Real-time complex event processing for cloud resources

M Adam¹,², C Cordeiro¹, L Field¹, D Giordano¹, L Magnoni¹
¹CERN, European Organization for Nuclear Research (CH)
²Acad. of Sciences of the Czech Rep. (CZ)
E-mail: cristovao.cordeiro@cern.ch

Abstract. The ongoing integration of clouds into the WLCG raises the need for detailed health and performance monitoring of the virtual resources in order to prevent problems of degraded service and interruptions due to undetected failures. When working in scale, the existing monitoring diversity can lead to a metric overflow whereby the operators need to manually collect and correlate data from several monitoring tools and frameworks, resulting in tens of different metrics to be constantly interpreted and analyzed per virtual machine. In this paper we present an ESPER based standalone application which is able to process complex monitoring events coming from various sources and automatically interpret data in order to issue alarms upon the resources' statuses, without interfering with the actual resources and data sources. We will describe how this application has been used with both commercial and non-commercial cloud activities, allowing the operators to quickly be alarmed and react to misbehaving VMs and LHC experiments' workflows. We will present the pattern analysis mechanisms being used, as well as the surrounding Elastic and REST API interfaces where the alarms are collected and served to users.

1. Introduction
The integration of cloud resources into the WLCG [1] has progressively been tested and adopted over the past few years. Such approach comes as one response to the computing challenge WLCG is facing resulting from the increasing performance of the LHC [2]. To comply with the current WLCG infrastructure, these cloud resources are being delivered by IaaS providers supplying seamless access to virtual machines (VMs) which will be configured to transparently behave as regular grid worker nodes. Unlike any other WLCG site, when working with external cloud providers the operators have limited control over the computing resources. In addition, the sharing of the underlying physical resources with other customers sometimes has a direct impact on the VMs' performance [3]. Virtualization adds an extra layer of complexity for the cloud operators which have to rely solely on data coming out of the VMs, without any extra information from the infrastructure level, making use of all the existing consumer side monitoring frameworks. Even though these monitoring systems are very detailed and can embrace every single metric needed for the lifecycle management of these virtual worker nodes, their data interpretation and analysis come with the drawback of having the cloud operators constantly analyzing several different and uncorrelated dashboards (job monitoring, data transfer monitoring, performance monitoring, machine monitoring, etc.) and making their subjective correlation of that data. Such approach is not effective nor sustainable and highlights the need
for a more intelligent monitoring approach to allow the automatic interpretation of the existing monitoring data in order to support the lifecycle management of the provisioned resources. This paper proposes an application called Data Analytics from Monitoring, from here on addressed as DAM, that is able to re-purpose the existing raw monitoring data, collected from several different sources, aggregating and analyzing metrics in memory as an incoming uninterrupted stream of events, providing a close to real-time feedback about the status and possible erratic behaviors of the cloud resources. A proof of concept has been conducted during the commercial cloud procurement activity at CERN, with T-Systems [4], and the results are here shown.

2. In-memory processing architecture
The DAM application is written in JAVA and its design (fig. 1) relies on multiple individual plugins that collect raw monitoring data from several different sources into a single queue using proper identifiers for each message type. Every message is then, according to its source, parsed and transformed into single events described via Event Processing Language (EPL) [5] schemas. An ESPER engine [6] makes use of stored EPL statements to run and process the streams of data through, performing all the desired analysis and issuing actions upon certain criteria.

![Figure 1: High level overview of the DAM application modular architecture.](image)

2.1. Collectors
For each monitoring data source there is at least one collector. At the moment, with the existing collectors, one can retrieve data from: Ganglia [7], ElasticSearch [8], ActiveMQ [9], BigPanda, MonAlisa and Dashboard. For ActiveMQ this collector runs as a message listener, consuming data as it arrives in the message brokers. For all the other data sources, each collector is implemented as a java.lang.Runnable for fetching the data. These are scheduled to run periodically from within a thread pool. Once fetched, data is parsed and pushed into a SynchronousQueue<String> queue of events to be processed. All data sources serve different types of metrics and data formats, as described below.

**Ganglia:**
A scalable monitoring system for distributed computing systems such as clusters and Grids. At CERN it is used to monitor cloud resources from either private or public IaaS providers, providing then a general overview of the disperse virtual clusters [10]. This DAM plugin is configured to fetch Ganglia’s raw monitoring data periodically, in a XML format, with a similar frequency to the default one set in the monitoring system (15 seconds).

**ElasticSearch:**
This distributed search and analytics engine is at the center of CERN’s IT unified monitoring architecture [11], storing and serving (through a RESTful API) the metrics collected from the LEMON [12] sensors installed in the Tier-0 and external clouds’ worker nodes. The
DAM plugin for ElasticSearch uses its native API to collect a pre-configured set of metrics periodically, on a variable frequency depending on the desired data.

**ActiveMQ:**
An open-source messaging system widely used at CERN for several use cases, including for the WLCG messaging service [13]. A cluster of ActiveMQ brokers is setup to allow users to publish and consume messages using the STOMP protocol. The messaging queue of interest for the DAM application regards the benchmarking of cloud resource and this data can be streamed in real-time by making use of the ActiveMQ JMS compliance.

**BigPanda:**
A Django based monitoring web service providing an overview of the ATLAS jobs workflow, together with a set of tools for operational support, built on top of the PanDA Workload Management System [14]. Through a RESTful API, the DAM plugin can easily query the BigPanda monitor to obtain a summary overview, in JSON, of the jobs workflow per site.

**MonAlisa:**
A distributed monitoring service [15] used by the ALICE experiment for monitoring all its grid environments including central services, site services, job status and much more. It provides an API from where the DAM plugin can retrieve the number of jobs per type and status.

**Dashboard:**
The Experiment Dashboard system is a framework providing all the components to build common solutions for monitoring of job processing, data transfers and site/service usability [16]. As for BigPanda monitor and MonAlisa, the DAM plugin for Dashboard uses a RESTful API to get an overview of the CMS jobs workflow in JSON format.

### 2.2. Event Processing

Once the ESPER engine is initialized, all the EPL modules previously specified by the user are loaded. These modules are represented by plain text files in which EPL statements are written, bundled with optional deployment instructions. The Event Processing Language is an extended SQL-standard language used to express filtering and aggregations, possibly over sliding windows of multiple event series. Instead of tables, the data sources are streams, having events as the basic data unit rather than rows. It also includes pattern semantics to express causality among events. An EPL statement reads all the data from one or more data streams, applies all the specified operations and can output the result into another data stream, which enables the user to create complex dataflows. EPL also provides a concept of table as a data structure with a primary key and multiple columns used to store a state of aggregation. Tables can be accessed by any EPL statement in order to enhance the stream being processed or for the sole purpose of fetching and outputting the data. All the data in tables are stored in memory, providing a fast response time. After initialization, the main application thread starts pulling data from the `SynchronousQueue<String>` events queue and sending into the engine for processing. Having as the main goal the behavioral analysis of virtual resources, the processing model used by the application features 4 different layers of EPL statements, as exemplified in figure 2, each applied on data outputs from the previous one. The 1st layer classifies the state of the virtual resource (host or virtual machine) at the current time $t$, assigning one of the three states: *OK*, *WARNING* or *ERROR*. The classification is based on one or a combination of several metrics from the raw data. To provide a higher level analysis, an average cluster performance is also computed and...
classified the same way in this processing layer. The following layer is applied on both of these resulting data streams. The 2\textsuperscript{nd} layer outputs a stream of time-based events representing state transitions of a given condition for a given host or cluster. This state transition model is the foundation for the DAM alarming system: "the user shall be alerted when either a host begins or stops behaving erratically". The 3\textsuperscript{rd} layer filters the 2\textsuperscript{nd} layer events trying to identify those worth of intervention and consequent notification to the respective users, while the 4\textsuperscript{th} enriches this classification by doing a pattern analysis of the sequence of events, aiming on detecting situations where the hosts are flapping back and forth between \textit{ERROR} and \textit{WARNING} states.

![Layered processing model](image)

**Figure 2: Layered processing model.**

2.3. Output

Data extraction from the ESPER engine is handled by an instance of a class implementing the \texttt{UpdateListener} interface, attached to statements through \texttt{@Listener} annotations in the EPL modules. In this project a broad range of output options were implemented in several listeners. The usual method of informing a user about an alarm is using email notifications, therefore a listener sending user formatted email alarms was created and attached to the top level statement. Through the application’s configuration file, users can opt for receiving emails with host or cluster granularity, depending on their use case. Other listeners were also developed as an integral part of the application, allowing the insertion of processed data into an ElasticSearch cluster, allowing further data evaluation and visualizations of the processed data with Kibana [17].

Being the goal of the application the automatic interpretation of monitoring data as an assisting mechanism for resource management, the processed data has to be accessed not only by the operators in forms of emails or other visualizations, but mainly by other applications, programatically. A REST API is deployed for this purpose, collecting and serving data temporarily in ESPER tables. On top of this API a web interface was created, based on the Spring MVC framework [18], serving the user live data processing results on host and cluster performance, in the form of browsable jQuery data tables and D3.js graphs.

3. Proof of concept

Along with the several commercial cloud activities conducted within the IT department at CERN, the DAM application has been gradually tested and evaluated with the existing monitoring systems at the time of each cloud activity. For a final and last proof of concept, the application has supported the operation of the last commercial cloud activity with TSystems, which lasted for approximately 90 days running up to 3800 vCPUs simultaneously.

For this exercise, the application has been configured to collect raw monitoring data from LEMON for the VMs’ status and from BigPanda and MonAlisa for job status summaries. As an output, all of the analysis have been stored in a dedicated ElasticSearch index, a RESTful
Web interface has been set and notifications (in the form of emails) have been sent. The data analysis for this PoC comprised conditions to detect and report erratic situations such as:

- the aggregated CPU idle average (%), per server and cluster, over a configurable sliding time window is higher than 70% (WARNING) and 80% (ERROR),
- the instant CPU IO Wait per node is higher than \( \left( \frac{1}{\text{#cores}} \right) \ast 100 \),
- the percentage of memory swap used is higher than 80% (WARNING) and 90% (ERROR),
- the current average growth rate \( r \) for the past day will imply a disk full within another day, driven from
  \[
  \frac{\log(MB_{\text{total}})}{\log(1 + r)}
  \]
  having \( r \) computed from the basic geometric growth over time given by
  \[
  MB_{\text{now}} = MB_{\text{1dayAgo}}(1 + r)^t \iff r = \frac{MB_{\text{now}}}{MB_{\text{1dayAgo}}^{(1/t)}} - 1,
  \]
  where \( t \) stands for the amount of data points during the 1 day interval.

In total, approximately 81 million messages have been stored in ElasticSearch, having around 80.9 million coming from three different host and cluster processing streams: CHECK, STATUS and NOTIFICATION. The remaining messages (~131k) are related to job workflow monitoring events processed to provide summary statistics about the ALICE and ATLAS workloads running in the cloud. Table 1 shows the stored messages’ splitting per data stream and granularity.

| message.stream     | message.granularity | count    |
|--------------------|---------------------|----------|
| check              | cluster             | 16161770 |
| clusterNotification| cluster             | 199      |
| check              | host                | 6386326  |
| hostStatus         | host                | 498313   |
| hostNotification   | host                | 385587   |
| jobStats           | job                 | 131854   |

The hostNotification and clusterNotification streams are directly mapped to email notifications. Cluster wise and over a period of 3 months, ~200 emails is a reasonable information level, while host wise, the almost 400 thousand emails are unbearable and considered as heavy spamming. This issue is addressed on the next chapter.

From what concerns the behavioral conditions listed above, ”high cpu_idle” was the most frequent issue (fig 3a), having 55% of these notifications alerting for erratic hosts not running any jobs or simply stuck, and therefore wasting computing power.

Regarding the job related analysis, during the activity the DAM application has processed summary data in order to infer and display the instantaneous amount of running jobs (fig 3b), weekly average and tendency for the amount of jobs to increase or decrease based on the previous metrics.
(a) Splitting of the processed events per condition, with "high cpu_idle" clearly dominating the most frequent condition verified.

(b) Stacked graph of the instantaneous amount of running jobs in TSystems for ATLAS and ALICE.

Figure 3: Statistics from the processed data collected in ElasticSearch.

The DAM web interface output has proven to be the most useful mechanism as it provided live feedback from the computed analysis, allowing the resource managers to promptly react upon undesired scenarios, constantly identifying and recycling VMs that were not being used properly, either due to misconfigurations or underuse of the infrastructure. The web interface also permitted users to get some extra visualizations that are not possible through the usual monitoring means, originating from the DAM event processing and providing overviews on a cluster level about the CPU usage instantaneous spread (like the example of fig 4) and history.

Figure 4: Instantaneous CPU usage spread on the full TSystems compute cluster.

4. Alarming optimization with machine learning (ML)
As part of the effort to improve the applications notifications usability, the integration of machine learning techniques was investigated. The input data was parsed into ESPER events, which were then fed into the machine learning module for classification. Its output was reinserted into ESPER as a Check data stream for the 2nd layer processing, and handled as if the classification was done using EPL. The main focus when developing the ML module was reducing the number of notifications caused by a small number of hosts rapidly changing states. This is handled
inefficiently by the original EPL statements that use averaging over a time window to smoothen the fluctuations of the incoming metric values. The classification of the ML module was done using the variance and mean of a vector with ten values. Only the vectors with extreme mean and low variance, that clearly suggested that the host was either without problems or behaving erratically, were used in the training set. A first optimization attempt using simple naïve Bayes classifier turned out to have a very comparable performance to the existing 1st EPL layer. The need for a more sophisticated classifier was evident, therefore on a second iteration, an implementation of the AdaBoost [19] algorithm in the JSAT [20] library was used. Being a compound classifier, AdaBoost also outputs both result and its confidence about the result correctness, enabling again to restrict the output to only the most liable ones (the threshold was set to 90%, meaning the classifier only outputs when 90% of it’s weak learners agree on the result). When sampling data in a 1 minute interval, meaning an input vector of the classifier representing a 10 minute data history, the number of notifications issued decreased by a factor of 3 when compared to EPL classification and the number of events in the status stream decreased by a factor of 6. The ML classifier withheld less than 7% check events due to the restrictions on confidence. This indicates that the classification by the ML module is recognizing a change in the server state more slowly, thus being able to smoothen peaks more effectively. Consequently, state transition from the ML point of view occurs more abruptly. Figure 5 shows exactly this, comparing on a logarithmic scale not only the large optimization in terms of number of status transitions but also how flapping proof the application can become by using ML to only notify on clear erratic scenarios. Even though the average number of ”right” and ”wrong” hits by the ML module were not measured, by looking at this plot one can clearly observe that the high peaks of the ML module correlate with the ones from the raw data in grey. Any non-matching hits have been considered to be a result of the above mentioned algorithm’s confidence threshold.

Figure 5: Histogram of the metric cpu_idle, with the raw values in grey and EPL (blue) and ML (green) status transitions.

The factor of improvement in the number of notifications is lower than the factor of improvement in the number of status changes. This is simply justified by implying that the notification filtering heuristics present on the 3rd and 4th layers (figure 2) of the DAM architecture were tailored to limit the spamming on the original EPL dataflow and not the tested ML one.

5. Conclusion
The management of remote virtual resources can be a very challenging task for an operator, mainly when the interpretation and correlation of the respective monitoring data is done
The introduction of a stream based in-memory processing model based on state transitions can play a crucial role in this area, not only assisting in the digestion and analysis of raw monitoring data but also re-purposing it for a richer and more meaningful prompt feedback. After tested with several private and commercial clouds, the DAM application has proven to be an added value when it comes to the lifecycle management of VMs, contributing to the identification of erratic behaviors which ultimately led to VM re-configuration or even termination and recycling. During the last CERN IT commercial cloud procurement activity with T-Systems, the DAM application successfully processed about 81 million raw monitoring messages, coming from 3 different monitoring infrastructures. During a period of 3 months, it has issued 200 alarms, cluster wise, with added information about every single VM and its behavior over adjustable time windows. As an improvement to the application model, an AdaBoost based classifier has been tested, enhancing the notifications quality on a host level and thus reducing the alarming spam by a factor of up to 3. This was achieved by using a training set of 3000 vectors containing extreme use cases.

References

[1] Ian Bird. Computing for the Large Hadron Collider. Annual Review of Nuclear and Particle Science, 61(1):99–118, 2011.
[2] I Bird, P Bunic, F Carminati, M Cattaneo, P Clarke, I Fisk, M Girone, J Harvey, B Kersevan, P Mato, R Mount, and B Panzer-Steindel. Update of the Computing Models of the WLCG and the LHC Experiments. Technical Report CERN-LHCC-2014-014. LCG-TDR-002, Apr 2014.
[3] D Giordano et al. Benchmarking Cloud Resources for HEP. to appear in J. Phys.: Conf. Ser., 2016.
[4] D Giordano et al. CERN Computing in Commercial Clouds. J. Phys.: Conf. Ser., 2016.
[5] Epl reference: Clauses. http://www.espertech.com/esper/release-5.3.0/esper-reference/html/epl_clauses.html. Accessed: 2016-12-06.
[6] Esper. http://www.espertech.com/products/esper.php. Accessed: 2016-12-06.
[7] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: Design, implementation and experience. Parallel Computing, 30:2004, 2003.
[8] Elasticsearch. https://www.elastic.co/products/elasticsearch. Accessed: 2017-03-23.
[9] Activemq. http://activemq.apache.org/. Accessed: 2017-03-23.
[10] C Cordeiro, A Di Girolamo, D Giordano, L Field, D Spiga, and L Villazon. Monitoring the delivery of virtualized resources to the LHC experiments. J. Phys.: Conf. Ser., 664(2):022013, 2015.
[11] A Aimar et al. Unified Monitoring Architecture for IT and Grid Services. to appear in J. Phys.: Conf. Ser., 2016.
[12] Babik Marian, Fedorko Ivan, Hook Nicholas, Lansdale Thomas Hector, Lenkes Daniel, Siket Miroslav, and Waldron Denis. Lemon - lhc era monitoring for large-scale infrastructures. J. Phys.: Conf. Ser., 331(5):052025, 2011.
[13] L Magnoni. Modern messaging for distributed sytems. J. Phys.: Conf. Ser., 608(1):012038, 2015.
[14] J Schovancova, K De, A Klimentov, P Love, M Potekhin, and T Wenaus. The next generation of the ATLAS PanDA Monitoring System. Technical Report ATL-SOFT-PROC-2014-003. CERN, Geneva, Apr 2014.
[15] H. B. Newman, I. C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu. MonALISA : A Distributed monitoring service architecture. eConf C0303241:MOET001, 2003.
[16] J Andreeva, M Cinquilli, D Dieguez, I Dzhunov, E Karavakis, P Karhula, M Kenyon, L Kokoszkiewicz, M Nowotka, G Ro, P Saiz, L Sargsyan, J Schovancova, and D Tuckett. Experiment dashboard - a generic, scalable solution for monitoring of the lhc computing activities, distributed sites and services. Journal of Physics: Conference Series, 396(3):032093, 2012.
[17] Kibana. https://www.elastic.co/products/kibana. Accessed: 2016-12-06.
[18] Spring mvc framework. http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html. Accessed: 2016-12-06.
[19] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119 – 139, 1997.
[20] Java statistical analysis tool, a java library for machine learning. https://github.com/EdwardRaff/JSAT. Accessed: 2017-02-09.