INSTRUMENTATION OF THE CERN ACCELERATOR LOGGING SERVICE: ENSURING PERFORMANCE, SCALABILITY, MAINTENANCE AND DIAGNOSTICS

C. Roderick, R. Billen, D. Dinis Teixeira, CERN, Geneva, Switzerland

Abstract

The CERN accelerator Logging Service currently holds more than 90 terabytes of data online, and processes approximately 450 gigabytes per day, via hundreds of data loading processes and data extraction requests. This service is mission-critical for day-to-day operations, especially with respect to the tracking of live data from the LHC beam and equipment.

In order to effectively manage any service, the service provider's goals should include knowing how the underlying systems are being used, in terms of: "Who is doing what, from where, using which applications and methods, and how long each action takes".

Armed with such information, it is then possible to: analyse and tune system performance over time; plan for scalability ahead of time; assess the impact of maintenance operations and infrastructure upgrades; diagnose past, on-going, or re-occurring problems.

The Logging Service is based on Oracle DBMS and Application Servers, and Java technology, and is comprised of several layered and multi-tiered systems. These systems have all been heavily instrumented to capture data about system usage, using technologies such as JMX.

The success of the Logging Service and its proven ability to cope with ever growing demands can be directly linked to the instrumentation in place.

This paper describes the instrumentation that has been developed, and demonstrates how the instrumentation data is used to achieve the goals outlined above.

INTRODUCTION

Born out of the LHC Logging project, and operational since 2004, the CERN accelerator Logging Service (herein referred to simply as the "LS") is used to store and retrieve billions of data acquisitions per day, from across the complete CERN accelerator complex, related sub-systems, and experiments [1].

The LS is considered a mission critical service, heavily relied upon to support day-to-day operation. As such the availability and performance of this service are paramount.

ARCHITECTURE OVERVIEW

Figure 1 shows a basic overview of the LS which is essentially comprised of:

- Two Oracle databases: A so-called Measurement database (MDB) where raw data from Java processes and other Oracle databases is persisted during seven days, and a Logging database (LDB) where a sub-set of MDB data and pre-filtered data from industrial SCADA systems are stored on-line indefinitely.
- Distributed Java APIs are responsible for loading data into the databases.
- A sub-set of MDB data is transferred to the LDB using in-house developed PL/SQL code that uses a comprehensive set of metadata to dynamically filter the data of long-term interest.
- A powerful distributed Java API is the sole means of extracting data from the databases, which includes a command line interface. Applications wishing to use the API must be pre-registered. At the time of writing there are 100 applications registered to a heterogeneous client community. Direct SQL access is not permitted.
- A generic Java GUI called TIMBER is also provided as a means to visualize and extract logged data. The tool is heavily used, with more than five hundred registered users.

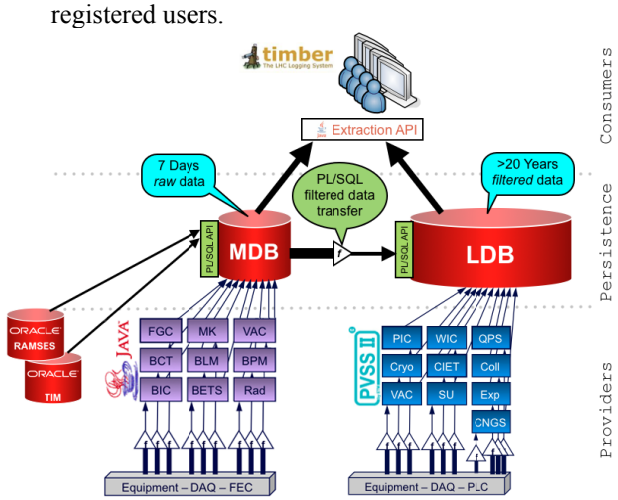

Figure 1: Logging Service architecture overview.

The Java APIs for both logging and extracting data are significantly optimized, and run on Oracle application servers. For data extraction clients, the fact that database access is actually made in a distributed manner via an application server is hidden within the Java API.

PERFORMANCE, SCALABILITY, STABILITY … AND USERS

The LS is a high performance service, which needs to deal with very high data throughputs. At the time of writing the MDB has to process approximately 5.4 billion records/day, which equates to around 270GB/day (annual throughput of approximately 100TB). Meanwhile the LDB needs to persist around 4 billion records/day for

some 850 thousand signals. This boils down to storing 140GB per day (50TB/year) and keeping it available online beyond the lifetime of the LHC.

The success of the LS leading to an increase in scope beyond the LHC, together with unforeseen events requiring more data to be available, has meant that the current data throughput levels far exceed initial expectations, which predicted 1TB/year during LHC operation.

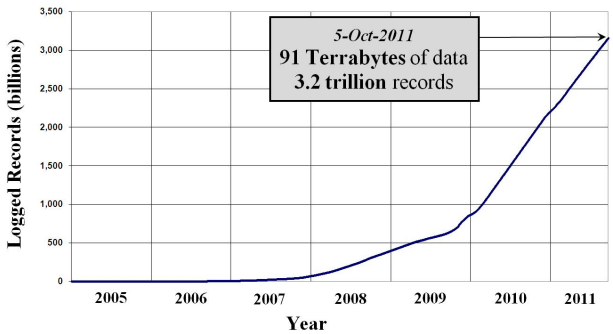

Figure 2: Evolution of logged data.

Figure 2 shows the evolution of logged data, and clearly illustrates how the LS has had to scale to satisfy evolving requirements. The ability to scale in such a manner is in no small part down to the design of the LS [2], however instrumentation also plays an important role, as will be explained later in this paper.

The amount of data logged only tells one side of the story, since data is actually logged in order to be extracted later on to support operational decisions, which often have to be made within short time constraints; therefore data extraction must be as fast as possible.

It is perfectly legitimate for users to ask for data spanning long time periods and/or from long ago. Therefore the LS must satisfy such diverse requests not only as quickly as possible, but also whilst remaining stable such that is can support other operations in parallel.

The determining factor in how a service performs is *always* how the service is used. Experience has shown that there is often a big difference between how service providers *think* the service will be used, how users *claim* they will use the service, and how users *actually* use the service. This is where instrumentation comes in…

WHAT IS INSTRUMENTATION?

In this paper, instrumentation refers to capturing information about service activity in real time, and over time, in order to know who is doing what, from where, how things are being done, and how long various actions take.

Who?

This should always indicate the *real* end-user of the service – somebody who can be contacted. In other words, in an n-tier environment, it should not just be the directly connected OS user on one of the tiers.

What?

In its simplest form, this could be the name of a method / function / procedure etc. A more comprehensive solution would also capture details of all of the dimensions that can affect the outcome of an action and/or the performance of the service. These details are domain specific, but an example from the LS when querying data would be: the API method, the names of the signals concerned, the time window, and any additional data manipulation parameters (see Figure 4).

Where?

This should be a host name or IP address, and process id, which can be used to physically locate calls being made to the service.

How?

This means identifying which application (by name) is using the service, and if the service is accessed via libraries – which versions of the libraries are being used.

How Long?

Knowing the amount of time spent doing something is an essential ingredient in understanding how a service is performing, and why problems may have occurred. Therefore it is necessary to capture the elapsed time for each significant action executed within the service.

Besides these key elements, it is also important to instrument if actions finish successfully or throw exceptions in order to understand unexpected behaviour.

WHY INSTRUMENT?

Instrumentation is often considered an unnecessary overhead, especially by developers who want their code to run as fast as possible. However, this is a rather shortsighted view on things.

Knowing the answers to the questions above enables service providers to understand how a service is really being used (or misused), and how it is performing in terms of both throughput and response times. In turn, this allows to pre-empt problems, identify potential bottlenecks, plan system upgrades, and when issues inevitably occur – diagnose and react swiftly and effectively.

Collectively, these benefits far outweigh any perceived run-time overhead of having instrumentation in place.

The rest of this paper will focus on particular examples of instrumentation deployed in the LS, and how it has helped meet the requirements for performance, scalability, and stability.

DATA LOADING

Every day, the LS treats millions of data loading requests, coming from hundreds of client processes. The distribution of these requests across clients is heavily skewed. For example, one client may be responsible for

sending up to 40% of the data, and other just 1%. In order to know how the systems are being used, it is important to capture these data distributions.

Likewise, the data distribution within data loading requests may be heavily skewed across clients, or over time. In other words a fixed size data loading request may contain a lot of data for a few signals, or a small amount of data for many signals. This distribution can have a significant impact on performance, and therefore also needs to be captured to support performance analysis.

The other factor impacting performance of data loading in the LS is whether or not a request contains duplicate data (same timestamp received for the same signal), which requires a special treatment taking 4 times longer to process than a request without duplicate data.

Initial Implementation

The instrumentation in the LS has evolved significantly over many years. Initially most of the above details were just captured in log files. The problem with this approach is that they were very difficult to analyze, especially as the parallel load on the LS began to increase, and non-related log entries become more and more interleaved.

To understand the data distribution across clients, an internal database job ran queries against the logged data, making aggregates of the amount of data received per client. This approach was not scalable, and as the data rates increased the aggregate queries were continually adapted to use increasingly smaller sample periods of data. A new approach to instrumentation was required.

Evolution

A well-structured instrumentation framework was developed and put in place at the level of the data loading API running on the Oracle application servers. This framework captures all details of all data loading requests, performs on-the-fly in-memory data aggregations, and writes the results into the database on a daily basis.

This approach is extremely accurate (since aggregates are based on actual data rather than data samples), and avoids the need to use significant database resources to estimate system usage. In addition the time spent on each action (parse, check, prepare, load) within each data loading request is captured and aggregated to facilitate analysis of system performance, and identify bottlenecks and bad clients.

The other major advantage with the instrumentation framework is that all information is well structured and exposed via JMX using Java *managed beans* (MBeans), which can be consulted in real-time via any JMX (Java Management Extensions) interface. This allows service providers to easily see what the systems are currently doing, and diagnose on-going problems.

INTERNAL DATA TRANSFER

The majority of the data logged in the MDB are candidates to be transferred to the LDB for long-term storage. What data actually gets transferred is governed by a comprehensive set of metadata defining things such as deltas, smoothing, fixed logging, precision etc. for each of the defined signals. The act of applying the metadata to the raw data, and filtering and transferring the results to the LDB are carried out using in-house developed PL/SQL code which is executed in parallel by 8 internal database jobs running every 5 minutes. The signals whose data is treated by 1 of the 8 jobs are distributed across the jobs according to a predefined category for each of the signals.

Knowing how each execution of the data filtering and transfer jobs performs, in terms of number of signals, number of candidate values per data type, number of logged values per data type, and times taken for each internal action is essential.

Data Capture & Diagnostics

The PL/SQL data filtering and transfer code captures all of the above information in memory, and writes the results into dedicated database tables after each execution.

This detailed information remains available for 7 days (lifetime of MDB data) and is extremely useful for diagnosing performance problems – identifying if long executions times are isolated to particular groups of data, specific data types, certain times of the day or a specific type of action (such as data collection and filtering in the MDB, or data transfer to the LDB).

The detailed information is also aggregated on an hourly basis (Figure 3), and results are stored long-term in the LDB. This aggregate data helps identify trends in system performance such as correlations with accelerator performance, or gradual performance decreases as demands on the system increase (e.g. requests to log data for more signals and / or at higher frequencies.

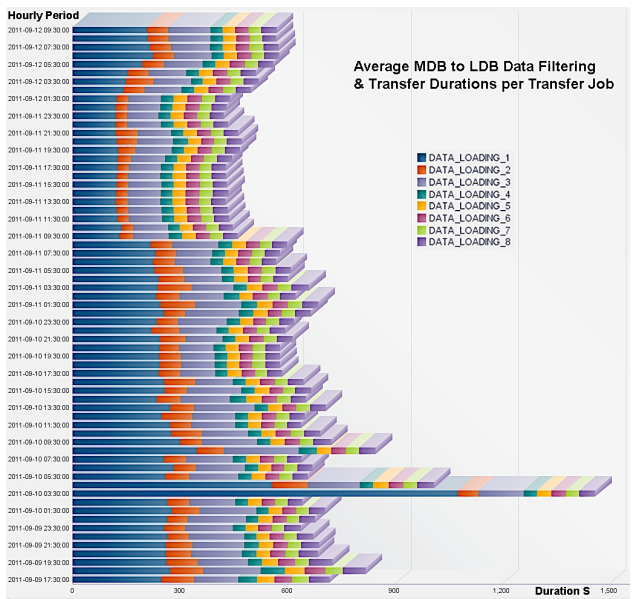

Figure 3: Example MDB to LDB instrumentation data.

DATA EXTRACTION

With up to 2 million requests per day to extract data for one or more signals over greatly varying time periods – data extraction from the LS represents a significant portion of overall activity.

Data rates vary significantly from one signal to another, and from one time period to another (e.g. according to whether or not there is beam present in the LHC).

In such an environment, users are often unaware of the amount of data that they are implicitly requesting, or of the best methods to use to extract with.

Aiming for Service Stability

As part of an attempt to assure service stability, every request to extract data is transparently instrumented, exposed, and logged using a framework similar to that for data loading, based on JMX. For each user: the running, last added, last finished, and last unsuccessful requests are always accessible via any JMX console. The *Who*, *What*, *Where*, *How*, and *How Long* information is embedded in each request, including signals involved, the extraction time window, invoked method, elapsed time, library versions, and the result (see Figure 4).

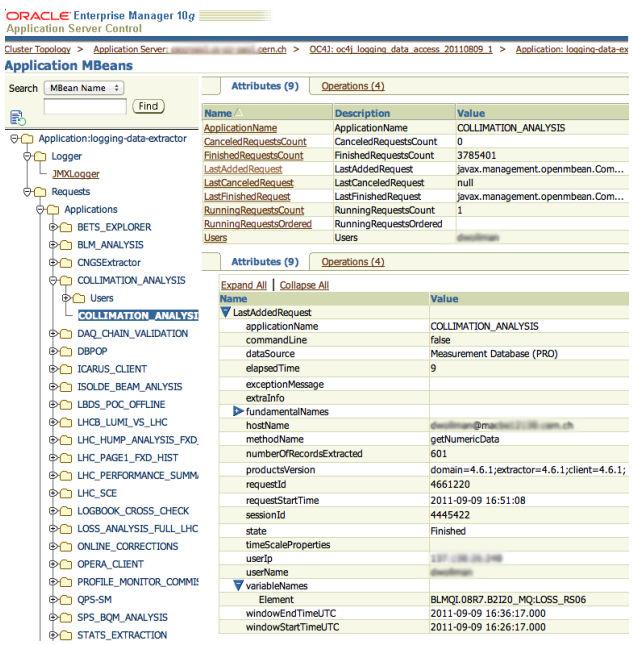

Figure 4: Data extraction instrumentation via JMX.

The ease of access of this information greatly facilitates following up support requests, since service providers can quickly access the full set of details of what the user is trying to do. Furthermore, because this information is accessible in real-time, a JMX agent connects every 5 seconds to the remote data extraction server, assesses the current situation, and can take various actions:

- If a request has been running for too long, a warning is first sent to service administrators, and if the situation continues – the request will be terminated. In such situations, it is common practice for the service administrators to diagnose the problem and pro-actively contact the user. More often than not – the users just need to be advised about which alternative methods to use or attribute values to apply.
- If any centralized data extraction server fails, service administrators are notified of the failure, together with details of all requests running prior to the failure, such that they can diagnose the cause, inform the user responsible, and adapt the service to be more resilient in the future.

Another way in which the captured data is used is related to backwards compatibility during upgrades to the API. Because all method calls are logged, it is possible to deduce whether or not certain users will be affected by necessary API changes, and contact them in order to adapt their code, or delay the changes.

SUMMARY

Instrumentation should not be considered as an overhead, but rather as an integral component of any software infrastructure. Once in place it quickly becomes part of the backbone of the system, allowing service providers to quickly and confidently diagnose problems, tune system performance, and plan upgrades.

The Logging Service instrumentation data is constantly used to support users, and has helped unravel otherwise impossible to diagnose problems in a complex and distributed environment.

The Logging Service is a stable, high performance, and heavily used service. The performance, proven ability to scale, and overall stability are testament to the value of the significant instrumentation in place.

REFERENCES

[1] C. Roderick and R. Billen, “Capturing, Storing and Using Time-Series Data for the World’s Largest Scientific Instrument”, November 2006, CERN-AB-Note-2006-046 (CO).

[2] C. Roderick et al., “The LHC Logging Service: Handling Terabytes of On-line Data”, ICALEPCS’09, Kobe, Japan, October 2009, WEP005.