In-Memory Parallel Computing for Partial Wave Analysis

Zhanchen Wei^{1,2}**, Qiulan Huang**^{1}**, Gongxing Sun**^{1}**, Xiaoyu Liu**^{1,2}

1 Institute of High Energy Physics, 19B Yuquan Road, Beijing 100049, China
2 University of Chinese Academy of Sciences, 19A Yuquan Road, Beijing 100049, China

E-mail: `weizc@ihep.ac.cn`

Abstract. The traditional partial wave analysis (PWA) algorithm is designed to process data serially which requires a large amount of memory that may exceed the memory capacity of one single node to store runtime data. It is quite necessary to parallelize this algorithm in a distributed data computing framework to improve its performance. Within an existing production-level Hadoop cluster, we implement PWA algorithm on top of Spark to process data storing on low-level storage system HDFS. But in this case, sharing data through HDFS or internal data communication mechanism of Spark is extremely inefficient. In order to solve this problem, this paper presents an in-memory parallel computing method for PWA algorithm. With this system, we can easily share runtime data in parallel algorithms. We can ensure complete data locality to keep compatibility with the traditional data input/output way and cache most repeatedly used data in memory to improve the performance, owe to the data management mechanism of Alluxio.

1. Introduction

Partial Wave Analysis[1] is a technique for physicists to solve scattering problems, which requires a large number of iterative and complicated fitting operations in high-dimensional space on large statistics to make the result more accurate. Partial Wave Analysis is a computing-intensive scientific application with tons of intermediate data generated during runtime to get the fitting result. With the increment of data volume and computing accuracy, traditional calculation models and algorithms are no longer sufficient. Using new computing methods has become the primary goal of improving efficiency of Partial Wave Analysis. Fortunately, each fitting operation of the analysis is independent of others. Therefore, parallelization of this type of analysis that many physicists are now working on becomes the main solution.

At present, there are two ways available based on existing infrastructures for algorithm parallelization. One is using multiple CPU threads and the other one is using GPU card. Although both solutions are reasonable for specified algorithms and have advantages for improving efficiency, there are still shortages in parallel Partial Wave Analysis with some unmet challenges. Multi-threaded algorithm mostly runs on a high performance server (also known as fat node) with a lot of computing resources, such as a 4-way SMP server with more than 120 CPU cores and more than 160 Gigabytes(GB) memory, to achieve an ideal performance improvement. This type of algorithm requires much memory allocated frequently during runtime, which may cause extra memory access and management latency. GPU algorithm and data structures require to be well-designed and well-optimized according to architectures of GPU card to maximize

performance. Transferring data between memory and GPU memory will bring extra latency due to the limit of GPU memory. Both parallel algorithms and traditional ones are fragile with no fault-tolerance. Calculated data will be lost and procedures cannot be recovered when the system is down and unavailable due to some errors.
This paper explores some new techniques and architectures to overcome the challenges above. We introduce Apache Spark, Alluxio and some other in-memory computing and cluster computing technology into Partial Wave Analysis to make full use of computing resources in clusters and improve the efficiency. With the help of clusters, we can achieve five main benefits which are good scalability, wide applicability, high availability, better performance and lower cost. On top of Spark and Alluxio, we propose a distributed in-memory computing platform and a new parallel Partial Wave Analysis algorithm to illustrate this platform's potential in scientific use case.

2. Background of Related Technology

With the era of Big Data emerging, a lot of new techniques and methods for dealing with big data have developed rapidly in IT industry. Google releases papers about GFS, MapReduce and BigTable. An open-source implementation of such a framework above called Hadoop is maintained by Apache Software Foundation (ASF). Hadoop provides us a computing model, which is moving computing to data. With this model, we can reduce network IO load in the cluster. University of California, Berkeley proposes a framework called Spark to accelerate data processing. Spark quickly becomes popular and now gets maintained by ASF. Spark has been widely used in IT industry as well as in many scientific fields[2][3][4], such as machine learning, hydrology, biology, health and life, remote sensing and so on. High energy physics also explores Spark[5]. Spark-ROOT project aims at processing ROOT files in Spark, which is maintained by CERN. Reference [6] introduces the work of processing data of NOvA experiment in Spark.
Spark is a lightning-fast open-source unified analytics engine[7] and suitable for both data-intensive and compute-intensive applications. Spark provides a parallel data processing model called Resilient Distributed Dataset (RDD). All the data are saved in RDD and automatically divided into several partitions. Each partition of an RDD can be calculated in parallel independently and data in RDD cannot be modified to ensure fault tolerance. RDD can be created by parallelizing existing data in the driver program, by referencing dataset in external storage system or by transforming from another RDD. There are narrow and shuffle dependencies between RDDs through transformation. Spark generates task Directed Acyclic Graph (DAG) by analyzing RDD dependencies and launch tasks on Executors. Figure 1 shows the Task Flow of Spark and PWA.
Alluxio is an open-source distributed virtual storage system that unifies and accelerates data access to different under file systems[8]. It is designed as a bridge with master-slave architecture between computing frameworks and storage systems, as it is illustrated in figure 2. The solid arrows stand for data interactions among services and the dashed arrows stand for instruction interactions and requests/replies. Alluxio Master manages all the metadata of the system including file metadata, block metadata and worker metadata. A client interacts with the master to read or modify this metadata. Alluxio workers are responsible for managing user-configurable local resources allocated to Alluxio(e.g. memory, SSDs or HDDs). Workers store data as blocks and serve client for reading and writing data. With these features, alluxio can unify memory spaces of different servers, provide data locality and automatically cache data in local file system. Alluxio also gives us the possibility to keep compatibility with traditional HEP programs.

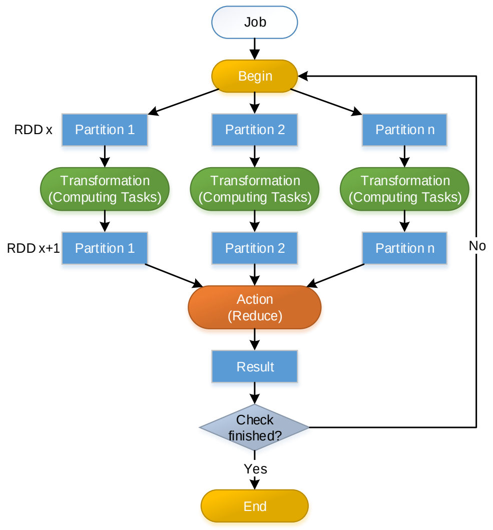

Figure 1. Spark Computing Flow

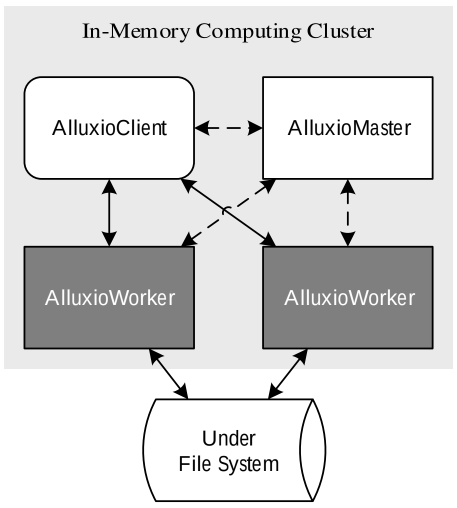

Figure 2. Alluxio System Architecture

3. Design and Implementation

3.1. Platform Architecture and System Design

In order to solve the problems mentioned in Section 1, we have designed and developed a more powerful in-memory computing system based on Spark and Alluxio for Partial Wave Analysis and some other similar data processing applications. The platform has been divided into five parts, as it is shown in figure 3. In this platform, all the persisted data will be stored in distributed file systems like HDFS, Lustre and EOS, and will be automatically cached in Alluxio while being accessed. All the volatile or temporary data will be stored in Alluxio. Resources of the whole cluster is managed by YARN. Jobs written in multi-languages can be submitted to Spark with Spark LLVM, JNI and LuaJIT plugins' support.

With the help of this platform, we can easily design and implement our new version of Partial Wave Analysis. The original PWA algorithm is split into sub-algorithms for individual event or event blocks. Meanwhile, data that algorithms analyzes are also split into corresponding blocks. Therefore, we aim at solving three main key technical difficulties when running PWA on Spark.

3.2. Storage of HEP data in memory

The key of parallel computing is processing data fragments concurrently. RDD partitions is a data splitting mechanism provided by Spark. We split HEP data into several blocks corresponding to RDD partitions according to the computing capability of a single server in the cluster and the scale of the whole data. All blocks of HEP data can be processed parallel.

RDD is an object-oriented data structure which can be treated as an array or a map of key-values. Due to the limitations of the language and Java Virtual Machine (JVM), there are still large restrictions in Java or some other languages based on JVM on the storage of complex data types and running complex calculations, especially some array or matrix operations. Therefore, we propose several storage strategies for partitioned data according to volume and characteristics of the data.

For simple physics events

Physics events with simple data layout will be directly stored as Java Objects in RDD to reduce the complexity of programming. Users can benefit from Spark by focusing only on the logic of

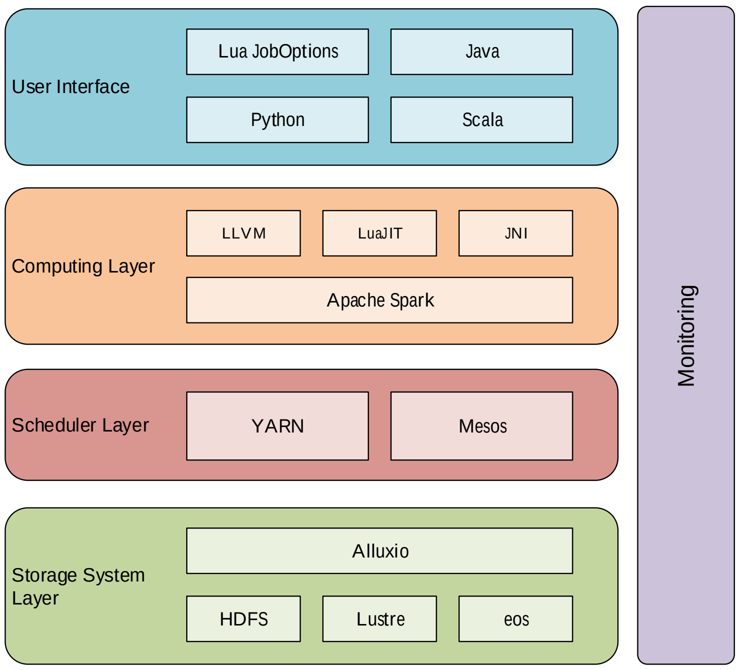

Figure 3. In-Memory Computing System Architecture

program and the implementation of algorithms. The structure of a typical simple physics event is shown in table 1.

Table 1. Data layout of a typical simple physics event

Data members	Data type	Root Description
f0_1_mass	double	p1x
f0_1_width		p1y
f0_1_mag		p1z
f0_1_phase		p1e
f0_2_mass		p2x
f0_2_width		p2y
f0_2_mag		p2z
f0_2_phase		p2e

For complex physics events
Complex physics event data have been divided into meta-data part and actual data part, as it is illustrated in figure 4. The meta-data stored in RDD describe basic attributes of each partition of HEP data. With these meta-data, Spark can to generate DAG and launch processing tasks. The actual data are stored in a specific file format which contains a set of meta-data fields to describe HEP data(see figure 5) in Alluxio. There is a header with a series of member at the beginning of this file to check with its meta-data. This format pre-defines the storage layout of common data types so that data layout keeps consistency in memory and files.

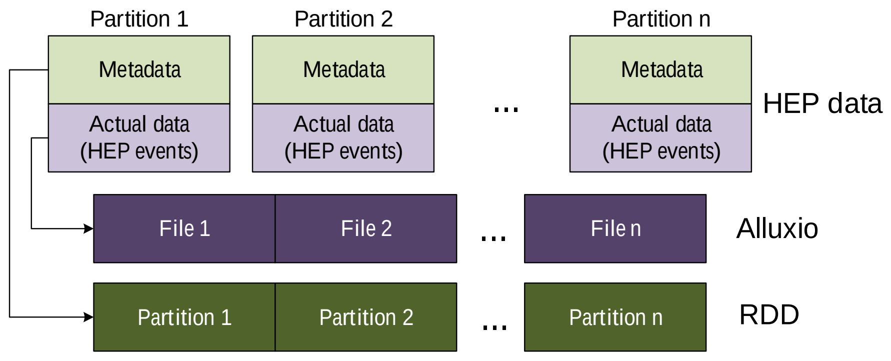

Figure 4. HEP data partitions in Spark

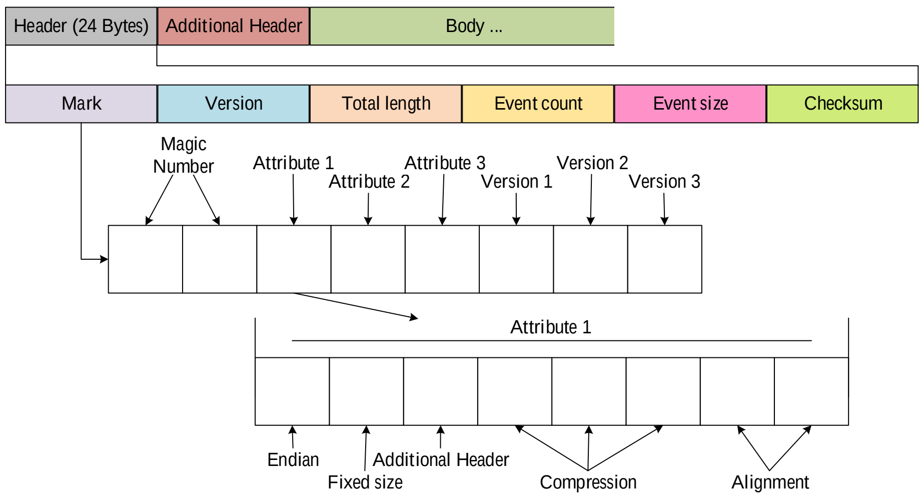

Figure 5. Data structure of actual data format

Storing actual data in Alluxio is a good way for us to manage and share HEP data across programming languages and data processing applications. User programs and procedures written in different languages are able to directly access data in Alluxio bypassing Spark and JVM. Large amount of actual data in Alluxio can help to decrease garbage collection time and avoid out-of-memory error and some other restrictions of JVM so that we can greatly improve performance. Through file persistence in Alluxio, we can reduce data redundancy across applications with the same input data.

For global and invariable data

This type of data, such as particle data tables, will not change during the physics calculations. Most of this data can be shared across calculations and data process applications. Therefore, global and invariable data are stored in a public database for access in a distributed environment.

3.3. Access to HEP data

In this platform, HEP data are stored in HDFS or cached in Alluxio to speed up processing. However, it is still difficult to get legacy applications to run efficiently on the platform. The main reason which lead to the difficulty mentioned above is random access is not supported

neither by HDFS nor by Alluxio. HDFS and Alluxio only provide Java-based interface so that traditional HEP programs cannot directly accessed data on them.
Therefore, after a thorough analysis of HDFS and Alluxio's data reading and writing principles, a new random access method is proposed, and on this basis, memory mapping technology is used to provide a high-performance data access interface. We provide a unified file service module for Alluxio and HDFS to provide complete POSIX access and data locality. The core idea of this module is to change the time of original data access and caching. Accessing data in Alluxio is transformed to accessing data in local memory file system to provide features of data random access and high-performance reading and writing. Generic reading and writing procedures are shown in figure 6 and figure 7.

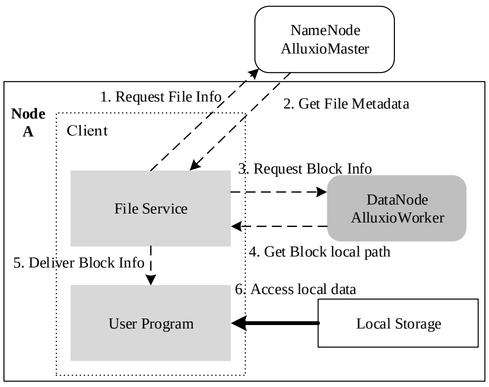

Figure 6. Generic Read Process

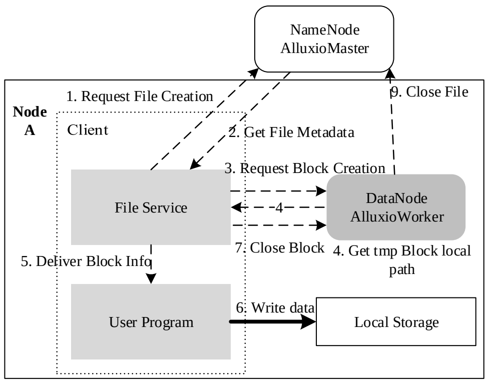

Figure 7. Generic Write Process

3.4. How PWA program running on Spark

Spark only supports a few programming languages such as Scala/Java, Python and R, while traditional HEP programs are mostly written in C/C++. It is a great need to make more languages available running on Spark in order to move to the new platform simply and quickly. The new PWA on Spark is implemented in Java and C++ through JNI interface. Java is used to prepare data and dispatch tasks and C++ is used to do data processing. JNI is an effective way of combining Spark with other languages and JNI is the only way for JVM to interact with an external runtime.
However, development with JNI is tedious and inflexible. We provide a full runtime environment integrated with LLVM and LuaJIT by JNI to make it possible to support multi-languages in the platform. Users focus on calculating and data processing instead of taking too much effort on transforming the traditional system to Spark. Submitting and running jobs simply and quickly make the platform more flexible and applicable.

4. Tests and Results

In order to test the distributed in-memory computing platform and the new version PWA program based on the platform proposed in this paper, we setup a 5-node cluster with one master node and 4 worker nodes (see table 2). Hadoop-2.7.5, Spark-2.4.0 and Alluxio-1.6.1 are deployed on this cluster with HDFS NameNode, Spark Standalone Master and AlluxioMaster running on the master node and HDFS DataNode, Spark Standalone Worker and AlluxioWorker running on the worker nodes. We have 4TB storage space on HDFS and 72GB storage space on Alluxio. Spark jobs are scheduled by Spark Standalone Scheduler.

Table 2. Test Environment

Environment	Master Node	Worker Node
CPU	Intel Xeon E5620, 2.40 GHz, 8 cores	
Memory	24GB	
Storage	320GB SAS HDD	2*2TB SAS HDD
Network	Gigabit Ethernet	
OS	CentOS 6 2.6.32-754.11.1.el6.x86_64	
Java Version	openjdk version "1.8.0_201"	

Due to the computational characteristics of Partial Wave Analysis, Spark task distribution and communication delay caused by a large number of iterations will greatly affected the performance of the whole application. There is still a large space for us to improve the performance of Spark and make this in-memory computing system better. We carry out a preliminary test of PWA on Spark. After the test, we get a 13-15 performance improvement compared to the traditional serial PWA (see table 3) which does not reach our expectation. We are going to focus on proposing a new computing model for Spark to deal with such a situation and optimizing Spark communications.

Table 3. Performance Comparison

Number of events (3KB each)	Serial PWA	Spark PWA
200000	1.7min	0.9min
400000	3.9min	1.2min
2000000	22.1min	1.7min
4000000	48.8min	3.4min
20000000	274.6min	18.3min

5. Conclusions

Spark and Alluxio have great application in High Energy Physics. Spark can be a good and cost-effective solution available for such big data or big runtime data application. Spark cluster provides good scalability, wide applicability, high availability and increased performance. Alluxio can act as cache as well as storage system to make HEP computing achieve good analysis efficiency and significantly reduce data I/O. In-memory computing system based on Spark and Alluxio is suitable for many occasions which require iterative computing. In addition, Alluxio, as a cache system, can be applied in applications requiring repeated access to data.

6. Acknowledgments

This work was supported by the National Natural Science Foundation of China (NSFC) under Contracts No. 11775249 and No. 11875283.

References

[1] Niklaus Berger, Liu Beijiang and Wang Jike, 2010, Partial Wave Analysis Using Graphics Units *Journal of Physics: Conference Series 219* 1-7

[2] Jesus Maillo, Sergio Ramíreza, Isaac Triguerое and Francisco Herrera, 1 February 2017, kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data *Knowledge-Based Systems* **117** 3–15

[3] Dries Harnie, Mathijs Saey and Alexander E. Vapirev, 2017, Scaling machine learning for target prediction in drug discovery using Apache Spark *Future Generation Computer Systems* **67** 409–417

[4] Khoa Doan, Amidu O Oloso, Kwo-Sen Kuo, et al, 2016, Evaluating the Impact of Data Placement to Spark and SciDB with an Earth Science Use Case *Proceedings of 2016 IEEE International Conference on Big Data* 341-346

[5] Ilija Vukotic and Robert William Gardner Jr, 8-14 October 2016, Big Data Analytics Tools as Applied to ATLAS Event Data *Proceedings of CHEP 2016 Conference* 53

[6] S. Sehrish, J. Kowalkowski and M. Paterno, 2016, Exploring the Performance of Spark for a Scientific Use Case *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* 1653-1659

[7] Apache Software Foundation, Apache Spark^{TM}- Unified Analytics Engine for Big Data *http://spark.apache.org*

[8] Alluxio Open Foundation, Alluxio - Open Source Memory Speed Virtual Distributed Storage *http://www.alluxio.org*