A High Performance Collective Communication Library Layer for the Software Stack of Big Data Analytics

Student: Bingjing Zhang ([zhangbj@indiana.edu](mailto:zhangbj@indiana.edu))  
Advisor: Judy Qiu ([xqiu@indiana.edu](mailto:xqiu@indiana.edu))   
School of Informatics and Computing  
Indiana University, Bloomington, IN 47405

# Research Problem Statement

The landscape of Big Data processing tools have evolved rapidly in recent years and are divided into different branches based on the big data problems targeted for. Initially MapReduce [1] has proven very successful as a general tool to process many problems but is later considered not optimized for many important analytics, especially those machine learning algorithms involving iteration. In this regard, Iterative MapReduce (-like) frameworks such as Twister [2] and Spark [3] improve performance of MapReduce job chains through caching. At the same time, to solve graph problems, Pregel [4], Giraph [5] and GraphLab [6] [7] abstract data as a graph, cache and process it in iterations.

However both the data dependency and control dependency between parallel tasks are very complicated and varied in the iterative applications. This feature is not well considered in tools mentioned above which are designed with limited collective communication support to synchronize application data and algorithm control states among parallel processes.

For example, in k-means clustering with Lloyd’s algorithm [8], the training data (points) can be easily split and distributed to all the tasks. But the model data (centroids) has to be collected and redistributed to all the parallel tasks in the successive iteration. Mahout [9] on Hadoop chooses to reduce the outputs from all the map tasks in one reduce task, generate the new model, store it on HDFS, and let all the tasks read the model data back to memory. The whole process can be summarized as “gather-broadcast”. But when both the size of the model data and the number of tasks go large, this method becomes inefficient because the time complexity of this communication process is where is the number of tasks, is the number of centroids, and is the number of dimensions in each centroid. The overhead of this process can be reduced to if “regroup-gather-broadcast” is applied or even if “regroup-allgather” is applied [10]. The “regroup-gather-broadcast” method has been provided in Twister [11] [12] and Spark. But the best method “regroup-allgather” is not seen in any of these tools.

In many machine learning algorithms, such data movements to synchronize the model data between all the parallel tasks is essential. They can be executed one or more times per iteration, therefore their performance is crucial to the efficiency of the whole application. We call this type of data movement “Collective Communication”. In this way, these iterative algorithms which were previously expressed as a chain of MapReduce jobs can now be re-abstracted as iterations of high performance collective communication operations.

The collective communication abstraction layer introduced here, provides a set of efficient collective communication operations on several common data abstractions such as arrays, key-values and graphs, and define a MapCollective programming model which serves the diverse collective communication demands in different parallel algorithms. I implement a library called Harp to provide the features above and plug it into Hadoop so that applications abstracted in MapCollective model can be easily developed on top of MapReduce framework and conveniently integrated with other tools in Apache Big Data Stack. With improved expressiveness in the abstraction and excellent performance on the implementation, we can simultaneously support various applications from HPC to Cloud systems together with high performance.

# Major Results and contributions



Fig. 1. Major contributions in this research:   
MapCollective model with collective communication abstractions and the Harp library

There are three contributions in this research. (a) A collective communication library layer is abstracted and separated out from other layers in the software stack of big data processing. In this layer, a common set of data abstractions and related collective communication operation abstractions are defined. (b) On top of this abstraction layer we define the MapCollective programming model, which allows users to invoke collective communication operations to synchronize a set of parallel processes. (c) These ideas are implemented in the Harp open source library [13] as a Hadoop plugin. The word “harp” symbolizes how parallel processes coordinate through collective communication operations for efficient data processing, just as strings in harps can make concordant sound. By plugging Harp into Hadoop, we can express the MapCollective model in a MapReduce framework and enable efficient in-memory collective communication between map tasks across a variety of important machine learning applications (Fig. 1).

## Hierarchical Data abstraction

To support various collective communication patterns, the data types are abstracted in a hierarchy. In Fig. 2(a), data are horizontally abstracted as arrays, key-values, or edges and messages in graphs and constructed from basic types to partitions and tables vertically. Firstly, any data which can be sent or received is an implementation of interface Transferrable. At the lowest level, there are two basic types under this interface: arrays and objects. Based on the component type of an array, there can be byte array, int array, long array or double array. Object type is used to describe keys and values in key-value pairs, or graph data such as vertex, edge and message. Next at the middle level, arrays and objects are wrapped as array partition, key-value partition and tuple partition for edge tuples or message tuples. Notice that tuple partition is built from byte arrays but not from objects directly. When reading, bytes are deserialized to a tuple of objects. When writing, the object tuple is serialized to byte arrays. At the top level are tables containing several partitions, each with a unique partition ID. If two partitions with the same ID are added to the table it will solve the ID conflict by combining them into one. Tables on different processes are associated with each other through a collective communication operation. These tables are considered as one dataset and the collective communication operation is defined as redistribution or consolidation of partitions in this dataset.

## Collective Communication Operations

Collective communication operations are defined on top of the data abstractions. Currently three categories of collective communication operations are supported:

1) Collective communication adapted from MPI [14] collective communication operations: e.g. “broadcast”, “reduce”, “allgather”, and “allreduce”.

2) Collective communication derived from MapReduce “shuffle-reduce” operation: e.g. “regroup” operation with “combine or reduce” support.

 (a) (b)  
Fig. 2. (a) Hierarchical Data Abstractions (b) Layered Architecture

3) Collective communication abstracted based on local data relevance: e.g. graph based communication “send messages to vertices”

These collective communication operations can be used on any data abstraction. For graph based communication, because only the data relevance between partitions is recorded, these operations are not tied with graph data abstractions but can be used to array tables and key-value tables. Additionally, each collective communication can be implemented in a rich set of algorithms. We choose candidate algorithms for optimization based on two criteria: the frequency of the collective communication and the total data size in the collective communication. For the operation which most frequently occurs in the application, we choose the algorithm with high performance to reduce the cost on application data synchronization. With different data sizes, some algorithms are good for small data while others favor large data.

For example, we have two versions of “allreduce”. One is “bidirectional-exchange” algorithm [10] and another is “regroup-allgather” algorithm. When the data size is large and each table has many partitions, “regroup-allgather” is more suitable because it has less data sending and more balanced workload per process. But if the table on each process only has one or a few partitions, “bidirectional exchange” is more effective.

## MapCollective Model

MapCollective model is defined to enable using collective communication operations. This programming model follows the BSP style. There are two levels of parallelism. At the first level, each parallel component is a process where the collective communication operations happen. The second is the thread level for parallel processing inside of each process. Thread level parallelism is not mandatory in the model but it can maximize memory sharing and multithreading in each process and save the data size in collective communications. To enable in-memory collective communications, every process has to be alive simultaneously. As a result, instead of dynamic scheduling, static scheduling is used: when processes are scheduled and launched, their locations are synchronized between all the processes for future collective communications.

For fault tolerance, currently the effort is on failure detection to ensure every process can report exceptions or faults correctly without getting hung up. Failure recovery poses a challenge because the execution flow in the MapCollective model is very flexible. Currently job level failure recovery is applied. An application with a large number of iterations can be separated into a small number of jobs, each of which contains several iterations. This naturally forms checkpointing between iterations. Since MapCollective jobs are very efficient on performance, this method is feasible without generating large overhead. At the same time, task level recovery by resynchronizing execution states between new launched tasks and other old live tasks is also under the investigation.

## Layered Architecture

The current Harp implementation targets Hadoop 2 based on YARN. It contains several levels and components (Fig. 2(b)). At the bottom level is the MapReduce framework. The original MapReduce framework is extended to expose the network location of map tasks. In the upper level, Harp builds collective communication abstractions which provide collective communication operators, various types of tables and partitions, and a memory allocation management pool for data caching and reuse. All these components interface with the MapCollective programming model. After wrapping, the MapCollective programming model provides three components to the application level: a MapCollective programming interface, a set of collective communication APIs, and data abstractions which can be used in the programming interface.

## Performance

Our research in this area led to some unexpected discoveries. One was identification of the importance of the HPC-ABDS software stack [13, 5, 8] illustrated by Hadoop (with the Harp plug-in) which can run K-means, Graph Layout, and Multidimensional Scaling algorithms with realistic application datasets over 4096 cores on the IU Big Red II Supercomputer (Cray/Gemini) while achieving linear speedup [13] (see Fig. 3). This demonstrates the portability of HPC-ABDS to HPC and eventually exascale systems.

|  |  |  |  |
| --- | --- | --- | --- |
| K-means Cluster | Forced-directed Graph Drawing | WDA-SMACOF (MDS) | |
|  |  |  |  |
| Fig. 3. Performance of Hadoop/Harp Applications on the Big Red II Supercomputer | | | |

# References

1. J. Dean and S. Ghemawat. “Mapreduce: Simplified data processing on large clusters”. OSDI, 2004.
2. J. Ekanayake et al. “Twister: A Runtime for iterative MapReduce. Workshop on MapReduce and its Applications, HPDC, 2010.
3. M. Zaharia et al. “Spark: Cluster Computing with Working Sets”. HotCloud, 2010.
4. Grzegorz Malewicz et al. “Pregel: A System for Large-scale Graph Processing”. SIGMOD. 2010.
5. Apache Giraph. <https://giraph.apache.org/>
6. Y. Low et al. “Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud”. PVLDB, 2012.
7. J. Gonzalez et al. “PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs”. OSDI, 2012.
8. S. Lloyd. “Least Squares Quantization in PCM”. IEEE Transactions on Information Theory 28 (2), 1982.
9. Apache Mahout. <https://mahout.apache.org/>
10. E. Chan, M. Heimlich, A. Purkayastha, and R. Geijn. “Collective communication: theory, practice, and experience”. Concurrency and Computation: Practice and Experience 19 (13), 2007.
11. J. Qiu, B. Zhang. “Mammoth Data in the Cloud: Clustering Social Images”. In Clouds, Grids and Big Data, IOS Press, 2013.
12. B. Zhang, J. Qiu. “High Performance Clustering of Social Images in a Map-Collective Programming Model”. Poster in SoCC, 2013.
13. B. Zhang, Y. Ruan, J. Qiu. “Harp: Collective Communication on Hadoop”, short paper in IC2E, 2015.
14. MPI Forum. “MPI: A Message Passing Interface”. SC, 1993.

# Selected Publications

1. **Bingjing Zhang**, Yang Ruan, Judy Qiu. “Harp: Collective Communication on Hadoop”, short paper in the Proceedings of IEEE International Conference on Cloud Engineering, 2015.
2. **Bingjing Zhang**, Judy Qiu. “High Performance Clustering of Social Images in a Map-Collective Programming Model”, poster in the Proceedings of ACM Symposium on Cloud Computing, 2013.
3. Judy Qiu, **Bingjing Zhang**. “Mammoth Data in the Cloud: Clustering Social Images”, Clouds, Grids and Big Data, IOS Press, 2013.
4. Thilina Gunarathne, **Bingjing Zhang**, Tak-Lon Wu, Judy Qiu. “Scalable parallel computing on clouds using Twister4Azure iterative MapReduce”, Future Generation Computer Systems, 2013.
5. Thilina Gunarathne, **Bingjing Zhang**, Tak-Lon Wu, Judy Qiu. “Portable Parallel Programming on Cloud and HPC: Scientific Applications of Twister4Azure”, 4th IEEE/ACM International Conference on Utility and Cloud Computing, 2011.
6. **Bingjing Zhang**, Yang Ruan, Tak-Lon Wu, Judy Qiu, Adam Hughes, Geoffrey Fox. “Applying Twister to Scientific Applications”, 2nd IEEE International Conference on Cloud Computing Technology and Science, 2010.
7. Judy Qiu, Jaliya Ekanayake, Thilina Gunarathne, Jong Youl Choi, Seung-Hee Bae, Hui Li, **Bingjing Zhang**, Tak-Lon Wu, Yang Ryan, Saliya Ekanayake, Adam Hughes, Geoffrey Fox. “Hybrid cloud and cluster computing paradigms for life science applications”, BMC Bioinformatics 11, 2010.
8. Jaliya Ekanayake, Hui Li, **Bingjing Zhang**, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, Geoffrey Fox. “Twister: A Runtime for Iterative MapReduce”, Proceedings of 1st International Workshop on MapReduce and its Applications of ACM HPDC 2010 Conference, 2010.