BESIII Physics Data Storing and Processing on HBase and MapReduce

Xiaofeng LEI\textsuperscript{12}, Qiang LI\textsuperscript{12}, Bowen KAN\textsuperscript{1}, Gongxing SUN\textsuperscript{1}, Zhenyu SUN\textsuperscript{12}

\textsuperscript{1}Institute of High Energy Physics, Beijing, China
\textsuperscript{2}Graduate University of Chinese Academy of Sciences, Beijing, China

E-mail: leixf@ihep.ac.cn

Abstract. In the past years, we have successfully applied Hadoop to high-energy physics analysis. Although, it has not only improved the efficiency of data analysis, but also reduced the cost of cluster building so far, there are still some spaces to be optimized, like inflexible pre-selection, low-efficient random data reading and I/O bottleneck caused by Fuse that is used to access HDFS. In order to change this situation, this paper presents a new analysis platform for high-energy physics data storing and analysing. The data structure is changed from DST tree-like files to HBase according to the features of the data itself and analysis processes, since HBase is more suitable for processing random data reading than DST files and enable HDFS to be accessed directly. A few of optimization measures are taken for the purpose of getting a good performance. A customized protocol is defined for data serializing and desterilizing for the sake of decreasing the storage space in HBase. In order to make full use of locality of data storing in HBase, utilizing a new MapReduce model and a new split policy for HBase regions are proposed in the paper. In addition, a dynamic pluggable easy-to-use TAG (event metadata) based pre-selection subsystem is established. It can assist physicists even to filter out 999\% uninterested data, if the conditions are set properly. This means that a lot of I/O resources can be saved, the CPU usage can be improved and consuming time for data analysis can be reduced. Finally, several use cases are designed, the test results show that the new platform has an excellent performance with 3.4 times faster with pre-selection and 20\% faster without pre-selection, and the new platform is stable and scalable as well.

1. Introduction

Physical analysis in High Energy Physics is a typical big data mining application. With the improvement of collider performance and data-taking efficiency, the volume of data accumulated over time grows linearly annually and the number of physical events has reached to $10^{10}$ magnitude in BESIII experiment. The traditional cluster architecture is a kind of “Data-to-Computation” architecture, where the data is stored and managed centrally and separated from computing nodes, so a huge volume of data will be moved from the storage cluster to computation cluster over network when physicists do physical analysis. So the traditional cluster architecture faces a strait challenge as the rapid increase of the amount of data poses to. In order to address this issue, researchers have tried to apply big data processing solutions (Hadoop\textsuperscript{[1]}), invented by internet community, to data processing in high-energy physics experiments. For example, University of Nebraska has been trying to use HDFS\textsuperscript{[2]} as storage system in Tier 2 of WLCG and there actually have been several US Tier2 sites adopting the technology, University of Iceland has been studying on replacing PROOF (Parallel ROOT Facility)
with MapReduce \cite{3} to conduct parallel analysis, and ATLAS has been studying at making use of HBase \cite{4} to store data files or metadata of dataset and so on.

The paper introduces the technical background firstly and then proposes the new data storing and processing platform based on HBase and MapReduce. Finally, the paper designs several use cases to validate the performance when working on the new platform and explains the reasons of the results.

2. Background

In past years, we have been studying on how to apply Hadoop to high-energy physics analysis and have already achieved some good results. A “Computation-to-Data” cluster for HEP (High Energy Physics) data analysis on Hadoop platform was designed and implemented. It has not only improved the efficiency of data analysis, but also reduced the cost of cluster building so far. However, there are still some deficiencies existing.

For example, by that time, we only provided some static patterns for pre-selection. Several index schemas need to be created according to users’ general data analysis behavior, this means users’ specific needs may not be satisfied and users themselves cannot change the schema as soon as their requirements change. This is very inflexible and unfriendly for physicists to do data analysis. In addition, there may be some overlaps existing among different index schemas, which lead storage spaces to be wasted due to the index duplications.

The more serious problem is caused by traditional data storage method and FUSE. At present, events are usually kept in DST (Data Summary Tapes) files and the in-memory data structure of physical events is TTree (Figure 1 \cite{5}). A tree uses a hierarchy of branches, and each branch can be read independently from any other branches. Each branch has a relevant buffer called Basket allocated for I/O operation and Basket is also the minimum operating unit. There are many events kept in one Basket and the application needs to load all Basket data into memory in one-time, decompress and deserialize them for enabling them to be recognized by program. Therefore, if neighboring events are not interested, all the resources, like I/O and CPU, used to process those data, will be wasted. So how to design the size of the Basket is really important for the performance of using TTree-like data storing \cite{6}.

![Figure 1 Data structure of TTree](image)

TTree is stored in TFile (the C++ class corresponding to DST file). Application uses TFile::Open(filename) method to open TFile with local path as parameter. So according to the features and DST file’s I/O characteristics, a third-party software called FUSE \cite{7} has to be used to access the DST files stored in HDFS. Although FUSE can enable HDFS to be accessed as a local file system, it will cause I/O performance loss. Because when FUSE is used to access data, multiple context switches between user state and kernel state will happen. These will cost a lot of resources and consume much time. On the other hand, FUSE needs to ask the master node for exact location of a file in HDFS whenever the file is accessed. So the more files will be accessed, the lower the I/O performance will be.
3. Design and implementation

To change the situation pointed above, the paper proposes an HBase based PaaS (Platform as a Service) platform for BES\[III\] physics data storing and processing. The architecture of the platform is shown in Figure 2.

The philosophy of platform’s design can be summarized as followings. Use HBase to store events data instead of DST files transparently and connect with the existing high-energy physics framework seamlessly. This means that user application programming interfaces should not be changed, users do not need to modify their original codes or to learn the relevant knowledge of HBase. Moreover, we try to save analysis results into .root files, so that users can use ROOT toolkit to do further analysis or draw histograms directly. According to the features of high-energy physics analysis, the platform use event metadata to build a flexible TAG-based pre-selection service to decrease the volume of data that needs to be processed and speedup the data analysis procedure.

3.1 Data storing

HBase is a distributed NoSql database run on HDFS. HBase can access to HDFS directly and be used as input data source. In ideal case, HBase can achieve a high concurrency random read performance by using the whole cluster’s capacity. It uses key-value (Figure 3) to store data and uses row key, column key and timestamp to build index and stores key-values in HFiles.

![Figure 2 Platform architecture](image)

![Figure 3 Data structure of key-value in HFile](image)
performance of HBase. The magnitude of differences between the amount of TAG data and the amount of event data is great, so we store TAG data and event data in two tables and use DST file’s name and event id to establish the mapping relation between these two tables (Figure 4). Event table’s row key is DST file name processed by MD5 plus event id. TAG table’s row key is DST file name processed by MD5. The value of TAG table is a serialized map (a kind of data structure in C++) object. The key of the map object is event id and the value of the map object is event metadata. The size of the TAG data of each event is only about 12Bytes. Since without doing event pre-selection, the whole TAG data of a DST file need to be loaded into memory and then be processed for the sake to promote the efficiency, so we put all that data in one row, so that they can be gotten in one RPC. To get a better performance, some optimization measures are taken. For example, turn off WAL mechanism, pre-split the table, build row-based bloom filter on each column family, prolong the cycle of compaction and increase the size of write buffer and so on.

---

**Figure 4 Mapping relations between tables**

Information produced by particles collision in detectors will be transferred to offline computing system and be endowed real physical meanings through reconstruction procedure. The reconstructed data corresponds to a version each time and every version has its own data structure. The paper takes version 655 as an example to explain the in-memory data structure of DST files. The TTree object has seven main branches, including TEvtRecObject, TDstEvent, TDigiEvent, TEvtHeader, THltEvent, TMcEvent and TTrigEvent (the paper calls the objects of these classes as ROOT object). A TTag class is added to store event metadata. The platform will serialize them into string object when putting data into HBase and deserialize them into corresponding ROOT object when load them back to memory.

**Figure 5 Format of HEPProtocol**

When importing data from DST files into HBase, the volume of data will expand due to the database schema information and the duplications of illustrative information of event data structure produced by common serialization protocol. But for the given version, the data structure is fixed and known, it is not necessary to store that information. Based on this fact, a specific memory data transfer
A protocol called HEPProtocol for BESIII reconstructed data to serialize/deserialize data is designed and implemented. We try to store data as little as possible. For the data which are of the same type but are used in different ways, we take different methods to process them. The format of HEPProtocol is shown in Figure 5. The principles of design and implementation of HEPProtocol can be extended to other physical experimental data. To further reduce the size of data, SNAPPY [8], a compression algorithm provided by HBase, is applied to tables.

3.2 Data Processing

HBase is developed in Java, while high-energy physics framework, like ROOT and BEAN, is developed in C++. Thus, the platform needs to support C++ to access Java. Originally, HBase provides a set of Thrift-based interfaces for C++ programmers. Thrift [9] is a kind of RPC framework based on socket. When the amount of data is too big, it will require higher configuration in both software and hardware platform. Therefore, a set of JNI-based interfaces to support the cross-language development as an optimizing way (Figure 6) is provided. Compared with Thrift, it can get processing speed faster, and CPU utilization higher and memory occupation less through using JNI [10] to access HBase.

![Figure 6 Schematic diagram of JNI communication](image)

The process of high-energy physics analysis involves a great deal of data. However, a high proportion of these data will be filtered out during the first step or the second step of analysis; it means a lot of resources of I/O and CPU will be wasted with no pre-selection. To solve this problem, the platform provides a flexible pluggable TAG-based event pre-selection service. The workflow of event pre-selection is shown as Figure 7.

![Figure 7 Flow chart of event pre-selection](image)

There are no relations among different DST files or events stored in the same DST file. We use MapReduce to analyse data in parallel. The MapReduce process is shown in Figure 8. The list of file
names is used as the input parameter and the input data is split by file name. Each map task will start a bean.exe thread to analyze the data whose row key prefix is the processed file name contained in the given split. The intermediate results will be stored in Lustre file system. When all map tasks are finished, a reducer task will start and merge the intermediate results into one .root file. Then move the final result to user’s output directory.

**Figure 8 Schematic diagram of MapReduce**

### 4. Evaluation

The analysis program is developed on ROOT 5.34/05, BEAN, HBase 0.94.2 and MapReduce0.20. The testing environment is set up with 6 nodes, every node with 8 cores CPU of 2.4GHz, 24GB memory and 1000M Ethernet card. The platform performance is evaluated by analyzing the real BESⅢ experimental data. The testing program is Rhopi which is used to analyze the process of \( J/\psi \to \rho \pi \).

The pre-selection conditions are the number of good +/- charged tracks, the number of good +/- K meson and the number of good photons. Several MapReduce use cases from different aspects are designed to evaluate the performance of the platform.

**Figure 9 Analysis speed comparison**

We call the scenario as case A when data is stored in HBase and the scenario as case B when data is stored in DST files. Figure 9 shows that the times of event selections were cost in case A and case B as the number of computing nodes increasing. Figure 10 shows that the processing time raises in case A and case B as the amount of data increasing, when the number of computing nodes is fixed. Both of these results show that when using event pre-selection, case A can get a 3.5 times faster than case B and the performance is improved significantly. Another test shows that when not using event pre-selection, case A can get at least 20% faster than case B. The reasons of this result can be attributed to two points: 1) HBase enable HDFS be accessed directly without FUSE, 2) HBase has a better random read performance compared to DFS files.

### 5. Conclusion

The paper proposes a new data storing and processing platform for BESⅢ physical analysis based on MapReduce and HBase. The platform fixed some deficiencies of the current system, like I/O bottleneck caused by FUSE and inflexible pre-selection method. The use cases show that the new platform is faster, stable and scalable.
The next work is to investigate and classify user’s behaviors, and try to establish the relationship between user’s behaviors and data collections. Finally, this information would be used to build secondary index to further optimize the platform performance.

6. Acknowledgment
This work was supported by the National Natural Science Foundation of China (NSFC) under Contracts No. 11375223 and also supported by Xie Jia Lin foundation of IHEP under Contracts No. Y5546140U2.

References
[1] White T. Hadoop: The definitive guide[M]. "O'Reilly Media, Inc.", 2012.
[2] Borthakur D. The hadoop distributed file system: Architecture and design[J]. Hadoop Project Website, 2007, 11(2007): 21.
[3] Holmes A. Hadoop in practice[M]. Manning Publications Co., 2012.
[4] Vora M N. Hadoop-HBase for large-scale data[C]//Computer Science and Network Technology (ICCSNT), 2011 International Conference on. IEEE, 2011, 1: 601-605.
[5] Brun R, Rademakers F, Panacek S, et al. The ROOT Users Guide[J]. CERN, http://root.cern.ch, 2003.
[6] van Gemmeren P, Malon D. Persistent Data Layout and Infrastructure for Efficient Selective Retrieval of Event Data in ATLAS[J]. arXiv preprint arXiv:1109.3119, 2011.
[7] Cheng Z, Zhang D, Huang H, et al. Design and Implementation of Data Encryptionin Cloud based on HDFS[J]. 2013.
[8] Zhang L, Chen Q, Miao K. A Compatible LZMA ORC-Based Optimization for High Performance Big Data Load[C]//Big Data (BigData Congress), 2014 IEEE International Congress on. IEEE, 2014: 80-87.
[9] Dimiduk N, Khurana A, Ryan M H, et al. HBase in action[M]. Shelter Island: Manning, 2013.
[10] Gordon R, Essential J N I. Java Native Interface[J]. Prentice Hall PTR, 1998.