

# MAKESPAN OPTIMIZATION USING JOB ORDERING AND SLOT CONFIGURATION OPTIMIZATION APPROACHES FOR MAPREDUCE WORKLOADS

Ch. Prashanth<sup>1</sup>, M. Praveen<sup>2</sup>, Anil. Degala<sup>3</sup>

<sup>1,2</sup>Assistant Professor, Dept of CSE, <sup>3</sup>Associate Professor, Dept of IT  
Vidya Jyothi Institute of Technology, India

**ABSTRACT:** *The MapReduce is an open source Hadoop framework applied for processing and generating distributed huge Terabyte facts on big clusters. Its principal motive is to lowering the of completion time of massive units of MapReduce jobs. Hadoop Cluster most effective has predefined constant slot configuration for cluster lifetime. This fixed slot configuration can also produce long completion time (Makespan) and system low resource utilization. The current open source Hadoop allows only static slot configuration, like fixed numbers of map slots and reduce slots throughout the cluster lifetime. Such static configuration may lead to long completion length as well as low system resource utilizations. Propose new schemes which use slot ratio between Map and Reduce tasks as a tunable knob for minimizing the completion length (i.e., makespan) of a given set. By leveraging the workload information of recently completed jobs, schemes dynamically allocates resources (or slots) to map and reduce tasks. Many scheduling methodologies are discussed that aim to improve execution performance as well as completion time goal.*

## I. INTRODUCTION

In present years, MapReduce has come to be a famous version for data-extensive computation. The schedulers are important in improving the performance of MapReduce/Hadoop in presence of multiple jobs with specific traits and overall performance goals. In a huge cluster with heterogeneous sources, maximizing a project's distribution can also result in large communication expenses. Therefore, the corresponding job's finishing touch time could be prominent. MapReduce is a processing method and a software model for dispensed computing based on java. The MapReduce algorithm contains two critical tasks, namely Map and Reduce. Map takes a hard and fast of data and converts it into some other set of data, where man or woman elements are broken down into tuples (key/value pairs). Secondly, lessen undertaking, which takes the output from a Map as an input and combines the ones information tuples right into a smaller set of tuples. As the collection of the name MapReduce implies, the reduce mission is continually carried out after the Map job. MapReduce is that it is straightforward to scale data processing over multiple computing nodes. Under the MapReduce version, the information processing primitives are called mappers and reducers. Decomposing a data processing utility into mappers and reducers is sometimes nontrivial. But, once we write

application within the MapReduce form, scaling the software to run over loads, lots, or maybe tens of heaps of machines in a cluster is simply a configuration change. This easy scalability is what has attracted many programmers to use the MapReduce model. There are two key performance metrics i.e. Makespan and total completion time (TCT). 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. In this works, we aim to optimize these two metrics. When large data sets are given to operate on them for each or every single set of jobs the makespan and total completion time should be able to minimize and optimize the job reduction issues using MapReduce algorithm. This including the total completion time for all datasets should be optimized. To improve the workload distribution among the multiple machines MapReduce Framework is used. To avoid more memory consumption using efficient technique, MapReduce can be used. The importance of MapReduce is escalating step by step as a parallel programming model for large scale data preparing. At the same time in the long run, we find out some customary MapReduce stages which have a poor execution in content of group asset use following the conventional multi-stage parallel model and some current time table approaches utilized as a part of the group environment have a few disadvantages. We address these issues through our involvement in planning a Dynamic Split Model of the assets usage which contains two advances, Dynamic Resource Allocation considering the stage need and employment prerequisite when distributing assets and Resource Usage Pipeline which can relegate errands alertly. The fundamental pipe line of parallel processing catches the scholastic world consideration. Further area/circle is an appropriated figuring stage which is like Google GFS/MapReduce. It comprises of a parallel runtime Sphere and in addition a conveyed file framework Sector. Another parallel runtime is phaser which is a direction build for element parallelism under the environment of multi-processors rather than multi-nodes.

## II. RELATED WORK

P.F. Dutot, G.Mounie, and D. Trystram offered an attractive version for scheduling effectively programs on parallel and disbursed systems primarily based on PTs. It is a pleasant

opportunity to conventional computational fashions particularly for huge conversation delays and new hierarchical structures. We have proven how to reap properly approximation scheduling algorithms for the distinct styles of PTs for two standards for both off-line and on-line instances. J. N. D. Gupta described that the two-level flowshop problem while there are same a couple of machines at each level, and shows that the problem is NP-complete. An efficient heuristic algorithm is developed for finding an approximate answer of a unique case whilst there may be most effective one machine at stage2.

Anil Sagar T, Ramakrishna V Moni proposed a DynamicMR Technique can be used to beautify the execution of MapReduce workloads even as retaining up the fairness. It incorporates of 3 strategies, mainly, DHSA, SEPB, Slot PreScheduling, all of which focus on the slot use optimization for MapReduce group from trade points of view. DHSA concentrates on the slot use enlargement by way of dispensing map or reduce slots to map and reduce obligations alterably. Especially, it doesn't have any presumption or require any earlier studying and can be applied for any kinds of MapReduce jobs (e.G., self reliant or subordinate ones). Two varieties of DHSA are brought, particularly, PI-DHSA and PD-DHSA, in view of exceptional levels of fairness. Client can choose each of them likewise. Slot Pre-Scheduling complements the talent of slot use through using growing its statistics locality.

Chen He, Ying Lu and David Swanson developed a new matchmaking algorithm to improve the data locality rate and the average response time of MapReduce clusters. We have carried out experiments to compare not only MapReduce scheduling algorithms with and without our matchmaking algorithm but also with an existing data locality enhancement technique.

Dawei Jiang, Beng Chin Ooi, Lei Shi and Sai Wu conducted an in-depth performance study of MapReduce in its open source implementation, Hadoop. We have identified five factors that affect the performance of MapReduce and investigated alternative implementation strategies for each factor.

### III. FRAMEWORK

#### A. Proposed System Overview

We goal at one subset of manufacturing MapReduce workloads that consist of a set of unbiased jobs (e.g., every of jobs approaches distinct information units and not using a dependency among each different) with one-of-a-kind procedures. For dependent jobs (i.e., MapReduce workflow), one MapReduce can most effective begin simplest whilst its preceding structured jobs end the computation concern to the input-output statistics dependency. In comparison, for independent jobs, there's an overlap computation among two jobs, i.e., while the current job completes its map-section computation and starts its map-phase computation, the subsequent task can begin to perform its map-segment computation in a pipeline processing mode by means of possessing the launched map slots from its preceding activity. To maximize the slot utilization for MapReduce and balance the performance exchange between a single job and a batch

of jobs with fair scheduling and improving the performance of MapReduce cluster in Hadoop. Goals and Objective the objective is to utilize the slots in MapReduce cluster.

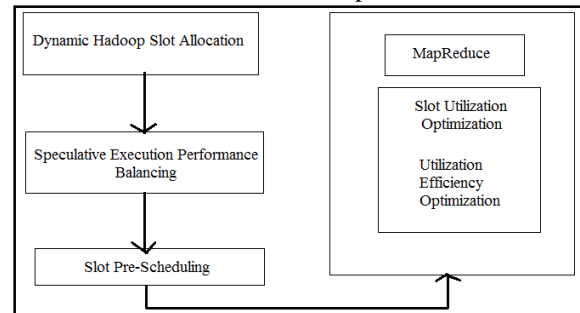


Fig1. Overview of Framework

The slot utilization remains a challenging task because of fairness and resource needs. It is truthful once all pools are allotted with a similar quantity of resources. The resources needs between the map slot and reduce slot are typically different. This is as a result of the map task and reduce task are often exhibit completely different execution patterns. We tend to review job ordering optimization. To model performance of system, make span and total completion time is used. Total time taken to complete job is calculated. We tend to describe the dynamic slot allocation framework that produces the optimized job order and additionally prove its approximation ratio. We tend to additionally describe the job order which provides the worst, i.e., longest make-span, which is used for derivation of the boundary make-span of a workload. We tend to propose an alternative technique known as dynamic Hadoop slot allocation by keeping the slot based model. It relaxes the slot allotment constriction to allow slots to be reallocated to either map or reduce tasks depending on their needs. Second, the speculative execution will tackle the straggler problem that is shown to boost the performance for single job however at the expense of the clustering. Within the view, we tend to propose speculative execution performance balancing to balance performance trade-off between single job and a batch of jobs. Third, delay scheduling has shown to enhance the data vicinity however at the cost of fairness.

#### B. Need for Makespan Optimization

In this paper, Makespan and Total Completion Time (TCT) are two major performance metrics to the MapReduce workloads. Generally, makespan is defined because the term for the reason that begin of the first task until the entirety of the final task 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 efficiency of a machine. In assessment, general of completion time is called the sum of finished time periods for all jobs for the reason that begin of the primary job. It is a generalized makespan with queuing time covered. We can use it to measure the satisfaction to the device from a single task's perspective thru dividing the entire of entirety time by using the quantity of jobs. In this paper, we describe the MK\_JR set of rules that produces the optimized task order and also prove its approximation ratio. We also describe the process order which offers the worst,

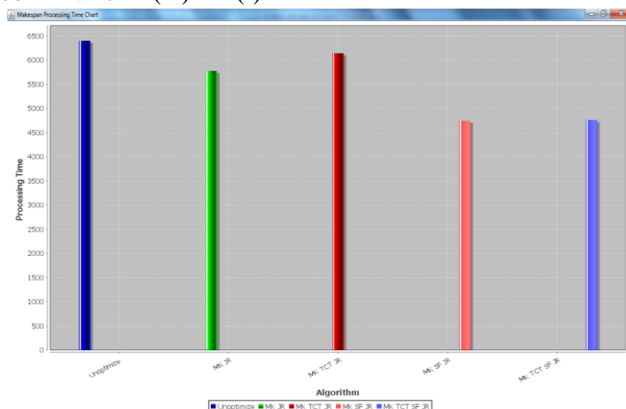
i.e., longest makespan, that's used for derivation of the higher sure makespan of a workload. Next, we describe the MK\_TCT\_JR algorithm, which optimizes each makespan and overall of completion time.

#### IV. EXPERIMENTAL RESULTS

In this experiment, we need run unoptimized (by using normal map reducer concept). Here we are running 3 types of jobs such as word count, sorting and creating inverted index. After run these three jobs, we can run the MK\_JR algorithm. As per this algorithm, we order jobs in J based on the following principles: Partition jobs set J into two disjoint subsets JobA and JobB:

JobA = when  $T(m) \leq T(r)$

JobB = when  $T(m) > T(r)$



MK\_TCT\_JR algorithm is also similar to MK\_JR algorithm but difference is based on time threshold value it arrange the jobs. After completion of these two algorithms, we can run the MK\_SF\_JR algorithm. This algorithm shows that how many processes required for job and MK\_TCT\_SF\_JR algorithm provide the processes information based on time threshold. Finally, we can see the makespan time for all algorithms.

#### V. CONCLUSION

In this paper we worked on the job ordering and map/reduce slot configuration problems for MapReduce production workloads, wherever the typical execution time of map/reduce tasks for a MapReduce job can be profiled from the history run. Two performance metrics are considered, i.e., Makespan and total completion time. We tend to initial specialize in the Makespan. We tend to propose job ordering optimization algorithm and map/reduce slot configuration optimization algorithm. We tend to observe that the entire completion time are often poor subject to obtaining the optimal Makespan, therefore, we tend to additional propose a brand 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 additionally given for our projected heuristic algorithms, as well as approximation ratio, higher and lower bounds on Makespan. Finally, we tend to conduct extensive experiments to validate the effectiveness of our projected algorithms and their theoretical results.

#### REFERENCES

- [1] S. R. Hejazi and S. Saghafian, "Flowshop-scheduling problems with makespan criterion: A review," *Int. J. Production Res.*, vol. 43, no. 14, pp. 2895–2929, 2005.
- [2] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou, "Re-optimizing data-parallel computing," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implementation*, 2012, p. 21.
- [3] P. Agrawal, D. Kifer, and C. Olston, "Scheduling shared scans of large data files," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 958–969, Aug. 2008.
- [4] W. Cirne and F. Berman, "When the herd is smart: Aggregate behavior in the selection of job request," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 2, pp. 181–192, Feb. 2003.
- [5] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implementation*, 2010, p. 21.
- [6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Oper. Syst. Design Implementation*, 2004, vol. 6, p. 10.
- [7] J. Dittrich, J.-A.-Quiane Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "adoop++: Making a yellow elephant run like a cheetah (without it even noticing)," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 515–529, Sep. 2010.
- [8] P.-F. Dutot, L. Eyraud, G. Mounie, and D. Trystram, "Bi-criteria algorithm for scheduling jobs on cluster platforms," in *Proc. 16th Annu. ACM Symp. Parallelism Algorithms Archit.*, 2004, pp. 125–132.
- [9] P.-F. Dutot, G. Mounie, and D. Trystram, "Scheduling parallel tasks: Approximation algorithms," in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, J. T. Leung, Ed. Boca Raton, FL, USA: CRC Press, ch. 26, pp. 26-1–26-24.
- [10] J. Gupta, A. Hariri, and C. Potts, "Scheduling a two-stage hybrid flow shop with parallel machines at the first stage," *Ann. Oper. Res.*, vol. 69, pp. 171–191, 1997.