Blaze: A High performance Big Data Computing System for High Energy Physics

Libin Xia[1,2], Wei Sun[1], Xiaoyu Liu[1,2], Gongxing Sun[1], Xiaowei Jiang[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: xialibin@ihep.ac.cn

Abstract. High energy physics (HEP) is moving towards extremely high statistical experiments and super-large-scale simulation of theory. In order to handle the challenge of rapid growth of data volumes, distributed computing and storage frameworks in Big Data area like Hadoop and Spark make computations easy to scale out. While the programming model based on in-memory RDD assumes that workload performs only local computation and rare message exchange, it's inefficient at some HEP use cases, because several scientific computations, such as partial wave analysis (PWA) and lattice quantum chromodynamics (LQCD), are based on numerical linear algebra and iterative algorithms that rely on message passing between tasks. In this paper, we present a computing system (Blaze) that modifies Spark to support OpenMPI, and performs as a unified system that integrates MPI in DAG and provides task scheduling policy. The results indicate that the insufficient expressiveness in Spark model are supplemented by inter-task message passing. Additionally, Blaze also empowers MPI with the ability of data-locality computing and provides a solution of fault tolerance.

1. Introduction

Large-scale parallel computing and big data analysis contribute to an in-depth understanding of physics by increasing the amount of data and the accuracy of computational precision. Nearly all of the calculations in particle physics are either computing-intensive or data-intensive or both. For example, Partial Wave Analysis (PWA) requires parametric fitting in extremely high-dimensional space on statistics[1], and lattice QCD calculation is focusing on solving large-scale sparse linear equations. Both of them use numerical linear algebra and iterative algorithms to solve the physical questions, and scatter/reduce operations or communications are also needed. So large-scale parallel computing technology based on Message Passing Interface(MPI) is adopted to improve the performance.

In order to deal with the increase in data volume and improve the computing speed, the design goals of the next-generation high energy physics computing systems should consider the massive data processing while satisfying increasingly diverse and complex use cases such as traditional scientific computing and machine learning. Accordingly, in this paper, we propose a new computing model derived from Spark to deal with high-performance big data computing problems in high energy physics, and design an in-memory computing system called Blaze based on Spark and OpenMPI. With the help of communications among tasks and efficient

memory access operations, our computing system provides a powerful combination of flexibility, performance, and ease of use for physicists to develop big data computing applications.

2. Related Research

The traditional high energy physics data analysis always uses multiple CPU threads and GPU cards to parallelize the application. In the case of computation-intensive applications, experts use MPI to improve performance on a larger scale. For example, USQCD software suite provides QMT (QCD Multi-Threading) and QMP (QCD Message Passing) modules based on OpenMP/MPI to perform high performance for Lattice QCD calculations[2], they also develop a library called QUDA to support GPU[3].

Beyond high-performance computing community, technologies in Big Data era are booming. With the development and evolution over the past 10 years, Apache Spark has become the de-facto standard framework for large-scale distributed data processing. One of the key contributions of Spark is the introduction of Resilient Distributed Datasets (RDDs), DataSets, and DataFrames that stored in memory for building a MapReduce-like programming model. Internal datasets represented by RDDs can be cached in memory for reuse, and the tasks within a job are distributed through a Directed Acyclic Graph (DAG) composed of their dependencies to improve the performance of iterative and interactive applications. Spark also provides many general-purpose data processing libraries based on this model, such as Spark SQL and MLlib. Therefore, it is convenient for users to easily implement complex data processing jobs. In the field of high energy physics, scientists also tried to introduce Spark into experimental data processing, including the data analysis program of Fermilab's neutrino oscillation experiment (NuMI Off-axis νe Appearance, NOvA)[4] and CMS dark matter experiment data analysis program[5]. However, the program performance lags behind that of MPI implementation. CERN also tries to construct a set of new ROOT data parallel processing tools by integrating Spark and ROOT, which supports interactive or semi-interactive data analysis[6].

3. Design and Implementation

3.1. Programming Model and System Architecture

The challenges in high energy physics computing technology mainly consists in the convenience of parallel programming and the efficiency of the program execution. The above two requirements are fully considered in Blaze' s design. As shown in Figure 1, the parallelization of high-energy physics programs can be easily implemented on multiple separated *partitions* of the dataset, and large applications can be cut into small *stages* by defining operators to execute on the dataset. Then, according to the dependencies of dataset partitions and stages, an application is packaged and formed as a DAG. Finally, the DAG will be placed into and run on the computing resources. To meet the requirements of computation performance, high-performance computing technologies has been adopted by Blaze, including hardware and software resources such as Infiniband, MPI and so on.

Full design of the system is shown in Figure 2. The computing engine based on Spark integrates OpenMPI to better fit high energy physics computing. Meanwhile, the engine is also responsible for the generation of the execution diagram for computing tasks and fine-grained task allocation. The overall resource scheduling is controlled by YARN, while persisted data are stored in HDFS. The datasets between different stages are uniformly cached in memory managed by BlazeMM, which aims to avoid overheads such as data replications and conversions in the dataflow. Therefore, HEP applications can be efficiently built and run on the system.

3.2. Task Communications in Blaze

As a dataflow system, Apache Spark implements a data-parallel interface where tasks can be distributed to different partitions within a dataset. Each task is a function that executes on

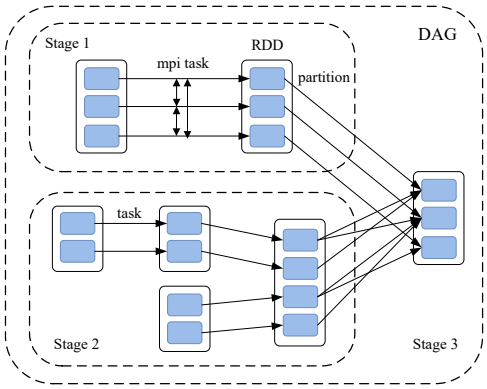

Figure 1. Blaze DAG execution model

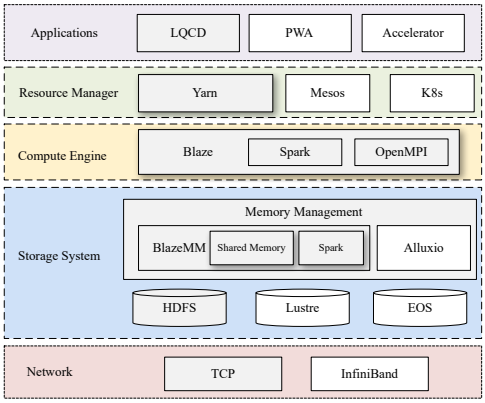

Figure 2. Blaze System Architecture

an independent data partition, so such MapReduce-like programming model is limited by the lack of inter-task communication. In Blaze, the low expressiveness of the traditional computing model in Spark is supplemented by message passing between tasks, which also establishes a bridge between high performance computing and big data area. Therefore, the message passing mechanism becomes a first-class component like the MapReduce model in Spark.

In practice, inter-task message passing is implemented through OpenMPI, which plays the role of task communication in user jobs. Blaze is responsible for the life cycle of the job, including the launching and scheduling of tasks to ensure that the application runs in a unified cluster, and the tasks invocation have the best data locality. After submission to the cluster, each job will create a *BlazeContext* instance which represents the connection to Spark and MPI. Scheduling decisions and other relevant information required by MPI, such as *namespace* and *ranks*, are created by the *DAGScheduler* and *TaskScheduler* in the Driver, which are then used to notify Executors to start tasks via the MPI library. The above processes implemented in Blaze are illustrated in Figure 3.

3.3. In-memory data management

As mentioned in Section 3.1, efficient data management is another challenge for computing system design. Caching internal data of computing lineage in memory and building a distributed data store as Spark are the key concepts of BlazeMM.

Spark stores datasets by splitting them into RDD partitions (blocks), which managed by Spark's *BlockManager* component. As Spark is running on Java Virtual Machine (JVM), this leads significant memory overheads for caching data. Although off-heap memory management has been introduced by Tungsten[7], which can help to reduce garbage collection overhead and serialization/deserialization cost. However, it mainly devotes to known object schema (eg. Spark SQLs), and cannot be used for sharing arbitrary data with MPI processes.

To meet the needs of supporting HEP native code and reducing performance loss between libraries or programming languages, we re-implement a distributed memory management module that inspired by Apache Arrow. The local object store is maintained through shared memory, so data can be shared between JVM process and MPI processes with zero-copy, reducing data format conversion and serialization/deserialization overhead. At present, we use RDD to record the *ObjectID* returned by the object store, which points to the actual data partition. The full implementation of Spark BlockManager replacement and more language-independent datasets implementation are still in progress. For example, the MPI operators executed in *MPIStage* will produce *MpiRDD* for further computations, which enhances MPI with data-locality computation and provides fault tolerant support.

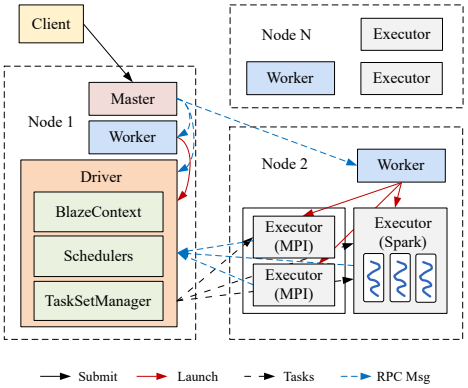

Figure 3. Schedule Flow of Blaze

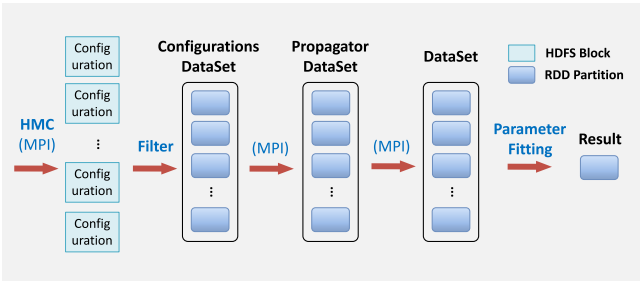

Figure 4. Data flow execution diagram of LQCD

3.4. HEP use cases

As a general computing system, a variety of applications can be built on the platform by following the rules of the programming model defined above. In this section, we choose the case of LQCD to illustrate the workflow in Blaze. LQCD calculations can be divided into three steps including generating configurations via Hybrid Monte Carlo algorithm, computing quark propagators and contracting propagators into correlation functions, and extracting physics. With the help of task communicating support, we can directly run the LQCD code written with MPI library. Therefore, the calculations will be converted to operations on data flow with multiple stages by DAG scheduler as shown in Figure 4, and the division granularity of the stages can continue to be reduced as needed. The result data is persisted in HDFS and the intermediate data between stages is temporarily stored or cached in distributed memory with key-value pairs RDDs. Due to the advantages of HPC and big data, the pre-processing stage like filtering with quark mass can be done by Spark operators, while computing intensive calculations such as propagators computation and contraction are implemented with MPI and processed in data-locality computation manner.

4. Tests and Results

In order to test the new programming model and the performance of the distributed in-memory computing system, we choose Conjugate Gradient (CG) algorithm as the test case in this paper, because iterative algorithms such as PWA are widely used in HEP computing. Meanwhile, CG is a method for solving linear equations that has been widely used in LQCD calculations.

Table 1. Test Environment

Environment	Master Node	Worker Node
CPU	Intel Xeon CPU E5-2630L v2, 2.40 GHz, 6 cores, 12 threads	
Memory	62GB	
Storage	2*2TB SAS HDD	
Network	Gigabit Ethernet	
OS	CentOS Linux release 7.9.2009	
MPI	OpenMPI 5.0.0a1	

The test environment is based on a cluster with one BlazeMaster node and 8 Worker nodes (Table 1), and the test case is solving a linear equation like $Ax=b$ with CG method, where A and b are stored as distributed matrices of RDDs. The implementation based on our programming model completes the parameter update in each iteration through `MPI_Allgather` or `MPI_Allreduce`, and makes all the calculation processes complete in a single Stage. Additionally, tasks running in Blaze obey the rule of data-locality. As a comparison, this paper also implements SparkCG which does not have task communication capabilities. The idea is to *cache* the distributed matrix A that needs to be reused in the iteration, and the parameters updated in each iteration are sent to Workers through *broadcast* to complete distributed matrix-vector multiplication and vector inner product operations in each task. The tests shows that we got at least 70% performance improvement compared with the traditional Spark programming model.

Table 2. Performance Comparison

Matrix dimensions	Spark CG	Blaze CG
10000	176s	12s
20000	251s	40s
30000	375s	92s
40000	560s	157s
50000	710s	220s

5. Conclusions

Many advanced ideas and projects from the Internet industry can be applied to HEP computing to make the data processing more powerful. The data-driven programming model inspired by Spark will make the work of physicists more intuitive and easier, but lacks flexibility and performance. In this paper, we enhance the RDD-based programming model with message passing and implement a prototype computing system with the capabilities of task communication and zero-copy memory access. Therefore, almost all applications in HEP can be represented by a high-performance dataflow model and perform data-locality computations with fault tolerance on Blaze.

6. Acknowledgments

This work was supported by the National Natural Science Foundation of China (NSFC) under Contracts No. 11775249, No. 12275295.

References

[1] Zhanchen W, Qiulan H, Gongxing S and Xiaoyu L 2020 *J. Phys.: Conf. Ser.* **1525** 012043
[2] Edwards R G and Joó B 2004 *arXiv: High Energy Physics - Lattice*
[3] Clark M, Babich R, Barros K, Brower R and Rebbi C 2010 *Computer Physics Communications* **181** 1517–1528
[4] Sehrish S, Kowalkowski J and Paterno M 2016 *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (IEEE) pp 1653–1659
[5] Gutsche O, Cremonesi M, Elmer P, Jayatilaka B, Kowalkowski J, Pivarski J, Sehrish S, Surez C M, Svyatkovskiy A and Tran N 2017 *arXiv: Distributed, Parallel, and Cluster Computing*
[6] Avati V, Blaszkiewicz M, Bocchi E, Canali L, Castro D, Cervantes J, Grzanka L, Guiraud E, Kaspar J, Kothuri P, Lamanna M, Malawski M, Mnich A, Moscicki J, Murali S, Piparo D and Tejedor E 2018 *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)* pp 5–6
[7] Project Tungsten URL `https://issues.apache.org/jira/browse/SPARK-7075`