The AAL project: automated monitoring and intelligent analysis for the ATLAS data taking infrastructure

A Kazarov[1], **G Lehmann Miotto**[2] **and L Magnoni**[2]

[1] CERN, on leave from PNPI, St.Petersburg, Russian Federation
[2] CERN, European Laboratory for Particle Physics (CERN), Geneva, Switzerland

E-mail: {`andrei.kazarov, giovanna.lehmann, luca.magnoni`}`@cern.ch`

Abstract. The Trigger and Data Acquisition (TDAQ) system of the ATLAS experiment at CERN is the infrastructure responsible for collecting and transferring ATLAS experimental data from detectors to the mass storage system. It relies on a large, distributed computing environment, including thousands of computing nodes with thousands of application running concurrently. In such a complex environment, information analysis is fundamental for controlling applications behavior, error reporting and operational monitoring. During data taking runs, streams of messages sent by applications via the message reporting system together with data published from applications via information services are the main sources of knowledge about correctness of running operations. The flow of data produced (with an average rate of O(1-10KHz)) is constantly monitored by experts to detect problem or misbehavior. This requires strong competence and experience in understanding and discovering problems and root causes, and often the meaningful information is not in the single message or update, but in the aggregated behavior in a certain time-line. The AAL project is meant at reducing the man power needs and at assuring a constant high quality of problem detection by automating most of the monitoring tasks and providing real-time correlation of data-taking and system metrics. This project combines technologies coming from different disciplines, in particular it leverages on an Event Driven Architecture to unify the flow of data from the ATLAS infrastructure, on a Complex Event Processing (CEP) engine for correlation of events and on a message oriented architecture for components integration. The project is composed of 2 main components: a core processing engine, responsible for correlation of events through expert-defined queries and a web based front-end to present real-time information and interact with the system. All components works in a loose-coupled event based architecture, with a message broker to centralize all communication between modules. The result is an intelligent system able to extract and compute relevant information from the flow of operational data to provide real-time feedback to human experts who can promptly react when needed. The paper presents the design and implementation of the AAL project, together with the results of its usage as automated monitoring assistant for the ATLAS data taking infrastructure.

1. Introduction

This paper presents the AAL project, a new tool meant at providing automated monitoring and intelligent analysis for the ATLAS data taking infrastructure. First we introduce the data taking facilities of the ATLAS experiment at CERN and the motivation which drove this new development. In Section 2 we describe the AAL project, the main functionalities and the different technologies we are using. In section 3 we present the AAL engine architecture, in section 4 the visualization strategies we implemented and finally in section 5 we describe how the tool has been adopted and the results and conclusions.

1.1. ATLAS TDAQ system

ATLAS [1] (A Thoroidal LHC Apparatus) is a particle physics experiment at the Large Hadron Collider at CERN. The Trigger and Data Acquisition (TDAQ) system of ATLAS is responsible for selecting and transferring ATLAS experimental data from the detector to the mass storage system. To cope with the very high data rate produced by the detector a complex and distributed computing infrastructure has been built. More than 200 switches and routers interconnect around 2000 hosts to build a real-time filtering system for particle collision events. On top of the hardware infrastructure there are over 20.000 applications responsible of analyzing, filtering and moving event data to permanent storage [2].

1.2. Motivation: improving monitoring with automation and event correlation

The TDAQ system is operated by a non-expert shift crew, assisted by a set of experts providing knowledge for specific components. The daily work of operators is made of procedures to run the system, periodic checks and controls on system status, well defined reaction in case of known problems and interaction with experts in case of non standard issues. The evaluation of correctness of running operations requires strong competence and experience in understanding problems and root causes, and often the meaningful information is not in the single message or update, but in the aggregated behavior in a certain time-line. The AAL project automatizes checks and controls, detects complex event patterns over time and helps operators with more pertinent and meaningful information for system analysis.

2. The AAL project

The AAL project is meant at assuring a constant high quality of problem detection by automating most of the monitoring tasks and providing real-time correlation of operational data and system metrics. AAL guides operators and experts in their daily work, for this reason is also known as *"The shifter assistant"*. It performs a real-time analysis of the whole TDAQ system, detecting problematic situations and misbehaviour and producing **alerts** suggesting expected reactions. This project combines technologies coming from different disciplines, in particular it leverages on an Event Driven Architecture to unify the flow of data to be monitored, on a Complex Event Processing (CEP) engine for real time correlation of events and pattern recognition and on a Message Queuing system for components integration and communication. The picture in Figure 1 presents an overview of the system together with the three operational stages: **information gathering**, **information processing** and **result distribution**.

2.1. Information gathering

Evaluation of TDAQ working conditions requires data collection from multiple data sources, each one with different technology, publication mechanism and format. AAL is able to gather data from these different information providers in order to detect problems and failures. Farm metrics, such as machines status and problems, are collected from the Nagios tool [3]. TDAQ operational data is gathered via the streams of messages produced by applications via the Message Reporting System (MRS) [4] and from the information published via the Information Services (IS) by running applications. As illustrated in Figure 1, system metrics and operational data can be abstracted as a flow of information composed on many single events over time. Moreover, AAL is also able to retrieve more static information at run time in a pull fashion via reader plug-ins, such as the case for system topology from the configuration database.

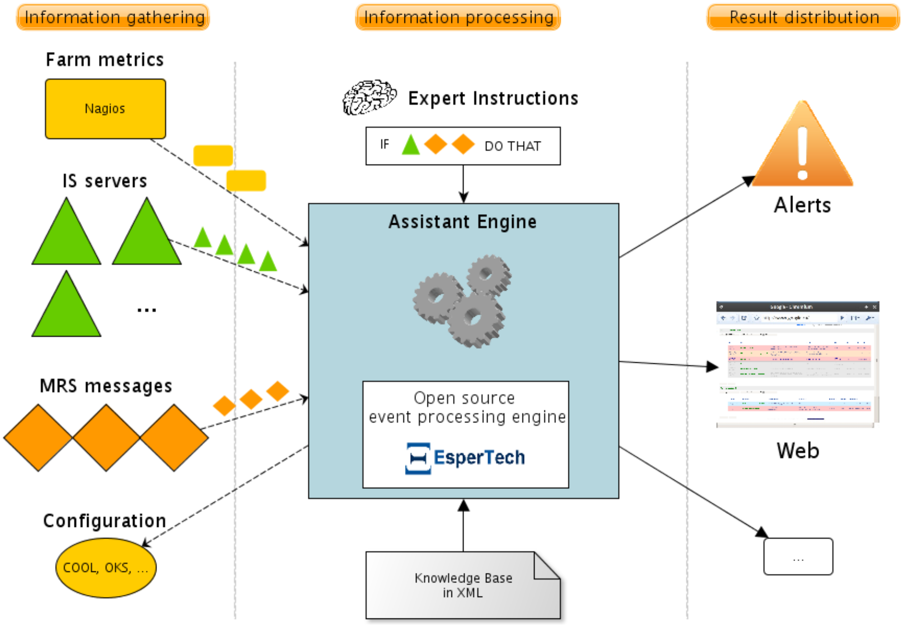


Figure 1. Overview of the AAL project.

2.2. Information processing
Information processing is the key functionality of the tool. To detect problems and to perform root cause analysis of failures AAL needs to correlate at real-time possibly different streams of events, detecting specific time-based patterns previously defined by experts. The main requirements for the processing system are:

- Real-time processing for fast error detection and reporting;
- Fine-grained information processing for specific event(s) detection;
- Events aggregation for detection of patterns over time or other properties;
- Effective representation of patterns and correlations, similar to the SQL query language for database systems.

We found a very effective solution coming from the Complex Event Processing field of research, mainly adopted in financial analysis and advanced system monitoring.

2.2.1. Complex Event Processing Complex Event Processing (CEP) [5] is one of the fundamental techniques in the so called Operational Intelligent procedures [6], where the goal is to to provide insight into business operations by running query analysis against live feeds and event data. In the last decade it has been adopted in very different contexts, from financial analysis to business process management and intelligent system monitoring. CEP systems are meant to analyze and correlate streams of event, but the great advantages they offer is to abstract events structure and to provide a standard language to express pattern and correlation as queries.

2.2.2. Esper In the CEP world there are several solutions, both commercial and open source, offering event processing functionalities based on SQL-like processing languages to express patterns and rules. Esper [7] is a Complex Event Processing engine written in Java that offers a very powerful Event Processing Language (EPL) with built-in capabilities for analysis over time and event correlation. Esper is able to handle very high flow of data, thanks to a non-persistent approach where events pass through memory and the engine continuously check for the matching criteria expressed by queries. The continuous queries approach perfectly matches with our real-time requirements and this, together with the powerful functionalities offered by the EPL language, convinced us to adopt Esper for building the information processing engine of AAL. EPL queries are used to express pattern describing problems and failures. The Figure 2 illustrate graphical examples of types of patterns detect by AAL relying on Esper capabilities.

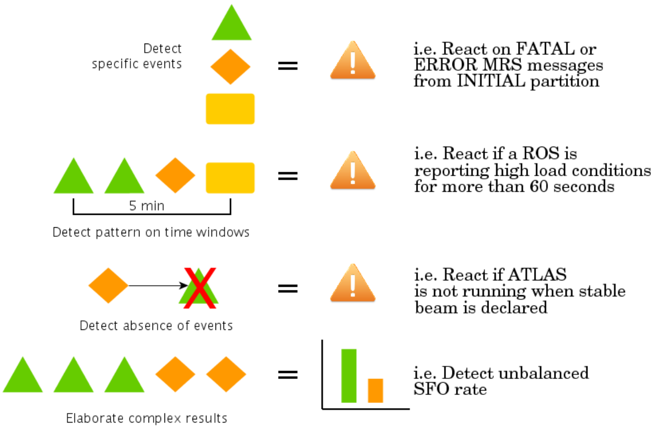
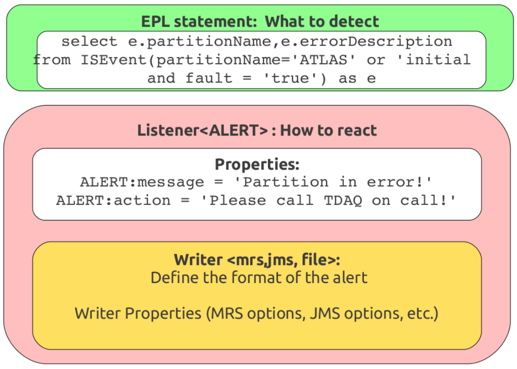

Detect specific events
=
i.e. React on FATAL or ERROR MRS messages from INITIAL partition
5 min
Detect pattern on time windows
=
i.e. React if a ROS is reporting high load conditions for more than 60 seconds
Detect absence of events
=
i.e. React if ATLAS is not running when stable beam is declared
Elaborate complex results
=
i.e. Detect unbalanced SFO rate

Figure 2. Simple and complex patterns detected by the AAL engine.


EPL statement: What to detect
Listener<ALERT> : How to react
Properties:
Writer <mrs,jms, file>:
Define the format of the alert
Writer Properties (MRS options, JMS options, etc.)

```
 select e.partitionName,e.errorDescription
from ISEvent(partitionName='ATLAS' or 'initial
      and fault = 'true') as e
```

```
ALERT:message = 'Partition in error!'
ALERT:action = 'Please call TDAQ on call!'
```

Figure 3. Directive schema.

2.3. Knowledge engineering: directives and alerts

TDAQ system experts have to feed AAL with instructions about what situations to detect and suggestions on how to react. This knowledge engineering process is fundamental to build and maintain an up to date data set of problems and errors. These instructions, named *directives*, are codified in XML documents and loaded by the engine at configuration time. Every directive generates one or more alerts. An alert is composed of:

- **Problem description**: brief description of the problematic situations.
- **Reaction**: expected reaction to be performed by the operator.
- **Domain(s)**: one or more category defining who should care about the problem. This information is mainly used for routing of the alerts at visualization.
- **Events details**: events that triggered the alerts.

A directive, as represented in Figure 3, is composed by two main elements: the *query*, that is the Esper EPL statement that defines the event pattern to react on and the *listener*. The listener defines alert details and one or more writers responsible for alert distribution. The same alert can be produced by the engine in different formats by dedicated *writers*, as described in section 3.2.

2.4. Alerts visualization and distribution

For a fast and effective reaction on system failures alerts have to be distributed in several ways to operators and experts. The main goal is to build an effective real time visualization for operators in the ATLAS control room, that need to promptly receive information with problem details and expected reactions. Moreover, additional alert distribution strategies exist for remote monitoring purpose. This has been accomplished with a generic architecture of the AAL engine, as presented in section 3, and by a message oriented communication, presented in section 4.

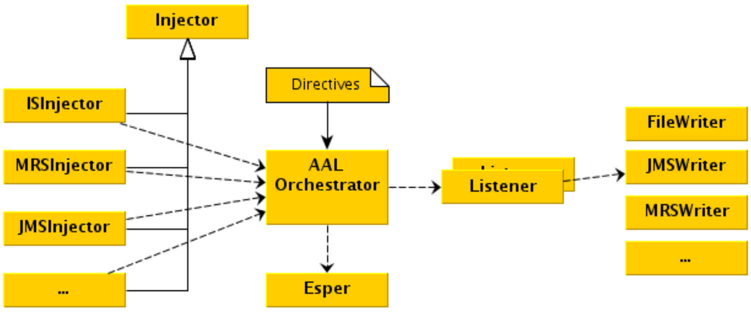


Figure 4. The AAL engine architecture.

2.5. AAL components

The AAL project is composed of the following components:

- **The AAL engine**: a standalone Java-based service responsible for data collection and information processing;
- **The AAL web frontend**:a web application to visualize and interact with alerts;
- **A message broker** for components communication.

3. The AAL engine

The main role of the AAL engine is to orchestrate the interactions between information providers and the complex event processing engine, as shown in Figure 4. It parses the knowledge base at configuration time and it initializes all the corresponding queries and statements in Esper. It then handles all the information gatherer components and create the writers and listeners ready to produce alerts when patterns are detected. The flow of events coming from all data sources is then constantly processed by the Esper engine and every time an event matches a directive query or pattern the proper alert is created and distributed.

3.1. Injectors

The set of components responsible for data collection are called *injectors*, as shown in Figure 4. Injectors implement a generic design that allows to extend the system with new information sources without affecting the collection logic. Injectors specialization exists for the different data source types but they expose functionalities via the same interface to the engine. For data sources with high rate streams of event, like the IS servers that can reach up to 50 KHz of information updates, injectors leverages on the CORBA-based IPC mechanism provided by TDAQ infrastructure [8]. In other cases, like for the Nagios metrics, we hide the complexity of the data source by using a message queuing system to collect and distribute information.

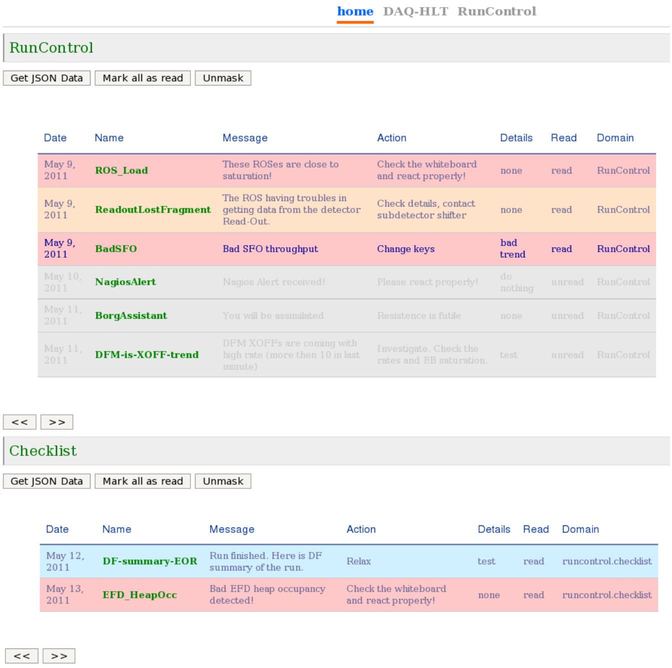
3.2. Writers

The *writer* components are responsible to codify and distribute alerts in specific formats. A directive specifies one or more *writers* elements to define how the generated alert have to be propagated. The default format is XML (Extensible Markup Language), suitable for machine-based processing and distributed via the message queuing as described in section 4.1. But alerts can also be written to file in more human-readable format or can be sent as standard TDAQ application messages.

3.2.1. JMS writer To build a scalable and modular system we adopted a message oriented architecture where the communication take place via exchanging messages. A message queuing system, or message broker, is used to distribute alerts offering a publish/subscribe interface. More information about the chosen tool is given in section 4.1. In order to be independent from the message broker implementation the JMS writer send alert as a XML message using the generic Java Message Service (JMS) API [9].

3.2.2. MRS writer To be compatible with the TDAQ architecture alerts can be distributed also as MRS messages relying on the existing message reporting facilities [4].This functionality is provided by the MRS writer. This integration allow to use the alert produced by AAL to trigger automatic reaction and recovery procedures via the existing error recovery facilities [10].

3.2.3. Mail and RSS When a critical situation requires a prompt reaction from users and experts alerts can be explicitly sent as e-mail to a set of addresses and can also be delivered as sms thanks to the mail to sms service provided by CERN. Moreover, AAL relies on the message broker functionalities to send alerts as a RSS feed stream, easy to integrate in any mail reader or feed reader application.



RunControl

Date	Name	Message	Action	Details	Read	Domain
May 9, 2011	ROS_Load	These ROSes are close to saturation!	Check the whiteboard and react properly!	none	read	RunControl
May 9, 2011	ReadoutLostFragment	The ROS having troubles in getting data from the detector Read-Out.	Check details, contact subdetector shifter	none	read	RunControl
May 9, 2011	BadSFO	Bad SFO throughput	Change keys	bad trend	read	RunControl
May 10, 2011	NagiosAlert	Nagios Alert received!	Please react properly!	do nothing	unread	RunControl
May 11, 2011	BorgAssistant	You will be assimilated	Resistence is futile	none	unread	RunControl
May 11, 2011	DFM-is-XOFF-trend	DFM XOFFs are coming with high rate (more then 10 in last minute)	Investigate. Check the rates and EB saturation.	test	unread	RunControl

Checklist

Date	Name	Message	Action	Details	Read	Domain
May 12, 2011	DF-summary-EOR	Run finished. Here is DF summary of the run.	Relax	test	read	runcontrol.checklist
May 13, 2011	EFD_HeapOcc	Bad EFD heap occupancy detected!	Check the whiteboard and react properly!	none	read	runcontrol.checklist

Figure 5. Web application for alerts visualization.

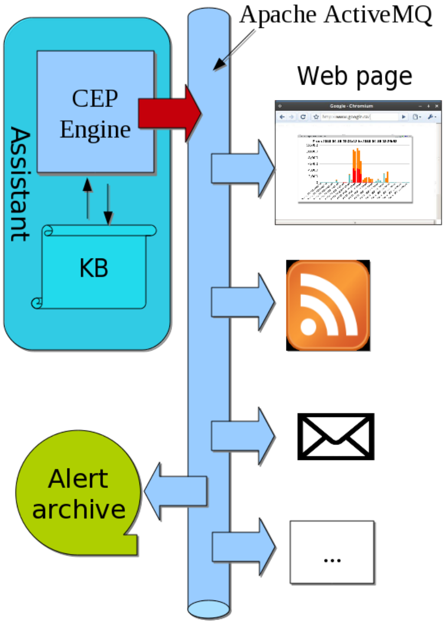


Figure 6. Message oriented communication for modules interaction.

4. Web based visualization

Building a web based visualization service was the natural choice to gain independence from user location, hardware and software, all critical features for a monitoring system. A Rich Internet web Application (RIA) has been built allowing users to perform actions on the page and interact with alerts visualized. The web application is developed using the Django framework [11]. Alerts are collected by an archiver plug-in connected to the message broker via the Streaming Text Orientated Messaging Protocol (STOMP) protocol. Once received, alerts are archived in a SQLite database and made available to the web application. Web pages are automatically updated when new messages arrive, providing a real-time feedback in case of problems.

Users can interact with the web page by making the following actions:

- Alerts can be marked as processed when the problem is addressed;
- Classes of alerts can be masked for a certain time interval;
- Select and sort alerts per properties (Date, Name, etc.).

Web pages have a basic structure composed by several sections, as presented in Figure 5. Every section is a table visualizing alerts that match with one or more criteria. The structure is easily customizable, so experts can built their own page layout, defining which alerts they want to receive in every sections.

4.1. Message oriented architecture

The Figure 6 presents the different software involved in building the web visualization: a web application, an alert archiver, the RSS server and possibly other generic alert consumers. As introduced in the previous sections AAL relies on a message oriented communication where a message queuing system, or message broker, distributes alerts offering a publish/subscribe interface. We chose ActiveMQ as open source message broker from the Apache software foundation [12]. It is compatible with the JMS [9] interface, the standard for message oriented solutions, it supports multiple wire protocols and multiple network protocols. Moreover, it provides several cross language clients for non-Java applications.

4.1.1. Network of message brokers The AAL engine runs on a dedicated host on the TDAQ computing farm of the ATLAS experiment. Access to this computing facility is highly restricted and direct communications are not possible from the general public CERN network. While shifters operate from a control room inside this restricted area, experts are in most cases outside this facility. To allow remote monitoring alerts produced by AAL have to be made available both inside and outside the confined network. This is accomplished by a master-slave configuration of 2 ActiveMQ instances. The master is running inside the TDAQ farm and it receives all the alerts from the engine. It then forwards all messages to the slave instance, running on the public network, via an outgoing TCP connection.

5. Use cases

The AAL tool is used in production to assist monitoring and operational procedures during ATLAS data taking runs. More than 100 directives has been coded gathering requirements from very different TDAQ domains. The following are some practical situations detected by the tool and the corresponding EPL statements:

- analysis of problems reported by Nagios (the farm monitoring tool):

  -
    ```
    select * from NagiosAlert(hostname regexp '(?!pc-atlas-pub-[0-9]+.*')
    group by hostname output every 1 minute
    ```
- detect bunch of error messages as symptom of an issue that impact most of TDAQ applications, such as the lost of connectivity for a read-out machine (ROS):

  -
    ```
    select ROS from ROS.win:time(40 sec).std:groupwin(ROS, Message)
    having size > 100 output first every 120 seconds
    ```
- detect critical amount of read-out errors on the detectors (not easy to spot by a human eye):

  -
    ```
    select Crate, NofBadLinks from SCTLinkErrorsCounter
    having NofBadLinks > 400 and PARTITION_IS_RUNNING output every 2 minute
    ```

6. Conclusion

Effective monitoring and promptly error detection are fundamental properties for maximizing the data taking efficiency of high energy physics experiments. The main goal of the AAL project is to assist ATLAS operators with automated analysis of system conditions and intelligent reasoning for root-cause problems analysis. AAL has been implemented leveraging on modern event correlation techniques combined with message driven architecture for components integration and web technology for alerts visualization. The tool has been quickly adopted in the ATLAS TDAQ community and it gathered the interest from other groups with similar monitoring requirements, such as networking and system administration.

The main area of research for the future is to investigate the possibility to build a learning module to detect anomalies in an unsupervised manner, as proven by the HOLMES project [13]. Furthermore, the web interface will be extended to include directives management and system administration together with improved user experience.

References

[1] ATLAS Collaboration 2008 *The ATLAS Experiment at the CERN Large Hadron Collider.* Journal of Instrumentation, S08003.

[2] ATLAS Collaboration 2003 *ATLAS High-Level Trigger, Data Acquisition and Controls Technical Design Report* http://cdsweb.cern.ch/record/616089/

[3] Nagios Enterprises http://nagios.org/

[4] Fedorko I 2007 *The Message Reporting System of the ATLAS DAQ System* ICATPP Conference on Astroparticle, Particle, Space Physics Detectors and Medical Physics Applications

[5] Luckham D 2002 *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems* (New York: Addison-Wesley Professional)

[6] Revised Selected Papers 2008 *Business Intelligence for the Real-Time Enterprise.* Second International Workshop, BIRTE 2008, Auckland, New Zealand

[7] Bernhardt T http://esper.codehaus.org/

[8] Liko, D et al 2004 *Control in the ATLAS TDAQ System.* Computing in High Energy Physics and Nuclear Physics. Interlaken, Switzerland

[9] Oracle-Sun The Java Message Service API http://docs.oracle.com/cd/B1409919/jms.htm

[10] Corso-Radu A, Garcia R M, Kazarov A, Miotto G L, Magnoni L and Sloper J 2010 *Applications of expert system technology in the Atlas TDAQ controls framework* International Conference on Software and Data Technologies, Athens, Grece.

[11] The Django software foundation Django framework https://www.djangoproject.com

[12] The Apache Software foundation Activemq http://activemq.apache.org/

[13] P H dos Santos Teixeira and R G Clemente and R A Kaiser and D A Vieira Jr 2010 *HOLMES: An event-driven solution to monitor data centers through continuous queries and machine learning.* DEBS10, July 12-15, 2010, Cambridge, UK