International Journal of Combined Research & Development (IJCRD)
eISSN:2321-225X;pISSN:2321-2241 Volume: 4; Issue: 3; March -2015

Hadoop Fair Sojourn Protocol Based Job Scheduling On Hadoop

Mr. Dileep Kumar J. S. [1] Mr. Madhusudhan Reddy G. R. [2]

[1]Department of Computer Science, M.S. Engineering College, Bangalore, Karnataka

[2]Assistant professor, Department of Computer Science, M.S. Engineering College Bangalore, Karnataka

ABSTRACT: ***We present Hadoop Fair Scheduler Protocol, a scheduler introducing this technique to a real, multi-server, complex and widely used system such as Hadoop. The scheduling discipline is based on the concepts of virtual time and job aging. These techniques are conceived to operate in a multi server system with tolerance to failures, scale-out upgrades, and multi-phase jobs a peculiarity of Map Reduce. The results indicate that size-based scheduling is a realistic option for Hadoop clusters, because HFSP sustains even rough approximations of job sizes. Which are based on realistic workloads generated via a standard benchmarking suite, pinpoint at a significant decrease in system response times with respect to the widely used Hadoop Fair scheduler, without impacting the fairness of the scheduler, and show that HFSP is largely tolerant to job size estimation errors.***

Keywords: Map Reduce, Performance, Data Analysis, Scheduling.

1. INTRODUCTION

Map Reduce has become a popular model for data-intensive computation in recent years. By breaking down each job in to small map and reduce tasks and executing them in parallel across a large number of machines, Map Reduce can significantly reduce the running time of data-intensive jobs. However, despite recent efforts toward designing resource-efficient map reduces schedulers, existing solutions that focus on scheduling at the task-level still offer sub-optimal job performance. The is because task can have highly varying resource requirements during their lifetime, which makes it difficult for task-level schedulers to effectively utilize available resources to reduce job execution time. The advent of large-scale data analytic, fostered by parallel frameworks such as Hadoop, Spark, and Naiad, has created the need to manage the resources of compute cluster operating in a shared, multi-tenant environment. Within the same company, many users share the same cluster because this avoids redundancy in physical deployments and in data storage, and may represent enormous cost savings. Initially designed for few very large batch processing jobs, data-intensive scalable computing frameworks such as map reduce are nowadays used by many companies for production, recurrent and even experimental data analysis jobs. The heterogeneity is substantiated by recent studies that analyze a variety of production-level workloads. An important fact that emerges from previous works is that there exists a stringent need for short system response times. Many operations, such as data exploration, preliminary analyses, and algorithm tuning, often involve interactivity, in the sense that there is a human in the loop seeking answers with a trial-and-error process. In addition, workflows schedulers such as Oozie contribute to workload heterogeneity by generating a number of small "orchestration" jobs. At the same time, there are many batch jobs working on big datasets: such jobs are a fundamental part of the workloads, since they transform data in to value transform data in to value. Due to the heterogeneity of the workload, it is very important to find the right trade-off in assigning the resources to interactive and batch jobs.

The solution implements a size-based, preemptive scheduling discipline. The scheduler allocates cluster resources such that job size information which is not available a prior is inferred while the job makes

progress toward its completion, scheduling decisions use the concept of virtual time, in which jobs make progress according to an aging function cluster resources are "focused" on jobs according to their priority, computed through aging. This ensures that neither small nor large jobs suffer from starvation. The outcome of our work materializes as a full-fledged scheduler implementation that integrates seamlessly in Hadoop: we called our scheduler HFSP, to acknowledge an influential work in the size-based scheduling literature.

2. Literature Survey

It shows existing system and how to overcome the existing system explanation of proposed system and its components.

An important fact that emerges is that there exists a stringent need for short system response times. Many operations, such as data exploration, preliminary analyses, and algorithm tuning, often involve interactivity, in the sense that there is a human in the loop seeking answers with a trial-and-error process. In addition, workflow schedulers such as Oozier [1] contribute to workload heterogeneity by generating a number of small "orchestration" jobs. At the same time there are many batch jobs working a big datasets: such jobs are a fundamental part of the workloads, since they transform data into value. Due to the heterogeneity of the workloads, it is very important to find the right trade-off in assigning the resources to interactive and batch jobs.

There are mainly two different strategies used to schedule jobs in a cluster. The first strategy is to split the cluster resources equally among all the running jobs. A remarkable example of this strategy is the Hadoop Fair Scheduler [2], [3]. While this strategy preserves fairness among jobs, when the system is overloaded, it may increase the response times of the jobs. The second strategy is to serve one job at a time, thus avoiding the resource splitting. An example of this strategy is First-In-First-Out(FIFO), in which the job that arrived first is served first.

The problem with this strategy is that. Being blind to job size, the scheduling choices lead inevitably to poor performance: small jobs may find large jobs in the queue, thus they may incur in response times that are disproportionate to their size. As a consequence, the interactivity is difficult to obtain. Both strategies have drawbacks that prevent them from being used directly in production without precautions. Commonly, a manual configuration of the both the scheduler and the system parameters is required to overcome such drawbacks. This involves the manual setup of a number of "pools" to divide the resources to different job categories, and the fine-tuning of the parameters governing the resources allocation. This process is tedious, error prone, and cannot adapt easily to changes in the workload composition and cluster configuration. In addition, it is often the case for clusters to be over-dimensioned , this simplifies resource allocation (with abundance, managing resources is less critical), but has the downside of costly deployments and maintenance for resources for resources that are often left unused [4].

We present the design of a new scheduling protocol that caters both to a fair and efficient utilization of cluster resources, while striving to achieve short response time. Our approach satisfies both the interactivity requirements of small jobs and the performance requirements of large jobs, which can thus coexist in a cluster without requiring manual setups and complex tuning: our technique automatically adapts to resources and workload dynamics. Our solution implements a size-based, preemptive scheduling discipline. The scheduler allocates cluster resources such that job size information – which is not available a priori is inferred while the job makes progress towards its completion. Scheduling decisions use the concept of virtual time, in which jobs make progress according to an aging function: cluster resources are "focused" on jobs according to their priority, computed through aging. This ensure that neither small nor large jobs suffer from starvation. The outcome of the work materializes a full-fledge scheduler implementation that seamlessly in Hadoop [5]

3. Traditional Scheduling

Traditional Scheduling processor Sharing (PS) and First Come First Serve (FCFS) are arguably the two most simple and ubiquities scheduling disciplines in use in many system for instance, Fair and FIFO are two schedulers for Hadoop, the first inspired by FCFS, and the second by process scheduling. In FCFS, jobs are scheduled in the order

of their submission, while in PS resources are divided equally so that each active job keeps progressing. In loaded system, these disciplines have server shortcomings in FCFS, large running jobs can delay significantly small ones in PS, each additional job delays the completion of all the others. In order to improve the performance of the system in terms of delay, it is important to consider the size of the jobs. Size-based scheduling adopts the idea of giving priority to small jobs as such they will not be slowed down by large ones. The Shortest Remaining Processing Time (SRPT) policy, which prioritizes jobs that need the least amount of work to complete, is the one that minimizes the mean response time (or sojourn time), that is the time that passes between a job submission and its completion [6] the below figure provides an example that compares PS to SRPT in this case two small jobs –j2 and j3- are submitted while a large job j1 is running. While in PS in three jobs run (slowly) in parallel in a size-based discipline j1 is preempted.

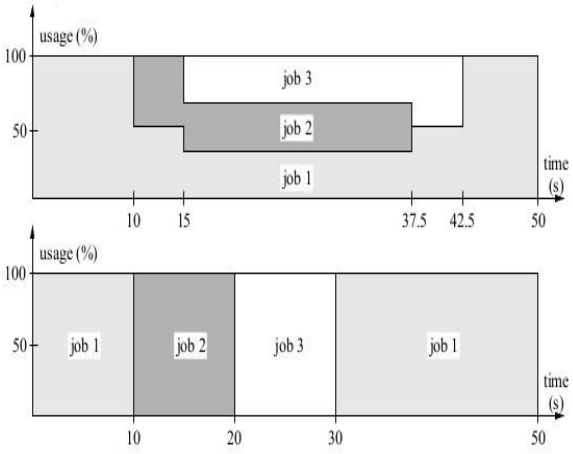
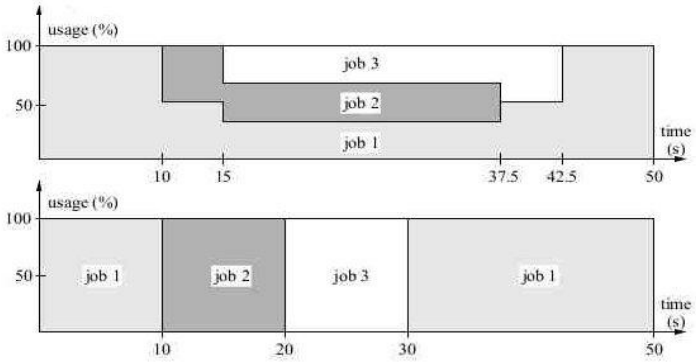


Figure 1.Comparison between PS and SRPT

The result is that j2 and j3 complete earlier. Like most size-based scheduling techniques, SRPT temporarily suspends the progress of lower-priority jobs fortunately, this is not a problem in a batch system like Hadoop, for which results are usable only after the jobs is completed. While policies like SRPT improve means response time, they may incur in starvation if small jobs are continuously submitted, large ones may never receive service[7],[8] this results in job mistreatment. To avoid starvation, a common solution is to perform job aging. With job aging, the system decrease virtually the size of jobs waiting in the queue, and keeps them sorted according to their virtual size, serving the one with the current smaller virtual size. Job size is perfectly known a priori, the FSP discipline exploits aging to provide a strong dominance fairness guarantee no job completes in FSP later than it would in PS.FSP also guarantees excellent results in terms of both job response time and fairness when job sizes are not know exactly for these reason, the design of HFSP is guided by the abstract ideas beyond FSP.

4. Hadoop Fair Sojourn Protocol (HFSP)

The Hadoop Fair Sojourn Protocol (HFSP) is a size-based scheduler with aging for Hadoop. Implementing HFSP raises a number of challenges a few come from Map Reduce itself. Example the fact that a job is composed by tasks while others come from the size based nature of the scheduler in a context where the size of the jobs is not known a priori.



Jobs: In Map Reduce jobs are scheduled at the granularity of tasks and they consist of two separate phases called MAP and REDUCE. We evaluate job sizes by running a subset of sample tasks for each job, however, reduce tasks can only be launched only after the Map phase is complete. The scheduler thus splits logically the job in the two phases and treats them independently therefore the scheduler consider the job as composed by two parts with different sizes, one for the MAP and the other REDUCE phase. When a resource is available for scheduling a MAP (resp. REDUCE) task, the scheduler sorts jobs based

on their virtual Map (resp. REDUCE) sizes, and grants the resource to the job with the smallest size for that phase.

Estimated and virtual size: The size of each phase to which we will refer as real size, ia unknown until the phase itself is complete. The scheduler therefore works using an estimated size starting from this estimate the scheduler applies job aging I.e., it computes the virtual size, based on the time spent by the job in the waiting queue. The estimated and the virtual sizes are calculated by two different modules the estimation module, that outputs the estimated size and the aging module that takes in input the estimated size and applies an aging function.

5. Modules in phase

1. The Estimation Module
2. The Aging Module
3. The Scheduling Policy

The Estimation Module: The role of the estimation module is to assign a size to a job phase such that, given two jobs, the scheduler can discriminate the smallest one for that phase. When a new job is submitted, the module assigns for each phase an initial size Si. Which is based on the number of its tasks .The initial size is necessary to quickly infer job priorities. A more accurate estimate is done immediately after the job submission, through a training stage in such a stage a subset of task called the training tasks is executed and their execution time is used to update Si to a final estimated size Sf. Choosing t induces the following trade-off a small value reduces the time spent in the training stage at the expense of inaccurate estimates a large value increases the estimation accuracy but result in a longer training stage. The scheduler is designed to work with rough estimates therefore a small t is sufficient for obtaining good performance.

$$\tilde{s} = \frac{1}{t}\sum_{k=1}^{t} \tilde{s}_k \,,$$

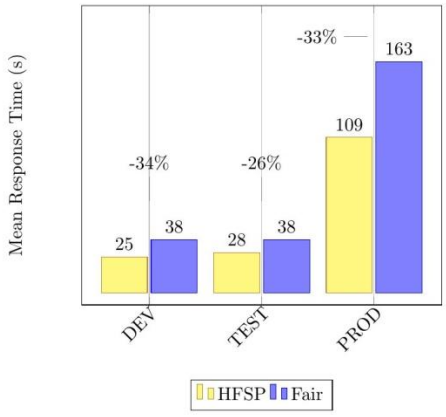
where

$$\tilde{s}_k = \begin{cases} s_k \text{ , if training task } t_k \text{ completes within } \Delta \\ \frac{\Delta}{p_k} \text{ , otherwise} \end{cases}$$

The Aging Module: The aging module takes as input the estimated sizes to compute virtual sizes. The use of virtual size 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 result is that even if the job doesn't receive resources and thus its real size does not decrease, in the virtual system the job virtual size slowly decreases with time.

Job aging avoids starvation, achieves fairness, and requires minimal computational load, since the virtual size does not incur in costly updates.

The 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.



Aggregate *mean* response times for all workloads.

DEV: This workload is indicative of a "development" environment, where by users rapidly submit several small jobs to build their data analysis tasks, together with jobs that operate on larger datasets. This workload is inspired is inspired by the Facebook 2009 trace observed by Chen et al, the mean interval between job arrivals is μ = 30 s.

Test: This workload represents a "test" environment, whereby users evaluate and test their data analysis tasks on a rather uniform range of dataset sizes, with 20% of the jobs a large dataset. The mean interval between jobs is μ = 60 s.

PROD: This workload is representative of a "production" environment, whereby data analysis tasks operate pre-dominantly on large datasets. The mean interval between jobs is μ = 60 s.

6. Conclusion: The 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. We presented an novel approach to the resource allocation problem, based on the idea of size-based scheduling. The effort materialized in a full fledged scheduler that we called HFSP, the Hadoop Fair Sojourn Protocol, which implements a size-based discipline that satisfies simultaneously system responsiveness and fairness requirements. The work raised may challenges evaluating job sizes online without wasting resources avoiding job starvation for both small and large jobs, and guaranteeing short response times despite estimation errors were the most noteworthy.

Our Feature work is related to job preemption. We are currently investigation a novel technique to fill the gap between killing running tasks and waiting for tasks to finish. Indeed killing a task too late is a huge waste of work, and waiting for a task to complete when it just started is detrimental as well. Our next goal is thus to provide a new set of primitives to suspend and resume tasks to achieve better preemption.

References

[1] Apache, "Oozie Workflow Scheduler," http://oozie.apache.org/.

[2] 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 Pro- ceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, 2012, pp. 2–2.

[3] Apache, "The Hadoop fair scheduler," http://hadoop.apache.org/doc/r1.2.1/fair_scheduler.html.

[4] 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.

[5] E. Friedman and S. Henderson, "Fairness and efficiency in web server protocols," in proc. Of ACM SIGMETRICS, 2003.

[6] L.E. scharge and L. W. Miller, "The queue m/g/l with shorestest remaining processing time discipline,"

[7] stalling,operating system. Prentice hall, 1995.

[8] Microsoft, "The naiad system," https://github.com/. MicrosoftResearchSVC/naiad.

[9] 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.

[10] [7] K. Ren et al., "Hadoop's adolescence: An analysis of Hadoop usage in scientific workloads," in Proc. of VLDB, 2013.

[11] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones." in NSDI, vol. 13, 2013