Distributed Complex Event Processing in Multiclouds

Vassilis Stefanidis[1], Yiannis Verginadis[1], Ioannis Patiniotakis[1] and Gregoris Mentzas[1]

[1] Institute of Communications and Computer Systems, National Technical University of Athens, Greece
{stefanidis, jverg, ipatini, gmentzas}@mail.ntua.gr

Abstract. The last few years, the generation of vast amounts of heterogeneous data with different velocity and veracity and the requirement to process them, has significantly challenged the computational capacity and efficiency of the modern infrastructural resources. The propagation of Big Data among different processing and storage architectures, has amplified the need for adequate and cost-efficient infrastructures to host them. An overabundance of cloud service offerings is currently available and is being rapidly adopted by small and medium enterprises based on its many benefits to traditional computing models. However, at the same time the Big Data computing requirements pose new research challenges that question the adoption of single cloud provider resources. Nowadays, we discuss the emerging data-intensive applications that necessitate the wide adoption of multicloud deployment models, in order to use all the advantages of cloud computing. A key tool for managing such multicloud applications and guarantying their quality of service, even in extreme scenarios of workload fluctuations, are adequate distributed monitoring mechanisms. In this work, we discuss a distributed complex event processing architecture that follows automatically the big data application deployment in order to efficiently monitor its health status and detect reconfiguration opportunities. This proposal is examined against an illustrative scenario and is preliminary evaluated for revealing its performance results.

Keywords: Distributed CEP, Cloud Monitoring, Multiclouds, Big Data.

1 Introduction

Nowadays, we witness a constant increase of the connected devices and services that continuously produce data and transmit health status events. The generation of vast amounts of heterogeneous data and their propagation among different processing and storage architectures, has amplified the need for adequate and cost-efficient infrastructures to host them. The recent uptake of Cloud computing adoption could be considered as a remedy to this situation where the growing needs of the so-called Big-Data applications, are met by the vastly improving offerings of the cloud providers [1]. Such applications require to efficiently deal with the volume, variety, velocity, and veracity of the data, using any resources available in a cost-effective and efficient way. In recent

years, Big-Data applications have been developed and used successfully over cloud infrastructures [2]. Such an adoption of the Cloud computing, theoretically satisfies unlimited hosting requirements, for storing and processing Big Data, in a reliable, fault-tolerant and scalable way. Cloud's ability comprises resources virtualization that requires minimum interaction with cloud service providers and enables users to access terabytes of storage, high processing power and high availability in a pay-as-you-go model [3]. As more enterprises started to trust the Cloud computing paradigm, they started outsourcing their workload to infrastructures, offered by single cloud providers. This led to vendor lock-in situations that didn't allow the use of the most optimal infrastructure (with respect to the processing location, efficiency, cost etc.) per each case and at each given time. Thus, the recent availability of a variety of public IaaS providers (e.g. Amazon, Google, HP, IBM, RackSpace) as well as private offerings (e.g. Openstack, VMware), still remains un-exploitable by the average cloud user [4;5;6;7;8;9]. In the majority of cloud applications, the cloud users seek to commit their entire processing stack over a single cloud provider, by considering only the planned or expected behaviour of the application. As a result, they neglect to consider flexible infrastructures able to mix and match platforms from multiple cloud providers, and meet, in such way, the dynamically changing requirements of their Big Data applications [10]. In other words, these users miss exploiting the benefits of the multicloud architectures.

Nevertheless, the use of multicloud offerings especially at the level of infrastructure in order to cope with the needs of big data applications still involves several challenges and open research questions. These mainly correspond to how data-intensive computing can be hosted in highly distributed and federated cloud environments by keeping the Quality of Service (QoS) guarantees. This generic challenge can be analysed into a number of fine-grained challenges that refer to scalability, resiliency, and security issues faced by big data and data-intensive applications on distributed platforms. The purpose of this paper is to address the applications' scalability and resiliency challenges when attempting to exploit the benefits of multiclouds. Such benefits become even more obvious, if the reader considers the new business and scientific needs for distributed data processing among various locations, for real time processing of data from various heterogeneous sources. Nevertheless, multicloud computing environments consist of many resources that are simultaneously accessed by several tenants, which means that often suffer from unanticipated behaviour such as performance degradations either for the infrastructure or the software parts, component failures and security threats. For these reasons, it is very important to constantly monitor and analyse multiclouds to detect situations that should lead to reconfigurations of the used processing topology (e.g. scale up or scale down according to the current workload). In a way, monitoring delivers the knowledge that is required to make appropriate decisions with respect to the way applications are deployed and hosted over multiclouds, thus answers the challenges mentioned above.

Monitoring and analysing Clouds correspond to challenging tasks that require sophisticated tools and methods. Event processing is a method of tracking and analysing streams of data about application-related occurrences that happen (i.e. events), and issuing some alerts based on them. Complex event processing (CEP), corresponds to event processing that combines data for inferring patterns of events that may suggest

more complicated circumstances [11]. CEP systems [12] are valuable in digesting and processing a multitude of event streams. Their big advantage is the ability to collect information from various heterogeneous data sources and filter, aggregate or combine them over defined periods of time (i.e. time windows). The idea of using CEP for monitoring cloud applications has been applied with respect to two types of architectural approaches: centralized and distributed. The centralized CEP architecture is based on a single CEP engine which processes all monitored data and detects patterns by using rules. On the other hand, the distributed CEP architecture consists of a set of cooperating CEP engines that exchange messages and are able to more efficiently detect event patterns by considering rules that differ according to the proximity of the processing engine to the event source. In existing centralised CEP approaches [13], [14], huge bandwidth and computational capabilities are required and usually they lack robustness and scalability because of the single point of failure when processing vast amounts of health status data. On the other hand, the distributed CEP architectures such as the parallel CEP processing architecture of Hirzel's [15], the work of Ku et al. [16] and that of Mdhaffar et al. [17], present better performance in terms of data processing throughput, due to workload sharing across multiple CEP engine, and establish better scalability results without any risk of single point of failure. Nevertheless, all these cases are bound to the use of a single cloud vendor, a fact that limits, by default, the big data-intensive applications capacities.

In this paper, a distributed CEP system is presented, appropriate for applications deployed over multicloud resources, proposing a multi-level event processing architecture. The paper is organized as follows: Section 2 describes the related work, while in section 3, the proposed conceptual architecture is discussed along with the relevant technological grounding details. In section 4, an illustrative example is presented for better demonstrating the benefits of the proposed approach. In section 5, a preliminary evaluation of the proposed system is given and in Section 6, we conclude by discussing the next steps of this work.

2 Related Work

CEP plays a very important role in detecting and integrating events by pattern matching and using rules in issuing alerts, in cases where increasing amounts of data streams are present. Concerning the distribution of CEP, few recent works focus on the technique of parallelization of pattern-matching processing in which pattern matching is seen as a stateful operator in a general purpose streaming system.

More specifically, Hirzel [15] by using the keys to partition the incoming events, proposes a pattern matching syntax and a way of translation based on the concept of partitions. Therefore, events with different keys can be processed in parallel. Hirzel exploits the partitioning constructs provided by the queries of the specific language he uses. However, this approach is sufficient only when queries of a language contain such constructs which is not always the case. The parallelism technique that the authors use, is either centralized in one machine or distributed in various machines from a single cloud vendor. Their distributed architecture approach present some issues with respect

to a vulnerability of parallelization overhead and noise. Moreover, the distributed parallelism CEP approach misses the multicloud benefits.

In a similar work, Ku et al [16] propose a distributed CEP architecture which splits various centralized Complex Event Processing tasks load across multiple stations. The core of communication architecture is achieved by using a distributed message broker based on Apache River, a network of distributed systems in the form of modular cooperating services. The authors use a distributed complex event detection algorithm with a Masters/Workers pattern. The innovation of this proposal can be found in geographical distribution of tasks of sub-detection using CEP engines in a Master/Slave flowchart concept. However, this technique presents an important communication overhead when the number of events to be processed is lesser than 500. Another issue is the focus on one cloud vendor without considering cross-cloud level deployments. Moreover, the case of scaling in or scaling out the cluster of deployed CEP engines is not supported and a static architecture of a pre-considered number of deployed CEP engines is used.

Paraiso et al [18] present a distributed CEP engine (DiCEPE) which is a platform that focuses on the integration of CEP engines in critical-case distributed systems. Appropriate communication protocols are used in order to integrate CEP engines easily and interconnect them across vast geographical areas. Despite the communication heterogeneity, adaptability and scalability of the proposed architecture, the introduction of Frascati [32] open source platform layer, induces overhead due to many messages exchanged. Moreover, the deployment of the DiCEPE on multiple cloud environments using various levels (i.e. 3-level : VM level, Cloud level, Cross-Cloud level) is still a challenge and is not solved in this effort.

Schultz-Moller et al [19] propose a distributed approach for event detection, called Next CEP, which uses a new high-level event query language for expressing event patterns. The approach involves the use of the same rules in all CEP engines used in the architecture without the option to provide dynamic rule adaptation. In our proposed architecture, a domain specific rule (for each CEP instance) can be changed and deployed at runtime as well as adapted according the monitoring needs.

Mdhaffar et al [17] introduces a dynamic architecture for measuring cloud performance and analysing various situations based on a complex event processing either in a centralized or a distributed architecture. The specific paper demonstrates a system that is designed to dynamically switch between different centralized and distributed CEP engines, depending on the current machine load and network traffic conditions. However, in this dynamic CEP system, no event processing takes place locally to each Virtual Machine concerning aggregation functions or processing query rules. The local processing is limited to outliers and anomaly detection. In our approach, local aggregation function operations take place and the aggregated data results are published to higher levels of CEP processing nodes. Moreover our approach does not face any latency or delay set up issues, originated from the need to switch between various architectures.

One additional approach for a centralized Complex Event Architecture which combines technologies such as Mule ESB and ESPER engine was presented by Boubeta et al [13], by using the Xively IoT platform. Data is gathered from various sources for

Home Automation operations. Nevertheless, this approach is centralized, this neglecting the benefits of multicloud environments and does not use any dynamic communication protocol suitable for event-driven architectures that dynamically may change their number of hosts. Leitner et al [14] propose an event-based approach for monitoring cloud applications by using a multi-step CEP-based event correlation schema which can be used for cloud applications with a large number of virtual resources. By that way, the application elasticity is increased. CloudScale [33] framework uses the monitoring data to dynamically acquire and release cloud hosts. Nevertheless, the specific approach does not investigate the co-existence of low-level metrics such as CPU utilization, memory consumption etc. along with the application specific metrics i.e. responsiveness of a very critical application that controls i.e. a nuclear factory. Moreover, there is a small evaluation of the specific architecture while the case of deploying applications in various cloud environments is not considered at all.

Finally, Flouris et al [20] present FERRARI which is a prototype that implements real-time CEP for large volume event data streams over distributed architectures by sharing the load over a set of streaming cloud platforms. In addition, intra-cloud CEP is used where appropriate. Nevertheless, this approach does not give any dynamic publish/subscribe model for communicating among various CEP engines but it uses push/pull techniques. Moreover, it does use only one CEP engine in each Cloud environment with partitioning (parallelism) in contrast to our architecture that uses many CEP engines in each Cloud operator. In addition, an Event Processing Network (EPN) is automatically deployed and configured with the assistance of a dedicated Event Processing Management Component. In the case of Flourish et al. work [20] a web-based authoring tool is used for manually building the EPN and performing the query optimization across the Cloud environments.

3 Distributed Complex Event Processing (DCEP) Architecture

3.1 Conceptual Architecture

In this section, we discuss our approach for a novel Event Processing Network (EPN) that can be efficiently distributed over several virtualized resources that may span multiple providers to monitor the deployment of multicloud applications. Such advanced monitoring capabilities are valuable for detecting reconfiguration opportunities that will safeguard the desired quality of service of the multicloud applications. An EPN is a conceptual model that refers to a set of Event Processing Agents (EPA) , Event Producers and Event Consumers all connected by a set of Event Channels(EC) [21]. The event producers are resources that generate events while the event consumers are components that receive such events. In multiclouds, the event producers involve VMs that host parts of a multicloud application and transmit monitoring events with respect to the health status of the hosting resource and any application specific information. The EPAs act both as event consumers (subscribe for monitoring events) and as event producers since they are able to relay any detected complex event patterns to other parts of the EPN. Each EPA filters, match and derivate complex events according to specific

rules, expressing patterns that reveal the multicloud application health status. In our approach, we consider the implementation of these EPAs by using interconnected Complex Event Processing (CEP) engines. The goal of CEP technique is to identify events and patterns with great importance such as opportunities or threats for the current multiclouds processing topology and respond to them as quickly as possible. The use of multiple EPAs in a distributed architecture brings about the advantage of multi-level complex event processing. Specifically, three distinct layers of CEP are considered (as seen in Fig. 1) for hierarchically detecting interesting complex events (e.g. average CPU>80% for an application server instance, average CPU>80% for all application server instances on Cloud X and average CPU>80% for all application server instances on all Clouds used for a certain multicloud application). Each of these EPAs are integrated with an appropriate pub/sub system for message queueing and event propagation across the three event processing layers, constituting the DCEP agents.

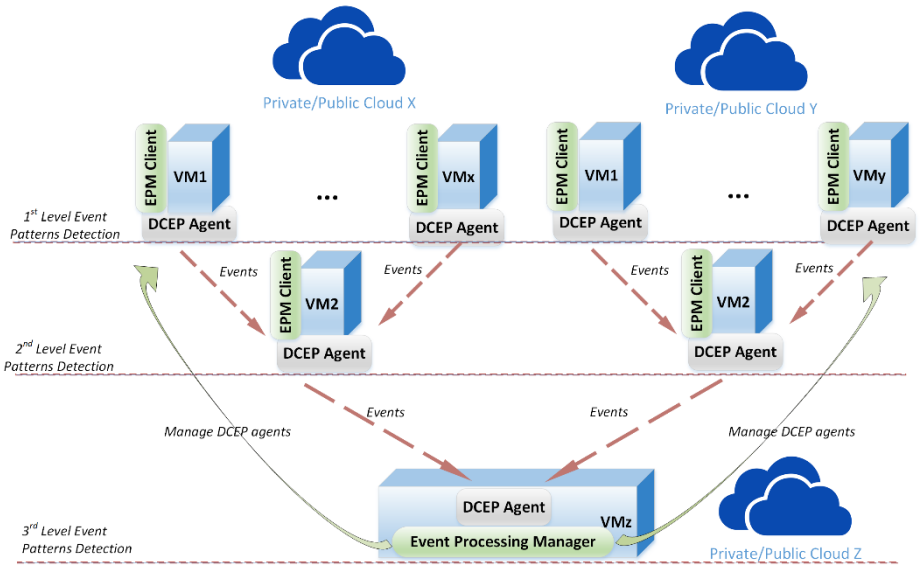

Fig. 1. Conceptual Architecture of a DCEP for Monitoring Multicloud Applications

As depicted in Fig.1, this network of DCEP agents is structured across three main layers: i) the VM instance layer (1st Level Event Patterns Detection), ii) the Cloud layer (2nd Level Event Patterns Detection) and iii) the Global layer (3rd Level Event Patterns Detection). The first one corresponds to the installation and configuration of Event Processing Agents on each VM instance in order to focus on the aggregation, filtering and propagation of raw health status events. The second layer involves the use of one such agent per Cloud for extracting higher-level information on the placed data and cloud application. This allows for a valuable consolidated view of all the resources' and applications' statuses deployed per cloud provider based on the aggregation and processing of the output of "local" DCEP Agents that report from each VM. Consequently, the third layer involves the output aggregation of the "second level" Agents in order to allow for a global overview of the status of the whole processing topology.

Moreover, in a dynamic environment where the multicloud resources to be used for hosting an application are not static and predefined, a dedicated mechanism for setting up and maintaining the described EPN according to the requirements of the DevOps or the application developers, is a necessity. Thus, we introduce the so-called Event Processing Management (EPM) server and its EPM Clients which are responsible for responsible for the deployment, synchronization and orchestration of the DCEP Agents, hosted in various VMs and heterogeneous Cloud providers. Upon successful deployment of these agents, the Event Processing Management also undertakes the configuration or enhancement of all the appropriate complex event pattern rules that should be used by each EPA.

3.2 Deploying and Managing DCEP Agents over Multiclouds

The EPM subsystem is responsible for deploying and managing the monitoring network of Event Processing Agents, and it uses a client-server architecture which comprises two distinct architectural components types:

i. The EPM clients, which are the DCEP controlling agents. They accompany the DCEP agents at each VM, on the first, second or third level of event pattern detection. They do not undertake monitoring tasks (in contrast to EPAs) but they are separate modules. These Clients contain configuration scripts for setting up and launching first level or second level DCEPs according to the instructions of the server (i.e. Event Processing Manager). They also contain information and credentials for connecting to server. EPM Clients are installed in a VM during VM initialization. An alternative approach would be that the server connects to each VM and install clients after VM initialization. This approach requires that each VM offer an interactive SSH shell and the VM network address and administrator credentials are available to server.

ii. The server (i.e. Event Processing Manager) is the controller of clients. It is a part of the EPM subsystem and resides at the third level of the DCEP architecture (see Fig.1). The server is responsible for installing clients to VMs (if they are not installed during VM initialization) and afterwards for instructing them to configure the respective DCEP Agents as first or second level event patterns detection. Moreover the server periodically checks whether clients and VMs are active and if one goes offline (e.g. if it crashes) it can reconfigure the EPN appropriately.

It is noteworthy to mention that before the multicloud application should start its operation the EPN must be in place and ready to capture and process monitoring events. Therefore, upon each VM boot, the installed EPM client attempts to connect to EPM server using SSH protocol. If it succeeds it sends VM identification information and the server assigns it a unique Id (which is stored for future sessions). The server will decide (using a specific strategy) which VMs will act as first level EPAs and which as second (or first and second level simultaneously). Subsequently the server signals clients about its decision, passing any needed information, and clients execute preconfigured setup scripts that prepare and launch the DCEP Agents. The EPM Client monitors the DCEP Agent launching and when it is ready the client updates the server. When the

EPN is in place and operational, the server signals that the multicloud application may be deployed and start operating.

It is also important to mention that second level DCEP Agents are configured and launched before the first level DCEP Agents, since their network information must be passed to the subordinate first level DCEP Agents. First level DCEP Agents are configured and launched afterwards and forward their events to the designated second level DCEP Agent, resulting in a hierarchical network structure. Second level DCEP Agents are also configured to forward their events to the third level where EPM resides. The process of bootstrapping the EPN is depicted in Fig.2.

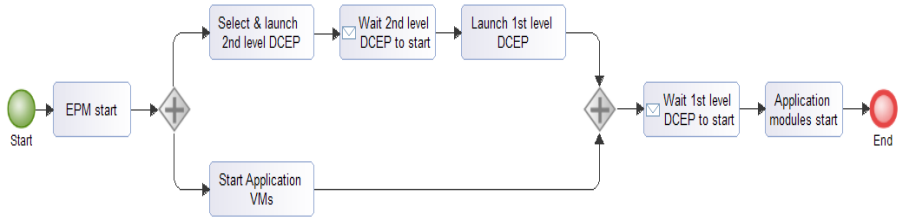

Fig. 2. Multi-Cloud DCEP Application Bootstrap

3.3 DCEP Implementation

In this section, we ground the conceptual architecture presented in section 3.1 and discuss the technologies used for each of the components of the proposed DCEP system. Based on the presented architecture, two basic functionalities should be supported. The first is related to the queueing and propagation (to subscribers) of monitoring events coming from multicloud resources, while the second corresponds to the complex event processing of these events. For the first functionality, the use of an Enterprise Service Bus (ESB) is in order while for the second basic functionality several instances of CEP engines have been adopted. The event producers perceived in this approach, refer to: i) virtualized resources-related sensors that capture information related to infrastructural performance issues (e.g. VMs' RAM usage, CPU load etc.); application-related sensors which propagate the multicloud application's performance (e.g. Response Time) and iii) the complex events produced by EPAs based on the previous two types of events. The data obtained from these event producers are published through the use of an ESB. An ESB instance is used in each VM employed for hosting the components of the multicloud application (e.g. DBs, application servers etc.). In parallel, a CEP engine instance is also installed per each VM used. This CEP engine uses event patterns (rules) that specify the conditions under which reconfiguration events are produced or specific aggregated events are propagated to a higher processing level.

In this work we have used and configured the MuleSoft open source software [22] for the implementation of ESB functionalities. MuleSoft ESB was evaluated by the Rademakers and Dirksen [23] as the best-of-breed products currently available according to the following criteria: ESB core functionality, quality of documentation, market

visibility, active development and support community, custom logic, transport protocols and connectivity options, integration capabilities with open source frameworks, and tool support. Due to the need for dynamic and adaptive deployments of various Virtual Machines among various Cloud environments, it is obvious that a flexible type of messaging protocol should be used to transfer the raw data coming from the data sources (hardware and software sensors) to the EPAs. Therefore, we have adopted the Advanced Message Queueing Protocol (AMQP) protocol to propagate monitoring events over the MulSoft ESB, according to the Publish/Subscribe paradigm. Apache ActiveMQ [27] is one of the most popular and powerful open source messaging and Integration Patterns server. It is an open source message broker written in Java together with a full Java Message Service (JMS) client. Many features that it provides, fits our Cross-Cloud distributed CEP Architecture:

- ActiveMQ is standards-based in that it is a JMS 1.1 compliant. The JMS specification provides many benefits and guarantees including asynchronous message delivery, message durability for subscribers which are very crucial for the dynamic cross-cloud scaling architecture;
- ActiveMQ provides a wide range of connectivity options including support for protocols such as HTTP/S, multicast, TCP, SSL, and others. This gives a substantial flexibility for the implementation of communication among publishers and subscribers;
- Due to the proposed distributed architecture, the use of tightly coupled architectures for message brokering can be problematic. Loosely coupled architectures exhibit fewer dependencies which are very useful in a dynamically changing (by scaling in or out) event-driven architecture [27].

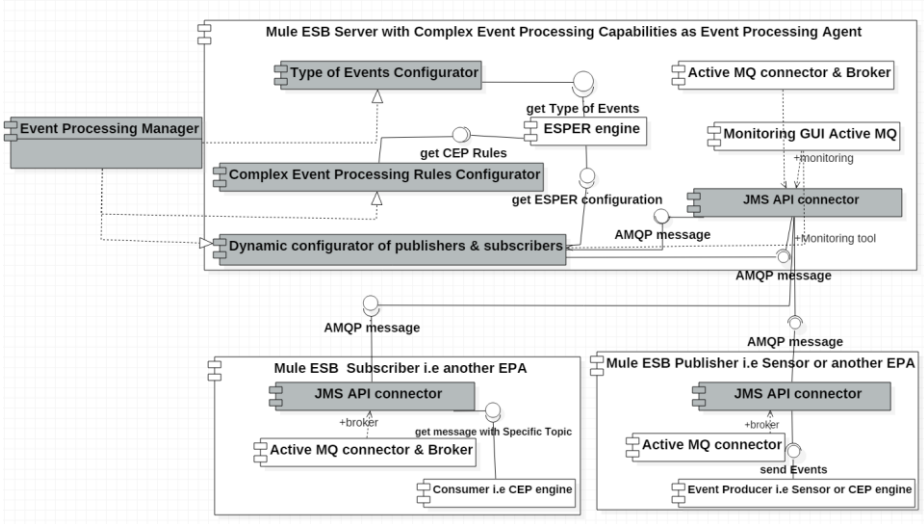

Fig. 3. Communication Details between ESB and EPAs

Moreover, ESPER [26] was used for the CEP capabilities required for this approach. ESPER is an open source engine that combines Event Stream Processing (ESP) and

CEP capabilities. ESPER uses the Event Processing Language (EPL) and provides a highly scalable, memory-efficient data stream processing tool to detect event patterns and create alerts. EPL is used to express filtering, aggregation joins, and define patterns over multiple events streams. In Figure 3, we provide a UML component diagram that conceptually depicts the detailed subcomponents of the proposed architecture per each layer and EPA. We note that grey color was used to denote newly developed subcomponents that augment the MuleSoft and Esper subcomponents that are offered as open-source software. The subcomponents include the:

- *Type of Events Configurator* - This is a subcomponent that provides to the ESPER engine the information about the type of events that the engine should process;
- *Complex Event Processing Rules Configurator* - This subcomponent injects to the ESPER engine the appropriate event patterns expressed in EPL for detecting complex event patterns at run-time.
- *ESPER Engine* - This corresponds to the core ESPER component re-used and spawned in multiple instances over the proposed distributed architecture for detecting complex event patterns;
- *Dynamic Configurator of publishers & subscribers* - This subcomponent can register any consumer to needs to subscribe to events according to a specific event topic that is defined via the JMS API of Active MQ service;
- *JMS API Connector* - This subcomponent is used as a software entity that propagates events to other subscribers hosted in several VMs;
- *Monitoring GUI Active MQ* - This subcomponent is used as a monitoring tool where valuable information is presented through a user interface concerning the way that various events are forwarded according to pub/sub model through the Active MQ broker;
- *Active MQ Connector & Broker* - This subcomponent is an open source message broker written in JAVA providing an efficient Java Message Service;
- *Event Processing Manager* - This subcomponent, as described in the previous section, is responsible for the synchronization and orchestration of the deployment and reconfiguration of EPAs and ESB instances to be hosted in all the VMs that will accommodate aspects of a big data-intensive application.

4 An illustrative example

To illustrate the details of this approach, we use the case of a Vehicular Simulation Traffic processing application, which is a big data application due to its dynamic and demanding nature, requires deployment over a multi-cloud environment, thus continuous and efficient monitoring for optimisation purposes. In this example, the traffic system includes many heterogeneous agents (e.g., people, cars, public transport, and traffic signals) and depends on several factors (e.g., weather, mass events, road works etc.). Therefore, it involves big data-intensive scenarios, where the capability to detect multiple complex events, is a necessity to recognise and react on situations that may jeopardise the health of the deployment topology and eventually the quality of service of

the target application. This involves the real-time analysis of huge amount of data coming from various sources that represented application fragments that undertake data intensive traffic simulations. To be more specific, our proposed framework can be used to run simulations with different input settings (e.g. traffic control settings) and produce output, such as congestion, travel times, average speeds and total waiting times. So, it may be used to evaluate a large number of traffic control settings, e.g. traffic signal settings or make decisions on the construction of new roads, bridges etc.

In this illustrative example, the complex event patterns deployed, use raw monitoring events coming from the traffic simulating sensors (i.e. RawExecutionTime, SimulationLeftNumber, RemainingSimulationTimeMetric and TotalCores events). The complex events patterns required for monitoring the application and making reconfiguration decision based on its current status, involve complex function such as the *percentile* and the *floor* functions which use specific time windows and specific output rates. Specifically, the following complex event processing rules have been used:

- MinimumCores event expressed as: *Ceil(SimulationLeftNumber /floor(RemainingSimulationTimeMetric / ETPercentile))*
- SimulationNotFinishOnTime scaling event expressed as: *(ceil(SimulationLeftNumber /TotalCores) * ETPercentile) - RemainingSimulatioTimeMetric*

5 Evaluation

The testbed used for this Vehicle Traffic scenario, include 2 VMs in a private cloud infrastructure (using Openstack) and a windows 10 PC. Each of the VMs has 64-bit CPUs with 20 GB of disk and 3.8 GB of RAM and come with Ubuntu 16 as the operating system. The following components were used for this evaluation: 1) an events generator that is able to publish events in a configurable rate; 2) the DCEP Agent and 3) the monitoring evaluation tool. The event generator is placed on the VM hosted in windows machine and corresponds to a specific Mule Application developed based on the Quartz[1] module which supports the scheduling of programmatic events.

For this evaluation scenario we measured the RAM and CPU usage on the machines that hosted the DCEP agents for a number of incoming event rates. Specifically, we used 500 events/sec and 1000 events/sec over a period of 5 minutes, corresponding to the RawExecutionTime events of the Vehicle Simulation Traffic scenario. The other events i.e. SimulationLeftNumber, RemainingSimulationTimeMetric and TotalCores that come from Simulation Manager (the big data- intensive application) do have smaller rates i.e. 100 events/sec etc. The results for the CPU and RAM usage are depicted in Figure 4.

Based on these results, we notice that our DCEP approach presents a stable memory consumption of around 30 percent, without any major fluctuations, while the CPU usage doesn't exceed the 36 percent. The latter seem even more improved as the number

[1] https://docs.mulesoft.com/mule-user-guide/v/3.6/quartz-connector

of events per second increase, a fact that denotes good queuing capabilities. An important advantage of this implementation of the proposed DCEP system is the appropriate use of complex event processing technologies that allow for the transparent joins of two or more streams and the use of complex mathematical formulas that may result in highly complex event patterns rules. In the Traffic Vehicle scenario several events of various topics were considered and joined in order to detect and emit a final scaling event called SimulationNotFinishOnTime. This was possible through the utilization of user defined functions on the definition of patterns and calculation of complex formulas (e.g. percentile function).

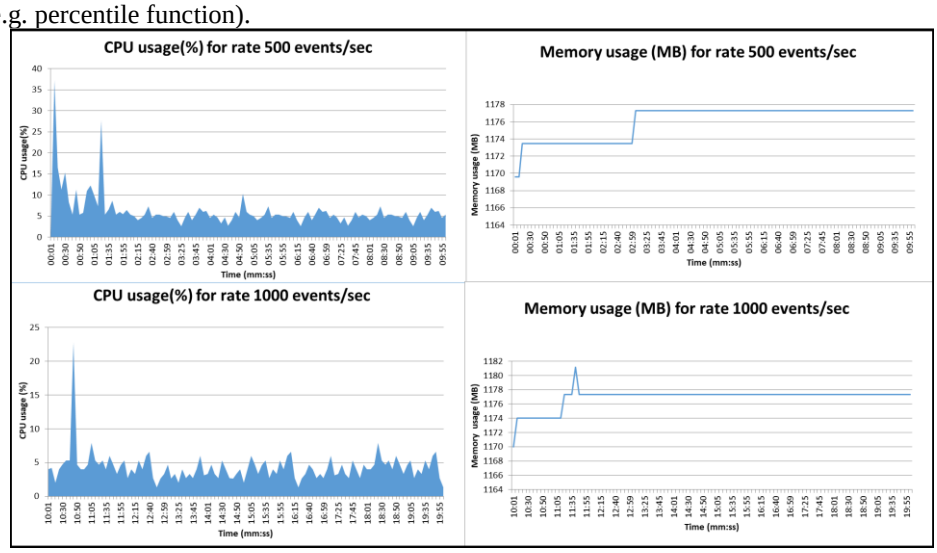

Fig. 4. Evaluation Results

6 Conclusions

In this paper, we presented a three level distributed architecture for monitoring big data intensive applications deployed over multicloud resources. The conceptual architectural design was discussed along with the implementation details and technological decisions. Furthermore, we used an illustrative scenario in order to present and evaluate the main benefits of this approach. The preliminary evaluation revealed adequate processing and memory consumptions levels that will be further compared, in the future, against other prominent complex event processing solutions. In addition, this work will continue in terms of integrating such an approach with a holistic platform that will be able to manage the complete lifecycle of multicloud applications and their processed data. The role of this solution will be to adequately monitor the health status of the application, in order to maintain a constantly optimised deployment of all the application components, over heterogeneous VMs that span the boundaries of several cloud providers.

Acknowledgements

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 731664. The authors would like to thank the partners of the MELODIC project (http://www.melodic.cloud/) for their valuable advices and comments.

References

1. Zanoon, N., Al-Haj, A., Khwaldeh, S.: Cloud Computing and Big Data is there a Relation between the Two: A Study. International Journal of Applied Engineering Research 12(17) , pp.6970-6982 (2017).
2. Hashema, I.,Yaqoob,I.,Anuar,N.,Gani, A., Khan,S.: The rise of "big data" on cloud computing: Review and open research issues. Journal of Information Systems 47 , pp. 98-115(2015).
3. Martinez, G., Bote, M., Gómez-Sánchez, E., Cano-Parra, R.: Cloud computing and education. Journal Computers & Education 80 (C), pp.132-151 (2015).
4. Amazon Web Services Homepage, https://aws.amazon.com/
5. Hewllet Packard Homepage, https://www.hpe.com/emea_europe/en/solutions/cloud.html
6. IBM Cloud Solutions Homepage, https://www.ibm.com/cloud/
7. Rackspace Homepage, https://www.rackspace.com/
8. Openstack Homepage, https://www.openstack.org/
9. VMware Homepage, https://cloud.vmware.com/
10. TheMultiCloudFuture:ChallengesandBenefitsHomepage,https://technodrone.blogspot.com/2014/03/the-multi-cloud-future-challenges-and.html
11. CEP definition in Wikipedia, https://en.wikipedia.org/wiki/Complex_event_processing
12. Cugola, G., Margara, Al.: Processing Flows of Information: From Data stream to complex event processing. Journal of ACM Computing Surveys (CSUR) 44 (Issue 3 –article 15), 15:1-15:62 (2012).
13. Boubeta-Puig, J., Ortiz, G., & Medina-Bulo, I.: Approaching the Internet of Things through Integrating SOA and Complex Event Processing. In Z. Sun, & J. Yearwood (Eds.), Handbook of Research on Demand-Driven Web Services: Theory, Technologies, and Applications (pp. 304-323). Hershey, PA: IGI Global. doi:10.4018/978-1-4666-5884-4.ch014, (2014).
14. Leitner,P., Inzinger, C., Hummer, W., Satzger, B., Dustdar, S.: Application-Level Performance Monitoring of Cloud Services Based on Complex Event Processing Paradigm. In: 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA) (2012).
15. Hirzel, M.: Partition and Compose: Parallel Complex Event Processing. In: DEBS '12- Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, pp. 191-200,ACM, Berlin, Germany (2012).
16. Ku,T., Long-Zhu, Y., Yuan-Hu, K.: A Novel Distributed Complex Event Processing for RFID Application. In: 2008 Third International Conference on Convergence and Hybrid Information Technology , Busan, South Korea (2008).
17. Mdhaffar, A., Halima, R., Jmaiel, M., Freisleben, B.: A Dynamic Complex Event Processing Architecture for Cloud Monitoring and Analysis. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Bristol, UK (2013).
18. Fawaz, P., Hermosillo, G., Rouvoy, R., Seinturier, L.: A Middleware Platform to Federate Complex Event Processing. In: 2012 IEEE 16th International Enterprise Distributed Object Computing Conference (EDOC), pp. 113-122, Beijing, China (2012).

19. Schultz-Møller, N., Migliavacca, M., Pietzuch, P.: Distributed Complex Event Processing with Query Rewriting. In: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS 2009, Nashville, Tennessee, USA (2009).
20. Flouris, I., Manikaki, V., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Mock, M., Bothe, S., Skarbovsky, I., Fournier, F., Štajcer, M., Križan, T., Yom-Tov, J., Volarević, M.: FERARI: A Prototype for Complex Event Processing over Streaming Multi-cloud Platforms. In: DEBS '16 Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems. pp. 348-349. Irvine, CA, USA (2016).
21. Etzion, O., Niblett, P.: Event Processing in Action. 1st edn. Manning Publications Co. Greenwich, CT, USA (2010).
22. Mule Soft Homepage, www.mulesoft.com
23. Rademakers, T., Dirksentt, J.: Open-source ESBs in action.1st edn. Greenwich, CT: Manning, (2009).
24. Case of Bank de Luxemburg from Mulesoft Homepage, https://www.mulesoft.com/case-studies/esb/banque-de-luxembourg
25. Case of Siemens from Mulesoft Homepage, https://www.mulesoft.com/case-studies/api/siemens
26. Esper CEP engine Homepage, http://www.espertech.com/esper/
27. Apache Active MQ Homepage, http://activemq.apache.org/
28. Wahl, A., Hollunder, B.: Automated Performance Measurement for CEP Systems. In: Forth International Conferences on Advanced Service Computing (Service Computation 2012). IARIA 2012, pp.116-121, Nice, France (2012).
29. Cao, K., Wang,Y., Wang, F.: Context-aware Distributed Complex Event Processing Method for Event Cloud in Internet of Things. Journal of Advances in information Sciences and Service Sciences(AISS) 5(8), 121-122 (2013).
30. Xiao, F., Zhan, C., Lai, H., Tao, L., Zhiguo, Q.: New parallel processing strategies in complex event processing systems with data streams. International Journal of Distributed Sensor Networks 13(8), (2017).
31. Dayarathna, M., Perera, S.: Recent Advancements in Event Processing. Journal of ACM Computing Surveys 51(2), 1-36 (2018).
32. Seinturier,L., Merle,P., Rouvoy, R., Romero, D., Schiavoni, V., Stefani, J-B.: A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures. Journal of Software: Practice and Experience (SPE) 42(5), 559-583 (2012).
33. Leitner, P.,Hummer, W., Satzger, B., Inzinger, C., Dustdar, S.: CloudScale- a Novel Middleware for Building Transparently Scaling Cloud Applications. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 434-440. ACM, Trento, Italy (2012).
34. Siddhi WSO2 Homepage, https://docs.wso2.com/display/CEP400/SiddhiQL+Guide+3.0#SiddhiQLGuide3.0-Joins