INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume3, Issue2)

Dynamic job ordering and slot configurations For MapReduce workloads

Prathamesh Chaudhari
Yadavrao Tasgaonkar College of Engineering & Technology, Maharashtra, India
pcpnc1996@gmail.com

Gaurav S. Salve
Yadavrao Tasgaonkar College of Engineering & Technology, Maharashtra, India
gsalve24@gmail.com

Nilesh Ghadge
Yadavrao Tasgaonkar College of Engineering & Technology, Maharashtra, India
nileshmaratha94@gmail.com

Harish Barapatre
Yadavrao Tasgaonkar College of Engineering & Technology, Maharashtra, India
ncnsc1967@gmail.com

Abstract: MapReduce is a popular parallel computing paradigm for large-scale data processing in clusters and data centers. A MapReduce workload generally contains a set of jobs, each of which consists of multiple map tasks followed by multiple reduce tasks. Due to 1) that map tasks can only run in map slots and reduce tasks can only run in reduce slots, and 2) the general execution constraints that map tasks are executed before reduce tasks, different job execution orders and map/reduce slot configurations for a MapReduce workload have significantly different performance and system utilization. This paper proposes two classes of algorithms to minimize the makespan and the total completion time for an offline MapReduce workload. Our first class of algorithms focuses on the job ordering optimization for a MapReduce workload under a given map/reduce slot configuration. In contrast, our second class of algorithms considers the scenario that we can perform optimization for map/reduce slot configuration for a MapReduce workload. We perform simulations as well as experiments on Amazon EC2 and show that our proposed algorithms produce results that are up to 15 80 percent better than currently unoptimized Hadoop, leading to significant reductions in running time in practice.

Keywords: MapReduce, Hadoop, Flow-Shops, Scheduling Algorithm, Job Ordering.

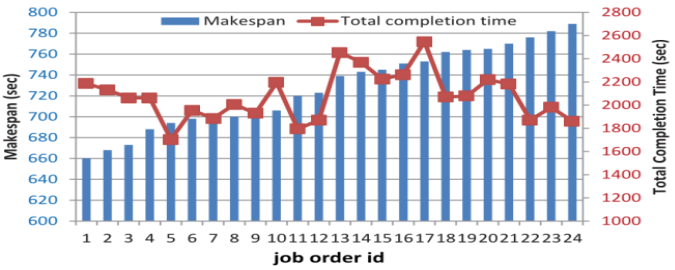
1. INTRODUCTION

MAPREDUCE is a widely used computing model for large scale data processing in cloud computing. A MapReduce job consists of a set of map and reduces tasks, where reduce tasks are performed after the map tasks. Hadoop [2], an open source implementation of MapReduce, has been deployed in large clusters containing thousands of machines by companies such as Amazon and Facebook. In that cluster and data center environments, MapReduce and Hadoop are used to support batch processing for jobs submitted from multiple users (i.e., MapReduce workloads). Despite many research efforts devoted to improving the performance of a single MapReduce job (e.g., [3], [11]), there is relatively little attention paid to the system performance of MapReduce workloads. Therefore, this paper tries to improve the performance of MapReduce workloads.

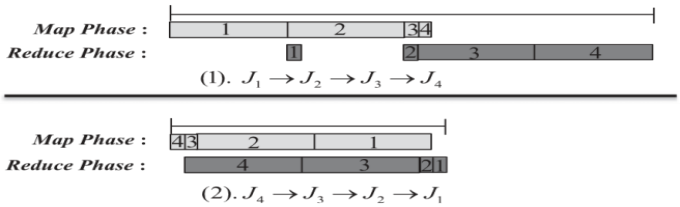
Makespan and total completion time (TCT) are two key performance metrics. Generally, makespan is defined as the time period since the start of the first job until the completion of the last job for a set of jobs. It considers the computation time of jobs and is often used to measure the performance and utilization efficiency of a system. In contrast, total completion time is referred to as the sum of completed time periods for all jobs since the start of the first job. It is a generalized makespan with queuing time (i.e., waiting time) included. We can use it to measure the satisfaction of the system from a single job's perspective through dividing

In this paper, we target at one subset of production MapReduce workloads that consist of a set of independent jobs (e.g., each of jobs processes distinct data sets with no dependency between each other) with different approaches. For dependent jobs (i.e.,

MapReduce workflow), one MapReduce can only start only when its previous dependent jobs finish the computation subject to the input-output data dependency. In contrast, for independent jobs, there is an overlap computation between two jobs, i.e., when the current job completes its map-phase computation and starts its reduce-phase computation, the next job can begin to perform its map-phase computation



(a) The experimental results for a batch of 4 jobs run on the Amazon EC2 Hadoop cluster under all 4! = 24 job execution •rders.



(b) MapReduce execution flow for different job orders.

Fig. 1. Performance comparison for a batch of jobs under different job

2. MOTIVATION

In this section, we show the importance and challenge of job ordering optimization as well as map/reduce slot configuration optimization by giving motivating examples experimentally.

2.1 Motivation for Job Ordering Optimization

To motivate the importance of job ordering optimization for MapReduce workloads on performance, we ran a Testbed workload consisting of four jobs (J_1 J4) from Table 2 of Section 7.1 in an Amazon EC2 Hadoop cluster configured with map slots of 57 and reduce slots of 19. We do so by comparing the performance of all possible job submission orders.

2.2 Motivation for Slot Configuration Optimization

To motivate the importance of optimization on map/reduce slot configuration, we perform a simulation experiment with a Testbed MapReduce workload consisting of four jobs, assuming that a cluster consisting of 10 slave nodes each configured with eight slots, i.e., the total number of map slots plus reduce slots for the cluster is 80, as shown in Fig. 2.

Our experiments are three folds. First, we examine the influence of slot configuration to the overall performance by running jobs in all possible map/reduce slot configurations in practice (i.e., configure map slots from 1 to 7 per slave node), under an arbitrary job submission order. The experimental results are given in Fig. 2a. It can be noted that the maximum performance difference between the worst-case map/reduce slot configuration (e.g., 10=70) and the optimal one (e.g., 60=20) is huge, up to $\frac{3;112 \quad 624}{624}$ 399%. Second, we consider the influence of job orderings on the performance by running jobs with all possible job orders, under an optimal map/reduce slot configuration (e.g., 60/20). Fig. 2 b shows that the performance difference of the worst-case job submission order and the optimal one can be large up to $\frac{768-578}{578}$ 33%, depending on the workload characteristic. Third, we evaluate and compare the optimal (minimum) makespan as well as its corresponding optimal map/reduce slot configuration for all possible job submission orders. Fig. 2c illustrates the optimal map/reduce slot configuration (i.e., blue and green bar) as well as its corresponding optimal makespan (i.e., red curve) for all 4! ¼ 24 possible job submission orders, sorted by makespan in non-decreasing order. The results show that there are varied optimal configurations of map/reduce slots for different job submission orders. Moreover, it's worth noting that the maximum performance difference between the worst-case job order and the optimal job order each under its corresponding optimal map/reduce slot configuration, is $\frac{744-578}{578}$ 28:7%.

In summary, the above motivating example poses three key challenging issues: (1). Different map/reduce slot configurations will have different performance under a given job order (2). Even under the optimal map/ reduce slot configuration, different job submission orders will result in varied performance (3). The optimal configurations of map/reduce slots, as well as its corresponding optimal makespan, are different under different job submission orders

3. RELATED WORK

In this section, we give an overview of related work from two aspects. First, we review batch job ordering optimization work in HPC literature. Second, we summarize the MapReduce job optimization work proposed in recent years.

3.1 Job Ordering Optimization

The batch job ordering problem has been extensively studied in the high-performance computing literature [25]. Minimizing the makespan has been shown to be NP-hard [25], and a number of approximation and heuristic algorithms (e.g., [14], [34]) have been proposed. In addition, there has been work on bi-criteria optimization which aims to minimize makespan and total completion time simultaneously, such as [13].

The previous works all focused on the single-stage parallelism, where each job only has a single stage. In contrast, MapReduce is an interleaved parallel and sequential computation model [23] which is related to the two-stage hybrid flow shop (2HFS) problem [17]. Minimizing the makespan for 2HFS is strongly NP-hard when at least one stage contains multiple processors [16]. There has been a large body of approximation and heuristic algorithms (e.g., [6], [24]) proposed for 2HFS. Additionally, there has been work (e.g., [31]) targeted at the bi-criteria optimization of both makespan and total completion time.

The main difference between MapReduce and traditional 2HFS is that MapReduce jobs can run multiple map and reduce tasks concurrently in each phase, whereas 2HFS allows at most one task to be processed at a time. In this way, MapReduce is more similar to the two-stage hybrid flow shop with multiprocessor tasks (2HFSMT) [28], [29] problem, which allows a task at each stage can be processed on multiple processors simultaneously. However, there is a requirement in 2HFSMT that a task at each stage can be scheduled only when the number of processors it requires is satisfied; otherwise, the task needs to wait [28]. In contrast, the number of running map/reduce tasks for a MapReduce job can be dynamically scaled up and down as idle map/ reduce slots become available.

In summary, MapReduce is a new computation model that is similar to but different from other models mentioned above. The works that are most related to ours are [26], [41]. In [26], Moseley et al. present an offline 12approximation algorithm for minimizing the total flow time of the jobs; this is the sum of the differences between the finishing and arrival times of all the jobs. Verma et al. [41] propose two algorithms for makespan optimization. One is a greedy algorithm job ordering method based on Johnson's Rule. Another is a heuristic algorithm called Balanced Pool. They discuss and evaluate the algorithms experimentally. We follow their job ordering approach (i.e., the MK_JR algorithm in our paper). But our main contributions go beyond it in a number of significant aspects. First, we prove a 1 þ d upper bound on the approximation ratio of our MK_JR algorithm. Second, we give the relationship between upper-bound makespan, lower-bound makespan, and the corresponding job orders.

3.2 MapReduce Job Optimization

There is a large body of research work that focuses on the optimization for MapReduce jobs. One optimization policy focuses on the architectural design and optimization issues. Jiang et al. [21] proposed a set of general low-level optimizations including improving I/O speed, utilizing indexes, using fingerprinting for faster key comparisons, and block size tuning. Rasmussen et al. [33] presented an I/O-efficient MapReduce system called Themis that improves the performance of MapReduce by minimizing the number of I/O operations. Likewise, Sailfish [32] improves MapReduce's performance through more efficient disk I/O. It mitigates partitioning skew in MapReduce by choosing the number of reduce tasks and intermediate data partitioning dynamically at runtime, using an index constructed from intermediate data. There are also methods that reduce I/O cost in MapReduce by using indexing structures (e.g., Hadoop++ [12]), column-oriented storage (e.g., [15]). Polo et al. [30] proposed a scheduling technique and implemented a prototype called Adaptive Scheduler that can adaptively manage the workload performance with the awareness of hardware heterogeneity, distributed storage to meet user's deadline requirement. Wolf et al. [42] propose a flexible scheduling allocation scheme called FLEX, which can optimize any of a variety of standard scheduling theory metrics, such as response time, stretch, makespan. Tang et al. [35], [37] proposed a dynamic slot allocation system called Dynamic MR to improve the performance for the slot-based Hadoop MRv1, by allowing map (or reduce) tasks can be run on map slots and reduce slots.

Adjusting Hadoop configuration is another optimization policy, including [7], [18], [19]. For example, Starfish [19] is a self-tuning framework that can adjust the Hadoop's configuration automatically for a MapReduce job such that the utilization of Hadoop cluster can be maximized, based on the cost-based model and sampling technique. Herodotou and Babu [18] propose a system named Elastisizer for cluster sizing optimization and MapReduce job-level parameter configurations optimization, on the cloud platform, to meet desired requirements on execution time and cost for a given workload, based on a careful mix of job profiling, estimation using black-box and white-box models and simulation. In contrast, Agarwal et al. [7] present a system RoPE that can re-optimize data parallel jobs by adapting execution plans based on estimates of code and data properties.

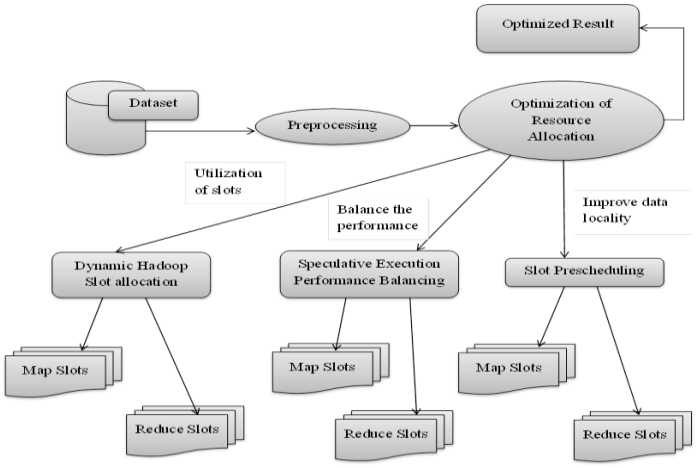
Another optimization policy is to share work and eliminate redundant data access and computation. Agrawal et al. [8] provide a method to maximize scan sharing by grouping MapReduce jobs into batches so that sequential scans of large files are shared among many simultaneous jobs as possible. MR Share [27] is a sharing framework that provides three possible work-sharing opportunities, including scan sharing, mapped outputs sharing, and Map function sharing across multiple MapReduce jobs, to avoid performing redundant work and thereby save processing time.

There is also an optimization policy of pipelining. MapReduce Online [10] is such a modified MapReduce system to support online aggregation for MapReduce jobs that run continuously by pipelining data within a job and between jobs.

In contrast, we improve the performance for a MapReduce workload by maximizing the cluster utilization as much as possible, through optimizing the map/reduce slot configuration and the job submission order. All these studies are complementary to our study and our approach can be incorporated into these modified MapReduce frameworks (e.g., MRShare [27], MapReduce Online [10]) for further performance improvement.

Moreover, there is a number of optimization works for MapReduce on the cloud, which primarily considers the deadline and budget, such as [20], [38], [39]. They optimize the task scheduling and resource allocation for MapReduce workloads by proposing algorithms and cost models for each metric. However, their work is atop of Hadoop system. We can combine these works and our approach to further optimize the deadline and budget for cloud computing.

4. SYSTEM ARCHITECTURE



4.1 Slot allocation and Slot pre-scheduling process

In this module, we are going to perform two processes. Slot allocation Slot pre- scheduling process.
In this slot allocation process, we are going allocate the slot based on dynamic Hadoop slot allocation optimization mechanism.
In the slot pre-scheduling process we are going to improve the data locality. Slot Pre-Scheduling technique that can improve the data locality while having no negative impact on the fairness of Map-Reduce jobs.
Some idle slots which cannot be allocated due to the load balancing constraint during runtime, we can pre-allocate those slots of the node to jobs to maximize the data locality.

4.2 Speculative Execution Performance Balancing
When a node has an idle map slot, we should choose pending map tasks first before looking for speculative map tasks for a batch of jobs.
Hadoop Slot is executed for determining the path for performing the MapReduce job. After this, the Speculative based process starts to execute the determined optimized Multi-execution path.
Executing individual MapReduce jobs in each datacenter on corresponding inputs and then aggregating results is defined as a MULTI execution path. This path used to execute the jobs effectively.

5. APPLICATIONS

1. Social Media: The Large data is generated from the social media platforms such as YouTube, Facebook, Twitter, LinkedIn, and Flickr. The amount of DATA being uploaded to the internet is rapidly increasing, with Facebook users uploading over 2.5 billion new Data every month. It can be used to improve applications performance by greatly reducing the file size and network bandwidth required to display your application.
2. Business Applications: online shopping application where the every item has data is shown. Company's data and scan copies of various documents.
3. Satellite images: This includes weather data or the data that the government captures in its satellite surveillance imagery.
4. Photographs and video: This includes security, surveillance, and traffic video.

6. CONCLUSION

This paper focuses on the job ordering and map/reduce slot configuration issues for MapReduce production workloads that run periodically in a data warehouse, where the average execution time of map/reduce tasks for a MapReduce job can be profiled from the history run, under the FIFO scheduling in a Hadoop cluster. Two performance metrics are considered, i.e., makespan and total completion time. We first focus on the makespan. We propose job ordering optimization algorithm and map/reduce slot configuration optimization algorithm. We observe that the total completion time can be poor subject to getting the optimal makespan, therefore, we further propose a new greedy job ordering algorithm and a map/reduce slot configuration algorithm to minimize the makespan and total completion time together. The theoretical analysis is also given for our proposed heuristic algorithms, including approximation ratio, upper and lower bounds on makespan. Finally, we conduct extensive experiments to validate the effectiveness of our proposed algorithms and their theoretical results.

