ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA043

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

CONCEPT AND FIRST EVALUATION OF THE ARCHIVING SYSTEM FOR FAIR

V. Rapp, GSI, Darmstadt, Germany
V. Čuček, XLAB d.o.o., Ljubljana, Slovenia

Abstract

Since the beginning of computer era the storing and analyzing the data was one of the main focuses of IT systems. Therefore, it is no wonder that the users and operators of the coming FAIR complex have expressed a strong requirement to collect the data coming from different accelerator components and store it for the future analysis of the accelerator performance and its proper function. This task will be performed by the Archiving System, a component, which will be developed by FAIRs Controls team in cooperation with XLAB d.o.o., Slovenia. With more than 2000 devices, over 50000 parameters and around 30 MB of data per second to store, the Archiving System will face serious challenges in terms of performance and scalability. Besides of the actual storage complexity, the system will also need to provide the mechanisms to access the data in an efficient matter. Fortunately, there are open source products available on the market, which may be utilized to perform the given tasks. This paper presents the first conceptual design of the coming system, the challenges and choices met, as well as the integration in the coming FAIR system landscape.

INTRODUCTION

Previous experience with the existing GSI facility showed the necessity of a centralized storage of historical data obtained and generated by individual accelerator components and their control system at a permanent location. The main focus of such data archive is to provide the possibility to correlate the actual and historic data in order to analyze the accelerator performance and its proper function. In the coming FAIR [1] facility this function will be offered by the Archiving System, which is currently developed by the Control Systems department, together with XLAB from Slovenia.

This system will allow storing the data on a configurable base of resolution in time, triggered by timing events or on-change. Collected data may be either values gathered from devices, higher level data like computed physics properties or generated abstract data. The Archiving System will include functionality to query, filter, correlate and display historical data. For data identification and synchronization purposes the system should be able to query, receive and store data from other control sub-systems, such as the accelerator's settings management system LSA [2], the Beam Transmission Monitor system or the Post-Mortem system.

The aim is to collect all the relevant data form the devices of the controls system. In numbers it means around 2000 devices, producing more than 30 MB of data per seconds. If stored plain, this amount of data will, in the long term, exceed the capacity possibilities of the controls systems infrastructure. Therefore the system must provide functionality to aggregate the historic data to reduce the required storage size.

SYSTEM OVERVIEW

In order to fulfill all requirements and manage the existing and future challenges in a fast changing environment, the archiving system is designed in a scalable and modular way. It consists of several components, which communicate with each other. Conceptually those components can be split in following functional parts:

- Downstream flow: collecting the data
- Storage component: storing the data
- Upstream flow: extracting the data from the storage

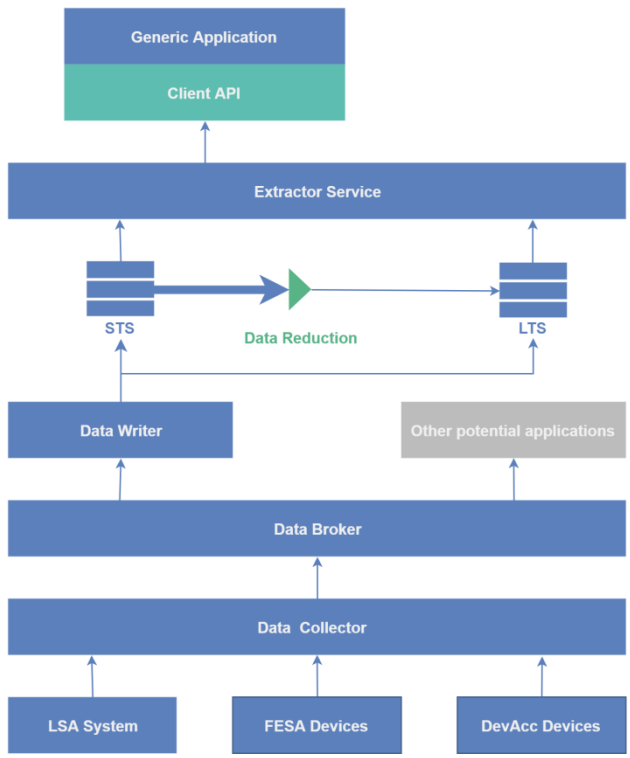

Figure 1: General system overview

In addition to those core parts, different supporting components will be a part of the Archiving System.

The basic idea of the whole systems architecture is to design and implement those components independent

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA043

from each other and clearly define the communication interface between them.

Generally, the workflow of the archiving systems starts with the collection of the relevant data. The actual retrieving of it is described here [3]. Basically, the archiving system must subscribe for all relevant device properties in order to get the desired information. This part of the system is performed by the component called Data Collector, which is a part of the downstream flow.

The information collected by the Data Collector may be of general interest in the future. Therefore, the collected data is disseminated using a message broker. Considering different broker products on their performance, available documentation as well as their current distribution in the market, the Apache Kafka [4] seems to be the most suitable solution for performing of this task. The Data Broker component aggregates all data received from the Data Collector component, originating from multiple sources. It provides a common interface for other components to retrieve data and acts as an optimal consumer for the components underneath, reducing data flow congestion.

The actual writing of the data into the database is performed by the Data Writer component, which acts as a consumer to the Data Broker and wraps all the database calls used for storing the data in Short-Term Storage (STS). From the database point of view, the storage is split into 2 parts. One is the Short-Term storage containing the fine granular data and the other is the Long-Term Storage (LTS), where the data is compressed over fixed time intervals, e.g. by storing only selected aggregated values of those time interval. The Data Reductor component collects and reduces data from the Short-Term Storage and stores it in the Long-Term Storage component.

The database storage components are based on the Elasticsearch [5].

In the upstream data flow, the Data Extractor is performing the actual database queries and aggregating the data. Additionally a client library will be provided to ease the data access. The Client makes requests to the Data Extractor, which checks the user's access level and then extracts data from Short-Term Storage and eventually Long-Term Storage and sends it back to the client. The Data Extractor GUI Client can be used for basic querying and visualization of extracted data. The Extractor Client can also be used to implement custom exports and make data available to other analytical tools. Query optimization and resource usage monitoring are an important task of the Data Extractor.

A number of supporting components called *controllers* are used for configuring the Archiving System at different stages. They can be accessed by the Administration Client directly through the API Gateway for simple operations that include only one controller.

Following the scalability concept the collector, writer and the extractor components are developed for the multi node deployment, i.e. multiple instances will be deployed on different servers. To coordinate the tasks performed by those particular components an Apache Zookeeper [6] configuration service will be used.

CORE COMPONENTS

Data Collector

As described previously the Data Collector component is responsible for subscribing and receiving the data coming from different devices. Those devices may be based on top of different communication protocols: RDA3 (FESA) or Device Access [3]. To achieve this task the Collector utilizes the JAPC interface, which provides a unified subscription interface for continuous data gathering. The Data Collector, as other Data Worker Group and Data Worker Controller Group components and libraries, is written using Java.

Each deployed Data Collector instance registers itself in the Zookeeper Service, making its computing resources (specified as number of threads) available to the Data Collector Controller. The Data Collector Controller is then able to send a list of device configurations to the Data Collector for subscription and processing. Implementing and deploying additional Data Collector device connectivity do not induce changes in the implementation of the controller or other components of the Archiving System, because the whole component is assembled modularly. Each Data Collector can subscribe to several devices at once.

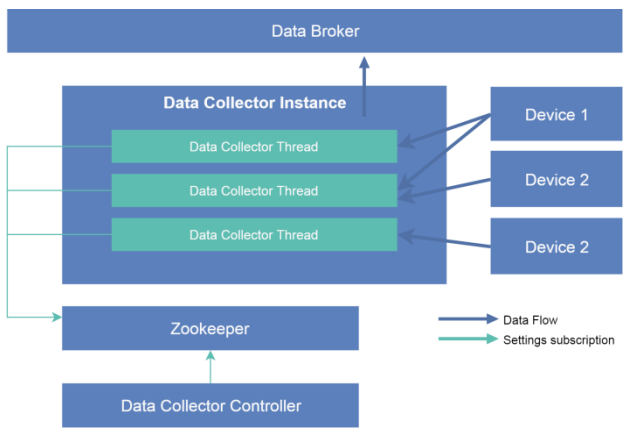

Figure 2: Data Collector overview

Data Collector instances are controlled with the Data Collector Controller component through the REST API. Figure 2 shows the high-level overview of the components and connections. Data acquired from the devices are parsed and encoded in JSON format before publishing them to the Kafka broker. The acquired data may contain device settings as well as actual measurements. The device settings go through an on-change filter before they are pushed to the Meta-Data topic on the Data Broker component. The measurements are stored continuously and pushed to a separate Data topic. The device parameter name and timestamp are stored in each document, so fields from the same device message can be rejoined in the Data Extractor, if needed. The benefit of using separate topics and consequently

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA043

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

separate indices in the Short-Term Storage is that meta-data will use much less storage, making filter queries - their main usage - on them much more efficient. Conversely, measurement data will mostly need to be displayed per-point and usually occupy much wider domain of ever changing values.
Each Data Collector receives the configuration at the start of operation by watching for changes in configuration service, as the Data Collector Controller updates the record that the Data Collector has created upon startup. The Data Collector Controller specifies the type of each device field - whether it produces data, meta-data, or timestamp.
Each Data Collector instance is also sending its utilization and data flow rates to the broker component, which are afterwards written to the database.

Multiple instances of the Data Collector nodes are coordinated by the Data Collector Controller component. The main purpose of this component is to distribute and monitor connections between the devices and Data Collector nodes. It is connected to Data Collector instances by watching and using common nodes of the Configuration Service, where it also watches for newly registered Data Collector instances. Each Collector instance is responsible for a specific set of devices and parameters. The distribution of the load, i.e. subscriptions, between those instances is done by the Controller component.

Data Writer

The main objective of the Data Writer is to encapsulate database specific calls. Similar to other components the Writer is controlled by the Data Writer Controller. The data from the Data Broker is consumed by the Data Writer and inserted into the Short-Term Storage. The meta-data is stored directly to both Short-Term and Long-Term Storage simultaneously. The Status Library is used to publish input and output data flow measurements from all components involved in data processing. In addition it also provides functionality for publishing notifications, relevant for the Administrators, events for triggering remote procedures and system messages, used for monitoring infrastructure resources.
Apart from storing device data, the Writer is also persisting component statuses, events and notifications in Long-Term Storage.
The Writer component follow the same multi thread and multi node approach as the Collector described previously. Every thread of the Data Writer instance is responsible for collecting the data from a particular Data Broker Topic. Data Writer plays an important role during the shutdown procedure, because it is responsible for safely storing all the data, before the maintenance can begin. The currently proposed idea is to shut down every instance, only after the collector group reaches the last record offset and after the Data Collector instances enter the shutdown state. Operation will be coordinated by the Data Writer Controller, using Configuration Service component.

Storage

As mentioned previously the storage concept consists of two parts: short and long term storage. First contains the fine granular data received from the devices. Depending on the device type and the accelerators mode the incoming data rate may go up to 1 kHz. Since the archiving system is aimed to store and manage the data for time periods of multiple years, the data amount here would exceed the data storing capacity which can be supported by the current control system infrastructure. Therefore a reduction of data needs to take place. Compressed data is stored in the long term storage. The compression and writing to the long term storage is performed by the Data Reductor component.
Due to different infrastructural and organizational constrains Elasticsearch database was chosen as a storage backbone for both, long and short term storage. It provides a simple and powerful interface for retrieving and inserting data into the database. In addition to this, it also has some basic analytic tools, which could simplify and reduce code inside the Data Reductor component.
During the compression the data reduction algorithm may change the structure of the data. One of the algorithms which will be implemented is to take multiple beam chain executions during a defined time interval and leave only particular aggregated values of those: mean, standard deviation, maximum and minimum values etc.

Data Extractor

The Data Extractor component is responsible for seamlessly querying data archived in both the Short Term and Long Term Storage. From the architectural point of view the extractor component can be split in several different parts, as shown on Figure 3.

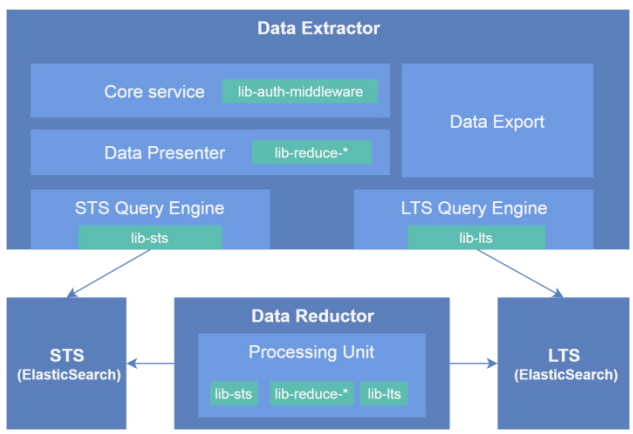

Figure 3: Data Extractor overview

All requests from clients are processed by the Data Extractor core service. Main role of this service is to delegate operations to other sub-components, log user usage statistics and to authorize request.
Data presenter is responsible for joining results from different databases. The actual knowledge behind this operation is possessed by the Data Reductor library,

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA043

because data joining highly depends on the reduction algorithm.

Data Extractor can request data from the databases of the short and the long storage. Query engines are responsible for interpreting requests and assembling consecutive lookups and results for more complex queries. Data Export service implements functionality to convert archived data to other requested formats, e.g. CSV. The Extractor provides RESTfull interface to query the data.

CURRENT STATUS AND FUTURE STEPS

Currently a prototype of the system was developed and successfully deployed in the GSI environment containing the future FAIR relevant components. The prototype however, offers a limited functionality. So, as example all components considering the long term storage are still in the development. On the other hand it provides sufficient functionality to judge over the overall stability and suitability of the whole design concept. Additionally, it allows the first performance tests of different parts of the system. Especially the stability and the long term performance of the Elasticsearch database will be in focus of the coming evaluations. Currently however, the main focus of testing lies in the functionality area.

Remaining functions of the system, described previously are currently under development.

In general the controls system department at GSI plans to use the prototype in the planned beam time in year 2018 at least for smaller amount of chosen devices. This usage must show how well the overall systems architecture is suited to the FAIR and GSI environment constrains. The experience from the usage will also show if the system must be redesigned in the future.

REFERENCES

[1] FAIR website: http://www.fair-center.de
[2] J. Fitzek et al., “Settings Management within the FAIR Control System based on the CERN LSA Framework”, WEPL008, PCaPAC’10, http://www.jacow.org
[3] V. Rapp et al., “Controls middleware for FAIR”, WCO102, PCaPAC’14, http://www.jacow.org
[4] Apache Kafka, website: https://kafka.apache.org/
[5] Elasticsearch, website: https://www.elastic.co/products/elasticsearch
[6] Apache Zookeeper, website: https://zookeeper.apache.org/

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.