A message-queuing framework for STAR’s online monitoring and metadata collection

D Arkhipkin, J Lauret and W Betts
Physics Department, Brookhaven National Laboratory, Upton, NY 11973-5000 USA
E-mail: arkhipkin@bnl.gov, jlauret@bnl.gov, wbetts@bnl.gov

Abstract. We report our experience on migrating STARs Online Services (Run Control System, Data Acquisition System, Slow Control System and Subsystem Monitoring) from direct read/write database accesses to a modern non-blocking message-oriented infrastructure. Based on the Advanced Messaging Queuing Protocol (AMQP) and standards, this novel approach does not specify the message data structure, allowing great flexibility in its use. After careful consideration, we chose Google Protocol Buffers as our primary (de)serialization format for structured data exchange. This migration allows us to reduce the overall system complexity and greatly improve the reliability of the metadata collection and the performance of our online services in general. We will present this new framework through its software architecture overview, providing details about our staged and non-disruptive migration process as well as details of the implementation of pluggable components to provide future improvements without compromising stability and availability of services.

1. Introduction

An acronym for the Solenoidal Tracker At RHIC (Relativistic Heavy Ion Collider), STAR tracks thousands of particles produced by ion collision, searching for signatures of a state of matter called the quark-gluon plasma (QGP), a form that is thought to have existed just after the Big Bang, at the dawn of the universe. A primary goal of STAR is to bring about a better understanding of the universe in its earliest stages, by making it possible for scientists to better understand the nature of the QGP.

The STAR collaboration consists of over 500 scientists and engineers representing 60 institutions in 12 countries. As the size of the collaboration and the scope of its work continues to grow, so does the challenge of having the computing power and data processing resources to carry out that work efficiently. To meet this challenge, a new framework was designed to enhance STAR metadata collector system, as presented in this paper.

STAR’s Data Acquisition system (DAQ), Slow Controls system and metadata collector daemons framework, collectively referred as STAR SCADA (Supervisory Control And Data Acquisition), use open-source and open-licensed EPICS framework [1]. In this work we will focus primarily on the improvement of the metadata collection system, which corresponds to about 2000 channels out of approximately 14000 being processed and monitored at STAR [2].

This paper is structured in the following way: first, we present a brief overview of the existing system and it’s limitations, then we discuss the requirements and expectations for the new metadata collection system, and, finally, we present our new MQ-based framework, performance testing results as well as concluding remarks and outlook.
Figure 1. Existing setup’s bottlenecks: 1) Communication failures between EPICS and STAR services; 2) Subsystem Monitors using Online Database directly; 3) Migration macros: no complex event processing capabilities.

2. Metadata Collectors Overview

There are many types of the metadata that STAR needs for timely and efficient physics data production. Most of that metadata comes from various STAR detector subsystems like calorimeters, tracking chambers, trigger detectors (voltages, currents, on/off status etc). Other metadata sources include collider information (beam state, various scalers), environment (temperature on a platform, humidity) and many run-time parameters such as run numbers, file locations and number of events recorded.

STAR has been in operation since 1999, with much of the online software written back in the 90’s, based on best estimates of STAR’s growth in the first decade. Now entering the second decade of operations, with greatly increased data rates and processing power, the time has come for the improvement of our online stack. In this work we describe our attempt to upgrade the online data collector system, going beyond the traditional Data Archiver as it is known in SCADA world.

The STAR software infrastructure utilizes a typical data warehousing schema, consisting of an operational database layer, data access, metadata layers and an informational access layer. Data Collectors (DCs) belong to the operational database layer, the front line of the detector data acquisition process. At the moment, we collect data from about 2000 EPICS-based channels in a “pull/push mode”, with read frequencies varying from “as fast as possible” to once per 5 minutes, with a prospect of reaching 5000 channels two years from now. This data is not only stored, but also complex real-time transformations get performed on it.

At the heart of our online data processing reside MySQL databases, with MyISAM tables as storage engine, which serve as a run-time and long-term storage at the same time. Users access MySQL directly, with no intermediate layer or API involved. While it is possible to achieve fairly good horizontal scaling with MySQL by adding more slave servers for read-only operations, the whole system is seriously constrained by using this database-centric design when it comes to having many users writing to the database simultaneously. The multi-master MySQL replication setup does not help to balance a system load for writes, and the MySQL Cluster Edition [3] provides the RAM-only data storage, not suitable for our task.

Our concerns about the existing setup include but are not limited to the strong coupling to the MySQL API, the lack of fault-tolerance, only limited load balancing is possible, and the data flow is strictly synchronous. Also, STAR will benefit from switching to publish/subscribe mode
of operations (reduces network traffic by a large factor, compared to constant polling mode of operations) and some way of “pluggable” data storage mechanism. Fig. 1 illustrates some of the existing bottlenecks in our system. The next section summarizes our ideas and expectations on how to deal with the aforementioned concerns.

3. Proposed Framework: Requirements and Expectations
Requirements for the new system include a standard API for all clients (unified, stable interface to access database), with minimal external library dependencies, and asynchronous data storage access for writes. Since we work in a near real-time mode, non-blocking IO is desired. We need a solution in line with “set it and forget it” ideal. This implies reliable data storage access without sacrificing performance. In addition, we need automatic load balancing and fail-over for both reads and writes, and a simple way to configure all clients, due to vast array of technologies used to process the data.

While we were primarily looking at an Online API, we thought that switching from a database-centric model to message-queuing (MQ) bus services will allow us to meet these requirements. Most MQ implementations enforce usage of a predefined message-level data format, or are available for specific languages or platforms only - this is not suitable for our task. After a careful study of all available solutions, we settled on the Advanced Message Queuing Protocol (AMQP), having an advantage of being interoperable and technology agnostic - it does not constrain us to a specific data format or programming language. Since our infrastructure is built primarily upon Scientific Linux, we decided to use the qpid server (implementation of AMQP 0.10 standard), developed and supported by Red Hat [4]. AMQP treats all messages as binary chunks, therefore we can choose absolutely any message serialization standard. Many different solutions for structured data serialization exist on the market, so in our search we focused on three things: fast (de)serialization speed, support for many programming languages and schema evolution. Google Protocol Buffers (protobuf) satisfied all our requirements, so we have built our framework using this library. Vendor’s performance report quotes the throughput rates of 547k messages per second for qpid (tuned 1-GigE network, 32-byte transfers, see [5] for details), and Google Protobuf’s (de)serialize speed could be as high as 300k objects per second per node (typical setup has several publisher nodes, so it is not considered a limit for our system).

Figure 2. Existing setup: no middleware layer, only Channel Access communications.

Figure 3. Proposed setup: using AMQP as middleware layer. Cloud icons indicate distributed capabilities.
4. Framework Implementation
To achieve the desired goals, we have defined data exchange structures in the protobuf format, and introduced three simple building blocks: EPICS to MQ service, MQ to database service, database to MQ service, named epics2mq, mq2db, db2mq. Fig. 2 illustrates the historical setup, and Fig. 3 shows the upgraded version.

epics2mq is a daemon which polls EPICS periodically, serializes results using Protocol Buffers, and sends messages to the AMQP server. EPICS and AMQP server parameters are read from a configuration file, as well as the EPICS channel names to poll. There is no need to write a single line of code to add more channels to the data collector system, just run another epics2mq instance with separate configuration file listing desired EPICS channels. As a bonus, epics2mq code is quite simple, which makes code quality assurance checking a breeze. Our future plans include a counterpart to this component named mq2epics, which will allow to propagate commands from MQ back to EPICS system - required component for remote control capabilities.

mq2db is a daemon, which receives messages published by epics2mq, de-serializes the results (Protocol Buffers), and stores data to the backend database via an abstract interface. There’s no need to modify mq2db daemon to store extra data channels, because it automatically converts “path” and data into database/table/record form for all messages arriving over AMQP (it will create new databases and tables automatically). Also, one can have two or more completely independent, parallelized data archivers using various databases as storage backends by running another instance of mq2db in parallel to the primary archiver.

db2mq is a daemon, which awaits client data requests over AMQP, fetches data from a configured database backend and, finally, sends data to clients via AMQP.

By introducing message-queuing and an interoperable serialization format we achieved the following:

• loose coupling: only protobuf structure descriptions are shared between all services;
• synchronous and/or asynchronous data transfer support for clients;
• database polling is completely eliminated, and we can apply fine-grained load balancing techniques as we see fit;
• ability to use any storage engine we like for speed, maintenance, or other reasons;
• easy extension/expansion of any component, better scalability;
• messages could be routed to external facilities - Remote Control Rooms or monitors (see Fig. 4);
• clients rely on clustered AMQP broker for reliable message transfer - no need to deal with db-specific error handling all over client codes;
• we can simultaneously utilize any database backend suitable for a task: it could be well-known, maintainable MySQL with tiered SAS/SSD/RAM storage, or, fast, RAM-based MonetDB, or distributed, scalable round-robin database like MongoDB, or write-tolerant NoSQL Cassandra database etc;
• clients may choose between request/response or publish/subscribe mechanisms, adding flexibility to the system.

5. Performance Testing and Deployment
With the most of the components ready we performed basic qpid performance testing using perftest. The results are encouraging: our 5-years old hardware was capable to process
approximately 50,000 messages per second, with no system parameter tuning involved. We fed the MQ server with simulated data from 120 live EPICS channels, exported to AMQP in the following groups: 20 integers, 40 floats, 40 doubles and 20 strings of 16 chars each. Next, the same test was repeated for 1-128 publishers (Data Sources) and one subscriber (Data Archiver). For 128 publishers, overall message processing rate observed was less than five percent smaller than the rate we have got for a single publisher.

The second test was to see if this data processing rate is sustainable for a long time. The same test was run for two weeks continuously with no single failure or delay in data transmission encountered, and the test server load was reported to be less than 0.5 for the whole period of observations.

Based on the test results, we decided to deploy a functional, MQ-based setup for the upcoming run in 2011 in parallel to the existing system. As one can see in Fig. 5, it includes two clustered instances of qpid, several data sources attached to EPICS channels via Channel Access, separate MySQL-based Data Archiver, and real-time web-based monitoring GUI.

6. Summary and Outlook

In this work we presented a message-oriented framework, designed and implemented for the STAR Online / DAQ domain. Since our primary goal is to employ a vendor-independent message-queuing system as a system bus, we developed an AMQP-based framework, which will allow us to simplify online metadata collection and maintenance, ensure easy expansion and scalability, and improve metadata collection performance and reliability for the upcoming run.

Preliminary performance tests indicate that, overall, system performance will be at least two orders of magnitude better compared to the EPICS-only setup. In addition, MQ-based setup allows easy real-time browser-based data flow monitoring, composed of widely available technologies like Web Sockets and AJAX, because of the possibility to talk to the messaging system directly from web browser, with no intermediate services involved (see [6] for example) - not achievable with our existing system. Having an MQ-based system will allow STAR to integrate dedicated stream-processing software into our SCADA, thus allowing more complex stream transformations to be implemented on demand in a scalable way. Future plans to extend our framework may include Control System Studio (see [7]) evaluation and integration, as well as extended hardware control capabilities up to device driver level.

Acknowledgments

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

References

[1] EPICS - Experimental Physics and Industrial Control System [Online] URL http://www.aps.anl.gov/epics/
[2] D Reichhold et al 2003 Nucl. Instrum. Meth. A 449 792
[3] MySQL Cluster home page [Online] URL http://www.mysql.com/products/cluster/
[4] Red Hat [Online] URL http://www.redhat.com/
[5] RHEL6 High Performance Network with MRG - MRG Messaging: Throughput & Latency (Preprint http://www.redhat.com/f/pdf/MRG-Messaging-1Gig-10Gig-IB-Xeon-v2.pdf)
[6] Kamaloka-js: AMQP bindings for native JavaScript [Online] URL https://fedorahosted.org/kamaloka-ja/
[7] Control System Studio [Online] URL http://cs-studio.sourceforge.net/