STATUS OF OPERATION DATA ARCHIVING SYSTEM USING Hadoop/HBase FOR J-PARC

N. Kikuzawa[#], H. Ikeda, Y. Kato, N. Ouchi J-PARC, Tokai-mura, Naka-gun, Ibaraki, Japan
A. Yoshii, NS Solutions Corporation, Shinkawa, Chuo-ku, Tokyo, Japan

Abstract

J-PARC (Japan Proton Accelerator Research Complex) consists of much equipment. In the Linac and the 3 GeV rapid cycling synchrotron ring (RCS) in J-PARC, data of about 64,000 EPICS records have been collected for control of these equipment. The data volume is about 2 TB every year, and the total data volume stored has reached about 10 TB. The data have been being stored by a Relational Database (RDB) system using PostgreSQL since 2006 in PostgreSQL, but it is becoming that PostgreSQL is not enough in availability, performance, and flexibility for our increasing data volume.

We are planning to replace PostgreSQL with Apache Hadoop and Apache HBase to accumulate enormous operation data produced from the Linac and the RCS in J-PARC. HBase is so-call NoSQL, which has scalability to data size at the cost of the high broad utility of SQL. HBase is constructed on a distributed file system provided by Hadoop, a cluster with advantages including automatically covering its cluster nodes' breakdowns and easily adding new nodes to expand its capacity. The new database system satisfies high availability, high performance, and high flexibility of storage expansion. The purpose of this paper is to report the present status of this archive system.

INTRODUCTION

J-PARC is controlled with a lot of equipment, and we have been archiving a time series of operation data provided from about 64,000 EPICS records for the Linac and the RCS since 2006 [1]. PostgreSQL has been used in the present data archiving system, but it has some problems of capacity, extensibility, and data migration. In order to deal with these problems, we proposed a next-generation archive system using Apache Hadoop [2], a distributed processing framework, and Apache HBase [3], a distributed database [4].

Hadoop is a widely used open-source cloud framework for large scale data processing. The HBase is a distributed, scalable big data store on a cluster built with commodity hardware. One of the cores of Hadoop is a file system called HDFS (Hadoop Distributed File System), and HBase runs on it. Hadoop and HBase are scale-out architecture, and we can expand storage volume dynamically by adding new nodes. Moreover, they are designed based on the assumption of frequent breakdown

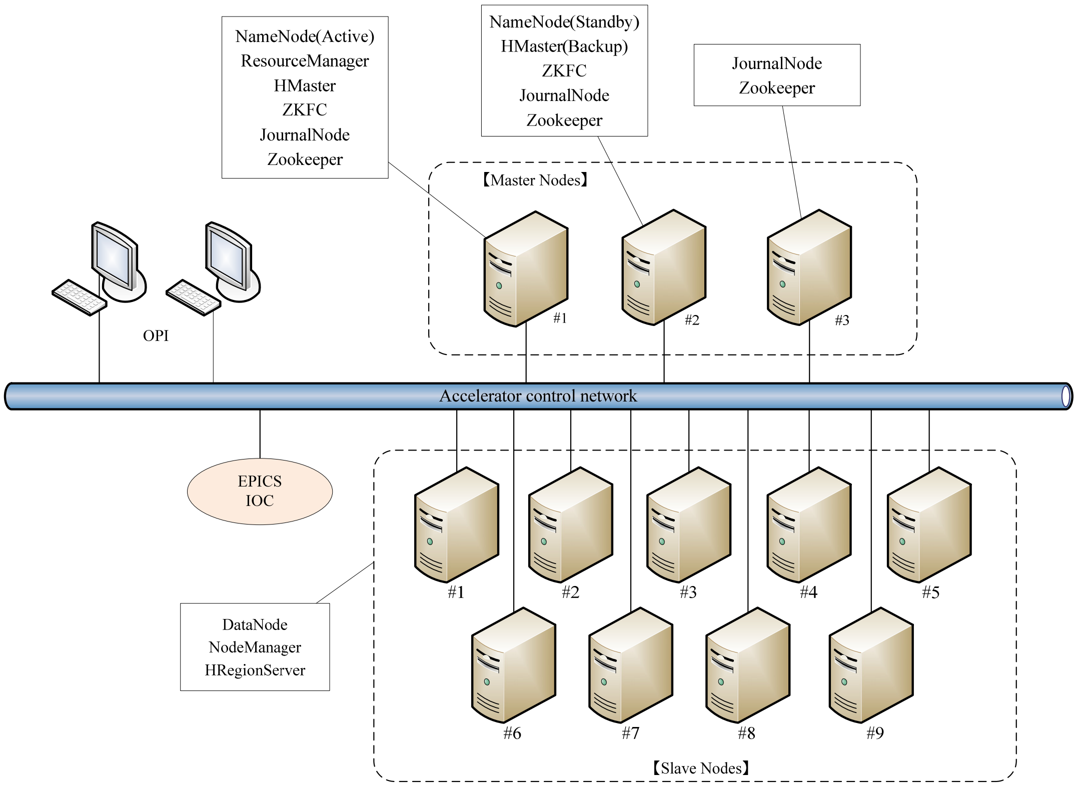

Figure 1: System configuration of the new archiving system.

[#]kikuzawa.nobuhiro@jaea.go.jp

ISBN 978-3-95450-146-5

of cluster nodes in large scale clusters, and they have a tolerance for the breakdowns and are easy to recover.

HBase is a type of "NoSQL" database and its scalability is established under restrictions on such functions as transaction, table combination, etc. that an ordinary RDB has. That means HBase doesn't take the place of RDBs and we should take account of trade-offs between them.

Having considered that HBase is the best for the time series data archiving system, we built and tested a small testing system, and we have updated the system with new versions of Hadoop/HBase that provide indispensable features. We will describe the present status of this new archiving system.

DATABASE SYSTEM CONFIGURATION

Hardware Configuration

Hadoop and HBase are designed to have master-slave architecture, and we use our machines as master nodes and slave nodes, with appropriately deploying their software components. We have built a cluster composed with nine slave nodes, each of which is a commodity server with four 2 TB local hard disk storages configured in RAID 5, and is interconnected using Gigabit Ethernet (GbE). About 50 TB of effective capacity contributes to the HDFS. Figure 1 shows the system configuration.

Update of Hadoop and HBase

With early versions of Hadoop/HBase we made a testing system by using Heartbeat [5], Pacemaker [6], and DRBD (Distributed Replicated Block Device) to add redundancy to NameNode. NameNode is one of the components deployed in a master node and manages metadata of the distributed file system, and prior to Hadoop 2.0.0 it was a single point of failure (SPOF), which is not acceptable for practical purposes.

One of the problems is, this is just cold standby and NameNode might take a quite long time to complete starting up, preventing you from accessing the cluster. It took about 5 minutes for failover and failback in our system, since it was necessary to perform starting of the HBase system service after the initialization processing of the Hadoop system service start-up is completed [7]. It might take an hour, depending on the size of the cluster. That is far from high availability.

We have updated Hadoop to the version 2.2.0 now. Hadoop 2.x provides a hot standby NameNode, which can take over the state that the previous active NameNode has provided, with no downtime. At least three master nodes are needed for this function to deploy ZooKeeper and JournalNode. ZooKeeper [8] is a high-performance coordination service for distributed applications, and both Hadoop and HBase depend on. JournalNode is one of the components of Hadoop. ZooKeeper and JournalNode are based on a majority decision among machines, and it is meaningful to deploy them on an odd number of machines. Table 1 shows the spec of our system. For now we don't prepare sufficient machines for the 2nd and 3rd

Table 1: Spec of the New Data Archiving System

Master node #1	DELL PowerEdge R610 CPU: Intel Xeon E5620 (4Core 2.4GHz) MEM: 24GB HDD: 600GB SAS 10 x 4 (RAID10)
Master node #2	DELL PowerEdge R200 CPU: Intel Xeon E3210 (4Core 2.13GHz) MEM: 8GB HDD: 160GB x 1
Master node #3	DELL PowerEdge 860 CPU: Intel Xeon X3210 (2Core 2.4GHz) MEM: 4GB HDD: 250GB x 1
Slave Nodes	DELL PowerEdge R410 CPU: Intel Xeon E5620 (4Core 2.4GHz) MEM: 24GB HDD: 2TB x 4 (RAID5)

Table 2: Components Deployment (Master Nodes)

Master Node	#1	#2	#3
NameNode	o	o	
Journal Node	o	o	o
ZKFC	o	o	
Resource Manager	o		
History Server	o		
ZooKeeper	o	o	o
HBase Master	o	o	

master nodes, and we are planning to enhance them in preparation for the practical operation phase. Table 2 shows components deployment of the master nodes.

The slave nodes have been built on RAID 5, because we have focused on making clear the procedures to restore nodes rather than evaluating the performance of the system. However, any RAID makes the disks to cooperate and prevents simultaneously accesses, and has possibility to give significant impacts to the performance of Hadoop/HBase. We are planning to unbind RAID in slave nodes and reevaluate.

As for the version of HBase, we developed our system with HBase 0.94.x, but even the documents of HBase don't make clear whether the HBase 0.94.x is compatible with Hadoop 2.2.0. For this reason, we have adopted HBase 0.96.1.1, which targets Hadoop 2.x from the beginning.

Table Structure

The present system uses PostgreSQL to store much data. PostgreSQL had various restrictions about data size, and the system was designed to divide the large amount of the data into multiple tables, with dynamically creating tables, according to a group the data belongs to and its monitoring time. That drops many advantages of the

RDBMS, and introduces complexity and restrictions into logic to store and retrieve data to/from the system.

HBase is a type of column-oriented database, and a simple structure of key-value is suitable. It is possible to have huge size tables as compared with the conventional database system. That means a schema design in the column-oriented database is very different from one in an ordinary RDB.

In HBase each record is identified with a binary sequence referred to as a row (a primary key in terms of RDBs) and is stored in ascending order of rows. Basically rows are the only index and the design of the rows is directly related to the performance to retrieve records. The design is also related to load distribution to store records as follows; Records are divided by automatically or manually selected rows into regions, which are distributed among nodes in a cluster. If two rows start with the same value, the records tend to be located in the same region, and consequently in the same node. For an instance, if you design the row which starts with the same value followed by a monotonically increasing value like a timestamp, when you are going to store multiple records they tends to be written in the same server [9], which results in slow writing speed.

We have carried out examinations about table structure [10]. In order to avoid the above problems, we have deigned the row that starts the binary expression of the EPICS record name followed by the binary expression of its measured time. The time is represented by progress milliseconds of the 1970 UTC epoch, and in order to search the latest data first, its binary representation is decided to subtract the time from the maximum of a signed 64-bit integer.

DATA BROWSER

An application has been developed which acquires data from the Linac/RCS and stores in HBase. The acquisition of data is performed via the EPICS channel access. The storing in HBase may be blocked temporarily, and the application has a buffer.

Another application has been also developed which retrieves the data stored in HBase [11]. The application is in the form of a plug-in into Control System Studio (CSS) [12]. This plug-in extends the class ArchiveReader in the plug-in org.csstudio.archive.reader provided by CSS, and you can refer data in HBase via Data Browser, one of the convenient GUI components in CSS. Figure 2 shows an example of the Data Browser plot.

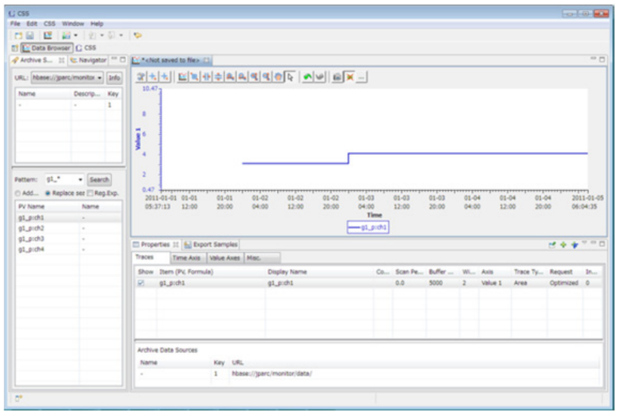

Figure 2: Example of data browser plot.

The performance of these tools is being checked after setting up the cluster correctly.

SUMMARY

We have proposed the next-generation archive system using Apache Hadoop, a distributed processing framework and Apache HBase, a distributed database. With new versions of Hadoop and HBase, it has turned out that the system needs to be revised. Data archiving and retrieval tools have been developed. The performance of the tools is being checked after revising the system.

REFERENCES

[1] S. Fukuta et al., "Development Status of Database for J-PARC RCS Control System (1)", Proceedings of the 4th Annual Meeting of Particle Accelerator Society of Japan, August 2007. [in Japanese]
[2] http://hadoop.apache.org/
[3] http://hbase.apache.org/
[4] N. Kikuzawa et al,. "Development of J-PARC Time-Series Data Archiver using Distributed Database System", Proceedings of ICALEPCS2013.
[5] http://www.linux-ha.org/wiki/Heartbeat
[6] http://www.linux-ha.org/wiki/Pacemaker
[7] A. Yoshii et al., "Status of J-PARC operation data archiving using Hadoop and HBase" Proceedings of the 10th Annual Meeting of Particle Accelerator Society of Japan. [in Japanese]
[8] http://zookeeper.apache.org/
[9] Apache HBase Reference Guide http://hbase.apache.org/book.html
[10] A. Yoshii et al., "J-PARC operation data archiving using Hadoop and HBase" Proceedings of the 9th Annual Meeting of Particle Accelerator Society of Japan. [in Japanese]
[11] N. Kikuzawa et al., "Development of tools for the J-PARC operation data archiving using HBase/Hadoop", Proceedings of the 11th Annual Meeting of Particle Accelerator Society of Japan. [in Japanese]
[12] http://controlsystemstudio.org/