MADOCA II DATA LOGGING SYSTEM USING NoSQL DATABASE FOR SPring-8

A. Yamashita[*], M. Kago, SPring-8, Hyogo, Japan

Abstract

The data logging system for SPring-8 was upgraded to the new system using NoSQL database, as part of the MADOCA-II framework. The data logging system has been collecting the log data that is required for accelerator control without any issues since the upgrade. In the past, the MADOCA system powered by a relational database management system (RDBMS) had been operating since 1997. The MADOCA system had grown with the development of accelerators. However, the system with RDBMScould not handle new requirements like variable length data storage, data mining from large volume data and fast data acquisition. Therefore, new software technologies were developed to address these problems. In our proposed system, we adopted two NoSQL databases, Apache Cassandra and Redis, for data storage. Apache Cassandra is utilized for providing a perpetual archive. It is a scalable and highly available column oriented database suitable for time series data. Redis is used for the real time data cache because of a very fast in-memory key-value store. Data acquisition part of the new system was also built based on ZeroMQ message packed by MessagePack. The operation of the new system started in January 2015 after an evaluation term over one year.

INTRODUCTION

The new MADOCA-II data acquisition and storage system took over from the old system in January 2015. Since then, it has been under operation at the SPring-8 accelerator and beam lines control system without major troubles.

In the old MADOCA system [1], one large RDBMS [2] managed not only the equipment and operation parameters but also the log data [3]. It began the production run during the commissioning of SPring-8 in 1997 and has since grown as the accelerator has evolved. The number of signal data increased from 871 at the commissioning in 1997 to 27,626 at the end of 2014 and the size of the database expanded from 17.2GB (1998 one year) to 4 TB. With the progress of the accelerator and for the next generation SPring-8-II [4], we required more scalability, flexibility and maintainability for the database system especially for the log database. In The old MADOCA database system, almost every control application depended on a single database server and the server was required to be reliable and without down time. On the other hand, increase in the number of signals required more performance from one server. Therefore, we employed a fault tolerant database server and SAN (Storage Area Network) storages for reliability. Although, they are very reliable and have worked without trouble in years, they are expensive and pose a difficulty during scale up.

In the previous ICALEPCS, we reported the design and implementation of MADOCA-II database, which used NoSQL databases [5]. The system consisted of two databases, Apache Cassandra [6] for perpetual data store and Redis [7] for the real time data cache. The Apache Cassandra runs on a redundant cluster of commodity servers. It makes the cluster scale-out by simply adding additional server nodes. Data are replicated and distributed to nodes. In our case, up to two simultaneous node downs are tolerable. On the other hand, Cassandra scarifies the consistency of data. A Cassandra cluster distributes data to its nodes. It takes time (about 1 sec in our case) until every node has consistent data. We overcome the inconsistency of Cassandra by writing data into a very fast in-memory data cache; Redis. The data acquisition system writes data into Cassandra and Redis simultaneously. Two Redis servers are run in parallel for redundancy.

We installed the test system in SPring-8 accelerator and beam line control environment. The system acquires and stores the same set of data as MADOCA database system in parallel. During the one year test, we operated the database and data acquisition system with no major trouble. We obtained many know-hows during the test run. From January of 2015, we replaced old data acquisition and database system with new MADOCA-II system with some modifications based on know-hows that were obtained during the test run.

IMPLEMENTATION

Figure 1 shows the entire data acquisition and database system. Our previous paper [8] explained the data acquisition system. We did not make major modifications to it. The data acquisition system operates with the old system in parallel.

We developed a data sender running embedded computer as well as libraries for data reading, web system and alarm system running on client computers.

Data Senders

We developed two types data sender one is called po2m2db which runs on an embedded computer system. It extracts data from the shared memory inside, produces messages, and transmits them to a relay server asynchronously. The other data sender is called cc2m2db for an embedded system that has not enough resources to run po2m2db. The old data acquisition system [9], which runs on a workstations, outputs data to stdout. cc2m2db captures the stdout data, parse them, generates messages and send to the relay server.

[*] aki@spring8.or.jp

ISBN 978-3-95450-148-9

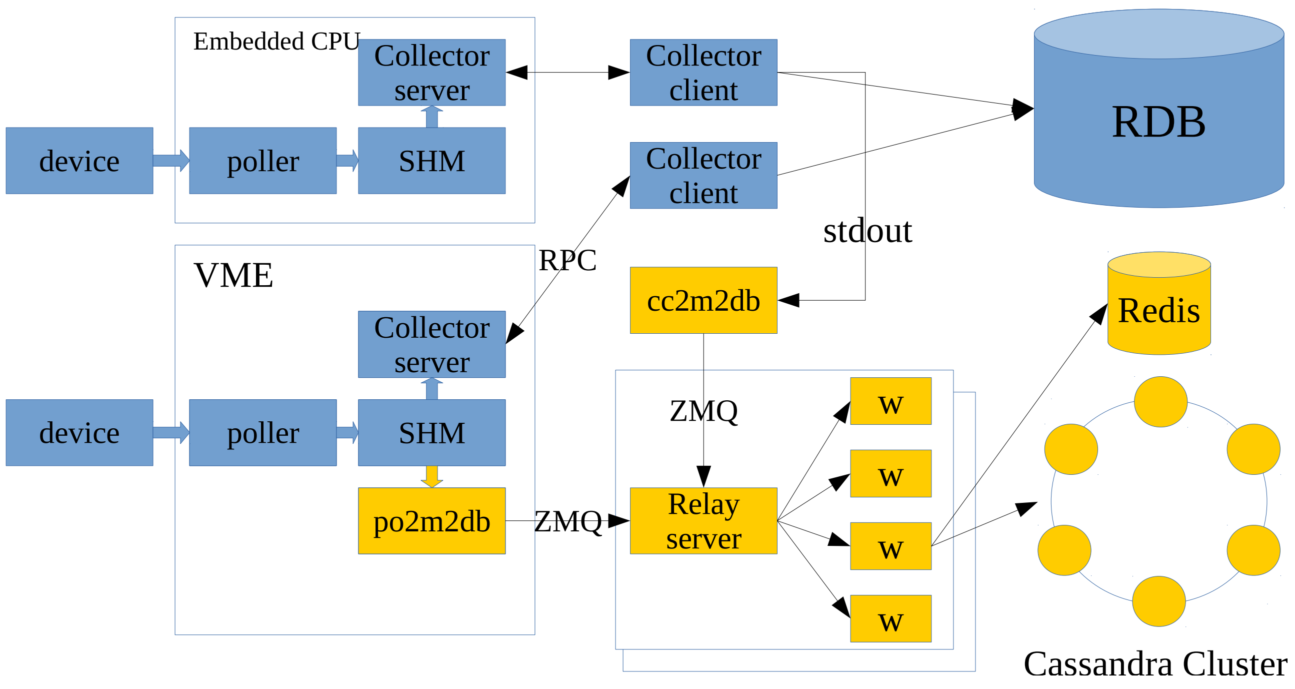


Figure 1: Entire data acquisition system. Blue ones are components of old MADOCA system. Yellow ones are newly developed MADOCA-II system. Both are running in parallel. Two relay servers are running for load balancing and redundancy. “w” means a writer thread.

Client Library

The client libraries that read data from Cassandra or Redis were developed for graphical user interface applications running on workstations. We named the functions identical to the old libraries so that no modifications to the source code of the applications were required. The new libraries did not use the messaging system to access databases but their own database client libraries . The new client library selects one of the Redis servers randomly to read real-time data. Because of the eventual consistency of Cassandra, one may obtain inconsistent values if one accesses only one node of the cluster. The library by default reads data in quorum mode. It returns data if the data from at least two nodes are identical.

Alarm Survey

We re-implemented alarm survey system for the new MADOCA-II database. The old alarm system reads the newest log data from RDBMS, compares them to the nominal values stored in RDBMS, and writes alarm data to RDBMS if an alarm event is detected [10]. We implemented RDBMS in the system for normal value and alarm event stores because those data are suitable for RDBMS. RDBMS has better equipped to manage the relation between data and flexibility of query over NoSQL. The survey application is completely re-written using Python2.7 [11] and PyQt4 [12] for GUI. Casandra, Redis, RDBMS and MessagePack [13] libraries were open source resources. The old survey system was running in parallel for log term test. The results of both systems were compared and examined. and were found to have no difference between them.

MODIFICATION FOR PRODUCTION RUN

Before the production run, we made some modifications to the system after the one completion of the year test run.

Table Structure of Cassandra

Initially , the table structure was changed from one big table to divided smaller tables. We implemented Cassandra with one big table that stored all the data in the test run and observed that a single big table is not equipped to manage large data. Casandra writes data into set of files named SSTables. As the size of SSTables grow, those files are combined into large SSTables. This process is called compaction. Compaction requires temporary disk space with the same size of SSTables. Therefore, the size of SSTables for one big table cannot exceed more than half of the disk space as it an inefficient disk utilization method. In addition, compaction for big SSTables requires large amount of disk i/o and CPU power.

We divided a big table into small tables for each month. Owing to this modification the compaction requires less temporary disk space, disk i/o and CPU resources. Although the database management increases in complexity, the backup job for files become much simpler with smaller files. The reading libraries also need to be modified to read many tables in one function call.

ISBN 978-3-95450-148-9

Alive Flag

If an embedded computer stops, the data generator running on it will stop sending messages. The alarm system detects the trouble. A message has a timestamp embedded in its meta-data part. One may detect trouble by comparing the timestamp with the system clock of the alarm system. However, each signal has its own data cycle, therefore differences in the time time required to trigger the alarm should be set for each signals. We added a cycle period to the meta-data part of the message. Alarm system identifies the trouble when the time stamp is delayed more than ten times the cycle.

Time to Live

Many signals keep same value in accelerator control system. Especially signals for status, like on/off, remain unchanged for long period of time. We will not store them if the value of a signal is identical to the previous value to save the total amount of data stored on disk. Cassandra provides a "time to live" option for each column. The columns are marked when the time limit is exceeded and the marked columns are deleted at the next compaction job. We add time to live values to meta data. The writer reads the time to live meta data and add them to Cassandra's insert command. For data that are unchanged for long time, unchanged data are stored in the Cassandra database every five minute. The final structure of the message is shown in Fig. 2.

Cluster Expansion and Upgrade

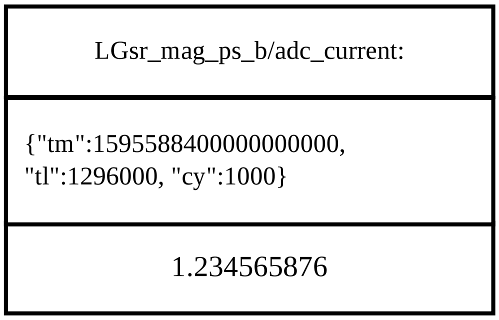
The six node test cluster was changed to 12 nodes to store large data sets. We constructed the other 12 node production cluster, upgraded Cassandra version from 1.2 to 2.0 which enables virtual nodes (vnodes) and implemented a new table scheme. The vnode builds virtual nodes on physical nodes and enables fast data repair by using the data replicas on other vnodes. The specifications of servers are described in Table 1.

Table 1: Server specifications

Server	Dell PowerEdge R420
OS	CentOS 6.6 (64bit)
CPU	Intel Xeon E5-2420 v2, 6c, 2.2GHz
Memory	16GB
Hard disk (system)	600GB SAS 15Kr/m x1
Hard disk (data)	3TB SATA 7,200r/m x3
Cassandra version	2.0.10
JavaVM	JRE1.7.0-67-b01

Data Migration

The archived log data on old RDBMS were duplicated into the production Cassandra cluster. 4TB of data sets on RDBMS stored from 1997 to the end of 2014 became 0.75TB per Cassndra nodes. The total data size in the Cassandra cluster was 9TB, which includes three replications of data. The raw size of data in RDBMS is 4TB and becomes larger in the real RAID5 file system.



LGsr_mag_ps_b/adc_current:

{"tm":1595588400000000000,
"tl":1296000, "cy":1000}

1.234565876

Figure 2: Structure of a message. The first part is a key and not Messagepacked. The second part is meta data. “tm” is timestamp in nanoseconds. “tl” is time to live in miliseconds and “cy” is data acquisition cycle in milliseconds. The last part is data. Data in second and third part is packed by Messagepack.

DIAGNOSTICS SYSTEM

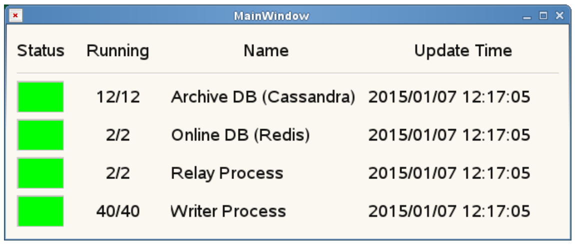
We built and installed several diagnostic systems to ensure the for healthy operation of the system.

Server Resource Monitoring

Server resources are monitored by Zabbix [14]. It not only displays information acquired from SNMP (Simple network management protocol) but also the data acquired from JMX (Java Management Extensions) on Web browsers. JMX monitor the heap status of Cassandra on the Servers.

Data Acquisition Monitors

We build two monitoring systems for relay and writer processes. One is a simple system shown in Fig. 3 that displays alive or dead status of processes and runs on the display wall in the main control room to display status to operators. The other is for system experts shown in Fig. 4. It displays show message per seconds, elapsed time to write into database, and other information. The application also send/receives messages to/from relay servers for the test. Both monitor applications were written in C++ with Qt4.



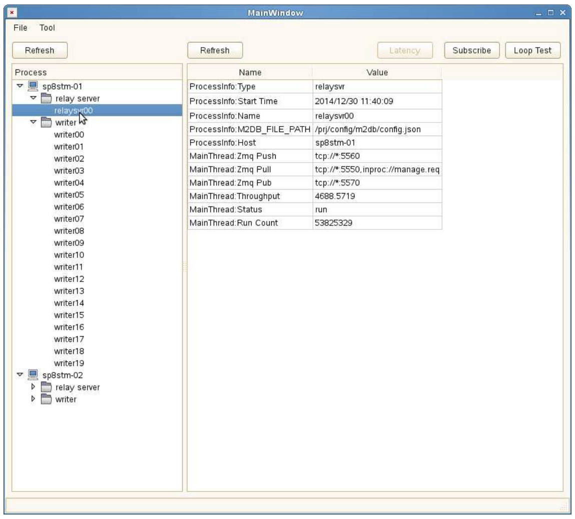
Status	Running	Name	Update Time
🟩	12/12	Archive DB (Cassandra)	2015/01/07 12:17:05
🟩	2/2	Online DB (Redis)	2015/01/07 12:17:05
🟩	2/2	Relay Process	2015/01/07 12:17:05
🟩	40/40	Writer Process	2015/01/07 12:17:05

Figure 3: Screen capture of monitoring tool for operators.

Mail Notification System

Although the system has high reliability with no single point of failure, we set up a mail notification system for system troubles. System manager also received message received check mail other than trouble notifications once a day.

ISBN 978-3-95450-148-9



Name	Value
ProcessInfo:Type	relaysvr
ProcessInfo:Start Time	2014/12/30 11:40:09
ProcessInfo:Name	relaysvr00
ProcessInfo:M2DB_FILE_PATH	/prj/config/m2db/config.json
ProcessInfo:Host	sp8stm-01
MainThread:Zmq Push	tcp://*:5560
MainThread:Zmq Pull	tcp://*:5550,inproc://manage.req
MainThread:Zmq Pub	tcp://*:5570
MainThread:Throughput	4688.5719
MainThread:Status	run
MainThread:Run Count	53825329

Figure 4: Screen capture of monitoring tool for expert.

CURRENT STATUS

We began operating the production system in January 2015. We have had no major trouble since then. We manage 27,000 signals on the system and handle 9,000 messages per second. We also monitor system resources like CPU loada and disk and memory usage and found no abnormal status.

The MADOCA-II database still remains a part of the old system. The system on embedded computers and cc2m2db are temporal ones that send data to old system in parallel. We will implement a new MADOCA-II dedicated data generation application for embedded computers. The data registration files for embedded computers and database systems are now separated into two files. The segregated data registration files were problematic for the old system, sometimes they are inconsistent. Therefore, we aim to unify the files in our proposed system for simplification and troubleshooting the inconsistency.

SUMMARY AND FUTURE PLANS

We started SPring-8 operation using the new MADOCA-II database. Before production, we modified the system using the knowledge obtained from long term test operation. We installed a monitoring system and re-wrote the alarm system, client libraries and Web systems. We are now planning to extend the dedicated MADOCA-II system to embedded computers and signal registration system.

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).

[2] A.Yamashita, et al., “The Database System for the SPring-8 Storage Ring Control”, Proceedings of ICALEPCS 1997, Beijing, China, (1997).

[3] A.Yamashita, et al., “Data archiving and retrieval for SPring-8 acceleratorl complex”, Proceedings of ICALEPCS 1999, Trieste, Italy, (1999).

[4] `http://rsc.riken.jp/pdf/SPring-8-II.pdf`

[5] M.Kago, et al., “Development of a Scalable and Flexible Data Logging System Using NoSQL Databases”, Proceedings of ICALEPCS 2013, SanFrancisco, USA, (2013).

[6] `http://cassandra.apache.org/`

[7] `http://redis.io`

[8] A.Yamashita, et al., “A New Message-Based Data Acquisition System for Accelerator Control”, Proceedings of ICALEPCS 2013, San Francisco, USA, (2013).

[9] T.Masuda, et al., “Data Acquisition System with Database at the Spring-8 Storage Ring”, Proceedings of ICALEPCS 1997, Beijing, China, (1997).

[10] A.Yamashita, et al., “The alarm system for the SPring-8 storage ring”, Proceedings of ICALEPCS 1997, Beijing, China, (1997).

[11] `http://www.python.org`

[12] `http://riverbankcomputing.com/`

[13] S. Furuhashi, Master Thesis, University of Tsukuba, Japan, (2012).

[14] `http://www.zabbix.com/`

ISBN 978-3-95450-148-9