International Journal of Engineering & Technology, 7 (2.27) (2018) 109-112

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

An improved hadoop load rebalancer

Geetha J [1] *, Uday Bhaskar N [2], Chenna Reddy P [3]

[1] Department of Computer Science and Engineering, M S Ramaiah Institute of Technology, Bangalore, 560054, India
[2] Department of Computer Science, Government College (UG & PG) Anantapur, 515001, India
[3] Department of Computer Science and Engineering, JNTU College of Engineering, Anantapur, 515001, India
**Corresponding author E-mail: geetha.yj@gmail.com*

Abstract

Hadoop has taken an important space in the market as a result of quick growth of data. Load rebalancing in Hadoop is an area of major concern due to the unpredictable nature of tasks, new nodes added to cluster and node computing capacities. A load rebalancer that is efficient can help to improve the performance and reduce computation time. Load rebalancer and schedulers are used interchangeably in many cases. The main idea of this paper is to explore how load balancers / schedulers work in case of native Hadoop also included insights from some of the works, which identify and addresses the problems around schedulers and rebalancers. In this paper, an Improved Hadoop Load Re-balancer adopts a strategy to move the task to the node which has replica, which is faster and is topologically closer, which reduces the network congestion and execution time of Hadoop.

Keywords: *HDFS; Load Balancer; Rebalance; Scheduler; Spark; Yarn*

1. Introduction

If Forbes is to be believed, for each person living, 1.7 megabytes of new information will be generated per second. This fact only reiterates the immediate need to direct our attention towards managing the humungous amount of data. Hadoop MapReduce is one important answer that comes to our rescue. Quoting another statistics presented by the Forbes, Hadoop market is predicted to grow at a compounded annual rate of 58%. So, optimizing the Hadoop framework is of utmost importance .

The Hadoop uses MapReduce paradigm for parallel processing. The two important phases in MapReduce are - the Map phase and the Reduce phase. Sometimes it is also considered to have another intermediate phase called the Spilling or Shuffling phase. The input to the Map phase is in the form of key-value pairs and the output of the Mapper is partitioned and fed to the reducer.

Hadoop also has different job schedulers like Fair scheduler, FIFO scheduler and capacity scheduler. The MapReduce framework or the scheduling algorithm of the Hadoop architecture as a whole can be considered as subjects to be optimized. Researchers from different parts of the world have attempted to optimize Hadoop framework by different approaches. In Hadoop, the workload is distributed amongst a cluster of nodes [1]. Tasks are distributed by the scheduler in a balanced manner. Due to heterogeneous nature of clusters a state of load imbalance is evident. Traditional load rebalancers try to shuffle tasks around the nodes in the cluster to maintain optimal throughput.

The HDFS architecture is capable of rebalance data by means of its data rebalancing schemes. The rebalancing scheme can spontaneously move data from one DataNode to the other if the free space on a DataNode falls below a definite threshold. If there is a high demand for a particular file the scheme can dynamically add replicas to rebalance data over the cluster. Schemes like these are not currently present in Hadoop. [3] In this paper, we intend to provide an insight into a few approaches taken by researchers in the load rebalance and scheduling and proposed an improved Hadoop rebalancer, which reduce the execution time [13].

2. Literature review

When MapReduce clusters get popular, their scheduling becomes ever more important. Data are distributed to individual nodes and stored in their disks in a MapReduce cluster. We need to first have its input data available on that node, to execute a map task on a node. While transferring data from one node to another takes time and delays task implementation, we must avoid unnecessary data transmission by designing an efficient MapReduce scheduler. It is efficient to move data processing operations to nodes where application data are located in a cluster of Hadoop nodes, where each node has a local disk. If data is not locally available in a computing node, data have to be migrate via network interconnect to the node that perform the data processing operations. Migrating vast amount of data lead to extreme network congestion, which in turn can reduce the system performance [4].

2.1. Scheduling and load balancing options

2.1.1. First in first out scheduler

This is the least complex one among all other Hadoop schedulers. It schedules the jobs in First come first serve basis. Architecture design is not required like other ones. FIFO scheduler is not suited for jobs, which require more resources. In case of Clustering it's good if we use capacity and fair schedulers.

2.1.2. Fair scheduler

The Fair Scheduler [5] aim to give each user a fair share of the cluster capacity over time. Users may allocate jobs to pools, with each pool allocated a definite minimum number of Map and Re-

duce slots. In idle pools the free slots may be allocated to other pools, as excess capacity within a pool is shared among jobs. The preemption is supported in Fair Scheduler, so if a pool has not received its fair share for a definite period of time, then the scheduler will destroy tasks in pools running over capacity in order to give the slots to the pool running below capacity. Administrators may enforce priority settings on certain pools. Therefore tasks are scheduled in an interleaved manner, based on their priority within the pool, and the cluster capacity and usage of the pool. As slots become vacant for scheduling, highest time deficit is assigned to the next free slot, this has the effect of ensuring that jobs receive roughly equal amounts of resources. Sufficient resources are allocated to shorter jobs to finish quickly and longer jobs are guaranteed not to be starved of resources.

2.1.3. Capacity scheduler

The Capacity Scheduler [6,7]is intended to allow sharing a large cluster while giving each user a minimum capacity guarantee .All the Queues in cluster ,that contain jobs are given their minimum capacity, while the unused excess capacity in a queue is shared among other queues. Capacity scheduler operates on a modified priority queue basis with definite user limits, with priorities adjusted based on the time a job was submitted. The queue with the lowest load is chosen, from which the oldest remaining job is chosen, When a Task Tracker slot becomes free. In general, this has the effect of enforcing cluster capacity sharing among users, rather than among jobs [14].

2.1.4. Late scheduler

This Scheduler is a substantial enhancement of the speculative execution. The LATE speculative scheduling implementation totally relies on some assumption: a) homogeneous progress of tasks on the node b) homogeneous computation at all the nodes. In heterogeneous clusters these assumptions easily splits down. When modified version of speculative execution is considered, the computation of estimated remaining time is done which gives a further clear estimation of straggling tasks impact on the overall job response time [8].

2.1.5. Delay scheduler

The Fair Scheduler aim is to assign fair share of capacity among all the users. The locality problem arises due to strict queuing policy in fair scheduler. First if the head of the line job cannot begin a local task, then the task is skipped and the locality problem is sticky slot problem [9]. Head of line problem is resolved by a scheduling task from a job on a node without having local data to maintain fairness. If it is not possible to run a task on a node that do not contain local data, then it is better to run task on same rack on some other node. In delay scheduling, once a node requests a task, if the head-of-line job cannot start on a local task, it is skipped and looks at succeeding jobs. Conversely, if a job has been skipped for ample times, then non-local tasks are to be scheduled to keep away from starvation.

2.1.6. Deadline constraint scheduler

Deadline Constraint Scheduler is designed to deal with the issue of deadlines. The schedule ability test is done, When a job is submitted to scheduler that determines whether the job can be finished within the particular deadline or not. The availability of free slots is computed at the given time. If the available slots are sufficient to complete the job within the given deadline than job is submitted for scheduling [10].

2.1.7. Scheduler with greedy strategy

The remote high performance storage devices are prioritized over local low performance storage devices. This kind of scheduler is suitable on the Hadoop cluster for which the local storage devices are mostly hard disks. However a few nodes in the cluster are set with huge memory or SSD and replica of the data blocks are also stored in SSD or in memory storage. We read data from the remote SSD or in-memory storage in a greedy manner, in such cases. This method performs better for large data sizes, as the performance gaps between the storage devices are huge and the underlying interconnects are fast. [11], [15]

3. Design of improved load rebalance

As mentioned earlier, HDFS already has a load rebalancing scheme, but it can be made more efficient. Currently rebalancing is done on demand. The administrator has to issue a command to ensure the load is balanced across the nodes [12-16]. We propose a method to make use of the replicated data to maintain the load balance with minimal cost. Our Load balancer doesn't require any manual intervention but it will be a self triggered process subject to any node being overloaded.

In case there is no node failure then replicated data blocks is redundant. Our solution makes use of these replicated data and achieves efficient rebalancing. Every task is tied with a data block. Traditional approach of rebalancer is to move the task and also the data to the node which is relatively free from a busy node. Our solution proposes that a task must be moved to the queue of a node with free slot on a pre condition - if the node's HDFS already has the task related data block in it, then move the task to this particular node instead. This invalidates the need to transfer the data block hence reducing the congestion and execution time. For the simplicity of explanation, we have assumed that data-block-id is the same as the task-id. The trigger for our rebalancer to kick in is when any node queue is experiencing heavy load, i.e if the number of tasks is above the threshold. We maintain a dynamic data structure - idle_queue_list. A node is added into this list as soon as it is idle (no tasks to process). Since nodes in a cluster execute tasks at different speeds, we also maintain another data structure - node_avg_execution_time_list. This example demonstrates the need to keep one. Let us assume node 'A' and node 'B' have 3 and 6 tasks lined up in their queue and have execution speeds of 1 task/second and 3tasks/second respectively. If we go by queue length, node 'B' appears busier than node 'A' where as the converse is true.

If there is a need for rebalancing, we search the idle queue for a node which has a replica of the data block necessary for the execution of the task that is considered for migration. If a suitable node is present, then the task is moved. Figure 1 and figure 2 we have considered replication factor to be 3, it further explains the idea.

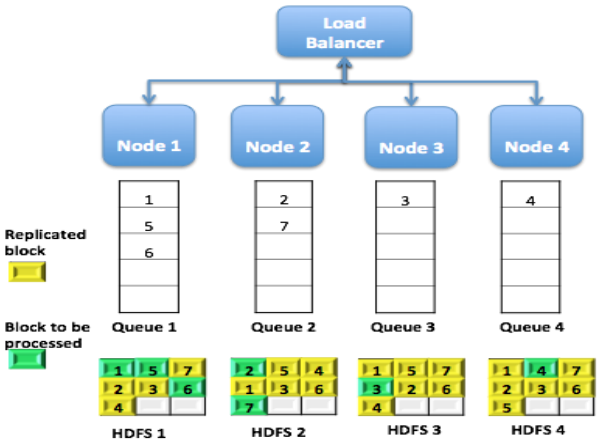

Fig. 1: Allocation of Tasks in the Queue.

If the idle queue is empty, we find the nodes whose utilization is below the threshold. On considering execution time as a factor, node 'B' will be the first one to be idle instead of node 'A'. Hence, if no node is in idle state, the next priority to assign the task is to the node, which has the least value for

node_avg_execution_time_list with task data present as replica in its HDFS.

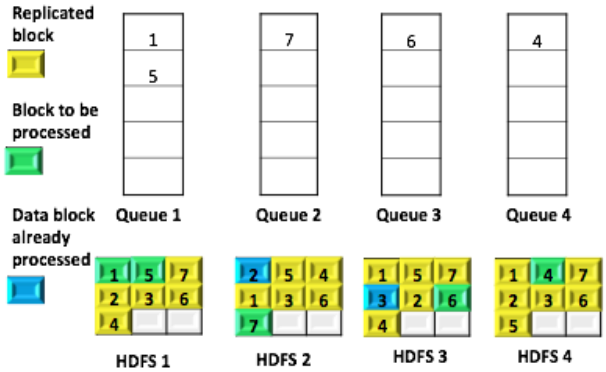

Fig. 2: Allocation of Tasks in the Queue after Load Rebalance.

In the worst-case scenario, i.e if data block is not found then our solution gives way to the existing implementation.

```
OptimizedLoadBalancer()
{
  /*Trigger for our rebalancing algorithm*/
  If(Any node's queue is above the threshold){
    last_task_data = id of the data block for the last task in the node queue;
    If(Any node entry is present in idle_queue_list){
      for(All the nodes in the idle_queue_list){
        if(replica(last_task_data in the node's HDFS)){
            last_task = remove(last_task_data from it's original node queue);
            insert(last_task in the node's queue for processing);
            break;
        }
      }
    }
      Else If(No node in idle queue has data replica or idle queue is empty){
        for(All the node's queue who are below the threshold){
          fast_node = min(node_avg_execution_time_list);
          if(replica of last_task_data found in the fast_node's HDFS){
              last_task = remove(last_task_data from it's original node queue);
              insert(last_task in the node's queue for processing);
              break;
          }
        }
      }
    }
}
```

Fig. 3: Algorithm for Load Rebalance.

4. Experimental results

The performance of the improved Hadoop rebalancer and the Native load rebalancer of Hadoop is compared using CloudSim simulator .An improved Hadoop rebalancer shows performance improvement compared with the default Scheduler by considering data locality of replica into account. We simulate a MapReduce job with two hundred machines which are organized in ten racks.To simulate MapReduce job we need to configure some of the characteristics of the CloudSim such as data center, network and scalability to show the improvements achieved by our proposed algorithm. A research-oriented simulator TeachCloud is an extension of CloudSim, used for the development and validation in cloud computing.

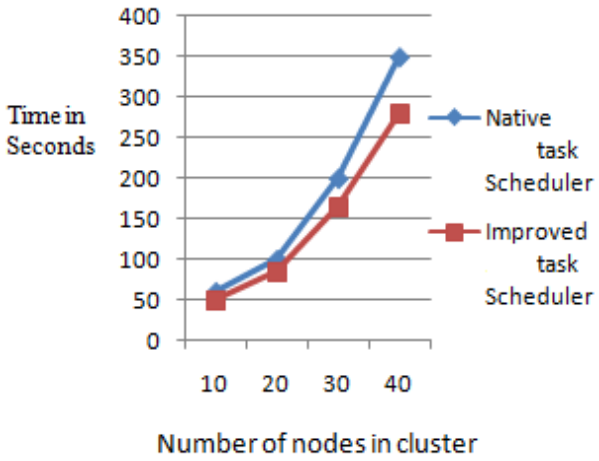

Fig. 4: Allocation of Tasks in the Queue.

The parameters which are used in our simulation are:
No_ File: number of files that contain data for each tasks and this will be extend among other nodes.
No_ Job: The number of jobs that are submitted by from different users.
No_Task: The number of tasks that required to be executed. The number of tasks are distributed randomly among the jobs.
No_nodes: is the number of nodes that need to be considered over which the data is split over. Huge chunks of data may be spread over different nodes at different location.
In our simulation, we need to show the performance of the improved hadoop rebalancer over the Native task Scheduler. The simulation results showed the improvement of our improved load rebalancer algorithm. Simulation is conducted to compare among them in terms of two factors which are the simulation time factor and the network traffic. Scheduling Time is the time required for scheduling of map tasks on the nodes that depends on the file size or data size that needs to be transferred. By Load rebalancing algorithm it schedules the map tasks on to the nodes where replica for that task is present with minimum load which reduce network congestion and execution time.

5. Conclusion

Out of all the schedulers we explored, Data Locality driven scheduler comes closest to our approach. Their main ideas is, for a task, when data isn't found locally then read the data from a node which has replica, in case it has multiple copies read from the node which has high performance and is topologically closer. The main difference between our Approach and the latter one is that instead of reading the data from a remote node, we adopt to a strategy to move the task to the node which has replica, which is faster and is topologically closer, which reduces the network congestion and execution time This involves a zero cost. Since replicated data blocks are central to our idea, our future plan is to look into the existing replication schemes. Replicas can be placed more efficiently, giving precedence to node's speed.

References

[1] Apache. Welcome to ApacheTM HadoopR. 10. July 2013
[2] Kwon, Y., et al. "A study of skew in mapreduce applications." Open Cirrus Summit (2011).
[3] HDFS Architecture guide - https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
[4] Tyagi, Sajal, and Shipra Saraswat. "Different Scheduling Options in YARN." In Microelectronics and Telecommunication Engineering (ICMETE), 2016 International Conference on, pp. 190-196. IEEE, 2016.
[5] Hadoop's Fair Scheduler. - https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.

[6] Chauhan, Jagmohan, Dwight Makaroff, and Winfried Grassmann. "The impact of capacity scheduler configuration settings on mapreduce jobs." In Cloud and Green Computing (CGC), 2012 Second International Conference on, pp. 667-674. IEEE, 2012.
[7] Hadoop's Capacity Scheduler.- http://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html.
[8] Kulkarni, Amogh Pramod, and Mahesh Khandewal. "Survey on Hadoop and Introduction to YARN." International Journal of Emerging Technology and Advanced Engineering 4, no. 5 (2014): 82-87.
[9] Yoo, Dongjin, and Kwang Mong Sim. "A comparative review of job scheduling for MapReduce." In Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on, pp. 353-358. IEEE, 2011.
[10] Rao, B. Thirumala, and L. S. S. Reddy. "Survey on improved scheduling in Hadoop MapReduce in cloud environments."arXiv preprint arXiv:1207.0780 (2012).
[11] Islam, Nusrat Sharmin, Md Wasi-ur-Rahman, Xiaoyi Lu, and Dhabaleswar K. DK Panda. "Efficient data access strategies for Hadoop and spark on HPC cluster with heterogeneous storage." In Big Data (Big Data), 2016 IEEE International Conference on, pp. 223-232. IEEE, 2016.
[12] Hadoop Load Rebalancer is on demand - https://issues.apache.org/jira/browse/HADOOP-1652.
[13] Thirumala Rao, B., Susmitha, M., Swathi, T., & Akhil, G. (2018). "Implementation of Hybrid Scheduler in Hadoop", International Journal of Engineering & Technology, 7(2.7), 868-871.
[14] S. Kalyan Chakravarthy, N., Sudhakar, N., & Srinivasa Reddy, E. (2018). "Implementation of cost effective hierarchical Hadoop cluster–a case study for education", International Journal of Engineering & Technology.
[15] Sujatha, J., & Meena, K. (2018). "A vibrant data placement approach for map reduce in diverse environments", International Journal of Engineering & Technology, *7*(2.4), 20-22.
[16] Nagalakshmi, M., Surya Prabha, I., & Anil, K. (2017). "Bigdata implementation of apriori algorithm for handling voluminous datasets". International Journal of Engineering & Technology, *7*(1.5), 217-220.