Fig. 13 Average total response time when the size of updated information was increased.

To be able to measure the average total response time from many publishers, lots of notification messages or updated information were sent out from publishers in the pull-based and push-based architecture respectively. Experimental results of the last scenario are shown in Fig. 14. When the number of publisher was increased up to 5, the average total response time of the pull-based architecture was higher than that of the push-based architecture. However, when the number of publisher went beyond 5, bottleneck problems occurred in the pull-based architecture which caused the total response time could not be determined. The main reason of the bottleneck problems came from that some publishers who at the same time acted as subscribers were not able to handle concurrent Web services invocation properly. In this scenario, we set up 2 subscribers per machine for 10 machines to be the total of 20 subscribers.

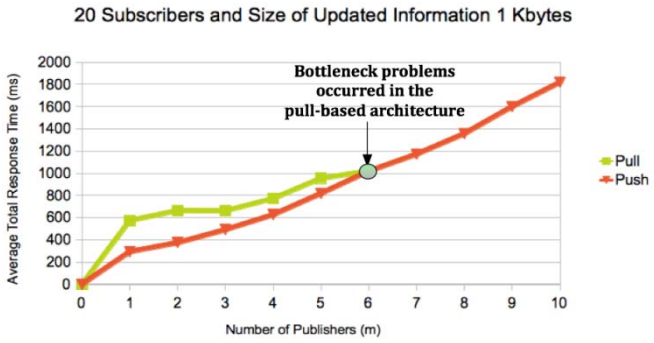

Fig. 14 Average total response time when the number of publisher was increased.

7. Conclusion and Future Work

The results of this research show that the push-based architecture surpasses the pull-based architecture by integrating Web services technologies with a pub/sub model. Using inversion-WSI instead of traditional-WSI, the broker is a core component of the push-based architecture. It acts as a middleware for receiving and sending updated information, as well it may perform several functions such as data transformation, code conversion, and conditional routing. Therefore, the broker should be designed to be able to handle heavy workloads whereas service providers can be very small and thin.

Since the push-based architecture can significantly minimize overall response time and workload on service providers, it is potentially applicable for some machine-to-machine (M2M) applications that need to speedily distribute updated information in urgent situation such as Tsunami alert system. The push-based architecture can efficiently support tracking updated information for many purposes as well.

Although this paper does not mention about the security, quality of service (QoS), and transaction, the concept of Web Services Atomic Transaction (WS-Atomic Transaction) [19] can be applied to enhance the reliability which is considered to be our future work. For further work, service clients may be able to choose which data

they want to receive via the pull-based architecture or the push-based architecture. The factor to choose between architectures may depend on a category or size of updated information. This may be called hybrid Web service invocation (hybrid-WSI).

Acknowledgments

This research was partially funded by the National Institute of Development Administration (NIDA). We also would like to thank the School of Applied Statistics of NIDA for providing us supports on this research work.

References

[1] J. Lee, K. Siau, and S. Hong, "Enterprise integration with ERP and EAI", in Communications of the ACM, 2003, 46 (2), pp. 54–60.
[2] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Pub/sub", in ACM Computing Surveys, 2003, 35 (2), pp. 114-131.
[3] I. Gorton, Essential Software Architecture, Springer, 2006.
[4] W3C, SOAP Version 1.2 Part 1, Available: http://www.w3.org/TR/soap12-part1/
[5] W3C, Web Services Description Language (WSDL) Version 2.0 Part 1, Available: http://www.w3.org/TR/wsdl20/
[6] Erl, Thomas, SOA Principles of Service Design, Service Oriented Computing Series, Prentice Hall, 2007.
[7] J. Wu and X. Tao, "Research of Enterprise Application Integration Based-on ESB", in 2nd International Conference on Advanced Computer Control (ICACC), 2010.
[8] OASIS, Web Services Notification (v 1.3), Available: http://docs.oasis-open.org/wsn/
[9] W3C, Web Services Eventing, Available: http://www.w3.org/Submission/WS-Eventing/
[10] M. Humphrey, G. Wasson, K. Jackson, J. Boverhof, M. Rodriguez, Joe Bester, J. Gawor, S. Lang, I. Foster, S. Meder, S. Pickles, and M. McKeown, "State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations", in 4th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), 2005.
[11] A. Quiroz and M. Parashar, "Design and Implementation of a Distributed Content-based Notification Broker for WS-Notification", in Grid Computing Conference, 2006.
[12] S. D. Labey and E. Steegmans, "Extending WS-Notification with an Expressive Event Notification Broker", in 2008 IEEE International Conference on Web Services, 2008.
[13] W3C, Web Services Addressing 1.0 – Core, Available: http://www.w3.org/TR/ws-addr-core/
[14] Y. Huang and D. Gannon, "A Comparative Study of Web Services-based Event Notification Specifications", in Proceedings of the 2006 International Conference on Parallel Processing Workshops (ICPPW'06), 2006.
[15] Y. Huang, A. Slominski, C. Herath, and D. Gannon, "WS-Messenger: A Web Services-based Messaging System for Service-Oriented Grid Computing", in Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), 2006.
[16] R. Jayasinghe, D. Gamage, and S. Perera, "Towards Improved Data Dissemination of Publish-Subscribe Systems", in 2010 IEEE International Conference on Web Services, 2010.
[17] X. Feng, F. Xue, and T. Zhang, "Research on data exchange push technology based on message-driven", in 2009 International Joint Conference on Artificial Intelligence", 2009.
[18] L. Brenna and D. Johansen, "Configuring Push-Based Web Services", in Proceedings of the International Conference on Next Generation Web Services Practices (NWeSP'05), 2005.
[19] OASIS, WS-AtomicTransaction (v 1.2), Available: http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec.html

Thanisa Numnonda is a Ph.D. candidate at the School of Applied Statistics, National Institute of Development Administration, Bangkok, Thailand. She received a Master's Degree in Computer Engineering from the Department of Electrical and Computer Engineering, University of Southern California, USA. Her research interests are in Web Services and Service-Oriented Architecture.

Rattakorn Poonsuph is a lecturer at the School of Applied Statistics, National Institute of Development Administration, Bangkok, Thailand. He received a Ph.D. degree in Computer Science from University of Massachusetts Lowell, USA. His fields of research include Software Engineering and Software Architecture. He has published several papers in various international conferences.