Fusion Engineering and Design 129 (2018) 207–213

Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

J-TEXT distributed data storage and management system

W. Zheng[a,b], Q. Liu[a,b], M. Zhang[a,b,*], K. Wan[a,b], F. Hu[a,b], K. Yu[a,b]

[a] State Key Laboratory of Advanced Electromagnetic Engineering and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
[b] School of Electrical and Electronic Engineering, Huazhong University of Science and Technology, Wuhan 430074, China

ARTICLE INFO

Keywords:
Tokamak
DAQ
Data storage
Distributed storage

ABSTRACT

As fusion experiment goes to steady state and high-performance data acquisition system been developed, traditional storage solutions cannot cope with the needs of fusion experiment data storage. Distributed file systems such as Lustre are being adopted by more and more facilities. The scaling-out of performance and capacity features addressed the main issue of traditional SAN (Storage Area Network) based storage. However, traditional file system based software solutions are still being used with the distributed file system. They lack modern data manage functions and limiting the performance of parallel storage. J-TEXT Cloud Database (JCDB) is a software stack that uses distributed database to provide fusion experiment data storage and management services. It ships with a storage engine powered by Cassandra database. This storage engine is designed for fusion experiment data and provides great performance. Data is divided into chunks when written and stored in a specially designed distributed database across the cluster. It has a MongoDB powered metadata management system which works seamlessly with the storage engine. JCDB is fully modular, handling different data type and metadata management functions are integrated as plugins. Even the storage engine can be changed. Though J-TEXT not supports long pulse experiments, the design of JCDB is aiming to meet long pulse discharge requirements, bring distributed database technology for future fusion devices such as China fusion engineering test reactor (CFETR). JCDB has the benefit of distributed file systems, provides complex metadata manage functions and comes with great performance.

1. Introduction

As more and more advanced diagnostic system and data acquisition system adopted by fusion experiment, as well as the experiment pulse gets longer, the fusion experimental data storage face a challenge. The fact that data produced by one shot grow tenfold in 5 years has been shown by multiple fusion facilities [1]. ITER is expected to generate quite large experimental data, up to 50GB/s, at peak times [2,3]. These issues not only challenge the performance and capability of future storage systems, but also the scalability.

Recently, there are researches in the application of distributed file system such as Lustre and GlusterFS in fusion experiment data storage [4–7]. The distributed file systems come with great performance and scalability, and they are compatible with POSIX (Portable Operating System Interface of UNIX) file system API (Application Programming Interface) [8]. This means the applications based on MDSplus and HDF5 can seamlessly migrate onto these types of storage [9,10]. The experiment data analysis tooling and workflow are well established on MDSplus and HDF5, which are very popular among fusion community [3,11]. But MDSplus and HDF5 are not designed for distributed file systems, they are optimized for block devices like hard drives or RAIDs (Redundant Arrays of Independent Disks). Distributed file systems are often based on a cluster with nodes connected via Ethernet, which differs a lot with RAID. MDSplus and HDF5 based applications cannot unleash the full power of distributed file systems without any optimizing. Moreover, implementing POSIX file system API on object storage based distributed storages will bring in overhead and little benefit for weakly structured data like fusion experiment data [12]. LHD once came up with a technology called IznaStor which is not a POSIX compatible storage solution for fusion experiment data. It stores data in a key-value pair cluster database. It shows many advantages over traditional file based solutions [1]. But it's kind of buggy and was dropped years ago.

Various kinds of diagnostics will generate different types of data. They are diverse in structure and metadata. Managing and accessing this huge amount of complex data efficiently is also a requirement for future storage system [13]. Relational database management systems (RDBMS) such as MySQL and PostgreSQL have its limitation to cope with these unstructured data [14,15]. It will be more difficult to change or update the schema. If some properties are used only a few times, but

* Corresponding author at: State Key Laboratory of Advanced Electromagnetic Engineering and Technology, Huazhong University of Science and Technology, Wuhan 430074, China.
E-mail address: zhangming@hust.edu.cn (M. Zhang).

https://doi.org/10.1016/j.fusengdes.2018.02.058
Received 9 June 2017; Received in revised form 1 February 2018; Accepted 16 February 2018
Available online 07 March 2018
0920-3796/ © 2018 Published by Elsevier B.V.

all records must contain the properties, leading to serious waste on disk space. In addition, MDSplus has limited metadata management function and HDF5 is just a file format that requires the metadata management and data access function to be developed from scratch. There are ways to search data by using information embedded in the scientific data, which is very useful if the information searched is hard to be stored in metadata [16]. However, it requires large amounts of computing power and it cannot replace the metadata managing function.

Attempting to address the above requirements, we designed a storage solution called J-TEXT cloud database (JCDB). It is a set of software that use NoSQL (Not only Structured Query Language) database to achieve management and storage of complex and large fusion experiment data efficiently [17]. It is a very flexible framework that can be expanded to work in many other scientific experiments. In the following sections the structure of JCDB and the implementation of some essential components will be presented, as well a performance test under 4 nodes JCDB cluster. Performance comparison to a GlusterFS distributed file system on the same hardware is also presented.

2. J-TEXT cloud database structure

Looking at the name of J-TEXT cloud database, one may get a few clues of what it is. J-TEXT [18] is the tokamak facility on which it is developed. Cloud database means data is stored in a cloud of computers, forming a NoSQL database cluster. JCDB is a set of software utilizing a computer cluster to provide experiment data management, storage and access services.

The block diagram of JCDB is shown in Fig. 1. JCDB core service is in charge of managing and accessing all the data stored in JCDB storage infrastructure, and other components are built around it. It implements metadata storage via storing the metadata as entities in the MongoDB [19]. The metadata for each experiment signal are organized in a hierarchical structure. Note that, MongoDB database only keeps the metadata, not the actual scientific data. The scientific data are stored in scientific data storage infrastructure by JCDB storage engines. The storage engine is controlled by the JCDB core service. The core service only defines an interface for the storage engine but has no concrete implementations, so user can implement their own different storage engine and inject into the core service. The core service also provides basic functions to manipulate and query the metadata. JCDB API is designed to integrate all essential parts of JCDB. It will create and manage JCDB core service instance and storage engine instances when

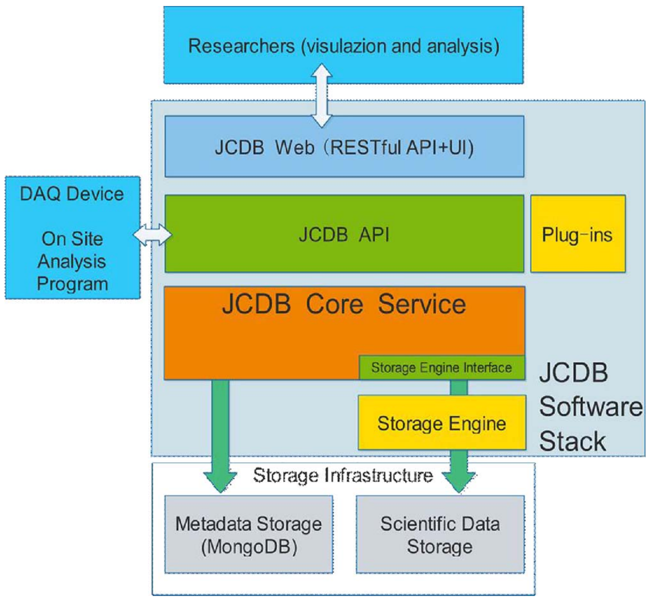

Fig. 1. The block diagram of JCDB software stack.

users try to work with JCDB. A few plugin interfaces are defined by JCDB API to provide data type support and metadata query support. JCDB API is also in charge of discovering, managing all the plugins and invoking the right one depending on the users' requests. The lowest level of Fig. 1 is the storage infrastructure for JCDB. There must be a MongoDB instance for the core service to keep all the structured metadata, but the storage infrastructure for actual scientific data depends on the implementation of storage engine. It can be a database or file system on a RAID. Storage engine normally stores the scientific data with a foreign key, pointing to its metadata objects for efficient writing and retrieval. The JCDB API is for the devices that require high performance such as acquisition devices and on-site data analysis programs. They can access to the JCDB storage infrastructure directly with IP address and port as long as they are authorized. The top level of Fig. 1 is called JCDB web, which is a web service running upon JCDB API. It provides a HTTP (HyperText Transfer Protocol) RESTful (Representational State Transfer) API and a few web base GUI (Graphical User Interface) for users that have no environment condition or permission to the JCDB storage infrastructure, which means they must use JCDB web instead of JCDB API.

The inversion of control pattern is used in designing the JCDB framework. The storage engine interface is defined in the core service, and the interface for various plugins are defined in JCDB API. This allow user to customize JCDB for different applications easily.

3. Data management and access in JCDB

Managing large sets of experiment metadata, searching and filtering the wanted signals are always demanding jobs. JCDB provides a flexible solution by using MongoDB to keep all the metadata.

3.1. Data model and management in JCDB

As mentioned above, JCDB split the experiment data into 2 parts, the metadata and the actual scientific data, as shown in the UML (Unified Modeling Language) in Fig. 2. There are 4 fundamental concepts of JCDB data model: experiment, signal, payload and sample. Experiment and signals are derived from entity object and they are defined and managed by JCDB core service. A signal is an entity that holds the metadata of a self-contained experiment data. For example, the plasma current waveform sampled in one shot is a self-contained experiment data, which means a set of data that include all information. That is to say, users can obtain all data they want without searching related information in other signal. An experiment can contain metadata of a set of related experiment and signal data. It should be noted that the signal does not hold any actual scientific data. JCDB core service provide basic operations to create, read, update and delete the experiments and signals. Furthermore, it stores all entities in a MongoDB collection and exposes a query interface, allowing user to build very powerful and complex queries to search experiments and signals with their metadata. The core service only defines the generic signal and experiment classes. With inheritance, users can implement their own types.

Unlike the metadata, the scientific data is unstructured. These data are stored by JCDB storage engine in unstructured way to reduce the overhead of structured data storages. The basic element or scientific data stored in JCDB is called a sample. A sample is just a value of certain type like a float number or an image. It often represents a value produced by a DAQ device on one sampling actions. It is the smallest data element in JCDB. Storage engine can fetch or write single or multiple samples, but it cannot fetch or write half a sample. Sample type can be specified on creating the signal. A set of samples is grouped together to form a payload, which is designed to improve the efficiency of a storage engine. Besides samples, a payload also carries some data about how sample are arranged, and most importantly, which signal does this payload belongs to. The signal entity and all the payloads