Hadoop MapReduce Scheduling Paradigms

Roger Johannessen, Anis Yazidi, Boning Feng
Oslo and Akershus University College
Department of Computer Science
Oslo, Norway

***Abstract*—Apache Hadoop is one of the most prominent and early technologies for handling big data. Different scheduling algorithms within the framework of Apache Hadoop were developed in the last decade. In this paper, we attempt to provide a comprehensive overview over the different paradigms for scheduling in Apache Hadoop. The surveyed approaches fall under different categories, namely, Deadline prioritization, Resource prioritization, Job size prioritization, Hybrid approaches and recent trends for improvements upon default schedulers.**

Keywords—Apache Hadoop, Map Reduce, Scheduling Paradigms.

I. INTRODUCTION

Bollier stated that "Big websites can generate terabytes of raw log data every day. The sheer size of its data set has led to the emergence of new cloud infrastructures, characterized by the ability to scale to thousands of nodes, fault tolerance and relaxed consistency" [1]. From 2005 to 2020, the digital universe is expected to grow dramatically by a factor of 300, from 130 exabytes to 40 trillion gigabytes, i.e more than 5,200 gigabytes per person in 2020. Moreover, the digital universe is expected to double every two years [2]. A big part of the growth is a defining trait of our current technology landscape - the Internet of Things, quickly evolving into; "The Internet of Everything" [3]. The benefits of "all interconnected" devices is immense in terms of potentially huge increase in quality of life. In the same time, it brings along the challenge of handling extreme amounts of data. Devices generate nowadays vast amounts logging data, but also functional data such as media streams that are key to the sole purpose of the device. Data is becoming the world's new natural resource [4]. The challenges represented by big data handling can be divided into three groups:

- Velocity
- Volume
- Variety

Hadoop possesses a sophisticated set of methods that handle the above challenges elegantly through the use of:

- Hadoop Common (a set of utilities and libraries)
- The file system called HDFS (Hadoop File System)
- YARN (Yet Another Resource Negotiator)
- MapReduce (a framework for distributing tasks and parallel processing)

II. RELATED WORK

In this section, we will survey some promising new scheduling algorithms, as well as some highly-cited and well established ones. We have categorized those scheduling algorithms by the type of the scheduling priority.

A. Deadline prioritization

1) Deadline Constraint Scheduler: Prioritizing deadline in a Hadoop clusters is done by predicting the completion time of jobs/tasks and then allocating them to nodes capable of processing them withing a time limit where the data is actually useful. The research reported in [6] was motivated by the fact that FIFO, the default scheduling algorithm of Hadoop clusters, has some visible drawbacks due to its rigid prioritization scheme. The paper explores real time cluster scheduling based on user specified deadlines. The authors give a preliminary evaluation of their algorithm reckoned as Deadline Constraint Scheduler, which is a scheduler that simply ignores new tasks that can not be processed within their deadline. This is achieved by calculating the deadline and comparing it to the execution time. The latter approach performs well in a homogeneous cluster, but is invalidated in a heterogeneous cluster where execution times might vary across nodes. In this case, the algorithm relaxes some of its parameters so that to allow processing times to be decoupled from the slowest node. However, this might lead to under-utilization of certain nodes in the cluster. The authors concede that they will address this issue in future work. To calculate schedulability, the work calculates the minimum amount of map tasks to get a job done, and compares it to the maximum amount of reduce tasks. If there is less available reduce slots than the maximum amount of possible reduce tasks in the job, the task gets dropped. Experimental results show greater task efficiency during MapReduce phases [6].

2) Cloud Least Laxity First: In another approach introduced in the paper entitled *A Deadline Scheduler for Jobs in Distributed Systems* [16], the authors propose an interesting deadline scheduler for Hadoop called Cloud Least Laxity First (CLLF), that orders tasks based on laxity (time left over to deadline, after task is finished). They argue that by using this technique one can reduce the amount of nodes needed while maintaining total execution time at acceptable levels. This was proven by comparing the algorithm to Time Shared and Space Shared scheduling in a controlled environment. The devised scheduler handles soft deadlines by introducing a penalty term as a function of the lateness (lateness defined as completion time minus deadline). The authors describe a system in which each worker node uses a FIFO-queue and notifies the master as soon as it has an idle processor. The master has the role of hosting the CLLF-algorithm and allocating tasks. As the authors themselves describe it: "The general idea of the algorithm is to sort the cloudlets by laxities (the first has the lowest one). Giving this sorted list, the algorithm takes the first element of this list and looks for a host that locally have the data of the cloudlet and which also have at least one free slot. If one matching host is found, the task is ran on it, otherwise, the algorithm restart the same procedure using the second element of the list" [16]. As in the case of the algorithm reported in [6] which falls in this category, this algorithm [16] is limited to homogeneous environments. In line with goal of reducing the amount of virtual nodes needed for job executions, the work [16] shows a considerable decrease in missed deadlines compared to Time Shared and Space Shared algorithms, thus, increasing the disparity in effectiveness of the latter two algorithms.

B. Resource prioritization

1) Coupling Scheduler: Algorithms delving into resource allocation and optimization of utilization of worker nodes are perhaps the most investigated topic within the field of MapReduce. As processing and handling of data is getting more and more centralized, virtualized Hadoop clusters seem to be the future of MapReduce and has emerged as enabler for business ventures through the cloud. In [5], it was argued that the widely used Fair Scheduler has a starvation problem involving the Map and Reduce operations. The paper focuses on coupling the two progresses, instead of treating them separately. The authors [5] also performed a performance comparison, proving that Coupling Scheduler performs better than Fair Scheduler in handling tasks with varying map service times. The main effect this has on resource usage is that reducers are gradually launched as more and more maps are completed, instead of allocating a quantity of reducers based on predicted need, i.e as seen in [6]. The benefits is that there is virtually no under-utilization of resources in the cluster, making the approach [6] energy efficient. However, Coupling Scheduler ignores job sizes and can therefore be inefficient when processing jobs with large map service times. This is due to the coupling nature of "sticky processor sharing" [5], where a map task gradually gets the amount of reducers it needs, disregarding task completion time, potentially allowing huge tasks to complete before allocating resources to smaller tasks [5].

2) Triple-Queue Scheduler and MR-predict: While [5] pre-emptively couples mappers and reducers regardless of task size, the authors behind the paper *A Dynamic MapReduce Scheduler for Heterogeneous Workloads* [7] have devised a prediction method called MR-predict to detect workloads in real time. MR-predict focuses on optimizing the utilization and balance between I/O-bound vs CPU-bound applications, which is not a concern in legacy Hadoop MapReduce. Based on MR-Predict, which classifies a type of workload, they propose Triple-Queue Scheduler to serve tasks based on the predicted workload. The standard First Come First Served strategy would not be able to handle scheduling different task types, as it has a single queue. With the Triple-Queue-Scheduler the authors solve this issue by paralleling queues, and delivering I/O bound tasks to nodes with I/O resources to spare, while at the same time serving CPU bound tasks to fitting nodes. MR-predict checks the history of a job to predict the future tasks, and from there describe the workload type. If a new job is received with no previous history, the job is sent to a waiting queue withing Triple-Queue-Scheduler, where the scheduler will assign one map task of that job to every TaskTracker whenever it has idle slots. When the map tasks finish, MR-predict calculates the MTCT (Map Task Completed Time), MID (Map Input Data) and MOD (Map Output Data) based on the data gathered from these tasks. The type of workload then gets determined, and the job is moved into either a I/O-Bound queue, or a CPU-Bound queue. Furthermore, the scheduler monitors tasks assigned to queues, checking if MTCT increases. If MTCT increase passes the threshold of 140%, the scheduler determines that the task was assigned to the wrong queue, and moves the task to an alternative queue. The tests were run on a native Hadoop cluster, running TeraSort, GrepCount and WordCount. The authors observed a 20% increase in resource utilization and an impressive 30% increase in throughput. This is naturally only meant for heterogeneous workloads, as there would be little use to predict a homogeneous job flow. The algorithm is also exclusively useful in a homogeneous environment [7].

3) Workload Characteristic and Resource Aware Scheduler: In this paper, the authors propose WCRA-scheduling of Hadoop clusters (Workload Characteristic and Resource Aware) [12]. WCRA-scheduling checks the CPU, RAM and I/O-load on the nodes first. Afterwards, all the tasks are sorted based on Estimated Completion Time then scheduled on the most fitting nodes. The work bears some similarity to [7], but also embraces RAM as an important parameter, ensuring that more than 25% of the primary memory is always available before scheduling a job. The authors argue that "is critical in case of CPU and Disk I/O bound tasks" [12]. It was found that "compute node works significantly if it has the available physical memory greater than 25%. Tasks are assigned to the node if the memory availability is greater than 25%" [12]. WRCA-scheduling has the benefit of being specifically designed to handle heterogeneous clusters. In a similar manner to [7], WRCA works by first completing a set of sample tasks for a new job in order to determine predicted type of workload, and then classifies the job as either CPU-bound or I/O-bound. Compared to [7], more extensive testing with more jobs were reported, albeit on a smaller cluster environment, and actually reached the same amount of increase in throughput (30%) when compared to FIFO, Fair-scheduler and Capacity scheduler.

4) Adaptive resource allocation schedulers: By definition, an adaptive resource allocation scheduler adapts to the capabilities and performance of each node in the cluster individually. In [14], the authors argued that legacy resource-aware schedulers give nodes a fixed amount of resources for each job, potentially causing over/under-utilization of resources, in contrast to their devised scheduler [14] which dynamically adapts its resource allocation over the course of the job. Based on the estimated amount of tasks that can be processed concurrently on each node, the devised algorithm shrinks/extends the amount of resources over the run time. This algorithm is also designed to optimize a heterogeneous cluster as in [7]. Instead of predicting tasks and queueing the workloads, the authors propose a strategy where the worker nodes and their "available/lack resources (CPU and memory) are monitored, and based on this, the scheduler will extend/shrink the capacity of the TaskTracker by increasing/decreasing the number of map/reduce slots of the TaskTracker" [14]. The approach was tested using different benchmarks including TeraSort, PiEstimator and WordCound, in a similar manner to the main stream of papers in this category. According to the experimental findings, it was observed an average increase of the completion time of all tasks by around 30%. As in the case of [16], this approach also yields an increased effectiveness with reduced nodes / slots in the cluster, compared to native algorithms like FIFO and Fair-scheduler [14].

C. Job size

1) Size-Based Scheduling: As was remarked in a series of papers such as in [5], ignoring the job size might halt throughput in cluster, although it is very resource-effective. Weighing job-size first should then logically considerably increase throughput in the cluster and is claimed to achieve "near-optimal system response times" [15]. The hard part about designing an algorithm focused on size, is that it has to prioritize jobs/tasks with the shortest remaining completion time to be effective. This can lead to bigger tasks starving to get resources. The authors behind *HFSP: Size-based Scheduling for Hadoop* [15] introduce HFSP, which is a scheduler that lets Hadoop determine job size during execution in real time. They claim that their 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" [15]. To be able to schedule tasks with short completion times without forcing starvation of larger tasks, the author implement a common aging policy, where the cost of a task in the queue gets gradually decremented as it waits for resources. They call the technique "Shortest Remaining Virtual Time (SRVT)" [15]. SRVT results in a slight increase in average throughput time, at the benefit of virtually eliminating errors and starvation in the queue. By applying a size-based scheduling algorithm the authors also argue that the scheduler has significantly reduced overhead, as its only concern is the direct size of the job, and no additional calculations are necessary. As seen in [12] and [7], this scheduler determines size by running a small set of sample tasks from a job. The approach [15] is endowed with a preemptive estimation module that sets a coarse size value for the job before the samples are processed, which gets gradually refined as samples are completed. The authors have measured the performance of their approach in a benchmarking suite, and found a significant decrease in system response times. Contrary to [16] and [14], this effectiveness disparity increased in larger jobs and larger clusters [9].

2) LsPS (Leveraging size Patterns Scheduler): The work reported in [15] proposes an algorithm that specializes in handling bursty workloads in a multi-user environment, by tuning resource shares among users, and even the scheduling algorithm for each user, based on job size. The authors have tested their algorithm both in a controlled environment, and in a production cluster: Amazon EC2, and observed reduced MapReduce job response times. The job tracker calculates how many slots and resources each user should have based on the history of task completion, and predicted completion time based on job size. Every time a task is finished, the statistics of that particular user are updated. Based on all this data, the Job Tracker continually sorts users instead of tasks, making sure to let the most efficient users get the most slots, without starving other users. This is done by granting a slot share ratio to the users that is inversely proportional to their job average sizes. In the case of new users entering the cluster have no history for determining average job size / completion time. In this case predefined job profiles are added to the scheduler, and assigned to users based on a couple of user-defined criteria. The authors have chosen a FIFO-algorithm as a fallback, in case of tasks getting the same cost and confusing LsPS, making sure that the first task submitted simply gets processed. The experimental results show huge promise in enterprise environments, with multiple users, heterogeneous clusters and heterogeneous workloads. For smaller clusters and predictable workloads however, the overhead cost might be too high [15].

D. Improving Native Hadoop

1) Fair and Efficient Slot Configuration and Scheduling: In many cases enterprises just want to use Hadoop for parallel big data processing out-of-the-box, without much configuration by experts. This frequently leads to using native Hadoop schedulers which may be inefficient. The team behind *FRESH: Fair and Efficient Slot Configuration and Scheduling for Hadoop Clusters* [10] argue that Hadoop is far too complex to tweak for many users. The authors propose a new Hadoop scheduling system called FRESH (FaiR and Efficient Slot scheduling for Hadoop) that dynamically configures slots and assign tasks to achieve optimal performance from a cluster. They introduce two different algorithms, one which statically assigns slots for each job submitted, and one that dynamically alters the amounts of slots for a job during run time. The static algorithm calculates optimal amounts of slots for each job, allocates slots for nodes, and then hands the jobs over to Fair Scheduler to server tasks to the nodes. The dynamic algorithm takes the whole process without help for Fair scheduler, and acts as both back end slot allocator and task server for nodes, dynamically monitoring each node and making sure all tasks are processed with optimal *fairness*. The authors present their own novel definition of fairness, named *overall fairness* with an algorithm that more accurately disperses resources between jobs. The reported tests show a significant improvement, especially with the dynamic slot allocation, increasing makespan by up to 30% compared to Fair scheduler across all types of workloads [10].

2) Chronos: Instead of creating a totally new "default" scheduler from scratch, a different approach for enhancing the native Hadoop scheduler is proposed by the authors of *Chronos: Failure-Aware Scheduling in Shared Hadoop Clusters* [11]. The authors argue that the performance of Hadoop systems in part depends on how failures are handled. Hadoop handles failures by re-executing all the tasks of the failed machines. In this case, the machines need to wait for resources to execute recovery tasks. They argue that the fact that this is black-boxed from Hadoop schedulers (not visible or configurable for schedulers) may hinder the schedulers from optimizing the workflow according to the scheduling goal efficiently, and thus significantly reducing performance of the cluster. In order to counter this problem, the authors introduce Chronos, a failure-aware scheduling strategy that preemptively allocates resources to nodes with task failures, and also considers data locality for optimized performance. Chronos is an optional component independent from schedulers, and works together with the scheduler of a users choice (like native

Hadoop - FIFO or Fair). Chronos works basically by listening to heartbeats from Hadoop. Upon a failure, Chronos queries the JobTracker for the nodes that has task failures. Chronos then attempts to "inject" the recovery tasks into the front of the task queue, based on an algorithm working together with the scheduler goal to determine priorities of tasks. When it finds an appropriate slot, it then allocates resources away from less important tasks in the queue and on to the recovering node. The latter strategy allows the failed slots to preemptively be freed, instead of waiting in starvation for recovery resources. The authors tested Chronos in combination with FIFO and Fair-scheduler and experienced a reduce in job completion times by up to an astonishing percentage of 55%.

E. Hybrid approach

1) Resource and Deadline-aware Job Scheduling: As aforementioned, there is a vast variety of interesting and effective scheduling algorithms with different prioritization, and different drawbacks. In general term, hybrid approaches aspire to combine one or more different approaches so that to distill the best of their combination [13]. In [13], the authors introduce a hybrid algorithm that takes both task deadlines and a predicted future resource availability into account when allocating tasks. To achieve this they apply a receding horizon control algorithm in combination with a self-learning model that learns to predict an estimate of future resource availability and job completion times. They do this by introducing control intervals in which actual resources and job sizes and predicted resources and job sizes gets calculated, and based on this they optimize the schedule while evaluating deadlines. This is especially useful in an environment where resources are dynamic and heterogeneous, as resources can be added or taken away during run time, and the controls will catch the updates and optimize for it. Tested in a controlled environment against Fair scheduler, the authors were able to reduce the penalty of deadline misses by 36%, and against Earliest Deadline First scheduler they show a reduced penalty of 10% [13].

2) Classification and Optimization based Scheduler: A truly hybrid solution is introduced in [8]. The authors analyze the performance of widely used schedulers like FIFO and Fair Share Scheduler, and an algorithm reckoned as COSHH (Classification and Optimization based Scheduler for Heterogeneous Hadoop) scheduler. Based on the performances of these algorithms the paper introduces a hybrid solution where all three algorithms are used in the same cluster based on different system loads. "When the system is underloaded, and the number of free slots is greater than the number of waiting tasks, the scheduler switches to the FIFO algorithm. Here, the simple FIFO algorithm can improve the average completion time with minimum scheduling overhead. However, as the system load increases such that the available number of slots is less than the number of waiting tasks, the hybrid scheduler selects the Fair Sharing algorithm. When the load increases such that the system is overloaded, and the number of waiting tasks in job queues is quickly increasing, the Fair Sharing algorithm can greatly increase the average completion time. Therefore, the scheduler switches to the COSHH algorithm which improves the average completion time, while avoiding considerable degradation in the fairness metric" [8]. The hybrid scheduler chooses the best scheduling algorithm for different scales of jobs and resources to address average completion time and fairness" [8]. The results of the experiments are thoroughly documented, showing that the approach works, in numerous cases cutting scheduling and completion-time by half [8].

III. Conclusion

After reviewing a number of related works on Hadoop scheduling, the conclusion that stands out is the potential for improvement in the default Hadoop scheduling algorithms. Almost all the surveyed schedulers in this paper have advantages in terms of fairness and completion time compared to the default Hadoop scheduling policy. Interestingly, FRESH [10] and COSHH-hybrid [8] have the potential to become a native part of Hadoop, replacing FIFO and Fair sharing, as well as Chronos [11] which holds a lot of promise while still needing further testing. When it comes to large enterprise environments, LsPS [15] represents a promising approach as it delivered unprecedented performance and user control in a scalable and dynamic cluster, vastly improving upon default schedulers.

References

[1] Bollier, D., & Firestone, C. M. (2010). *The promise and peril of big data (p. 1).* Washington, DC: Aspen Institute, Communications and Society Program.

[2] Gantz, J., & Reinsel, D. (2012). *The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east.* IDC iView: IDC Analyze the future, 2007, 1-16,

[3] Richards, N. M., & King, J. H. (2013). *Three paradoxes of big data.*

[4] Huang, G., He, J., Chi, C. H., Zhou, W., & Zhang, Y. (2015, June). *A Data as a Product Model for Future Consumption of Big Stream Data in Clouds.* In 2015 IEEE International Conference on Services Computing (SCC), (pp. 256-263). IEEE.

[5] Tan, J., Meng, X., & Zhang, L. (2012, March). *Performance analysis of coupling scheduler for mapreduce/hadoop.* In INFOCOM, 2012 Proceedings IEEE (pp. 2586-2590). IEEE.

[6] Kc, K., & Anyanwu, K. (2010, November). *Scheduling hadoop jobs to meet deadlines. In 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)* (pp. 388-392). IEEE.

[7] Tian, C., Zhou, H., He, Y., & Zha, L. (2009, August). *A dynamic mapreduce scheduler for heterogeneous workloads.* In 2009 Eighth International Conference on Grid and Cooperative Computing (pp. 218-224). IEEE.

[8] Rasooli, A., & Down, D. G. (2012, November). *A hybrid scheduling approach for scalable heterogeneous hadoop systems.* In High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion: (pp. 1284-1291). IEEE.

[9] Pastorelli, M., Barbuzzi, A., Carra, D., Dell'Amico, M., & Michiardi, P. (2013, October). *HFSP: size-based scheduling for Hadoop.* In 2013 IEEE International Conference on Big Data (pp. 51-59). IEEE.

[10] Wang, J., Yao, Y., Mao, Y., Sheng, B., & Mi, N. (2014, June). *Fresh: Fair and efficient slot configuration and scheduling for hadoop clusters.* In 2014 IEEE 7th International Conference on Cloud Computing (pp. 761-768). IEEE. ISO 690

[11] Yildiz, O., Ibrahim, S., Phuong, T. A., & Antoniu, G. (2015, October). *Chronos: Failure-aware scheduling in shared hadoop clusters.* In 2015 IEEE International Conference on Big Data (Big Data) (pp. 313-318). IEEE.

[12] Divya, M., & Annappa, B. (2015, July). *Workload characteristics and resource aware Hadoop scheduler.* In 2015 IEEE 2nd International Conference on Recent Trends in Information Systems (ReTIS) (pp. 163-168). IEEE.

[13] Cheng, D., Rao, J., Jiang, C., & Zhou, X. (2015, May). *Resource and deadline-aware job scheduling in dynamic hadoop clusters.* In 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 956-965). IEEE.

[14] Elkholy, A. M., & Sallam, E. A. (2014, December). *Self adaptive Hadoop scheduler for heterogeneous resources.* In 2014 9th International Conference on Computer Engineering & Systems (ICCES) (pp. 427-432). IEEE.

[15] Yao, Y., Tai, J., Sheng, B., & Mi, N. (2015). *LsPS: A Job Size-Based Scheduler for Efficient Task Assignments in Hadoop.* IEEE Transactions on Cloud Computing, 3(4), 411-424.

[16] Perret, Q., Charlemagne, G., Sotiriadis, S., & Bessis, N. (2013, March). *A deadline scheduler for jobs in distributed systems.* In 2013 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA) (pp. 757-764). IEEE.