Cloud-Enabled Scaling of Event Processing Applications

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

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

Ahto Kalja
Institute of Cybernetics
Tallinn University of Technology
Tallinn, Estonia
ahto@cs.ioc.ee

***Abstract*—Event processing is an important established concept for event-driven system development – with database triggers and event processing engines being typical examples of event processing technology. With nowadays movement into cloud computing, highly flexible scalability in cloud environments becomes an important challenge for event processing applications as they have many event sources and events to be processed there. As the core contribution of our work, we propose a novel approach to providing event processing applications with cloud-enabled scalability transparently to users (viz., the application developers) as part of an event-driven system itself.**

Keywords—Infrastructure-as-a-Service (IaaS) clouds; IaaS scalability; event processing applications; agents; event-driven systems.

I. INTRODUCTION

Traditional applications execute in a sequential way. But the real world is driven by events, which can come from several event sources. So how can these events be caught by traditional applications? One can create threads, which run in loops to catch the events and dispatch them to event consumers that can perform actions in response to the events. The biggest problem with this approach is that the applications can waste a lot of resources with otherwise not needed loops. Another big problem is an increased time between the raise of the events and their catch. Event processing applications provide a solution to these problems.

Event processing applications can be defined as sense-and-respond applications, i.e., the applications that can react to and process events. An event processing application can play the role of an event source, an event sink, or both. Event sources can handle off events to event sinks. It should be noted that an event source does not necessarily generate an event, nor an event sink is necessarily an event consumer. Furthermore, event sources and event sinks can be completely decoupled from each other: one can add and remove event sources and event sinks as needed without impacting other event sources and event sinks.

Event processing applications use the following concepts:

- **Event:** In an event processing application, every event is represented as an event object. This object holds all information about the event such as the timestamp when the event was caught, the event type, the event source, etc. After the catch of an event and transforming it to an application object, it is handed to an event stream.
- **Event stream:** An event stream is like a FIFO (First In, First Out) queue. Application objects in the stream are handled sequentially in the order of their arrival. The speciality of this type of queue is that an agent can subscribe to the stream and select which events it wants to receive.
- **Agent:** The drivers of an event processing application are one or more agents. They get the events from an event stream and react to or operate on those events. Examples of operations on events: selection, aggregation and composition. To structure agents and create a high cohesion with loose connections, an event processing network is used.
- **Event Processing Network (EPN):** An EPN models an event processing application as a set of interconnected application components whose execution is driven by events. Therefore, it is typically represented as a directed graph, where events are flowing through edges into nodes, which in their turn represent application components.
- **Event channel:** This is typically a messaging backbone, which transports the (formatted) events between event sources and event sinks. Because of the variety of event sources, not all events will be created in the format required for processing them by agents. In those cases, the events need to be formatted prior to being deposited them in an event channel.

Next we are presenting an example of event processing applications. This example is a door access log into a company, which uses a radio frequency identification (RFID) transponder to control the work time of its employees.

1. Employee A comes to work and activates the RFID transponder at the door with his chip, thus generating an access event.
2. The information on the chip is scanned and given to the adapter of an event processing application.
3. The application creates an event object and injects the data into it.
4. With a bundle of the subscriptions, the application knows which agents are interested on this event type (say Agents A and B) and put the event into the agent's event streams.
5. Agent A only reacts to the access event and logs the timestamp of the event and the information on the employee's chip to a database.

 ISBN: 978-1-61208-271-4

6. Agent B waits for another access event by the same employee in a time window of 10 hours.
7. Employee A activates the RFID transponder at the door with his chip again, when leaving work.
8. The application creates an another event object with the information on the employee's chip and passes it to the agents.
9. Agent A logs this event to the database.
10. Between the first and the second access events, Agent B produces a new event with the time which has passed between them.
11. Due to some other subscription, Agent B knows another agent, say Agent C, which is interested in the new event because it needs to gain the employee's work time out of it.

Step 6 shows how the agent uses a selection operation to get the information it needs. In this case, the agent also uses a technique, which is called windowing. It is possible to define a window by time (as it is in the example) or by the number of events in an event stream. Step 10 is an example of the composition of events. Here two events are merged into a new one. Once the new event has been composed, any agent in the application can use that event.

The remainder of this paper is organized as follows. The next section gives the motivation for our approach. This is followed by a description of our approach and a brief overview of the work related to the combination of event processing and cloud computing. The final section concludes the paper.

II. Motivation

Event processing applications are important because the real world is event-driven [12]. With great demand on high-speed and cost-efficient processing of events, event processing applications are calling for IaaS (Infrastructure-as-a-Service) scalability. IaaS scalability lets the applications make optimum utilization of resources such as CPU and RAM at different workload levels in order to avoid over-provisioning (i.e., having too many resources), under-utilization (i.e., not using resources adequately) and under-provisioning (i.e., having too few resources) [1]. In traditional environments, over-provisioning and under-utilization can hardly be avoided [2]. There is an observation that in many companies the average utilization of servers ranges from 5 to 20 percent, meaning that many resources are idle at no-peak times [3]. On the other hand, if the companies shrink their infrastructures to reduce over-provisioning and under-utilization, the risk of under-provisioning will increase. While the costs of over-provisioning and under-utilization can easily be calculated, the costs of under-provisioning are more difficult to calculate because under-provisioning can lead to a loss of users and zero revenues [3].

Since event processing applications experience variability in utilization of resources, they are calling for an infrastructure that can dynamically scale according to the application demand. IaaS scalability is one of the major advantages offered by IaaS clouds. This gives rise to the idea to deploy event processing applications into IaaS clouds. However, IaaS scalability is not just about having a scalable (virtual) infrastructure, but also about writing scalable applications. Valuable rules of thumb have been provided by Amazon.

Amazon provides a best practices guide [4] on how to write applications for the best fit for IaaS clouds. The most important guidelines are: an application should be divided into loosely coupled components that can be distributed across several servers and executed in parallel. Furthermore, the application should be as stateless as possible. If an application component fails or is temporarily not available, the application should continue to run. This can be achieved by developing the component as self-rebooting and using a message queue [5]. If the component is temporarily not available, messages will be stored in the queue and delivered later when the component comes alive again. These rules clearly indicate that IaaS scalability depends on the application design as well as the communication mechanism used to implement the application components. Therefore, IaaS scalability cannot be achieved by simply deploying applications into IaaS clouds. Rather, an IaaS cloud can guarantee an infrastructure equal to the application demand only when applications are designed properly or their design is amenable to appropriate scaling (horizontal or vertical).

However, event processing applications typically rely on a centralized event coordinator and could easily become a scalability bottleneck as a result of that [11]. Event processing applications are inherently stateful, which implies that services cannot be migrated or located anywhere, without affecting the application performance. Therefore, the deployment of event processing applications to IaaS clouds typically requires redesigning the applications for leveraging on-demand resource utilization. Therefore, the biggest problem is how to minimize the changes need to be done to the application design.

Another big problem is how to scale EPNs in IaaS clouds. An event-driven application can specify an EPN, which assembles the other components (e.g., event sources, event sinks and event streams) together. Virtual machines in IaaS clouds can scale horizontally by cloning a virtual machine or vertically by adding more resources to a virtual machine. Besides the scaling of virtual machines, the virtualization technologies inherent to IaaS clouds allow for the scaling of EPNs. Unfortunately, this very desirable feature is not supported by IaaS clouds yet, thus further complicating the deployment of event processing applications into IaaS clouds.

As an attempt to solve the problems above, in our previous work [6][10] we proposed to make event processing applications scalable through the integration of an event processing engine into a cloud architecture. In this paper, however, we propose a different approach.

III. Our Approach

IaaS scalability is important for event processing applications because these applications experience variability in resource utilization. Therefore, our approach is aimed at

 ISBN: 978-1-61208-271-4

providing event processing applications with IaaS scalability. IaaS scalability is service-oriented, meaning that scaling decisions are made on the basis of infrastructural metrics such as CPU and RAM utilization [1].

The basic idea behind of our approach was to bring IaaS scalability into an event–driven system itself. An event-driven system can generally be comprised of several event sources, event processing applications and event sinks. Event sinks have the responsibility of applying a reaction as soon as an event occurs. The reaction might or might not be completely provided by the sink itself. For example, the sink might just have the responsibility to filter, transform and forward the event to another component or it might provide a self-contained reaction on such an event.

Event sources, event processing applications and event sinks can be decoupled of each other; one can add or remove any of these components without causing changes to the others. However, an event-driven system could get quiet complex due to a large number of agents and event sinks to synthesize events out of aggregated data. Moreover, the agents are independent of each other – they can be distributed across several servers and executed in parallel. The problem is that it is very difficult for a scaling mechanism to decide which agents should use which rules to produce which output. Also how could this decision be made when the cloud should scale itself? Therefore, it was not an easy task to bring IaaS scalability into an event–driven system.

Figure 1 gives an overview of our approach, which includes the following components:

- **Load Balancing Agent (LBA):** Each EPN has its own LBA monitoring and interpreting (internal) technical events occurring in an Event-Processing-as-a-Service cloud and their data. LBAs ensure the performance and the availability of each EPN (or its agents), as they are the ones, which perceive the need to provision or decommission resources. Scaling decisions are made by LBAs on the basis of the current resource utilization and calculated by LBAs themselves. The resource utilization is aggregated out of technical events. For example, if the minimum or maximum threshold is crossed, scaling rules will be fired and a scaling mechanism will kick in.
- **Scaling Agent (ScA):** In addition to the LBA, each EPN has its own ScA, which can clone the EPN for horizontal scaling or restart it on a bigger virtual machine for vertical scaling.
- **Central Scaling Agent (CScA):** The CScA evaluates technical events against scaling rules. Scaling actions may include, e.g., the invocation of a service or the triggering of a scaling process. In addition, the CScA maintains the EPN topology.

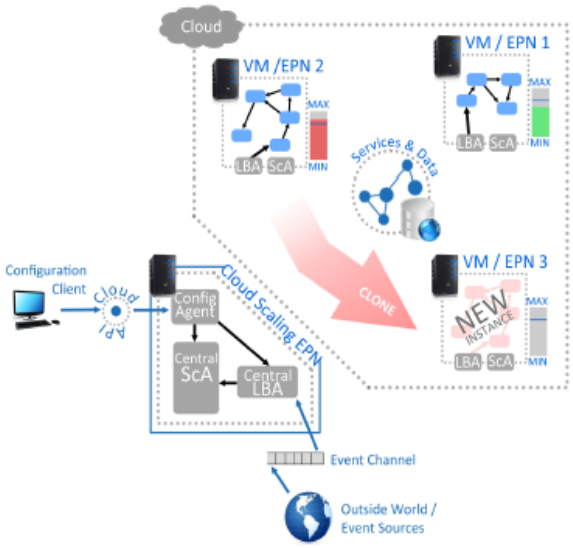

Figure 1. Cloud-enabled scaling of event processing applications

- **Central Load Balancing Agent (CLBA):** If the CScA defines how to scale, the CLBA defines what to scale. The CLBA takes the load of each EPN into account. Each LBA has to periodically send the information on the current resource utilization of its EPN to the CLBA. The CLBA then instructs the CScA to provision or decommission resources. This allows the CScA to foresee critical situations and to make scaling decisions beforehand. The CLBA is also responsible for all external events. An exposed interface (e.g., web services) make the interaction between the outside world and the cloud possible.
- **Configuration Agent (CA):** The CA allows for the configuration of the whole scaling mechanism (e.g., scaling rules and thresholds) and the EPN topology through the cloud API. The CA could be implemented as an agent fitting into the idea of a Dynamic Control Plane [7], which gives users (viz., the application developers) the possibility to configure the cloud through an easy-to-use administrative interface.
- **Cloud-Scaling EPN:** The CLBA, the CScA and the CA are "networked" together to form an EPN for the scaling of an Event-Processing-as-a-Service cloud. This cloud hosts services to be leveraged by event processing applications as needed. As a result, the cloud can scale up and down according to the application demand.

Our architecture can be used by the following event processing applications:

- Disaster management, where the input data need to be gathered from various heterogeneous distributed sources (e.g., scientific sensors) and processed using the event processing technology to react on disasters.

 ISBN: 978-1-61208-271-4

- Online business development, where the clicks of website visitors need to be processed as events to identify the interest to the website.

IV. Related Work

Technical events occurring in an IaaS cloud are related to resource utilization. Event processing engines can help in monitoring and high-speed processing of these events. Therefore, recently it was proposed to integrate an event processing engine into an elastic controller in order to enhance IaaS scalability [8][9].

An IaaS cloud requires that applications are designed especially for the cloud. The scaling of traditional applications is typically easy. The question is how to scale event processing applications. These applications follow their own design rules and thus, they have to be tailored to the cloud. Therefore, in our previous work [6][10] we proposed to integrate an event processing engine into a cloud architecture itself, providing scaling decisions out of scaling rules through the cloud API.

However, in this paper we decided to move from a different direction – we tried to adapt IaaS scalability to an event-driven system.

V. Conclusion and Future Work

Event processing applications need to handle a lot of information. Thus, the ability to process this information quickly is important for those applications. But processing the information quickly implies processing it efficiently, which in turn implies spending less money on an infrastructure. And this is the point where event processing applications could benefit from the deployment into IaaS clouds whose scalability enables efficient and cost-saving event processing. However, a cloud architecture that allows event processing applications to benefit from IaaS scalability is currently missing [6][10]. Therefore, with our approach and its components described below, we aim to fill this gap.

Each EPN will have a Load Balancing Agent (LBA), which periodically sends the load of its EPN to the Central Load Balancing Agent (CLBA). If the minimum or maximum thresholds specified by users through the Configuration Agent (CA) are crossed, the CLBA will instruct the Central Scaling Agent (CScA) to provision or decommission resources. In addition to the LBA, each EPN will have a Scaling Agent (ScA) acting on behalf of the CScA. The CScA will translate the CLBA's instructions into an appropriate scaling action taken by the ScA to adjust the load of its EPN. It should be noted that users will be kept totally unaware of these scaling actions and delivered with the illusion of a scalable infrastructure, the infrastructure that can scale horizontally (by cloning an EPN) or vertically (by restarting an EPN on a bigger virtual machine).

Our approach is geared to make event processing applications scalable, while minimizing changes to be done to the application design and allowing for the scaling of EPNs as if they were virtual machines.

In the future, we are going to implement our approach and evaluate its performance.

Acknowledgment

Irina Astrova's and Ahto Kalja'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 and Ahto Kalja's work was also supported by the Estonian Ministry of Education and Research target-financed research theme no. 0140007s12.

References

[1] J. Cáceres, L. Vaquero, L. Rodero-Merino, Á. Polo, and J. Hierro. Service scalability over the cloud, Handbook of Cloud Computing, eds. B. Furht and A. Escalante, Springer Verlag, Berlin, Heidelberg, 2010

[2] C. Braun, M. Kunze, J. Nimis, and S. Tai. Cloud Computing, Web-based dynamic IT-Services. Springer Verlag, Berlin, Heidelberg, 2010

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. Communications of the ACM, 53(4), 2010, pp. 50–58

[4] J. Varia. Architecting for the cloud: best practices. last accessed: January 2013, http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf

[5] P. Marshall, K. Keahey, and T. Freeman. Elastic site: Using clouds to elastically extend site resources, Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, IEEE, 2010, pp. 43–52

[6] I. Astrova, A. Koschel, and M. Schaaf. Automatic scaling of complex-event processing applications in Eucalyptus. Proceedings of the 15th IEEE International Conference on Computational Science and Engineering (CSE), IEEE, 2012, pp. 22–29

[7] L. MacVittie, A. Murphy, P. Silva, and K. Salchow. Herscheruber die wolke: Anforderungen an cloud-computing. Technical report, 2010

[8] H. Lim, S. Babu, and J. Chase. Automated control for elastic storage, Duke University, 2010

[9] L. Vaquero, L. Rodero-Merino, and R. Buyya. Dynamically scaling applications in the cloud. ACM SIGCOMM Computer Communication Review, 41, 2011, pp. 45–52

[10] A. Koschel, I. Astrova, M. Schaaf, S. Gatziu Grivas, S. Priebe, J. Raczek, J. Reehuis, and K. Scherer. Integrating complex event processing into Eucalyptus, Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2011

[11] N. Shalom. Interview with Michael Di Stefano from Integrasoft on their complex event processing cloud services using Esper GigaSpaces. last accessed: January 2013, http://blog.gigaspaces.com/interview-with-michael-di-stefano-from-integrasoft-on-their-cep-cloud-services-using-esper-gigaspaces/

[12] M. Schaaf, A. Koschel, S. Gatziu Grivas, and I. Astrova. An active DBMS style activity service for cloud environments. Proceedings of the 1st International Conference on Cloud Computing, GRIDs, and Virtualization, IARIA, 2010, 80–85

 ISBN: 978-1-61208-271-4