Improving the performance of Hadoop MapReduce Applications via Optimization of concurrent containers per Node

Than Than Htay
University of Computer Studies, Yangon
Yangon, Myanmar
thanthanhtay@ucsy.edu.com

Sabai Phyu
University of Computer Studies, Yangon
Yangon, Myanmar
sabaiphyu@ucsy.edu.mm

Abstract

Apache Hadoop is a distributed platform for storing, processing and analyzing of big data on commodity machines. Hadoop has tunable parameters and they affect the performance of MapReduce applications significantly. In order to improve the performance, tuning the Hadoop configuration parameters is an effective approach. Performance optimization is usually based on memory utilization, disk I/O rate, CPU utilization and network traffic. In this paper, the effect of MapReduce performance is experimented and analyzed by varying the number of concurrent containers (cc) per machine on yarn-based pseudo-distributed mode. In this experiment, we also measure the impact of performance by using different suitable Hadoop Distributed File System (HDFS) block size. From our experiment, we found that tuning cc per node improve performance compared to default parameter setting. We also observed the further performance improvement via optimizing cc along with different HDFS block size.

Keywords: *MapReduce, parameter tuning, concurrent containers, block size*

I. INTRODUCTION

Apache Hadoop is an open-source distributed software framework for large scale data storage and processing of data-sets on clusters of commodity hardware. Apache Hadoop architecture consists of the three main components: YARN (Yet Another Resource Negotiator), HDFS (Hadoop Distributed File System) and MapReduce. HDFS is used as a big data storage layer (for structured and unstructured data) in a distributed fashion with high throughput access to application data in a reliable manner and Hadoop MapReduce is the software layer implementing the MapReduce paradigm (only one of many possible framework which runs on top of YARN) and provides YARN based parallel processing of large data sets [5, 6]. YARN is an improved architecture of Hadoop that 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 it uses static resource management based on pre-defined resource units (a unit could represent to 1GB memory and 1 CPU core) called containers that are assigned to the map tasks and reduce tasks of MapReduce application. Therefore, the size of a unit and the total number of available containers per node are static in nature, i.e., we need to determine it before creating the cluster and/or prior to executing the application and it cannot be changed after start up the cluster [4, 11]. This form of static resource management has a limited ability to address with diverse MapReduce application resulting in poor performance. Therefore, the performance limitation of Hadoop yarn framework are tested and measured by running representative MapReduce applications on pseudo-distributed mode with YARN. In MapReduce framework, the reduce stage depend on output from the map tasks and they can start as soon as any map task finishes. Therefore, map tasks are important to finish as quickly as possible. In this work, map tasks are selected to optimize the map elapsed time of MapReduce applications along with optimal cc. In other word, finding optimal cc per node is determining the optimal number of concurrent map tasks per node. The experimental result shows that optimal cc yields significant performance improvement over Hadoop-default configurations. The number of map tasks for a job depends on HDFS block size. For example, if the input file size is 1024MB, the number of map tasks is 8 and 4 for HDFS block sizes with 128MB and 256MB, respectively. Where, additional time of map tasks (such as creating/destroying JVM, setting up/cleaning up at task level, etc.,) can be reduced by reducing the number of map tasks, Therefore, we also analyzed the impact of performance by running on different suitable HDFS block sizes. Optimizing cc along with different HDFS block size can achieve the further performance improvement over the default parameter setting.

In the following sections of the paper, we present background theory, determining the number of cc and why cc is selected for performance tuning, describe the performance evaluation on pseudo-distributed mode with Hadoop-2.7.2 and finally conclude the proposed system.

II. BACKGROUND THEORY

A. Hadoop MapReduce Framework

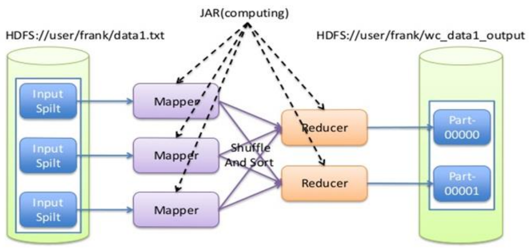


Figure 1. Overview of MapReduce Architecture

In Hadoop, the input files reside in HDFS (e.g., HDFS://user/frank/data1.txt) by dividing it into blocks with block size of 128MB. Hadoop InputFormat divides the input data into logical input splits in Hadoop. They just refer to the data which is stored as blocks in HDFS. In map stage, Hadoop creates one map task per input split and process records in that input split [13]. That is how parallelism is achieved in Hadoop framework. When a MapReduce application (JAR) is run, mappers start producing intermediate output internally; shuffling and sorting is done by the Hadoop framework before the reducers get their input. In reduce stage, once the reducer has got its respective portion of intermediate data from all the mappers, it performs user specified reduce function. The final reduce output is written on HDFS (e.g., HDFS://user/frank/wc_data1_output) as shown in figure 1.

B. YARN

YARN is essentially a distributed operating system that provides computational resources in the Hadoop cluster needed for running distributed applications. Apache Hadoop YARN Architecture consists of three main components: Resource Manager RM (one per cluster), Node Manager NM (one per node) and Application Master AM (one per application).

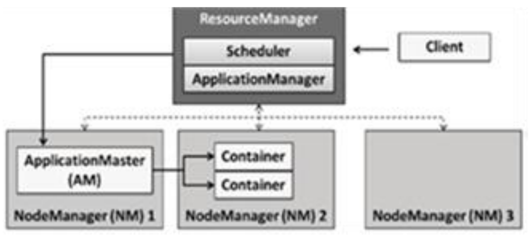


Figure 2. Overview of YARN Architecture

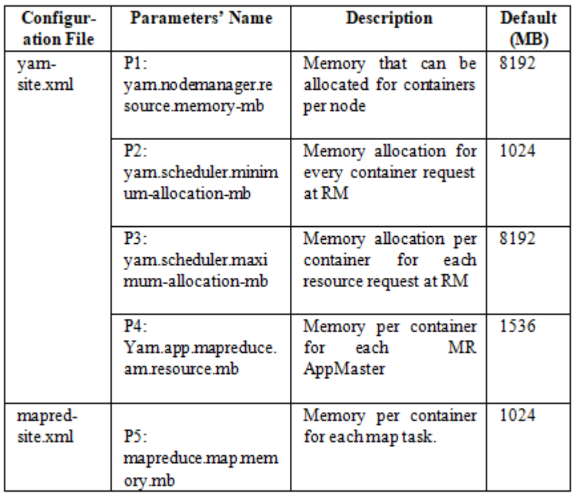
RM is the master of the YARN framework. It knows where the slaves (NM) are located and how many resources they have. RM has two main components: Scheduler and Applications Manager. Scheduler is responsible for allocating resources to the various running applications based on resource availability and the configured sharing policy. It also performs its scheduling function based on the resource requirements of the applications. Application Manager manages running the Application Masters in a cluster. It is responsible for accepting job submissions from the client, negotiating the first container from the Resource Manager to execute the application specific AM [14].

NM is the slave and offers some resources to the cluster. Its resource capacity is the amount of memory and the number of virtual cores (vcores). It takes care of individual nodes in a Hadoop cluster and manages application containers assigned to it by the RM. Container is a specified resource fraction (memory, CPU etc.) of the NM capacity on a host machine and many containers can reside on it [4]. How many number of cc can be run on a node, it depends on default resource calculator of capacity scheduler (default yarn scheduler). The DefaultResourceCalculator only checks memory amount specified by requests and the available memory of the node. By default, processing CPU resource is not considered.

AM is responsible for the execution of a single application. It asks for containers from the scheduler and executes specific programs on the obtained containers. It works along with the NM and monitors the execution status of tasks and progress for the application. Where, an application is a single job submitted to the framework. Each application has a unique Application Master associated with it which is a framework specific entity [14].

III. DETERMINING THE NUMBER OF AND WHY CC IS SELECTED FOR PERFORMANCE TUNING

TABLE I. MEMORY RELATED PARAMETERS THAT CONTROL THE NUMBER OF CC



Configur-ation File	Parameters' Name	Description	Default (MB)
yam-site.xml	P1: yam.nodemanager.resource.memory-mb	Memory that can be allocated for containers per node	8192
	P2: yam.scheduler.minimum-allocation-mb	Memory allocation for every container request at RM	1024
	P3: yam.scheduler.maximum-allocation-mb	Memory allocation per container for each resource request at RM	8192
	P4: Yam.app.mapreduce.am.resource.mb	Memory per container for each MR AppMaster	1536
mapred-site.xml	P5: mapreduce.map.memory.mb	Memory per container for each map task.	1024

The table 1 describes some important memory-related parameters and their corresponding default values based on the machine specifications for determining the maximum number of cc per node in a cluster. According to the default settings in table 1, the maximum number of cc is 6 (available RAM for tasks =p1-p4=8192MB-2048MB=6144MB, cc= 6144MB/1024MB based on p5, where p4 is set based on increment of p2 if the p4 is greater than p2). Therefore, the number of concurrent container per node is within the range of 1 to 6. Where, we need to select how many number of cc per node should be run for improving the performance of Hadoop. Therefore, we experimented on cc per node. According to experimental result, tuning cc choice can improve performance of map stage significantly. In other word, tuning cc allow to take advantage of node performance leading to faster execution time of MapReduce applications. Therefore, other performance impact parameters relating to Hadoop 2.x (such as the parameters in [12] collected from many research papers) can be tuned to achieve better performance improvement based on proposed system.

IV. EXPERIMENTAL EVALUATION

This section presents the experimental environment and the experimental results obtained for various workloads.

Our motivation for this experiment is two-fold: to measure the execution time difference of various cc (on the same HDFS block size) per machine instance and to measure the performance impact of HDFS block size for possible cc per node, where possible cc is within the range of 1 to 6.

A. Experimental Setup

In our experiment, Hadoop-2.7.2 is used and is deployed on virtual machine (vm) using VMware Workstation 12 Pro. This machine is setup with 8G RAM and 4vcores and 200GB disk space. On this vm, pseudo-distributed mode with YARN is setup. This mode is a single-node cluster where all the Hadoop components will run on a single machine and is mainly used for testing purpose.

In this paper, we use a single machine instance and run only one MapReduce application (i.e., concurrently running application is one) at each test run. Thus, cc is calculated for running map tasks per application.

B. Experimental Benchmark

TABLE II. EXPERIMENTAL BENCHMARK USED IN THE EXPERIMENT

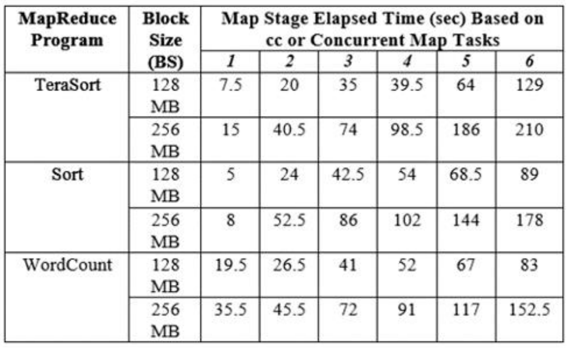
Benchmark MapReduce Program	Map Tasks or cc	Benchmark Dataset(MB)		
		Data Generator	***BS=128***	***BS=256***
TeraSort	1, 2, 3, 4, 5, 6	TeraGen	128, 256, 384, 512, 640, 768	256, 512, 768, 1024, 1280, 1536
Sort	1, 2, 3, 4, 5, 6	Random Writer	128, 256, 384, 512, 640, 768	256, 512, 768, 1024, 1280, 1536
Wordcount	1, 2, 3, 4, 5, 6	RandomText Writer	128, 256, 384, 512, 640, 768	256, 512, 768, 1024, 1280, 1536

Table II lists the selected representative MapReduce programs and benchmark datasets generated by data generators used in our evaluation. In benchmark programs, different MapReduce applications with a variety of typical Hadoop workload characteristics are considered: WordCount (CPU-bound), Sort and TeraSort (I/O bound) [3]. These programs are taken from the Hadoop distribution and serve as standard benchmarks and they are commonly used to measure MapReduce performance of an Apache Hadoop cluster [7, 8, 2].

For each MapReduce program, we use 128 MB (default) and 256MB to evaluate the block size effect. Therefore, for each MapReduce program, the corresponding data generators are used to generate six input datasets (in Table 2) for each block size that produce required number of map task based on the number of cc. That is, cc is set to 1, we run 1 map task on 1 data block.

C. Analyzing the Effect of Map Stage Elapsed Time (MSET) Based on cc and BS for Experimented Applications

TABLE III. MAP TASK ELAPSED TIME BASED ON CC AND BS



MapReduce Program	Block Size (BS)	Map Stage Elapsed Time (sec) Based on cc or Concurrent Map Tasks					
		1	*2*	*3*	*4*	*5*	*6*
TeraSort	128 MB	7.5	20	35	39.5	64	129
	256 MB	15	40.5	74	98.5	186	210
Sort	128 MB	5	24	42.5	54	68.5	89
	256 MB	8	52.5	86	102	144	178
WordCount	128 MB	19.5	26.5	41	52	67	83
	256 MB	35.5	45.5	72	91	117	152.5

Table III shows the map task elapsed time of each MapReduce program (TeraSort, Sort and WordCount) based on possible concurrent containers for the experimental cluster. Each program is run on both HDFS block sizes: 128MB (default) and 256MB; and collect Map elapsed time on the corresponding input data sizes for three workloads. For each test run, we run three times in order to consider the execution time variances and for each test run, we choose the median values for better performance measurement

TABLE IV. PERFORMANCE IMPROVEMENT % BASED ON OPTIMAL CC FOR EACH PROGRAM AND BS

Map Reduce Program	Optimal cc	Improvement % for Input Data=15G	
		BS=128MB	*BS=256MB*
TeraSort	1	52.3322	59.15813
Sort	1	47.90419	64.23517
WordCount	4	4.076878	18.05475

Table IV shows the performance of map stage for all three MapReduce applications in terms of map stage elapsed time. In order to compare the performance difference between optimal cc based on BS and Hadoop's default cc. It is complex to explicitly/directly compare the differences because different cc run on different data size. (e.g. map task elapsed time on cc of 4 is longer than that of 2, however cc of 4 can run two times of input data size on cc of 2). To clearly compare the map elapsed time differences, the case of input data with 15GB is considered and the map stage elapsed time is computed based on the relation of map elapsed time on the number of cc along with block sizes in table 3 without running the workloads. The optimal number of concurrent containers per node is 1 for both TeraSort and Sort and 4 for WordCount.

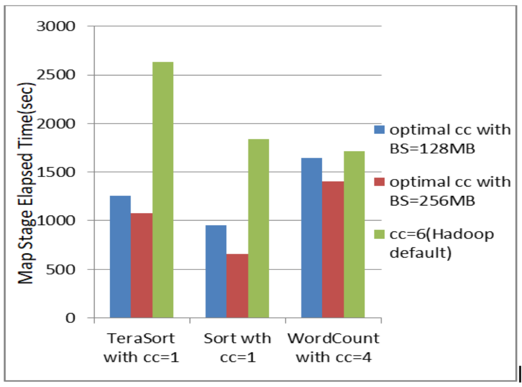


Figure 3. MSET Comparisons Between Hadoop Default cc and Observed Optimal Number of cc

This figure briefly describes the Map stage elapsed times for performance comparisons between observed optimal cc based on two block sizes(one is Hadoop default and other is suitable block size based on rule-of –thumb) over default cc. For all three workloads, optimal number of cc with block size of 256MB for each map task result in optimal performance with minimum the Map stage elapsed time, while default block size of 128MB is suboptimal.

D. Result and Discussion

Based on the computed results based on input data size of 15GB, the optimal number of concurrent containers per node is 1 for both TeraSort and Sort and 4 for WordCount. Where, sort and terasort require more disk other than CPU as these resources are shared among tasks. Consequently, the default based cc of 6 per disk on a single node cluster results in poor performance. Therefore, the optimal performance on Map stage can be obtained when the number of cc is one per disk. TeraSort program with optimal cc of 1 improve the performance of map stage by 52.3322% and 59.15813 with HDFS block size of 128MB and 256MB , respectively. Sort program with optimal cc of 1 improve the performance of map stage by 47.90419

% and 64.23517 % with HDFS block size of 128MB and 256MB, respectively.

However, in the case of WordCount, the optimal performance on map stage can be reached when the number of cc is 4 containers per node because the workcount is mostly CPU bound and the experimented machine instance has 4 vcores. For CPU bound job, using one map task per CPU can achieve optimal performance and for heavy disk or I/O, running only one map task per disk results in optimal performance in terms of map stage elapsed time. WordCount program with optimal cc of 4 improves the performance of map stage by 4.076878% and 18.05475% with HDFS block size of 128MB and 256MB, respectively. The optimal number of concurrent containers per node varies depending on the applications characteristics, such as heavy I/O and heavy CPU. Hence, prior to running MapReduce jobs, the cluster performance can be optimized via improving the node performance on map tasks by tuning the number of map tasks or cc per node along with suitable HDFS block sizes.

V. CONCLUSION

Performance optimization of MapReduce applications via Apache Hadoop Parameter configuration is still an open issue for researcher. This work investigates the performance limitation of Hadoop framework depending on container based static resource management and analyzes the performance improvement. The experimental results showed that different settings of concurrent containers running map tasks on the same default HDFS block size lead to different performance. In addition, we also observed that the performance can be further improved by tuning CC along with suitable HDFS block size. To show the performance differences among cc (along with two HDFS block sizes) per node, we measure the performance in terms of map stage elapsed time with example input data of 15GB for three MapReduce applications. This work achieves improvement of 59.15813, 64.23517 and 18.05475 in map stage elapsed time for the MapReduce applications: TeraSort, Sort and WordCount, respectively when compared to running them with Hadoop-default settings. In the future, the dynamic optimization on cc is performed by classifying the job types on fully-distributed mode by running multiple jobs concurrently.

REFERENCES

[1] Kc, Kamal, and Vincent W. Freeh, "Dynamically controlling node-level parallelism in Hadoop", In Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing (CLOUD'15), IEEE, 309–316.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", Communications of the ACM, 51(1), 2008, pp.107-113.

[3] Huang, Shengsheng, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis", In 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010), IEEE, 2010, pp. 41-51.

[4] Lee GJ, Fortes JA, "Improving Data-Analytics Performance Via Autonomic Control of Concurrency and Resource Units", ACM Transactions on Autonomous and Adaptive Systems (TAAS), 13(3), 2019, pp.1-25.

[5] Zhang, Zhuoyao, Ludmila Cherkasova, and Boon Thau Loo, "Parameterizable benchmarking framework for designing a MapReduce performance model", Concurrency and Computation: Practice and Experience, 26(12), 2014, pp.2005-2026.

[6] Xueyuan, Brian, Yuansong, "Experimental evaluation of memory configurations of Hadoop in Docker environments", In 2016 27th Irish Signals and Systems Conference. ISSC 2016", No.1, 2016.

[7] O. O'Malley and A. C. Murthy, "Winning a 60 Second Dash with a Yellow Elephant", Available: http://sortbenchmark.org/Yahoo2009.pdf.

[8] Grzegorz Czajkowski, "Sorting 1PB with MapReduce", Available: http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html.

[9] Yanfei Guo, Jia Rao, Dazhao Cheng, Xiaobo Zhou, "iShuffle: Improving Hadoop Performance with Shuffle-on-Write", IEEE Transactions on Parallel and Distributed Systems, 2017.

[10] Hoo Young Ahn, Hyunjae Kim, WoongShik You, "Performance Study of Distributed Big Data Analysis in YARN Cluster", In 2018 International Conference on Information and Communication Technology Convergence (ICTC), 2018.

[11] Gil Jae Lee, Jose A. B. Fortes, "Hierarchical Self-Tuning of Concurrency and Resource Units in Data-Analytics Frameworks", In 2017 IEEE

International Conference on Autonomic Computing (ICAC), 2017.

[12] Bonifacio AS, Menolli A, Silva F., “Hadoop mapreduce configuration parameters and system performance: a systematic review”, In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2014.

[13] DataFlair Team, “Hadoop InputFormat, Types of InputFormat in MapReduce”, Available:https://data-flair.training/blogs/hadoop-inputformat.

[14] Edureka, “Big Data and Hadoop”, Available:https://www.edureka.co/blog/introduction-to-hadoop-2-0-and-advantages-of-hadoop-2-0.