Controls & Computing

Development, implementation and operation of MADOCAII middleware for accelerator and beamline control

1. Introduction

1.1. What is MADOCA SPring8 Control Middleware?

The control group of SPring-8 developed MADOCA (Message and Database Oriented Control Architecture) middleware for the SPring-8 storage ring control system in 1997. The SPring-8 control system consists of workstations in the central control room and embedded computers placed close to accelerator components such as the magnet power supplies, vacuum pumps, beam monitors and radio frequency generators. These computers are connected by a network, and the operators send commands from the workstations to the embedded computers. Over 500 embedded computers distributed around SPring-8 control with more than 30,000 control points.

1.2. Communication management

The MADOCA middleware transports control commands issued by graphical user interface (GUI) applications, which run on operator workstations, to the equipment control applications (Equipment Manager, EM), which run on embedded computers placed close to the accelerator components. The operator sends a command such as 'set b-magnet power supply to 12.34A'. We build a command such as set/sr_mag_ps_b/12.34A and send this string as a message. In this command, the operator does not care which embedded computer controls the b-magnet power supply and how the embedded computer controls a power supply to give 12.34A. MADOCA takes care of these issues. We refer to this scheme as *device abstraction*. When MADOCA middleware running on the operator workstation receives this message from a GUI, it delivers this message to another MADOCA middleware running on an embedded computer. The relationships between the embedded computers and components are registered in the parameter database. An application running on the embedded computer receives a message from the MADOCA middleware, interprets the command and converts it to the machine language for the components. This device abstraction makes GUI development easy because detailed information is not required to control components. The messages between the GUI and EM must be delivered in an accurate, fast, reliable and simple way by the middleware. The MADOCA middleware was designed to satisfy these requirements.

1.3. Database

1.3.1. Parameter management

The other aspect of MADOCA is its database-oriented architecture. MADOCA manages almost every parameter required for the accelerator and beamline operations using a single relational database management system (RDBMS). It manages accelerator parameters including magnet positions, components, embedded computer relations, alarm thresholds and setting parameters for the components of the accelerator.

1.3.2. Logging

The RDBMS not only manages parameters but also logs data collected from almost every component around SPring-8. SPring-8 has over 30,000 control points. If one piece of equipment malfunctions, one may be notified by the electron beam stopping, but how do we figure out which equipment is broken? MADOCA answers this question by monitoring every piece of equipment continuously. Monitored data are stored in the logging database implemented on the same RDBMS server. The logging database makes troubleshooting easy. The MADOCA logging database has stored all data since SPring-8 was commissioned. Storing the logging data in the RDBMS provides a consistent and simple means of data access. One can access logged data from GUI using C-library or browser using web interfaces.

2. MADOCAII, the next-generation MADOCA

The first MADOCA system was developed to control the storage ring. It now covers beamlines, injector synchrotron, injector linac, New SUBARU and SACLA. Since its development about 17 years ago, we have experienced many shortcomings of the MADOCA system. We have thus developed a new next-generation control system (MADOCAII) based on new technologies. We have experienced several limitations of MADOCA that were serious obstacles in developing the next-generation control system. We will discuss the limitations of MADOCA regarding both messaging and the logging database in the following section.

2.1. Messaging in MADOCAII

2.1.1. Messaging in MADOCA

The messaging in MADOCA was based on an ONC-RPC (open network computing remote procedure call) system. Although ONC-RPC can handle data structures of any length, we limited exchanges to fixed-length strings. This works well for exchanging short or scalar messages such as 'switch on' or 'set 12.3A'; however, it cannot handle large amount of data, such as images, or complex data structures. These data were exchanged using a network file system (NFS). This system is slow and lacks real-time capability. Moreover, MADOCA messaging was limited to only Unix-like systems and a C-language environment. Although Windows OS, which is not a Unix-like system, is embedded in many commercial instruments, it cannot handle MADOCA messaging. We had to set Unix gateways to use Windows OS in the SPring-8 control system. It was also difficult to use ONC-RPC with other computer languages such as LabView or Python, which have recently become commonly used in the SPring-8 accelerator and beamline control system. When one issues an ONC-RPC command to an application, it is necessary to wait for the callback. This means applications must be written in a "send command, wait for answer" which is called synchronous programming. This slows down the control application because while waiting for the callback, the application cannot perform other tasks. In the next-generation control system, asynchronous programming is used to solve this problem. In asynchronous programming, the application works in a "send, send, send, send commands, do something while waiting and receive callbacks" manner, which reduces the waiting time compared with that in synchronous programming. In the MADOCA system, the roles of the client (GUI) and server (embedded computer) were fixed and it was difficult to change the roles, for example, to send commands from an embedded computer to an operator console or to communicate between GUIs.

2.1.2. Messaging in MADOCAII

The messaging in MADOCAII overcomes the problems in the old MADOCA messaging system. The most important point is that we use the ZeroMQ asynchronous messaging library instead of ONC-RPC. ZeroMQ can exchange variable-length messages asynchronously on multiple platforms and in multiple computer languages. The ZeroMQ library can transport variable-length strings. We can use not only scalar values like voltage, current and vacuum measurement but also complex data structures and images by packing them into strings by using the MessagePack object serialization library. Because the ZeroMQ and MessagePack libraries run on various platforms including Windows OS and support major languages, MADOCAII can run on a wide range of platforms and languages. Because ZeroMQ provides flexible connections, it is possible to communicate not only from operator workstations to embedded computers but also from workstations to workstations and between embedded computers. This enables many opportunities for control in the future. Although MADOCAII has dramatically changed internal communication, we designed MADOCAII without rewriting the applications written in the MADOCA environment. Other than a very small number of applications, old applications now work with MADOCAII without any modifications.

2.2. Data logging in MADOCAII

2.2.1. Data logging in MADOCA

Data logging in MADOCA also had many problems. The RDBMS has low performance for data logging to maintain critical consistency, which is essential in parameter management. We inserted a group of values into the RDBMS in each transaction to compensate for its low performance. While *one value by one insertion* is ideal for data management, it requires too many transactions for the RDBMS. Moreover, for consistency, the RDBMS requires one unified memory space in the system, which means that to obtain better performance by upgrading hardware, the computer hardware has to be an expensive shared-memory type multicore CPU.

2.2.2. Data logging in MADOCAII

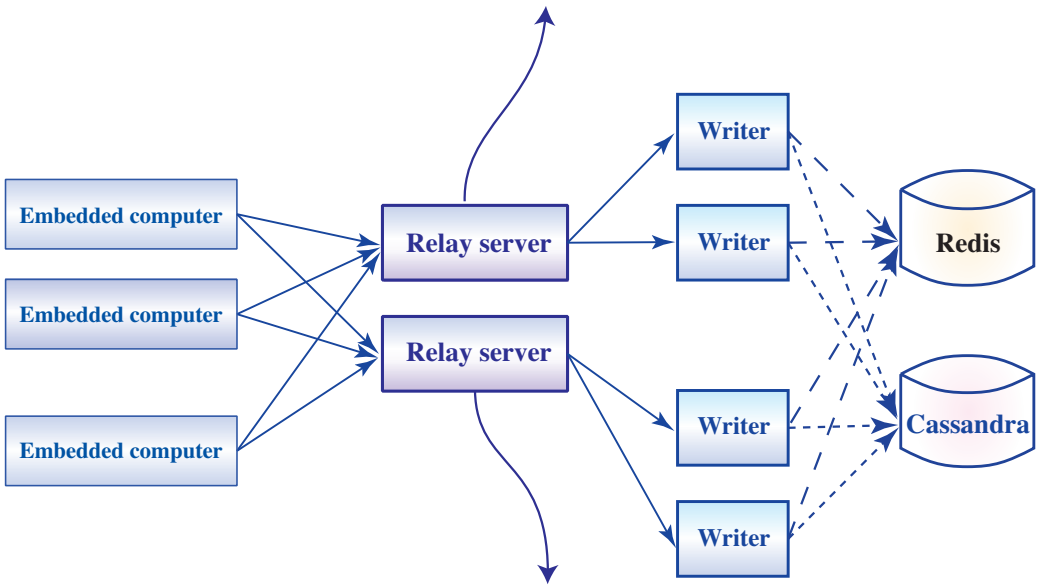
2.2.2.1. *NoSQL database*

In the logging database management of MADOCAII we use the NoSQL (not only SQL) database instead of the RDBMS. We use the Cassandra database for perpetual data storage and the Redis database for the newest data cache. Both databases are NoSQL databases. Cassandra runs on a homogeneous inexpensive multi-node computer cluster, which means there is no master or slave. We found that the performance of Cassandra is linearly proportional to the number of nodes. Therefore, if it is necessary to improve performance of the Cassandra cluster, we will simply add computers.

In the Cassandra database, each data record is copied into three replicas and distributed to different computer nodes. Thus, if one or two computers in the cluster are broken, data are kept safely and clusters will still run. Redis is an ultrafast in-memory database with which one can obtain the newest values one order of magnitude faster than the RDBMS. Both Redis and Cassandra pack data into string format using the MessagePack library. Thus, we build only one data table format to reduce the workload in database management.

2.2.2.2. *Data acquisition*

When we introduced the NoSQL database, we drastically changed our data acquisition processes. MADOCA data acquisition process running on a workstation periodically requests data to the embedded computers, packs received data into a RDBMS command and sends it to the RDBMS. The newly developed MADOCAII data acquisition system reverses this process. MADOCAII performs processes on the embedded computers, which send data to a relay server at their own timing. The relay server then writes the data into NoSQL servers. We installed two identical relay servers for fault tolerance and load balancing. If one server is broken, the other relay server will work, and, if necessary, performance can be improved by adding another relay server. Similarly to the messaging in MADOCAII, the MADOCAII database library has backward compatibility to the MADOCA database system so that no modifications were needed to the source code.



MADOCA II data acquisition system schematics. Data from embedded computers are packed in ZeroMQ messages and sent to the relay servers (solid line). Two relay servers are in operation for redundancy. The relay servers pass the messages to the writer processes. The writer processes write data into the NoSQL database servers. The dashed lines mean the data path to the Redis servers and the Cassandra API is expressed in the dotted line. The relay servers also publish messages with pub/sub mechanism (curly lines).

3. Performance and Operation

3.1. Performance

We measured the performance of MADOCAII. For the messaging, the round-trip messaging between processes running in the same computer takes 0.4ms. It takes 1.8 ms for the round-trip between operator workstations and embedded computers. The data insertion performance of Cassandra exceeds about half a million per second in the case of six computer nodes. This is about 50 times higher than that for the previous MADOCA RDBMS system. The performance will also linearly increase with the number of nodes.

3.2. Operation

MADOCAII messaging has been in operation since September 2012 and the database has been working since January 2015 in the SPring-8 accelerator and beamlines after prolonged testing. Both have been working with few major problems.

4. Conclusion

We have developed and implemented the new MADOCAII system. Although it has an almost identical interface to the old MADOCA system, the internal processes are radically changed by using new information technology such as asynchronous messaging and NoSQL databases. The main aim of the development has been achieved and we expect MADOCAII to serve as the future SPring-8 control system.

Akihiro Yamashita* and Tomohiro Matsushita

Japan Synchrotron Radiation Research Institute (JASRI)

*E-mail: aki@spring8.or.jp