Dynamic Job Ordering and Slot Configuration for MapReduce Workloads

Sonali S. Birajadar
M B E Society's College of Engineering, Ambajogai
Maharashtra, India

B. M. Patil
M B E Society's College of Engineering, Ambajogai
Maharashtra, India

V. M. Chandode
M B E Society's College of Engineering, Ambajogai
Maharashtra, India

ABSTRACT
In today's world the amount of data being generated is growing exponentially and use of internet is also increasing it leads to handle lots of data by internet service providers. MapReduce is one of the good solutions for implementing large scale distributed data application. A MapReduce workload generally contains a set of jobs, each of job consists of multiple map and reduce tasks. Map task executed before reduce task and map tasks can only run in map slot and reduce tasks can only run in reduce slot. Due to that different job executions orders and map/reduce slot configurations for a MapReduce workload have different performance metrics and different system utilization. Makespan and total completion time are two key performance metrics. This paper proposes two algorithm for these two key metrics, The first class of algorithms mainly focuses on the job ordering optimization for a MapReduce workload under given slot configuration and the second class of algorithms perform optimization for slot configuration for a MapReduce workload.

Keywords
MapReduce, Hadoop, Flow-shops, Scheduling algorithm, Job ordering.

1. INTRODUCTION
MapReduce is a processing method and a software model for dispensed computing based on java. Hadoop, an open source implementation of MapReduce, has been deployed in large clusters containing thousands of machines by companies such as Amazon and Facebook. The MapReduce algorithm contains two critical tasks, namely Map and Reduce, where the reduce tasks are performed after the map tasks. Map takes a hard and fast of data and converts it into some other set of data, where 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. 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) and we aim to optimize these matrics. Generally, make span is defined as the timeperiod 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 to the system from a single job's perspective through dividing the total completion time by the number of jobs (i.e., average completion time). Therefore, in this paper, we aim to optimize these two metrics the number of jobs (i.e., average completion time). Therefore, in this paper, we aim to optimize these two metrics.

Objectives:-

- To improve the performance for MapReduce workloads with job ordering and slot configuration optimization approaches.
- Propose slot configuration algorithms for make span and total completion time.
- Perform extensive experiments to validate the effectiveness of proposed algorithms and theoretical results.

2. LITERATURE REVIEW
Wolf et al. [2] implemented flexible scheduling allocation scheme with Hadoop fair scheduler. A primary concern is to optimize scheduling theory metrics, response time, makespan, stretch, and Service Level Agreement. They proposed penalty function for measurement of job completion time, epoch scheduling for partitioning time, moldable scheduling for job parallelization, and malleable scheduling for different interval parallelization.

Dean et al. 2008 [1] have discussed MapReduce programming model. The MapReduce model performs operations using the map and reduces functions. Map function gets input from user documents. It generates intermediate key/value for reducing function. It further processes intermediate key/value pairs and provide output key/value pairs. At an entry level, MapReduce programming model provided the best data processing results. Currently, it needs to process the large volume of data. So it provides some consequences while processing and generating data sets. It takes much execution time for task initialization, task coordination, and task scheduling. Parallel data processing may lead to inefficient task execution and low resource utilization.

Verma et al. [3] proposed two algorithms for makespan optimization. First is a greedy algorithm job ordering method based on Johnson's Rule. Another is a heuristic algorithm called BalancedPool. They have introduced a simple abstraction where each MapReduce job is represented as a pair of map and reduce stage duration. The Johnson algorithm was designed for building an optimal job schedule. This framework evaluates the performance benefits of the constructed schedule through an extensive set of simulations over a variety of realistic workloads. It measures how many numbers of slots required for scheduling the slots dynamically with a particular job deadline.

Tang et al. [4] have proposed three techniques to improve MapReduce performance. First technique is Dynamic Hadoop Slot Allocation. They categorized utilized slot into the busy slot and idle slot respectively. The primary concern is to increase the number of the busy slots and decrease number of idle slots. DHSA observes idle map and reduce slots. Dynamic Hadoop Slot Allocation allocate the task only to the unallocated map slots and due to Speculative Execution Performance Balancing provides performance upgrade for a batch of jobs. It gives the highest priority to failed tasks and next level priority to pending tasks. Due to slot prescheduling it improves the performance of slot utilization.

Tang, Lee and He [5] have proposed DynamicMR: A Dynamic Slot Allocation Optimization Framework for improving the performance for a single job but at the expense of the cluster efficiency. They proposed Hazardous Execution Performance Balancing technique for balancing the performance tradeoff between a single job and a batch of jobs. Slot PreScheduling is the new technique and that can improve the data locality but with no impact on fairness. Finally, integrating these two techniques, new technique is implemented called DynamicMR that can improve the performance of MapReduce workloads.

Tang, Lee and He [6] have proposed MROrder: Flexible Job Ordering technique which optimizes the job order for online MapReduce workloads. MROrder is designed to be flexible for different optimization metrics, e.g., makespan and total completion time. Kyparisis and Koulamas [7] considered a scheduling problem in two-stage hybrid flow shop, where the first stage consists of two machines formed an open shop and the other stage has only one machine. The main objective is to minimize the makespan, i.e., the maximum completion time of all jobs. They first show the problem is NP-hard in the strong sense, then we present two heuristics to solve the problem. Computational experiments show that the combined algorithm of the two heuristics performs well on randomly generated problem instances.

Agrawal et al. [8] have proposed a method called Scheduling shared scans of large data files and it is used to maximize scan sharing by grouping MapReduce jobs into batches so that sequential scans of large files are shared among many simultaneous jobs where it is possible. MRShare [9] is a sharing framework and it gives three possible work-sharing opportunities, they are scan sharing, mapped outputs sharing, and Map function sharing across multiple MapReduce jobs. Due to sharing it avoides the redundant work and saves the processing time.

Herodotou et al. [10] provide Hadoop configuration optimization policy. Starfish is a self-tuning framework and it can adjust the Hadoop's configuration automatically for a MapReduce job. Based on the cost-based model and sampling technique the utilization of Hadoop cluster can be maximized and it also proposes 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.

3. JOB ORDERING OPTIMIZATION

We first focus on makespan optimization. We describe the MK_JR algorithm that produces the optimized job order. Next, we describe the MK_TCT_JR algorithm, which optimizes both makespan and total completion time.

3.1 Makespan Optimization

The optimal job order for the simplified case can be obtained by using Johnson's Rule [11], which is an efficient job ordering algorithm for the minimum makespan. Johnson's rule works as follows. Divide the jobs set J into two disjoint sub-sets J_A and J_B. Set J_A consists of those jobs J_i for which $T_i^M < T_i^R$. Set J_B contains the remaining jobs (i.e. J \ J_A). Sequence jobs in J_A in non-decreasing order of T_i^M and those in J_B in non-increasing order of T_i^R. The optimal job order is obtained by appending the sorted set J_B to the end of sorted set J_A. In this sometimes the makespan minimization problem becomes NP-hard, because the number of tasks is not divisible by the number of slots. Verma et al. [3] first noted it and proposed an algorithm based on Johnson's rule. and re-formated in the following algorithm MK_JR.

Algorithm 1. Greedy Algorithm Based on Johnson's Rule(MK_JR)

Input:

J: the MapReduce workload

$|S^M|$: The given number of map slots.

$|S^R|$: The given number of reduce slots.

Output:

Φ : The optimized job submission order.

1. For each job J_i , we first estimate its map-phase processing time T_i^M and reduce-phase processing time T_i^R by using the following formula:

$$\left(T_i^M, T_i^R\right) = \left(\frac{\sum_{j=1}^{|J_i^M|} t_{i,j}^M}{|S^M|}, \frac{\sum_{j=1}^{|J_i^R|} t_{i,j}^R}{|S^R|} . t_i^R\right)$$

2. We order jobs in J based on the following principles:

a) Partition jobs set J into two disjoint sub-sets J_A and J_B:

$$J_A = \{J_i | (J_i \varepsilon J) \wedge (T_i^M \leq T_i^R)\}$$
$$J_B = \{J_i | (J_i \varepsilon J) \wedge (T_i^M > T_i^R)\}$$

b) Order all jobs in J_A from left to right by non-decreasing T_i^M. Order all jobs in J_B from left to right by nonincreasing T_i^R.

c) Make an ordered jobs set J' by joining all jobs in J_A first and then J_B in order, i.e.,

$$\emptyset_1 : J' = \{(J_A), (J_B)\}$$

3.2 Bi-Criteria Optimization of Makespan and Total Completion Time

In this we consider two key performance metrics i.e. Makespan and total completion time. Generally, makespan is nothing but maximum completion time for a batch of jobs. It considers the computation time of jobs and is often used to measure the performance and utilization efficiency of a system. Total completion time is the sum of completion time of all jobs. It is a generalized makespan with queuing time i.e. waiting time included. So far, we focus only on the optimization of makespan. Here the total completion time that can be poor subject to gain optimal makespan. Therefore, there is a need for bi-criteria optimization for both key performance matrics. Intuitively, the makespan is affected primarily by the positions of large-size jobs. In contrast, the total completion time is mainly influenced by the positions of small-size jobs. The algorithm shortest processing time first (SPTF) is used to optimize the total completion time. However, MK_JR is not aware of varying job sizes. In some scenarios the job order produced by MK_JR can have adverse effect on the total completion time. For example, there can be

a job J_i whose processing time (e.g. $T_i^M + T_i^R$) is very small but $T_i^M > T_i^R$. We should schedule J_i early if we want to minimize the total completion time, whereas MK_JR might put it in the middle or later part of the order list according to Johnson's Rule. Therefore we design a new greedy algorithm by combining SPTF and Johnson's Rule called MK_TCT_JR. In MK_TCT_JR, we first divide job set J into two subsets, $J_A^{'}$ and $J_B^{'}$. Let $J_A^{'}$ contain small-size jobs and $J_B^{'}$ contain large-size jobs. We schedule jobs in $J_A^{'}$ first and then $J_B^{'}$. Within each set, we use MK_JR to minimize its makespan. We estimate the processing time for each job by adding its map and reduce phase running times, given the whole map reduce slots of the Hadoop cluster. Particularly, our classification of small-/large-size jobs is based on the geometric mean of processing time of all jobs, considering that unlike the arithmetic mean that favors large-size jobs, geometric mean has a good unbiased property for all jobs.

Algorithm 2. Greedy algorithm based on Shortest Processing Time First and Johnson's Rule (MK_TCT_JR)

Input:

J: the MapReduce workload

$|S^M|$: The given number of map slots

$|S^R|$: The given number of reduce slots

Output:

ϕ : the optimized job submission order.

1. For each job J_i we first compute its processing time T_i by using the formula below:

$$T_i = \frac{\sum_{j=1}^{|J_i^M|} t_{i,j}^M}{|S^M|} + \frac{\sum_{j=1}^{|J_i^R|} t_{i,j}^R}{|S^R|}$$

2. Let $T = (\prod_{1 \le i \le n} T_i)^{\frac{1}{n}}$ We divide jobs set J into two disjointsub-sets $J_A^{'}$ and $J_B^{'}$:

$$J_A^{'} = \{J_i | (J_i \in J) \cap (T_i \le T)\},$$
$$J_B^{'} = \{J_i | (J_i \in J) \cap (T_i > T)\}$$

3. Order all jobs in $J_A^{'}$ and $J_B^{'}$ using MK_JR respectively.
4. Make a ordered jobs set $J^{'}$ by joining all jobs in the ordered set $J_A^{'}$ first and then the ordered set $J_B^{'}$, i.e.,

$$\emptyset_2 : J^{'} = \left\{\{J_A^{'}\}, \{J_B^{'}\}\right\}$$

4. SLOT CONFIGURATION OPTIMIZATION

In this section, we first propose mapreduce slot configuration algorithm MK_SF_JR to optimize makespan. Then, the bi-criteria algorithm MK_TCT_SF_JR is described to optimize the makespan and total completion time together.

4.1 Makespan Optimization

Given a MapReduce workload and the total number of slots, and we have to search and compare all combinations of job submission orders and map/reduce slot configurations as shown in Algorithm 3. It can optimize the makespan and in this we can include efficient job ordering optimization algorithms i.e. MK_JR.

Algorithm 3. Search algorithm for optimized slot configuration and job submission order. (MK_SF_JR)

Input:

J: The MapReduce workload

$|S|$: The total number of slots.

Output:

ϕ: the optimized job submission order.

$|S^M|$: The optimized number of map slots.

$|S^R|$: The optimized number of reduce slots.

Mini_Makespan: the minimized makespan.

```
Mini_Makespan ← ∞,ϕ ←null.
for |S_0^M| from 1 to |S| − 1 do
    |S_0^R| ← |S| − |S_0^M|
    ∅_0 ← MK_JR(J, |S_0^M|,|S_0^R|)
    Makespan ← MREstimator (J , ∅_0, |S_0^M| , |S_0^R|).
    if Mini_Makespan > Makespan then
      Mini_Makespan← Makespan.
      (|S^M|, |S^R|) ← (|S_0^M|, |S_0^R|)
      ∅ ← ∅_0
    end if
end for
return (ϕ,|S^M|,|S^R| , Mini_Makespan).
```

4.2 Bi-Criteria Optimization of Makespan and Total Completion Time

Algorithm 4, it is a bi-criteria optimization algorithm for makespan and total completion time with regard to slot configuration optimization. It is a search algorithm that incorporates the bi-criteria job ordering algorithm MK_TCT_JR.

Algorithm 4. Search algorithm for optimized slot configuration and job submission order. (MK_TCT_SF_JR)

Input:

J: The MapReduce workload

$|S|$: The total number of slots.

Output:

ϕ: The optimized job submission order.

$|S^M|$: The optimized number of map slots.

$|S^R|$: The optimized number of reduce slots.

Mini_Makespan: the minimized makespan.

Mini_TCT: the optimized total completion time.

```
Mini_Makespan ← ∞,ϕ ←null
for |S_0^M| from 1 to |S| − 1 do
    |S_0^R| ← |S| − |S_0^M|
    ∅_0 ← MK_TCT_JR(J, |S_0^M|,|S_0^R|)
    (Makespan,TCT)←MREstimator (J, ∅_0 , |S_0^M| , |S_0^R|).
    if Mini_Makespan > Makespan then
      Mini_Makespan← Makespan.
      Mini_TCT ←TCT.
      (|S^M|, |S^R|) ← (|S_0^M|, |S_0^R|)
      ∅ ← ∅_0
    end if
end for
return (ϕ , |S^M| , |S^R| , Mini_Makespan, Mini_TCT).
```

5. EVALUATION

In this section, we evaluate algorithms using workloads. To well reflect practical workloads, we generate our workloads by choosing three benchmarks and using their provided datasets. The detailed benchmarks are described as follows:

- Word Count -Computes the occurrence frequency of each word in a document.
- Sort -Sorts the data in the input files in a dictionary order.
- Inverted Index. - Takes a list of documents as input and generates word-to-document indexing.

In our experiments, we are taking the three types of jobs like word count, sorting and creating inverted index after that run the UnOptimized means run the normal map reduce concept it shows the Job ID like serial number Job Name like Word Count, Sorting and Creating Inverted Index, Processing Type like Un Optimized, Processing Time, Mapper Time and Reducer Time the total time taken by Mapper and reducer to process the job is represented as processing time in miliseconds and their individual timings in nanoseconds after that apply the MK_JR, MK_TCT_JR, MK_SF_JR and MK_TCT_SF_JR algorithms based on that we are minimize the slot utilization for Multiple MapReduce Jobs through Job Ordering Technique. In the below chart we can observe that difference between the lengths of UnOptimized, MK_JR, MK_TCT_JR, MK_SF_JR and MK_TCT_SF_JR Algorithms.

We can observe that MakeSpan Processing Time chart in that difference between the lengths of UnOptimized, MK_JR, MK_TCT_JR, MK_SF_JR and MK_TCT_SF_JR Algorithms. The difference will be shown in the sense of Makespan Processing Time (as shown in Figure 1) and Total completion time (as shown in Figure 2). Through our implementation we can improve the performance of the system at lower cost then compare to current methods as well as minimize the Makespan and the total completion time and job ordering optimization for a MapReduce workload under a given map/reduce slot configuration through job ordering technique.

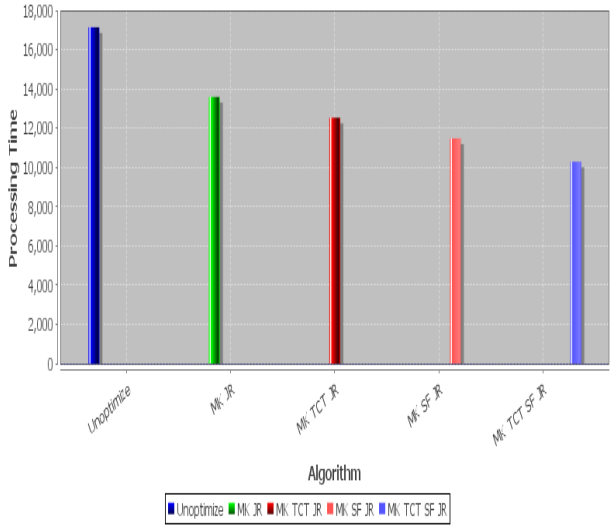

Fig 1: Makespan processing time chart

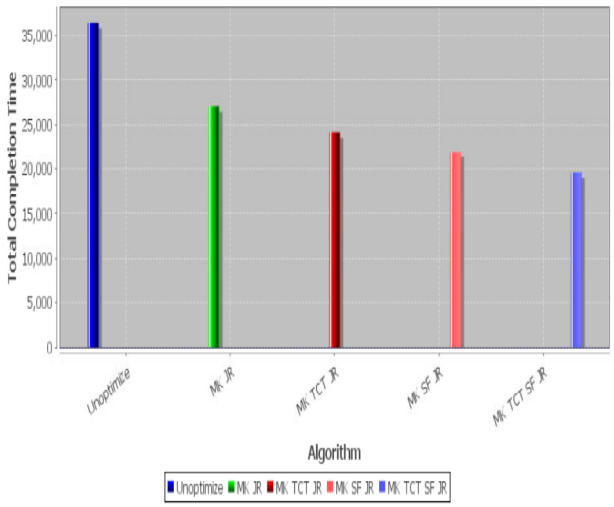

Fig 2: TCT processing time chart

6. CONCLUSION

In this paper we worked on the job ordering and map/reduce slot configuration problems for MapReduce production workloads that run periodically in a data warehouse, 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 first focus on the makespan. We tend to propose two types of algorithms i.e. 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 optimize 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 proposed algorithms and their theoretical results. In future we can prefer a dynamic slot allocation strategy that includes active jobs workload estimation, optimal slot assignment, and scheduling policy. As for future work, we are interested in extending our dynamic slot allocation algorithms to environments like Cluster/cloud has become heterogeneous with different architecture and extend our previous study to handle the slot configuration on CPUs and GPUs.

7. REFERENCES

[1] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” in Proc. 6th Conf. Symp. Oper. Syst. Design Implementation, 2004

[2] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. balmin, “Flex: A slot allocation scheduling optimizer for mapreduce workloads,” in Proc. ACM/IFIP/USENIX 11th Int. Conf. Middleware, 2010

[3] A. Verma, L. Cherkasova, and R. H. Campbell, “Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance,” in Proc. IEEE 20th Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst.,2012

[4] S. Tang, B.-S. Lee, and B. He, "Dynamic slot allocation technique for mapreduce clusters," in Proc. IEEE Int. Conf. Cluster Comput.,Sep. 2013, pp. 1–8.

[5] S. Tang, B.-S. Lee, and B. He, "Dynamicmr: A dynamic slot allocation optimization framework for mapreduce clusters," IEEE Trans.Cloud Comput., vol. 2, no. 3, pp. 333–347, Jul. 2014.

[6] S. Tang, B.-S. Lee, and B. He,, "Mrorder: Flexible job ordering optimization for online mapreduce workloads," in Proc. 19th Int. Conf. Parallel Process., 2013, pp. 291–304.

[7] G. J. Kyparisis and C. Koulamas, "A note on makespan minimization in two-stage flexible flow shops with uniform machines," Eur. J.Oper. Res., vol. 175, no. 2, pp. 1321–1327, 2006.

[8] 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.

[9] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas, "Mrshare: Sharing across multiple queries in mapreduce," Proc.VLDB Endowment, vol. 3, nos. 1/2, pp. 494–505, Sep. 2010.

[10] 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 Proc. 5th Conf. Innovative Data Syst. Res., 2011

[11] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," Naval Res. Logistics Quart., vol. 1, no. 1, pp. 61–68, 1954.