Performance Evaluation of OM4SPACE's Activity Service

Irina Astrova
Institute of Cybernetics
Tallinn University of Technology
Tallinn, Estonia
irina@cs.ioc.ee

Arne Koschel
Alexander Olbricht, Matthias Popp
Faculty IV, Department for Computer Science
University of Applied Sciences and Arts Hannover
Hannover, Germany
akoschel@acm.org

Marc Schaaf
Stella Gatziu Grivas
Institute for Information Systems
University of Applied Sciences Northwestern Switzerland
Olten, Switzerland
marc.schaaf@fhnw.ch

***Abstract*—OM4SPACE provides cloud-based event notification middleware. This middleware delivers a foundation for the development of scalable complex event processing applications. The middleware decouples the event notification from the applications themselves, by encapsulating this functionality into a component called Activity Service. This paper presents preliminary results of the performance evaluation for the Activity Service.**

***Keywords*—OM4SPACE; Activity Service; WebLogic JMS; Amazon SQS; Event-Driven Architecture (EDA); Service-Oriented Architecture (SOA); Complex Event Processing (CEP); cloud computing.**

I. INTRODUCTION

In 2010, the *University of Applied Sciences Northwestern Switzerland* in cooperation with the *University of Applied Sciences and Arts Hannover Germany* started a project called OM4SPACE [1]-[6]. The idea behind OM4SPACE was to merge Event-Driven Architecture (EDA), Service-Oriented Architecture (SOA), Complex Event Processing (CEP) and cloud computing together to provide cloud-based event notification middleware for *decoupled* communication between CEP application components on all the layers of a cloud stack, including infrastructures, platforms, components, business processes and presentations (see Figure 1). By decoupled, we mean that events are posted to the middleware without knowing if and how they are processed later.

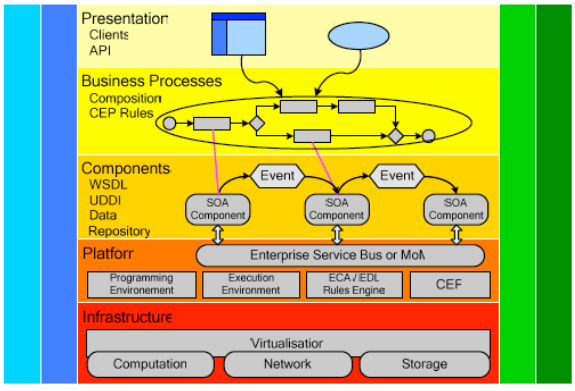

Figure 1. Cloud stack [3].

Performance is typically one of the top evaluation criteria for middleware products in general and OM4SPACE in particular. Since OM4SPACE is still relatively new, users expect that it will continue over time to improve its functionality, usability and reliability. However, users typically do want to get the best performance possible. Since the user's level satisfaction with OM4SPACE is largely determined by its performance, in this paper we evaluate the performance of OM4SPACE's Activity Service.

The rest of the paper is organized as follows. Section II presents the architecture of OM4SPACE. Section III describes the performance tests run against OM4SPACE. Section IV summarizes the results obtained during the performance tests and outlines future directions in the development of OM4SPACE.

II. ARCHITECTURE

Figure 2 gives an overview of the architecture of OM4SPACE, which includes the following components: *Event Producers* (also called *Event Sources*), *Event Consumers* and *Activity Service*.

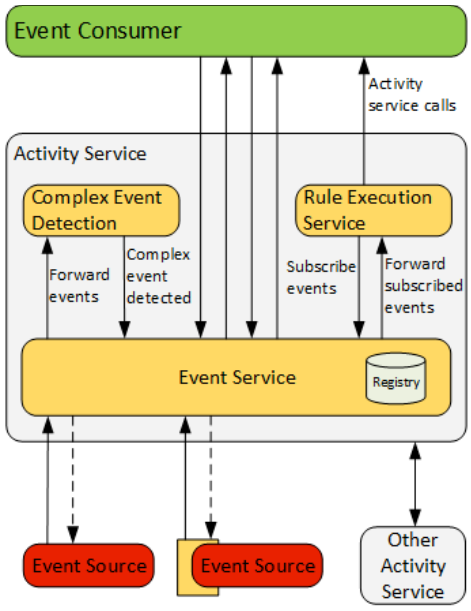

Figure 2. Architecture of OM4SPACE [3].

 ISBN: 978-1-61208-337-7

The Activity Service itself includes the following components:

- *Event Service:* This component receives events from Event Producers, pre-processes the events and delivers them to Event Consumers subscribed for those events. The Event Service contains a registry. Event Consumers look up events in the registry. If an Event Consumer finds an event of interest, it subscribes to that event.
- *Complex Event Detector:* This component receives events from the Event Service and derives from them new complex events, which are fed back into the Event Service for further processing.
- *Rule Execution Service:* This component receives events from the Event Service, evaluates them against CEP rules and triggers the rules into execution, which results in the execution of external action handlers that are provided by other third-party components.

The communication between all the components in the architecture is done through events, where an event is any kind of information sent as a *notification* from one component to another.

III. PERFORMANCE EVALUATION

One of the main advantages offered by OM4SPACE is its independence of channel service providers such as WebLogic, Amazon and Google because the Activity Service enables the transparent use of different transport technologies.

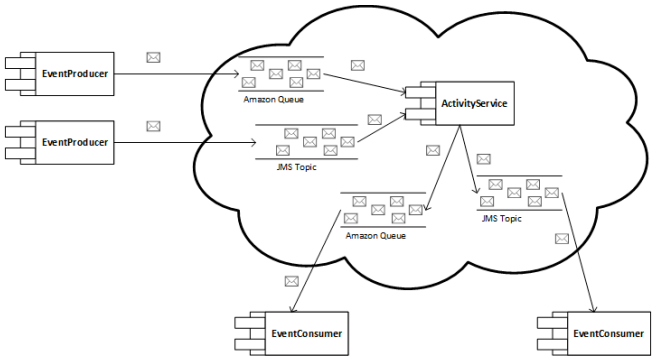

Figure 3. Transport technologies used by Activity Service.

In the current version of OM4SPACE, the Activity Service supports the following transport technologies:

- WebLogic JMS, which serves as an example of a *topic* service.
- Amazon SQS, which serves as an example of a *queue* service.

Once an Event Producer has sent events to the channel, the Activity Service located in a public cloud will forward the events to the channel of an Event Consumer that is subscribed for those events (see Figure 3). A decision on which channel to use for sending events is left solely to the Event Producer. Similarly, a decision on which channel to use for receiving events is left solely to the Event Consumer. For example, the Event Producer can select a JMS topic because it is not chargeable, whereas the Event Consumer can select an SQS queue because it is highly available (i.e., the availability of an SQS queue is not affected if the cloud instance fails).

A. Tests

We conducted the performance evaluation to answer the following questions:

- Will the Activity Service (sitting between the Event Producer and the Event Consumer) affect the time needed for events to reach their destination?
- If it does, will the performance still be good?

The answers to these questions were important because the application areas for OM4SPACE include smart grids [6] that need to address the challenges related to the constantly increasing number of events and near real-time reaction on those events.

To answer the questions above, we performed the following tests:

- *T1*: The Activity Service was not used. Events were sent via a JMS topic and received via the same topic.
- *T2*: The Activity Service was used. Events were sent via a JMS topic and received via another JMS topic.
- *T3*: The Activity Service was not used. Events were sent via an SQS queue and received via the same queue.
- *T4*: The Activity Service was used. Events were sent via an SQS queue and received via another SQS queue.
- *T5*: The Activity Service was used. Events were sent via a JMS topic but received via an SQS queue.
- *T6*: The Activity Service was used. Events were sent via an SQS queue but received via a JMS topic.

These tests were intended to prove or disprove the following hypotheses:

- *H1*: JMS alone can achieve better performance than JMS interconnected with the Activity Service.
- *H2*: SQS alone can achieve better performance than SQS interconnected with the Activity Service.
- *H3*: There can be a difference in the performance of JMS alone and SQS alone.
- *H4*: There can be a difference in the performance of JMS interconnected with the Activity Service and SQS interconnected with the Activity Service. This difference can be the same as above.
- *H5*: The number of events can affect the performance of JMS alone.
- *H6*: The number of events can affect the performance of SQS alone.
- *H7*: The number of events can affect the performance of JMS interconnected with the Activity Service.
- *H8*: The number of events can affect the performance of SQS interconnected with the Activity Service.

We performed the tests in the following way:

 ISBN: 978-1-61208-337-7

- Each test was executed with a different number of events (100, 500 and 1000) to see how the event number affects the performance.
- Each test was executed ten times to calculate the average where outliers were still visible.
- In each test, the time from sending the first event to receiving the last one was measured using a Java method `System.currentTimeMillis` (which returns the current time in msecs).
- Depending on the test, either all the components (the Event Producer, the Activity Service and the Event Consumer) were running on the same cloud instance or each component was running on its own cloud instance. Because of the decision to use SQS, Amazon EC2 was used as the cloud. Generally, Event Producers and Event Consumers are not limited to the components of a public cloud where the Activity Service is located. Rather, they can be located in private clouds or in some other public clouds (see Figure 6).

The measurements were made with two Ubuntu Linux 9.10 systems, which both used Sun Java 1.6.0. The machine, which hosted the Event Producer, the Activity Service and the Event Consumer, was a dual core system with 4GB memory. The machine for the cloud was a quad core system with 8GB memory. The two machines were interconnected with a gigabit Ethernet.

B. *Test Results*

The test results proved H1, H2, H3, H5, H6, H7 and H8, but disproved to some degree H4.

Figure 4 summarizes the test results for T1 and T2. What attracts our attention is a very good performance that JMS demonstrated in all the tests. For example, sending and receiving 100 events via JMS interconnected with the Activity Service took only 1286 msecs. But as one could expect, this time was longer than without the Activity Service.

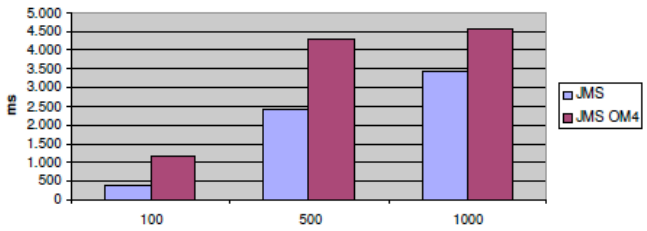

Figure 4. Sending and receiving 100, 500 and 1000 events: JMS alone vs. JMS interconnected to Activity Service.

One could expect that the time would increase with an increase of the number of events. Indeed, for sending and receiving 500 events, JMS interconnected with the Activity Service needed 3184 msecs more than for sending and receiving 100 events. However, of peculiar interest is the fact that for sending and receiving 1000 events, JMS interconnected with the Activity Service needed only 305 msecs more than for sending and receiving 500 events. In both cases, the average time was about 4500 msecs. Therefore, we suggest that extra time needed for sending and receiving 100 events was the time that the Activity Service needed for initialization.

The left column in Table I shows the time needed for JMS to send and receive 500 events without the Activity Service, whereas the right column with the Activity Service. What attracts our attention is the sharp deviation in the ten test runs in both cases. For example, the time needed for sending and receiving 500 events via JMS interconnected with the Activity Service was between 3332 and 6300 msecs (i.e., the test results differed in almost two times).

TABLE I. SENDING AND RECEIVING 500 EVENTS: JMS ALONE VS. JMS INTERCONNECTED TO ACTIVITY SERVICE

JMS	JMS OM4
851	6300
836	3942
3956	3484
3895	4247
1525	3323
713	4522
3865	3360
3835	5247
4023	4168
887	4258
2439	**4285**

Figure 5 summarizes the test results for T3 and T4. What attracts our attention is that SQS alone was much slower than JMS alone – in fact, it was even slower than JMS interconnected with the Activity Service. For example, sending and receiving 100 events via SQS already took 13,412 msecs. With the Activity Service interconnected, that time was even longer (viz., 373,678 msecs). However, as one could expect, the time increased with an increase of the number of events but quickly, especially when SQS was interconnected with the Activity Service.

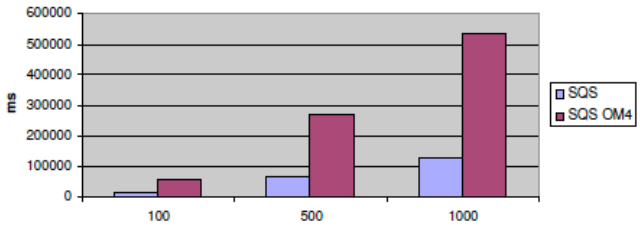

Figure 5. Sending and receiving 100, 500 and 1000 events: SQS alone vs. SQS interconnected to Activity Service.

Our tests showed that SQS alone was up to 36 times slower than JMS alone. One could expect that the same would keep true if the Activity Service were used. In fact, SQS interconnected with the Activity Service was up to 120 times slower than JMS interconnected with the Activity Service. Therefore, we suggest that the Activity Service greatly affected the performance, when SQS was used as the transport technology.

The left column in Table II shows the time needed for SQS to send and receive 500 events without the Activity Service, whereas the right column with the Activity Service.

 ISBN: 978-1-61208-337-7

Although the time was extremely long, it was almost constant for all the ten test runs (viz., between 262,867 and 270,603 msecs for sending and receiving 500 events) when SQS was interconnected with the Activity Service.

TABLE II. SENDING AND RECEIVING 500 EVENTS: SQS ALONE VS. SQS INTERCONNECTED TO ACTIVITY SERVICE

SQS	SQS OM4
65206	265602
65489	264818
66067	270338
64678	264498
67736	264092
65099	270603
64350	266591
65396	266645
65240	268499
64476	262867
65374	**266455**

While executing the tests, we noticed that the Activity Service demonstrated the worst performance when events were sent via an SQS queue and received via another SQS queue (T4). The performance improved when events were sent via a JMS topic but received via an SQS queue (T5). The performance became even better when events were sent via an SQS queue but received via a JMS topic (T6). Therefore, we suggest that sending events via an SQS queue does not take extra time but receiving events does. That is, the problem is that when the Activity Service deposits events to an SQS queue, the Event Consumer receives them with a big delay. Therefore, the performance problem might be resolved by optimizing the way the Activity Service works or with better implementation of the source code (which is written in Java).

IV. CONCLUSION AND FUTURE WORK

The performance of the Activity Service was evaluated. Our tests showed that sending and receiving events via JMS interconnected with the Activity Service took up to three times longer than without the Activity Service. However, that time was still short and increased slowly with an increase of the number of events. Therefore, we consider the performance to be very good, when JMS is used as the transport technology.

By contrast, the use of SQS could cause a performance bottleneck. Our tests showed that SQS itself was up to 36 times slower than JMS. (This was probably due to the distributed nature of an SQS queue). But with the Activity Service interconnected, the time for sending and receiving events increased up to 20 times more, resulting in almost 330,000 msecs delay.

Since OM4SPACE is relatively new, it will continue over time to improve its performance. In addition, OM4SPACE seeks to support more transport technologies, including Google App Engine and WS Notification. Therefore, in the future, we intend to execute more performance tests in order to obtain new test results.

ACKNOWLEDGMENT

Irina Astrova's work was supported by the Estonian Centre of Excellence in Computer Science (EXCS) funded mainly by the European Regional Development Fund (ERDF). Irina Astrova's work was also supported by the Estonian Ministry of Education and Research target-financed research theme no. 0140007s12.

REFERENCES

[1] M. Schaaf, A. Koschel, and S. G. Grivas, "Event processing in the cloud environment with well-defined semantics," The 1st International Conference on Cloud Computing and Services Science (CLOSER 2011), May 2011, pp. 176-179.

[2] A. Koschel, M. Schaaf, S. G. Grivas, and I. Astrova, "An ADBMS-style Activity Service for cloud environments," The 1st International Conference on Cloud Computing, GRIDs and Virtualization (CLOUD COMPUTING 2010) IARIA, Nov. 2010, pp. 80-85.

[3] R. Sauter, A. Stratz, S. G. Grivas, M. Schaaf, and A. Koschel, "Defining events as a foundation of an event notification middleware for the cloud ecosystem," The 15th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES 2011), Sep. 2011, LNCS, vol. 6882, pp. 275-284, doi:10.1007/978-3-642-23863-5_28.

[4] M. Schaaf, A. Koschel, and S. G. Grivas, "Towards a semantic definition for a cloud-based event notification service," The 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013), May 2013, pp. 345-349.

[5] I. Astrova, A. Koschel, L. Renners, T. Rossow, and M. Schaaf, "Integrating structured peer-to-peer networks into OM4SPACE project," The 27th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA 2013), Mar. 2013, pp. 1211-1216, doi:10.1109/WAINA.2013.88.

[6] A. Koschel, A. Hödicke, M. Schaaf, and S. G. Grivas, "Supporting smart grids with a cloud-enabled Activity Service," The 27th International Conference on Informatics for Environmental Protection (EnviroInfo 2013), Sep. 2013, pp. 205-213.

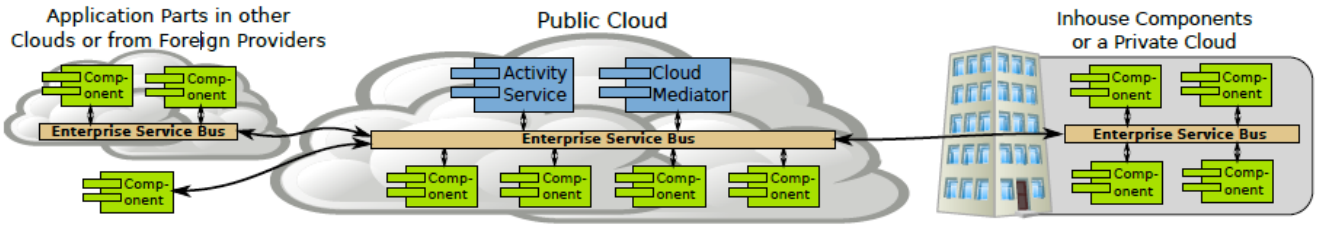

Figure 6. Distribution of OM4SPACE components [2].