ISSN (online): 2250-141X
Vol. 6 Issue 1, February 2016

DYNAMIC MR: A DYNAMIC SLOT ALLOCATION OPTIMIZATION FRAMEWORK FOR MAPREDUCE CLUSTERS

Prof. Balaji Bodkhe[1] , Abhijeet Ahire[2], Mayur Chaudhari[3], Tushar Fulsoundar[4], Akshay Kadam[5]
[1, 2,3,4,5] Department of Computer Engineering,Mes College of Engineering,Pune,India

Abstract:- *MapReduce is a popular computing paradigm in Hadoop. MapReduce is used for large-scale data processing in Big Data. However, the slot-based MapReduce system (e.g., Hadoop MRv1) can suffer from poor performance due to its unoptimized resource allocation. To solve this problem this paper identifies and optimizes the resource allocation from three key aspects. First, due to the pre-configuration of distinct map slots and reduce slots which are not fungible, slots can be severely under-utilized. Because map slots might be fully utilized while reduce slots are empty, and vice-versa. This paper also proposes an alternative technique called Dynamic Hadoop Slot Allocation by keeping the slot-based model. It relaxes the slot allocation constraint to allow slots to be reallocated to either map or reduce tasks depending on their needs. Second, the speculative execution can tackle the straggler problem, which has shown to improve the performance for a single job but at the expense of the cluster efficiency. In view of this, we propose Speculative Execution Performance Balancing to balance the performance between a single job and a group of jobs. Third, delay scheduling has shown to improve the data locality but at the cost of fairness. Additionally, the paper propose a technique called Slot PreScheduling that can improve the data locality but with no impact on fairness. Finally, by combining these techniques together, we form a step-by-step slot allocation system called DynamicMR that can improve the performance of MapReduce workloads substantially.The abstract is to be in fully-justified italicized text as it is here, below the author information.*

Keywords: *MapReduce, Hadoop Fair Scheduler, Slot PreScheduling, Delay Scheduler, DynamicMR SlotAllocation.*

INTRODUCTION

A Government monitors many states, districts, Taluka and villages. These regions consist of different statistical data based on the residents. As this data is in large volumes, today most of that data is documented in traditional ways i.e. documented in files. This results into human errors, corruption, mishandling of data and requires infrastructure and money. Our project i.e. “Implementing Dynamic MapReduce Slot Allocation Framework”. In Hadoop for Government Managed Applications**”** aims at bringing this large amount of data on the web using Hadoop. Our application will have feature to submit document like domicile certificate required by the Government. These documents will then be used by concerning officer to verify the details of residents in future. Our main goal is to implement Dynamic Map reduce in Hadoop in order to overcome the limitations faced by the present Hadoop framework. This will help in improving the performance of our application and make effective use of system resources. Upon successful completion of our project, we will be able to show benefits of using DynamicMR in Hadoop and digitalize data of government that was so far stored traditionally. Our application will be able to overcome the

ISSN (online): 2250-141X
Vol. 6 Issue 1, February 2016

difficulties faced by the common man as well as the government. This will improve accountability and efficiency of government applications, bring transparency in government transactions, makes people aware of advantages of Computer and Internet.

In recent years, MapReduce has become a popular high performance computing paradigm for large-scale data processing in clusters and data centers .Hadoop , an open source implementation of MapReduce, has been deployed in large clusters containing thousands of machines by companies such as Yahoo and Facebook to support batch processing for large jobs submitted from multiple users (i.e., MapReduce workloads).Despite many studies in optimizing MapReduce/Hadoop, there are several key challenges for the utilization and performance improvement of a Hadoop cluster. Firstly, the compute resources (e.g., CPU cores) are abstracted into map and reduce slots, which are basic compute units and statically configured by administrator in advance.

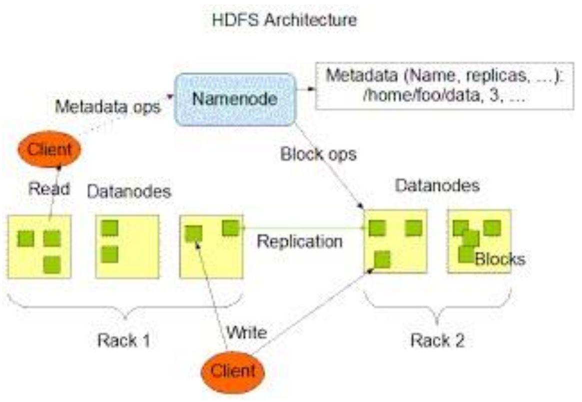
EXISTING SYSTEM

Hadoop is a storage technique used in big data storage. The data retrieval from Hadoop was previously done with MapReduce technique. The MapReduce also known as MRV1 consists of 2 main modules: The Mapper and The Reducer. The mapper is used to divide the data into standardized configured slots which are then reduced into 2 slots by the reducer. There is a collector also known as combiner which also consists of 2 techniques: Shuffling and Sorting. The shuffling algorithm integrates all the files and then sorts them accordingly to the search result.

A. Hadoop MRV1

MapReduce is a popular computing paradigm for large-scale data processing in cloud computing. However, the slot-based MapReduce system (e.g., Hadoop MRv1) can suffer from poor performance due to its unoptimized resource allocation. To address it, this paper identifies and optimizes the resource allocation from three key aspects. First, due to the pre-configuration of distinct map slots and reduce slots which are not fungible, slots can be severely under-utilized. Because map slots might be fully utilized while reduce slots are empty, and vice-versa.

Hadoop MRV1 Architecture



“Figure 1 : Architecture of MRV1”

B. Proposed system

As the MRV1 has the drawback of slot allocation we have proposed a system: DynamicMR. Due to the slot allocation problem of MRV1 the slot utilization was inefficient which ultimately affected the hardware drivers of a system and its resources. DynamicMR consists of 3 techniques: Pre-Scheduling, Post-Scheduling and Delay Time.

C. System implementation:

This system consists of following main modules that are used for building up the project.

1. MapReduce
2. Hadoop Fair Scheduler
3. Slot Pre-Scheduling

D. MapReduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.

E. Hadoop Fair Scheduler

The Fair Scheduler supports moving a running application to a different queue. This can be useful for moving an important application to a higher priority queue, or for moving an unimportant application to a lower priority queue. Apps can be moved by running yarn. When an application is moved to a queue, its existing allocations become counted with the new queueâTMs allocations instead of the old for purposes of determining fairness. An attempt to move an application to a queue will fail if the addition of the appâTMs resources to that queue would violate the its maxRunningApps or maxResources constraints.

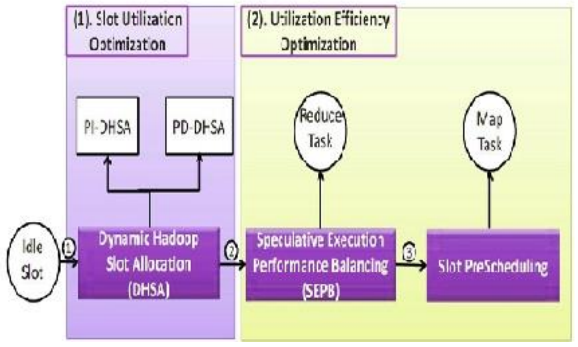
F. Slot Prescheduling

Slot Pre-Scheduling technique that holds ability to improve the data locality while having no negative impact on the fairness of MapReduce jobs. The basic level idea is that, in light of the fact that there are often some idle slots which cannot be allocated due to the load balancing constrain during runtime, we can pre-allocate those slots of the node to jobs to maximize the data locality.

G. Delay Scheduler

It delays the scheduling for a job by a small amount of time, when it detects there are no local map tasks from that job on a node where its input data reside.

SYSTEM ARCHITECTURE


(1). Slot Utilization Optimization
(2). Utilization Efficiency Optimization
PI-DHSA
PD-DHSA
Reduce Task
Map Task
Idle Slot
1
Dynamic Hadoop Slot Allocation (DHSA)
2
Speculative Execution Performance Balancing (SEPB)
3
Slot PreScheduling

“Figure 1 : Architecture of DynamicMR showing the various techniques used such as: DHSA, SPEB”

We improve the performance of a MapReduce cluster via optimizing the slot utilization primarily from two perspectives.

1. We can classify the slots into two types, namely, busy slots (i.e., with running tasks) and idle slots (i.e., no running tasks).

2. Given the total number of map and reduce slots configured by users, one optimization approach (i.e., macro-level optimization) is to improve the slot utilization by maximizing the number of busy slots and reducing the number of idle slots

ISSN (online): 2250-141X
Vol. 6 Issue 1, February 2016

A. Dynamic Hadoop Slot Allocation (DHSA)

The current configuration of MapReduce experiences an under-usage of the slots as the quantity of map and reduce tasks shifts over the long run.

Our dynamic slot allocation approach is taking into account the perception that at distinctive time there may be idle map(or reduce) slots, as the jobs continues from map stage to reduce stage. We can utilize the unused map slots for those overburden reduce tasks to enhance the execution of the MapReduce workload, and the other way around.

That is, we break the certain presumption for current MapReduce structure that the map tasks can just run on map slots and reduced tasks can just run on reduce slots.

There are two challenges specified below that must be considered:

(C1): Fairness is an imperative metric in Hadoop Fair Scheduler (HFS). We proclaim it as reasonable when all pools have been designated with the same amount of resource. In HFS, task slots are first allocated over the pools , and later then the slots are distributed to the jobs inside the pool. Also, a MapReduce job computation embodies two sections: map-phase task computation and reduce-phase task computation.

(C2): The resource requirement between the map slots and reduced slots are especially diverse. The purpose for this is the map tasks and reduced tasks regularly show totally different execution designs. Reduce task has a tendency to expend considerably more resources, for example, memory and system network speed. Basically permitting reduce tasks to utilize map slots configuring every map slots to take more resources, which will therefore lessen the powerful number of slots on every node, creating resources under-used amid runtime. With a due appreciation towards (C1), we set forth a Dynamic Hadoop Slot Allocation (DHSA). It contains two choices, to be specific,pool- free DHSA(PI-DHSA).

B. Pool Independent DHSA (PI-DHSA)

HFS utilizes max-min fairness to allocate slots crosswise over pools with least ensures at the map-phase and reduce-phase, individually. Pool-Independent DHSA (PI-DHSA) extends the HFS by dispensing slots from the clusters of worldwide level and free of pools.

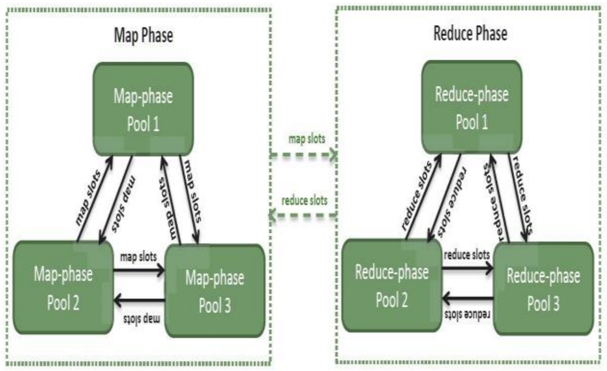
The allocation procedure is comprised of two sections:

C. Intra-phase dynamic slot allocation

Each pool is part into two sub-pools, i.e., map phase pool and reduce phase pool. At every stage, every pool will get its share of slots.

D. Inter-phase dynamic slot allocation

After the intra-phase dynamic slot allocation for both the map-phase and reduced phase, next we can perform the dynamic slot allocation crosswise over typed phase



“Figure 2 : Fairness-based slot allocation flow for PIDHSA.”

ISSN (online): 2250-141X
Vol. 6 Issue 1, February 2016

The entire dynamic slot allocation flow is that, at whatever point a pulse is gotten from a computing node, at first we process the aggregate demand for map slots and reduce slots for the current MapReduce workload. At that point we focus alertly the need to acquire map (or reduce) slots for reduce (or map) tasks in light of the interest for map and reduce slots, with respect to these four situations. The specific number of map (or reduce) slots to be obtained is based on the account of quantity of unused reduced (or map) slots and its map (or reduce) slots needed.

To accomplish the reservation usefulness, we give two variables rate Of Borrowed Map Slots and rate Of- Borrowed Reduce Slots, defined as the rate of unused map and reduced slots that can be obtained, separately. Thus, we can restrict the quantity of unused map and reduced slots that ought to be distributed for map and reduced tasks at every pulse of that task tracker. With these two parameters, clients can flexibly adjust the exchange off between the performance execution optimization and the starvation minimization.

In addition, Challenge (C2) makes us to review that we can't treat map and reduce slots as same, and just obtain unused slots for map and reduce tasks. Rather, we should be mindful of shifted resource sizes of map and reduce slots. A slot weight- based methodology is therefore proposed to address the issue. We allot the map and reduce slots with distinctive weight values, regarding the asset configurations. Particular to the weights, we can alterably decide the amount of map and reduce tasks which has to be generate in the length of runtime.

E. Pool-Dependent DHSA (PD-DHSA)

As an opposite point on checking towards PI-DHSA Pool-Dependent DHSA (PD-DHSA) considers fairness for the dynamic slot allocation across pools. Accepting that every pool, includes two sections: Map phase pool and Dynamic Phase pool, is selfish. It is considered fair when aggregate quantities of map and reduce slots allocated across pools are the same with one another. PD-DHSA will be performed with the accompanying two courses of actions:

(1). Intra-pool dynamic slot allocation

At a early stage, each typed- phase pool will receive its share of typed-slots based on max-min fairness at each phase. There are four possible relationships cases for every pool regarding its demand (denoted as mapSlots Demand, reduceSlots Demand) and its workload (marked as mapShare, reduceShare) between two phases:

Case (a). mapSlotsDemand < reduceShare, and reduceSlots-Demand > reduceShare. We can use some of the unused map slots for its overloaded reduce tasks from its reduce-phase pool first before using other pools.

Case (b). mapSlotsDemand > mapShare, and reduceSlots- Demand < reduceShare. we can use some unused reduce slots for its map tasks from its map-phase pool first before using pools.

Case (c). mapSlotsDemand < mapShare, and reduceSlots- Demand < reduceShare. Both map slots and reduce slots are enough for its use. It can give some unused map slots and reduce slots to other pools.

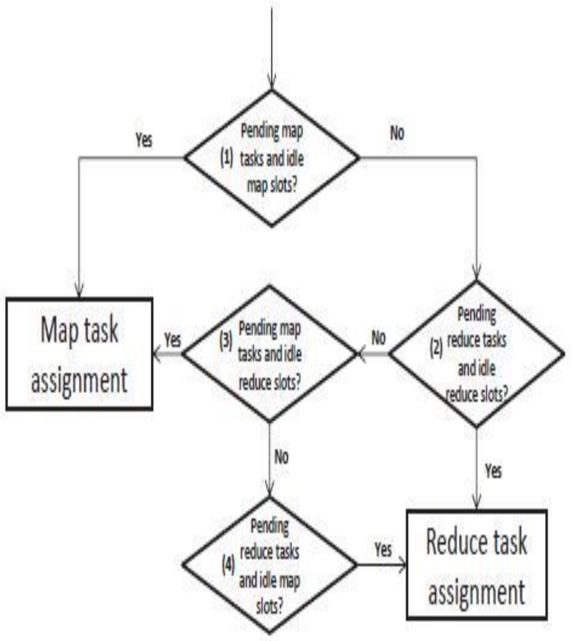
Case (d). mapSlotsDemand > mapShare, and reduceSlots- Demand > reduceShare. If both map slots and reduce slots of a pool have become insufficient. It may have to borrow some unused map or reduce slots from other pools through inter-Pool dynamic slot allocation is shown below.

(*2*). Inter-pool dynamic slot allocation

It is obvious that, (i). if a pool, has mapSlotsDemand + reduceSlotsDemand < mapShare + reduceShare. The slots are enough for the pool and there is no need to get some map or reduce slots from other pools

(ii).On the contrary, when mapSlotsDemand + reduceSlotsDemand mapShare + reduceShare, the slots are not enough even after Intra-pool dynamic slot allocation.

The overall slot allocation process for PD-DHSA is as sketched down below in figure


(1) Pending map tasks and idle map slots?
Yes
No
Map task assignment
Yes
(3) Pending map tasks and idle reduce slots?
No
(2) Pending reduce tasks and idle reduce slots?
No
Yes
(4) Pending reduce tasks and idle map slots?
Yes
Reduce task assignment

"Figure 3 : The slot allocation flow for each pool under PD-DHSA.The numbers labeled in the graph corresponds to Case (1)-(4) in Section 2.1.2, respectively."

At first, it computes the maximum number of free slots that can be allocated at each round of heartbeat for the tasktracker. Next it starts the slot allocation for pools. For every pool, there are four possible slot allocations as illustrated in Figure above.

Case(1): We try the map tasks allocation, if there are idle map slots for the task tracker, and there are pending map tasks for the pool.

Case(2): If the attempt of Case(1) fails, the condition does not hold good, and it cannot find a map task satisfying the valid data-locality level, we continue to try reduce tasks allocation when there are pending reduce tasks and idle reduce slots.

Case(3): If Case(2) fails due to the required conditions does not hold, we try for map task allocation again. If Case(1) fails then there might not have to be any idle map slots available. In contrast, if Case(2) fails then there are no pending reduce tasks. In this case, we can relay on reduce slots for map tasks of the pool.

Case(4): If Case(3) fails, we try for reduce task allocation once again. Case(1) and Case(3) fail might be because of no valid locality-level pending and map tasks available, but there are idle map slots. In contrast, Case(2) maight not have any idle reduce slots available. At such cases, we can allocate map slots for reduce tasks for the pool.

Furthermore, there is a special scenario that needs to be considered particularly. Note, it is possible that all the above four possible slot allocation attempts fail for all pools, due to the data locality consideration for map tasks.

CONCLUSION

This paper present the idea of enhancing the storage techniques for Big Data using Hadoop. DynamicMR provides extra features that can be used to accelerate the information retrieval using the Hadoop technology. MRV1 lacks in the efficient storage of data. The MRV1 uses a collector which has a collection of algorithms to sort the data. As in DynamicMR the collector is not present and the data is stored dynamically. The DynamicMR also uses a mapper to map the accurate data and a reducer that can reduce the storage slots of this data. Thus, DynamicMR utilizes the idle as well as busy slots when retrieving information or when storing data.

ACKNOWLEDGEMENT

We take this opportunity to express our profound gratitude and deep regards to our respected Principal Dr. A. A. Keste , H.O.D. Prof N. F. Shaikh and our project guide

Prof.B. K. Bodkhe for their guidance, monitoring and constant encouragement throughout the course of this thesis .The blessing, help and guidance given by them time to time shall carry us a long way in the journey of life on which we are about to embark.

REFERENCES

I. F.Ahmad,S.Y.Lee, M. Thottethodi, T. N. Vijaykumar. *PUMA: Purdue MapReduce Benchmarks Suite*. ECE TechnicalReports, 2012.

II. G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, *Reining in the outliers in map-reduce clusters using mantri*, in OSDI'10, pp. 1-16, 2010.

III. J. Dean and S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*, In OSDI'04, pp. 107-113, 2004.

IV. Hadoop. http://hadoop.apache.org.

V. A Recommender System Based on a Machine Learning Algorithm for B2C Portals http://ieeexplore.ieee.or

VI. A Case-Based Recommendation Approach for Market Basket Data http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6894473newsearch=truequeryText=Casebased

VII. Trust-based decision-making for the adaptation of public displays in changing social contexts http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6596068newsearch=Study

VIII. Trust enabled Argumentation Based Recommender System http://ieeexplore.ieee.org/articl

IX. A Big Data Model Supporting Information Recommendation in Social Networks http://ieeexplore. ieee.org/xpl/articleDetails.jsp? number=6382911queryText=Social