Usable Disk Space Control Based on Hadoop Job Features

Makoto Nakagami
Electrical Engineering and Electronics
Kogakuin University Graduate School
Tokyo, Japan
cm19036@ns.kogakuin.ac.jp

Jose A. B. Fortes
Advanced Computer and Information
Systems (ACIS) Lab, Department of ECE
University of Florida
Gainesville, USA
fortes@ufl.edu

Saneyasu Yamaguchi
Department of Information and
Communications Engineering
Kogakuin University
Tokyo, Japan
sane@cc.kogakuin.ac.jp

***Abstract*—Hadoop is an open-source platform for big data processing. In the case of analyzing extremely big data, hard disk drives are usually used. Hard disk drives have different performance depending on where data is placed on the disk. In other published work, an approach for improving Hadoop I/O performance by using features of Hadoop jobs to decide on the location of data placement was proposed. This approach separates the entire hard disk drive space into two areas, which are outer and inner areas. However, this approach did not actively utilize the fastest available zones in each area. In this paper, we propose a new method for improving this method by active usage of the fastest available zones in each area. Our evaluation of the proposed method with a popular Hadoop benchmark shows that the new method can improve job performance by 9% when compared with existing approach.**

Keywords—Big data, Hadoop, SWIM, Filesystem

I. INTRODUCTION

Hadoop is an open-source platform for big data processing based on the MapReduce model [1]. In many cases, Hadoop is used to analyze large-scale datasets stored and accessed sequentially in massive storage devices, such as hard disk drives. In the work [2], We proposed a method for improving sequential I/O performance of a single job by allocating its files to the outmost disk locations. In the work [3], We proposed a method for optimizing file placement location according to features of the jobs when multiple different jobs are executed in the Hadoop system.

This paper introduces a method for improving I/O performance by optimizing file placement location based on job features [3] and shows its issue. The approach reported in [3] divides the entire space of the target hard disk drive into two areas, which are the outer and inner ones, and chooses the area in which to place each file according to job features. However, the approach does not effectively utilize the fastest available zones within each area. The new method proposed in this paper addresses this issue by forcing files to be stored in the outmost zones within in each area by specifying what zones are to be used.

The rest of this paper is organized as follows. Section II reviews related work. Section III explains the features of SWIM jobs. Section 0 reviews the existing approach described in [3]. Section V proposes a new method that improves the existing approach. Section VI comparatively evaluates the proposed and the existing approach. Section VII discusses the proposed method. Section VIII concludes this paper.

II. RELATED WORK

A. MapReduce

Here, we introduce the basic MapReduce concepts that are relevant to the goal of this paper. Additional information can be found in [1]. Each MapReduce job is composed of three phases, namely the Map phase, Shuffle phase, and Reduce phase. In the Map phase, JobTracker splits Input Data in the HDFS into multiple Input splits. Map tasks are allocated to TaskTrackers. A TaskTracker receives an Input split, executes the user-defined Map process, and creates key-value pairs. These key-value pairs are stored in intermediate files. In the Shuffle phase, the JobTracker sorts the intermediate key-value pairs, groups the key-value pairs by the key and transmits them to the reducers. In the Reduce phase, Reduce tasks are allocated to the TaskTracker, each TaskTracker executes the user-defined Reduce process using the received key-value pairs as inputs and creates outputs.

B. SWIM

SWIM is a workload emulator that can generate practical MapReduce jobs based on real workloads from a Hadoop cluster used in production environments such as Facebook. Each SWIM job has configurable parameters such as input size (bytes) per map operation, shuffle size, and output size per reduce operation. Moreover, each job submission interval can be controlled [4]. In this paper, we evaluate Hadoop performance with various usage scenarios by varying these parameters in Facebook traces.

III. BASIC PERFORMANCE EVALUATION

In this section, we categorize SWIM jobs as Map-heavy, Shuffle-heavy, or Reduce-heavy jobs. We then investigate the I/O, CPU, and disk usages by each job. In our evaluation, the existing and proposed methods place the files according to these features.

We executed SWIM jobs on an experimental Hadoop system. The parameters are set as follows. *Submit time* and *inter job submit gap* is set to one for all the jobs. The *Input file size* is 20GB. In the case of Map-heavy jobs, only the *map input bytes* is set to 1.0×10^{12} and the others, *shuffle bytes* and *reduce output bytes*, are set to one. In the case of Shuffle-heavy jobs, *shuffle bytes* is set to 1.0×10^{12} and the others are set to one. In the case of Reduce-heavy jobs, *reduce output bytes* is set to 1.0×10^{12} and the others is set to one.

978-1-7281-5268-4/19/$31.00 ©2019 IEEE
DOI 10.1109/CANDARW.2019.00093

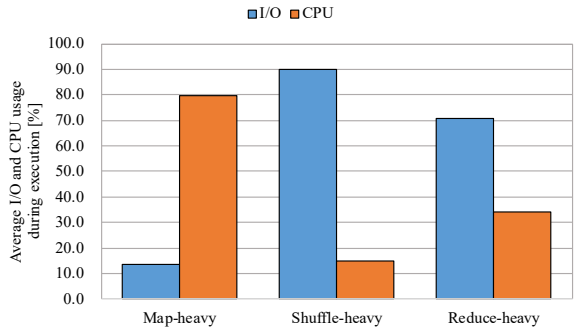

Fig.1. Average CPU and I/O usage of SWIM jobs

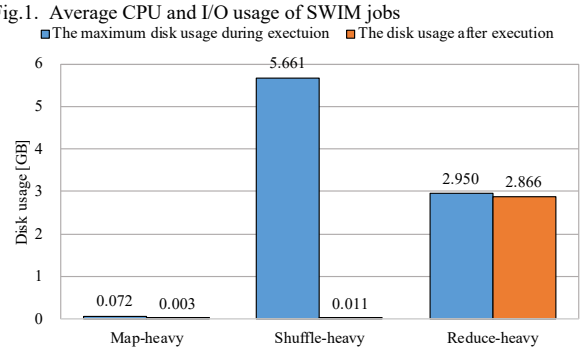

Fig.2. Maximum disk usage during the execution and disk usage after the execution of SWIM jobs

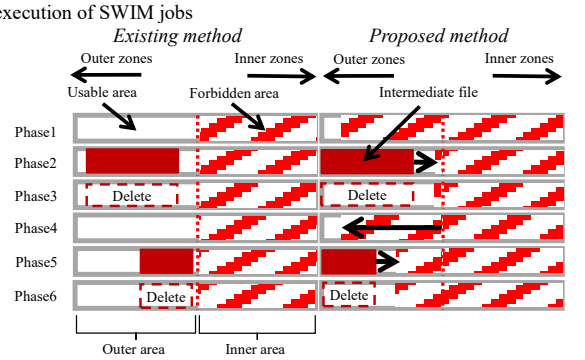

Fig.3. File placement of temporal files in the existing and proposed methods

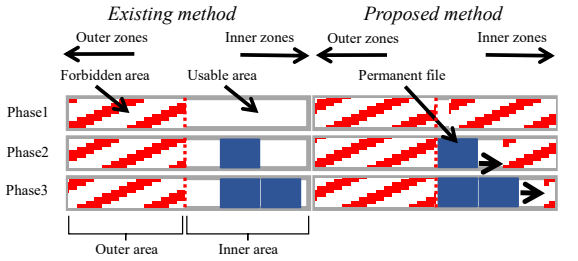

Fig.4. File placement of permanent files in the existing and proposed methods

The Hadoop system is set to the pseudo-distributed mode. The specification of the computer is as follows: the CPU is AMD Phenom2 X4 965 Processor, the HDD sizes are 150 GB and 500 GB, the memory size is 4GB, the OS is CentOS 6.5 x86_64, the kernel is Linux 2.6.32, and the filesystem for the system file HDD is Ext4, and the filesystem for Hadoop file HDD is Ext3. Hadoop version is 2.0.0-cdh4.2.1. All the Hadoop data, including intermediate data, are stored in the 500 GB HDD with the Ext3 format. System files are stored in the 150 GB HDD. The specification of the HDD for Hadoop data is as follows: its model is DT01ACA050, the interface is SATA3.0 6.0 Gbps, the capacity is 500 GB, the buffer Size is 32MB, and the rotation rate is 7200 rpm.

The average I/O usage and CPU usage during the execution of Map-heavy, Shuffle-heavy, and Reduce-heavy jobs are shown in Fig. 1. The maximum disk usage during executions and the disk usages after executions of these jobs are depicted in Fig. 2. These results lead to several conclusions. First, a Map-heavy job is CPU-intensive. It temporarily stores the intermediate data in the storage and deletes almost all of these intermediate data during the execution. Therefore, most of their files are temporary ones. Second, a Shuffle-heavy job is I/O-intensive and their files are temporary ones. Third, a Reduce-heavy job is I/O-intensive and permanently stores the output data (instead of deleting them). In other words, their files are permanent ones.

IV. Job-aware File Placement Optimization

In this section, we introduce the existing method for improving the I/O performance of Hadoop jobs by optimizing file placement based on job features[3]. This method assumes that jobs are submitted and executed sequentially. This method avoids placement of permanent files in the outer area and utilizes the outer area many times.

It divides the target hard disk drive into two areas, the outer and inner areas. It places files in the outer area according to the following priority: 1) The file is temporary and is used by an I/O intensive process. 2) The file is temporary and is used by a non-I/O intensive process. 3) The file is not temporary and is used by an I/O intensive process. 4) The file is not temporary and is used by a non-I/O intensive process.

In the case of executing Map-heavy, Shuffle-heavy, and Reduce-heavy jobs, the files of Shuffle-heavy and Map-heavy jobs are stored in the outer area and the files of Reduce-heavy jobs are stored in the inner area.

In our implementation, this method is constructed with ext2/3/4 filesystems [5]. These filesystems create block groups and every block group has its block bitmap for managing the usages of its blocks. The method forces Map-heavy and Shuffle-heavy jobs to use the outer area by setting the bits of the inner area blocks to 1, which indicates *used.* This method prevents Reduce-heavy jobs from using the outer area by changing their bits into 1. As a result, the permanent files of the Reduce-heavy jobs are not placed in the outer area.

This method has room to improve performance as shown in the left area in Fig. 3 and 4. A file for the outer area is not always stored in the outer zones inside the outer area. Similarly, a file for the inner area is not always stored in the outer zones inside the inner area. For example, a file is stored in inner zones in the outer area in phase 5 in Fig. 3.

V. Proposed Method

In this section, we propose to improve the existing method [3] by actively utilizing the fastest zones in each area. The right sides of Fig. 3 and 4 illustrate the proposed method. This method

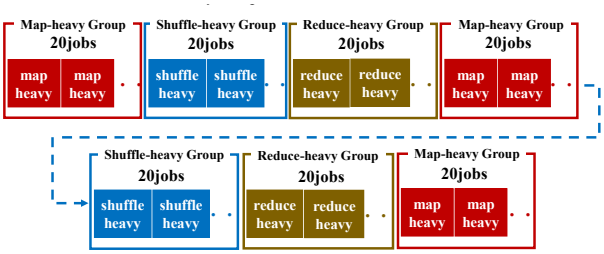

Fig.5. Job set.

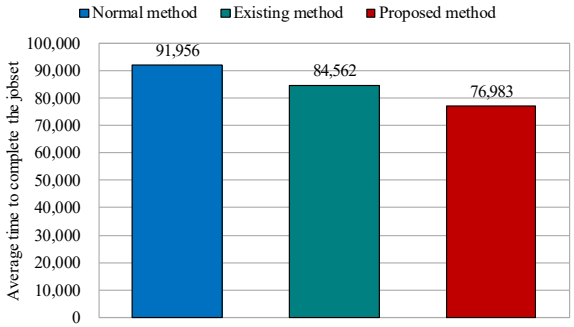

Fig.6. Total execution time of the job set

limits the zones in the area that are usable. As a result, every file is forced to be placed at the outmost zoned in each area by dynamically controlling the usable zones.

This method is implemented by three functions, the monitoring, expanding, and shrinking functions. The monitoring function periodically checks the number of free blocks in the filesystem. When the monitoring function detects that the usable space size is less than a threshold, the expanding function is invoked to expand the usable space until the size exceeds the threshold. The shrinking function is called when the monitoring function detects that the usable space size is more than the threshold and shrinks the usable space until the threshold.

VI. Evaluation

In this section, we evaluate the performance of the proposed method. We executed a job set that is illustrated in Fig. 5. It is composed of multiple job groups, sequenced as Map-heavy group, Shuffle-heavy group, Reduce-heavy group, Map-heavy group, and so on. A job set contains three Map-heavy groups, three Shuffle-heavy groups, and three Reduce-heavy groups. Each job group consists of 20 jobs.

In the case of the Map-heavy job group, the *map input bytes* is selected from $10^{10.5}$, 10^{11}, $10^{11.5}$, 10^{12}, and $10^{12.5}$ and the other parameters are set to one byte. Each size job is selected 4 times in a group. In the case of Shuffle-heavy and Reduce-heavy, only the *shuffle bytes* and *reduce output bytes* are selected from these five values, respectively. The others are one. Each execution of a set of jobs starts with the hard disk drive empty and then the drive is almost fully occupied by the output files of the jobs after execution of a set. Other setups are the same as Section III.

Fig. 6 shows the average time to complete a job set with 10 executions. Fig. 6 shows that the performance of the proposed method is the best. The execution time of the proposed method is smaller than the normal and existing methods by 16.3% and 9.0%, respectively. On the contrary, the existing method outperformed the normal method by only 8%.

VII. Discussion

First, we discuss the monitoring overhead. The proposed method has to execute a function to keep monitoring the size of usable space. This function periodically checks the number of free blocks which is stored in the superblock of the filesystem. This information is always stored in the page cache except for the first access. Thus, the effect of this monitoring on I/O performance is very small. The effect of CPU processing is also very small. The CPU usage by this monitoring 0.4%.

Second, we discuss an alternative for the control of file placement location that uses partitions instead of bitmaps. The existing method could be implemented by creating two partitions, which correspond to the outer and inner areas. However, this approach does not allow the control of the file location inside an area, (i.e. inside a partition). Thus, the detailed location control of the proposed method cannot be achieved by simply creating partitions.

VIII. Conclusion

In this paper, we reviews a method for improving I/O performance considering the features of the target jobs and discussed its shortcomings. We then proposed a new method that more actively utilize outer zones than the existing method. Our evaluation demonstrated that the proposed method improves the performance of the Hadoop jobs by 16.3% while the existing method did so by 8.0%. The proposed method outperformed the existing method by 9.0%.

In future work, we plan to extend and evaluate similar methods for concurrent jobs when Hadoop runs in fully distributed mode.

Acknowledgment

This work was supported in part by JST CREST Grant Number JPMJCR1503, Japan. This work was also supported by JSPS KAKENHI Grant Numbers 26730040, 15H02696, 17K00109. This work is also funded in part by a grant (NSF ACI 1550126 and supplement DCL NSF 17-077) from the National Science Foundation, USA.

References

[1] G. Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113. DOI: https://doi.org/10.1145/1327452.1327492

[2] Eita FUJISHIMA Kenji NAKASHIMA Saneyasu YAMAGUCHI, "Hadoop I/O Performance Improvement by File Layout Optimization", IEICE TRANSACTIONS on Information and Systems, Vol.E101-D No.2 pp.415-427, doi: 10.1587/transinf.2017EDP711

[3] Makoto Nakagami, Jose A.B. Fortes, Saneyasu Yamaguchi, "Job-aware Optimization of File Placement in Hadoop, " BDCAA 2019 The 1st IEEE International Workshop on Big Data Computation, analysis, and Applications, COMPSAC 2019, July 2019.

[4] GitHub - SWIMProjectUCB/SWIM: Statistical Workload Injector for MapReduce-Project at UC Berkeley AMP Lab, https://github.com/SWIMProjectUCB/SWIM

[5] R.Card and T.Ts'o and S.Tweedle, "Design and Implementation of the Second Extended Filesystem," First Dutch International Symposium on Linux, 1994