isrjournals

International Journal of Advanced Research in Computer Science Engineering and Information Technology

Volume: 6 Issue: 3 Dec,2016,ISSN_NO: 2321-3337

Job Scheduling based on Size to Hadoop

First M.MD.Shahbaz Hussain, Second R.Vidhya

Abstract— **Job scheduling based on size with aging has been recognized as an effective approach to guarantee near optimal system response times. HFSP scheduler introducing this technique to a real, multi-server, complex and widely used system such as Hadoop. Job scheduling according to size requires a priori job size information, which is not available in Hadoop and estimates it on-line during job execution. Size based scheduling in HFSP adopts the idea of giving priority to small jobs that they will not be slowed down by large ones. HFSP is a size based and preemptive scheduler for Hadoop. HFSP is largely fault tolerant and tolerant to job size estimation errors. Here Scheduling decisions use the concept of virtual time and cluster resources are focused on jobs according to their priority, computed through aging. This protocol never faces Starvation Problem for small and large jobs.**

Key words: **MapReduce, Performance, Data Analysis, Scheduling, Master Slave, SRPT, FCFS, Process Sharing.**

I. INTRODUCTION

Hadoop is a Java-based programming framework and free that supports the processing of large data sets in a Parallel and distributed computing environment. It makes Use of the commodity hardware Hadoop is Highly Scalable and Fault Tolerant. Hadoop runs in cluster and eliminates the use of a Super computer. Hadoop is the widely used big data processing engine with a simple master slave setup.
Big Data in most companies are processed by Hadoop by submitting the jobs to Master. The Master distributes the job to its cluster and process map and reduce tasks sequentially. But now a days the growing data need and the and competition between Service Providers leads to the increased submission of jobs to the Master. This Concurrent job submission on Hadoop forces us to do Scheduling on Hadoop Cluster so that the response time will be acceptable for each job.

In this work, we mainly focus the problem of job scheduling, that is how to allocate the available resources of a particular cluster to the number of concurrent jobs according to their specific resources, and focus on Hadoop, the most widely adopted open-source implementation of MapReduce. Currently, there are two main strategies used to schedule jobs. The first strategy is to divide the cluster resources into equal among all running jobs. A remarkable example of this type strategy is Hadoop Fair Scheduler .While this type of method preserves fairness and consistency among jobs, when the system is overloaded, it may increase the response time of jobs. The second type strategy is to serve one job at a time, thus avoiding the resource splitting. This is similar example to First-In-First-Out (FIFO) strategy, in which the job that arrived first is served first. The problem with this strategy is that, the job size is not given importance and blind to size, whatever the available scheduling choices available may lead inevitably to poor performance and inconsistency: as small jobs are possible to find large jobs in queue, thus they may incur in response times that are not proportionate to their size. As a consequence, the interaction is difficult between them to obtain. Both strategies have drawbacks that prevent them from using it directly in production without taking precautions. Commonly, a manual configuration of both the scheduler and the parameters of system are required to overcome such drawbacks. This causes the manual setup of a number of "pools" to divide all the resources in to different job categories, and fine-tuning of parameters governing the resource allocation. This process is error prone, and not possible to adapt easily to the change in workload composition and in cluster dimensions. In addition to that it is often the case for clusters to get over-dimension. This

simplifies resource allocation but has the disadvantage due to its costly deployments and the maintenance for resources which are left unused.

We present the new proposed design of a new protocol for scheduling that caters both to a fair and efficient utilization of cluster resources, while trying to achieve shorter response times. Our solution implements a size-based, preemptive scheduling discipline. Our scheduler allocates resources of cluster such that job size information is inferred while the job makes progress towards its own completion. Scheduling decisions use the method called virtual time and resources of cluster which are focused on priority of jobs and computed through aging. This ensures that neither small nor large jobs suffer from starvation. The output of our work progresses as a full-fledged scheduler implementation that integrates seamlessly in Hadoop named HFSP. Size based scheduling in HFSP adopts the idea of giving priority to small jobs so that they will not get slowed down by large jobs The Shortest Remaining Processing Time (SRPT) policy, which prioritizes jobs that need the least amount of work to complete i.e., the one that minimizes the mean response time (or sojourn time), that is the time that is passed between the submission of job and the time of completion of job. We Extend HFSP to pause jobs with Higher SRPT and allow other waiting jobs in Queue based on FCFS.

II. PROBLEM STATEMENT

In this section, we describe the problem the problem with this strategy is that, job size is not considered and blindly followed, the scheduling choices which are available lead inevitably to poor performance and inconsistency, while in PS resources are divided equally so that each active job keeps progressing. In loaded systems, these disciplines have severe shortcomings: in FCFS, large running jobs may leads to delay significantly the smaller ones; in Process Sharing, each additional job delays the completion of all the others. Both FCFS and Process sharing strategies have their own drawbacks that prevent them from being used in production without precautions.

III. USER CLASSES AND CHARACTERISTICS

The contribution of our work can be summarized as follows

A. Big Data and Environment

Huge Collection of data is retrieved from open source datasets that are publicly available from major Application Providers like Amazon. Big Data Schemas were analyzed and a Working Rule of the Schema is determined. The CSV (Comma separated values) and TSV (Tab Separated Values) files are Stored in HDFS (Highly Distributed File System) and were read through Master and manipulated using Java API that itself developed by us which is developer friendly, light weighted and easily modifiable.

B. Running a Batch Job through FCFS

A batch job is a backend job running in Hadoop clusters and also called as long running jobs as it is scheduled to process bulk data so that the application would makes use of the results produced for updating. Sample jobs are submitted to Hadoop master and Hadoop master will run the jobs based on a well-known technique called First come first serve manner (FCFS).Parallel execution of job is done by Hadoop cluster and the results are shown through a well-known Framework called Map Reduce. The Mapper task is done first in slave nodes and reduce task will be done in Master to throw the output.

C. Size based Scheduling on Concurrent jobs

Here n number of jobs are submitted to the Hadoop Master and Master will schedule the jobs based on FCFS and PS in a hybrid way. The Capacity of cluster will be analyzed

so as to share resources between concurrent jobs arriving to Master. A threshold will be maintained to balance load in slaves and Resource scheduling will not be done further if limit is reached. The Arriving jobs will put in queue until resource gets free in cluster.

D. Extending HFSP for job mistreatment ie.Starvation

As jobs may find long waiting time in queue, we extend our hybrid Approach which clubs FCFS and PS to put running jobs on hold for some time, if the particular job has high Shortest Remaining Processing Time (SRPT).Depending upon aging of the waiting jobs and SRPT the long running jobs may be put on hold and the waiting jobs which have high priority will be executed for a while and constant evaluated for SRPT for new jobs to arrive for execution. Our Proposed methodology shows high throughput in job completion.

E. Traditional Scheduling

First Come First Serve (FCFS) and Processor Sharing (PS) are arguably the two most simple and ubiquitous scheduling methods in use in many of systems; for instance, FIFO and Fair are two schedulers for Hadoop, the first inspired by FCFS method and the second by PS method . In FCFS, jobs are scheduled in the order of their submission, while in PS resources are divided equally among all, so that the job which is active keeps progressing. In loaded systems, these types of mentioned process have severe shortcomings that is one case is in FCFS, large running jobs can delay significantly small ones; in PS, job delays are unpredictable and this additional delays of jobs affects the job completion of all the others. In order to improve the performance of the system in terms of delay, it is important to mind the size of jobs. Job scheduling based on size adopts the idea of giving priority to small jobs.

F. Hadoop Fair Sojourn Protocol

The Hadoop Fair Sojourn Protocol (HFSP) is a Job scheduling based on size with aging for Hadoop. Implementing HFSP raise number of challenges: a few of them come from MapReduce itself and the fact that a job is composed by tasks – while others come from the scheduler which is a size based in a context where the size of the jobs is not known a priori. In this section we describe the challenges which has to face and the proposed solutions. Jobs: In MapReduce, jobs are scheduled according to the of tasks which it handles, and they consist of two phases, called MAP and REDUCE. We evaluate the sizes of jobs by executing the subset of tasks for each job; however, REDUCE tasks can be launched only after the MAP phase is getting complete. Our scheduler thus divides the job logically in to two phases and treats them independent and individually so the scheduler assumes the job as consisting of two parts with two different sizes, one for the MAP and the other for the REDUCE phase. When a resource is provided available for scheduling the MAP (resp. REDUCE) task, the scheduler sorts and evaluates jobs according to their specific virtual MAP (resp. REDUCE) sizes, and provides resources to the job with smallest size for that phase.

G. The Aging module

The aging module takes as input the estimated sizes to compute virtual sizes. The use of virtual sizes is a technique applied in many practical implementations of well-known schedulers it consists in keeping track of the amount of the remaining work for each job phase in a virtual “fair” system, and update it every time the scheduler is called. The output is that, although if the job not receive resources and thus its real size does not decrease, in the virtual system the job virtual size slowly decreases with time.

H. Scheduling Policy

In this section we describe how the estimation and the aging modules coexist to create a Hadoop scheduler that strives to be both efficient and fair.

Job Submission: Figure 1 shows the lifetime of a job in HFSP, from its submission to its completion and removal from the job queue. When a job is submitted, for each phase of the job, the scheduler asks to the estimation module if that phase is tiny. If the answer is affirmative, the scheduler assigns $S_f = 0$, meaning that the job must be scheduled as soon as possible. Otherwise, the scheduler starts the training stage and sets the virtual time to S_i which is an initial size given by the module of estimator. Periodically, the scheduler asks to the estimation module if it has completed its training stage, and, if the answer is positive, it notifies the aging module to update the virtual size of that job and removes the job from the training stage.

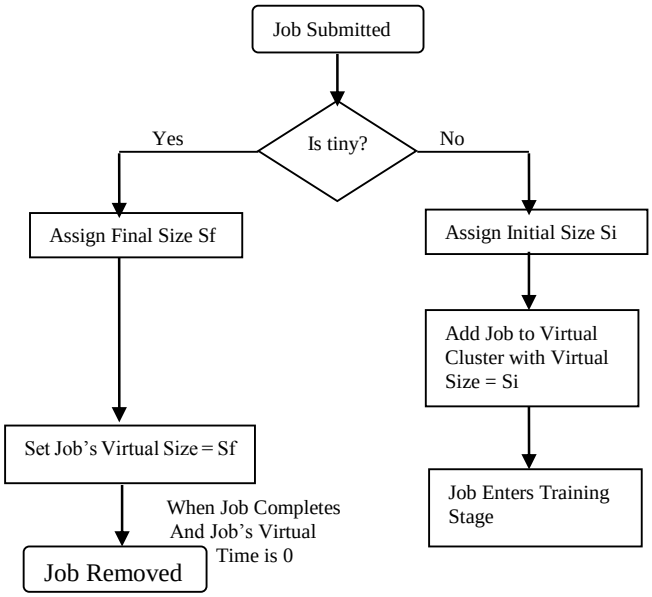

Fig. 1: Job lifetime in HFSP

Priority to the Training Stage

The training stage is important because, as discussed in Section III-A, the initial size Si is imprecise, compared to the final size Sf. Completing the training stage as soon as possible is fundamental for an efficient scheduling policy. There are two strategies that are used by the scheduler to speed up the training stage: the first strategy is to set a low number of training tasks t, as discussed in Section III-A; the second strategy is to give priority to the training tasks across jobs – up to a threshold of T 2 [0; Tmax] where Tmax is the total number of resources in the cluster. Such threshold avoids starvation of "regular" jobs in case of a busty job arrival pattern. When a resource is free and there are jobs in the training stage, the scheduler assigns the resource to a training task independently from its job position in the job queue. In other words, training tasks have the highest priority. Conversely, after a job has received enough resources for its training tasks, it can still obtain resources by competing with other jobs in the queue.

Virtual Time Update

When a job phase completes its training stage, the scheduler asks to the estimation module the final size Sf and notifies the aging module to update the virtual size accordingly. This operation can potentially change the order of the job execution. The

scheduler should consider the new priority and grant resources to that job, if such job is the smallest one in the queue. Unfortunately, in Hadoop MapReduce the procedure to free resources that are used by the tasks, also known as task preemption, can waste. Work. The default strategy used by HFSP is to wait for the resources to be released by the working tasks. Section III-E describes the preemption strategies implemented in HFSP and their implications.

IV. ALGORITHMS USED

Algorithm 1 HFSP resource scheduling for a job phase.

```
Function ASSIGNPHASETASKS (resources)
   For all Resources 2 resources do
     If 9 (Job in training stage) and Tcurr < T then
       Job Select job to train with smallest initial
         Virtual Size
       ASSIGN(s, job)
        Icurr      Icurr + 1
     Else
       Job select job with smallest virtual time
       Assign(s, job)
     End if
   End for
End function

Function ASSIGN (resource, job)
  If task is a training task then
    Icurr     Icurr
  End if
End function
```

Scheduling Algorithm

HFSP Scheduling which is invoked every time a MapReduce Slave Claims work to do the MapReduce master-behaves as described by algorithm 1. The Main function that is AssignPhaseTasks is responsible for assigning tasks for a certain phase. First it checks if there are jobs in training stage for that phase. IF there are any, and the number of current resources used for training tasks $T_]$ is smaller or equal than T, the scheduler assigns the resources to the first training task of the smallest job. Otherwise, the scheduler assigns the resource to the job with the smallest virtual time. When a task finishes its work, the procedure release resource is called. If the task is a training task, then the number T_{curr} of training slots in use is decreased by one.

SRPT Algorithm Calculation

```
Function CALCSRPT (Jobid, Result)
   Take input of jobid as SRPT.Map.Get (jobid)
    Assign noOfRcrds to stk.nextToken
    Assign floatvalue to (noOfRcrds)/ (totCount)
    Assign Calculate percent to (floatvalue * 1000)
          If   SRPTControlMap.Size is greater than 1
            While SRPTCalMap.size is greater than 2
                  Create Thread and sleep for 1sec
            End while
```

```
                Create Array List Arr [Jid] and input all jobid' s
                Sort Array List Arr [Jid]
                Assign ii to SRPTCalcMap.KeySet
                While ii.hasNext
                    Assign Mapkey to ii.Next
                    Assign val to SRPTCalcMap.get (Mapkey)
                    If Bigjobval equals val
                        Return Val as Bigjob val
                End While
            End if
    End Function
```

Here Jobid is the ids of Jobs and Result is a String variable. SRPTMap.Get returns id's of jobs, then the Number of Records and Percent are calculated. If SRPTControlMap.size is greater than 1 then it checks for SRPTCalMap.size is greater than 2 and then create a thread which sleeps for 1 sec and create an Array List which should be sorted with jobid' s. Get MapKey and compare with BigJobVal.

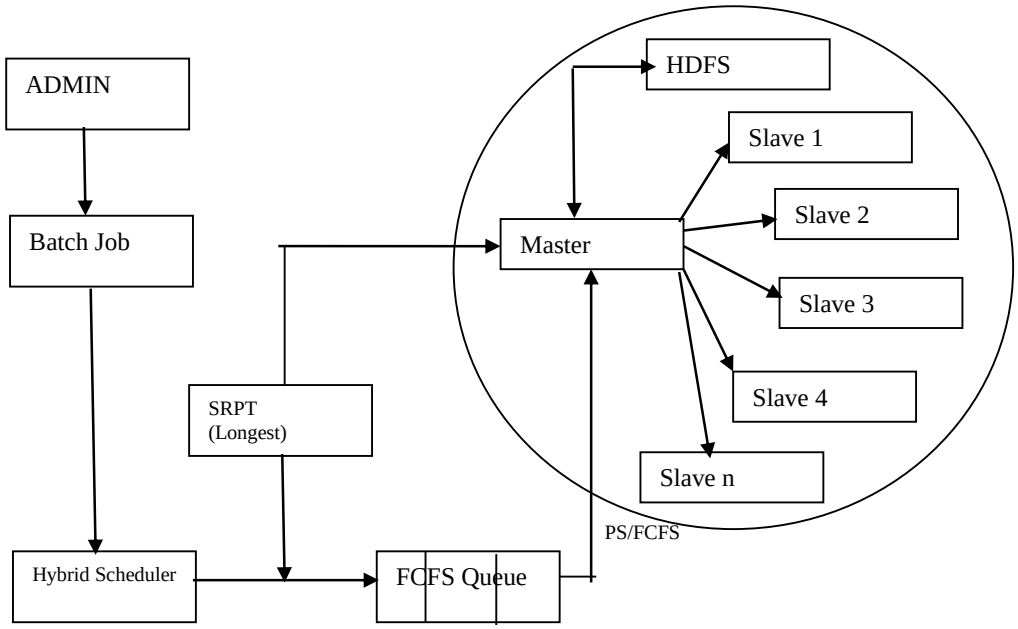

Fig. 2: Architecture of Job Scheduling

V. HOW WE DEAL

The new proposed design of a new protocol for scheduling that caters both to a fair and efficient utilization of cluster resources, while trying to achieve shorter response times. Our solution implements a size-based, preemptive scheduling discipline. Here Admin is the superior authority to submit jobs to Hybrid Scheduler. Whatever the jobs are present, it collects and submits the Batch jobs and send those to Hybrid Scheduler. Hybrid Scheduler sends all batch jobs to FCFS Queue where it follows its own policy which is not at all related to Hybrid Scheduler. In FCFS Queue whatever the batch job arrives first that is submitted to Master. Master is linked to all its available Slaves with in its environment. Master follows Process Sharing policy where it shares resources to all its freely available resources. It counts the time of each resource processed in slave.

Whenever the slave crosses the estimated time then that job will be preempted and send to SRPT Longest queue which in turn submits to FCFS Queue whenever the FCFS Queue reduces its size by 1. Our scheduler allocates resources of cluster such that job size information is inferred while the job makes progress towards its own completion. Scheduling decisions use the concept of virtual time and cluster resources are focused on priority of jobs and computed through aging. This ensures that neither small nor large jobs suffer from starvation. The output of our work progresses as a full-fledged scheduler implementation that integrates seamlessly in Hadoop named HFSP. Size based scheduling in HFSP adopts the idea of giving priority to small jobs so that they will not get slowed down by large jobs The Shortest Remaining Processing Time (SRPT) policy, which prioritizes jobs that need the least amount of work to complete i.e., the one that minimizes the mean response time (or sojourn time), that is the time which is passed between the submission of job and its time of completion. We Extend HFSP to pause jobs with Higher SRPT and allow other waiting jobs in Queue based on FCFS

VI. CONCLUSIONS

Resource allocation plays an increasingly important role in current Hadoop clusters, as modern data analytics and workloads are becoming more complex and heterogeneous. Our work was motivated by the increasing demand for system responsiveness, driven by both interactive data analysis tasks and long-running batch processing jobs, as well as for a fair and efficient allocation of system resources.

Alas, system responsiveness and fairness requirements have been traditionally at odds: a scheduling discipline that would satisfy one, had to sacrifice the other. For example, in our work we argued that the default scheduling mechanism used in typical Hadoop deployments, the Fair scheduler, achieves fairness but trades on system response times. Only a tedious, manual process involving an expert administrator could miti-gate the shortfalls of a processor sharing-like discipline, albeit for a rather static workload composition.

In this paper we presented based on the idea of job scheduling based on size. Here the full-fledged scheduler that is known as HFSP(Hadoop Fair Sojourn Protocol), which implements a job scheduling based on size that satisfies system fairness and responsiveness requirements.

Our work raised many challenges

Evaluating jobs and their sizes on-line without wasting or deviating resources, avoiding job starvation for all kinds of jobs that is both small and large jobs, and guaranteeing shorter response times of jobs despite estimation errors were the most noteworthy. HFSP uses a practical design: estimation of size trades, starvation is largely alleviated and accuracy for speed and starvation by introducing the mechanisms of virtual time and aging.

A large part of this article was dedicated to a thorough experimental campaign to evaluate the benefits of HFSP when compared to the default Fair scheduler in Hadoop. We defined several realistic workloads that are representative of typical uses of an Hadoop cluster, and proceeded with a comparative analysis using our deployment, configured according to current best practices. Our experiments, that amount to more than 1500 real jobs, indicated that HFSP systematically – and in some cases, by orders of magnitude – outperformed the Fair scheduler, both with respect to system response times and fairness properties.

VII. FUTURE ENHANCEMENTS

Currently, we are extending HFSP such that it can use recent job preemption primitives, a necessary condition to allow even faster response times; moreover, we will consolidate our codebase and contribute it to the Hadoop community, casting HFSP to work for modern frameworks such as YARN and Mesos.

REFERENCES

1. Apache, "Hadoop: Open source implementation of MapReduce," http: //hadoop.apache.org/.
2. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in Proc. of USENIX OSDI, 2004.
3. Apache, "Spark," http://spark.apache.org/.
4. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, 2012, pp. 2–2.
5. Microsoft, "The naiad system," https://github.com/ MicrosoftResearchSVC/naiad.
6. D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, "Naiad: A timely dataflow system," in Proceedings of the 24th ACM Symposium on Operating Systems Principles, 2013, pp. 439– 455.
7. Y. Chen, S. Alspaugh, and R. Katz, "Interactive query processing in big data systems: A cross-industry study of MapReduce workloads," in Proc. of VLDB, 2012.
8. K. Ren et al., "Hadoop's adolescence: An analysis of Hadoop usage in scientific workloads," in Proc. of VLDB, 2013.
9. G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones." in NSDI, vol. 13, 2013.
10. Apache, "Oozie Workflow Scheduler," http://oozie.apache.org/.
11. Katarina Grolinger, Michael Hayes, Wilson A.Higashino, "Challenges for MapReduce in Big Data".
12. Matei Zaharia, Mosharaf Chowdary, Tathagata Das, Ankur, Scott Shenker, "Resilent Disturned Datasets: A Fault-Tolerant , In-Memory Cluster Computing".
13. Kai Ren, YongChul Kwon "Hadoop's Adolscence"
14. Ganesh Ananthanarayanan, Ali Ghodsil, Scott shenker , Ion Stoical "Effective Straggler Mitigation : Attack of Clones"
15. Jeffrey Dean and Sanjay Ghemawat "MapReduce: Simplified Data Processing on Large Clusters"