On The Implementation of ZFS (Zettabyte File System) Storage System

Eko D. Widianto, Agung B. Prasetijo, and Ahmad Ghufroni
Department of Computer Engineering, Faculty of Engineering – Diponegoro University
Jl. Prof. Soedarto, SH, Tembalang, Semarang, Indonesia
E-mail: didik@live.undip.ac.id; agungprasetijo@ce.undip.ac.id; ahmad@ghufroni.com

Abstract – Digital data storage is very critical in computer systems. Storage devices used to store data may at any time suffer from damage caused by its lifetime, resource failure or factory defects. Such damage may lead to loss of important data. The risk of data loss in the event of device damage can be minimized by building a storage system that supports redundancy. The design of storage based on ZFS (Zettabyte File System) aims at building a storage system that supports redundancy and data integrity without requiring additional RAID controllers. When the system fails on one of its hard drive, the stored data remains secure and data integrity is kept assured. In addition to providing redundancy, the ZFS-based storage system also supports data compression for savings on storage space. The results show that the ZFS with LZ4 compression has the highest read and write speed. For real benchmark, there is no significant difference in reading speed for a variety of different variables, whereas a significant increase in speed occurs when writing compressible files on the ZFS system with compression configuration.

Keywords – Data Compression; Data Integrity; Proxmox VE; Redundancy; ZFS

I. INTRODUCTION

Data storage is one of the critical resources in computer systems. Hard drives as a means of storing data may fail at any time due to power failure or factory defects. The damage may lead to the loss of important data. To minimize the risk of such data loss, a storage system that supports both redundancy and integrity can be implemented. This paper presents the building of a storage system based on ZFS (Zettabyte Storage System).

The ZFS storage system is redundant so that if one disk fails, the stored data will not be lost and data integrity is kept ensured. This system works like RAID (Redundant Array of Independent Drives) but this is merely based on software, thus no additional RAID Controller devices required on the server's side.

II. PREVIOUS WORKS DONE

Literature [1] compares the performance between the Solaris ZFS file system with Red Hat Enterprise Linux using EXT3 file system. The research was conducted with two identical hardware and standard configuration. Tests were carried out synthetically using IOZone software. The parameter used in this research is the read and write speed on each file system. The test results for read process throughput on the size of I/O less than 32 KB is as follows: EXT3 is 500 MB/s faster than ZFS, while on the size of the I/O more than 32 KB, ZFS is 200MB/s faster than EXT3. The throughput of write process in ZFS is as good as EXT3 for the size of I/O less than 32 KB, whilst, for the size of I/O larger than 32 KB ZFS is 150 MB/s faster compared to EXT3.

Literature [2] explored the performance, capacity and integrity of ZFS raidz. The tests for its read and write speed is as follows: in mirror mode using two 4TB hard drive, ZFS has 488 MB/s for read speed and 106MB/s for write speed. When using RAIDZ mode on 3 hard drives with 4TB capacity, the results is 619MB/s of read speed and 225 MB/s of write speed.

Literature [3] tested the speed of ZFS when data is resilvered in the event of hard drive failure and is replaced with another hard drive. Data resilvering is the process of copying data and calculating of its parity between the hard drive of interest with another hard drive in a RAID group when such a hard drive has been replaced. This study used two hard drives with 1TB capacity for mirroring mode and 3 hard drives with 1TB capacity for RAIDZ mode. Data resilvering is set when the capacity of the pool is used by 25% and 50% of total capacity. The result in mirroring mode with 25% capacity with resilvering time is 37 minutes, while on 50%, the resilvering time is 75 minutes. RAIDZ mode takes 39 and 82 minutes with the respective capacity of 25% and 50%.

File system is a method used by the operating system to manage files on a disk or partition [4]. ZFS uses the concept of storage pools to manage physical storage [5]. ZFS can apply checksum mechanism to check errors when reading data from storage. ZFS can also apply data compression algorithms with choices of LZ4, LZJB, or GZIP [6]. RAID is one method to make redundancy and improve performance of storage [7]. The RAID levels of RAID 0, RAID 1, RAID 5, RAID 60, RAID 00, RAID10, RAID 50, RAID 60 are commonly used in storage systems [8]. Proxmox VE (Virtual Environment) is an open source operating system for virtualizing environments based on the Debian Linux distribution [9]. Proxmox has features such as providing multinode cluster, providing HA, and supports the Ceph storage type such as, NFS, ZFS, Gluster and iSCSI [10].

III. SYSTEM DESIGN

The design of this system requires a server, a client and a switch to connect between the client and the server. Figure 1 shows the design of the system.

978-1-5090-0890-2/16/$31.00 ©2016 IEEE

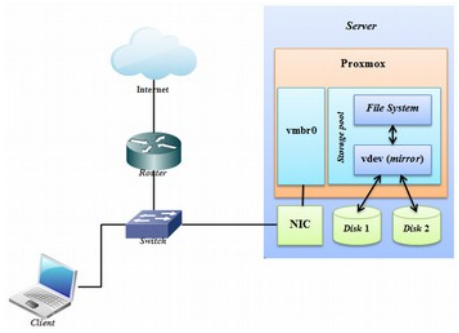

Figure 1 Design of the system

The server computer has a gigabit network ethernet card connected to the switch. Proxmox 4.1 is installed on a server inside a hard drive with a capacity of 120 GB. Storage pool is a collection of storage devices and aggregated so that they can share storage spaces. Storage pool in this study included a vdev. Vdev is a virtual device that can be formed from a single disk or many hard drives. If the system has many hard drives on one vdev, it can be configured to mirror mode, RAIDZ, RAIDZ-2 or RAID-Z3 to provide data redundancy. In this research, two hard disks are configured in mirror. The ZFS file system can be composed of one or several vdev. In this research, the file system in the storage pool is composed of only 1 vdev. The Proxmox connects to the network using a virtual interface with an IP address of 10.0.1.2 to vmbr0 using gigabit network ethernet card. The client computer is a laptop with Windows 7 Professional operating system installed on a 128 GB SSD and a gigabit ethernet card connected to the switch. VLAN of the switch is set to inactive. The router having an IP address of 10.0.1.1 is used as a server and client's gateway.

IV. System Benchmark and Analysis

Testing performance of the system was conducted in two ways: synthetic benchmark using bonnie ++ software and real benchmark by transferring files between the client and the server using FTP protocol. The Mdadm-based system is used for comparison purposes. The ZFS system is separately configured on a device with identical specifications and operating system. The testing was conducted with three repetitions. When every single test has finished, the server is restarted, so there are no data cached in the memory and disk cache. In ZFS system data collection was conducted with 5 different variables: checksum off, compression off, LZ4 compression, LZJB compression and GZIP compression. In Mdadm there are no variables as such in the ZFS. In this ZFS the checksum feature is enabled employing Fletcher4 algorithms.

A. Synthetic Benchmark

Synthetic benchmark was conducted by running command on the computer command prompt `# bonnie++ -u root -r 3917 -s 16384 -d /raid1 -f -b -n 1`. Command `-u root` means the user runs this test with root privilege. `-r 3917` means the total RAM installed on the server (which is 3917 MB), total RAM can be obtained using `fdisk -m` command. The command `-s 16384` shows the size of file generated in this test (16,384 MB (16 GB)). Command `-d / raid1` is used to determine the location of the file system being test (i.e., `/raid1`). Command `-b` is used to run the sync write. Command `-n` is the number of iteration. Data taken includes the read speed, write speed and CPU use on the server. Figure 2 shows a graph of average read speed result using Bonnie++ with the CPU use.

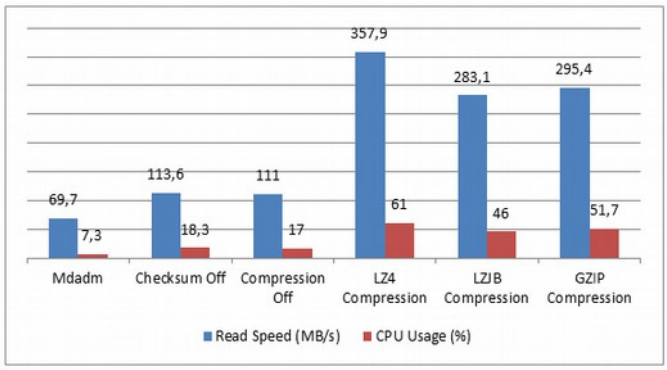

Figure 2 Graph of read speed using Bonnie++

Read speed performance benchmark show the highest result owned by ZFS system with compression LZ4 with the result of 357.9 MB/s. The least read speed performance has been reached by Mdadm system with the result of 69.7 MB/s. The highest CPU use is 61% performed by the ZFS system with LZ4 compression, while the least CPU use is 7.3% performed by the Mdadm system. The ZFS system with checksum on has a difference of 2.6 MB/s in read speed and 1.3% in CPU use compared with ZFS system with checksum off. Figure 3 shows graphs of average write speed result using Bonnie++ compared to CPU use.

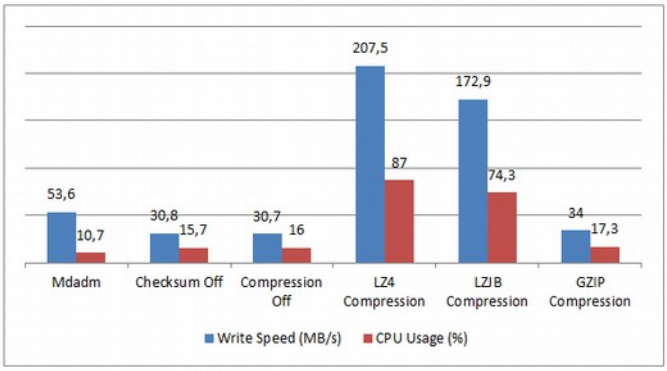

Figure 3 Graph of write speed using Bonnie++

The write speed performance benchmark shows the highest result performed by ZFS system with LZ4 compression (i.e., 207.5 MB/s). The least write speed performance was performed by ZFS system with LZ4 compression (i.e., 30.7 MB/s). The highest CPU use is 87% performed by ZFS system with LZ4 compression, while the lowest CPU use (10.7%) was performed by Mdadm system. The ZFS system with checksum

on has a difference of 0.1 MB/s in write speed and 0.3% in CPU use compared to the ZFS system with checksum off.

B. Real Benchmark

Real benchmark was conducted using ProFTPd software installed on the server and CuteFTP installed on the client. There are two types of files transferred: compressible files (with extension text) and incompressible files (with extension mp4). Compressible file transferred in size of 100 MB with an amount of 160 pieces while incompressible file transferred with the size of 110 MB 160 pieces. Data retrieved includes read and write speeds, CPU use and disk space use. The read speed measurement of data retrieval is conducted by transferring the file from the client to the server while the write speed of data retrieval is conducted by transferring the file from the server to the client. The CPU utilization was retrieved using RRDtool on the Proxmox web management. Figure 4 shows graphs of average read speed with CPU usage result for compressible file.

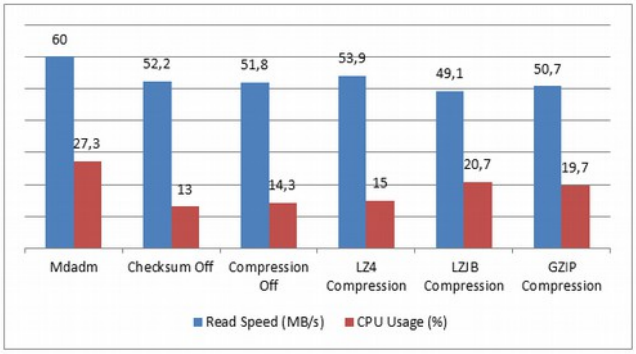

Figure 4 Graph of read speed for compressible file

The read speed performance benchmark for compressible file shows the highest result performed by Mdadm system (60 MB/s). The least read speed performance was performed by ZFS system with LZJB compression (49.1 MB/s). The highest CPU use is 27.3% performed by Mdadm system, while the lowest CPU use is 13% performed by ZFS system with checksum off. The ZFS system with checksum on has a difference 0.4 MB/s in read speed and 1.3% in CPU use compared to the ZFS system with checksum off. Figure 5 shows graphs of average write speed with CPU use for compressible file.

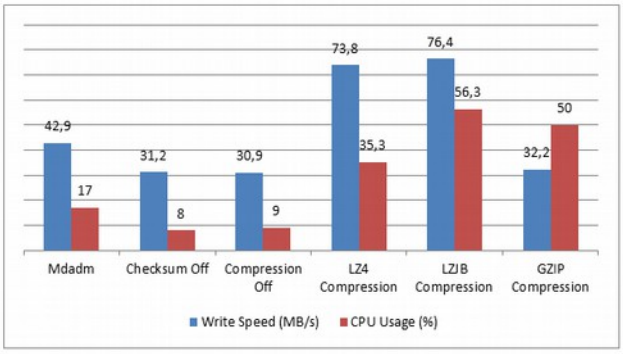

Figure 5 Graph of write speed for compressible file

The write speed performance benchmark for compressible file shows the highest result performed by ZFS system with LZJB compression (76.4 MB/s). The least write speed performance is with ZFS system with compression off (30.9 MB/s). The highest CPU use is 56.3% performed by ZFS system with LZJB compression, while the lowest CPU use is 8% with ZFS system with checksum off. The ZFS system with checksum on has a difference of 0.7 MB/s in write speed and 1% in CPU use compared to the ZFS system with checksum off. The graphs also show the GZIP compression consumes high computing resources (write speed is 32.2 MB/s, but uses 50% of CPU). Compression LZ4 requires a considerably low resource (write speed 73.8 MB/s but only uses 35.3% of CPU).

Figure 6 shows the graphs of storage space used containing compressible file with the amount of 160 pieces and the capacity of each file is 100MB. Figure 7 shows graphs of compression ratio on each variable.

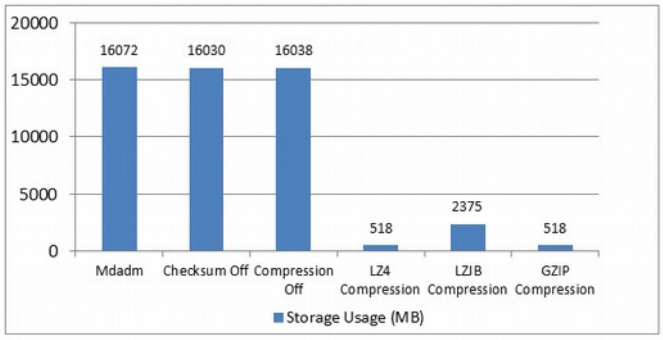

Figure 6 Graph of storage space used for compressible file

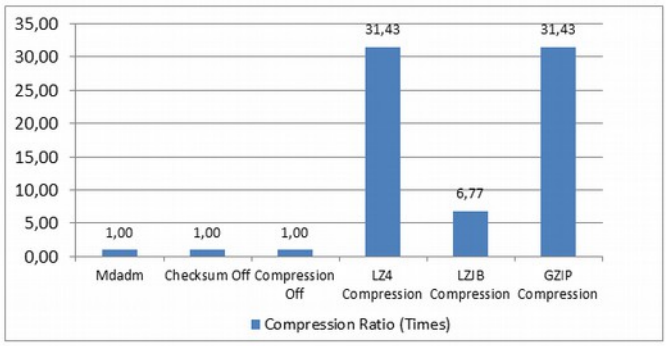

Figure 7 Graph of compression ratio for compressible file

Systems with different variables require different storage space even though the same amount of data is being stored. Data stored on each system for compressible files has the same types and sizes with a total size of 16 GB with text extension. ZFS compression algorithms have many effects for the use of storage space for storing compressible files of the same size. LZ4 and GZIP compression algorithm requires 518 MB of storage space with compression ratio of 31.43 times. The LZJB compression algorithm requires 2,375 MB of storage space with a compression ratio of 6.77 times. The Mdadm system requires 16,072 MB of storage space. The ZFS with checksum off requires 16,038 MB and ZFS with

checksum on requires 16,030 MB of storage space. Between the ZFS system with checksum off and ZFS system with checksum on, there are differences of only 8 MB, which means there are hash values of the data on system with checksum on.

The following is the real benchmark with incompressible file. Figure 8 shows graphs of average read speed with CPU use for the incompressible file.

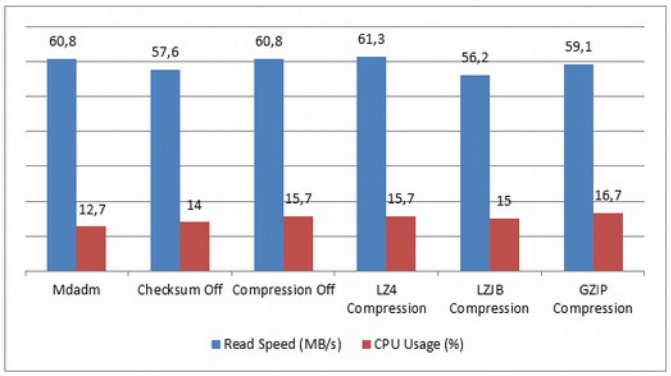

Figure 8 Graph of read speed for incompressible file

The highest read speed performance benchmark for incompressible file is performed by ZFS system with LZ4 compression (61.3 MB/s). The least read speed performance is performed by ZFS system with LZJB compression LZJB (56.2 MB/s). The highest CPU use is 16.7% performed by ZFS system with GZIP compression, while the lowest CPU use is 12.7% performed by Mdadm system. The ZFS system with checksum on has a difference of 3.2 MB/s in read speed and 1.7% in CPU use compared to the ZFS system with checksum off. Figure 9 shows graphs of average write speed with CPU use for incompressible file.

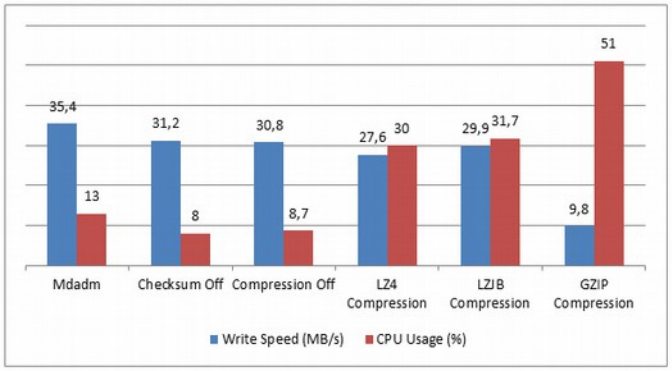

Figure 9 Graph of write speed for incompressible file

The highest write speed performance benchmark for incompressible file is performed by Mdadm system (35.4 MB/s). The least write speed performance is performed by ZFS system with GZIP compression (9.8 MB/s). The highest CPU use is 51% performed by ZFS system with GZIP compression, while the lowest CPU use is 8% performed by ZFS system with checksum off. The ZFS system with checksum on has a difference of 0.7 MB/s in write speed and 0.7% in CPU use compared with the ZFS system with checksum off. The graphs also show the GZIP compression requires high computing resources, even with write speed of 9.8 MB/s, but it consumes 51% of CPU.

In Figure 10, graphs of storage space used containing compressible file with the amount of 160 pieces and the capacity of each file is 110MB are presented. Figure 11 shows graphs of compression ratio on each variable.

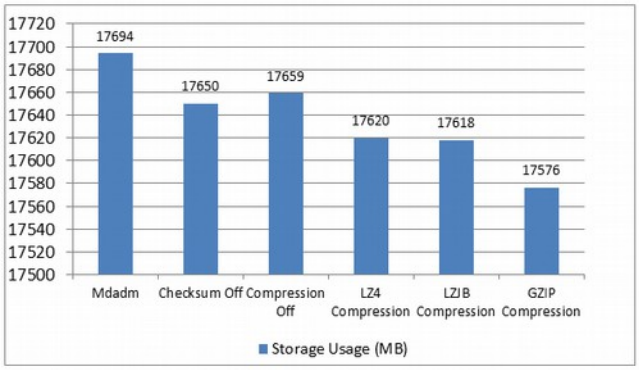

Figure 10 Graph of storage space used for incompressible file

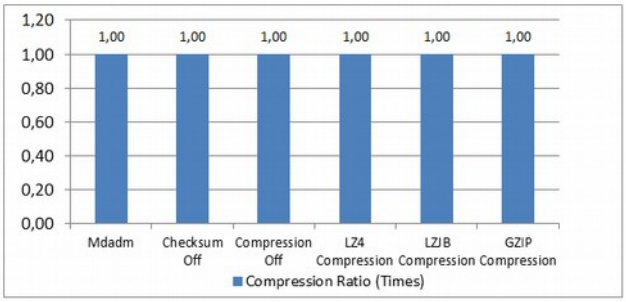

Figure 11 Graph of compression ratio for compressible file

Compression algorithms on ZFS does not significantly affect the use of storage space for incompressible files shown on the compression ratio equal to any different compression algorithms. Difference in the use of storage space is also not significant. The LZ4 compression only saves space of 39 MB compared with no compression and the LZJB compression saves space of 41 MB compared with no compression. Whilst, the LZJB compression saves space 83 MB compared with no compression. Between ZFS system with checksum off and ZFS system with checksum on, it is shown that both have differences in the use of storage space 9 MB. This means the space is used to store data checksum.

C. Data Resilvering

Data resilvering in this test is aimed at finding the required time the system rebuilding the RAID after a failure occurs on one hard drive. This test is simulated by removing one hard drive, removed the partitions and reinstall hard drive to the server. This test was conducted three times. Collecting data for the Mdadm system is conducted by using the capacity of 40 GB and 80 GB. Collecting data for the ZFS system conducted by using the capacity of 40 GB, 80 GB and 120 GB. Figure 12 shows the graphs of average time to perform data resilvering.

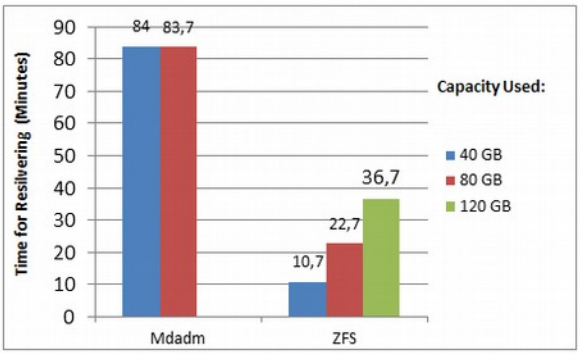

Figure 12 Graph time of data resilver

The time required to conduct data resilvering on the Mdadm system with capacity of 40 GB and 80 GB have no significant difference (84 minutes and 83.7 minutes) as data resilvering process is conducted on all blocks on the disk in a RAID member. In contrast, the significant difference can be seen in the ZFS system. The ZFS system takes 10.7 minutes to resilvering 40 GB of data, 22.7 minutes for 80 GB of data, and takes 36.3 minutes to resilvering 120 GB of data. It can be concluded that the ZFS system performs data resilvering only for blocks containing the data.

D. Integrity Check

Data integrity check is performed to ensure that the ZFS system can actually provide data integrity in the event of failure on one hard drive. Checks are conducted through data checksum using SHA256 algorithm on the file with iso extension. These checks are conducted 3 times; the first check is conducted when the system is at normal running. The check result is shown in Figure 13.

```
root@server:~# zpool status
  pool: raid1
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        raid1       ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            sdb     ONLINE       0     0     0
            sdc     ONLINE       0     0     0

errors: No known data errors
root@server:~# cd /raid1/
root@server:/raid1# sha256sum 1.iso
7d88431c4783640aa97e83a2ef21bdfd697a2c76dcd0ea6ff4ca927459f08a2b  1.iso
root@server:/raid1#
```

Figure 13 Data integrity check when the system is normally running

The result of this check shows that the hash value obtained is `7d88431c4783640aa97e83a2ef21bdfd697a2c76dcd0ea6ff4ca927459f08a2b`. The second check is performed when there is one disk fails. This check is simulated by removing one of the hard drives so that the system is degraded. The result of data integrity checking is shown in Figure 14.

The hash value obtained when the system is degraded is `7d88431c4783640aa97e83a2ef21bdfd697a2c76dcd0ea6ff4ca927459f08a2b`. As the hash value has the same value to the first check, the system integrity is not lost. Although a failure occurs on one hard drive data integrity still ensured. The third check is aimed at determining whether the data is still valid after data resilvering process is completed. The result of data integrity check is shown in Figure 15.

```
root@server:~# zpool status
  pool: raid1
 state: DEGRADED
status: One or more devices could not be used because the label is missing or
        invalid.  Sufficient replicas exist for the pool to continue
        functioning in a degraded state.
action: Replace the device using 'zpool replace'.
   see: http://zfsonlinux.org/msg/ZFS-8000-4J
  scan: none requested
config:

        NAME                    STATE     READ WRITE CKSUM
        raid1                   DEGRADED     0     0     0
          mirror-0              DEGRADED     0     0     0
            sdb                 ONLINE       0     0     0
            7057224403872404870 UNAVAIL      0     0     0  was /dev/sdc1

errors: No known data errors
root@server:~# sha256sum /raid1/1.iso
7d88431c4783640aa97e83a2ef21bdfd697a2c76dcd0ea6ff4ca927459f08a2b  /raid1/1.iso
root@server:~#
```

Figure 14 Data integrity check when system degraded

```
root@server:~# zpool status
  pool: raid1
 state: ONLINE
  scan: resilvered 40.9G in 0h11m with 0 errors on Sat Aug  6 21:19:47 2016
config:

        NAME        STATE     READ WRITE CKSUM
        raid1       ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            sdb     ONLINE       0     0     0
            sdc     ONLINE       0     0     0

errors: No known data errors
root@server:~# sha256sum /raid1/1.iso
7d88431c4783640aa97e83a2ef21bdfd697a2c76dcd0ea6ff4ca927459f08a2b  /raid1/1.iso
root@server:~#
```

Figure 15 Data integrity check after system back to normal

This third check of checksum matches with the hash value `7d88431c4783640aa97e83a2ef21bdfd697a2c76dcd0ea6ff4ca927459f08a2b`. The hash value obtained at this third check is the same as the first and second checks. Hence, results of data integrity checks on the ZFS storage systems indicate that the storage system in this research maintains the integrity of data even when there is a failure on one hard drive.

V. CONCLUSIONS

Synthetic benchmark shows that the ZFS with LZ4 compression has the highest read and write speeds in the amount of 357.9 MB/s and 207.5 MB/s. For real benchmark, there is no significant difference in reading speed for a variety of different variables, whereas a significant increase in speed occurs when writing compressible files on the ZFS system with compression configuration. The GZIP Compression uses most of the CPU resources and makes the writing speed very slow for incompressible files. GZIP and LZ4 compression algorithms generate a compression ratio of 31.43 times, while LZJB algorithm generates a compression ratio of 6.77 times for compressible files. In incompressible file, the compression algorithm does not affect the compression ratio of the stored files. Differences in the use of storage space between checksums on and checksum off show that checksum value also needs disk space. The selection of compression algorithm for incompressible file does not affect compression ratio. The time for data resilvering in ZFS system varies depending on the amount of data stored.

REFERENCES

[1] "Solaris ZFS and Red Hat Enterprise Linux EXT3 File System Performance," Sun Microsystems White Paper, June 2007.

[2] "ZFS Raidz Performance, Capacity and Integrity," 01 April 2016. [Online]. Available: https://calomel.org/zfs_raid_speed_capacity.html. [Accessed 21 July 2016].

[3] Lourentius, "ZFS: Resilver Performance of Various RAID Schemas," 31 January 2016. [Online]. Available: http://louwrentius.com/zfs-resilver-performance-of-various-raid-schemas.html. [Accessed 21 July 2016].

[4] B. Nguyen, Linux File System Hierarchy, GNU Free Documentation License, 2004.

[5] Oracle Solaris ZFS Administration Guide, Oracle, 2013.

[6] "Architectural Overview of the Oracle ZFS Storage Appliance," Oracle White Paper, December 2015.

[7] D. Santi, R. R. M dan Y. Purwanto, "Implementasi dan Analisis Performansi RAID pada Data Storage Infrastructure As A Service (IAAS) Cloud Computing," *JSM STMIK Mikroskil,* vol. 14, pp. 99-107, 2013.

[8] Cisco UCS Servers RAID Guide, San Jose: Cisco System, Inc., 2015.

[9] Adi, Y.R., Nurhayati, O.D., and Widianto, E.D., "Perancangan Sistem Cluster Server untuk Jaminan Ketersediaan Layanan Tinggi pada Lingkungan Virtual." *Jurnal Nasional Teknik Elektro dan Teknologi Informasi (JNTETI)* 5, no. 2 (2016).

[10] W. Ahmed, Proxmox Cookbook, Birmingham: Packt Publishing, 2015.