JEITECH – Journal of Electrical Engineering and Information technology
Department of Electrical Engineering University of Mataram
Vol. 2, No. 2, July 2024, pp. 56-63 ; E-ISSN: 2987-7830

Implementasi Sinkronisasi File Server Dengan *Secure File Transfer Protocol*

Feldy Novanda Mulkan[1], L. A. Syamsul Irfan Akbar[1], Cipta Ramadhani[1]
[1]Jurusan Teknik Elektro Universitas Mataram, Mataram, 83127, Indonesia

ARTICLE INFO

Article history:

Received July 26, 2024
Revised July 31, 2024
Accepted July 31, 2024

Keywords:

Secure File Transfer Protocol;
Synchronization ;
Server ;

ABSTRACT

The web serves as a platform for disseminating and retrieving information. However, during periods of server overload, server performance and client access times can significantly slow down. One viable solution to address this issue is server clustering, where multiple servers are combined into a unified system. Among the methods employed in such scenarios is load balancing, which aims to evenly distribute client requests across multiple servers to optimize performance. Nevertheless, ensuring file synchronization across these clustered servers remains crucial to maintain file availability. Python's watchdog library provides a solution for monitoring file changes and facilitating file synchronization using Secure File Transfer Protocol (SFTP). In synchronization tests, file transfer speeds ranged from 39.54 to 42 Mb/s, with transfer durations varying between 2.02 seconds and 19.05 seconds for file sizes ranging from 10MB to 100MB. Concurrent synchronization experiments revealed that 9 files failed due to data collisions caused by simultaneous file transfers to both servers. Furthermore, the experiments successfully differentiated and compared the MD5 values of each file, demonstrating the capability to detect changes when files were created, deleted, or modified during alternate synchronization processes.

Corresponding Author:

Feldy Novanda Mulkan, 1Jurusan Teknik Elektro Universitas Mataram, Mataram, 83127, Indonesia
Email: fmulkhan@gmail.com

1. PENDAHULUAN

Sejalan dengan perkembangan teknologi *internet* dalam berbagai bidang, maka permintaan pada *server* semakin meningkat. Contoh layanan popular yang meningkat adalah aplikasi bisnis (*e-business*), pendidikan (*e-learning*), berita (*news*), dan lain-lain. Dalam meningkatkan sistem layanan *e-learning*, dibutuhkan suatu sistem *server* yang dapat memberikan ketersediaan yang handal [1].

Web merupakan layanan yang digunakan untuk menyebarkan dan mencari informasi. Layanan web menggunakan *protocol* HTTP dan HTTPS yang dilakukan oleh *web server* [2]. Suatu server memiliki masalah ketika terjadi *overload* pada saat banyak pengguna ingin meminta file pada *server* diwaktu yang bersamaan. Menggabungkan beberapa *server* menjadi satu kesatuan adalah salah satu solusi yang dapat digunakan pada masalah tersebut dengan aman [3],[4]. Untuk menggabungkan *server* dibutuhkan sebuah platform haproxy untuk membagi beban secara merata pada *server* atau bisa disebut load balance [5], haproxy[6] menerapkan konsep *reverse proxy*[7] yang berfungsi sebagai jembatan client ke internet. Menggabungkan *server* di sisi lain juga membutuhkan ketersediaan file agar tetap dapat memberikan akses yang optimalakan tetapi untuk membangun *server* yang *High Availability* [8] dibutuhkan sebuah mekanisme sinkronisasi *file* untuk menyediakan file pada kedua *server*.

Sinkronisasi file adalah proses sinkronisasi atau penyesuaian antara satu *file* di suatu lokasi dengan *file* lain jika ada perubahan yang akan digunakan dan dijalankan dalam suatu sistem[9]. Sinkronisasi *file* umumnya merupakan proses pertukaran data agar memiliki jumlah data yang sama. Untuk mempertahankan kerahasiaan data yang diperlukan untuk menjaga layanan demi keamanan data.

Secure File Transfer Protocol (SFTP) adalah salah satu solusi yang banyak digunakan untuk mengamankan transfer *file* antara client dan *server* [10]. Lebih dari sekadar alat untuk remote login dan menjalankan perintah di jarak jauh, SSH juga berperan sebagai sarana untuk membangun tunnel jaringan

yang aman, meneruskan port TCP, serta mengatur koneksi X11. Di samping itu, SSH juga berperan sebagai alat untuk mentransfer *file* dengan menggunakan protokol SFTP.

Watchdog adalah sebuah *library* dalam bahasa pemrograman *python* yang berfungsi untuk mendeteksi perubahan *file* pada sebuah direktori berdasarkan perubahan waktu modifikasi dan metadata pada sebuah *file*[11]. Pada penilitian ini penulis mencoba membangun sebuah program sinkronisasi yang dapat mendeteksi perubahan file pada *server* kemudian dikirimkan menggunakan protokol SFTP.

2. METODOLOGI

Penilitian ini dilakukan di Laboratorium Jaringan dan Komputer Jurusan Teknik Elektro Fakultas Teknik Universitas Mataram dilakukan mulai bulan Desember tahun 2023 hingga bulan April tahun 2024.

Gambar 1 Diagram Alir Penelitian

2.1. Kebutuhan Harware dan Software

Penelitian yang dilakukan di Jurusan Teknik Elektro Fakultas Teknik Universitas Mataran ini membutuhkan *hardware* dan *software* seperti pada Tabel 1 dan Tabel 2

Tabel 1. Kebutuhan *Hardware*

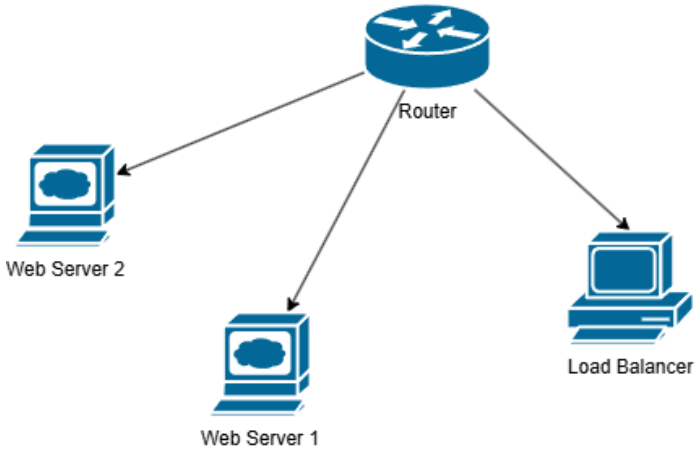
No	*Hardware*	Spesifikasi
1.	PC *Load Balancer*	• *Prosesor Intel Core* I7 12650H • RAM 16 GB • *Hardisk* 1 TB • *Onboard Ethernet Card*
2.	PC *Web Server* 1	• *Intel Core* I5 2.4 GHz • RAM 4 GB • *Hardisk* 512 GB • *Onboard Ethernet Card*
3.	PC *Web Server* 2	• *Intel Core* I5 2.4 GHz • RAM 4 GB • *Hardisk* 512 GB • *Onboard Ethernet Card*
4.	*Router Board*	• *Mikrotik* RB450G • AR7161 680MHz • 512MB *Storage*

		• 256MB RAM • 5 Gigabit LAN *Port* • *Router* OS V6
5.	Kabel UTP	5 Buah

Tabel 2. Kebutuhan *Software*

No	*Software*	Spesifikasi
1.	OS *Server* L*oad Balancer*	Ubuntu
2.	OS *Server*	Ubuntu *Server*
3.	OS PC *Client*	*Windows* 10
4.	Bahasa Pemrograman	Python 3
5.	*Load Balancer*	HAproxy

2.2. Mendesain Topologi Jaringan



Gambar 2 Konfigurasi *Router*

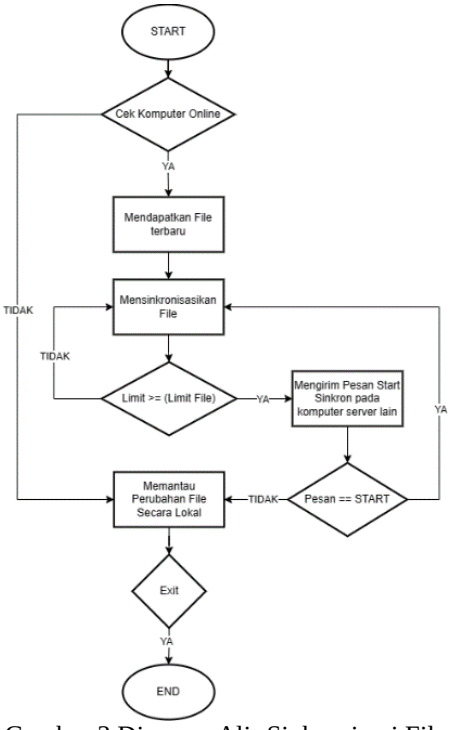
Perancangan topologi jaringan pada penilitian dapat dilihat pada gambar 2 yang terdiri dari awan atau representasi dari internet, 1 buah *router* mikrotik, 1 *load balancer*, dan 2 buah *web server* dimana dengan perangkat-perangkat tersebut akan dibangun rancangan *multiple server*. Dalam pengujiannya nanti akan dilakukan pengaksesan terhadap *load balancer* yang kemudian akan diteruskan ke *web server* yang telah ditentukan oleh *load balancer*.

2.3. Menkonfigurasi *Load Balance*

Pada penelitian ini menggunakan Haproxy sebagai load balancer untuk mendistribusikan beban request pada setiap web server. Pada tahap ini akan dilakukan untuk mengkonfigurasi *IP address* pada komputer load balancer, *Web server* 1 dan *Web server* 2. Kemudian mengkonfigurasi *file* haproxy.cfg untuk konfigurasi rule-rule proxy server untuk melakukan load balance dengan algoritma *round robin*.

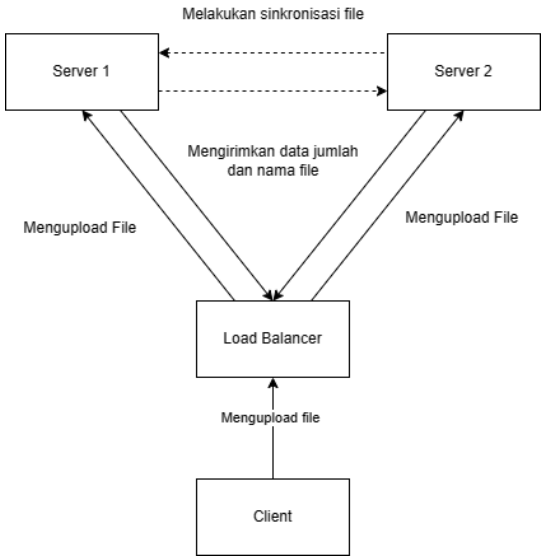
2.4. Membuat Program Sinkronisasi *File*

Pada penelitian ini dibuat program yang berguna untuk melakukan sinkronisasi *file* pada *server*, program ini akan dibuat menggunakan bahasa pemrograman python karena memiliki efisiensi pada saat pembuatan dan memiliki *library watchdog* yang dapat memantau perubahan *file* pada suatu direktori. Program ini akan memanfaatkan library paramiko untuk pengiriman data menggunakan protokol *Secure File Transfer File Transfer Protocol* (SFTP), dengan itu proses sinkronisasi yang dilakukan akan aman karena terdapat proses enkripsi antar data yang dikirim. Komunikasi yang digunakan adalah protokol TCP untuk saling bergantian mengirim *file* dalam jangka waktu tertentu dengan batas *file limit* yang telah ditentukan. Untuk melihat alur programnya dapat dilihat pada gambar 3 diagram alir sinkronisasi *file* dibawah ini:



Gambar 3 Diagram Alir Sinkronisasi File

Pada Gambar 3 menjelaskan alur sinkronisasi yang akan dilakukan dengan menggunakan watchdog yang berfungsi untuk memantau perubahan file pada komputer *server* lokal kemudian mengirimkan file yang berubah dengan menggunakan SFTP (*secure file transfer prtocol*) ke server lainnya.



Gambar 4 Sinkronisasi *File*

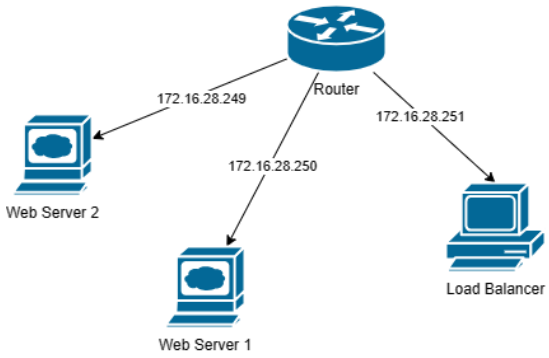
Pada gambar 4 di atas merupakan gambar yang menjelaskan proses sinkronisasi secara dua arah dimana ini dinamakan *master* to *master*[5].

3. HASIL DAN PEMBAHASAN

3.1 Implementasi *Load Balance Server*

Implementasi *load balance* dilakukan dengan melakukan instalasi haproxy dan konfigurasi pada komputer *load balancer*. Pada proses ini akan dilakukan skenario percobaan akses *server* tanpa dan dengan

load balancer, dimana masing masing komputer akan mendapatkan alamat ip seperti gambar 5 dibawah ini:



Gambar 5 Topologi Jaringan *Load Balance*

Pada gambar 5 di atas dapat dilihat bahwa *router* memberikan alamat IP secara dinamis pada komputer *server* dan komputer *load balancer*. Komputer *Load balancer* mendapatkan IP 172.16.28.251, kemudian masing-masing komputer *web server 1* dan *web server* 2 mendapatkan alamat IP 172.168.250 dan 172.16.28.249.

3.2 Konfigurasi Load Balance

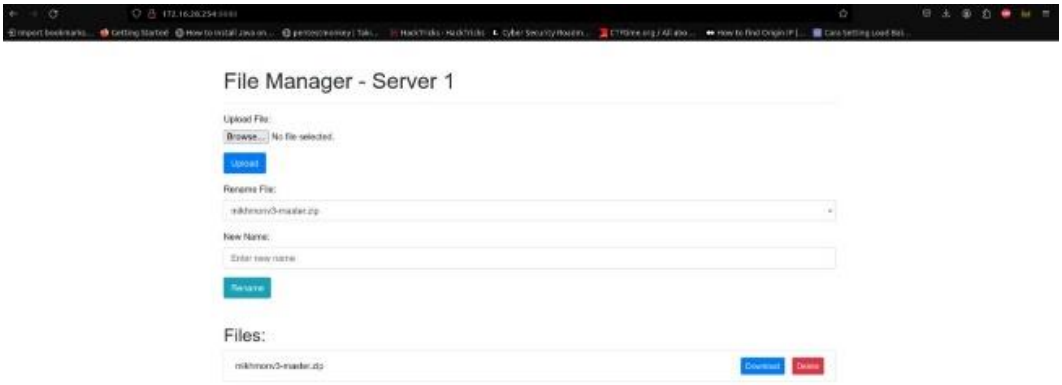
Pada proses ini melakukan instalasi haproxy:

```
sudo apt install haproxy
```

Kemudian melakukan konfigurasi untuk menentukan jalur *web server* dengan memasukkan ip dan *port* yang digunakan pada 2 node *web server* yang akan bertugas merespon *request* yang diteruskan oleh haproxy [6] dan memilih algoritma round robin yang digunakan. Untuk melakukan hal tersebut memasukkan perintah *nano* /etc/haproxy/ haproxy.cfg.

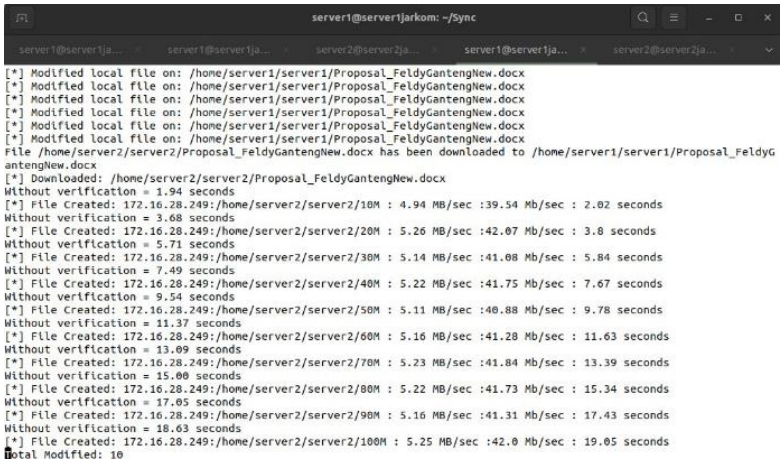
3.3 Hasil Pengujian Sinkronisasi *File* Pada *Server*

Sinkronisasi *server* ini bekerja dengan menggunakan protokol SFTP yang memiliki *mode fileover* untuk *backup* dan sinkronisasi dua arah, program ini dapat menyesuaikan berapa *threads* yang akan digunakan untuk membagi tugas dari sinkronisasi *file*.



Gambar 6 Tampilan Halaman File Manager

Pada gambar 6 pengujian ini menggunakan sebuah *file manager* berbasis *web* yang dijalankan pada kedua *server* yang kemudian *file* tersebut akan tersimpan di direktori yang sudah ditentukan pada kedua *server* tersebut. Setelah mengunggah *file* ke *server* maka maka *file* di *server* 1 akan dikirimkan ke *server* 2 sehingga terjadi sinkronisasi antara kedua server tersebut.



Gambar 7 Proses Sinkronisasi Server

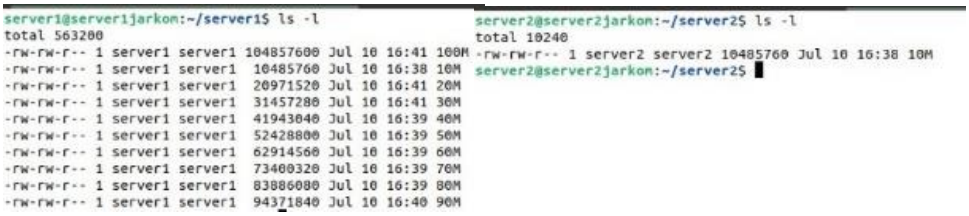
Pada gambar 7 di atas dapat dilihat adalah proses pengiriman *file* dari *server* 1 dengan direktori yang ditetapkan adalah /*home*/*server1*/*server1* dengan ip *address* 172.16.28.250 menuju *server* 2 dengan alamat ip 172.16.28.249 dengan direktori */home/server2/server2* seperti yang ditunjukkan pada gambar di atas. Ketika terjadi penambahan atau pengurangan *file* pada *server* 1 akan di deteksi dengan memanfaatkan *library watchdog* kemudian akan dikirimkan menggunakaan SFTP ke *server* 2. Proses pengiriman ini bergantung pada alamat ip dan direktori *server* pada *server* 2 yang akan digunakan untuk berkomunikasi melaui SSH, kemudian untuk proses validasi menggunakan *username* dan *password* dari target untuk verifikasi *credential user* pada *server* dengan protokol SFTP sehingga dapat mengirimkan *file* secara aman. Pada gambar tersebut dapat dilihat setelah melakukan pengujian pengiriman didapatkan berapa kecepatan pengiriman dan lama pengiriman pada 10 *file*, variasi ukuran *file* mulai dari 10 MB hingga 100 MB.

Gambar 8 Mendeteksi *file* secara *local*

Sesuai dengan gambar 8 ketika terjadi *down* pada salah satu *server* maka dengan memanfaatkan *library watchdog* [11] komputer akan mendeteksi perubahan *file* secara lokal tanpa mengirimkan *file* tersebut, ketika *server* memulai kembali maka akan dilakukan penyesuaian pada kedua *server*. Pada Pengujian dilakukan dengan dua skenario dimana sinkronisasi dijalankan bersamaan pada kedua *server* dan secara bergantian pada kedua *server*.

A. Pengujian Menjalakan Sinskornisasi Secara Bersamaan

Pada Pengujian ini dilakukan dengan menjalankan program sinkronisasi dengan waktu yang bersamaan. Untuk hasil pengujiannya dapat dilihat pada gambar berikut:



Gambar 9 Hasil Pengujian Sinkronisasi

Pada Gambar 9 di atas merupakan hasil pengujian sinkroniasi pada kedua *server* dimana dapat diuraikan pada tabel 3 di bawah ini:

Tabel 3 Hasil Pengujian Sinkronisasi Secara Bersamaan

Ukuran File	Status
10 MB	Berhasil di sinkronisasikan secara utuh, dengan nilai md5 096a872c8346e937fef8e8f90202378a dan ukuran file sama-sama 10 MB
20 MB	File gagal di sinkronisasikan (file tidak ditemukan)
30 MB	File gagal di sinkronisasikan (file tidak ditemukan)
40 MB	File gagal di sinkronisasikan (file tidak ditemukan)
50 MB	File gagal di sinkronisasikan (file tidak ditemukan)
60 MB	File gagal di sinkronisasikan (file tidak ditemukan)
70 MB	File gagal di sinkronisasikan (file tidak ditemukan)
80 MB	File gagal di sinkronisasikan (file tidak ditemukan)
90 MB	File gagal di sinkronisasikan (file tidak ditemukan)
100 MB	File gagal di sinkronisasikan (file tidak ditemukan)

Sesuai dengan tabel 3 di atas dapat dianalisa pada saat pengujian sinkronisasi dengan kedua *server* menjalankan sinkronisasi secara bersamaan terdapat 1 *file* yang berhasil dilakukan dengan melihat ukuran *file* pada kedua *server* memiliki kesamaan. Kemudian dari *file* dengan ukuran 20 MB hingga 100 MB gagal dilakukan yang dapat dibuktikan dengan gambar 9 yang memperlihatkan pada *server* 2 tidak terdapat *file* yang sama pada *server* 1. Hal ini dikarenakan terjadi konflik data dimana ketika *server* 1 berhasil melakukan pengiriman *file* ke *server* 2, pada *server* 2 berusaha mengirimkan file 10 MB kembali ke *server* 1 sehingga terjadi tabrakan data. Kemudian ketika mencoba meng*upload file* pada *server* 2, tidak terjadi sinkronisasi yang disebabkan proses *file transfer* memprioritaskan proses yang dilakukan terlebih dahulu pada *server* 1.

B. Pengujian Menjalankan Sinkronisasi Secara Bergantian

Gambar 10 Hasil Pengujian Sinkronisasi Secara Bergantian

Pada gambar 10 di atas merupakan hasil pengujian sinkronisasi yang dilakukan pada kedua *server* secara bergantian yang dapat di uraikan pada tabel 4 berikut ini:

Tabel 4 Hasil Pengujian Sinkronisasi Secara Bergantian

Nama *File*	Hasil Verifikasi Nilai MD5 Server1	Hasil Verifikasi Nilai MD5 Server2	Status
10M	096a872c8346e937fef8e8f90202378a	096a872c8346e937fef8e8f90202378a	Berhasil
20M	892aac0b59e4479b4b792f831dd718	892aac0b59e4479b4b792f831dd718	Berhasil
30M	7a15860a6be1d809cec9b0c826c5248d	7a15860a6be1d809cec9b0c826c5248d	Berhasil
40M	121009bfae622b8d2dbabf03f00252cd	121009bfae622b8d2dbabf03f00252cd	Berhasil
50M	6a30c1dd69249a4d7bad49af49f57380	6a30c1dd69249a4d7bad49af49f57380	Berhasil
60M	60370ef489b41683dbc1d6fe6a415253	60370ef489b41683dbc1d6fe6a415253	Berhasil
70M	00252edcdc936906a66cfec9519dda62	00252edcdc936906a66cfec9519dda62	Berhasil
80M	353fcf8002ac447f07f11e2f274ce8a8	353fcf8002ac447f07f11e2f274ce8a8	Berhasil
90M	bf5ef5a4d02e4753934b041a5672423	bf5ef5a4d02e4753934b041a5672423	Berhasil
100M	c113496537bd69277465bcaa82b89614	c113496537bd69277465bcaa82b89614	Berhasil

Tabel 5 Kecepatan dan Durasi Pengiriman *File*

Ukuran *File*	Kecepatan *Transfer* (Mb/s)	Durasi Pengiriman (detik)	Durasi Verifikasi Nilai MD5 File (detik)	Total Durasi Pengiriman (detik)
10 MB	39.54	1.94	0.08	2.02
20 MB	42.07	3.68	0.12	3.8
30 MB	41.08	5.71	0.13	5.84
40 MB	41.75	7.49	0.18	7.67
50 MB	40.88	9.54	0.24	9.78
60 MB	41.28	11.37	0.26	11.63
70 MB	41.84	13.09	0.3	13.39
80 MB	41.73	15	0.34	15.34
90 MB	41.31	17.05	0.38	17.43
100 MB	42	18.63	0.42	19.05

Pada Tabel 5 dapat dianalisa bahwa kecepatan pengiriman yang didapatkan dalam satuan *megabit* per detik (Mb/s) menghasilkan hasil yang stabil dan durasi pengiriman terus bertambah sesuai dengan penambahan ukuran *file* yang dilakukan pada saat pengujian, nilai kecepatan tertinggi yang didapatkan adalah 42.07 Mb/s dengan durasi pengiriman 3.68 detik pada pengiriman *file* dengan ukuran 20 MB kemudian kecepatan pengiriman terendah yang didapatkan adalah 39.54 Mb/s dengan durasi pengiriman 1.94 detik.

4. KESIMPULAN

Pengujian sinkronisasi file pada server yang dilakukan berhasil mengirim antar file menggunakan protokol SFTP [10], kecepatan yang didapat adalah 39.54 hingga 42 Mb/s dan durasi pengiriman 2.02 detik hingga 19.05 detik pada variasi percobaan dengan ukuran file 10MB sampai 100MB. Pada saat percobaan sinkronisasi secara bersamaan terdapat 9 file yang gagal di sinkronisasi karena terjadi tabrakan data yang disebabkan oleh pengiriman file secara bersamaan pada kedua server. Kemudian dapat membedakan dan membandingkan nilai md5 masing-masing file ketika file tersebut dibuat, dihapus, dan dimodifikasi pada percobaan dengan menjalankan sinkronisasi secara bergantian.

5. REFERENSI

[1] Riskiono, S. D., & Pasha, D. (2020). Analisis Perbandingan Server Load Balancing dengan Haproxy & Nginx dalam Mendukung Kinerja Server E- Learning. *Jurnal Telekomunikasi dan Komputer*, *10*(3), 135. https://doi.org/10.22441/incomtech.v10i3.8751
[2] Oktarini, A., Ari, S.;, & Sunarti, A. ; (2019). *Web Programming* (1 ed.). Graha Ilmu.
[3] Aprilliandi, R. A., & Efendi, R. (2019). Perancangan Dan Implementasi Load Balancing Web Server Menggunakan Haproxy (High Availability Proxy) Studi Kasus di SMK Telekomunikasi Tunas Harapan Kab. Semarang. *Fakultas Teknologi Informasi Universitas Kristen Satya Wacana*.
[4] Amalia, R. (2020). *Analisis Kinerja Web Server Menggunakan Metode Load Balancing As A Service*. POLITEKNIK NEGERI JAKARTA.
[5] Cynthia, E. P., Iskandar, I., & Sipayung, A. A. (2020). Rancang Bangun Server HAproxy Load Balancing Master to Master MySQL (Replication) Berbasis Cloud Computing. *Jurnal Ilmu Komputer dan Informatika*, *04*, 45–54.
[6] Haproxy. (2023). *Haproxy*. https://www.haproxy.com/solutions/load-balancing
[7] Ma, C., & Chi, Y. (2022). Evaluation Test and Improvement of Load Balancing Algorithms of Nginx. IEEE Access, 10, 14311–14324. https://doi.org/10.1109/ACCESS.2022.3146422
[8] Rahmatulloh, A., & Firmansyah, M. (2017). Implementasi Load Balancing Web Server menggunakan Haproxy dan Sinkronisasi *File* pada Sistem Informasi Akademik Universitas Siliwangi. *Jurnal Nasional Teknologi dan Sistem Informasi*, *3*(2), 241–248.https://doi.org/10.25077/teknosi.v3i2.2017.241-248
[9] Murti, A. C., & Triyanto, W. A. (2022). PERANCANGAN MODEL IMPLEMENTASI KRIPTOGRAFI DALAM SEBUAH SISTEM INFORMASI BERBASIS WEB. Indonesian Journal of Technology, Informatics and Science (IJTIS), 3(2), 57–60. https://doi.org/10.24176/ijtis.v3i2.8255
[10] Sukaridhoto, ST., Ph. D., S., & Putri Nourma Budiarti, ST. MT., R. (2020). *Dasar dan Aplikasi Jaringan Komputer* (1 ed.). UNUSA PRESS. https://press.unusa.ac.id/
[11] Watchdog. (2024). *Watchdog*. https://python-watchdog.readthedocs.io/en/stable/
[12] Widana, F. (2015). Komparasi Sinkronisasi File Antara Metode Mobile Agent Dan Metode Peer-To-Peer. Jurnal Masyarakat Telematika dan Informasi, 6(2), 101–110.