Large Data Processing in Clusters and Data Centers by Optimizing Makespan and Total Completion Time of MapReduce Workloads

Dr.K.Nagi Reddy,Professor in CSE, LORDS Institute of Engineering & Technology,
Himayathsagar, Hyderabad, India

ABSTRACT— ***Now a days, analyzing the large and unstructured data is a difficult task in many data sharing and processing areas. To process the large datasets or large data we have a most popular paradigm i.e MapReduce paradigm. A MapReduce workload generally includes a hard and fast of jobs; each of jobs consist of a couple of map and reduce tasks. To improve the performance of the MapReduce workloads, we want to beautify the makespan in addition to general final touch time of the MapReduce. In this paper, author has used two algorithms which are based on Johnson's Rule, to enhance the performance for MapReduce workloads with job ordering and slot configuration optimization approaches. The proposed algorithms are: one for optimizing the makespan by dynamic job ordering and another for reducing the total completion time of the MapReduce by slot configuration. By using the proposed algorithms, we can optimize or mitigate the makespan of the MapReduce and also we can mitigate the total completion time for the MapReduce workloads.***

Keywords: MapReduce, Clusters, Job Ordering, Slot Configuration, Makespan

1. INTRODUCTION

1.1 Introduction about MapReduce

The MapReduce programming model has been developing dramatically in reputation for some of years now [1], [2], [4], [5]. There are many motives for this success, most are associated with MapReduce's inherent simplicity of use, even when carried out to massive applications and installations. For example, a MapReduce is designed to parallelized routinely. It can be carried out on huge clusters of commodity hosts, and it inherently scales well. Scheduling, fault tolerance and essential communications are all treated robotically, without direct consumer help. Finally and possibly most significantly, the programming of MapReduce programs is surprisingly directly-ahead and thus appropriate for much less state-of-the-art programmers. These blessings, in turn, bring about lower prices [14]. Originally MapReduce aimed at large (typically periodic) production batch jobs. As such, the herbal goal would be to reduce the duration of time required for a batch window.

The MapReduce version can be used for predicting the finishing touch time of the Map and Reduce levels as a feature of the enter dataset size and allocated sources. Here, we take into account Map and Reduce two levels wherein Map level consists of time for activity set-up, splitting, mapping to supply (key, price) pairs and Reduce level includes time for combining, sorting, shuffle and producing final outputs [13], [15].

Originally, Hadoop hired an easy FIFO scheduling coverage [6], [7], [8], [1], [11] designed with an aim of minimizing the makespan of massive mechanically finished batch workloads. However job management the usage of this policy could be very rigid: once long, manufacturing jobs are scheduled within the MapReduce cluster the later submitted brief, interactive adhoc queries have to wait until the earlier jobs end, which could make their results much less relevant. The Hadoop Fair Scheduler (HFS) solves this hassle by enforcing a few fairness on few of the jobs and making sure that every job at least gets a predefined minimum of allocated slots. While this technique lets in sharing the cluster amongst multiple customers and their programs, HFS does no longer offer any guide or manage of allotted assets with a purpose to achieve the utility overall performance desires and storage level targets (SLOs). In MapReduce environments, many manufacturing jobs are run periodically on new information. For example, Facebook, Yahoo!, and eBay system terabytes of statistics and event logs in step with day on their Hadoop clusters for unsolicited mail detection, business intelligence and different sorts of optimization.

1.2 Job Execution MapReduce

MapReduce jobs are dispensed and done over more than one machines: the map stage is partitioned into map tasks and the reduce tasks is partitioned into reduce jobs [1].

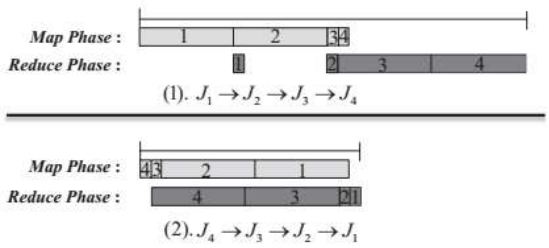

Fig1. MapReduce Execution flow under different job orders

Each map task processes a logical split of the input information that generally resides on a distributed file system. The map task applies the person-defined map feature on every record and buffers the resulting output. This intermediate statistics is hash-partitioned for the exceptional reduce tasks & written to the local hard disk of the worker executing the map task. The reduce stage consists of 3 phases: shuffle, sort and reduce phase. In the shuffle phase, the map responsibilities fetch the intermediate data documents from map tasks, thus following the "pull" model [3]. In the sort phase, the intermediate documents from all of the map tasks are taken care of. An outside merge type is utilized in case the intermediate records do no longer match in reminiscence. After all the intermediate facts are shuffled, a final skip is made to merge a lot of these looked after files. Thus, the shuffle and type stages are interleaved. Finally, within the map phase, the sorted intermediate facts (within the form of a key and all its corresponding values) is exceeded to the consumer-defined reduce characteristic. The output from the map characteristic is usually written again to the dispensed record machine [1]. In fig1, the nonoverlap computation constraint between map and reduce tasks of a MapReduce job, ensuing in special aid utilizations for map/lessen slots below exclusive job submission orders for batch jobs.

Job scheduling in Hadoop is performed with the aid of the task master, which manages some of employee nodes inside the cluster. Each employee has a set wide variety of map and reduce slots, that can run responsibilities. The variety of map and reduce slots is statically configured (commonly to 1 or in keeping with middle). The people periodically ship heartbeats to the master to report the wide variety of free slots and the development of the responsibilities that they're currently running. Based at the availability of unfastened slots and the rules of the scheduling coverage, the grasp assigns map and reduce tasks to slots inside the cluster [10], [1], [12].

2. RELATED WORK

While MapReduce has demonstrated a prevalent execution display for expansive cluster analysis, as of late, numerous associations have begun to share their MapReduce (Hadoop) groups among various clients, which run a blend of bunch and short intuitive employments [3]. To empower this utilization display, Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker and Ion Stoica have proposed FAIR [2], a reasonable scheduler that gives disengagement, ensures a base offer to every client (work), and accomplishes measurable multiplexing. Amid its underlying arrangement, we have distinguished two parts of MapReduce-information: region and errand relationship-which extensively hurt FAIR's throughput. To address this issue they have created two basic yet hearty procedures: postpone booking and duplicate process part. Utilizing a wide cluster of examinations they have demonstrated that FAIR accomplishes disconnection, low reaction time, and high throughput.

Abhishek Verma, Ludmila Cherkasova and Roy H. Campbell [3], considered the issue of finding a calendar that limits the general culmination time of a given arrangement of autonomous MapReduce occupation. They composed a novel system and another heuristic, called Balanced Pools that effectively use attributes and properties of MapReduce occupations in a given workload for developing the streamlined activity plan. As of now, they are assessing this heuristic with a wide range of MapReduce workloads to quantify achievable execution picks up. Information investigation errands are frequently determined with more elevated amount SQL-type reflections like Pig and Hive [5], which may bring about MapReduce employments with conditions.

Parag Agrawal, Daniel Kifer and Christopher Olston [4], [9] examined how to plan occupations that can share look over a typical arrangement of info documents. The objective is to amortize costly document checks crosswise over numerous occupations, however without excessively harming singular employment reaction times. Their approach constructs a straightforward stochastic model of employment entries for each info document, and considers foreseen future occupations while planning employments that are right now enqueued. The main idea of authors is: if an enqueued job J requires checking a big size file F, and forecast the close term access of extra jobs that likewise check F, at that point it might made sense to delay J if job J has not already in queue too long and other, less sharable, occupations are accessible to run. They formalized the issue and determined a basic and compelling booking approach, under the target of limiting apparent hold up time (PWT) for culmination of client occupations.

3. FRAMEWORK

3.1 Overview of the Proposed System

The main aim of this paper is to improve the performance of the MapReduce workload by minimizing the makespan as

well total completion time. Here, the makespan and total completion time both are the performance metric for the MapReduce workloads.

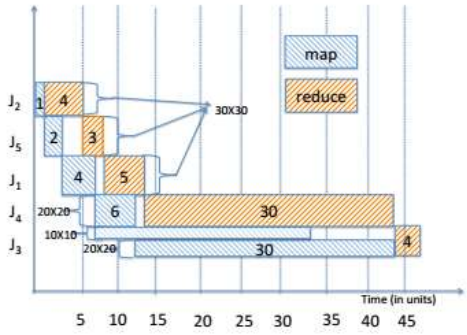

Fig2. MapReduce jobs execution in a cluster by Makespan_ Johnson Rule (MK_JR)

These two metrics are performed based on Johnson Rule (JR) as shown in figure 2.

Makespan:

The definition of makespan is that the time period for the first process until the completion of the closing activity for a hard and fast of jobs. It considers the computation time of jobs and is often used to degree the overall performance and utilization performance of a system.

Total Completion Time:

The general final touch time is called the sum of finished time durations for all jobs for the reason that begin of the first job. It is a generalized makespan with queuing time (i.e., waiting time) protected. We can use it to degree the delight to the gadget from a unmarried process's perspective thru dividing the full final touch time by the number of jobs (i.e., average final time).

To optimize the makespan and total completion time, we proposed two algorithms such as;

1. Optimized Job Ordering algorithm
2. Optimized Job Ordering and Slot Configuration algorithm

3.2 Optimized Job Ordering Algorithm (MK_SF_JR)

Inputs:

MapReduce workloads (jobs), Map slots, reduce slots

Output:

Optimized job submission order

Procedure:

1. Estimate the map-phase processing time and reduce-phase processing time of each job
2. To order the jobs, partition the job set into two subsets such as, J1 and J2
3. Here, J1 = (Processing time of Map-phase < Processing time of Reduce-phase) and J2 = (Processing time of Map-phase > Processing time of Reduce-phase)
4. Order all jobs into job set, using increasing and decreasing order based on condition of job sets.
5. Make all optimized ordered job set
6. Estimate the makespan of all slots of Map and Reduce
7. Compare estimated makespan with mini-makespan
8. If (mini-makespan>makespan) then makespan = mini-makespan

Output: Optimized Makespan with submission order

Given a MapReduce workload and the total number of slots, an sensitive method is to search & evaluate all combinations of job submission orders & map/reduce slot configurations exhaustively, as shown in Algorithm. It produced the optimal execution plan for makespan optimization.

This algorithm can produce the minimized makespan with ordered jobs. To get the optimized makespan, we have to estimate the makespan with minimized makespan and after that we can compare the makespan values. If (mini-makespan>makespan) then makespan = mini-makespan.

3.3 Optimized Slot Configuration and Job Ordering Algorithm(MK_TCT_SF_JR)

Inputs:

MapReduce workloads (jobs), Map slots, reduce slots

Output:

Optimized job submission order with slot configuration

Procedure:

9. Estimate the map-phase processing time and reduce-phase processing time of each job
10. To order the jobs, partition the job set into two subsets such as, J1 and J2
11. Here, J1 = (Processing time of Map-phase < Processing time of Reduce-phase) and J2 = (Processing time of Map-phase > Processing time of Reduce-phase)

12. Order all jobs into job set using increasing and decreasing order based on condition of job sets.
13. Make all optimized ordered job set
14. Estimate the makespan of all slots of Map and Reduce
15. Estimate (Makespan, total completion time (TCT)) of map and reduce
16. Compare estimated makespan with mini-makespan
17. If (mini-makespan>makespan) then makespan = mini-makespan
18. TCT = Mini-TCT

Output: Optimized job submission order with slot configuration

The above slot configuration rule is a bi-criteria optimization algorithm for makespan and total completion time with considerion for slot configuration optimization.

This algorithm optimizes two metrics named as makespan as well as total completion time simultaneously for those workloads which contains lots of small jobs.

4. EXPERIMETAL RESULTS

We evaluate our algorithms with the average execution time for map and decrease tasks. Particularly, we validate that it is suitable for using average execution time in our algorithms by means of showing that the impact of various venture execution time is minor. For each process, we estimated its average project execution time.

Fig3. Makespan speedup under different number of jobs

For makespan (or general total time), we normalized it by the usage of makespan speedup (or total completion time speedup), defined as the ratio of makespan (or overall finishing touch time) from the unoptimized case to that from the targeted activity order. Therefore, the larger speedup shows the better overall performance for the detailed job order as shown in fig.3.

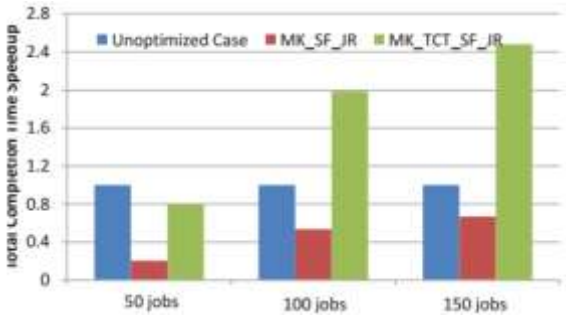

Fig4. Total completion time speedup under different number of jobs

Moreover, there is a slight drop in makespan speedup for MK_TCT_JR in comparison to MK_JR, sacrificing a bit performance improvement in makespan for a good total completion time as shown in fig4. We can observe the total completion time of the different mapreduce workloads.

5. CONCLUSION

We conclude that in this paper, we proposed job ordering optimization algorithm & MapReduce slot configuration optimization algorithm. We observed that full of completion time may be terribly difficult to get the most suitable makespan, consequently, we further suggest a new greedy process ordering set of rules and a map/lessen slot configuration set of rules to limit the makespan and total of completion time collectively.

REFERENCES

[1] Shanjiang Tang, Bu-Sung Lee, Bingsheng He, "Dynamic Job Ordering and Slot Configurations for MapReduce Workloads", DOI 10.1109/TSC.2015.2426186, IEEE Transactions on Services Computing

[2] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker and Ion Stoica, “Job Scheduling for Multi-User MapReduce Clusters” April 30, 2009

[3] Abhishek Verma, Ludmila Cherkasova and Roy H. Campbell, "Two Sides of a Coin: Optimizing the Schedule of MapReduce Jobs to Minimize Their Makespan and Improve Cluster Performance".

[4] Parag Agrawal, Daniel Kifer and Christopher Olston, “Scheduling Shared Scans of Large Data Files”, VLDB ‘08, August 24-30, 2008, Auckland, New Zealand.

[5] P. Dutot, L. Eyraud, G. Mounie, D. Trystram. Bi-criteria Algorithm for Scheduling Jobs on Cluster Platforms, SPAA, pp. 125-132, 2004

[6] C. Rajendran. Two-Stage Flowshop Scheduling Problem with Bicriteria. Journal of the Operational Research Society, vol. 43, pp. 871-884, 1992.

[7] C. Oguz, M.F. Ercan, T.C.E. Cheng, Y.F. Fung, ˘ Heuristic algorithms for multiprocessor task scheduling

in a two-stage hybrid flow-shop, European Journal of Operational Research, Vol. 149, pp. 390-403, 2003.

[8] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A Self-tuning System for Big Data Analytics. In CIDR, pages 261C272, 2011.

[9] S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves. Sailfish: a framework for large scale data processing, SoCC 2012

[10] H. Herodotou and S. Babu, Profiling, What-if Analysis, and Costbased Optimization of MapReduce Programs. in Proc. of the VLDB Endowment, Vol. 4, No. 11, 2011.

[11] K. Howard, S. Siddharth and V. Sergei. A model of computation for MapReduce, Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 938-948, 2010.

[12] P.F. Dutot, G. Mounie, and D. Trystram. Scheduling parallel tasks approximation algorithms. In J.T. Leung (ed.), Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman Hall, CRC Press, 2004.

[13] P. Sanders, J. Speck. Efficient Parallel Scheduling of Malleable Tasks. IPDPS, pp. 1156-1166, 2011.

[14] T. Condie, N. Conway, P. Alvaro, J.M. Hellerstein. MapReduce online. In Proceedings of the 7th USENIX conference on Networked systems design and implementation, pp. 21C21, 2010.

[15] S.J. Tang, B.S. Lee, and B.S. He. MROrder: Automatic Job Ordering Optimization for Online Hadoop Clusters, Euro-Par 2013.