Improving I/O Performance by Coscheduling of I/O and computation on Commodity based Clusters

Saba Sehrish, Grant Mackey and Jun Wang
University of Central Florida
ssehrish, gmackey, juwang@eecs.ucf.edu

A fault tolerant system like Hadoop [1] running on a commodity based cluster schedules two types of tasks, *Regular Map/Reduce tasks* and *Speculative Map/Reduce tasks*. Regular Map/Reduce tasks are spawned when a MapReduce [3, 2] application is launched, whereas speculative tasks are instigated when there is a slow or a failed task. Hadoop's task scheduler exploits *data locality*, and tries to schedule the regular map tasks near data. By assigning one DFS block per map task, a large number of map tasks are generated that are cumbersome to manage and also impacts I/O performance. The number of map tasks can be reduced by increasing the split size from one DFS block to multiple DFS blocks. For example, random sort with 294 map tasks took 388 sec to finish, while 154 map tasks took 292 sec. A logical contiguous split may consist of DFS blocks that are spread across multiple nodes. The challenge is to schedule map tasks using the locations of multiple DFS blocks. Apart from improving performance, multiple blocks can also be combined in a split for applications like matching and clustering of long strings. In this case, blocks are statically combined into splits i.e. the application developer specifies the split size. The regular map tasks with multiple blocks are scheduled based on the data locality of these blocks such that the node with maximum block contribution becomes the host node for this task. However, the remaining blocks must be transferred from remote nodes but their I/O latency is not incorporated in the initial scheduling decision, and the remote node(s) for remaining blocks is not determined at that time.

We propose a scheduling algorithm for the regular map tasks, which considers data locality and I/O latency when more than one DFS blocks are assigned to a map task. Additionally, we will also combine our scheduler with LATE scheduler [4] for speculative tasks by introducing data locality for further performance gains. We have identified three different cases where scheduling can improve performance for regular map tasks as shown in the Figure 1, 2 and 3. **Scheduling when independent DFS blocks are combined statically, as a result of increase in split size to improve performance:** The file splits are provided by the application programmer, and each split consists of a number of DFS blocks. Our scheduler uses the location of participating blocks provided by HDFS's namenode to determine the candidate nodes. We introduce *virtual splits* for independent blocks on a node that can be combined logically into one split. We consider a cost for creating virtual splits, for example, the cost for three virtual splits is 2, etc. Similarly, there is also a cost for data transfer because we want to choose between virtual splits and data transfer. We measure the latency using heartbeat timestamps. For each node, an average latency cost is maintained over a small interval. Essentially, the idea is to divide the initial splits into smaller splits if the cost of data transfer is higher than cost of dividing the splits into virtual splits. **Scheduling when dependent DFS blocks are combined statically, as an application requirement:** This case is similar to the previous one, except that the virtual splits can not be created, and in some cases data transfer becomes unavoidable. We measure average cost for each data transfer using heartbeat timestamps. For example, if heartbeat returns in 2ms, 3ms and 2.6ms in an interval of 2sec, then the cost for that node is 2.53ms. The node with maximum number of blocks is chosen as the local node, and the node with minimal latency cost is chosen as the remote node for the transfer of remaining data blocks. The remaining blocks have to be transferred because the application requires these blocks to be processed together. **Scheduling when independent DFS blocks are combined dynamically to improve performance:** If the file splits are not specified by the application developer, then scheduler is responsible to make the splits and assign them to map tasks. In this case, we determine all the required file blocks per node and combine them into one split, although they are not logically contiguous. Hence, the size of splits and number of map tasks are determined dynamically based on the file size.

If there are x DFS blocks of the input file(s), r replicas and N nodes in the cluster, then the upper and lower bound on the number of blocks per map task B is give by $x/rN <= B < x/N$. Ideally, the number of blocks per map task per node should be kept closer to x/rN, so that there are no redundant data processing unless it is a speculative task. Also, the number of local I/O per map task per node should be close to x/rN, and number of remote I/O per map task should be 0 for the independent data blocks. For dependent data blocks that are combined statically, the lower bound on remote I/O can not be determined because it is application specific.

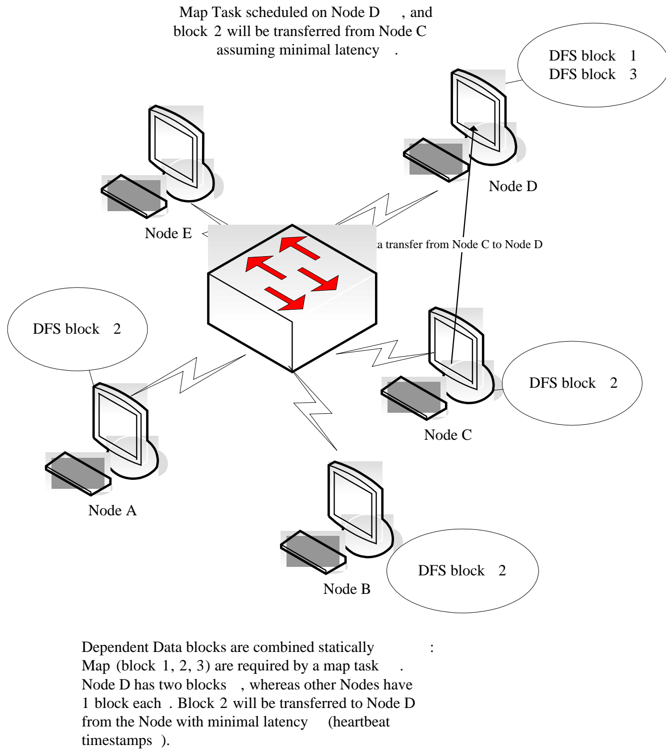

Figure 1. Scheduling when dependent DFS blocks are combined statically.

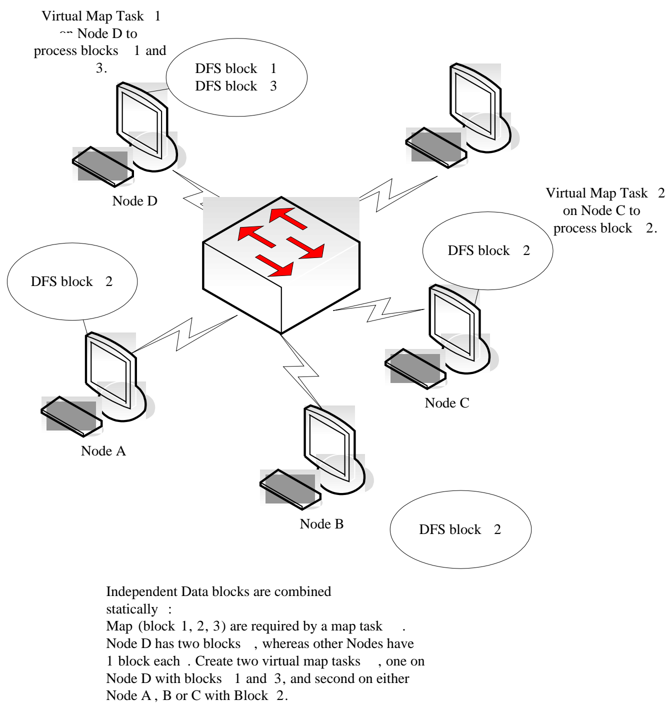

Figure 2. Scheduling when independent DFS blocks are combined statically.

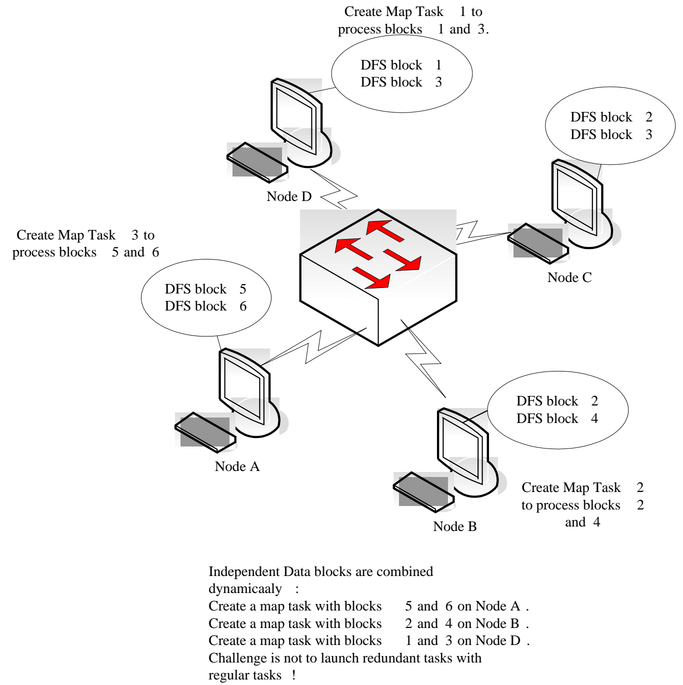

Figure 3. Scheduling when independent DFS blocks are combined dynamically.

References

[1] Hadoop. http://hadoop.apache.org/core/.

[2] Jeffrey Dean. Experiences with mapreduce, an abstraction for large-scale computation. In *PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, pages 1–1, New York, NY, USA, 2006. ACM.

[3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[4] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy H. Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In Richard Draves and Robbert van Renesse, editors, *OSDI*, pages 29–42. USENIX Association, 2008.