Enhanced Task Scheduling Scheme for Hadoop Map Reduce Systems

1ARULMURUGAN R, 2ANGURAJ S, 3M.SRINIVASAN, 4G.SINGARAVEL
1M.Tech (IT), K.S.R College of Engineering, Tiruchengode, Namakkal, Tamilnadu.
2AP, Dept. of IT, K.S.R College of Engineering, Tiruchengode, Namakkal, Tamilnadu.
3Vice President - Business Analytics & Big Data Research at Ethna Attributes Soft Technologies.
4HOD, Dept. of IT, K.S.R College of Engineering, Tiruchengode, Namakkal, Tamilnadu.
1kavarul@gmail.com, 2s_anguraj@yahoo.co.in, 3srinivasan.m@eattributes.com, 4singaravelg@gmail.com

ABSTRACT

As we are literally drawn in the era of big data the hadoop and its ecosystem has become the integral part of the big data analytics. However maps reduce which take care of the processing part. Hadoop requires the some efficient performance up gradations in order to cope up with the data taken under consideration of processing. In this paper we describe three up gradation which will fine tune the map reduce. The initial step is to replace the HFS to DFS their by improving the job performance. Secondly a prefetching mechanism is proposed which reduces the execution time. Finally a service component with eminent I/O feature is implemented to rebuild the shuffle task as a service. Thus by implementing the modification mentioned above we obtained a map reduce framework that is enhanced to work to work in a hadoop environment.

Keywords: Hadoop, Map Reduce, Scheduler, Shuffle, Prefetching.

I. INTRODUCTION

Map Reduce has been most preferred and suitable for distributed and parallel computations. Even Since it was introduced by Google[1].as mentioned earlier hadoop has been widely used in industry and also scientist as base of their research. However hadoop can be improved in many aspects such as data management, scheduling [2], execution time etc.

This paper focuses on improving the Map Reduce by employing an enhanced scheduler called as opportunistic fair scheduler which guarantees that a fair share of resources is allocated to all the job there by maintaining an opportunity environment. Then the prefetching mechanism ensures that data movement after each task does not affect the execution time of the job. The last change is done the in the shuffle part of the map reduce.

During the shuffle phase the map task output are shuffled and sorted and then provided as input to the reduce task. Usually the shuffle task are involved in series of I/O operations which amputees the performance of Map Reduce model. This problem is dealt by providing shuffle as service rather than shuffle as task which means reduce the I/O operations.

II. HADOOP:

The hadoop runtime consist of two types of processes called ‘Job Tracker’ and ‘Task Tracker’. The job tracker partitions the input data into splits using a splitting method defined by the programmer, populates a local task-queue based on the number of splits and distributes work to the Task Trackers that in turn process the splits. If a Task Trackers becomes idle, the Job Trackers picks a new task from its queue for that Task Tracker to execute. Thus, the granularity of the splits has considerable influence on the balancing capability of the scheduler. Another consideration is the location of the data blocks: among all the tasks of the selected job to be executed, it priorities those tasks with data local to the Task Tracker. Notice that this data locality consideration never affects the decision about the scheduled job.

Each Task Tracker controls the execution of tasks on a node. It receives a split descriptor from the Job Tracker, and spawns a new worker process which runs a so-called map task to process the split received from the Job Tracker. The Task Tracker also runs the so-called reduce task as soon as they can be initiated. Notice here that a map task will eventually result in the execution of a map () function, and that a reduce task similarly results in the execution of reduce () function. The programmer can also decide how
many simultaneous map() and reduce() functions can be run concurrently on a node.

Communication between Task Tracker and Job Tracker is a Heartbeat protocol. Hadoop Distributed File System includes one Name Node and multiple Data Nodes; Name Node keeps the directory tree of all files in HDFS, and tracks where across the cluster the file data is kept. Name Node does not store the data of files, while Data Nodes store the data. Developers usually centralize nodes to process the Job Tracker and Name Node procedure.

III. MAPREDUCE:

Map Reduce has been used at Google [3] for processing large datasets in distributed computing environment. The map and reduce are two processes in the Map Reduce. The map process is a function that processes a set of input<key, value> pairs that is a portion of a large input dataset to generate a set of intermediate <key, value> pairs. Following the map process is the reduce process in which a reduce function merges all of intermediate values generated by the map process associated with the same intermediate key to form a possibly smaller set of <key, value> pairs which is the output of the Map Reduce. Figure 1 illustrates workflow in Map reduce.

In real applications, copying and sorting may consume considerable amounts of time, especially in the copy phase. However, in the Hadoop implementation, all copy actions of reduce tasks will start when the first map action is finished [5].

IV. SCHEDULING:

The scheduling in Map Reduce is a key area of research since it forms the core of the Hadoop System.

Several scheduling algorithms are proposed in order to intensify the scheduling performance. Some of the related works in the scheduling are:

1) First Input First Output “FIFO”: this scheduler usually processes the first job submitted then the next respectively, and so on. In this criteria, the jobs are processed by the order of jobs as first comes first served.

2) Matchmaking scheduling: It gives every slave node a chance to process local tasks. The main idea in this algorithm is to allow nodes to process local task from another job in the queue. Hence, the scheduler does not consider the job order to be important. To explain, when a scheduler does not find local task in first job, the scheduler keeps searching in the next job to find local map task for the node. If the node does not find the more local tasks, at this heartbeat, no any non-local task will be assigned which gives more fair. In the second cycle, if a node still does not find more local task to free slots and to avoid starvation, the scheduler will assign only one nonlocal map task to the node that has free slots in every heartbeat interval.

3) Capacity scheduler: It makes partitions for the resources and divides them into multi pools with dedicated job queue for every pool. The authors of this scheduler observed that the order of executing job has an impact on all execution times for all jobs. So they try to concentrate more on ordering jobs to achieve best system performance. Balanced pool uses Johnson’s algorithm that has optimal scheduling for two stages problem such as Map and Reduce Scheduling.

4) Dynamic priority scheduler: It uses parallel scheduling. That is, it uses one of the previous methods with a priority condition. This condition differs from algorithm to another, but most of priority algorithm use job deadline, pricing, or other thresholds.
5) **COSHH**: It consists of two main processes where each is triggered by receiving one of these messages. Upon receiving a new job, the scheduler performs the queuing process to store the incoming job in an appropriate queue. Upon receiving a heartbeat message, the scheduler triggers the routing process to assign a job to current free resource.

The high level architecture of COSHH consists of four components: The Hadoop system, the task scheduling, the queuing process, and the routing process. The task scheduling process estimates the execution time of an incoming job on all resources. These estimates are passed to the queuing process to choose an appropriate queue for the incoming job. The routing process selects a job for the available free resource, and sends it to the task scheduling process. Using the selected job’s characteristics, the task scheduling process assigns tasks of the selected job to available slots of the free resource.

6) **Fair Scheduler**: It gives every job a fair share of the cluster over a respected time slot. This time slot is predefined in order to prevent any greedy jobs from resources reserving.

Every scheduler from the above scheduler has disadvantages. For example, in FIFO scheduler, small jobs have a problem in waiting large job processing.

- FIFO also does no respect data locality for jobs that are needed in Map-reduce scheduling Framework.
- Capacity scheduler does not respect data locality.
- Dynamic priority scheduler interest to achieve special goals. So, this scheduler does not respect data locality too.
- COSHH scheduler fails in an opportunistic environment since the classification and optimization consumes more time as it needs to perform routing, task scheduling and queuing process in the stipulated time.
- Fair share scheduler demands more time for scheduling in context switch between jobs.

Many other schedulers exist in the literature. Most of those schedulers suffer from the same problems that the above mentioned schedulers suffer from.

V. PROPOSED SYSTEM:

A. **Scheduling**:

As mentioned earlier the proposed system alters the performance of the Map Reduce in three phases. The initial phase is the scheduling. Since all the schedulers discussed in the earlier section carries several disadvantage we adapt a fair scheduling algorithm that does not amputated in an opportunistic environment. Figure 2 shows the data flow in our proposed system.

![Data Flow in the system](image)

**Figure 2.** Data Flow in the system

Fairness has been a key objective of Map Reduce scheduling in a multi-user environment, and Hadoop Fair Scheduler (HFS) has been one of the mostly adopted scheduler. The current HFS is designed for dedicated homogeneous resources and may perform poorly in an opportunistic environment, where nodes typically exhibit different front end loads, even if the hardware capabilities are homogeneous.

To address the issues mentioned above, we propose an extension to HFS, called opportunistic Fair Scheduler (OFS), to improve fairness in Map Reduce scheduling in an opportunistic environment. We introduce two notions, Fractional task slot and availability correlation.

- Fractional task slot: To identify the fractional task slot across the various nodes the following notions are introduced.
- Availability state: The availability state in an binary value either deception ’1’ for the availability (or) ’0’for non availability of the node executing the Map Reduce task.
- Availability rate: Due to the dynamic nature of the availability state there are several ways adopted to assign weights to
the tasks slots such as slowdown factor, the long term availability rate or the factions of the active execution time during most recent task. Here the availability rate is calculated between the average availability period until time by the sum of the average period and average unavailable period until time ends.

- Fractional share: Instead of balancing the number of task slots occupied by each jobs, OFS tries to balance the total weight per job computed based on all its running tasks as below, where each task slot is weighted by the availability rate of its hosting node. This total weight, referred to as the fractional share of the job, replaces running task slots as the new measure of resource in our scheduler design.

More over the availability correlation is implementation since a job may have intermediate data at multiple nodes. Thus the correlation between the host of the reduce slot and host of each piece of intermediate data as the measure of Map Reduce correlation if the selected job receives the reduce slot.

B. Prefetching:

Upon the arrival of a request from the Job Tracker, the predictive scheduler triggers the prefetching module that forces preload worker threads to start loading data to main memory. The following three issues must be addressed in the prefetching module.

- Prefetching at a right time: It controls how early to trigger prefetching actions. Before a block finishes, the subsequent block will be loaded into the main memory of the node. The prediction module assists the prefetching module to estimate the execution time of processing each block in a node. It is noted that the block processing time of an application on different nodes may vary in a heterogeneous cluster. The estimates are calculated by statistically measuring the processing times of blocks on all nodes in a cluster. This statistic measuring can be performed offline.

- Prefetching the appropriate data: The prefetching module must determine blocks to be prefetched. Initially, the predictive scheduler assigns who tasks to each task tracker in a node. When the prefetching module is triggered, it proactively contacts the Job Tracker to seek required information regarding data to be processed by subsequent tasks.

- Prefetching methodology: when the prefetching action is triggered, the prefetching module automatically fetches data form disks. Due to the large block size in HDFS, we intend not to make our prefetching module very aggressive. Thus, there is only one block being perfected at a time.

The most important part of the prefetching work is to synchronize two resources in the Map Reduce system: The computing task and the data block. The scheduler in the Map reduce always collects all the running task information and constructs a running Task List. It separately catches the different types of tasks in a map task list and a reduce task list. The job tracker can manage the current task according to these lists. The prefetching manager in the master node constructs a list known as the data list, a collection of all the data block location information. The Worker thread in each running node can finish the data loading job all by itself before the task is received.

Arrays of prefetching techniques have been proposed to improve the performance of main memory in computing systems. Cache prefetching techniques used to improve effectiveness of cache-memory systems have been widely explored for a variety of hardware and software platforms. Increasing prefetching approaches to boosting I/O performance in computer systems.

C. Enhancing the shuffle:

The shuffling is enhanced by employing three basic steps.

1. Separating the shuffle from the reduce task: Firstly extract the shuffle related operations such as copy, merge from reduce task as an individual task similar to map task. In this way, each old reduce task will be divided into a reduce task and a corresponding shuffle task which run on the same node. Shuffle task is responsible for preparing the input data for its matching reduce task as usual.

   Number of request from the reduce tasks is the same as usual even though the network and memory utilization may be improved after the splitting operation.

2. Establishing Shuffle as a service:
Usually there is more than one slot in each Task Tracker, resulting in more than one reduce task running on the same node. Therefore, it needs more than one Shuffle task on one Task Tracker. Based on the former work, the previous shuffle task is improved as another isolated component, which is called shuffle service in Map Reduce.

Shuffle service evolves from shuffle task. This service is a common component integrating copy and merge operations and offering service for all the reduce tasks on the same Task Tracker.

One of the advantages by implementing shuffle service is that the number of random HTTP request to copy intermediate data for map tasks decreases for that the request from reduce located on the same node could be combined as only one request.

3) Tuning I/O scheduling policy on Map sides:

Based on the shuffle service, a better disk accessing policy is designed for decreasing the amount of random disk I/O request. With this policy Task Tracker shows more powerful supporting capability of reading and sending data to reduce task. The below shown architectural diagram in figure 3 provides the architectural idea of our proposed system.

Firstly, this policy gains more efficient disk read/write performance and decrease the number of random I/O requests. Secondly, based on the shuffle service with efficient reading policy, the network bandwidth and memory could be utilized fully and reasonably.

VI. CONCLUSION:

In this paper, we observed that in the Map Reduce Procedure Of Hadoop certain difficulties exists in achieving the appropriate performance and we have proposed the fair schedulers which takes care of eminent scheduling which in turn parallel promotes the prefetching and finally improving the shuffling in the reduce task by providing it as a service. The proposed system will be further extended in areas of scheduling and also in the shuffling phase as a part of our future work.

REFERENCES


