International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

A Novel Network-Levitated Merge Algorithm for Hadoop Acceleration

Rina Sao, Naveen K.

[1]SRM University, M.Tech Cloud Computing (IT), Tamilnadu, India

[2]SRM University, Information Technology, Tamilnadu, India

Abstract: ***Large companies like Facebook, Google, and Microsoft as well as a number of small and medium enterprises daily process massive amounts of data in batch jobs and in real time applications. This generates high network traffic, which is hard to support using traditional, oversubscribed, network infrastructures. To address this issue, several novel network topologies have been proposed, aiming at increasing the bandwidth available in enterprise clusters. Hadoop faces a number of issues to achieve the best performance from the underlying systems. These include a serialization barrier that delays the reduce phase, and the lack of portability to different interconnects. To keep up with the increasing volume of data sets, Hadoop also requires efficient I/O capability from the underlying computer systems to process and analyze data. We describe Hadoop-A, an acceleration framework that optimizes Hadoop with plug-in components for fast data movement. A novel network-levitated merge algorithm is introduced to merge data without repetition and disk access Our experimental results show that Hadoop-A significantly speeds up data movement in MapReduce and doubles the throughput of Hadoop.***

Keywords: Hadoop, MapReduce, Network-levitated merge, Hadoop acceleration, Cloud Computing

1. Introduction

MapReduce programs are being written for a wide variety of application domains including business data processing, text analysis, natural language processing, Web graph and social network analysis, and computational science. The MapReduce programming model mostly only requires programmers to describe the computation using two primitives inspired by functional programming languages, Map and Reduce. The map function usually independently processes a portion of the input data and emits multiple intermediate key/value pairs, while the reduce function groups all key/value pairs with the same key to a single output key/value pair. Additionally, users can provide an optional combine function that locally aggregates the intermediate key/value pairs to save networking bandwidth and reduce memory consumptions.

Many data-parallel applications could be easily implemented with MapReducemodel, such asWord Count, DistributedGrep, Inverted Index and Distributed Sort. A MapReduce program p expresses a computation over input data d through two functions: map(k1; v1) and reduce(k2; list(v2)). The map(k1; v1) function is invoked for every key-value pair hk1; v1i in the input data d to output zero or more key-value pairs of the form hk2; v2i. The reduce (k2; list(v2)) function is invoked for every unique key k2 and corresponding values list(v2) in the map output. reduce(k2; list(v2)) outputs zero or more key-value pairs of the form hk3; v3i. The keys k1, k2, and k3 as well as the values v1, v2, and v3 can be of different and arbitrary types.

Hadoop implements MapReduce framework with two categories of components: a JobTracker and many Task-Trackers. The JobTracker commands TaskTrackers (a.k.a. slaves) to process data in parallel through two main functions: map and reduce. In this process, the JobTracker is in charge of scheduling map tasks (MapTasks) and reduce tasks (ReduceTasks) to TaskTrackers. ReduceTask needs to fetch a part of the intermediate output from all finished MapTasks. Globally, this leads to the shuffling of intermediate data (in segments) from all MapTasks to all ReduceTasks. For many data-intensive MapReduce programs, data shuffling can lead to a significant number of disk operations, contending for the limited I/O bandwidth.

Hadoop MapReduce's has three data processing phases, i.e., shuffle, merge, and reduce.

More importantly, the current merge algorithm in Hadoop merges intermediate data segments from MapTasks when the number of available segments (including those that are already merged) goes over a threshold. These segments are spilled to local disk storage when their total size is bigger than the available memory.

This algorithm causes data segments to be merged repetitively and, therefore, multiple rounds of disk accesses of the same data. To address these critical issues for Hadoop MapReduce framework, we have designed Hadoop-A, a portable acceleration framework that can take advantage of plug-in components. Several enhancements are introduced: 1) a novel algorithm that enables ReduceTasks to perform data merging without repetitive merges and extra disk accesses; 2) a portable implementation of Hadoop-A that can support both TCP/ IP and remote direct memory access (RDMA). Since ReduceTasks are able to merge data by staying above local disks, we refer to this new algorithm as network-levitated merge (NLM). We have carried out an extensive set of experiments to evaluate the performance of Hadoop-A. Our evaluation demonstrates that the network-levitated merge algorithm is able to remove the effectively overlap data merge and reduce operations for Hadoop ReduceTasks. Overall, Hadoop-A is able to double the throughput of Hadoop data processing.

Reliable Storage: HDFS
Hadoop includes a fault-tolerant storage system called the Hadoop Distributed File System, or HDFS. HDFS is able to store huge amounts of information, scale up incrementally and survive the failure of significant parts of the storage infrastructure without losing data. Hadoop creates *clusters* of machines and coordinates work among them. Clusters can be built with inexpensive computers. If one fails, Hadoop continues to operate the cluster without losing data or interrupting work, by shifting work to the remaining machines in the cluster.

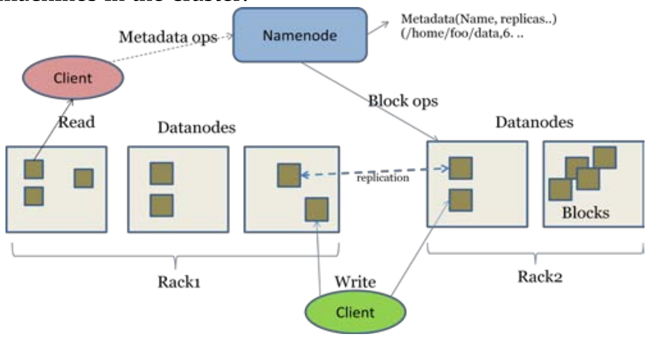

Figure 1: HDFS Architecture

The project includes these modules:

- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

Hadoop Acceleration: Hadoop Acceleration is an acceleration framework that optimized hadoop with plug-in component implemented in "JAVA" for fast data movement. Hadoop Acceleration doubles the data processing throughput of hadoop and reduces CPU utilization by more than 36%.

Network-Levitated Merge: RDMA (Remote Direct Memory Access) will be used for merging process in hadoop. It will not required disk access and it can directly access memory of other computer in the network, that's why it is called network –levitated merge. Since it is not access the hard disk it is only using RAM or Memory directly over network, So latency will be reduced.

2. Problem Statement

Our characterization and analysis reveal a number of issues, including
1) Repetitive merges and disk access, and
2) The lack of portability to different interconnects.

2.1 Repetitive Merges and Disk Access

- ReduceTasks merge data segments when the number of segments or their total size goes over a threshold.
- A newly merged segment has to be spilled to local disks due to memory pressure. However, the current merge algorithm in Hadoop often leads to repetitive merges, thus extra disk accesses.
- Figure shows a common sequence of merge operations in Hadoop. For the purpose of illustration, we hereby choose a very small threshold parameter io.sort.factor = 3. A ReduceTask fetches its data segments and arranges them in the order of their size. When the number of data segments reaches six, i.e. twice the threshold, the smallest three segments is merged, shown as Step 1 in Figure 2. Under memory pressure, this will incur disk access. The resulting segment is inserted back into the heap based on its relative size.

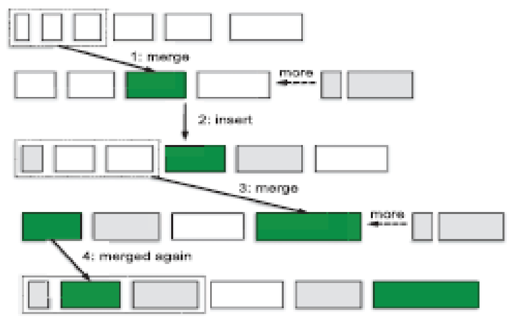

Figure 2: Repetitive Merging and Disk Access

- When more segments arrive, the threshold is reached again.
- It is then necessary to merge another set of segments, shown as Step 3. This again causes additional disk access, let alone the need to read segments back if they have been stored on local disks.
- As even more segments arrive, a previously merged segment will be grouped into another set and merged again, as shown in Step 4.
- Furthermore, any segment merged from a subset of segments eventually needs to be merged for final results. Altogether, this means repetitive merges and disk access, causing degraded performance for Hadoop. Therefore, an alternative merge algorithm is critical for Hadoop to mitigate the impact of repetitive merges and extra disk accesses.

2.2 The Lack of Network Portability

Besides the TCP/IP protocol, Hadoop does not support other transport protocols such as RDMA on InfiniBand and 10-Gigabit Ethernet (10GigE) that have matured in the high-performance computing (HPC) community.

Simply replacing the network hardware with the latest interconnect technologies such as InfiniBand and 10GigE and continuing to run Hadoop on TCP/IP will not enable Hadoop to leverage the strengths of RDMA.

It is worth noting that despite the high-price differential between RDMA-capable interconnects and traditional commodity Gigabit Ethernets, such price differences have shrunk significantly over the past few years. Many popular commodity interconnects, such as 10GigE, are becoming RDMA-capable as well. Thus, the lack of portability on multiple interconnects will prevent Hadoop from keeping up with the advances of other computer technologies, particularly when more powerful processors, storage, and interconnect devices are deployed to various computing and data centers.

3. Existing System

- Jobtracker commonds multiple tasktracker in mapreduce framework.
- Shuffling of data from maptask to all reducetasks.
- Data shuffling lead to significant no. of disk operations.
- The overall processing is performed using shuffle, merge and reduce phases in hadoop.
- MapReduce: Simplified Data Processing on Large Clusters.
- A MapReduce Framework on Graphics Processors.

4. Proposed Solution

4.1 Network-Levitated Merge:

- In Hadoop repetitive merges because of limited memory compared to the size of data. At the time of merging pull the entire data, and store locally in memory or on disk. This incurs many memory loads/stores and/or disk I/O operations.
- Design an algorithm that can merge all data partitions exactly once and, at the same time, stay levitated above local disks.

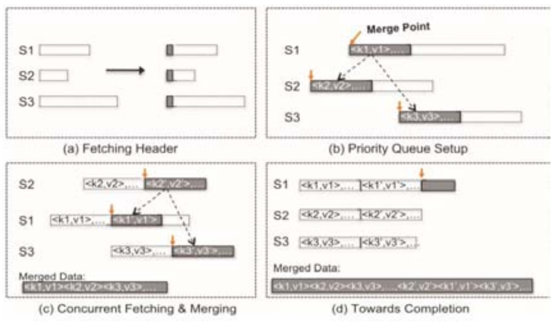
S1
S2
S3
(a) Fetching Header
Merge Point
<k1,v1>,...
<k2,v2>,...
<k3,v3>,...
(b) Priority Queue Setup
<k2,v2>,... <k2',v2'>,...
<k1,v1>,... <k1',v1'>
<k3,v3>,... <k3',v3'>,...
Merged Data:
<k1,v1><k2,v2><k3,v3>,...
(c) Concurrent Fetching & Merging
<k1,v1>,... <k1',v1'>,...
<k2,v2>,... <k2',v2'>,...
<k3,v3>,... <k3',v3'>,...
Merged Data:
<k1,v1><k2,v2><k3,v3>,... <k2',v2'><k1',v1'><k3',v3'>,...
(d) Towards Completion

Figure 3: Network Levitated Merge Algorithm

- There are three remote segments S1, S2, and S3 are to be fetched and merged.
- Instead of fetching them to local disks, this new algorithm only fetches a small header from each segment.
- Each header is especially constructed to contain partition length, offset, and the first pair of <key,val>. These <key,val> pairs are sufficient to construct a priority queue (PQ) to organize these segments.
- More records after the first <key,val> pair can be fetched as allowed by the available memory. Because it fetches only a small amount of data per segment, this algorithm does not have to store or merge segments onto local disks.
- The leading <key,val> pair will be the beginning point of merge operations for individual segments, i.e., the merge point.
- This algorithm merges the available <key,val> pairs in the same way as is done in Hadoop. When the PQ is completely established, the root of the PQ is the first <key,val> pair among all segments.
- Update the order of PQ based on the first <key,val> pairs of all segments. The next root will be the first <key,val> among all remaining segments.
- The headers of all three segments are safely merged; more data records are fetched, and the merge points are relocated accordingly.
- Figure 3 shows a possible state of the three segments when their merge completes.

4.2 RDMA based MapReduce Design

We introduce the major components in our RDMA based MapReduce design and then explain our design details. We first discuss our updated Shuffle design followed by the Merge design.

1) RDMA based Shuffle: In the shuffle stage, both TaskTracker and ReduceTask are modified to achieve the benefits of RDMA. We have added the following new components in the TaskTracker side:

RDMA Listener: Each TaskTracker initiates an RDMAListener during its startup. RDMAListener in TaskTracker waits for incoming connection requests from the ReduceTask side, adds the connection to a pre-established queue, and starts an RDMAReceiver if necessary.

RDMA Receiver: Each RDMAReceiver is responsible for receiving requests from ReduceTasks. RDMAReceiver gets the end-point list that is currently being used and receives request from those end-points. After receiving the request, it places the request in DataRequestQueue.

DataRequestQueue: DataRequestQueue is used to hold all the requests from ReduceTasks.

RDMA Responder: RDMAResponder belongs to a pool of threads that wait on DataRequestQueue for incoming requests. Whenever a new request gets inserted in Data Request Queue, one of the RDMAResponders responds to that request.

RDMA Copier: In the default design of MapReduce, the copier threads are responsible for requesting data to Task-Tracker and storing data for Merge.

2) Faster Merge: In the default design of MapReduce, each HTTP response consists of the entire map output file, dividing it into packets of the default packet size, 64 KB. Which in turn creates the opportunity to transfer one map output file in multiple communication steps instead of one. By doing this, we can start the merge process as soon as some key-value pairs from all map output files reach at the reducer side.

3) Overlap of Shuffle, Merge and Reduce: In our design, we start reduce operation as soon as the first merge completes. In this way, we can achieve maximum benefit by introducing pipelining between merge and reduce stages.

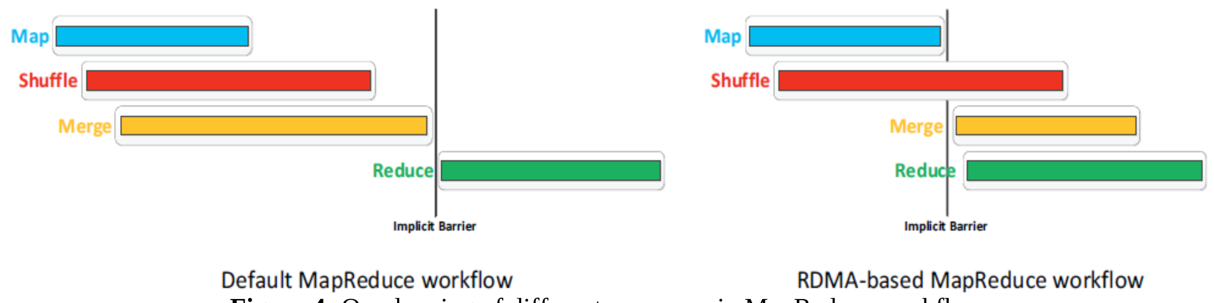

Figure 4: Overlapping of different processes in MapReduce workflow

5. System Architecture

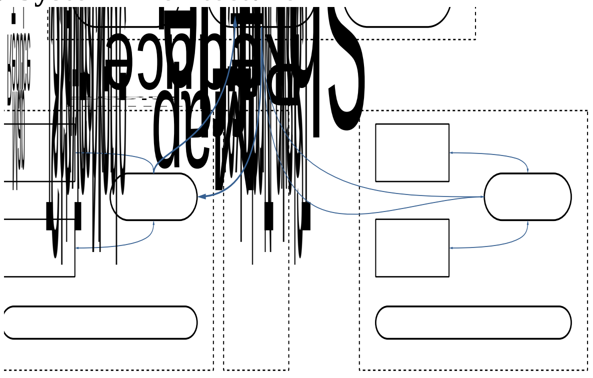

Figure 5: System architecture of hadoop MapReduce

6. Conclusions

- Examined the design and architecture of Hadoop MapReduce framework and reveal critical issues faced by the existing implementation.
- Designed and implemented Hadoop-A as an extensible acceleration framework which addresses all these issues.
- We reveal that there are several critical issues faced by the existing Hadoop implementation, including its merge algorithm, as well as its lack of portability for multiple interconnects. We have designed and implemented Hadoop-A as an extensible acceleration framework that can allow plug-in components to address all these issues. By introducing a new network levitated algorithm that merges data without touching disks. we have successfully accomplished an accelerated Hadoop framework, Hadoop-A. In addition, Hadoop-A has been designed as a portable framework that can run on both high-performance RDMA protocol and ubiquitous TCP/IP protocol.Hadoop-A can significantly reduce disk accesses during Hadoop's shuffling and merging phases, thereby speedingup data movement. Furthermore, we have quantified the performance benefits of network-levitated merge and the RDMA protocol, respectively, on the Hadoop MapReduce.

References

[1] Weikuan Yu, Member, IEEE, Yandong Wang, and Xinyu Que "Design and Evaluation of Network-Levitated Merge for Hadoop Acceleration" IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 25, NO. 3, MARCH 2014 page- 602-611.

[2] R. Recio, P. Culley, D. Garcia, and J. Hilland, "An RDMA Protocol Specification (Version 1.0)," Oct. 2002.

[3] http://en.wikipedia.org/wiki/MapReduce.

[4] web.cs.wpi.edu/~cs4513/d08/OtherStuff/MapReduce-TeamC.ppt.

[5] Infiniband Trade Association, http://www.infinibandta.org 2013.

[6] T. Condie, N. Conway, P. Alvaro, J.M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce Online," Proc. Seventh USENIX Symp. Networked Systems Design and Implementation (NSDI), pp. 312-328, Apr. 2010.

[7] M. Zaharia, A. Konwinski, A.D. Joseph, R.H. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," Proc. Eighth USENIX Symp. Operating Systems Design and Implementation (OSDI '08), Dec. 2008