CS 5204 Project Final Report
A Feasible Study of MPI-IO on Top of HDFS

Luna Xu (xuluna@cs.vt.edu) Adam Binford (adamq@vt.edu)

December 1, 2014

1 Team member

Luna Xu (xuluna)
Adam Binford (adamq)

2 Introduction

MapReduce [?] and its most popular implementation Hadoop [?] have become the dominant distributed processing framework for big data analytics. Despite of the ease-of-use and scalibility of Hadoop, researchers also found the limitations of Hadoop lie in for example, inter-process communication. For such limitations, the well established Message Passing Interface (MPI) [?] is more suitable due to its ability to support any communication pattern. X.Lu et al [8] find that the message latency of MPI is about 100 times less than Hadoop primitives. The average peak bandwidth of MPI is about 100 times higher than Hadoop RPC – the fundamental communication mechanism in Hadoop. Moreover, there exists data analytics workflows such as Metagenomics [?] that consist both compute- and communication-intensive computations. To better conduct such workflows and to avoid data movement between clusters [?], resource coordination platforms such as mesos [?], omega [?], YARN [?] enable different programming paradigms including MPI and MapReduce to co-exist in the same cluster. Though not realized yet, hosting MPI and Hadoop in the same cluster is highly promising as YARN claims to embrace MPI as a first class citizen.

One of the biggest challenges of this marriage is the underlying shared file system. As a big feature of MPI-2 standard [?], MPI-IO provides parallel IO support to MPI programs and enables MPI to process data-intensive workloads as well. Currently, such support requires an underlying network/parallel file system such as NFS [?], PVFS [?], Lustre [?], GPFS [?] to achieve the best performance. However, these file systems are focused on optimization for MPI-IO [?, ?, 7] and have a big network overhead on hosting Hadoop with the absence of data locality [9]. IBM's GPFS is originally designed as a SAN file system as the data is striped and placed in a round-robin fashion [?], which prevents it from being used in Hadoop. With a support of File Placement Optimization (FPO), GPFS-FPO makes it possible to efficiently support Hadoop. However, GPFS is shipped with IBM SP system and is not available as opensource as Lustre. Moreover, IBM tailors MPI-IO according to GPFS in their own MPI implementation [?], which is not supported in more wildly used implementations such as MPICH [] and OpenMPI [5].

Another solution is to support MPI-IO on top of the distributed file systems used by MapReduce such as GFS [] and HDFS [6]. HDFS is integrated inherently in Hadoop releases and is the default file system used in Hadoop community. By bringing MPI to HDFS, it is possible to keep existing applications in Hadoop ecosystem without any changes. C. Cranor et al [] explore the performance of MPI-IO on HDFS using PLFS. However, HDFS is supported as a component of PLFS and no data locality is achieved for MPI jobs. As far as we know, there is no such work on supporting MPI on top of HDFS directly. This project focuses on exploring the feasibility and performance of enabling MPI-IO on top of HDFS. This study is based on the observation that MPI-IO provides great flexibility that it is possible for users to decide the process-to-block mapping based on the information that HDFS provides.

Challenges of enabling MPI-IO on top of HDFS.

3 Project Progress

We have finished investigating the possible ways to mount HDFS as a regular file system that can be interacted with by any file I/O. We got the throughput for manual copy, native NFS as the ideal performance, fuse-dfs throughput for parallel read. However we could not perform parallel write using fuse-dfs. Table 1 shows the errors we encountered during our tries. We tried using `MPI_File_write_at` where each process holds an individual file pointer, as well as `MPI_File_write_shared` where all processes hold a shared file pointer. We open the file using different mode and with the combinations we get mainly two errors. The error we get from the `APPEND` mode is reported in the MPI program side, others are shown in the fuse-dfs side. Another method that we explored is Native HDFS Fuse [4], which utilizes only protobuf to communicate with Namenode directly. Hence no fuse or native lib is involved. However, the program dumped a segmentation fault when we tried to run. HDFS-NFS solution is also not successful nor desirable because either it has requirements for specific (2.3.0) Hadoop version [1] or it only supports the cloudera distribution of Hadoop [2].

We are now focusing on creating a library to hook MPI [3] I/O function calls to use the HDFS native library to interact with HDFS. Our goal is to allow unmodified MPI applications to interact with HDFS by simply loading our library at runtime. So far we have successfully hooked MPI functions at runtime, and verified our functions were being called. Additionally, we have read from and written to files in our running HDFS using the HDFS native library. When reading a single file from multiple processes, we have observed an increase in bandwidth when increasing processes. This confirms that multiple processes can read from the same file at once using the native library. To complement this work, we have developed scripts to compile and run these HDFS native library applications easily.

4 Future work

The final steps we have to do are implementing the necessary MPI I/O functions in our hooking library to use the HDFS native library as the file I/O method. We have already done each of these pieces individually, hooking and using the native library, we simply must combine them. The hooking functions need to be able to use the parameters they are given to seamlessly work with HDFS without the MPI program knowing anything is different.

Additionally, we must find out if it is possible to implement some support for writing to HDFS through the hooked MPI routines. The native library only allows appending to a file, and only one thread can access a file for writing at one time. We must either modify the behavior of the I/O of the MPI program or set restrictions on what MPI programs running on HDFS are allowed to do. Finally, we must simplify the scripts required for our solution to work to put as small of a burden on the user as possible.

References

[1] Hdfs nfs gateway. `http://hadoop.apache.org/docs/r2.3.0/hadoop-project-dist/hadoop-hdfs/HdfsNfsGateway.html`.

[2] hdfs-nfs-proxy. `https://github.com/cloudera/hdfs-nfs-proxy`.

Function	Mode	Error
MPI_File_write_at	CREATE|RDWR	cannot open an hdfs file in O_RDWR mode
MPI_File_write_at	CREATE|WRONLY	cannot open an hdfs file in O_RDWR mode
MPI_File_write_at	WRONLY	cannot open an hdfs file in O_RDWR mode
MPI_File_write_shared	WRONLY	cannot open an hdfs file in O_RDWR mode
MPI_File_write_shared	APPEND	file open. code: 201388309

Table 1: Error codes for parallel writes on fuse-dfs.

[3] Mpich. www.mpich.org.

[4] Native hdfs fuse. https://github.com/remis-thoughts/native-hdfs-fuse.

[5] Open mpi. http://www.open-mpi.org/. Accessed: 2014-08-11.

[6] Hadoop Distributed File System (HDFS). http://hortonworks.com/hadoop/hdfs/, 2014.

[7] Dickens, P., and Logan, J. Towards a high performance implementation of mpi-io on the lustre file system. In *On the Move to Meaningful Internet Systems: OTM 2008*. Springer, 2008, pp. 870–885.

[8] Lu, X., Wang, B., Zha, L., and Xu, Z. Can mpi benefit hadoop and mapreduce applications? In *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on* (Sept 2011), pp. 371–379.

[9] Rutman, N. Map/reduce on lustre-hadoop performance in hpc environments. *Langstone Road, Havant, Hampshire, P09 ISA, England, Tech. Rep* (2011).