Workload Shifting: Contention-Insular Disk Arrays for Big Data Systems

Fengfeng Pan
University of Chinese Academy of Sciences, Beijing, China
State Key Laboratory of Computer Architecture, ICT, CAS
Email: panfengfeng@ict.ac.cn

YinliangYue
Institute of Information Engineering, CAS
Email: yueyinliang@iie.ac.cn

JinXiong
State Key Laboratory of Computer Architecture, ICT, CAS
Email: xiongjin@ict.ac.cn

***Abstract*—It is well known that in-place update index, unordered log structured index and ordered log structured index are three typical data organizations which are designed to meet different workload requirements respectively and wildly used in big data storage systems. Differentiated workload requirements in different phase of the data lifecycle, e.g. various types of data are injected into the big data storage systems in the write optimized manner, then they are needed to be read in the read optimized manner for analysis, lead to data organization transformation(data transformation for short). However, the simple mixture of foreground data injection and background data transformation causes serious disk contention. Frequent disk head seeks result in low disk throughput, and not only prolong the data transformation process, but also increase foreground data injection latency.**

In this paper, we propose *Workload Shifting*, a novel log-structured design that shifts background data transformation away from the foreground data injection. Compared with conventional RAID0 disk array, *Workload Shifting* effectively isolates background data transformation and foreground data injections, avoids the disk contention between them to boost their performance. We have implemented *Workload Shifting* prototype on one multiple disks based disk array. Extensive experimental evaluation results show that compared with conventional RAID0 disk arrays, *Workload Shifting* can avoid disk contention and speed up both data injection and data transformation significantly.

Keywords*—*data organization transformation, disk contention, log-structured, disk array, big data

I. Introduction

In big data era, massive data are generated by various types of clients, and converge together into the massive data storage systems. Then data are read from the storage systems and analysed by various data analysis tools and models, e.g. MapReduce, to produce useful and valuable information, which is used in business intelligence, decision supporting and so on. Data organization is important for storage systems because it largely determines the efficiency of data operations, such as read, write, scan and search. The data organization can be categorized into three classes: in-place update index, unordered log structured index, and ordered log structured index. B+tree, LFS (Log Structured File System [16]) and LSM-tree (Log Structured Merge tree [15]) are representative of the in-place update index, unordered log structured index and ordered log structured index respectively. B+tree and its variants provide worst-case write latency due to the small and random updates to disks, but has high read and scan performance, while LFS writes data to disk immediately without any sorting, and typically has excellent write throughput, at the expense of poor scan and read performance. LSM-tree is different from both of them, it weighs read performance against write performance. Initial evaluation results show that the read and scan performance of B+tree is higher than both LFS and LSM-tree. As well as, the write performance of LFS is higher than both B+tree and LSM-tree. Therefore, we'd better adopt LFS and B+tree in the data injection phase and data analysis phase respectively.

In fact, LSM-tree can be regarded as one of the data transformation schemes. Key value pairs are written to the MemTable in LFS manner and then the unordered key value pairs are transformed to sorted key value pairs via compaction. Although LSM-tree has been popularly used in the Google LevelDB [1], which is the local storage engine for BigTable [17], and other storage systems, including HBase [20], Hyperdex [19], PNUTS [18], and Cassandra [21]. We can find that there exists serious write amplification problem with LSM-tree and the throughput of LevelDB is only several mega bytes per second under common SATA disks environments. How to efficiently transform data from write oriented LFS manner to read and scan oriented B+tree manner is one of the critical problems in big data storage systems.

There are two advantages of LFS manner used in the foreground data injection. First, it is well-known that LFS typically has excellent write throughput and can speed up the process of data injection. Second, at the full speed of the underlying media, LFS can support sequential or random write streams from multiple applications, so it can avoid disk bandwidth contention among foreground applications. However, when data transformation is introduced to improve the read and scan performance of storage systems there exists serious disk bandwidth contention between the foreground data injection and background data transformation. Specifically, the random reads and writes incurred by data transformation often interfere with foreground data injection. This problem is also common in disk array, such as RAID0.

In this paper, we propose *Workload Shifting* which separates the foreground data injection from background data transformation and distributes them into different hard disks, in order to avoiding the disk contention. In details, we inject

978-1-5090-3315-7/16/$31.00 ©2016 IEEE

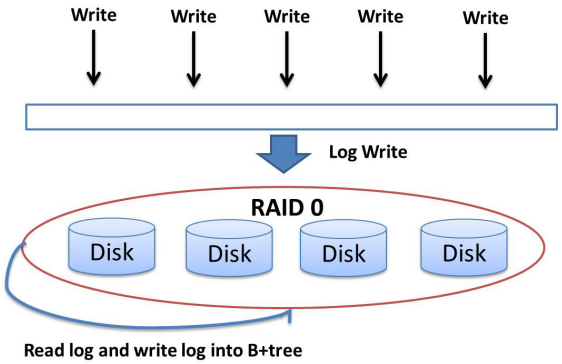


Fig. 1: The conventional data transformation scheme based on RAID0

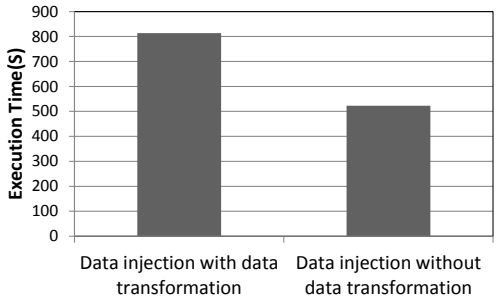


Fig. 2: The impact of disk contention between data injection and data transformation

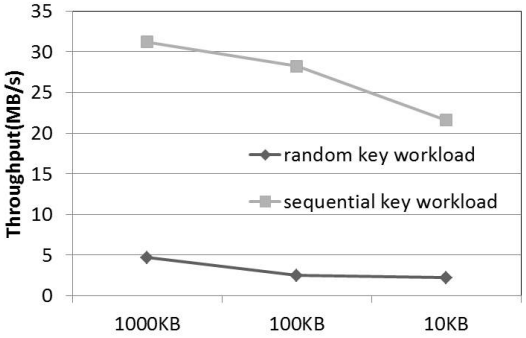


Fig. 3: The throughput of LevelDB under different workloads

data into one hard disk for some time and then shift the data injection task to another hard disk. All the disks take turns to be responsible for the data injection task. Meanwhile, background data transformation task is scheduled to handle the injected data. By this way, the injected data can be transformed as soon as possible, so it can facilitate the consequent read and scan operations.

We implement *Workload Shifting* based on one disk array with multiple disks. We also conduct extensive real experiments to evaluate the performance of *Workload Shifting* compared with the conventional data transformation schemes. Extensive experimental results demonstrate that *Workload Shifting* can avoid disk contention and speed up both data injection and data transformation effectively.

The rest of this paper is organized as follows. The background will be presented in Section II. In Section III, we will discuss the design of *Workload Shifting* in details. The performance evaluation will be described in Section IV. We will also present related work in Section V and conclude this paper in Section VI by summarizing our main contributions.

II. Background

The conventional data transformation scheme based on RAID0 array is showed in Figure 1. From Figure 1 we can see that the log-structured design converges lots of sequential and random write streams from multiple applications into sequential write streams and triggers data transformation periodically. Although there does not exist disk contention among foreground writes due to the log-structured design, there are still two types of disk contention in the conventional data transformation scheme based on RAID0 array. One is that according to the description above, data transformation is a background process, it will compete disk resources with foreground data injection. The disk contention disrupt not only data injection but also data transformation. The other is that there are both reads and writes during the data transformation, i.e. transferring data organization from log structured manner to B+tree manner. So when both reads and writes fall in one device, there exists disk contention between them, and the disk contention may slow down data transformation.

We conduct experiments on a 6-core machine with 4 disks configured as a RAID0 array to show the problems. We use log-structured designs to process data injection, as well as, trigger data transformation periodically. The key size and value size are respectively set as 16B and 100KB . We keep the volume of data set as 10GB. We mainly focus on the execution time of data injection, in order to guarantee the data reliability, we use *sync write* to inject data.

Since both foreground data injection and background data transformation run on top of RAID0 array concurrently, there exists serious disk contention between them. As the Figure 2 shows the execution time of data injection with data transformation and data injection without data transformation are 813.6s and 522.6s respectively. Due to the mixture I/Os of foreground data injection and background data transformation, the execution time of data injection with data transformation increases 55.7% compared with that of data injection without data transformation. So we can apparently make the conclusion that disk contention degrades the performance of foreground data injection significantly.

In addition, we conduct another experiment on a RAID0 array consisting of 4 disks to show the efficiency of LSM tree which can be regarded as one of the data transformation schemes. In the experiment, we run Google LevelDB based on a local filesystem EXT4 to observe its throughput. Note that Google LevelDB is one representative implementation of LSM tree. The workloads of this experiment can be categorized into two classes according to the type of key: *sequential key* workload and *random key* workload. The key size is set as 16 bytes and the value size is set as 1000KB, 100KB and 10KB respectively. Similarly, the volume of data set is kept as 10GB.

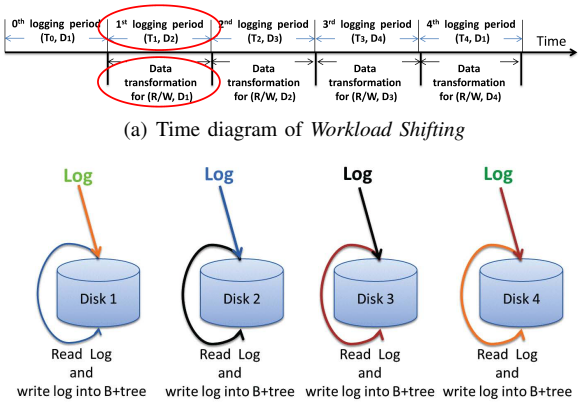
Figure 3 shows the throughput of LevelDB under two dif-

ferent workloads. For any value size under two different workloads, its throughput is limited to less than 32MB/s. The reason has two aspects. On the one hand, foreground data injection uses *sync write* to do a log for data reliability, unfortunately, *sync write* limits the performance of data injection. On the other hand, background data transformation and data injection interfere with each other. In Figure 3, the throughput under sequential key workload is distinctly larger than the throughput under random key workload. The reason is that there is no compaction (i.e. data transformation) in the background under sequential key workload. However, background compaction is triggered under random key workload, and background compaction leads to significant extra disk reads and writes which disturbs the foreground data injection I/O activities. The different throughput of LSM tree between the sequential key workload and random key workload demonstrate that the background compaction I/Os and foreground write I/Os interfere with each other significantly.

In conclusion, there are two challenges in the conventional big data systems. One is how to separate foreground data injection from background data transformation. The other is how to separate data transformation reads from data transformation writes. In the following section, we will describe *Workload Shifting* step by step based on the above two challenges.

III. Workload Shifting Design

A. The basic model of Workload Shifting



(a) Time diagram of *Workload Shifting*

(b) The basic model of *Workload Shifting*

Fig. 4: The process of basic *Workload Shifting*. D_i presents the i^{th} disk, T_j presents the j^{th} logging period, and R/W represents read and write respectively.

1) Data Injection: Multiple writes are organized as log-structured manner by *Workload Shifting* to avoid the disk contention among foreground write I/Os naturally. Then, instead of treating multiple disks as an entire device(i.e. RAID0 array), *Workload Shifting* uses the disks separately to separate the foreground data injection workloads from background data transformation workloads and assigns them into different hard disks to avoid the disk contention. Specifically, *Workload Shifting* injects data into one hard disk for some time and then shifts the data injection task to another hard disk. All the disks take turns to be responsible for the data injection task. Meanwhile, background data transformation task is scheduled to handle the just injected data as shown in Figure 4. By this way, the just injected data can be transformed as soon as possible and it can facilitate the consequent read and scan operations. For instance, D_1 is used as the on-duty log disk in logging period T_0 firstly(i.e. all of the foreground key value pairs are written to D_1). Similarly, when entering logging period T_1 foreground I/Os fall into D_2, and key value pairs are written to D_3 during T_2. This process continues until data injection has been finished.

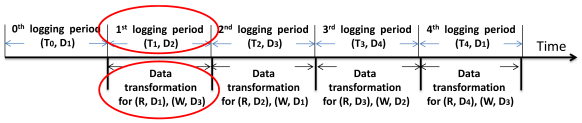
2) Data Transformation: A new data transformation process is triggered when foreground data injection rotates to a new disk and it will not finish until all the key value pairs of the corresponding disk have completed data transformation. For example, the data transformation process for D_1 is triggered immediately when the foreground data injection rotates to D_2. The only responsibility of data transformation process is to reorganize key value pairs from log structured manner into B+tree manner in the same disk. So the data transformation involves two steps named read data which is organized as log-structured and write data into B+tree. As the Figure 4 shows, only when all the key value pairs of the corresponding disk have been updated in D_1, this data transformation process can be terminated. Similarly, the data transformation process for D_2 is triggered immediately after D_3 is selected as the on-duty log disk.

Compared to the conventional data transformation scheme based on RAID0 array, the major improvement of the basic model of *Workload Shifting* is separating foreground data injection from background data transformation. However, data transformation reads are mixed together with writes on the same disk, so the current basic model of *Workload Shifting* can not avoid the disk contention between data transformation reads and writes. One weakness of *Workload Shifting* is that it can not make good use of RAID0's parallelization to speed up read and write performance, but from the following experiments, we can know that the overheads resulting from disk contention is more serious than performance improvement from multiple disks' parallelization, hence the weakness of *Workload Shifting* is negligible.

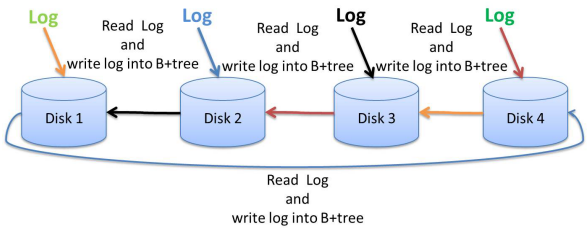
B. The enhanced model of Workload Shifting

In order to avoid the disk contention between the read and write I/Os of background data transformation in the basic model of *Workload Shifting* which is shown in Figure 4, we propose the enhanced model of *Workload Shifting* as shown in Figure 5.

The enhanced model of *Workload Shifting* is similar with the basic model shown in Figure 4(a) and 4(b), the main difference is the separation of data transformation read and write I/Os. In Figure 4(b), data transformation read I/Os and write I/Os are in the same disk. Unlike the basic model of *Workload Shifting*, the enhanced model of *Workload Shifting* reads the data from one disk and writes them into another disk to separate the read I/Os and write I/Os. Note that in Figure 4(a) and 5(a), the logging period and data transformation period are the same under our assumption(the cycle parts). Our assumption of the logging period and data transformation period is ideal, however in actual scenes, logging period and data transformation period can not be completely synchronized. In consequence, we analyze some typical assumptions as



(a) Time diagram of *Workload Shifting*



(b) The enhanced model of *Workload Shifting*

Fig. 5: The process of enhanced *Workload Shifting*

follows to validate the feasibility and effectiveness of *Workload Shifting*.

C. Analysis of Workload Shifting

1) ***Ideal alternation of disk stages****:* As shown in Figure 6, there are four stages in every disk of *Workload Shifting* named Log, Read log, Write B+tree, and Idle. Table I gives the detailed description about the four stages.

TABLE I: The meaning of four stages

Log	Write foreground data as log-structured manner, and once the log size of one disk is reached the threshold value, *Workload Shifting* changes the disk to log and triggers the data transformation
Read log	Read data from one disk during the transformation process
Write B+tree	Write data to B+tree into another disk during the transformation process
Idle	The disk is idle, and nothing to do

From the Figure 6, we can see the progress of Log, Read log and Write B+tree will not disturb with each other by carefully adjusting the time period of Log, Read log, Write B+tree and Idle, hence the disk contention among them can be eliminated perfectly. However, it is difficult for us to coordinate the steps.

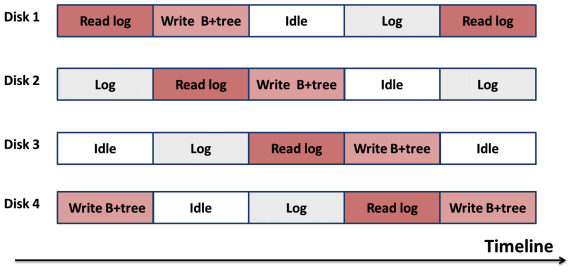
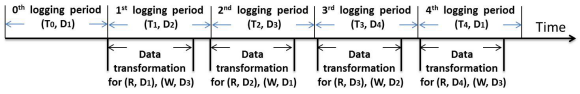
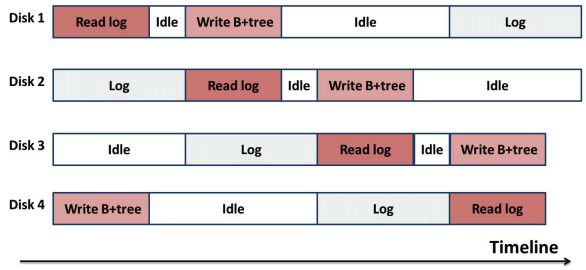


Fig. 6: The ideal *Workload Shifting*

2) ***Acceptable alternation of disk stages****:* Figure 7 describes another situation of *Workload Shifting*.



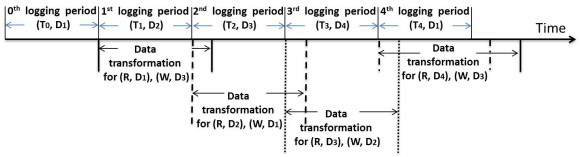
(a) Time diagram of the alternation



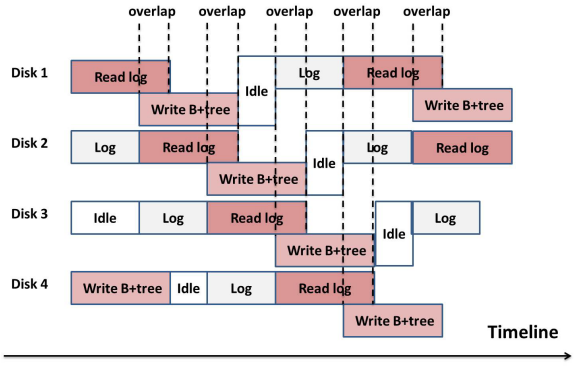
(b) The acceptable alternation of disk stages in *Workload Shifting*

Fig. 7: The acceptable *Workload Shifting*

Although the progress of Log is slower than that of the others from the Figure 7(a) (i.e. the logging period is longer than the data transformation period), data transformation are triggered when the logging rotates to another disk. As shown in Figure 7(b), there is no disk contention.



(a) Time diagram of the alternation



(b) The acceptable alternation of disk stages in *Workload Shifting*

Fig. 8: The bad *Workload Shifting*

3) ***Bad alternation of disk stages****:* From the Figure 8(b), one can see that Read log overlaps with Write B+tree on the same disk, also the overlapping of them leads to disk contention because the progress of Log is always faster than the others that as shown in Figure 8(a) (i.e. the logging period is shorter than the data transformation period). We can get that *Workload Shifting* can not avoid disk contention from Figure 8

Through the above analysis, in the ideal situation like the ideal alternation of disk stages, *Workload Shifting* can avoid not only disk contention between data injection and

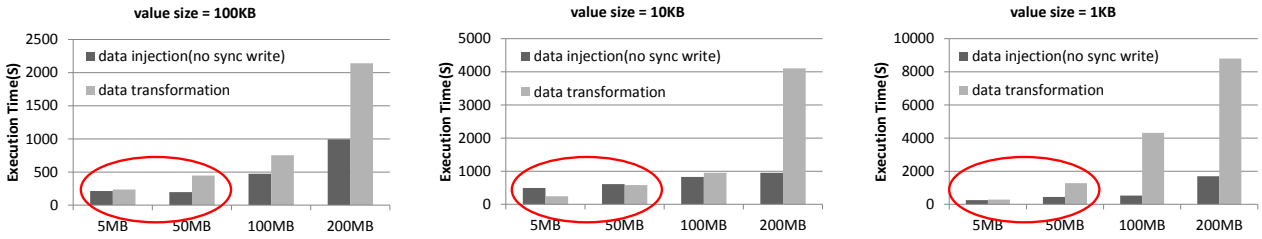


Fig. 9: The execution time of data injection(no sync write) and data transformation

data transformation but also disk contention between data transformation reads I/Os and write I/Os. Actually, in the real situation, there still exists disk contention in *Workload Shifting* such as Bad alternation of disk stages due to the different logging period and data transformation period.

Moreover, it is very difficult to fullfill the requirement of the ideal alternation of disk stages, so we focus on the acceptable alternation of disk stages. In the implementation of the acceptable alternation of disk stages, we use sync write instead of no sync write to prolong the logging period. In our opinion, there are some remarkable advantages of using sync write as follows. The first one is *Increasing reliability*. Using sync write ensures that the previous data will not lost when the power is off as well the method is common in conventional storage systems (e.g. LevelDB, Berkeley DB). Data in memory can be recovered under the power off condition via this method. However, using no sync write obviously leads data in memory to be lost if the power goes off. Second, *reducing energy consumption*. As shown in Figure 7(b), there exists *idle* stage in some disks. During the idle stage, *Workload Shifting* sets the disk to the IDLE state in order to save power. On the contrary, conventional RAID0 array keeps the disks ACTIVE all the time. The third one is *eliminating disk contention* that can speed up not only foreground data injection but also background data transformation. Here, we conduct a simple experiment to validate the problem about no sync write. In the experiment, we mainly focus on the execution time of data injection which uses no sync write and data transformation under different value size(1KB, 10KB, 100KB), also different data volume of data injection and data transformation are applied into the experiment(5MB, 50MB, 100MB, 200MB).

As the Figure 9 shows, the tendency of data injection and data transformation in execution time is different when the value size and log size change, especially when the log size is 5MB and 50MB. Also, when the log size becomes larger, the data injection period is shorter than the data transformation period, hence, there exists disk contention which is similar with *Bad alternation of disk stages*.

IV. Evaluation

A. Evaluation Methodology

1) Baseline for Comparison: We use two systems as the main baseline for comparison. One is LevelDB which is used for the efficiency of data transformation. The other is a conventional log layered over RAID0 (called Log+RAID0) as shown in Figure 1, which is used for comparing the reduction of redundant time resulting from disk contention. So we mainly compare the time gap in two types of experiments. One is the execution time of data injection without data transformation. The other is the execution time of data injection and data transformation. Besides, we compare the energy consumption of *Workload Shifting* with that of Log+RAID0.

2) Evaluation Platform: We evaluate the performance of all the systems using a 6-core server with 16GB of DDR3 RAM. In order to eliminate the impact of memory, we limit memory size to 2GB. A disk array consisting of up to 10 disks is adopted in our evaluation. Table II shows the detailed server hardware and system software configuration information.

3) Workload Generation: We use YCSB [13] to generate key value workloads. The workloads in our evaluation can be categorized into two classes according to the type of key: sequential key workload and random key workload. Since the main design goal of *Workload Shifting* is to avoid disk contention between data injection and data transformation, therefore sequential key workload and random key workload have no read operations, just write operations for use in the main evaluation and sensitivity studies to show how different design parameters may impact the performance of *Workload Shifting*.

TABLE II: The detailed hardware configuration information

CPU Type	Intel R Xeon E5645
#Cores	6 cores@2.4G
#Threads	12 threads
Memory	16GB, DDR3 (Just limit it to 2GB in our experiments)
Disk	**12 disks** one disk for system, others for data **Disk Model** Seagate ST1000NM0011 **Capacity** 1TB **Rotational Speed** 7200RPM **Avg.Seek/Rotational Time** 8.5ms/4.2ms **Sustained Transfer Rate** 150MB/s
OS	CentOS release 6.1 (2.6.32)

B. Main Evaluation

We conduct experiments on a disk array consisting of 4 disks, and use workloads with 16 bytes keys and different value size (the volume of data set is kept as 10GB) to evaluate the performance of *Workload Shifting*. For Log+RAID0 and *Workload Shifting*, the default log size is set to be 5MB.

First, we compare the efficiency of LSM-tree with *Workload Shifting* under random key workload. From the Figure 10, one can see that the throughput of *Workload Shifting* is

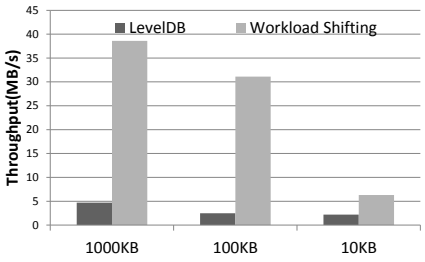


Fig. 10: The throughput of LevelDB and *Workload Shifting* under random key workloads

38.6MB/s, 31.1MB/s and 6.3MB/s respectively under different value size, similarly, the throughput of LevelDB is 4.7MB/s, 2.5MB/s and 2.2MB/s. Statistically, the throughput of *Workload Shifting* is larger than that of LevelDB. The reason is that on one hand, the write amplification caused by the compaction of LSM tree results in lower throughput; on the other hand, there exists disk contention during the LSM tree compaction. Here, we do not present the throughput of LevelDB and *Workload Shifting* under sequential key workload, because the key value pairs are ordered and does not need data transformation, so both of their throughput are the same with each other.

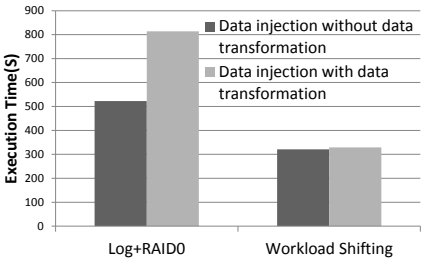


Fig. 11: The impact of data transformation on Log+RAID0 and *Workload Shifting*

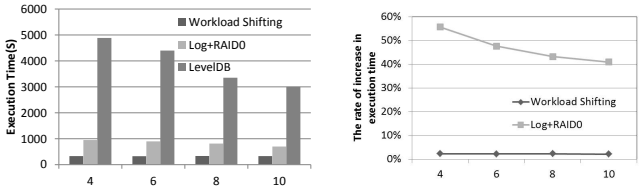
Second, we examine the impact of data transformation on Log+RAID0 and *Workload Shifting*. We use a random workload in which the key size and value size are set as 16 bytes and 100KB respectively. The result is shown in Figure 11. We can get the following two conclusions. For the one, data transformation disrupts the performance of systems due to the disk resources competition with data injection. For instance, the execution time of Log+RAID0 increases from 522s to 813s. For *Workload Shifting*, the execution time increases from 320s to 329s. Statistically, *Workload Shifting* minimizes the overheads caused by disk contention. The execution time of *Workload Shifting* increases by only 2.7%, however, Log+RAID0 increases by 55.7%. The reason is that *Workload Shifting* uses disks separately to separate the foreground data injection workloads from background data transformation workloads and assigns them onto different hard disks to avoid the disk contention. For the other, whether it is under data injection with data transformation or not, the execution time of Log+RAID0 is larger than that of *Workload Shifting*.The reason is that in Log+RAID0, multiple disks are viewed as an entire logical device to do read and write I/O operations, when foreground data injection do the sync write, every disks in RAID0 array will perform this process concurrently, and it will disrupt the performance of foreground data injection. However, due to using the disks separately, there is only one disk to do the sync write at any time in Workload Shifting.

Overall, *Workload Shifting* has a high-efficiency data transformation scheme and eliminate the disk contention between data injection and data transformation.

C. Sensitivity Studies

According to the above analysis and the design of *Workload Shifting*, the following three key factors have great impact on the performance and energy consumption of *Workload Shifting*, named the number of disks, the log size and the value size. The precondition of *Workload Shifting* is that the number of disks must be greater than 3 so that *Workload Shifting* can speed up both data injection and data transformation with minimal interference. In addition, the log size is also important. It determines the starting time and execution time of data transformation, and unreasonable log size may result in extra overheads.

To examine the impacts of the number of disks, the log size and the value size to the *Workload Shifting*, we conduct a series of sensitivity studies about performance and energy consumption of *Workload Shifting*. Here, we mainly focus on the time gap in two types of experiments. one is the execution time of data injection without data transformation. The other is the execution time of data injection and data transformation. From the results of these two experiments, we can get the *the rate of increase in execution time* under different configurations of *LevelDB, Log+RAID0, Workload Shifting*.



(a) The impact of the number of disks on the execution time

(b) The impact of the number of disks on the rate of increase in execution time

Fig. 12: The impact of the number of disks

1) ***The number of disks:*** To examine the impact of the number of disks, we conduct experiments on different numbers of disks (4, 6, 8, 10) in a disk array with 16B keys and 100KB values (the volume of data set is 10GB), and the log size is set to be 5MB.

Figure 12(a) and 12(b) shows the impact of the number of disks on the execution time. From the Figure 12(a), we can know that the execution time of Log+RAID0 and LevelDB decrease as the number of disks increases because the two systems make good use of RAID0's parallelization which can speed up write and read performance, however, the execution time of Workload Shifting keeps unchanged due to using the disks separately. As shown in the Figure 12(b), we can also get the following conclusions. First, the rate of increase in execution time of Log+RAID0 is consistently greater than that of *Workload Shifting*, no matter what the number of disks is. It mainly results from the disk contention between data injection

and data transformation, just like the result of Figure 11. Second, increasing the number of disks can reduce the rate of increase in execution time of Log-RAID0, because more disks means faster read and write performance of RAID0, so that data injection and data transformation can be accelerated to reduce the overheads of disk contention, but the rate of increase in execution time of *Workload Shifting* remains unchanged due to using the disks separately.

2) ***Log size****:* To evaluate *Workload Shifting* with different log size, we conduct experiments on different log size (5MB, 50MB, 100MB, 200MB) in a disk array configured with 4 disks. The key size and value size are set as 16 bytes and 100KB respectively. We keep the volume of data set as 10GB. Here, we do not present the rate of increase in execution time of LevelDB since that execution time and Log size have nothing to do with each other for LevelDB.

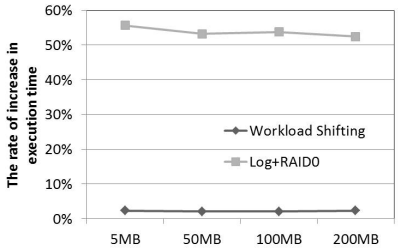


Fig. 13: The impact of log size on the rate of increase in execution time

Figure 13 shows the impact of log size on the rate of increase in execution time. As shown in Figure 13, we can get the following several conclusions. First, the rate of increase in execution time of *Workload Shifting* is smaller than that of Log+RAID0 due to the impact of disk contention. Second, as the log size expands, the rate of increase in execution time of Log+RAID0 and *Workload Shifting* keep unchanged. The reason is that from the Figure 7, one can see that the logging period is larger than the data transformation period, hence the execution time is still determined by the foreground data injection, including Log+RAID0. From another perspective, we can know that no matter what the log size is, the execution time of Workload Shifting and Log+RAID0 remain unchanged (331.3s and 813.6s respectively).

3) ***Value size****:* To examine the sensitivity of LevelDB, Log+RAID0 and *Workload Shifting* on the value size, we conduct experiments on a 4-disk disk array with the value size of 100KB, 50KB and 1KB respectively.

Figure 14 shows the impact of value size on the execution time. From the Figure 14, we can get several observations as follows. First, due to the elimination of disk contention, the execution time of Workload Shifting is smaller than the other two systems. Second, as the value size expands, the rate of increase in execution time of *Workload Shifting* keeps unchanged due to the elimination of disk contention. Third, the rate of increase in execution time of Log+RAID0 decreases as the value size expands. The reason is that within the same data volume, the smaller value size, the larger the number of key value pairs, so it results in more logging period. The gap between data injection period and data transformation period will increase and then the overheads of disk contention in Log+RAID0 will decrease.

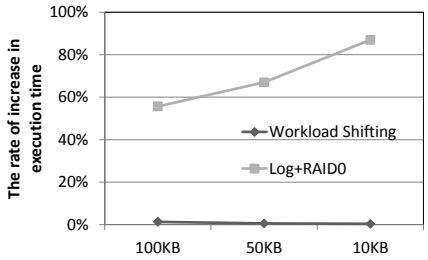
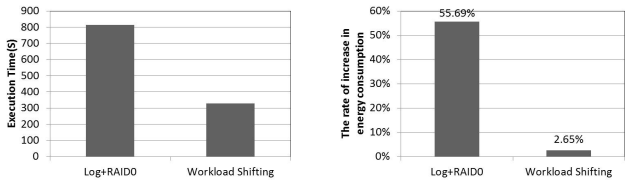


Fig. 14: The impact of value size on the execution time

D. Energy Consumption

In this section, we approximately evaluate the energy consumption of Log+RAID0 and *Workload Shifting* by the execution time.



(a) Execution Time

(b) The impact of data transformation on the rate of increase in energy consumption

Fig. 15: Energy consumption

Figure 15(a) shows the execution time of Log+RAID0 and *Workload Shifting*, as shown in Figure 15(a), the execution time of Log+RAID0 is larger than that of *Workload Shifting*. In order to estimate the energy consumption, we make some assumptions as follows: 1) a single disk has two states, named ACTIVE and IDLE; 2) during the ACTIVE state, the energy consumption of disk is a constant W; 3) during the IDLE state, the energy consumption of disk is 0. Based on our fundamental assumptions, we can estimate the energy consumption of Log+RAID0 and *Workload Shifting* associated with execution time.

Since Log+RAID0 treats multiples disks as an entire logical device for use, during the execution, all of these disks are kept ACTIVE state. So the energy consumption of Log+RAID0 E can be got by

$$E = N * W * T \tag{1}$$

where N denotes the number of disks in disk array, T denotes the execution time of Log+RAID0. For estimating the energy consumption of *Workload Shifting*, it is different from Log+RAID0. The reason is that as shown in Figure 7(b), one can see that there exists IDLE state during the execution, so the energy consumption of *Workload Shifting* meets the following formula:

$$E < N * W * T \tag{2}$$

Obviously, from the Figure 15(a), the execution time of Log+RAID0 is 813s and it is larger than that of *Workload*

Shifting (i.e. 329s), even if their execution time are the same with each other, the energy consumption of *Workload Shifting* is still less than that of Log+RAID0. The reason is that during the execution, Log+RAID0 keeps all the disks ACTIVE. In contrast, some disks are kept IDLE during the execution in *Workload Shifting*. Above all, *Workload Shifting* consumes less energy than Log+RAID0.

In addition, as the Figure 15(b) shows, data transformation makes different impacts on the increase of energy consumption. In Log+RAID0, the execution time is prolonged due to the disk contention that means more energy are consumed. So compared to Log+RAID0 without data transformation, the energy consumption of Log+RAID0 with data transformation increases by 55.69%. While in *Workload Shifting*, it avoids disk contention to keep the execution time essentially unchanged, so the energy consumption of *Workload Shifting* increases less than 2.65% due to the IDLE state existing in some disks, compared to *Workload Shifting* without data transformation.

V. Related Work

Data transformation is a method to improve write, read, scan performance [8] [11]. For instance, Chameleon [8] introduce data transformation into the systems to solve the problem that typical big data systems not only employ one or more extra dedicated disks to reorganize data into another data organization, but also use sequential data transformation which incur extremely long time and significant energy consumption, but Chameleon does not consider disk contention between application writes and data transformation.

Disk contention is increasingly a significant problem, as applications are forced to co-exist on machines and share physical disk resources. Existing solutions to mitigate the effects of disk contention revolve around careful scheduling decisions, either spatial or temporal. For instance, one solution to minimize interference involves careful placement of applications on machines [2] [3]. However, this requires the cloud provider to accurately predict the future I/O patterns of applications. Additionally, placement decisions are usually driven by a wide number of considerations, not just disk I/O patterns; these include data/network locality, bandwidth and CPU usage, migration costs, security concerns, etc. A different solution involves scheduling I/O to maintain the sequentiality of the workload seen by the disk array. Typically, this involves delaying the I/O of other applications while a particular application is accessing the disk array. However, I/O scheduling sacrifices access latency for better throughput, which may not be an acceptable trade-off for many applications.

Workload Shifting is mainly inspired by Gecko [7], which is a novel log-structured design that eliminates read-write contention by chaining together a small number of drives into a single log, effectively separating the tail of the log (where writes are appended) from its body. One one hand, although Gecko focuses on eliminating disk contention that is the same with *Workload Shifting*, the disk contention of Gecko mainly results from application writes and garbage collection [5] [6] which is different from *Workload Shifting*; on the other hand, the function of garbage collection and data transformation are different. Garbage collection turns dirty segments into clean ones by copying live blocks from the dirty segment into the current segment and skipping the rest, and data transformation changes the data organization. In *Workload Shifting*, the data organization is changed from log-structured to B+tree.

VI. Conclusion

In this paper, we propose *Workload Shifting*, which separates the foreground data injection workloads from background data transformation workloads and distribute them onto different hard disks to avoid the disk contention. Specifically, we inject data onto one hard disk for some time and then shift the data injection task to another hard disk. All the disks take turns to be responsible for the data injection task. Meanwhile, background data transformation task is scheduled to handle the just injected data. By this way, the just injected data can be transformed as soon as possible and thus can facilitate the consequent read and scan operations. Furthermore and most importantly, *Workload Shifting* can efficiently speed up the data transformation with minimal interference with foreground data injection workloads.

VII. Acknowledgments

We thank the anonymous reviewers for their helpful feedback. This work is supported by the National Science Foundation of China under grants No.61303056, No.61379042 and No.61202063, National Basic Research Program of China under grant No.2011CB302502, and Huawei Research Program YB2013090048.

References

[1] J. Dean, S. Ghemawat. *Leveldb.* http://leveldb.googlecode.com/svn/trunk/doc/index.html.

[2] GULATI, A., KUMAR, C., AND AHMAD, I. *Storage Workload Characterization and Consolidation in Virtualized Environments*. In Workshop on Virtualization Performance: Analysis, Characterization, and Tools (2009).

[3] GULATI, A., SHANMUGANATHAN, G., AHMAD, I., WALDSPURGER, C., AND UYSAL, M. *Pesto: Online Storage Performance Management in Virtualized Datacenters*. In Proceedings of the ACM Symposium on Cloud Computing (2011), pp. 19-19.

[4] ROSENBLUM, M., AND OUSTERHOUT, J. K. *The Design and Implementation of a Log-Structured File System.* In Proceedings of the thirteenth ACM symposium on Operating systems principles (1992).

[5] SELTZER, M., SMITH, K., BALAKRISHNAN, H., CHANG, J., MCMAINS, S., AND PADMANABHAN, V. *File System Logging Versus Clustering: A Performance Comparison.* In Proceedings of USENIX Annual Technical Conference (1995), pp. 21-21.

[6] MATTHEWS, J., ROSELLI, D., COSTELLO, A., WANG, R., AND ANDERSON, T. *Improving the Performance of Log-Structured File Systems with Adaptive Methods.* In Proceedings of the ACM Symposium on Operating System Principles (1997), pp. 238-251.

[7] Ji-Yong Shin, Mahesh Balakrishnan, Tudor Marian, Hakim Weatherspoon *Gecko: contention-oblivious disk arrays for cloud storage.* In Proceedings of the 11th USENIX conference on File and Storage Technologies (2013).

[8] Fengfeng Pan, Yinliang Yue, Jin Xiong. *Chameleon: a data organization transformation scheme for big data systems.* In Proceedings of the 11th ACM Conference on Computing Frontiers (2014).

[9] PATTERSON, D. A., GIBSON, G., AND KATZ, R. H. *A Case for Redundant Arrays of Inexpensive Disks (RAID).* In Proceedings of the ACM SIGMOD International Conference on Management of Data (1988), pp. 109-116.

[10] H. Amur, D. G. Andersen, M. Kaminsky, and K. Schwan. *Design of a write-optimized data store.* 2013.

[11] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky. *Silt: A memory-efficient, high-performance key-value store.* In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (2011), pp. 1-13.

[12] HANSEN, J., AND JUL, E. Lithium. *Virtual Machine Storage for the Cloud.* In Proceedings of the ACM Symposium on Cloud Computing (2010), pp. 15-26.

[13] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. *Benchmarking cloud serving systems with ycsb.* In Proceedings of the 1st ACM Symposium on Cloud Computing (2010), pp. 143-154.

[14] Y. Yue, L. Tian, H. Jiang, F. Wang, D. Feng, Q. Zhang, and P. Zeng. *Rolo: A rotated logging storage architecture for enterprise data centers.* In Proceedings of IEEE International Conference on Distributed Computing Systems(2010), pp. 293-304.

[15] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil. *The log-structured merge-tree(lsm-tree).* In Acta Informatica(1996), pp. 351-385.

[16] M. Rosenblum and J. K. Ousterhout. *The design and implementation of a log-structured file system.* In Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles(1991), pp. 1-15.

[17] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. *Bigtable: A distributed storage system for structured data.* In Proceedings of the 7th Symposium on Operating Systems Design and Implementation(2006), pp. 205-218.

[18] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. *Pnuts: Yahoo!'s hosted data serving platform.* In Proceedings of the VLDB Endowment(2008), pp. 1277-1288.

[19] R. Escriva, B. Wong, and E. G. Sirer. *Hyperdex: A distributed, searchable key-value store.* In Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication(2012), pp. 25-36.

[20] L. George. *Hbase: the definitive guide.* https://hbase.apache.org/.

[21] A. Lakshman and P. Malik. *Cassandra: A decentralized structured storage system.* In Proceedings of ACM SIGOPS Conference on Operating Systems Review(2010), pp. 35-40.