User Centric Job Monitoring – a redesign and novel approach in the STAR experiment

D Arkhipkin1, J Lauret1, Y Zulkarneeva

1 Physics Department, Brookhaven National Laboratory, Upton, NY 11973-5000 USA
E-mail: arkhipkin@bnl.gov, jlauret@bnl.gov, julia.zulkarneeva@gmail.com

Abstract. User Centric Monitoring (or UCM) has been a long awaited feature in STAR, whereas programs, workflows and system “events” could be logged, broadcast and later analyzed. UCM allows to collect and filter available job monitoring information from various resources and present it to users in a user-centric view rather than an administrative-centric point of view. The first attempt and implementation of “a” UCM approach was made in STAR 2004 using a log4cxx plug-in back-end and then further evolved with an attempt to push toward a scalable database back-end (2006) and finally using a Web-Service approach (2010, CSW4DB SBIR). The latest showed to be incomplete and not addressing the evolving needs of the experiment where streamlined messages for online (data acquisition) purposes as well as the continuous support for the data mining needs and event analysis need to coexists and unified in a seamless approach. The code also revealed to be hardly maintainable. This paper presents the next evolutionary step of the UCM toolkit, a redesign and redirection of our latest attempt acknowledging and integrating recent technologies and a simpler, maintainable and yet scalable manner. The extended version of the job logging package is built upon three-tier approach based on Task, Job and Event, and features a Web-Service based logging API, a responsive AJAX-powered user interface, and a database back-end relying on MongoDB, which is uniquely suited for STAR needs. In addition, we present details of integration of this logging package with the STAR offline and online software frameworks. Leveraging on the reported experience and work from the ATLAS and CMS experience on using the ESPER engine, we discuss and show how such approach has been implemented in STAR for meta-data event triggering stream processing and filtering. An ESPER based solution seems to fit well into the online data acquisition system where many systems are monitored.

1. Introduction
User-centric monitoring (UCM) has been envisioned to be the panacea to logging both information typically streaming from users application through log and error files as well as broadcasting event properties such as CPU time, memory usage or IO throughput. Such information, whenever jobs are run in a distributed environment (Cloud or Grids) are not accessible in real-time and this creates significant issues in detecting problems as they occur. Understanding the sheer amount of information from a Cloud based processing data production campaign or user analysis also strongly requires (a) a highly scalable scheme able to support tremendous IO rate (b) a highly available service (utilizing local cache in case of network downtimes) (c) the ability to present data in layers (drilling from global to more detailed information) and (d) the ability to summarize and mine the data for problem detection and troubleshooting. UCM was also seen as an over-arching component able to collect information
from operating system components or gatekeeper’s sanity [1], seen as yet another kind of “user’s application” helping to gather a more complete picture of a workflow.

To this aim, the RHIC/STAR experiment launched three waves of upgrade of its application framework with installments in stages to prepare for a fully scalable UCM implementation. The project first replaced all IO layers by a “logger” components leveraging log4cxx plug-in back-end by putting in place the basic building blocks for plug-and-play “adapters”. The second phase of this project was to study (via a Small Business Innovation Research (SBIR) proposal [2] in 2006/2007) the feasibility of developing a custom UCM appender suitable for distributed computing based processing. The notion of message format, layered structure of the information and messages, firewall and message proxy, caching were all laid out in the work concept but the project came short on deliverables as the database back-end scalability was barely ensured by separating the access by users. In its third phase, a Web-service approach was attempted (2010) generalizing on any database access. While the resulting UCM logger handled local caching, it only supported direct pushes to the database via a web service emulation. The scheme would not scale for large numbers of clients or messages. Both the second and third implementations featured a web frontend, which was also designed but showed to be slow and hard to maintain (far too many convoluted java classes, some rapidly becoming obsolete).

In parallel, STAR investigated the use of AMQP [3] for its online meta-data collection [4], a sub-set of the same theme (i.e. properties only) but using a totally different set of technologies. The system was in full use in RHIC Run 13 and the system showed to support up to 2,300 messages/second. This project showed promising capabilities with its easy scalability, proxy, filtering and remote access features. The UCM idea was hence revived as a “Cloud Logger” idea with a focus on an evolution to a Grid/Cloud Logger (cloud was then a production level reality in STAR). The approach and functional design objectives were simple: (a) evaluate the possibility to create a new appender leveraging AMQP technology and replace the obsolete UCM initial approach and (b) study and better understand how to proceed with event aggregation, filtering and analysis. This paper presents the steps toward those two goals as a third evolution stage of the UCM idea.

**Figure 1.** Cloud Logger workflow overview. In its concept, a user initiates the “task” of submitting many jobs on a distributed architecture. Each job as well as edge services may broadcast either directly (via a Web-service) or a relay service (proxy) user level messages for later (near real-time) visualization of the workflow progress.
2. Application Message Logger Framework for Distributed Job Processing (Grid and Cloud)

The components of the STAR Cloud Logger are (a) a set of logging interfaces, (b) a reliable messaging bus (represented by the qpid broker [5]), (c) a schema-free database backend, and (d) a web interface to present logged information to our users in a user-centric view rather than an administrative-centric point of view. All components are new and bare no relation with the previous UCM work.

The Cloud Logger is designed around the principle of multi-tier architecture, which allows scalability and easier maintenance, a new schema-free database backend to ensure smooth upgrade in future, full support of the Grid/Cloud usage and external site logging, and an improved Ajax-based user interface, which allows to track jobs performance via automatic job statistics gathering and histograming. Fig. 1 shows the Cloud Logger workflow overview.

The upgraded logger framework keeps the valuable features of the original UCM system. The log properties structured into three categories or layers: a user task level, a job level, and job event level. From task to event, one can “drill down” to a higher cardinality of information while the structure reflects an intuitive workflow structure (a task is composed of jobs and each jobs broadcast events). This categorization directly translates to our data store collections: Tasks, Jobs and Events. Document-based storage inherent to MongoDB allows in-place updates for event messages, job status and task parameters, which significantly reduces storage space allocation overhead in a situation where many hundreds of messages come in parallel. Essentially, MongoDB pre-allocates 4MB of storage for each entry in advance, allowing to “append” new messages to a “document” before requesting a next chunk allocation.

To reduce data conversion overhead, we demand all data structures to be passed, stored and processed in JavaScript Object Notation format (or JSON), which matches our logging implementation (JSON-encoded messages over MQ and HTTPS channels). In other words, our log messages are not just plain text strings, but key:value pairs belonging to the log object produced by the framework. Such composition allows us to display process information easily via the web frontend whether the information pertains to memory consumption, CPU time or even user-defined properties. Finally, javascript application employing Asynchronous Javascript and XML (AJAX) techniques allows to fulfill the task of visualization of the Task, Job and Event data.

The STAR framework component refactoring to support the Cloud Logger was minimal and mostly focused on message transport and storage mechanism upgrades. We kept the working log4cxx [6] dynamic log flow routing layer untouched (as it worked very well), thus preserving full compatibility with existing STAR software stack (see Fig. 2). After the upgrade to the Cloud Logger, STAR jobs running on a grid or cloud may use the messaging middleware for logging purposes if an appropriate broker is installed at site boundary, or may skip it and use alternative https-based data transfer directly, without a need for a dedicated middleware infrastructure. The latter option assumes larger overhead and higher latencies compared to the messaging bus route avenue but ensures relaxed requirements on the job deployment and implied site configuration.

The web front-end features a standard three-column layout, having task, job, event columns dynamically populated upon request. It includes an intuitive click-to-expand navigation, embedded search filters, automatic histograming, and other instruments requested by collaboration members. Currently, web user authentication is implemented using standalone Kerberos-based authentication module, but we are planning to employ web Single Sign On solution (SAML2 or Shibboleth) in a near future.

3. Performance and Scalability

The performance and scalability of the Cloud Logger framework primarily depends on the following key factors: the messaging broker throughput, the web service and the database
Figure 2. Cloud Logger components overview after refactoring. The messages are either sent through a message bus or a direct Web service access. The software architecture and design specifications before the messaging are fully preserved.

backend scalability.

The messaging bus performance and scaling depends on the AMQP implementation. As noted before, STAR is using the qpid daemon, supported by Red Hat [7] corporation and Apache [8] foundation, with reported performance of about a million messages per second throughput on a tuned system, and anywhere between 10,000 and 200,000 messages/second on a typical commodity system. STAR demands on messaging layer performance are fully satisfied, as our requirements are modest: we currently run about 5,000 jobs in parallel at a single facility, which produce up to 10,000 messages/second on average, with registered peaks at about 50,000 messages/second. Thus, even a growth by a factor of x50 (10 sites, 5 times more jobs, for example) would still be fine with our currently tuned system and expanding beyond that is a matter of deploying additional services.

For the web service scaling — STAR has not yet run into web service performance issues and we utilize just one web server for this task. However, to be fully prepared to an increased load, we plan to ensure horizontal scaling by adding web servers on demand, presumably by utilizing a load-balancing front-end web router services as shown on Fig. 3 and by providing an http/https proxy in front of our primary web server.

Our database performance and scaling strategies rest on the optimized space allocation, which assumes in-place updates most of the time, and automatic data partitioning across working nodes — embedded sharding — of the MongoDB database (see Fig. 4). We hence expect a rather smooth performance and scaling on the db backend side for both writes and reads.

4. Complex Event Processing Proof of Principle in Online MetaData Filtering Context

While we did not have the possibility to test the principles of applying Complex Event Processing (or CEP) to our Cloud Logger, we could leverage our online AMQP meta-data collection (essentially a collector of “properties”) to test the CEP idea where this concept is more intuitive and needed. Online, a typical workflow can be described as follows: an event is created from various data sources, distributed to clients, then stored and/or visualized using various applications and web interfaces. A processing phase occurs: events are read from the storage element (database, memcache, ...), matched to other events from the same or different source,
analyzed by various subsystem-specific services and/or scripts and migrated to long-term storage. We could do much better by making sure event streams are processed simultaneously and be acted upon to discover event correlation, relationship and patterns. Essentially, we needed to add multi-stream, multi-event processing capabilities to the STAR message/event processing system, which lead to the study of CEP solutions and their features. While the objects to be processed are slightly more complex, the principle and the design solution as well of the study of an implementation online is similar to the Cloud Logger.

In general, CEPs operate on continuous streams of data coming from many sources and are designed to understand and manage stream relations (time based, relational syntax), handle high event rates, and feature an ability to detect patterns in data. CEPs output their own event streams as a reaction to conditionally-constrained individual event streams.

Typical CEP middleware includes an input/output data broker to handle incoming data streams (MQ, REST, WebService), an event processing engine to apply persistent queries to the data streams, a stream manager to add and remove data sources on the fly, and, finally, a query manager to configure additional queries on the fly without a need to stop event filtering process.

STAR detector’s sub-system services located in the online domain depend on other subsystem states including Trigger, DAQ and the data exported from collider run-time systems. Thus,
possible STAR applications would include unification of stream data processing, provision of the single interface for subsystem experts keeping the same three-tier schema, controlled experiment workflow including shift leader hints, experiment runtime, online tasks orchestration, alarms and more. Typical use-case example could be seen on Fig. 5.

After careful considerations and comparisons between the available products, we decided to opt for the open-source product named WSO2 CEP [9], available under the Apache License v2.0. Its list of features matched our expectations from the stream processing service, including easy integration with our existing infrastructure.

![Data Provider Layer](image)

**Figure 6.** STAR Online service infrastructure layout. Complex Event Processing component is inserted at the Message Bus level, intercepting and analyzing the events in real-time.

The WSO2 CEP consists of the following components: a CEP Core, a Broker Core, a Broker Manager. The CEP Core contains CEP Buckets which are instances of back-end CEP runtime engines (Esper [10], Fusion, Siddhi) that process events, and Data Converters for transforming events from Map, XML, and Tuple types to the back-end CEP engine’s event type. All the processing of the received events and the triggering of new events happen at the backend CEP runtime engine running inside each bucket. There are four types of brokers available to the service administrator, which are Local, WS-Event, JMS/JMSqpid and Agent. These brokers are responsible for receiving and publishing event on Thrift, SOAP, REST, and JMS transports.

As we expected, the WSO2 processor installation and integration was easy — the embedded JMS/AMQP broker instantly recognized our messaging bus server. A wide variety of the
monitoring metrics is available to help locate problematic areas during the initial integration phase, also helping to monitor the overall processing performance later on. Fig. 6 show the placement of the CEP component in the STAR online service infrastructure.

5. Summary and Outlook
We presented an upgrade of the UCM framework, currently known as the STAR Cloud Logger, a redesigned approach using modern message queuing principles. The Cloud Logger features a scalable multi-tier architecture, based on a combination of messaging infrastructure and web service approach designed to work everywhere from the intranet to a large set of Cloud or Grid distributed resources. To ensure worry-free data archival and schema evolution, we employed schema-free database backend based on MongoDB database. Web view of the data is ensured by the AJAX-based user interface, and we have discussed that our design is easily scalable.

In addition to the refactored core design, we showed that Complex Event Processing workflow, implemented in the STAR Online domain, brings new and enhanced capabilities for correlating events and therefore, empower our framework with capabilities ranging from raising alarms to warning of problems passing through executing specific actions. We achieved this by incorporating the WSO2 middleware backed by the Esper event processing engine. WSO2 Event Processor integrates smoothly with our existing Message-Queuing bus based on Red Hat qpid service, and allows us to employ stream processing techniques in online domain, including basic error detection and notification features. Such processing could also be used to detect issues in user processing tasks.

Acknowledgments
This work was supported by the Office of Nuclear Physics within the U.S. Department of Energy’s Office of Science.

References
[1] D A Alexander, C Li, J Lauret and V Fine 2008 J. Phys.: Conf. Ser. 119
[2] Small Business Innovation Research URL http://www.sbir.gov/
[3] Advanced Message Queuing Protocol [Online] URL http://www.amqp.org/
[4] D Arkhipkin, J Lauret, W Betts, G Van Buren 2012 J. Phys.: Conf. Ser. 396
[5] RHEL6 High Performance Network with MRG - MRG Messaging: Throughput & Latency (Preprint http://www.redhat.com/f/pdf/MRG-Messaging-10Gig-10Gig-IB-Xeon-v2.pdf)
[6] Apache log4cxx [Online] URL http://logging.apache.org/log4cxx/
[7] Red Hat [Online] URL http://www.redhat.com/
[8] The Apache Software Foundation [Online] URL http://www.apache.org/
[9] WSO2 Complex Event Processor [Online] URL http://wso2.com/products/complex-event-processor/
[10] ESPER: Event Stream and Complex Event Processing [Online] URL http://esper.codehaus.org/