On studying the performance of Hadoop Map Reduce vs MPI for Aggregation Operations: A Big Data Challenge

IT Revolution has been chiefly responsible for generating tremendous amount of data, often referred to as Big Data. Parallel and distributed computing systems have largely been utilized for solving problem involving big data. Of various distributed & parallel computing systems like vector processing systems, co-processors (e.g. GPU's), and others; Map Reduce and MPI (Message Passing Interface) have found increased attention in recent times. MPI has been a de facto standard of parallel programming for decades [11], while Map Reduce is of comparatively recent development. The major difference between both the platforms is that Map Reduce uses HTTP/RPC protocols for communication which are clearly slower than the MPI's communication (both point-to-point and collective). On the flip side, programming MPI is clearly more difficult and requires lots of skills (in programming, networking, domain knowledge, and others) than Map Reduce, which much simpler because programmer does not have to handle networking problems.

There are many data related computing tasks that needs to be accomplished. Of these, aggregation tasks are the most common. These are to summarize the data before proceeding with any type of analysis or visualization. Also, aggregation tasks are required by both the scientific and business community. The challenge here is to optimize the performance of MPI and Map Reduce for aggregation tasks.

For the given work, we would study the MPI performance on SHARCNET. MPI task optimization is usually based on the algorithm and the input data characteristics (like file system, size of the data, and others). Since, we have the data in multiple files of almost equal size (as discussed in Section 4), we find that representing the MPI problem of processing using Drinking Philosopher's problem and solving the same is one of the good solution. However, we have not been able to establish the fact to-date if an MPI program may access multiple files at the same time on SHARCNET cluster. If SHARCNET does not support such an operation, than we may have to read chunks of data into master and distribute the same for processing.

Also in the given work, we would study the optimization parameters of the Map Reduce implementation of Apache Hadoop on a single node setup. Optimization parameters for the Apache Hadoop are as given below:

1) Optimal size of the block: A block (AKA physical record) is a sequence of bytes or bits, usually containing some whole number of records. In Apache Hadoop, the recommended minimum block size is 64MB and may be increased to higher size. However, since each task is carried out on a block of data, a single failure may result into redoing the work for the data of the size of one block. Hence, keeping a large block size may not be ideal. For the given problem we wish to compare the time consumption for aggregation task when block size is 64MB and 128MB.
2) Optimal input-split size: While processing the data in blocks, each block is split into manageable sizes called input-split. The size of input-split is a critical factor that determines the total main memory that may be utilized by the Map Reduce function. The decision regarding input-split size is based on the minimum values set in the configuration files and a heuristic that determines a good size based on block size and other parameters or as set by the programmer/developer. These parameters may be controlled easily by using relevant functions in java Map Reduce API.
3) Optimal compression format: Data transmission using intermediate stages like between Mapper and Reducer tasks occur by re-writing output of the Mapper onto disk. Since, disk operations are 300 times slower than the main memory operations, compressing the data during intermediate stages may save both space and time. Since, data needs to be retrievable even after selecting only a portion of the compressed data. It becomes essential to compress using only splittable compressing techniques like LZO, bzip2, or snappy. Hence, for the given work we wish to test the optimal compression format for no compressing vs snappy compression technique.

Performance Comparison?

Dataset?

Issues that came up?

Bibliography

1. Map reduce, Sandia Labs, http://mapreduce.sandia.gov/
2. Hadoop Documentation, https://hadoop.apache.org/docs

3. NYC Taxi & Limousine Commission, New York http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
4. Lee, K. H., Lee, Y. J., Choi, H., Chung, Y. D., & Moon, B. (2012). Parallel data processing with MapReduce: A Survey. ACM SIGMOD Record, 40(4), 11-20.
5. Dittrich, J., & Quiané-Ruiz, J. A. (2012). Efficient big data processing in Hadoop MapReduce. Proceedings of the VLDB Endowment, 5(12), 2014-2015.
6. Doulkeridis, C., & NØrvåg, K. (2014). A survey of large-scale analytical query processing in MapReduce. The VLDB Journal—The International Journal on Very Large Data Bases, 23(3), 355-380.
7. Chen, C. P., & Zhang, C. Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. Information Sciences, 275, 314-347.
8. Wes Kendall, Launching an Amazon EC2 MPI Cluster. http://mpitutorial.com/tutorials/launching-an-amazon-ec2-mpi-cluster/
9. Lu, X., Liang, F., Wang, B., Zha, L., & Xu, Z. (2014, May). DataMPI: extending MPI to hadoop-like big data computing. In Parallel and Distributed Processing Symposium, 2014 IEEE 28th International (pp. 829-838). IEEE.
10. Kang, Sol Ji, Sang Yeon Lee, and Keon Myung Lee. "Performance comparison of OpenMP, MPI, and MapReduce in practical problems." Advances in Multimedia 2015 (2015): 7.
11. Jin, Hui, and Xian-He Sun. "Performance comparison under failures of MPI and MapReduce: An analytical approach." Future Generation Computer Systems 29.7 (2013): 1808-1815.