A NEW MESSAGE-BASED DATA ACQUISITION SYSTEM FOR ACCELERATOR CONTROL

A. Yamashita*, M. Kago, JASRI/SPring8, Hyogo, Japan

Abstract

In SPring-8, we are constructing MADOCA II, the next generation accelerator control framework. It will be installed in the spring of 2014. We describe the part of the data acquisition system of MADOCAII. The old MADOCA data acquisition system was built on the bases of ONC-RPC for communication between embedded processes and data collector. We designed the new data acquisition system with the long experience on MADOCA. We employ ZeroMQ messages packed by MessagePack for communication. We obtained a high aperformance, highly reliable, well scalable and flexible data acquisition system. In this paper, we will discuss requirements, design, implementation and the result of the long run test of MADOCA II data acquisition system.

INTRODUCTION

The data logging system for the SPring-8 accelerator complex has been operating as a part of MADOCA [1] system. It collects all the data without selection and stores them in the database perpetually and uniformly. It has been serving the development and stable operation of accelerators for 16 years.

In MADOCA data logging system, collector processes periodically request to distributed embedded computers to collect groups of data by synchronous ONC-RPC [2] protocol at fixed cycles. Its *group of signals* strategy reduced number of commands to issue and made data acquisition fast. But it requires many efforts to manage the signals. Also it sacrifices flexibility over the data acquisition timing.

On the other hand, we also developed another MyDAQ [3] [4] system for casual or temporary data acquisition. Data acquisition processes running on embedded computers push BSD socket stream into a server at random time. Its *one stream per one signal* strategy made data management simple while the system has no scalability at all.

The new MADOCA II system has been developed for next generation accelerator. For new MADOCA II, we need a data acquisition system which has a super-MADOCA scale and MyDAQ's simplicity and flexibility.

Also, the next generation accelerators require a control system to collect high density and complex data. The new data acquisition system has to handle them with high reliability.

We define the functions of the new data acquisition system. It collects data from data source and writes them to various data consumers. The data consumers are database and data subscribers.

* aki@spring8.or.jp

DESIGN GOALS

We set the following design goals of new generation MADOCA II data acquisition system.

- Easy to manage. Almost zero configuration is the goal. Only the data source knows metadata, data about signals. The metadata are packed with data. Thus, other components need no configuration files to see metadata.
- Data format. The data acquisition system should accept any format of data and relays to the data consumer.
- Scalability. The system should be horizontally scaled out. If the density of data acquisition gets higher, we can handle them by adding servers. And there should be no limit to the number of servers.
- Preserve the naming scheme of the current MADOCA system. The MADOCA system uses the human readable signal naming scheme like "sr_mag_ps_b_/current_adc", it means current ADC values of storage ring's bending magnet power supply. That naming scheme is well matched to key value data store. The data acquisition system should be built on that naming scheme.
- The system should run on major OS, Windows and Unix like OS. And data source can be written in major modern languages.
- Flexibility in data structure. The old MADOCA supported float and integer data type only . In addition to the datatypes, the new system should handle data structure like array, maps and their combinations.
- Flexibility of data acquisition timing. Every signal data should be acquired at arbitrary timing.
- It should be reliable. Every component should be redundant. No single point of failure should be in the system.
- Two types of data consumers. One is database type, that stores data in database and application reads data from databases at any time. Number of databases should not limit to one. One can add the other database with little efforts. Another type is published and subscribe. User application subscribes to the server and waits data until it published. It enables event driven programming style.

According to above requirements we design data acquisition system.

ISBN 978-3-95450-139-7

ARCHITECTURE

The entire structure of the data acquisition system is shown in Fig. 1 The system is built on the three-tier standard model of the accelerator control system. The data source on embedded computers send messages to the relay processes on the server computers via Ethernet networks. The relay processes relay messages to the writing processes which convert messages to database formats and write them to the databases. The relay servers also publish messages to other processes which subscribe to messages by arbitrary keywords. The messaging is asynchronous, one-way and

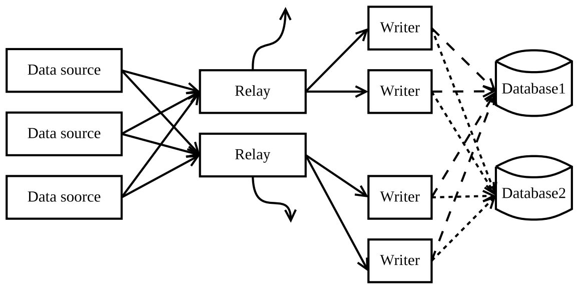


Figure 1: Data acquisition system. Solid line is Push/Pull pattern of ZeroMQ. Curve is Publish/Subscribe pattern of ZeroMQ. Broken line and dotted line is API of each databases.

no call back, that enables fast messaging. We had many troubles with synchronous ONL-RPC in which a message sender sends a message and waits for a reply. If the reply delays, the sender cannot send the next message. Asynchronous messaging avoids such a trouble.

In the original MADOCA system, both data acquisition process and collector process have their own metadata. It sometimes got confused by inconsistent metadata between them. In the new system, only the data source process have metadata. And they are packed with data into a message. Therefore, no inconsistent situation can be happened.

We put multiple relay serves for load balancing and high availability. Data source process sends a message to relay server using ZeroMQ's [5] Push/Pull pattern in the round robin way for load balancing and redundancy. If one relay process fails, the ZeroMQ library detects it and the next message will never send to the failed processes until it recovers.

MESSAGING

Messaging Library

We choose ZeroMQ as the messaging library. In this system, the ZeroMQ serves not only the messages between network connected processes but also the messages between threads inside a process.

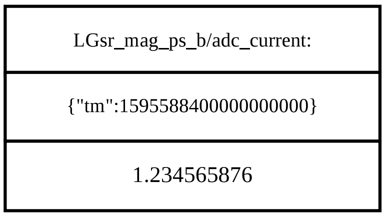
ZeroMQ is an easy to use asynchronous and multi-patterned messaging library which runs on multi-OS and multi-language environment.

Serialization Library

The MessagePack [6] is a library which serializes not only simple data but also complex data structures into binary strings. It is like binary-JSON and generate fast, compact and self described data. It needs no IDL (Interface Description Language) like ONL-RPC. The library has been implemented in over 16 major computer languages and major operating systems. String packed by MessagePack can be exchange between those languages and operating systems.

Structure of Message

The format of the messaging is shown Fig. 2 We split a message into three parts and send them by ZeroMQ's multi-part message feature.



LGsr_mag_ps_b/adc_current:
{"tm":1595588400000000000}
1.234565876

Figure 2: Structure of a message. The first part is keyword. The second part is metadata. Map structure contains timestamp in ns. The third part is data itself. Second and third pard are serialized by MessagePack.

The first part is the keyword part which is used for keys for the key value data store and key for publishing. In MADOCA system, every signal is uniquely named and the systems are built on the naming system. We add a two letter keyword at the head of signal names. The two letter keyword indicates the kind of message. For example, "LG" means the message contains log data and "AL" means the message has alarm data. We also add a colon (:) at the tail of the signal name.

The second part of the message is metadata which is data for describing data. A metadata format has a map data structure which has set of key and value. That has only one keyword 'tm' describing timestamp in long int format at the current time. The map data structure will be expandable to add other metadata in the future. MessagePack library serializes the map data structure into a binary string.

The acquired data from the device are also serialized into a string by MessagePack. And the string forms the third part of the message. MessagePack has so flexible and rich feature that serializes not only single value but also complex data structures including lists, map and their combination.

Using three part messages the applications uses the first messages as a key of Publish/Subscribe and key of key-value stores. For example, the process which writes to Apache Cassandra [7] uses the first part as keyspace key and the timestamp in the second part as column key. If

ISBN 978-3-95450-139-7

metadata and data are not separated, applications have to unpack both of them. It may consume computing resources, especially if the data part has complex data structure.

IMPLEMENTATION

Data Source

A data source generates messages and push them into the relay processes. We developed a polling process that runs on embedded computers, collect data from devices and send messages to the relay servers. We provide a messaging library for them. This process is same as old MADOCA system. But the new system has great degree of freedom, because only the data source has metadata in the new system. We developed a filter program that accepts the standard output of the application and convert to messages. A user who wants to store data writes just an application which write data by printf function. The filter program converts them to messages and send them. It can be applied to application's log data store and can replace logging system like syslog.

Relay Server

The relay servers accept messages from data sources by ZeroMQ's Push/Pull pattern and relays them to writers by same patterns. Also the relay servers publish messages to the subscribers by Publish/ Subscribe pattern of ZeroMQ. ZeroMQ has a function which relays Push/Pull message. We did not use the function because we not only Push/Pull but also also publish the message. Published message may use event-based applications which cannot be implemented with MADOCA. Event-based application with Publish/Subscribe functions may enable stream processing without storing data.

Data sources send messages to multiple servers in round robin way that enhances the scalability and reliability. The multiple relay server may cause broken the order of the message. The older messages may come after new messages. We observed no order breaking at long run test but it could happen with high density data. If message ordering is essential, applications have to care with timestamp in the message.

Writer

The writer receives messages, process them and send to the database using the individual API. One relay server sends messages to one or more writers and no cross connections between relays and writers i.e one writer received from one relay servers. The ZeroMQ's push pull pattern allows cross connection, but it often breaks message order. In the writer, the receiving thread and the writing threads are separated Fig. 3. The receiving thread sends messages using in-process publish and subscribe pattern. Currently two writing threads write data to the databases. One of the databases is Redis [8] for real-time data cache and another is Apache Cassandra for perpetual data storage. It can be easy to add another thread which writes data to another database. And no modification is needed for other threads. The Apache Cassandra gets higher performance by higher number of writing processes [9]. Six node Cassandra cluster is never saturated by 100 writer clients. So many writer processes utilize Cassandra's writing performance.

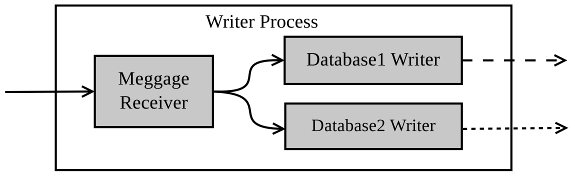


Figure 3: Inner structure of the writer. Receiving thread receives messages by Push/Pull pattern of ZeroMQ via network. It publishes messages by inter-thread communication of ZeroMQ.

Security

Currently no security mechanism is implemented. We assume this system is running on the network protected by firewalls and every computer connected to the network are well trusted. There is no authentication nor encryption for now.

Configuration

Configuration file manages information about servers. Hostnames ports and database information name of keyspace are written in one JSON [10] file. The file is shredded by NFS mounted filesystem.

TEST

We performed a long term test run and measured its performance. The parameter of the test is shown in Table 1 The names of signals are taken from real 47,397 SACLA [11] [12] signals. Simulated data sources generated messages in 1Hz. Three relay serves and 24 writer processes wrote to six node Cassandra cluster and two Redis servers. The speed of writes are over six times higher than current SPring-8 writing speed. We continue writing tests for 24x7 in three months. We examined Cassandra data and no data are dropped. We also examined Redis data and no time reversal has happened in this test term. The relay process itself can handle over 180,000 messages per second.

We examined behavior in the process fails. When one relay process failed, the writer process sent messages to the other relay process after the buffer was filled. After the relay process brings back the messages in the buffer sent to the relay process. Because the size of the buffer was adjustable, we were able to minimize the delay of the message.

ISBN 978-3-95450-139-7

Table 1: Test Parameters

CPU	Intel Xeon X3470 2.93GHz 4Core
OS	CentOS 6.2 64bit
Number of data source processes	240
Number of signals	47397
Writing speed per signal	1Hz
Average length of keyword	38.7 bytes
Length of metadata	13 bytes
Length of data	9 bytes
Number of relay server	3
Number of writer process/relay server	8

We could not perform scalability test adding signals and servers, because our computer resource was limited. We expect the network will be the bottleneck. Until the network is saturated, the system will scale.

CONCLUSION

We constructed easy configured, reliable, scalable and high performance data acquisition system. This system will acquire and store data of SPring-8 during the run in the spring 2014. For long run test, the new system coexist with the old system. We expect the new system will serve the new generation accelerator like old MADOCA system has been serving SPring-8 for a long time.

REFERENCES

[1] R.Tanaka, et al., "The first operation of control system at the SPring-8 storage ring", Proceedings of ICALEPCS 1997, Beijing, China, (1997) p.1.

[2] A.D. Birrell, B.J.Nelson, "Implementing remote procedure calls". ACM Transactions on Computer Systems 2: 39. (1984).

[3] A.Yamashita and T.Ohata "MyDAQ, a Simple Data logging and Display Server", Proc. of PCaPAC'05, Hayama, Japan, (2005).

[4] T.Hirono, et al., "Development of Data Logging and Display System, MyDAQ2", Proc. of PCaPAC'07, Ljubljana, Slovenia, (2007).

[5] Pieter Hintjens, "Zeromq Messaging for Many Applications",O'Reilly Media,(2013).

[6] Sadayuki Furuhashi, Master Thesis, University of Tsukuba, Japan, (2012).

[7] http://cassandra.apache.org/

[8] http://redis.io

[9] M.Kago, et al., "Development of a Scalable and Flexible Data Logging System Using NoSQL Databases", TUPPC012, proceedings of ICALEPCS 2013.

[10] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, (2006) http://tools.ietf.org/html/rfc4627

[11] T. Ishikawa, et al., "A Compact X-ray Free-electron Laser Emitting in the Sub-angstrom Region", Nature Photonics 6, p. 540-544 (2012).

[12] R. Tanaka, et al., "Inauguration of the XFEL Facility, SACLA, in SPring-8", Proceedings of ICALEPCS2011, Grenoble, France (2011).

ISBN 978-3-95450-139-7