JISKA (Jurnal Informatika Sunan Kalijaga)
Vol. 9, No. 1, JANUARI, 2024, Pp. 39 – 49
ISSN: 2527 – 5836 (print) | 2528 – 0074 (online)

Implementasi *Load Balancing* dengan HAProxy di Sistem Informasi Akademik UIN Sunan Kalijaga

Adi Wirawan [(1)], Rahmadhan Gatra [(2)*], Hendra Hidayat [(3)], Daru Prasetyawan [(4)]
Pusat Teknologi Informasi dan Pangkalan Data, UIN Sunan Kalijaga, Yogyakarta
e-mail : {adi.wirawan,rahmadhan.gatra,hendra.hidayat,daru.prasetyawan}@uin-suka.ac.id.
* Penulis korespondensi.
Artikel ini diajukan 4 September 2023, direvisi 1 November 2023, diterima 7 November 2023, dan dipublikasikan 25 Januari 2024.

Abstract

Efficiently managing academic information systems (AIS) is essential for educational institutions to provide reliable services to students and faculty. This research explores the integration of HAProxy load balancing and file synchronization techniques to optimize the performance of AIS. HAProxy is employed to distribute incoming requests across multiple backend servers, and the backend will call web service to access the data saved in the database to facilitate seamless data sharing and access. Additionally, file synchronization mechanisms are implemented to maintain consistency across scripts used in the backend system. The study conducts performance evaluations and benchmarks to assess the impact of HAProxy load balancing and file synchronization on AIS responsiveness and reliability. The results reveal significant system scalability and fault tolerance improvements, reducing downtime and enhancing user experience. This research contributes to optimizing academic information systems, enhancing their ability to handle increased loads, and ensuring the efficient delivery of educational services.

Keywords: HAProxy, Load Balancing, File Synchronization, Web Service, Academic Information Systems

Abstrak

Pengelolaan sistem informasi akademik (SIA) secara efisien sangat penting bagi institusi pendidikan untuk memberikan layanan yang dapat diandalkan kepada mahasiswa dan dosen. Penelitian ini mengeksplorasi integrasi teknik *load balancing* HAProxy dan sinkronisasi *file* untuk mengoptimalkan kinerja SIA. HAProxy digunakan untuk mendistribusikan permintaan masuk ke beberapa server *backend*, dan *backend* akan memanggil *web service* untuk mengakses data yang disimpan dalam *database* untuk memfasilitasi berbagi dan akses data tanpa hambatan. Selain itu, mekanisme sinkronisasi *file* diterapkan untuk menjaga konsistensi antar skrip yang digunakan dalam sistem *backend*. Studi ini melakukan evaluasi kinerja dan tolok ukur untuk menilai dampak penyeimbangan beban HAProxy dan sinkronisasi file terhadap respons dan keandalan SIA. Hasilnya menunjukkan peningkatan signifikan dalam skalabilitas sistem dan toleransi kesalahan, mengurangi *downtime* dan meningkatkan pengalaman pengguna. Penelitian ini berkontribusi pada optimalisasi sistem informasi akademik, meningkatkan kemampuan mereka untuk menangani peningkatan beban, dan memastikan penyampaian layanan pendidikan yang efisien.

Kata Kunci: HAProxy, *Load Balancing,* Sinkronisasi *File*, *Web Service*, Sistem Informasi Akademik

1. PENDAHULUAN

Kebutuhan sebuah infrastruktur teknologi yang handal seperti server merupakan salah satu permasalahan yang sering dihadapi oleh instansi pemerintahan maupun swasta dalam melakukan pengelolaan data. Pengelolaan data tersebut dapat terjadi ribuan bahkan jutaan kali dalam setiap harinya. Server merupakan sebuah sistem perangkat *compute* yang menyediakan layanan-layanan tertentu seperti sistem operasi, program aplikasi, maupun data informasi kepada komputer lain yang saling terhubung dalam sebuah jaringan komputer (Harefa et al.,

2021). Pada dunia pendidikan seperti universitas, aplikasi akademik yang terinstal di server merupakan sebuah aplikasi yang banyak digunakan untuk memenuhi kegiatan administrasi dalam dunia pendidikan. Meningkatnya aktifitas kinerja pada saat pemilihan mata kuliah yang akan diambil di setiap awal semester membuat peningkatan permintaan data yang terjadi di server akademik sehingga selalu mengalami kelebihan kapasitas (*overload*).

Merealisasikan kebutuhan infrastruktur yang handal untuk dapat mengimplementasikan aplikasi berbasis web perlu adanya penyesuaian terlebih dahulu. Sebagai contoh pengimplementasian sebuah aplikasi berbasis web harus membutuhkan konfigurasi server yang handal dengan jaringan yang sudah terintegrasi. UPT. Pusat Teknologi Informasi dan Pangkalan Data (PTIPD) sebagai unit pelaksana teknis yang mengelola semua infrastruktur dan sistem informasi termasuk sistem informasi akademik di UIN Sunan Kalijaga Yogyakarta mempunyai kewajiban untuk mengatur, mengelola dan melakukan manajemen di bidang sistem informasi akan dapat digunakan secara lancar oleh Civitas academica baik di dalam lingkungan kampus maupun di luar lingkungan kampus (internet). Ada waktu-waktu tertentu di mana kerja aplikasi begitu tinggi, yang akhirnya mengakibatkan sever menjadi *overload*, sebagai contoh saat ada proses KRS. Pada saat terjadinya *overload* ini yang dilakukan adalah meningkatkan kemampuan server dalam melayani permintaan dari *user* serta dengan mengatur ulang penjadwalan akses *user* untuk melakukan *request* ke aplikasi.

Implementasi aplikasi berbasis web membutuhkan suatu konfigurasi server yang handal dan dapat mengantisipasi kebutuhan masa yang akan datang (Rijayana, 2005). Beberapa solusi yang bisa diterapkan dalam mengatasi kejadian *overload* server ketika melayani transaksi permintaan data dari server ke *client* yakni dengan melakukan penambahan unit server baru serta menerapkan metode *clustering* (Rahmatulloh & MSN, 2017). Terdapat dua fungsi yang diterapkan pada metode *clustering* yaitu *failover cluster* dan *load balancing cluster*. *Failover* adalah situasi aplikasi melakukan *restart* aplikasi di *node* yang berbeda ketika yang seharusnya bisa melayani *request* gagal untuk memberikan *response* (M. López et al., 2012). Server berfungsi sebagai metode *failover* jika ada *node cluster* yang mengalami kerusakan maka akan digantikan dengan server yang lain, sehingga permintaan layanan tidak mengalami gangguan, sedangkan metode *load balancing* berfungsi ketika server menangani permintaan data maka akan terbagi secara merata ke semua *node* sehingga tidak mengalami kelebihan kapasitas (*overload*) dalam pemintaan data (Harefa et al., 2021).

Beban yang diterima oleh server bisa dibagi ke beberapa server untuk menghindari *bottleneck,* cara ini disebut dengan *load balancing. Load balancing* mengacu kepada sebuah kondisi di mana beban yang diterima oleh tiap server dalam kondisi sama (R. M. Zebari & O. Yaseen, 2011). Hal yang pertama dilakukan pada proses *load balancing* adalah membagi beban yang datang ke server kemudian melakukan pengecekan terhadap bentuk distribusi beban yang akan dijalankan. Apabila proses pengiriman beban telah dilakukan, maka beban yang datang ke server telah terbagi secara merata (Adil Yazdeen et al., 2021; Bathiya et al., 2016; Shukla & Kumar, 2018). Pemilihan metode yang sesuai dalam proses *load balancing* diperlukan untuk meningkatkan kemampuan distribusi kepada web server. Untuk mendapatkan hasil yang terbaik dalam membagi beban, *request* yang datang dari pengguna perlu dibagi melalui jalur DNS di antara web server yang ada di dalam *cluster* berdasar strategi yang diterapkan untuk melayani *request* yang dilakukan oleh pengguna. Selanjutnya, beban yang diterima oleh web server perlu dibagi kepada web server lain untuk meningkatkan kemampuan cluster dan memaksimalkan penggunaan sumber daya yang dimiliki oleh server (Abdulmohsin, 2016; Amanuel & Ameen, 2021).

HAProxy merupakan solusi yang murah, cepat, dan merupakan solusi handal yang menawarkan *high availability, failover,* dan skema *load balancing,* yang juga bisa digunakan sebagai *proxy* untuk aplikasi yang menggunakan TCP dan HTTP di dalam aplikasinya (Kaushal & Bala, 2011). HAProxy bisa digunakan untuk membagi beban *request* yang bisa dijalankan di sistem operasi Linux, Solaris, dan FreeBSD (Ahmad et al., 2021). HAProxy juga sudah umum untuk bisa digunakan dalam membagi beban kerja dari beberapa server seperti *web server*,

database server, smtp server, dan yang lainnya. Salah satu algoritma yang digunakan di HAProxy adalah Round Robin. Round Robin merupakan algoritma yang banyak digunakan di bidang *computer science*. Hal penting dari algoritma Round Robin ini adalah karena dia mudah untuk diimplementasikan dan mudah untuk dimengerti (Kumari, 2016). Diibaratkan terdapat dua server yang menunggu *request* yang masuk dari pengguna kepada sebuah server *load balancer.* Misalkan ada sebuah *request* masuk, maka berikutnya *load balancer* akan meneruskan *request* yang masuk kepada server yang pertama. Ketika ada *request* kedua masuk kepada *load balancer* maka *request* tersebut akan diteruskan kepada server yang kedua (Khiyaita et al., 2012).

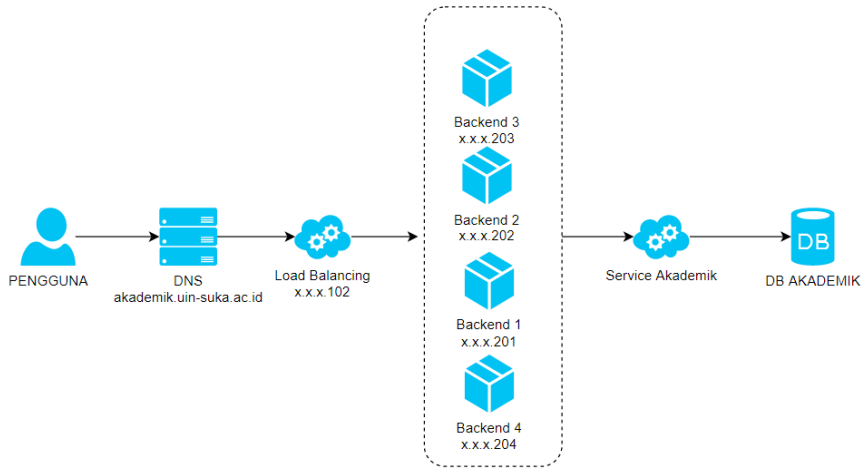
Server yang berada di belakang *load balancing* akan meneruskan setiap permintaan data menuju ke satu alamat *web service*. Menurut Chen et al. (2005), arsitektur *web service* mampu memodelkan interaksi antara tiga peran yang dimiliki yaitu peran penyedia layanan, peran konsumen layanan, dan peran pendaftar layanan. Dalam penggunaan *web service* ada beberapa *protocol* yang digunakan untuk melakukan proses transfer data, yakni Simple Object Access Protocol (SOAP) dan Representational State Transfer (REST) (Fauziah et al., 2022). Menurut penelitian Memeti et al. (2018) REST lebih baik dibandingkan dengan SOAP, karena REST bisa mendukung pertukaran data dengan format JSON maupun XML.

Tujuan dari penelitian ini adalah menerapkan fitur *load balancing* dengan menggunakan HAProxy pada *website* untuk membagi *request* yang masuk sehingga beban yang diterima *website* bisa dibagi ke beberapa server, serta untuk menghindari downtime layanan yang ada di *website* jika ada *node* server yang tidak bisa melayani *request* yang masuk.

2. METODE PENELITIAN

2.1 Arsitektur Yang Akan Diterapkan

Arsitektur yang akan diterapkan pada server sistem informasi akademik dengan menggunakan metode *load balancing* setidaknya terdapat beberapa bagian, di antaranya: DNS server, *load balancing* (HAProxy), server *backend*, *service* akademik, dan *database*. Dalam arsitektur HAProxy terdiri empat server yang berfungsi sebagai server *backend* dan satu server yang difungsikan sebagai server *load balancing* dalam mendistribusikan beban kinerja yang dilakukan oleh *end user*, sehingga koneksi yang didistribusikan merata ke masing-masing server *backend*. Penambahan server yang dijadikan sebagai *backend* bisa meningkatkan performa dari aplikasi, karena *request* yang datang ke server bisa dibagi ke beberapa server yang ada di dalam *cluster* (Data et al., 2019). Berikut ini tampilan dari alur arsitektur server *load balancing* seperti terlihat pada Gambar 1.



Gambar 1 Arsitektur Sistem

2.2 Kebutuhan *Software* dan *Hardware*

Sistem operasi yang digunakan untuk menjalankan *load balancing* berupa HAProxy, *backend,* dan *web service* yaitu Ubuntu 20.04.1 LTS, CPU 25 core, dengan menggunakan model Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz. Sedangkan untuk menjalankan *database* digunakan Windows Server 2012 R2, CPU 16 core, dengan seri Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz.

2.3 Konfigurasi HAProxy

Tahapan dalam menggunakan HAProxy sebagai *load balancer* dimulai dengan melakukan instalasi HAProxy di server yang akan digunakan sebagai *load balancer*. Perintah dimulai dengan menambahkan *repository* HAProxy, kemudian dilanjutkan dengan proses instalasi. Konfigurasi HAProxy pada server terlihat pada perintah pada Gambar 2.

```
sudo add-apt-repository ppa:vbernat/HAProxy-1.8
sudo apt-get update
sudo apt-get install HAProxy
```

```
global
        log /dev/log local0
        log /dev/log local1 notice
        chroot /var/lib/HAProxy
        stats socket /run/HAProxy/admin.sock mode 660 level admin expose-fd listeners
        user HAProxy
        group HAProxy
        daemon

frontend Local_Server
        mode http
        bind x.x.x.102:80
        bind x.x.x.102:443 ssl crt /etc/sertifikat2022/uin2022.pem
        http-request redirect scheme https unless { ssl_fc }
        default_backend nodes
        maxconn 4000

backend nodes
        balance roundrobin
        cookie SERVERID insert indirect nocache
        server s1 x.x.x.201:80 check cookie s1
        server s2 x.x.x.202:80 check cookie s2
        server s3 x.x.x.203:80 check cookie s3
        server s4 x.x.x.204:80 check cookie s4
```

Gambar 2 Konfigurasi HAProxy

Gambar 2 memberikan informasi konfigurasi yang dilakukan terhadap HAProxy. Pada bagian *backend nodes* konfigurasi yang dilakukan adalah menggunakan metode Round Robin dengan menggunakan *sticky cookie* untuk membuat pengguna tetap berada pada server yang sama ketika pengguna membuka kembali aplikasi di lain waktu (P. López & Baydal, 2018). Terdapat 4 buah *backend* yang digunakan yang diberikan nama s1, s2, s3, dan s4.

2.4 Konfigurasi Lsyncd

Dalam proses pengembangan aplikasi, hanya satu server yang dituju untuk diubah atau ditambahkan, server yang dituju tersebut berada di alamat x.x.x.201. Server yang dituju tersebut akan melakukan sinkron kepada 3 server lainnya, sehingga 4 server yang digunakan

akan selalu sinkron untuk perubahan atau penambahan yang terjadi di bagian *script* aplikasi. Untuk proses sinkronisasi *file* maka digunakan Lsyncd yang bisa dimanfaatkan untuk melakukan replikasi terhadap data-data yang berada dalam direktori tertentu dalam server (Pratama et al., 2021).

Aplikasi sistem informasi akademik yang dikembangkan di UIN Sunan Kalijaga tidak menyimpan data yang diunggah oleh pengguna di server aplikasi, tetapi semua data yang diunggah oleh pengguna disimpan di server *database*. Hal itulah yang menjadi alasan bahwa yang perlu dilakukan proses sinkron adalah bagian *script* saja.

```
sudo apt-get install lsyncd -y
```

```
settings {
      logfile = "/var/log/lsyncd/lsyncd.log",
      statusFile = "/var/log/lsyncd/lsyncd.status",
      statusInterval = 20,
      nodaemon = false,
      insist = true
}

sync {
      default.rsyncssh,
      source = "/var/www/html/",
      host = "x.x.x.202",
      targetdir = "/var/www/html/"
}

sync {
      default.rsyncssh,
      source = "/var/www/html/",
      host = "x.x.x.203",
      targetdir = "/var/www/html/"
}

sync {
      default.rsyncssh,
      source = "/var/www/html/",
      host = "x.x.x.204",
      targetdir = "/var/www/html/"
}
```

Gambar 3 Konfigurasi Lsyncd

Gambar 3 merupakan konfigurasi dari program Lsyncd, program tersebut akan melakukan sinkronisasi *file* kepada server yang lain. Terdapat tiga buah server yang menjadi tujuan proses sinkronisasi yang dijelaskan seperti yang ada di Gambar 3. Konfigurasi tersebut dimasukkan ke dalam program dengan inisial *sync,* di mana di dalamnya terdapat parameter di antaranya *source* yang menandakan sumber asal, *host* merupakan lokasi dari server tujuan, dan *tergetdir* yang merupakan folder tujuan.

3. HASIL DAN PEMBAHASAN

3.1 Pengujian Waktu *Response*

Tahap pertama yang dilakukan adalah melakukan pengecekan bahwa *request* yang masuk ke aplikasi bisa terbagi dengan menggunakan algoritma Round Robin yang disediakan oleh

HAProxy. Pengujian dilakukan dengan menggunakan Apache Jmeter, yang bisa memberikan hasil yang lebih baik dibanding *tools* yang lain (Abbas et al., 2017).

Langkah yang pertama dilakukan adalah menyiapkan satu server untuk menangani di bagian *backend* aplikasi. Dari server tersebut kemudian dibandingkan waktu *response* yang dihasikan dari penambahan server yang ada. Semakin rendah waktu yang diperlukan oleh aplikasi untuk menerima *response*, maka akan semakin membuat pengguna tetap berada di aplikasi yang sama (Nah, 2004). Rekap waktu *response* ditampilkan di Tabel 1. Dilakukan beberapa percobaan *request* ke server pertama dengan jumlah *request* yang berbeda-beda. Dengan meningkatnya jumlah *request* yang dilakukan, maka waktu yang diberikan oleh server untuk melakukan *response* menjadi meningkat.

Tabel 1 Rekap Distribusi *Request* oleh HAProxy dengan Satu Server

No.	Jumlah *Request*	*Response*	Beban Server 1
1	150	1523 ms	150
2	200	1698 ms	200
3	250	1721 ms	250
4	300	2092 ms	300

Langkah yang kedua adalah menambahkan satu buah server, sehingga menjadi 2 buah server yang menangani *request* di bagian *backend* aplikasi. Rekap waktu *response* dan pembagian server yang dilakukan oleh HAProxy ditampilkan pada Tabel 2. Diperlihatkan di Tabel 2 bahwa dengan menambahkan server di bagian *backend*, maka *response* yang diberikan oleh aplikasi menjadi lebih rendah dibandingkan ketika hanya menggunakan 1 server saja. Pada Tabel 2 juga diperlihatkan bahwa pembagian beban yang diterima tiap server adalah merata.

Tabel 2 Rekap Distribusi *Request* oleh HAProxy dengan Dua Server

No.	Jumlah *Request*	*Response*	Beban Server 1	Beban Server 2
1	150	1319 ms	75	75
2	200	1373 ms	100	100
3	250	1544 ms	125	125
4	300	1572 ms	150	150

Langkah yang ketiga adalah menambahkan satu buah server kembali, sehingga menjadi 3 buah server yang menangani *request* di bagian *backend* aplikasi. Rekap waktu *response* dan pembagian server yang dilakukan oleh HAProxy ditampilkan pada Tabel 3. Di mana pada tabel *response* yang diberikan aplikasi menjadi berkurang dibandingkan dengan hanya menggunakan 2 buah server dan tiap-tiap server menerima beban yang relatif merata.

Tabel 3 Rekap Distribusi *Request* oleh HAProxy dengan Tiga Server

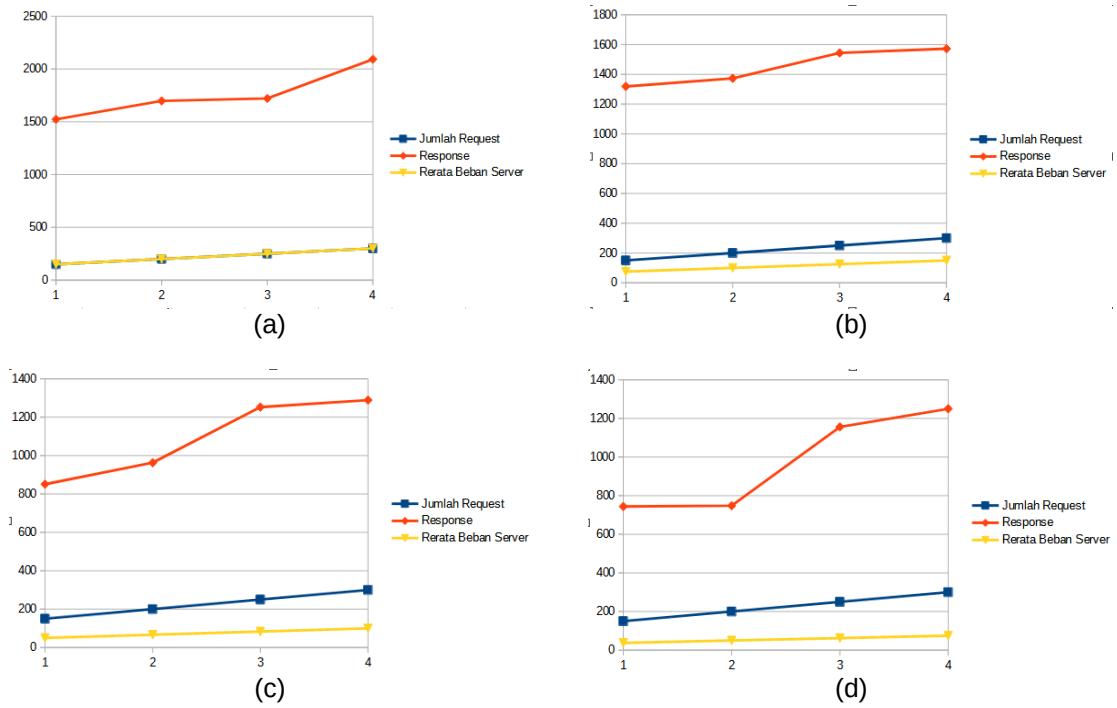
No.	Jumlah *Request*	*Response*	Beban Server 1	Beban Server 2	Beban Server 3
1	150	851 ms	50	50	50
2	200	963 ms	67	67	66
3	250	1253 ms	83	84	83
4	300	1289 ms	100	100	100

Langkah yang keempat adalah menambahkan satu buah server, sehingga menjadi 4 buah server yang menangani *request* di bagian *backend* aplikasi. Rekap waktu *response* dan pembagian server yang dilakukan oleh HAProxy ditampilkan pada Tabel 4. Di mana *response* yang diberikan oleh aplikasi menjadi berkurang dibandingkan hanya dengan menggunakan 3 buah server dan tiap-tiap server mendapatkan beban yang relatif merata.

Tabel 4 Rekap Distribusi *Request* oleh HAProxy dengan Empat Server

No.	Jumlah *Request*	*Response*	Beban Server 1	Beban Server 2	Beban Server 3	Beban Server 4
1	150	744 ms	37	37	38	38
2	200	748 ms	50	50	50	50
3	250	1156 ms	62	62	63	63
4	300	1250 ms	75	75	75	75

Dari seluruh pengujian yang dilakukan, dilakukan perbandingan *response time* yang diberikan dengan menambahkan jumlah server di bagian *backend* aplikasi. Dengan menambahkan server *backend* pada aplikasi, maka waktu yang diperlukan untuk aplikasi memberikan *response* semakin berkurang. Gambar 4 memberikan gambaran perbandingan *response time* pada seluruh pengujian *response time*.

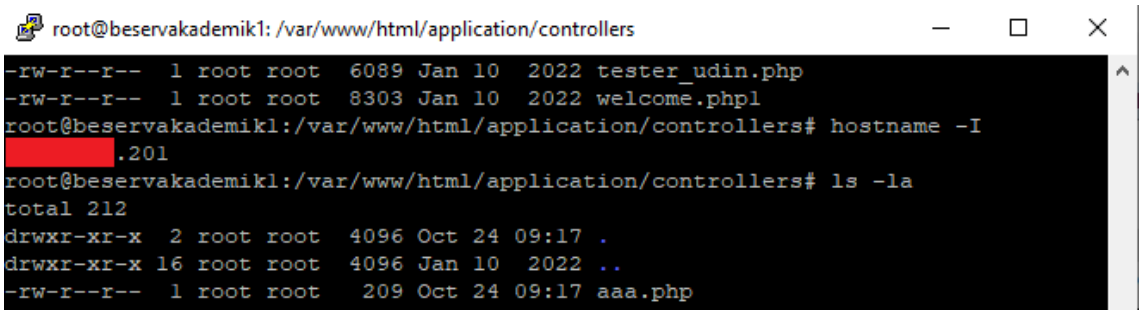


Gambar 4 Perbandingan *Response Time* pada (a) Jumlah Server 1, (b) Jumlah Server 2, (c) Jumlah Server 3, dan (d) Jumlah Server 4

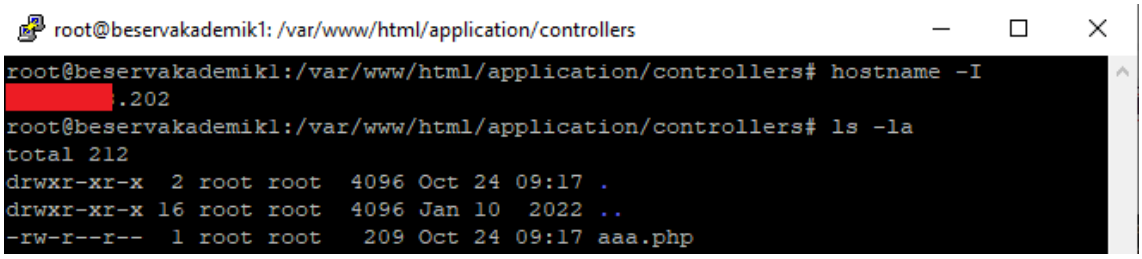
3.2 Pengujian Proses Sinkronisasi *File*

Proses pengujian berikutnya adalah memastikan bahwa proses sinkronisasi *file* yang terjadi antara server utama dalam hal ini adalah server dengan ip x.x.x.201 dengan ketiga server lainnya bisa berjalan dengan baik. Proses pengujian seperti yang tampak di Gambar 5 dilakukan dengan membuat sebuah *file* dengan nama ‘aaa.php’ dan diisikan *script* untuk melakukan pengecekan server yang sedang berjalan. Proses sinkronisasi dianggap berhasil apabila di ketiga server yang lain, yakni server x.x.x.202, x.x.x.203, dan server x.x.x.204 terbentuk *file* dan dengan isi yang sama.

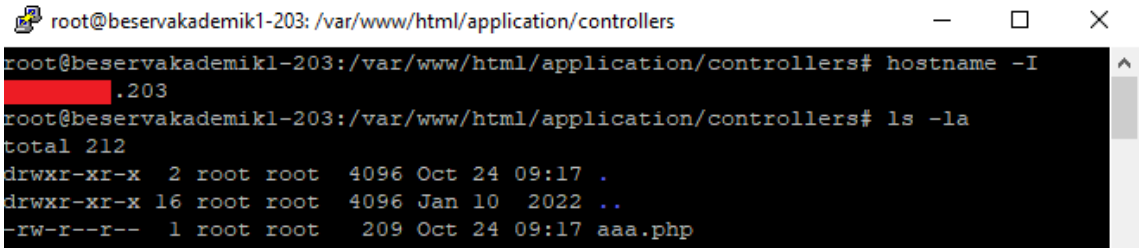
Hasil menunjukkan pada Gambar **Error! Reference source not found.**, 7, dan 8 bahwa secara otomatis akan tercipta *file* ‘aaa.php’ di tiga server yang lain dengan isi yang sama dengan *file* yang berada di server x.x.x.201. Selanjutnya pada Gambar 9 dan 10 menunjukkan bahwa apabila *script* dijalankan, maka akan memberikan informasi ip server yang berbeda-beda tergantung lokasi server yang menjalankan *script* tersebut.



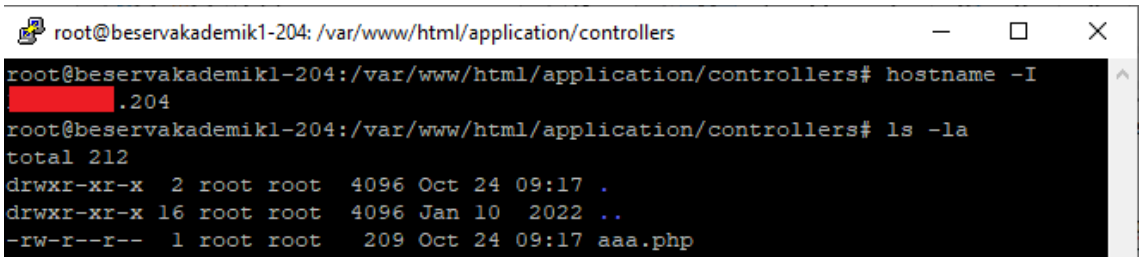
Gambar 5 Server x.x.x.201



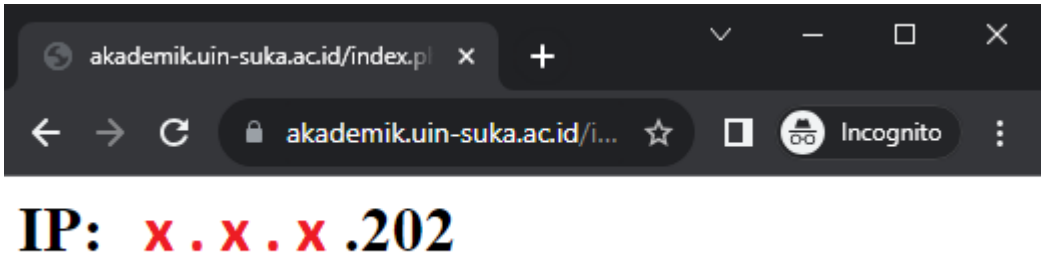
Gambar 6 Server x.x.x.202



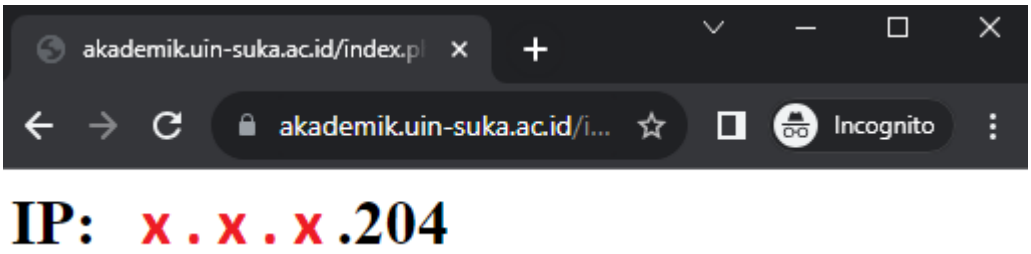
Gambar 7 Server x.x.x.203



Gambar 8 Server x.x.x.204



Gambar 9 Tampilan *Script* di Server x.x.x.202



Gambar 10 Tampilan *Script* di Server x.x.x.204

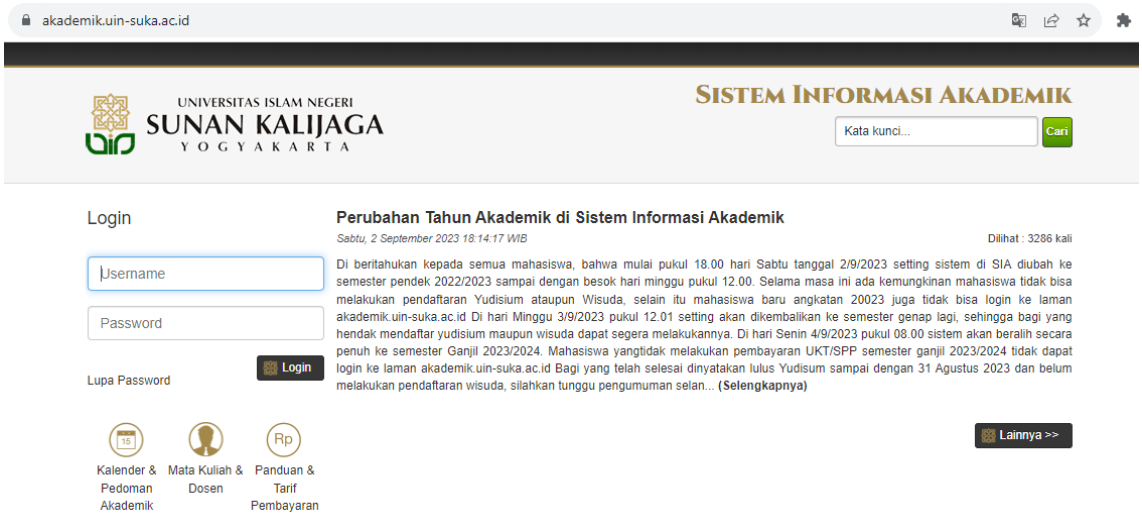
3.3 Pengujian Failover

Node server yang memiliki IP x.x.x.202 akan dimatikan untuk *apache service* yang dimiliki untuk melakukan pengujian proses *failover*. Gambar 11 menunjukkan untuk server s2 yang memiliki IP x.x.x.202 ditampilkan dengan *background* warna merah yang menunjukkan bahwa status server s2 dalam posisi mati atau tidak bisa melayani *request* yang masuk. Gambar 12 menunjukkan bahwa aplikasi tetap bisa melayani *request* yang masuk dengan mengarahkan *request* menuju ke *node* yang masih hidup dan siap melayani *request*.

nodes

	Queue			Session rate			Sessions						Bytes		Denied	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp
s1	0	0	-	4	804		7	1 035	-	5 453 273	835 890	1s	8 206 661 006	92 412 222 938		0
s2	0	0	-	2	2 724		0	3 999	-	5 769 689	826 446	1s	7 958 436 325	87 811 585 945		0
s3	0	0	-	4	805		0	966	-	5 351 760	835 520	1s	8 236 587 137	88 207 501 403		0
s4	0	0	-	3	805		1	1 070	-	5 317 905	835 754	1s	8 230 182 167	92 419 079 469		0
Backend	0	0		11	2 623		8	3 999	400	21 824 143	3 333 610	1s	32 631 917 954	360 850 402 131	0	0

Gambar 11 *Statistic* Server *Backend* yang *Running*



Gambar 12 Tampilan Aplikasi Akademik

4. KESIMPULAN

Hasil penelitian menunjukkan bahwa dengan menggunakan HAProxy dan penambahan server *backend* dapat mengoptimalisasi kerja aplikasi sistem informasi akademik UIN Sunan Kalijaga. Hal ini ditunjukkan dengan ketika ada *request* yang datang kepada suatu aplikasi, maka beban aplikasi bisa didistribusikan secara merata kepada setiap *backend* yang dimiliki oleh aplikasi serta mengurangi *response time* yang diberikan aplikasi. Penggunaan Lsyncd untuk melakukan proses sinkronisasi *file* antar server juga bekerja dengan baik pada server. Dilihat dari

keberhasilan dalam membuat *script* yang ada di tiap *backend* aplikasi menjadi sama antara satu server dengan server yang lain.

DAFTAR PUSTAKA

Abbas, R., Sultan, Z., & Bhatti, S. N. (2017). Comparative analysis of automated load testing tools: Apache JMeter, Microsoft Visual Studio (TFS), LoadRunner, Siege. *2017 International Conference on Communication Technologies (ComTech)*, 39–44. https://doi.org/10.1109/COMTECH.2017.8065747

Abdulmohsin, H. A. (2016). A Load Balancing Scheme for a Server Cluster Using History Results. *Iraqi Journal of Science*, 2121–2130. https://ijs.uobaghdad.edu.iq/index.php/eijs/article/view/6946

Adil Yazdeen, A., Zeebaree, S. R. M., Mohammed Sadeeq, M., Kak, S. F., Ahmed, O. M., & Zebari, R. R. (2021). FPGA Implementations for Data Encryption and Decryption via Concurrent and Parallel Computation: A Review. *Qubahan Academic Journal*, *1*(2), 8–16. https://doi.org/10.48161/qaj.v1n2a38

Ahmad, U. A., Saputra, R. E., & Harahap, R. M. (2021). Implementasi High Availability Server Menggunakan Platform Haproxy (studi Kasus: Aplikasi Zammad Untuk Online Help Desk). *EProceedings of Engineering*, *8*(5). https://openlibrarypublications.telkomuniversity.ac.id/index.php/engineering/article/view/16305/16011

Amanuel, S. V. A., & Ameen, S. Y. A. (2021). Device-to-Device Communication for 5G Security : A Review. *Journal of Information Technology and Informatics*, *1*(1), 26–31. https://www.qabasjournals.com/index.php/jiti/article/view/27

Bathiya, B., Srivastava, S., & Mishra, B. (2016). Air pollution monitoring using wireless sensor network. *2016 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*, 112–117. https://doi.org/10.1109/WIECON-ECE.2016.8009098

Chen, M., Zhang, D., & Zhou, L. (2005). Providing web services to mobile users: the architecture design of an m-service portal. *International Journal of Mobile Communications*, *3*(1), 1. https://doi.org/10.1504/IJMC.2005.005870

Data, M., Kartikasari, D. P., & Bhawiyuga, A. (2019). The Design of High Availability Dynamic Web Server Cluster. *2019 International Conference on Sustainable Information Engineering and Technology (SIET)*, 181–186. https://doi.org/10.1109/SIET48054.2019.8986069

Fauziah, A., Noer, S., Junaedi, J., & Riyandi, M. A. (2022). Sistem Penjualan Sayur Menggunakan Framework Laravel. *Jurnal RESTIKOM : Riset Teknik Informatika Dan Komputer*, *3*(1), 42–50. https://doi.org/10.52005/restikom.v3i1.80

Harefa, H. S., Triyono, J., & Raharjo, S. (2021). Implementasi Load Balancing Web Server untuk Optimalisasi Kinerja Web Server dan Database Server. *Jurnal Jarkom*, *9*(1), 10–20. https://ejournal.akprind.ac.id/index.php/jarkom/article/view/3670

Kaushal, V., & Bala, A. (2011). Autonomic Fault Tolerance Using HAProxy in Cloud Enviorment. *(IJAEST) International Journal of Advanced Engineering Sciences and Technologies*, *7*(2), 222–227. https://www.semanticscholar.org/paper/Autonomic-Fault-Tolerance-Using-HAProxy-in-Cloud-Kaushal-Bala/84c5afc5d807fd76ddf7b7ae27e3e44887f5cfe6

Khiyaita, A., Bakkali, H. El, Zbakh, M., & Kettani, D. El. (2012). Load balancing cloud computing: State of art. *2012 National Days of Network Security and Systems*, 106–109. https://doi.org/10.1109/JNS2.2012.6249253

Kumari, P. (2016). A Round-Robin based Load balancing approach for Scalable demands and maximized Resource availability. *International Journal Of Engineering And Computer Science*, *5*(8), 17375–17380. https://doi.org/10.18535/ijecs/v5i8.04

López, M., Castro, L. M., & Cabrero, D. (2012). Failover and takeover contingency mechanisms for network partition and node failure. *Proceedings of the Eleventh ACM SIGPLAN Workshop on Erlang Workshop*, 51–60. https://doi.org/10.1145/2364489.2364498

López, P., & Baydal, E. (2018). Teaching high-performance service in a cluster computing course. *Journal of Parallel and Distributed Computing*, *117*, 138–147. https://doi.org/10.1016/j.jpdc.2018.02.027

Memeti, A., Imeri, F., & Cico, B. (2018). *REST vs. SOAP: Choosing the best web service while developing in-house web applications* (Vol. 2).

Nah, F. F.-H. (2004). A study on tolerable waiting time: how long are Web users willing to wait? *Behaviour & Information Technology*, *23*(3), 153–163. https://doi.org/10.1080/01449290410001669914

Pratama, K. A., Subagio, R. T., Hatta, M., & Asih, V. (2021). Implementasi Laod Balancing pada Web Server Menggunakan Apache dengan Server Mirror Data Secara Real Time. *Jurnal Digit*, *11*(2), 178. https://doi.org/10.51920/jd.v11i2.203

R. M. Zebari, S., & O. Yaseen, N. (2011). Effects of Parallel Processing Implementation on Balanced Load-Division Depending on Distributed Memory Systems. *Journal of University of Anbar for Pure Science*, *5*(3), 50–56. https://doi.org/10.37652/juaps.2011.44313

Rahmatulloh, A., & MSN, F. (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

Rijayana, I. (2005). Teknologi Load Balancing untuk Mengatasi Beban Server. *Seminar Nasional Aplikasi Teknologi Informasi (SNATI)*. https://journal.uii.ac.id/Snati/article/view/1385

Shukla, P., & Kumar, A. (2018). CLUE Based Load Balancing in Replicated Web Server. *2018 8th International Conference on Communication Systems and Network Technologies (CSNT)*, 104–107. https://doi.org/10.1109/CSNT.2018.8820224