Efficient Performance Optimization on Yarn-Based MapReduce Hadoop Framework

Than Than Htay, Sabai Phyu
Thanthanhtay@ucsy.edu.mm, sabaiphyu72@gmail.com

Abstract

Apache Hadoop exposes 180+ configuration parameters for all types of applications and clusters, 10-20% of which has a great impact on performance and efficiency of the execution. The optimal configuration settings for one application may not be suitable for another one leading to poor system resources utilization and long application completion time. Further, optimizing many parameters is a time consuming and a challenging job because configuration parameters and search space are huge, and users require good knowledge of Hadoop framework. The issue is that the user should adjust at least the important parameters, e.g. the number of map tasks that can run in parallel for a given application. This paper introduces the parameter optimization algorithm to the key application level parameter based on input data size and dynamic resource capabilities at any given time for a given application to improve execution time and resource utilization with nearly zero optimization overhead.

1. Introduction

In this Big Data world, huge data storage and faster processing is a big challenge because many companies and organizations become interested in processing and analyzing big data to extract valuable information from big data. Hadoop is a desirable solution to this challenge.

The Apache Hadoop framework has become a popular, reliable, and scalable open-source framework for storing and processing big data in a distributed fashion on large clusters of low cost commodity hardware by using HDFS and MapReduce programming framework, respectively. HDFS is highly fault-tolerant through replication management like multiple copies on block of data on different node and scalability and is designed to be deployed on low cost hardware [5].

As Hadoop allows for the distributed processing of large datasets across clusters of low cost commodity machines with generality, scalability, high availability and fast processing, MapReduce has been widely accepted as the most popular distributed processing framework for data-intensive applications in different contexts like click-log mining, web crawling, bioinformatics processing and machine learning, image processing, and data analysis, etc. A typical characteristic of these applications is that they run repeatedly with different input data sets [4]. YARN is an improved architecture of Hadoop and separates resource management from application logic [1]. The generalization of resource management makes it easier to deploy not only MapReduce applications, but also other applications such as Spark and Tez [1]. Despite the improvement in architecture of Hadoop in YARN and its wide adoption, performance optimization for MapReduce applications remains challenge because parameter tuning to optimize the performance is largely manual and is done via configuration parameters.

Hadoop exposes 180+ parameters providing users the flexibility to customize them according to their need. Some parameters have significant impact on MapReduce performance and manual tuning is time consuming and difficult because of the large parameter space and the complex interactions among the parameters [7]. Therefore, automatically tuning the configuration parameters for a given MapReduce Hadoop application on a given cluster to achieve optimized performance is highly desirable [6]. There are many recent research works for performance optimization of MapReduce Applications by using auto-tuning approaches: a cost-based approach, popular and flexible Machine learning models and a search-based [9, 6, 10]. The auto-tuning approaches (primarily working with task level parameters) are complex and require good enough tuning time to find optimal settings with a certain amount of overhead because they usually consider many task (map and reduce) level parameters with their respective possible values or range. Therefore, the major challenge is to quickly identify the optimal parameter configuration settings for a particular application on a given cluster that effectively utilizes system resources, ensures faster completion of applications.

To this end, it is desirable to select a small number of parameters that significantly affect the performance of MapReduce application. [1].

In this paper, we propose efficient performance optimization approach on Yarn based Hadoop in order to improve cluster resource utilization leading to faster completion time of MapReduce. We focuses on application level parameters that make impact the cluster resource utilization and overall application execution time on map stage with nearly zero overhead of optimization approach (just take a function call time). We proposed efficient parameter optimization algorithm to dynamically adjust the application level parameters: the number of map tasks via finding optimal split size based on dynamic resource capacity (number of containers).

In the following sections of the paper, we present background theory, discuss MapReduce Hadoop configuration parameter tuning for performance optimization, describe the proposed optimization system and expected performance improvement discussion and finally concludes the proposed system.

2. Background Theory

This section describes the Apache Hadoop and how clients request resources from Apache Hadoop Yet Another Resource Negotiator (YARN) to execute their applications.

2.1. Apache Hadoop

Hadoop is a framework for distributed data storage and data processing. Distributed data storage is provided by the HDFS (Hadoop Distributed File System) service, and distributed processing is provided by YARN. YARN is essentially a distributed operating system that provides a baseline of services for distributed applications. It can still perform MapReduce framework and it also provides a suitable framework to a variety of other distributed applications like search, data streaming, in-memory and real-time distributed data processing [3]. To take advantage of data locality the program of MapReduce application is also distributed across the nodes, preferably on the same nodes where data block resides.

Hadoop MapReduce framework divide the input data into smaller chunks, referred as input splits in Hadoop. These input splits and records are logical; they don't store or contain the actual data. They just refer to the data which is stored as blocks in HDFS. For each input split Hadoop creates one map task to process records in that input split. That is how parallelism is achieved in Hadoop framework. When a MapReduce application is run, mappers start producing intermediate output internally; lots of processing is done by the Hadoop framework before the reducers get their input. Once reducer has got its respective portion of intermediate data from all the mappers to create reduce task input and performs user specified reduce function. The final reduce output is written on HDFS.

2.2. Execution model of Apache Hadoop YARN

The application execution in Yarn starts with a client contacting Application Manager component of the Resource Manager (RM) and requesting resource for an Application Master (AM). Next, step 1: the client notifies the RM that it wants to submit an application. Step 2: The RM responds with an ApplicationID and information about the capabilities of the cluster that will aid the client in requesting resources for AM. Next, step 3: the client responds with a Application Submission Context and Container Launch Context (CLC). [4] When the RM receives the application submission context from a client, it schedules an available container for the Application Master (AM). Step 4: If a suitable container is available, then RM communicates the corresponding Node Manager (NM) that resides the container and starts the AM and at first AM sends registration request to the RM. The Container represents an allocated resource partition with configured size of such as memory and CPU on a single node in the cluster and many containers can exist on a single node [4].

When an AM starts up, step 5: the RM will send information about the minimum and maximum capabilities of the cluster in response to the registration request of Application Master. YARN allows applications to adapt (if possible) to the current cluster environment and the AM need to decide how to use the currently available resource capabilities. If an application is small and container resources are sufficient to complete the application, the application runs locally on the node that run AM of a given application. Otherwise, step 6: an AM calculates the number and configuration of more containers needed for the application, then requests a certain number of containers by specifying the amount of memory and virtual cores on the available

resources' capabilities reported from the RM. Step 7: the RM will respond, as best as possible based on scheduling policies, to this request with container resources that are assigned to the AM. The scheduler distributes currently available resources by partially satisfying incoming requests, as total of requested resources is usually much greater than the total of currently available resources. This action is repeated periodically, on each heartbeat. Simultaneously allocated containers form a wave. The size of memory and the number of virtual cores per container are set in configuration file by the user, and are static throughout the application execution. They affect application completion time resource utilization significantly and we can improve the performance of MapReduce applications by tuning the corresponding performance impact parameters. The growing importance of Apache Hadoop, the severity of the impacts of parameter selection, along with the limitation of resources, has led to vivid interest in the research community [4]. When the application finishes, the AM sends a finish message to the RM and exits [4].

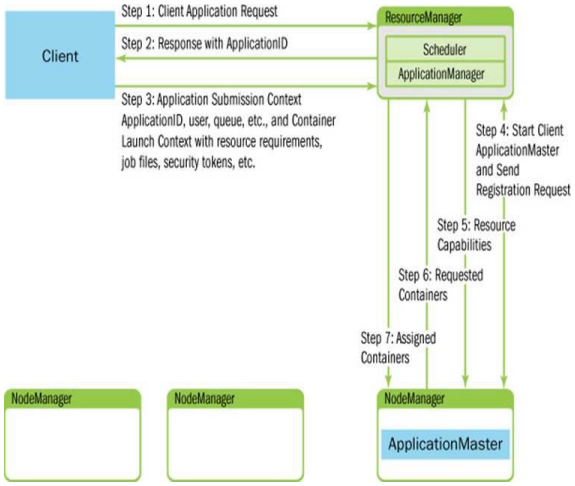

Figure 1. YARN Execution model with client resource request

3. Related Works

Hadoop MapReduce performance optimization is an interesting research area and optimization opportunities are taken from the large space of configuration parameters for the MapReduce jobs. There are many previous works in order to find a good setting of Hadoop configurations that achieves optimized performance for a Hadoop program running on a given cluster. H. Herodotou et al. [9] proposed a cost-based optimization system of MapReduce program to estimate the optimal configuration setting by using three components: profiler to collect detailed statistical information online from MapReduce job, optimizer to search optimal configuration, What-if-Engine based on analytical models to predict the cost (i.e. phase level execution time based on previously profiled information for the same program) needed by the optimizer. As per the experimental results of RFHOC [6], this system [9] is ineffective because it cannot detect the non-linear effect of parameters (experimented parameter: io.sort.factor) leading to suboptimal configuration and makes simple assumption to estimate execution time per operation (per-byte or per-record processing) to be constant across different configurations. To do end of these limitations, Z. Bei et al. [6] proposed RFHOC, an automated performance tuning approach that adjusts the Hadoop configuration parameters for an application running on a given cluster to achieve optimized performance. RFHOC constructs random forest based models to accurately predict the performance of Hadoop programs without making any assumption and applies a genetic algorithm to search the optimization space, yielding overall better Hadoop performance. G. Liao et al. [10] developed Gunther, a search based parameter tuning approach that aggressively identifies parameter settings which leads to near-optimal job execution times. The experimental results showed that this approach can obtain near-optimal performance within 30 trials using six configuration parameters with specified value range. However, Gunther needs to run the target Hadoop program once for each iteration of the genetic algorithm (GA), which is time-consuming for the applications with large input data sets. In contrast, RFHOC applies random forest based performance models to predict performance for each iteration of the GA (without executing the application), which is much faster. M. Li et al. [2] proposed MRONLINE, an online parameter tuning approach by using a gray-box based hill climbing algorithm to efficiently search a desirable configuration for each task of specific application on task-level configuration framework. The evaluation result showed that the online tuning approach gains performance improvement up to 30% compared to default setting in YARN.

In summary, previously proposed methods conducted performance optimization in various ways on many task level Hadoop MapReduce

configurations (do not optimize on key job level parameter considered in this system) with a considerable amount of optimization overhead and they do not consider resource availability for effective cluster resource utilization. In this paper, we focus on key job level parameter which affects performance of the overall MapReduce job. In this paper, we proposed an approach to efficiently optimize the performance of application on YARN-based Hadoop framework via calculated optimal split size with nearly zero optimization overhead by leveraging the cluster resource effectively. We can gain further improvement for the existing Hadoop MapReduce performance optimization system by integrating the proposed approach with the optimization systems that mainly focus on task level configuration parameters.

4. Performance Optimization on YARN-based MapReduce Hadoop framework

In this section, we describe parameter classification and selection, the impacts of the number of map tasks in MapReduce applications and the corresponding configuration parameter, split size that can govern the impact, the performance optimization via proposed algorithm and present the expected performance improvement by creating a scenario.

4.1 Parameter Classification and Selection

We can classify the Hadoop configuration parameter into task level parameters that affect the performance of the task and application level parameters that affect the performance of application. In former class, we can change the different parameters' values for different task. Application level parameters (such as input split size, the number of map tasks, and the number of reduce tasks, etc.,) are impossible to tune after the application has been initialized. Therefore, application level parameters, input split size, the number of map tasks is important and selected for parameter optimization.

4.2 Issues of Selected Parameter on MapReduce Performance

When a MapReduce job is run to process an input data, Hadoop framework divide the input data into independent smaller chunks, these chunks are referred as input splits which are processed by the map tasks in a parallel manner. Hadoop creates one map task per input split, thus number of map tasks is equal to the number of input splits. Furthermore, the input split size and the application input data size of the Hadoop MapReduce application control the total number of Map tasks spawned by the Hadoop framework [8]. In Hadoop, the default value of input split size is equal to HDFS block size. In this case, it is not suitable for different input data sizes that usually run in MapReduce applications. For the MapReduce application running with a large number of short running Map tasks that each task takes only a few seconds most time of the application will be spent on setting up and scheduling tasks as process time is very less. The aggregate pressure on the scheduler and the cluster incurs in slow application execution time on application performance. Therefore, the proposed system optimizes the number of map tasks via tuning split size based on the size of input data by considering dynamic Hadoop cluster resources.

4.3 Performance Optimization Approach

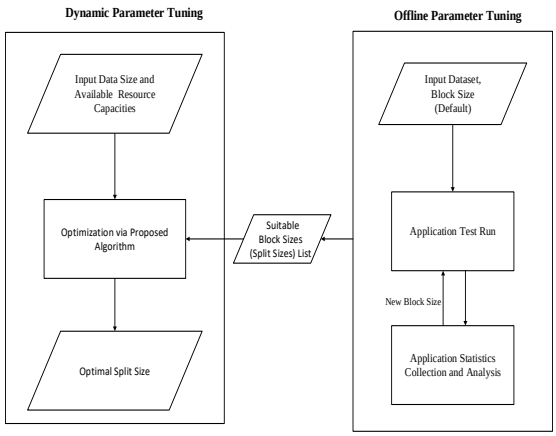

Figure 2. Performance Optimization System Architecture

In this paper, we propose performance optimization system on YARN-based MapReduce Hadoop framework. This system employs dynamic parameter tuning and offline parameter tuning. Dynamic parameter tuning find the optimal the number of map tasks for a given application via optimal split size by applying efficient parameter optimization algorithm. The input data size from hdfs, the dynamic available resources from RM and suitable split size list produced from offline parameter tuning are taken to the algorithm as input. Whereas, offline parameter tuning is manually performed on representative MapReduce applications by using small dataset with various block sizes to

reduce the testing time because changing the parameter block size is easy and simply. The overall performance optimization system architecture is shown in figure 2.

Table 1. Notation _meaning table

Notation	Meaning
D	input data
numContainer	number of containers that can run in parallel
total_numContainer	total numContainer for a running application
SS	split size
SS_List	suitable spilt sizes list
numSplit	number of splits
numMap	number of map tasks for a given application
numWave	number of waves

Before presenting the proposed algorithm, we describe how to implement the input split size in MapReduce framework. The MapReduce application creates Input splits by employing InputFormat class. Actually the input split size calculation for the input file is in the FileInputFormat class which is the super class for all the implementations of InputFormat. FileInputFormat method computes splits size via computeSplitSize method Math.max (minSize, Math.min (maxSize, blockSize) (inorg.apache. hadoop.mapreduce.lib.input.FileInputFormat class) which return split size. The parameter minSize is set via mapreduce.input.fileinputformat.split.minsize and maxSize is set via mapreduce.input.fileinputformat. split.maxsize, the splittable minimum chunk size and maximum chunk size input data [8]. Default values of them are 0(zero means no limit, thus split size is HDFS block size). In our proposed system, the computSplitSize method will be implemented by using proposed parameter optimization algorithm that computes optimal split size resulting in optimal number of map tasks.

4.4 Efficient Parameter Optimization Algorithm

The proposed efficient parameter optimization algorithm takes the size of input data from hdfs, input data size from hdfs, the dynamic available resources from RM and suitable split size list produced from offline parameter tuning and return optimal split size resulting in optimal number of map tasks leading to performance optimization. In order to find the optimal split size and derive number of map task for running the application on a given input data the proposed algorithm processes the following steps considered for possible conditions of available resources' capacity on YARN-based Hadoop cluster.

Algorithm 1: Efficient Parameter Optimization

Input: D; numContainer (it is based on available cluster resource capacity for a given time and application); SS_List, whereas those split sizes are obtained from offline manual tuning;
Output: optimal SS

```
1.  Current SS=D / numContainer
2.  if current SS ∈ SS_List
3.     optimal SS=current SS
4.     then return optimal SS
       //numMap=D/optimal SS or numContainer

5.     else if current SS < smallest SS from
                                SS_List
6.        optimal SS=smallest SS
7.        then return optimal SS
          //numMap=D/optimal SS
8.        else
9.         current numSplit =D/largest SS from
           SS_List
10.        numWave =numSplit/numContainer
11.        Total_numContainer = numWave *
                                  numSplit
12.        optimal SS=D/total_numContainer
13.        return optimal SS
           //numMap= D/optimal SS
14. end if
```

Step-1 (line 1-4: available resources' capacity fits application's resource requirement) (formula 1: Total time taken =number of wave * execution time per map task). In this step, all map tasks are concurrently run within a single map execution wave and complete at the same time, thus the execution time of application is a single optimal map execution time due to optimized SS. Therefore, map task execution time affects the application execution time, it is extremely important to optimize the split size that affects the map execution time and numMap.

Step 2 (line 5-7: available resources' capacity is larger than application's resource requirement) All map tasks from one application can be run in parallel within a single execution wave and they finish at the same time. In this case, we need to modify the SS with suitable smallest SS from SS_list to reduce the network traffic of reduce stage in copying multiple intermediate map outputs resulting from many map task executed on smaller split size.

Step 3 (line 9-13: available resources' capacity is smaller than application's resources requirement) In this step if last wave run small number of task, the other available resource cannot fully utilize. In this

case, we need to adjust the number of map task via optimal SS to effectively use the available resource by means of performance optimization in step 3 leading to improve execution time and cluster resource utilization. The execution time of application can be predicted via formula 1.

4.5 Expected Performance Improvement

In parameter tuning for performance optimization, increasing the input split size to get more data to process results in creation of less map tasks and reduce overhead. In this case, split size can be increased in order to run the Map task for 1-3 minutes at least according to rule of thumb. However, if available resource is higher than number of mappers with increased split size, the cluster resource cannot be used fully and the execution time of the application can achieve sub-optimal solution only.

In this paper, we present how proposed system can improve the execution time of MapReduce job and resource utilization by creating a scenario: If an application is run on YARN architecture with input data of 5120 MB (2G), hdfs block size of 256MB (take 120 seconds per split/task to execute) and 128 MB (take 80 seconds) and total numMap is 20 and 40, respectively (256MB and 128MB include in SS_list). If the numContainer that can run in parallel is 20, the proposed system can detect the optimal numMap via optimal SS (256MB) because the proposed algorithm can determine that the execution time to complete all map tasks for the application with the block size 256MB (that can run within 1 wave and execution time is 120s) is better than those of 128MB (that can run two wave and application execution time is about 2*80=160 s). If the cluster has enough resource to run all map tasks of 20 or 40 simultaneously, the proposed system can detect the optimal numMap via optimal SS (128MB) because the execution time of all map tasks for the application with 20 and 40 tasks is about 120s and 80s, respectively. Therefore, the proposed system can improve not only execution time but also cluster resource utilization. In this case, if we cannot detect the optimal split size based on input data and available cluster resource, the system cannot achieve optimal performance.

In the future, we will perform experimental evaluation by running on multi-user YARN cluster and compare the performance of proposed system with the Hadoop default configuration.

5. Conclusion

Performance optimization for Hadoop MapReduce applications is still an open issue in research community. Performance of MapReduce application can be improved without increasing the hardware costs, by tuning key configuration parameters. In this case, the major challenge lies in quickly identifying the best settings for a particular application on a given cluster with negligible overhead. Therefore, this paper proposed the performance optimization system on YARN-based MapReduce Hadoop cluster to efficiently improve the performance of MapReduce application by proposed algorithm. The effective resource usage and optimal SS due to the proposed algorithm ensures faster completion time for applications with nearly zero overhead for optimization. Further, the proposed system does not keep track of application performance, thus it will not incur the performance monitoring overhead. In the future, we intend to develop performance optimization method to include both application level and task level parameter, whereas optimization on task level parameters is based on the SS of the current proposed system on application level parameter.

References

[1] K. Kc, and V. W. Freeh, "Dynamically Controlling Node-Level Parallelism in Hadoop", *2015 IEEE 8th International Conference on Cloud Computing*, IEEE, New York, NY, USA, 2015.

[2] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, and N. Fuller, "Mronline: Mapreduce Online Performance Tuning", *Proc. of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, ACM, 2014, pp. 165–176.

[3] M. Fudge, *Introduction to Hadoop [Online] Available:* http://mafudge.mysite.syr.edu/BigData/intro-hadoop/ [Accessed: November-2018]

[4] A. C. Murthy, V. K. Vavilapalli, D. Eadline, J. Niemiec, and J. Markham, *Apache Hadoop YARN : moving beyond MapReduce and batch processing with Apache Hadoop 2, Addison-Wesley, 2014.*

[5] B. J. Mathiya,, and V. L. Desai, "Apache Hadoop Yarn Parameter Configuration Challenges and Optimization", *2015 International Conference on Soft-Computing*

and Networks Security (ICSNS), Coimbatore, India, February 25–27, 2015.

[6] Z. Bei, Z. Yu, H. Zhang, W. Xiong, L. Eeckhout, C. Xu, and S. Feng, "RFHOC: A Random-Forest Approach to Auto-Tuning Hadoop's Configuration", *IEEE Transactions on Parallel and Distributed Systems*, Vancouver, BC, Canada, 2015.

[7] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema, "Towards Machine Learning-Based Auto-tuning of MapReduce", *2013 IEEE 21st International Symposium on Modelling Analysis and Simulation of Computer and Telecommunication Systems*, IEEE, San Francisco, CA, USA, 2013.

[8] T. Revathi, K. Muneeswaran, and M. B. B. Pepsi, *Big Data Processing with Hadoop*, IGI Global, USA, August 24, 2018.

[9] H. Herodotou and S. Babu, "Profiling, What-if Analysis, and Cost Based Optimization of MapReduce Programs", *Proc. of the VLDB Endowment*, vol. 4, no. 11, Seattle, Washington, 2011, pp. 1111–1122.

[10] G. Liao, K. Datta, and T. L. Willke, "Gunther: Search-Based Auto-Tuning of MapReduce", *Proc. of Euro-Par 2013 Parallel Processing*, Springer, Aachen, Germany, August 26-30, 2013, pp. 406–419.