CS 239 Final Report
An Alternative Scheduling Algorithm for Apache Spark

Chia-Hung Ni
nch83@cs.ucla.edu

Chun Chen
joris@cs.ucla.edu

Yu-Chen Lin
ericlin8545@cs.ucla.edu

Nurrachman Liu
rachliu@cs.ucla.com

November 2019

Abstract

We explore the job scheduling policies of Spark and implement a new self-designed one. In the evaluation section, we analyze each scheduling policy and compare the differences between them.

1 Introduction

As data volumes that requiring processing grow dramatically, researchers have come up with different big data frameworks to provide efficient data processing solutions, including Hadoop[6], Dryad [2] and MapReduce[1].

Among big data systems, Spark[8] is a distributed processing program widely used in both industry and research environments. A Spark cluster runs user-submitted programs as jobs; each user job is further broken down into tasks. The tasks are submitted to a single cluster-wide *TaskManager* that is responsible for allocating these tasks to JVM execution units (*executors*).

One main theme of big data system optimization focuses on scheduling algorithms. Intuitively, a good scheduling algorithm focused on productivity will ensure a high usage rate of the CPU. Many previous works try to achieve this, such as the decentralized solution Sparrow [4], and Google's Borg system [7].

There are two main scheduling schemes within Spark on the task level - FIFO and FAIR. FIFO is straightforward in that the task that is submitted first will be run first. On the other hand, FAIR will keep switching between tasks to achieve fair resource allocations. However, in our view, both scheduling schemes are somewhat myopic in that neither utilizes enough information from the task itself. Therefore, our work considers different attributes of the tasks and provides a harmonic scheduling solution that aims to achieve a balance between the default FIFO and FAIR scheduling schemes in Spark.

The rest of the paper is organized as follows. Section 2 briefly describes the existing Spark scheduler, the background context for this work. Section 3 describes our proposed Spark scheduler, *Harmony.* Section 4 shows our experimental results. Sections 5 and 6 conclude the paper and introduce some potential future work.

2 Background

We use several tools to efficiently evaluate how our scheduler works compared to the other built-in schedulers or potential scheduling algorithm candidates.

2.1 DAG Scheduler

The Spark internal system breaks the Spark job down into a directed acyclic graph (DAG), which is then passed to a *DAG Scheduler* entity that is responsible for breaking down the nodes of the DAG into the individual tasks submitted to a master host (*TaskManager*) that manages groups of tasks (*taskSets*).

2.2 TaskManager

There are different ways the *TaskManager* can allocate its given tasks to the available *executors.* Executors run one task per thread. Generally, a *Spark Scheduler* accepts two *Schedulable* tasks, and gives a boolean result for which one should be allocated first. Related tasks are bundled into a set (taskSet) or pool (*taskPool*). The specific task allocation policy employed is called the Scheduling Mode. Currently, Spark supports two scheduling modes for taskSets and taskPools: *FIFO Scheduling* and *Fair Scheduling.* Both are available for taskPools, while only the former, FIFO, is available for taskSets.

In the FIFO Scheduler, tasks are sorted based on their priority value; lower integer priorities are allocated first. If two priorities are tied, the tie-break goes to the task which is at a lower stage number; this gives taskSets with tasks at earlier stages a chance to complete them. The priority number is assigned using the task's entry time.

For the Fair Scheduler, tasks are compared based on a parameter *minShare*, an integer representing the minimum number of tasks that **every** taskSet should be running at any time. Thus, this mode is 'fair' in that

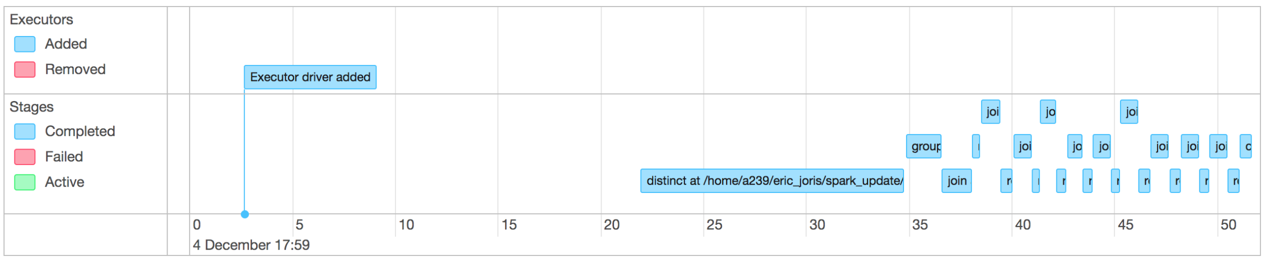

Figure 1: FIFO with light workload

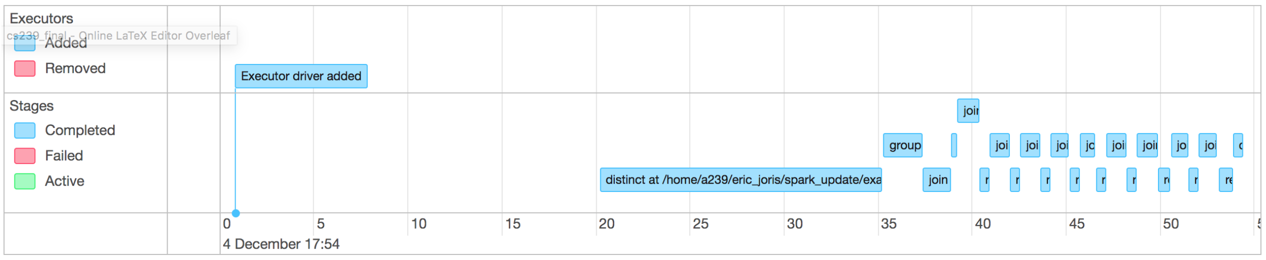

Figure 2: Harmony with light workload

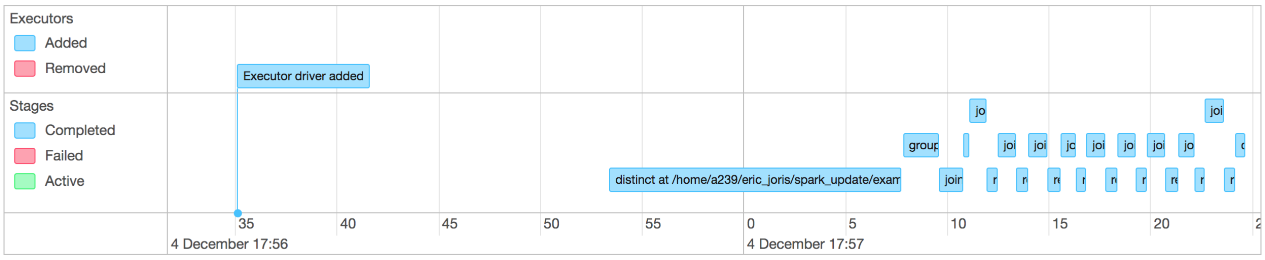

Figure 3: FAIR with light workload

it equalizes the total number of tasks each taskSet runs at all times. When any taskSet falls below its configured minimum running tasks, it is fed tasks until it is satisfied. When every taskSet is satisfied (i.e., is running its minimum configured number of tasks), priority goes to the taskSet with the least number of running tasks. This gives more resources to taskSets that are less busy.

2.3 PageRank

We are using PageRank [5] as our sample workload. PageRank is an algorithm that evaluates the popularity of a webpage according to its hyperlink characteristics. Calculating PageRank involves processing a huge graph, so it is an ideal sample workload for our scheduler.

3 Harmony Scheduler

We propose a scheduler that prioritizes the tasks that have completed the least amount of work, weighted by priority. Currently, the Scheduling algorithm interface contains several metrics: priority, stage, number of running tasks, and min share of resources. However, these metrics are not considered in totality as an overall figure of merit that can be minimized. We combine them as:

$$HarmonyScore = \\ (Priority) * (stage) * \frac{RunningTask}{minShare} * \frac{1}{weight} \tag{1}$$

This new metric allows us to prioritize the tasks in a general manner by calculating the Harmony Score. With the consideration of priority and stage, this metric maintains the characteristics of FIFO scheduling. To prevent starvation, we add the ratio of number of running tasks and the min share resources, to the assigned weight of the task. In this way, if enough running tasks are assigned, the overall priority will be downgraded, leaving the scheduling opportunity to others. In the meantime, if the weighted running tasks all meet their given min share equally, this metric reduces to a balance between FIFO and fair scheduling.

4 Evaluation

We employ a leased cloud-based Linux server to evaluate our Spark scheduler implementation. The server comes with 80 GB of SSD disk space, 4 GB of RAM, and 2 Intel Skylake CPUs, running in Ubuntu 18.04 x64.

We generate three kinds of workloads for PageRank to evaluate the characteristics of the various scheduling algorithms. The light workload consists of 1K vertices and 0.1M edges. The medium workload consists of 2K vertices and 1M edges. The heavy workload consists of 3K vertices and 1M edges.

In the experiments, we apply the three scheduling policies (FIFO, FAIR, and Harmony) to several workloads. We find that Harmony can achieve a balance between the other two. Figures 1, 2, 3 show the results of each scheduling policy. For FIFO, the jobs' running processes are separated into many small pieces; we think this is because the running priority of each job is easily preempted, due to the FIFO scheduling policy of comparing the natural priority of each job. For the FAIR scheduling policy, the running priority is shared between jobs more fairly, because of its similarity to round-robin scheduling. For our Harmony scheduling policy, since we designed it to combine characteristics of FIFO and FAIR policies, we observe through our results that it is indeed a balance between FIFO and FAIR. In the result, the jobs' running process shows that it has a preempted characteristic similar to that of the FIFO policy, while also having the fairness of the FAIR policy.

5 Conclusion

In this paper we introduce our scheduling algorithm that takes the task count into consideration in the overall figure of merit. We then compared our algorithm with the existing ones in Spark.

6 Future Work

6.1 Aging

In our current design, we don't consider the age of a task. Arguably, our design already has a minShare parameter for each pool, implying that each pool will always receive a minimum allocation of resources. However, minShare doesn't guarantee that tasks have reasonable response times; i.e., the Harmony score doesn't account for desired response latency. A simple solution would be to store a task's initial queue time, then factor this time (age) into its Harmony score. However, achieving an optimal balance between age and the other existing factors of the Harmony score may take some more effort.

6.2 DAG Characteristic as Scheduling Factor

By taking consideration of the DAG characteristics when scheduling the task, we believe that it is possible to further optimize the scheduling. However, this entails a large effort of refactoring the Spark program to allow for task scheduling to collaborate with the DAG scheduler.

7 Acknowledgments

We thank our Professor H. Xu and TA A. Padmanabhan for teaching and guiding about big data systems, and to work on this research.

References

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA, 2004.

[2] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[3] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[4] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: Distributed, low latency scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 69–84, New York, NY, USA, 2013. ACM.

[5] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[6] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[7] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[8] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets.