LESS: Logging Exploiting SnapShot

Hanseung Sung*, Minhwa Jin*§, Mincheol Shin*, Hongchan Roh**, Wongi Choi*, Sanghyun Park*†
*Department of Computer Science, Yonsei University
Seoul, Republic of Korea
{hssung, mhjin, smanioso, cwk1412, sanghyun}@yonsei.ac.kr
**ICT R&D Center, SK Telecom
Seoul, Republic of Korea
hongchan.roh@sk.com

Abstract— **The in-memory key-value store provides a persistence method to ensure data durability. The currently provided methods are either to create a snapshot file of a current dataset or to write the log of the performed command in the log file. However, the snapshot method has a risk of data loss and append only logging method cause a system failure due to an increase in log file size. To prevent excessive AOF file size growth, the in-memory key-value store provides a reconstruction method, but also a performance degradation and excessive memory usage occur. In this paper, we propose a new persistence method for effective memory usage and throughput. The new approach is called Logging Exploiting SnapShot (LESS). LESS is a method that combines the advantages of a snapshot using low memory usage and the benefits of an append only logging method that guarantees data persistence. We implemented LESS on Redis and conducted experiments. A benchmark test demonstrated that the proposed approach reduces the maximum memory usage by 57% and it is 2.7 times faster than the original approach. Overall, the experimental results showed that LESS is effective for Redis.**

Keywords—NoSQL, In-memory Key-Value Store, Redis, Persistence Method, Recovery

I. Introduction

Redis, the most famous key-value store available[1-6], provided two persistence methods to guarantee the data durability from the volatility of DRAM. First, the Redis Database file (RDB) is a method for saving to persistence storage such as HDD or SSD in the form of a compact binary dataset to a point in time. The other method, Append Only File (AOF), logs all commands to update the dataset into an AOF file. For greater durability, Redis mainly uses the AOF method because RDB may not guarantee the data durability of the newly inserted data immediately after RDB file generation. However, as data continue to be inserted into Redis, the AOF file size increases. The bigger the size of the AOF file is, the slower the Redis performance and the recovery time become. Redis supports the AOF Rewrite method to prevent the AOF file size from growing exponentially. The AOF Rewrite method is a reconstruction method that reduces the size of the currently stored AOF file which also results in excessive memory usage during this process. Increasing the memory usage during the AOF Rewrite process degrades the Redis performance.

In this paper, we propose a novel persistence method for Redis called Logging Exploiting SnapShot (LESS) for troubleshooting problems when performing an AOF Rewrite. LESS is a persistence method that exploits the generation of RDB instead of generating the shortest sequence of log records. Furthermore, the log records are written directly to the new AOF file instead of being stored in the AOF Rewrite buffer which causes excessive memory usage. LESS increases the available memory space for storing a dataset as compared to the original approach. Moreover, LESS has been reduced in a blocking state compared to the original persistence methods during a workload, which denotes the high availability of our approach.

II. Background

A. RDB (Snapshot)

The RDB method generates point-in-time snapshots of a dataset at specific intervals. Given that Redis is a single thread-based process, requests received from a client are delayed during generation of RDB for the main process. To avoid a performance degradation from a request delay, Redis creates a child process to generate the RDB using a *fork* system call. The child process generates the RDB utilizing the copy-on-write method[7]. The RDB method has an advantage of smaller file size than the AOF when loading a dataset of the same size. Additionally, the recovery speed of RDB exceeds that of the AOF for the same dataset. However, if a system crash occurs before the RDB file is recreated, all data loaded during that time will be lost. Owing to this problem, the durability of the dataset is not guaranteed after the creation of the RDB. Furthermore, as the dataset grows larger in size, the CPU usage for generating the RDB increases significantly. Under this situation, the request received from a client takes milliseconds or even a full second because of CPU overhead.

B. AOF (Append Only File)

The AOF is a method that writes all log records in an AOF file whenever a key-value pair is inserted, modified, or deleted. When a command modifying the key-value data is requested, it stores the requested key-value pair in the dataset. Next, the log record for the stored key-value pair is stored in the AOF buffer. Subsequently, the log records stored in the AOF buffer are guaranteed to be written to the AOF file using the *fsync* function.

§ This author contributed equality to this work.
† Corresponding author.

The durability of the AOF persistence method surpasses the RDB because the AOF stores a record of all data changes. Accordingly, Redis performs the recovery using an AOF file if such a file and an RDB file exist. However, when using the AOF, Redis is slower than other methods because the AOF method creates a disk I/O while continuously writing log records to the disk. In addition, the AOF file size exceeds the RDB file size for the same dataset.

C. *AOF Rewrite*

Using AOF mode, the size of the AOF file increases linearly because data are continuously inserted into Redis. Increasing the size of AOF file results in performance degradation and excessive resource usage. In addition, Redis takes a long time to recover when using the AOF if the size of the AOF file continues to increase.

To solve these problems of the AOF method, Redis provides an AOF Rewrite method for reconstructing the size of an AOF file by preserving only log records for the final state of the current dataset. The AOF Rewrite method is triggered when the AOF file exceeds the threshold size. The AOF Rewrite operation process is as follows. First, if an AOF Rewrite method is triggered, Redis creates a child process using a *fork* system call. AOF Rewrite also exploits the copy-on-write mechanism. Then, the child process creates a temporary AOF file and generates the shortest sequence of SET log records. During an AOF Rewrite, log records generated by the main process are appended in the current AOF file. The logs are also stored in the AOF Rewrite buffer. If the child process completes the generation of a temporary AOF file, a terminal signal is sent to the main process. After the main process receives a terminal signal, the main process flushes the log records in the AOF Rewrite buffer to the temporary AOF file. Finally, Redis renames the temporary AOF file to the current AOF file and changes the file to write the log records from the current AOF file to the temporary AOF file.

III. Motivation

AOF Rewrite causes two problems, namely, memory overhead and throughput degradation.

- **Memory overhead**: After an AOF Rewrite is triggered, the log record for the newly requested key-value pair is stored in both the AOF buffer and Rewrite buffer until the child process completes the generation of a temporary AOF file. That is, the log records of the same contents are simultaneously written to two buffers. Therefore, the AOF Rewrite method causes an out-of-memory to occur when executed on systems with limited memory capacity.
- **Throughput degradation**: Redis is a single thread-based process. This means that only one command can be processed at a particular time. When the child process completes the creation of the temporary AOF file, the main process appends all log records stored in the Rewrite buffer to a temporary AOF file. If a new key-value pair insertion is requested from the client while the main process is performing the flush operation, the request waits until the operation of writing the contents of the Rewrite buffer to the temporary file is completed. A delay in requests leads to a decline of the overall throughput.

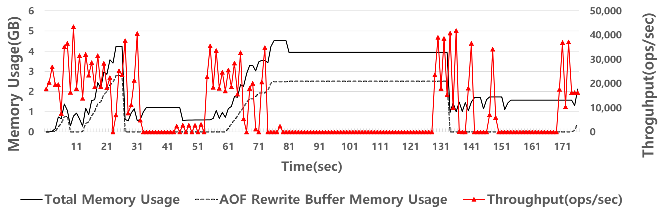

Fig. 1. An experiment with workloads to conduct an AOF Rewrite. The x-axis represents the flow of time during the workload, and the y-axis represents the memory usage and throughput over time.

We experimented with using the Memtier-benchmark[8] to verify memory overhead and throughput degradation. An experiment was conducted to simulate the workload of hot data that are frequently updated by users. This is because there is no variation in the size of the data, but the effect of the AOF file can be identified. A workload composed of 100,000 SET commands and 900,000 duplicated SET commands was used. The key and value size applied in the requests correspond to 16B and 10KB, respectively.

The experiment result is shown in Figure 1. Figure 1 indicates that the memory usage increases sharply and the throughput is zero during the workload. The AOF Rewrite occurred four times during the following elapse-time durations: 3-8s, 13-26s, 60-133s, and 175-177s. When the AOF Rewrite occurs in Redis, the amount of AOF Rewrite buffer memory usage is increased. Similarly, the overall memory usage increased in proportion with the AOF Rewrite buffer memory usage. Moreover, the throughput was declined because Redis incurs CPU overhead during the generation of command logs when inserting key-value pairs into the dataset. During a certain period of using the AOF Rewrite buffer, no operations were conducted because the main process in Redis has to write log records in the AOF Rewrite buffer to a temporary AOF file.

IV. Design

A. *LESS : Logging Exploiting SnapShot*

Our approach was designed to reduce the memory usage and improve the performance of Redis. The LESS method creates an RDB file when Redis begins to conduct an AOF Rewrite, as opposed to creating log records by inserting key-value pairs. Algorithm 1 describes the overall process of LESS.

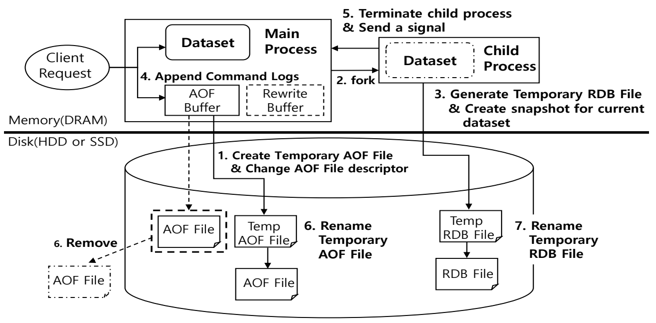
Client Request
Dataset
Main Process
4. Append Command Logs
AOF Buffer
Rewrite Buffer
5. Terminate child process & Send a signal
2. fork
Dataset
Child Process
3. Generate Temporary RDB File & Create snapshot for current dataset
Memory(DRAM)
Disk(HDD or SSD)
1. Create Temporary AOF File & Change AOF File descriptor
AOF File
Temp AOF File
6. Rename Temporary AOF File
6. Remove
AOF File
AOF File
Temp RDB File
7. Rename Temporary RDB File
RDB File

Fig. 2. Operation procedure after LESS is triggered

Algorithm 1 LESS(AOF_Current_Size, AOF_LESS_Min_Size, TempRDB, TempAOF, rdbSaveInfo)

```
     Input:
     AOF_Current_Size: current AOF file size
     AOF_LESS_Min_Size: Minimum AOF file size of starting LESS Method
     TempRDB: Filename of temporary RDB
     TempAOF: Filename of temporary AOF
     rdbSaveInfo : Metadata to the RDB file
     Output: none

     /* Start LESS method */
1    if AOF_Current_Size > AOF_LESS_Min_Size then
         /* Create Temp AOF file */
2        TempAOF_fd ← createFile(TempAOF)
         /* Change File Descriptor to Temp AOF file, command logs are stored to
         Temp AOF */
3        AOF_fd ← TempAOF_fd
         /* Create Child Process using fork system call */
4        childpid ← fork()
5        if childpid == 0 then /* Child process */
             /* Generate Temp RDB file*/
6            retval = rdbGenerate(TempRDB, rdbSaveInfo)
7            if retval == OK then
                 /* Send terminate signal to parent process */
8                    exitCode ← OK
9                    exitFromChild(exitCode)
10       else /* Parent process */
         /* This procedure everything directly called here will be called 10 times per
         second */
11           ServerCron :
12               if TerminalChild() == True then
13                   exitCode ← readexitCode()
14                   if exitCode == OK then
                 /* If success generating RDB file, rename AOF & RDB file */
15                       renameFile(TempAOF, AOF_filename)
16                       renameFile(TempRDB, RDB_filename)
17           End Procedure
18   End
```

LESS operates the same as the AOF method until it is triggered. LESS is triggered when the AOF file size exceeds the threshold. The detailed procedure for LESS is as follows.

In this case, as shown in Figure 2, the main process creates a temporary AOF file and changes the AOF file descriptor to this temporary AOF file. The child process is then forked and a snapshot file is created for the current dataset. While the child process generates an RDB, the main process calls a *fsync* function to ensure that log records in the AOF buffer are written to the temporary AOF file that the AOF file descriptor is pointing to. Requests newly received from the client during LESS operation are processed through the main process.

After the creation of an RDB file for the child process, the procedure is as follows. First, the child process sends a terminal signal to the main process. Then, the main process renames the name of the temporary AOF file to the name of the current AOF file and subsequently changes the name of the temporary RDB file to the name of the current RDB file. When the LESS operation is complete, the AOF and RDB files are stored separately on a disk.

There are two main differences between LESS and AOF Rewrite. First, LESS does not use a Rewrite buffer, which is used in the AOF Rewrite. It writes the log record directly in the AOF file. Thus, the memory usage of LESS is smaller than that of AOF Rewrite. Second, the AOF Rewrite method generates an AOF file, but with the LESS method, the child process generates a serialized RDB file as opposed to an AOF file. Therefore, the disk I/O of the LESS method is relatively lighter than the AOF Rewrite. For this reason, the proposed approach increases the throughput compared to the AOF Rewrite method.

LESS does not use Rewrite buffer and does not perform an append operation in the main process. Because a Rewrite buffer is not used, the memory usage does not sharply increase with LESS. As a result, LESS is safe from an out-of-memory occurrence when compared to AOF Rewrite. Furthermore, with respect to the same memory capacity, our approach increases the available memory space to store the dataset when compared to the original persistence methods. In addition, LESS does not conduct a heavy disk I/O caused by the merging operation performed by the AOF. Our suggestion solves the heavy disk I/O problem by storing the files separately rather than merging contents of Rewrite buffer and temporary AOF file.

Therefore, the proposed approach has an advantage with regard to memory usage and avoiding a heavy disk I/O because real-time processing applications using Redis receives a quick and continuous response, thereby denoting the high availability of the proposed approach.

B. *LESS Recovery Mechanism*

The recovery mechanism of LESS for cases which a failure occurs is described as follows. If a system crash occurs between the creation of a temporary AOF file and temporary RDB file, the existing files on disk are AOF, Temporary AOF, and RDB files. In this case, the AOF and Temporary AOF guarantee the durability of the data after the creation of the RDB file. Therefore, LESS conducts a recovery using the RDB file to reorganize the dataset and then reads the AOF and Temporary AOF files to replay the command logs. If a crash occurs during the generation of an RDB file, the existing files on disk are AOF, Temporary AOF, RDB, and Temporary RDB files. As in the previous case, LESS conducts a recovery using RDB, AOF, and Temporary AOF files. If a crash occurs after the Temporary AOF file is renamed, the existing files on disk are AOF, RDB, and Temporary RDB files. In this case, the Temporary RDB and AOF files guarantee the durability of the data. Therefore, LESS conducts a recovery using the Temporary RDB and AOF files. Finally, when LESS completes its process, the existing files on disk are AOF, and RDB files. In this case, LESS conducts a recovery using the RDB file and then reads the AOF file to replay the command logs.

V. EXPERIMENTAL EVALUATION

A. *Experiment Setting*

We conducted all experiments by comparing the AOF to LESS in terms of memory usage and throughput of the workload. RDB only mode is excluded from the experiment because RDB only mode does not guarantee the durability of the dataset. The hardware and software configurations for experiments are shown in Table 1.

TABLE I. EXPERIMENT SETTING

Hardware Setting	
CPU	Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz 10cores
RAM	DDR3 64 GB
Disk(SSD)	Crucial_CT250MX200SSD1 250 GB * 3
Software Setting	
OS	Cent OS 7.3.1611 (Core)
Linux Kernel Version	3.10.0-514.26.2.el7.x86_64
Redis Version	4.0.10
AOF Option	Default(everysec)
Max memory Option	30 GB
Memtier benchmark version	1.2.13

B. *Experiments Comparing Our Approach to Original Approach Using the Same Workload*

This experiment compared the memory usage and throughput with the LESS method according to the AOF mode (Rewrite–On) processed for the same workload. The workload used in this experiments is the same as that described in Section 3.

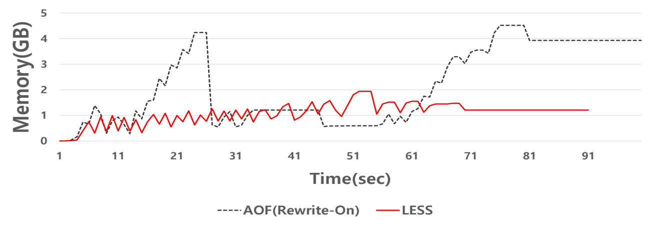

Fig. 3. Experimental results using the same workload in terms of memory usage.

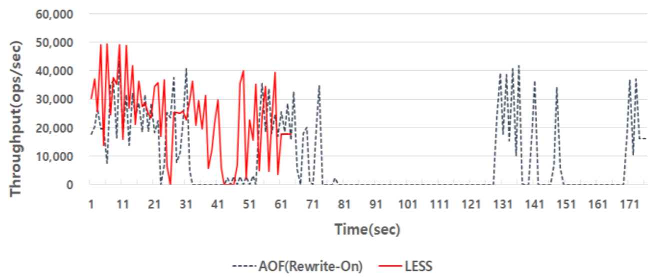

Fig. 4. Experimental results using the same workload in terms of throughput.

Figure 4 shows that LESS completed the workload in 64s, whereas AOF mode completed the workload in 177s. The throughput of the LESS is 2.7 times higher than that of the AOF Rewrite. Compared to the AOF method, our approach has no time during which the use of the memory increases rapidly, as shown in Figure 3. Given that our approach writes log records to a temporary AOF file instead of storing the log records in the AOF Rewrite buffer; Our approach does not use memory for the AOF Rewrite buffer. AOF mode increases the memory usage dramatically because of the use of this buffer. The results show that the original method can cause an out-of-memory problem to occur owing to the use of an AOF Rewrite buffer.

Given that our approach also incurs CPU overhead during the generation of an RDB file, an interval of degradation of the throughput occurs, as shown in Figure 4. Nevertheless, our approach does not conduct a heavy disk I/O during the flushing of log records in the AOF Rewrite buffer.

As a result, our method improves the throughput over the traditional method. There were fewer cases in which the throughput was reduced to zero. Even when the throughput was reduced to zero, there were almost no cases in which the throughput continued at this level.

VI. CONCLUSTION

In this paper, we proposed a new persistence method of Redis, for effective memory usage and throughput, which is called LESS. It exploits the advantage of generating RDB as opposed to AOF Rewrite. We conducted experiments comparing LESS to a conventional method using the Memtier-benchmark. The results show that the throughput of LESS is 2.7 times faster than an AOF Rewrite. The maximum memory usage was reduced by up to 57% compared to an AOF Rewrite. Since we implemented LESS directly in Redis code, LESS can be immediately applied to practical areas by using Redis.

Our method has the advantage of being able to efficiently use memory from an out-of-memory occurrence in systems that provide limited memory usage. Consequently, LESS outperforms the original method in terms of memory usage and throughput for the Redis performance.

ACKNOWLEDGMENT

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the SW Starlab support program (IITP-2017-0-00477) supervised by the IITP (Institute for Information & communications Technology Promotion).

REFERENCES

[1] Redis. https://redis.io/

[2] Memcached. https://memcached.org/

[3] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, “Mica: A holistic approach to fast in-memory key-value storage,” in 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). Seattle, WA: USENIX Association, pp. 429–444, April 2014.

[4] J. Ousterhout, A. Gopalan, et al. “The RAMCloud Storage System,” ACM Trans. Comput. Syst., 33(3):7:1–7:55, August 2015.

[5] Repcached. http://repcached.lab.klab.org/

[6] Ignite. https://ignite.apache.org/

[7] Copy-on-Write. https://en.wikipedia.org/wiki/Copy-on-write

[8] Memtier benchmark. https://redislabs.com/blog/memtier_benchmark-a-high-throughput-benchmarking-tool-for-redis-memcached/

[9] M. Xu, X. Xu, J. Xu, Y. Ren, H. Zhang, and N. Zheng, “A Forensic Analysis Method for Redis Database based on RDB and AOF File,” in Journal of Computers, Vol 9, No 11, 2538-2544, November 2014.

[10] X. Bao, L. Liu, N. Xiao, Y. Lu, and W. Cao. “Persistence and Recovery for In-Memory NoSQL Services: A Measurement Study” IEEE International Conference on Web Services (ICWS) pp. 530-537, July 2016.