ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA117

BIG DATA ARCHIVING FROM ORACLE TO HADOOP

I. Prieto Barreiro, M. W. Sobieszek. CERN, Geneva, Switzerland.

Abstract

The CERN Accelerator Logging Service (CALS) is used to persist data of around 2 million predefined signals coming from heterogeneous sources such as the electricity infrastructure, industrial controls like cryogenics and vacuum, or beam related data. This old Oracle based logging system will be phased out at the end of the LHC's Long Shut-down 2 (LS2) and will be replaced by the Next CERN Accelerator Logging Service (NXCALS) which is based on Hadoop. As a consequence, the different data sources must be adapted to persist the data in the new logging system. This paper describes the solution implemented to archive into NXCALS the data produced by QPS (Quench Protection System) and SCADAR (Supervisory Control And Data Acquisition Relational database) systems, which generate a total of around 175,000 values per second. To cope with such a volume of data the new service has to be extremely robust, scalable and fail-safe with guaranteed data delivery and no data loss. The paper also explains how to recover from different failure scenarios like e.g. network disruption and how to manage and monitor this highly distributed service.

INTRODUCTION

CALS is used to persist and retrieve billions of data acquisitions per day and is considered a mission critical service. The data is coming from heterogeneous sources such as the CERN accelerator complex, related subsystems and experiments [1]. The logging system was initially designed for a throughput of 1 TB/year and it is currently operating far over its design limits, storing over 1.2 TB/day. The underlying storage of CALS is based on Oracle Database and uses an old monolithic architecture which is difficult to scale up [2]. CALS will be phased out at the end of the LHC's Long Shut-down 2 (LS2) in 2020 and will be replaced by the Next CERN Accelerator Logging Service [3] (NXCALS) which is based on Hadoop.

The replacement of the logging system will have an impact on many client applications which use a database to database transmission mechanism to archive the data in the logging system. This is the case for QPS (Quench Protection System) and SCADAR (Supervisory Control And Data Acquisition Relational database) [4], which are first storing their data in a temporary database and then transferring a sub-set of the data to CALS for permanent storage using PL/SQL. As NXCALS is based on Hadoop, this approach of data transmission is technically not possible and therefore it was necessary to implement a new mechanism to feed the new logging system with the data.

This paper describes the solution implemented to archive in NXCALS the data produced by QPS and SCADAR. The client systems of NXCALS must register the signals before archiving their data values, and these two systems have registered over 1.6 million signals in the logging system which generate a total of around 175,000 values per second. Therefore, the implemented solution must cope with big data volumes and it was designed to be extremely robust, scalable and fail-safe with guaranteed data delivery and no data loss. The paper also explains how to recover from different failure scenarios like e.g. network disruption and how to manage and monitor this highly distributed service.

SYSTEM OVERVIEW

The tasks performed by the new data source for NXCALS, named WinCC OA Data Source (WCCDS), are conceptually simple:

1. Get the list of signals registered in the logging system.
2. For each registered signal, get the list of values not yet transmitted to the logging system and send them to NXCALS.
3. Wait for the reception acknowledge of the transmitted values and mark them as logged.
4. Allow data re-transmissions on user demand.
5. Manage the registration, removal and modification of the signals in the logging system.

The implementation of the WCCDS service was split into two main processes: Datasource process for the first four aforementioned tasks and Metadata process for the last task. Figure 1 shows the system overview; the different QPS and SCADAR applications store their data and metadata in two different database schemas. The data schema contains the values and timestamps produced by all the signals of the system. The metadata schema keeps track of the signals registered in the logging system and contains additional information, like the name of the table where the signal values are stored or the timestamp of the last values archived in NXCALS for each signal. The WCCDS service queries both schemas and transmits the data and metadata to NXCALS. When the service receives the reception acknowledge for the transmitted data, it marks the data as transferred to NXCALS.

Data Partitioning and Scaling

The volume of data to be transmitted to the logging system is too big to be handled by a single JVM (Java Virtual Machine) process and the WCCDS service profits from the data partitioning defined by CALS, where a signal belongs to a data category and a data category belongs to a data transfer group. For example, QPS has over 135,000 signals registered in the logging system which are partitioned into 53 data categories. The data categories are spread over 20 data transfer groups.

This data partitioning allows a simple distribution of the data transmission among different JVM processes and machines depending on the capacity of the available resources.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA117

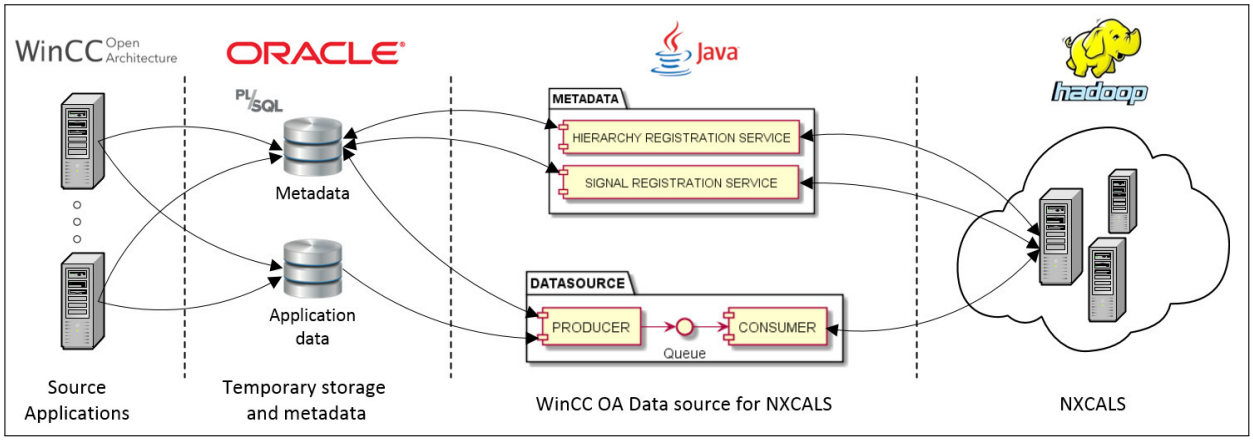

Figure 1: System overview.

After performing stress tests, performance tests and process tunning for QPS, it was established that each datasource process will manage two transfer groups and each machine will run five datasource processes. If the volume of data grows in the future, it will be possible to modify the distribution by i.e. having a single transfer group per process or even to modify the strategy to distribute the data categories among different processes. The whole system can be easily scaled up by distributing the different datasource processes on additional machines.

DATASOURCE PROCESS

The datasource process is in charge of getting the signal values not yet transmitted to the logging system, send them to NXCALS, wait for the reception acknowledge and mark them as sent in the database. The process is divided into two logical modules, data producer and data consumer, linked by a communication channel (Figure 1). The default implementation of the channel is based on a Concurrent Blocking Queue.

Data Producer

One of the difficulties of the data producer resides in how to optimize the queries to the temporary database containing the application data in order to have a low impact on its performance.

The values of the signals belonging to a transfer group can be stored in different tables, therefore, the first optimization implemented was to group the queries for the signal values by table. The second optimization was to query the data in configurable time windows defaulting to 300 seconds. Finally, the last optimization was to introduce a timestamp aligner algorithm for grouping multiple signals in the queries used to obtain the values to transmit into the logging system. Without such an algorithm, the queries to obtain the values would be executed individually for each signal, having a negative impact in the database performance. For example, Table 1 shows the metadata table containing the timestamp of the last logged values for five different signals and their aligned timestamps. Using the aligned timestamp, it is possible to execute only two queries to obtain the new values for the five signals. For example, the first three signals will be grouped by their aligned timestamp in the query:

```
select * from data
  join metadata on
    metadata.signal_id = data.signal_id
  where
    metadata.aligned_stamp = '09:20:30'
    and data.timestamp > '09:20:30'
    and data.timestamp <= '09:25:30';
```

The only inconvenience of the timestamp aligner approach is that some values will be sent twice to the logging system, which is not an issue for NXCALS as such data will be deduplicated by the system. However, it improves greatly the performance on the queries to the database.

Table 1: Example of aligned timestamps for several signals

Signal ID	Timestamp Last Logged Value	Aligned Timestamp
1	09:20:45	09:20:30
2	09:20:49	09:20:30
3	09:20:37	09:20:30
4	09:20:07	09:20:00
5	09:20:11	09:20:00

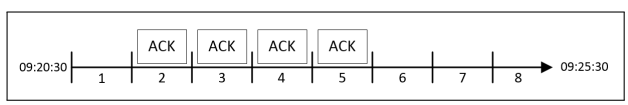

Figure 2: Acknowledge of time window batches.

When the data of a time window is obtained from the database, it is split into multiple data batches and sent to the communication channel in order to be processed by the data consumer. The last task to be performed by the data producer is to wait for the reception acknowledge of the data

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA117

batches and update the timestamps in the database accordingly. The acknowledge of the data batches can be received in any order, therefore, the data producer waits until it receives the acknowledge of a consecutive block of batches from the beginning of the time window before updating the timestamps. Figure 2 shows a time window of five minutes split into eight data batches with the acknowledge received for the batches 2 to 5. If the next acknowledge received corresponds to the first data batch, the data producer will update the last logged and aligned timestamps in the database to include the values sent in the first five batches. The entire communication, described above, has asynchronous nature therefore its performance is very good.

Data Consumer

The data consumer digests the data batches and sends the individual signal values to NXCALS. Then it waits asynchronously for their reception acknowledge. When the confirmation is received for all the values of a batch, it sends the batch reception acknowledge to the data producer.

If the publication of a value to NXCALS throws an exception, the data consumer will check whether it is a recoverable error in which case it will try to re-publish the same data. If the exception is not recoverable, for example, due to a bad data formatting, the exception will be propagated to the data producer and the complete batch will be invalidated, storing the error in the database and locking the data to avoid its re-transmission. The system monitoring and alerting, explained in a subsequent section, will detect the locked data and send an alert to the service administrators for a further investigation of the problem.

Blocking Queue

The communication channel between the data producer and consumer was implemented using a blocking queue. If the producer is faster than the consumer processing the data, the queue will be filled and the process will be blocked until the consumer removes items from it. This approach avoids having out of memory errors due to the increasing heap space used by the application in the case of having slow consumers.

Even if the process is blocked temporarily, there is no risk of data loss because the metadata tables contain the timestamp of the last values archived in the logging system. As soon as the process resumes its normal operation, it will start processing the data sequentially from the oldest values not yet transmitted.

Data Re-Transmission

In certain situations, the administrators of the systems generating the data might request a data re-transmission to the logging system. For example, a typical use case is to add a new signal in the logging system and archive its values from a number of hours in the past.

Since the values of all the signals are stored in the temporary database, the data re-transmission can be triggered just by re-setting the aligned timestamp of the required signals in the metadata table. When the data producer starts processing data for the signals, it will get the values to transmit starting from the time specified in the aligned timestamp.

Network Disruption Events

In the case of a network disruption event, the data flow between the different system layers might be affected (Figure 1). If the source applications cannot send their values to the Oracle temporary storage, they will buffer the data in the local file system until the connection is restored. At that point, the applications will resume the data transmission sending first the oldest values stored in the buffers. This sequential data transmission will guarantee that there are no gaps in the Oracle database for the application data and all the values will be processed by the datasource.

If the disruption event affects the communication between the Oracle temporary storage and the datasource process, the data producer will not be able to obtain any new data. As soon as the connection is restored, the data producer will see the new data and it will resume its normal operation.

Finally, the network issue might affect the communication between the datasource and NXCALS. In this scenario, the data consumer will try to send the batch data and wait asynchronously for the reception acknowledge. Since the acknowledgement will not be received until the network disruption is resolved, the confirmation of the batch reception will not be sent to the data producer and the data transmission for the signals will be temporarily blocked until the communications are restored.

METADATA PROCESS

The metadata process manages the registration, removal and modification of the signals in the logging system and their organization in a hierarchical, tree-like structure, which allows a better navigation and categorization of the signals.

The process checks regularly the metadata tables looking for new signals to register, remove or modify and propagates the changes to the logging system.

An additional task of the metadata process is to look for inconsistencies between the signals registered in CALS and NXCALS. This feature will be helpful during the initial deployment of the WCCDS service to ensure that the signals registered in CALS are also registered in NXCALS. The number of inconsistent signals between both systems is exposed by JMX (Java Management Extensions) and is available for the system monitoring and alerting.

TESTING

The testing of the WCCDS service was split into functional and stress tests. The functional tests were performed using a mock application and they aimed to verify the correct implementation of the service. The stress tests were performed with live data coming from the QPS production system which is producing around 150,000 values per second. The objectives of the stress tests were the following:

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA117

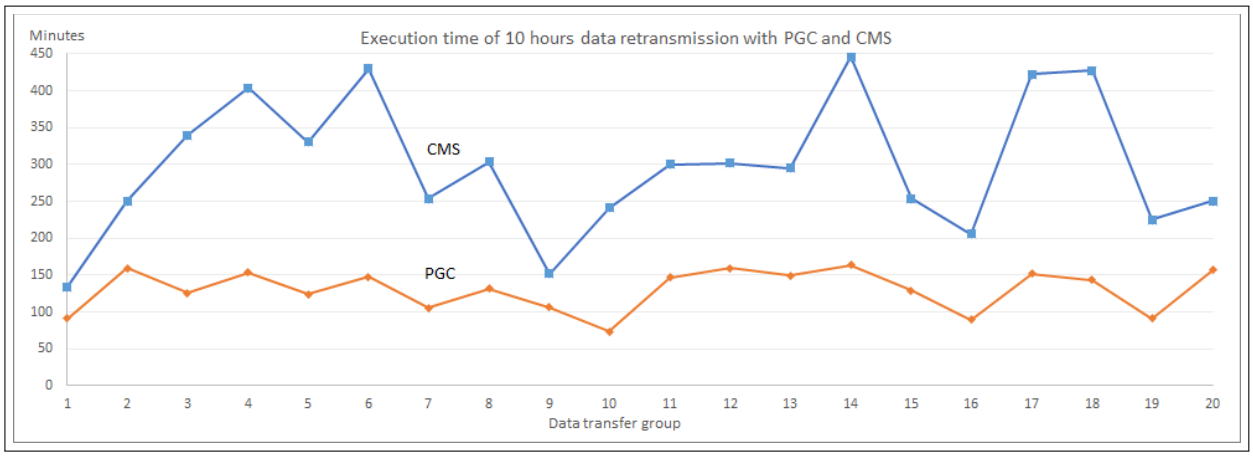

Figure 3: Performance comparison between PGC and CMS garbage collectors.

- Measuring the data transfer rates for each of the 20 data transfer groups.
- Grouping the data transfer groups in order to have a similar transfer rate for each process.
- Optimizing the number of data transfer groups to be included in each datasource process in order to obtain the best possible performance.
- Tuning the the number of concurrent executors used in the different modules of the datasource process.
- Testing the data re-transmission up to 12 hours in the past.
- Estimating the number of machines required to execute the service.

The initial stress tests were performed using a pool of 20 virtual machines, each of them executing a single datasource process handling a unique data transfer group. An additional virtual machine was used to execute the metadata process. The first round of tests allowed the measuring of the data transfer rates for the transfer groups and the initial tuning of the datasource parameters to optimize its performance.

The second round of tests was performed using two physical machines, each of them having 128GB of RAM and two CPUs of type Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, having a total of 40 CPU virtual threads. The results obtained from the tests demonstrated that the optimal configuration for the service was to include two data transfer groups per datasource process with a distribution of five datasource processes per machine. This configuration has a relatively low usage of the hardware resources during normal operation but allows data re-transmissions up to 12 hours in the past without having a big impact on the normal operation of the service. The JVM processes were running under Java HotSpot 64-Bit Server VM (build 1.8.0_152-b16).

Java Garbage Collector Issues

During the stress tests of the system several kind of issues occurred; in some cases it crashed the execution of the datasource process and in other cases it threw very strange exceptions while processing the data. The issues could not be reproduced systematically and they appeared randomly in the different datasource processes. After analyzing the issues in detail, it turned out to be a bug in the Java's Garbage-First Collector (G1).

Therefore, the subsequent tests aimed to compare the performance and stability of the garbage collectors PGC (Parallel Garbage Collector) and CMS (Concurrent Mark Sweep). Figure 3 shows the execution time of data re-transmission of 10 hours for the 20 transfer groups using PGC and CMS garbage collectors. The data re-transmission took between 70 and 160 minutes when PGC was used and between 130 and 445 minutes for CMS. In the comparison between the data re-transmissions of each transfer group, the processes executed with CMS was between 1.4 and 3.2 times slower than PGC. Therefore, PGC was selected as the garbage collector for the execution of the processes.

The modification of the garbage collector completely removed the instability of the system and the datasource processes were running for over a month without further interruptions.

SYSTEM MONITORING AND ALERTING

The status of the different processes and the hosts is constantly monitored by the Prometheus [5] ecosystem (Figure 4).

The WCCDS processes expose relevant metrics for monitoring their current state via JMX. The datasource process exposes the number of metadata groups being processed and how many are locked due to an exception, the number of errors reading from the database and publishing to NXCALS, and the number of records read from the database. The metadata process exposes the number of signals which are registered in CALS and not registered in NXCALS.

Prometheus gathers the metrics exposed by the WCCDS processes along with the host metrics which are exposed by Prometheus' node exporter, like CPU, memory or network usage. They can be then displayed in Grafana [6] in a form of custom charts and dashboards. Prometheus also compares the metric values against a set of customized alerts defined for monitoring the healthy state of the processes and the hosts.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA117

For example, if the data processing of a transfer group is active for more than a certain amount of time, if the number of signals locked due to an exception is greater than 0 or if the CPU usage of a host is higher than 75% for more than a given threshold, an alarm will be triggered. The alert manager process will receive the alarms, group them by type and send notifications by email or sms to the relevant receivers, depending on the type of the alarm.

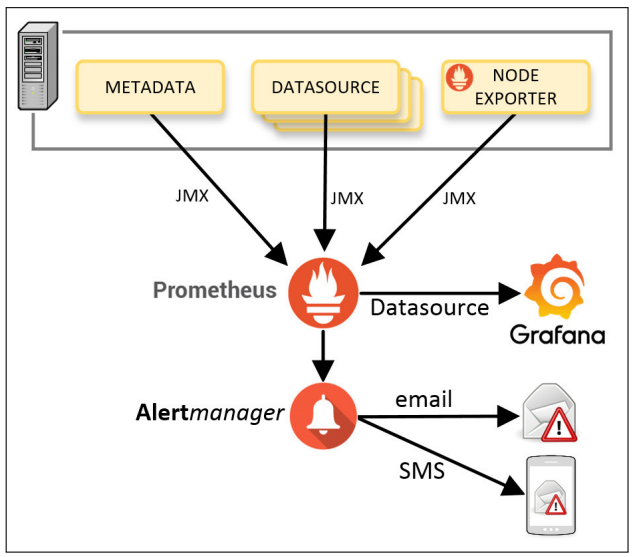

Figure 4: System monitoring and alerting.

Finally, Monit [7] is used for managing (start/stop/restart) the different processes and, more importantly, for monitoring their execution state and restarting them if they stop for any reason.

RELEASE AND DEPLOYMENT

The software release and deployment procedures were fully automated and can be triggered from Jenkins [8] with a single mouse click. The software release is managed by Apache Maven [9] which takes care of building, testing, packaging and releasing the different modules to an artifact repository.

The software deployment, from the artifact repository to the test or production environments, was implemented using Ansible [10], a software provisioning, configuration management, and application-deployment tool. Ansible allows the automation of repetitive and time consuming tasks, allowing to apply the same actions on different environments. For example, deploying the system to the test environment implied installing software packages on 21 virtual machines, a tedious task to be done manually which was repeated very often during the development phase. With Ansible, the service can be deployed to the test or production environments specifying a different *inventory* file. The file contains different configuration settings depending on the environment, such as the host names or the connection strings for the test and production databases.

CONCLUSION

The WinCC OA Data Source service was implemented to archive data produced by the QPS and SCADAR systems into the Next CERN Accelerator Logging System, which will replace the old CERN Accelerator Logging System at the end of the LHC's Long Shut-down 2 in 2020. The data produced by QPS and SCADAR in normal operation is around 175,000 values per second. During the data re-transmission tests the transfer rate reached peaks of 800,000 values per second.

The new service substitutes the previous approach of data transmission to the logging system, from Oracle to Oracle using PL/SQL, by a Java distributed service which is extremely robust and fail-safe with guaranteed data delivery and no data loss. The service can scale up easily by distributing the data generated by the source systems among new processes and distributing the Java Virtual Machine processes on additional machines.

The code-base implemented for the service will be re-used at CERN by other services that store their data into their own temporary database and have a need for a permanent storage solution.

REFERENCES

[1] C. Roderick et al., "The CERN accelerator Logging Service - 10 years in operation: A look at the past, present, and future". *14th ICALEPCS Int. Conf. on Accelerator and Large Expt. Physics Controls Systems.* San Francisco (USA), 7-11 October 2013, TUPPC028.

[2] C. Roderick and R. Billen, "Capturing, Storing and Using Time-Series Data for the World's Largest Scientific Instrument", November 2006, CERN-ABNote-2006-046 (CO).

[3] J. Wozniak, C. Roderick, "NXCALS - Architecture and Challenges of the Next CERN Accelerator Logging Service", *presented at the 17th Int. ICALEPCS Conf., New York, USA, October 2019, paper WEPHA163, this conference.*

[4] P. Golonka et al., "Database archiving system for supervision systems at CERN: A successful upgrade story". *15th ICALEPCS Int. Conf. on Accelerator and Large Expt. Physics Controls Systems.* Melbourne (Australia), 19-23 October 2015, MOPGF021.

[5] Prometheus
`https://prometheus.io`

[6] Grafana
`https://grafana.com`

[7] Monit
`https://mmonit.com/monit/`

[8] Jenkins
`https://jenkins.io`

[9] Apache Maven
`https://maven.apache.org`

[10] Ansible
`https://www.ansible.com`