Coding : Jurnal Komputer dan Aplikasi
Volume 07, No.01 (2019), hal. 33-42 ISSN : 2338-493X

IMPLEMENTASI *HEALTH CHECKS* MONITORING PADA *LOAD BALANCER* DI SISI *WEB SERVER* BERBASIS ANDROID

[1]Bagus Jati Kuncoro, [2]Ikhwan Ruslianto, [3]Syamsul Bahri
[1][2] [3]Jurusan Rekayasa Sistem Komputer, Fakultas MIPA Universitas Tanjungpura
Jl. Prof. Dr. H. Hadari Nawawi, Pontianak
Telp./Fax.: (0561) 577963
e-mail:
[1]bjkuncoro@untan.ac.id, [2] ikhwanruslianto@siskom.untan.ac.id,
[3]syamsul.bahri@siskom.untan.ac.id

Abstrak

Aplikasi web yang semakin meningkat penggunaannya dewasa ini menyebabkan kebutuhan akan kinerja web server yang memiliki ketersediaan serta kecepatan tinggi semakin meningkat. Teknologi load balancer yang merupakan salah satu metode yang dapat meningkatkan ketersediaan dan kecepatan kinerja web server. Pada penerapan teknologi load balancer tersebut tetap dapat terjadi masalah seperti, kegagalan web server dan lain – lain yang dapat mengurangi kinerja arsitektur web itu sendiri. Penelitian ini ditujukan untuk membangun sistem arsitektur web server load balancer yang memiliki sistem health check di dalamnya dan dapat dilakukan pemantauan secara real-time berbasis android guna meminimalisir masalah pada web server. Sistem dibuat dengan membangun arsitektur load balancer menggunakan modul pada nginx web server. Pada setiap server dibuat sebuah sistem pemantauan menggunakan pemrograman Node Js. Seluruh informasi terkait ketersediaan sumber daya pada server dipantau melalui sistem tersebut dan kemudian diolah untuk dikirim pada aplikasi klien menggunakan metode WebSocket client-server. Data yang diperoleh dari sistem pemantauan diolah pada sisi aplikasi untuk dapat dianalisa dan dipantau kondisi server tersebut secara real-time. Sistem pada aplikasi dapat membaca nilai data server dan memberikan peringatan atau penanganan otomatis pada server yang terjadi masalah berdasarkan pemantauan real-time pada aplikasi android. Riwayat masalah pada server juga dapat dilihat pada sisi aplikasi android tersebut. Pengujian dilakukan dengan 5 skenario yang menggambarkan berbagai masalah yang mungkin terjadi pada server. Hasil pengujian menghasilkan pengaruh penggunaan load balancer bagi server dari segi peningkatan kecepatan penanganan request sebesar 67% pada pengujian pertama dan 72% pada pengujian kedua. Sistem health check berhasil mendapatkat data sumber daya server dan mencegah pengiriman request kepada server yang bermasalah.

Kata Kunci: *Health Check Server, Load Balancer, Sistem Android, Nginx Server, Node Js, Web Socket*

1. PENDAHULUAN

Penggunaan web server pada sebuah arsitektur aplikasi web merupakan penunjang utama pada aplikasi web tersebut. Web server bertugas sebagai penerima *request* dari klien, mengolah data *request* dan mengirim *respone* balik berdasarkan *request* yang dikirim oleh klien. Kinerja web server berpengaruh pada ketersediaan layanan dan kecepatan dari layanan aplikasi web server dalam penggunaannya memiliki kerentanan terhadap kegagalan dalam melalukan layanan kepada klien. Masalah terjadi ketika suatu *single server* mengalami kegagalan, hal ini dapat disebabkan oleh klien dengan jumlah yang mencapai ribuan bahkan jutaan yang mengakses *server* tersebut dalam waktu yang bersamaan, biasa disebut dengan *overload request*.

Teknologi yang dapat digunakan dalam menyelesaikan masalah ini salah satunya adalah penggunakan metode *load balancer* dan sistem *health check*. Metode

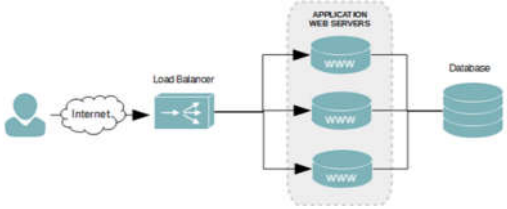
load balancer adalah metode membagi beban server kepada beberapa server lain nya yang berada dalam satu *cluster*. Sistem *health check* adalah sistem pemantauan sumber daya dan ketersediaan server dalam sebuat arsitektur. Supramana pada tahun 2012 membuat implementasi *load balancer* dengan web server Apache [1]. Sistem menggunakan web server Apache dan tidak memiliki sistem *health check* server yang dapat dipantau kapanpun dimanapun. Salah satu contoh penelitian yang meneliti sebuah sistem pemantauan pada server adalah Ray Endang yang membuat pemantauan pada web server[2]. Namun penelitian tersebut tidak menggunkana sistem *load balancer* di dalamnya.

Berdasarkan permasalahan dari latar belakang maka dibuatlah sistem Implementasi *Health Checks Monitoring* pada *Load Balancer* Web Server berbasis android sebagai solusi dalam memantau jalan nya *server* beserta *load balancer* dalam menjalankan fungsinya. Sehingga jika terjadi masalah dapat langsung diketahui dan ditangani secara cepat sehingga sistem tetap berjalan baik

2. DASAR TEORI

2.1 Load Balancer

Secara konsep, load balancer adalah jembatan antara server dan jaringan. load balancer memahami banyak protokol lapisan, yang lebih tinggi, sehingga mereka dapat berkomunikasi dengan server secara cerdas[3]. Arsitektur *load balancer* secara umum dapat dilihat pada Gambar 1.



Gambar 1. Arsitektur Umum Load Balancer.

Penelitian ini akan menggunakan metode distribusi round robin. Cara kerja sistem ini adalah dengan mendistribusikan *request* client ke *backend server* secara berurutan. Pada penjadwalan tipe round-robin, manager mendistribusikan client request sama rata ke seluruh real *server* tanpa memperdulikan kapasitas *server* ataupun beban request. Jika ada tiga real *server* (A,B,C), maka request 1 akan diberikan manager kepada *server* A, request 2 ke *server* B, request 3 ke *server* C dan request 4 kembali ke *server* A. Mekanisme ini dapat dilakukan jika seluruh real *server* menggunakan spesifikasi komputer yang sama[4].

Algoritma ini pada Nginx membagi beban secara bergiliran dan berurutan dari satu *server* ke *server* lain sehingga membentuk putaran.

2.2 Health Check

Sistem *health check* merupakan mekanisme yang diterapkan terhadap sebuah *server* sebagai penerima *request*. Penanganan terhadap kondisi *server* yang di lakukan dengan mekanisme *health check* berupa pemantauan berkala pada sebuah *server* dalam menangani *request.* Untuk pemeriksaan kesehatan *server*, memantau transaksi saat terjadi, dan mencoba melanjutkan koneksi yang gagal. Jika transaksi masih belum bisa dilanjutkan, mekanisme akan menandai *server* sebagai tidak tersedia dan sementara menghentikan pengiriman permintaan ke sana sampai ditandai aktif kembali. Mekanisme tersebut jika tidak dilaksanakan akan membuat *request* terus dikirim ke *server* yang bermasalah dan akan memberikan *respone* yang salah pada user.

Proses-proses yang terjadi pada suatu sistem monitoring dimulai dari pengumpulan data seperti data dari *host status, traffic information,* dan lain-lain yang kemudian data tersebut dianalisis pada proses analisis data dan pada akhirnya data tersebut akan ditampilkan dan disatukan dengan mekanisme modul *health check* dari Nginx.

2.3 Web Socket

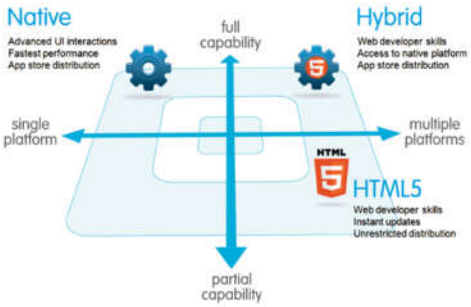
WebSocket adalah standar baru untuk komunikasi *realtime* pada Web dan aplikasi mobile. WebSocket dirancang untuk diterapkan di browser web dan *server* web, tetapi dapat digunakan oleh aplikasi client atau *server*. WebSocket adalah protokol yang menyediakan saluran komunikasi full-duplex melalui koneksi TCP tunggal.

Websockets menyediakan sebuah pendekatan untuk melakukan komunikasi *full duplex* tanpa *overhead* dari *header* http dan itu berarti beberapa perbaikan kinerja serius, terutama untuk aplikasi yang membutuhkan *update real-time* yang cepat. Sederhananya, HTTP dirancang untuk menjadi protokol tanpa kondisi dinamis dan bukan untuk komunikasi *full-duplex real-time*. Dan saat itulah Websockets terwujud[5].

2.4 Android Hybrid Application

Aplikasi *hybrid* adalah aplikasi web yang ditransformasikan menjadi kode native pada platform seperti iOS atau Android. Aplikasi *hybrid* biasanya menggunakan browser untuk mengijinkan aplikasi web mengakses berbagai fitur di device mobile seperti *Push Notification, Contacts,* atau *Offline Data Storage*. Beberapa tools untuk mengembangkan aplikasi hybrid antara lain Cordova, Rubymotion dan lain-lain.

Adapun platform Android yang menggunakan bahasa pemrograman Java. Membangun aplikasi native harus menyediakan pengalaman produk yang optimal pada perangkat mobile. Meskipun begitu, budget yang tinggi dibutuhkan untuk membangun aplikasi cross platform yang mampu mempertahankan aplikasi native tetap update. Konsep dari aplikasi *hybrid* dapat dilihat pada Gambar 2.



Gambar 2. Konsep dasar Android Hybrid Aplication
(sumber : codepolitan.com)

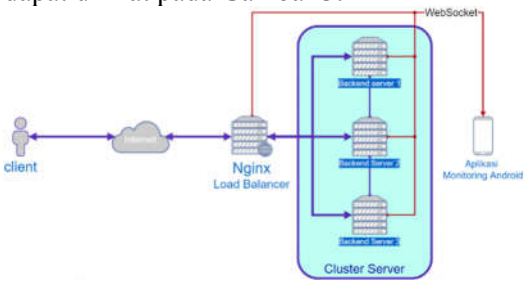
3. METODOLOGI PENELITIAN

Proses penelitian dimulai dengan melakukan studi pustaka terkait server, system kerja *load balancer*, konfigurasi arsitektur web server, dan konsep aplikasi *hybrid* pada android dalam melayani layanan. Selanjutnya dilakukan perancangan dari keseluruhan sistem yang akan diimplementasikan dalam bentuk komunikasi antara system pemantauan *load balancer* pada server dengan antar muka pada aplikasi android. Selanjutnya melakukan pengujian untuk mengetahui kinerja dan pengaruh system. Pengujian dilakukan dengan beberapa scenario uji berupa pengecekan pembagian beban *load balancer*, system pemantauan *health check,* komunikasi dan fitur pada antarmuka aplikasi android. Setelah dilakukan pengujian dilakukan analisa untuk mendapatkan kesimpulan akhir dari proses penelitian.

4. PERANCANGAN SISTEM

4.1 Rancangan Sistem

Rancangan sistem secara keseluruhan dapat dilihat pada Gambar 3.



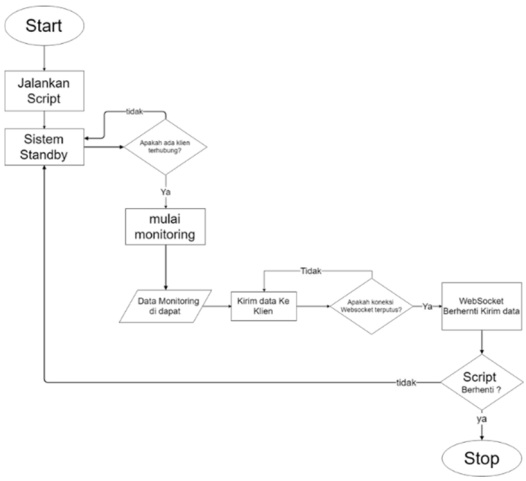
Gambar 3. Arsitektur Keseluruhan Sistem

Server Nginx *load balancer* akan bertindak sebagai penerima *request* dari klien melalui internet kemudian *load balancer* akan membagi setiap *request* yang datang dengan cara melempar *request* tersebut ke *cluster server* yang berisi tiga buah *backend server* sebagai penampung *request* dari klien dan memberikan *respone* dari *request* tersebut. *Respone* yang akan di berikan ke klien akan dikirim ke *server load balancer* terlebih dahulu dan kemudian akan di kirim langsung ke klien melalui jaringan. Konfigurasi *load balancer* pada Nginx memiliki sistem *health check* yang akan menangani masalah jika salah satu *backend*

server bermasalah (*down).* Masing – masing *server* pada arsitektur tersebut juga akan menjadi WebSocket *server* yang akan selalu mengirim data terkait pemantauan ke aplikasi Android secara *real-time.* WebSocket akan menjadi media komunikasi antara aplikasi dan arsitektur di sisi *server.* Data yang diterima oleh aplikasi akan diolah dan ditampilkan pada antarmuka aplikasi.

4.2 Perancangan Sistem Pemantauan Pada Server

Sistem ini merupakan penerapan dari pemantauan yang menggunakan modul Node Js dan mekanisme WebSocket sebagai komunikasi antara klien dan *server*. Pembacaan dan pengiriman data merupakan bagian utama pada tahap ini. Agar sistem dapat berfungsi dan berjalan sesuai dengan yang dirancang, maka diperlukan alur sistem seperti pada Gambar 4.



Gambar 4. Diagram Alir Pemantauan

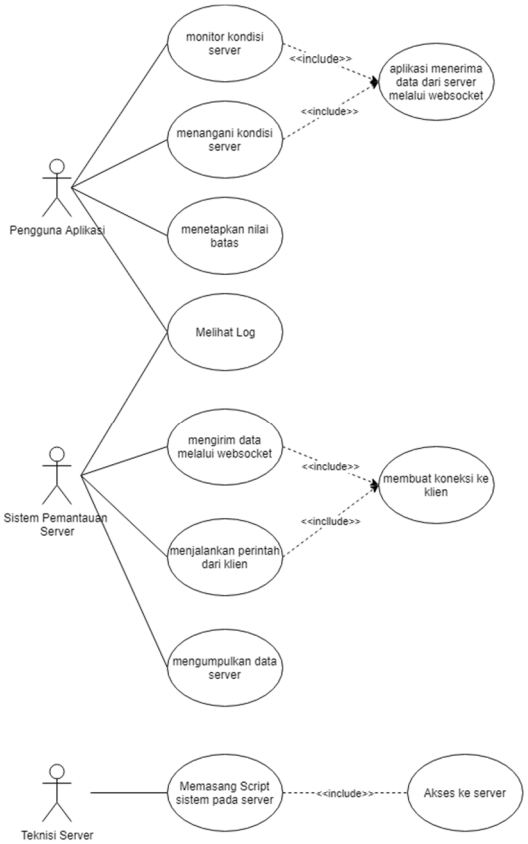
Sistem *monitoring* akan berada disetiap *server* untuk membaca penggunaan CPU, memori dan *traffic request* pada setiap *server.* Namun untuk pemantauan kondisi *respone server* hanya akan terdapat pada *load balancer server.* Masing – masing *server* tersebut juga dibuat sebagai WebSocket *server* dimana data dari masing - masing *server* akan dikirim ke WebSocket klien(Android).

4.3 Perancangan Aplikasi *Hybrid* pada Android

Aplikasi pada *platform* Android ini berfungsi sebagai penyedia data hasil pemantauan ke pengguna sekaligus komunikasi antara sistem dan pengguna melalui antarmuka aplikasi. Aplikasi Android menjadi tujuan akhir data yang dikirim oleh WebSocket. Antarmuka aplikasi menampilkan seluruh data server, status, jumlah *request* dan lain-lain. Data yang masuk diolah sesuai logika untuk menjalankan beberapa fitur yang disediakan oleh aplikasi seperti sistem notifikasi, perintah *remote* server, dan *remote* otomatis jika mencapai kondisi tertentu. Melalui aplikasi user dapat mengetahui kondisi setiap server dan keseluruhan arsitektur tanpa harus masuk kedalam server tersebut serta dapat melakukan *remote* sederhana terkait kondisi server yang dipantau.

4.3.1 *Use Ca se Diagram* Sistem

Use case diagram ditunjukkan pada Gambar 5.



Gambar 5. Use case diagram *sistem*

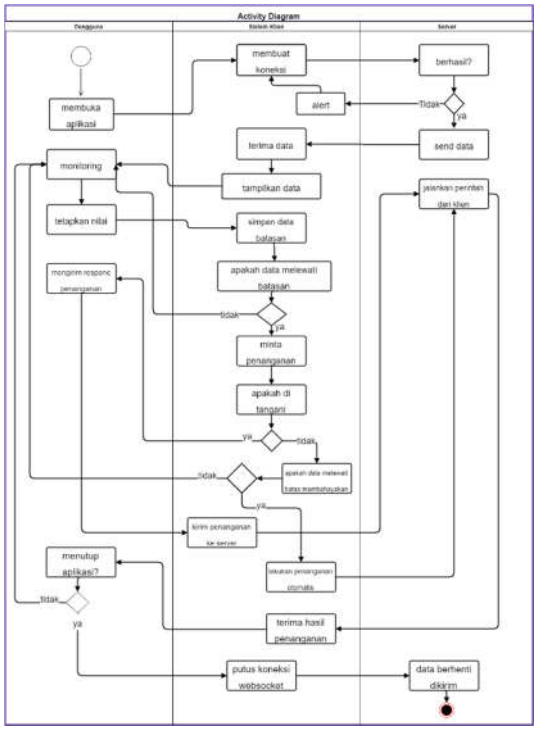
Dalam tahap perancangan sistem digunakan *use case diagram* untuk menggambarkan fungsionalitas sistem yang dilakukan aktor. Terdapat tiga aktor pada sistem ini yaitu pengguna aplikasi, teknisi server, dan sistem pemantauan. Aplikasi

dapat digunakan oleh pengguna yang telah memasang aplikasi pada perangkat Android dan koneksi WebSocket telah terjadi.

Pengguna dapat menggunakan aplikasi jika sistem telah berhasil melakukan koneksi ke WebSocket server. Penggunaan aplikasi oleh pengguna mencakup menetapkan nilai penggunaan maksimal sumber daya server yang di gunakan untuk sistem notifikasi, pemantauan server, dan melakukan *remote* server sederhana dari aplikasi.

4.3.2 *Activity Diagram* Sistem

Activity Diagram ditunjukkan pada Gambar 6.



Gambar 6. *Activity Diagram* sistem

Alir dari aktifitas yang dilakukan pada *activity diagram* adalah:

1. Pengguna memulai penggunaan aplikasi yang sekaligus membuka koneksi ke WebSocket server.

2. Jika koneksi berhasil maka server akan melakukan monitor dan mengumpulkan data hasil pemantauan kemudian dikirim ke klien. Jika koneksi gagal maka sistem akan memberi peringatan dan mencoba untuk terus melakukan koneksi ulang.

3. Apabila data telah dikirim maka tugas sistem adalah mengolah data di sisi klien yang kemudian ditampilkan pada antarmuka aplikasi untuk pengguna melakukan monitoring.

4. Selanjutnya pengguna menentukan batasan nilai penggunaan yang jika di lewati akan memberi peringatan dalam bentuk notifikasi.

5. Sistem menyinpan data masukan dari pengguna dan melakukan pengecekan setiap data masuk dari server.

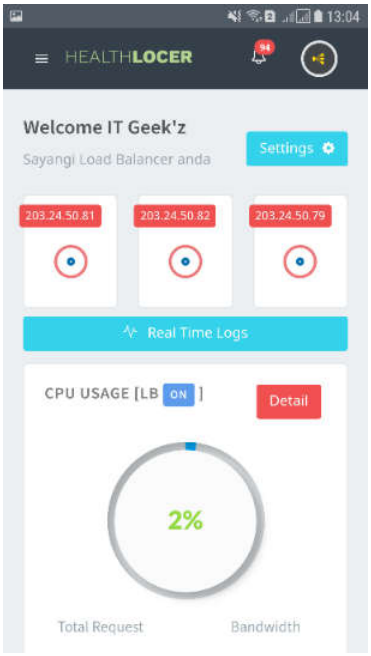
6. Jika nilai data melebihi batas penggunaan maka sistem akan meminta pengguna melakukan penanganan. Jika penanganan tidak dilakukan dan data melewati batas membahayakan maka sistem akan mengambil penanganan otomatis terhadap server.

7. Selanjutnya pengguna melanjutkan monitoring terhadap data. Jika aplikasi ditutup maka koneksi WebSocket akan tertutup dan berhenti mengirim data.

5. IMPLEMENTASI DAN PENGUJIAN SISTEM

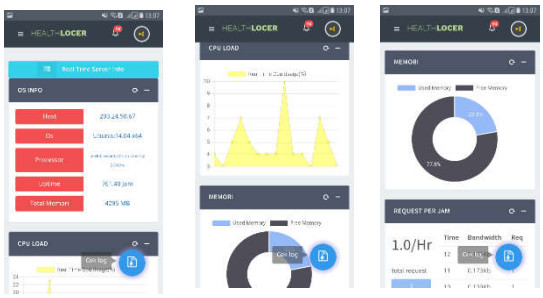
5.1 Implementasi Apikasi Pemantauan pada Android

Hasil dari perancangan aplikasi pemantauan pada android dapat diakses pada pada perangkat yang telah terpasang aplikasi berfomat *.apk di perangkat tersebut. aplikasi yang dijalankan akan menampilkan *splash screen* dalam beberapa detik pertama sebagai waktu inisiasi membuat koneksi ke server, mengambil data, dan menampilkan data tersebut pada antarmuka aplikasi. Setelah seluruh data berhasil diperoleh dari sisi server aplikasi akan menampilkan antarmuka halaman utama aplikasi seperti yang diperlihatkan pada Gambar 7.



Gambar 7. Halaman Utama Aplikasi

Pada label alamat di setiap *cardbox* jika ditekan maka akan mengarah pada halaman detail dari server yang berisi informasi sistem beserta rincian pemantauan yang terjadi secara *real-time.* Halaman terdiri dari beberapa *portlet widget* yang masing – masing nya memiliki informasi dari hasil pemantauan server, informasi yang ditampilkan pada halaman ini antara lain os info, *realtime* cpu *load,* menggunaan memori, *request* perjam, riwayat http status, *network traffic,* serta daftar pengakses server. Antarmuka halaman dapat dilihat pada Gambar 8.



Gambar 8. Detail Info Sumber Daya Server

5.2 Pengujian Arsitektur *Load Balancer* dan *Health Check*

Pengujian arsitektur bertujuan untuk mengetahui fungsi *load balancer* dan penerapan *health check* dapat berjalan dan diterapkan dengan baik.

5.2.1 Pengujian *Load Balancer*

Pengujian pada load balancer dilakukan dengan melakukan skenario uji, mensimulasikan server diakses oleh beberapa klien. Untuk melihat perubahan server yang melayani klien maka setiap server akan diberi penamaan berbeda pada konten *respone.* Hasil yang diperoleh dari hasil mengakses IP server *load balancer* 3 kali berturut turut didapat hasil sebagai berikut yang ditujukan pada Gambar 9.

Gambar 9. Hasil Akses ke Server *Load Balancer.*


```
{
  host: "SERVER 1",
  status: "200",
  message: "success",
```

```
{
  host: "SERVER 2",
  status: "200",
  message: "success",
```

```
{
  host: "SERVER 3",
  status: "200",
  message: "success",
```


Dari hasil pengujian tersebut didapat hasil dari masing – masing pengaksesan. *Respone* yang diperoleh dari hasil berupa data dalam format JSON yang telah disiapkan pada setiap *backend* server, namun terlihat perbedaan *value* pada *key* 'host' yang menunjukan setiap akses yang dilakukan ditangani oleh server yang berbeda. Dari hasil tersebut terlihat bahwa sistem *load balancer* bekerja sesuai mekanisme.

5.2.2 Pengujian *Health Check*

Pengujian pada mekanisme *health check* dilakukan dengan melakukan simulasi jika *backend* server mati atau bermasalah. *Request* akan dikirim ke *load balancer* server pada waktu server melayani *request* salah satu atau beberapa *backend server* akan dimatikan sebagai simulasi jika pada kondisi asli terjadi masalah pelayanan pada server. Pengujian dilakukan saat kondisi seluruh server berjalan baik, kemudian diberi *request* sebanyak 100, 200, dan 300 ke *load balancer server*, ketika *load balancer* melayani *request* yang sedang diberikan salah satu server dimatikan sebagai indikasi jika server tidak dapat melayani *request*

yang datang. Data hasil pengujian dapat dilihat pada Tabel 2.

Tabel 2. Kondisi server *load balancer* menerima *request* dengan salah satu server mati

test	Host *backend Server*		
	203.24.50.81	203.24.50.82	203.24.50.79
100	24	38	38
200	28	88	84
300	66	115	119

Dapat dilihat hasil dari uji coba pertama yang dilakukan pada *load balancer* server yang menangani *request* disaat salah satu server mati. Pada tabel diperlihatkan hasil pembagian beban *request* pada setiap *backend server* disaat melayani 200 *request.* Pembagian hasil beban terlihat tidak berimbang hal tersebut dikarenakan pada saat melayani *request* server 1 dimatikan guna melihat mekanisme *health check* yang diterapkan.

Sesuai dengan kinerja *health check* bahwa setiap server yang mengalami kegagalan sama dengan *max_fails* maka server tersebut akan ditandai sebagai *unhealth/unavailable* dan server *load balancer* tidak akan mengirim *request* sampai waktu yang ditentukan pada *fail_timeout.* Pengujian dengan mengirim 100 *request* dilakukan dalam jangka waktu 60 detik dari pengujian pertama guna melihat apakah mekanisme *health check* bekerja dengan baik. Data hasil pengujian dapat dilihat pada Tabel 3.

Tabel 3 Kondisi pembagian beban setelah pengujian pertama

test	Host *backend Server*		
	203.24.50.81	203.24.50.82	203.24.50.79
100	0	50	50
200	0	100	100
300	0	150	150

5.2.3 Pengujian Waktu *Respone* dan Kondisi Server Saat Menerima *Request*

Pengujian ini dilakukan dengan tujuan membandingkan waktu *respone* dari server terhadap *request* klien saat ditangani oleh *load balancer* dan saat ditangani oleh server tunggal. Simulasi yang dilakukan dengan mengirim request pada server dengan permintaan data mahasiswa dalam bentuk JSON dengan jumlah 7385 data dalam sekali *request.* Hasil pengujian dapat dilihat pada Tabel 4 berikut.

Tabel 4. Hasil Pengujian Kecepatan Server

Test	Tunggal			Cluster		
	respone time (s)	req/ sec	time/req(ms)	respone time (s)	req/ sec	time/req(s)
100	20,0	4,98	803	6,49	15,4	259
500	98,0	5,10	784	30,5	16,3	244
1000	186	5,37	744	52,0	19,2	208

Berdasarkan Tabel 4 pengujian dengan jumlah *request* 100 pada saat ditangani server tunggal memiliki waktu *respone* 20,08 detik sedangkan saat ditangani oleh sistem *load balancer* memiliki waktu *respone* 6,49 detik dari perbandingan tersebut diperoleh bahwa sistem *load balancer* memiliki waktu *respone* 67 % lebih cepat dibanding server tunggal. Pengujian selanjutnya dengan jumlah *request* 500 dan 1000 juga mengalami peningkatan waktu *respone* yaitu masing – masing 68 % dan 72 %. Berdasarkan hasil pengujian dapat dilihat bahwa penggunaan *load balancer* berhasil dilakukan dan berpengaruh terhadap kecepatan kinerja pelayanan terhadap klien.

5.3 Pengujian Aplikasi Pemantauan Android

Pengujian aplikasi pada android dilakukan untuk mengetahui kinerja dari sistem yang dirancang pada sisi klien. Aplikasi pada android akan bertintak sebagai websocket klien yang menerima data dan mengolah seluruh data yang diperoleh dari server dengan metode yang telah diracang. Pengujian dilakukan dengan melakukan beberapa skenario uji yang diatur sesuai dengan kemampuan aplikasi. Pengujian dilakukan pada sistem koneksi, penerimaan data, permintaan data, pengujian

jika terjadi masalah pada server, pengujian pada sistem notifikasi, dan sistem penanganan otomatis oleh aplikasi.

5.3.1 Pengujian Jika Webserver Mati

Pengujian kali ini dilakukan dengan melakukan skenario uji berupa simulasi jika webserver pada arsitektur dalam kondisi mati secara tiba-tiba sehingga tidak dapat melakukan layanan pada klien. Pengujian dianggap berhasil jika setiap perubahan kondisi pasa setiap server dapat ditampilkan pada aplikasi dan sistem aplikasi dapat memberi pemberitahuan pada perangkat. Data hasil pengujian sistem pada kondisi web server dapat dilihat pada Tabel 6.

Tabel 6. Pengujian web Server mati

kondisi server uji			indikator pada aplikasi		
server 1	server 2	server 3	server 1	server 2	server 3
mati	aktif	Aktif	✗	○	○
aktif	mati	Aktif	○	✗	○
aktif	aktif	Mati	○	○	✗
mati	mati	Aktif	✗	✗	○

5.3.2 Pengujian Jika Sumber Daya Melewati Batas

Pengujian jika sumber daya server melewati batas dilakukan dengan melakukan simulasi pemberian beban besar pada server sehingga sumber daya pada server akan digunakan secara besar-besaran kemudian akan dilihat cara kerja aplikasi berdasarkan kondisi tersebut. Beban sumber daya yang akan diuji antara lain penggunaan cpu dan memori pada server. Pengujian akan berhasil jika sumber daya server dapat ditampilkan pada aplikasi dan aplikasi dapat memberikan pemberitahuan pada perangkat. Data hasil pengujian dapat di lihat pada Tabel 7.

Tabel 7. Hasil Pengujian Sumber Daya Server

Simulasi	Server	Keluaran berdasarkan waktu *respone*(s)	
		peringkatan ditampilkan oleh antar muka	pemberitahuan
cpu melebihi batas	server 1	1,05	2,06
	server 2	0,9	1,98
	server 3	1,00	2,1
	server lb	1,3	2,05
memori melebihi batas	server 1	0,99	2,23
	server 2	1,15	2,09
	server 3	1,50	2,07
	Server lb	0,98	2,03

Pengujian dilakukan pada setiap server dalam arsitektur. Berdasarkan hasil pengujian dapat dilihat sistem pemantauan sumber daya pada aplikasi dapat berjalan dengan sesuai dengan perancangan.

5.3.3 Pengujian Sistem Penanganan pada Aplikasi

Pada pengujian pertama dilakukan dengan menguji fitur penanganan yang ada pada aplikasi, fitur yang diuji antara lain *restart service, reboot service,* dan *autoclean*. Pengujian berhasil jika setiap perintah yang dikirim ke server dapat berjalan dan dieksekusi dengan baik. Data hasil pengujian dapat dilihat pada Tabel 8.

Tabel 8. Hasil Pengujian fitur Penanganan dari aplikasi

server	fitur yang diuji berdasarkan waktu *respone*(s)		
	restart service	reboot server	autoclean
server lb	0,96	3,6	1,55
server 1	0,98	4,01	1,47
server 2	0,89	3,87	1,89
server 3	1,02	4,6	2,05

Pengujian kedua dilakukan untuk mengecek kinerja sistem penanganan otomatis jika sumber daya telah melewati batas penanganan maka dari itu dibuat skenario uji berupa beban berlenih yang akan ditanggunng server. Pengujian berhasil jika aplikasi berhasil melakukan penanganan otomatis tanpa sentuhan pengguna dan server dapat mengeksekusi perintah yang dikirim dengan baik. Data hasil pengujian dapat dilihat pada Tabel 9.

server	Batas penanganan (%)	penggunaan cpu(%)	penanganan otomatis(s)
server lb	60	68	3,6
server 1	60	87	2,9
server 2	60	61	3,09
server 3	60	70	4,2

Berdasarkan tabel 8 dan 9 dapat dilihat seluruh perintah dan penanganan

yang diberikan dari klien dapat dijalankan dengan baik, dan penanganan otomatis dapat berjalan sesuai dengan perancangan yang telah dibuat. Percobaan di lakukan pada setiap server yang ada pada arsitektur. Berdasarkan hasil yang diperoleh dapat dilihat sistem penanganan pada aplikasi berhasil berjalan sesuai rancangan

6. KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan hasil percobaan implementasi *health check monitoring* pada *load balancer* berbasis android didapatkan hasil sebagai berikut:

1. Arsitektur load balancer pada server dibuat menggunakan *ngx_http_upstream_module* yang terdapat pada web server nginx pada port 81 sebagai pembagi beban ke setiap *backend server*. Server *load balancer* pada penelitian ini dibuat pada server dengan alamat IP 203.24.50.67 dan masing – masing *backend server* pada alamat IP 203.24.50.81, 203.24.25.82, dan 203.24.50.79.
2. Sistem *health check monitoring* pada setiap server berhasil dibuat pada sisi server sehingga berpengaruh pada ketersediaan layanan server. Sistem dapat meninimalisir kegagalan penyediaan layanan pada server yang bermasalah berdasarkan informasi yang dikirim ke aplikasi android. Pada pengujian sistem dapat menandai server yang bermasalah sebagai '*unavailable'* sehingga tak akan ada *request* yang dikirim pada server tersebut.
3. Penggunaan *load balancer* pada web server berpengaruh pada ketersediaan layanan web server dan mempengaruhi peningkatan kecepatan layanan pada aplikasi web server. Pada percobaan dengan 500 *request* waktu yang dibutuhkan pada server tunggal sebesar 98,01s dan kecepatan *respone* meningkat sebesar 67 % pada penggunaan load balancer dengan memakan waktu sebesar 30,52s. Pada percobaan dengan 1000 *request* waktu yang dibutuhkan pada server tunggal sebesar 186,18s dan kecepatan *respone* meningkat sebesar 72 % pada penggunaan load balancer dengan memakan waktu sebesar 52,02s
4. Komunikasi data antara perangkat android dan arsitektur server dilakukan menggunakan metode websocket antara server sebagai websocket server dan perangkat android sebagai websocket klien sebagai penerima data secara *real-time.*
5. Aplikasi pemantauan kondisi arsitektur server pada perangkat android berhasil dibuat dengan fitur – fitur yang dapat menampilkan antarmuka berisi informasi server, dapat melakukan penangan terhadap server yang bermasalah, hasil rekap jumlah *request*, dan pengecekan log pada arsitektur.
6. Kendala terjadi pada saat menggunakan fitur cek log untuk mencari riwayat kondisi server, dimana terdapat beberapa waktu yang tidak memiliki data di karenakan sistem pemantauan pada server tidak berjalan. Tidak berjalannya pemantauan dikarenakan server tidak mendapat koneksi websocket dari klien yang merupakan pemacu dari dijalankannya sistem pemantauan pada server.

6.1 Saran

Berdasarkan penelitian yang telah dilakukan pada implementasi *health check monitoring* pada *load balancer* berbasis android maka diperoleh saran untuk penelitian lebih lanjut yaitu:

1. Penggunaan server pada arsitektur untuk mengatur agar seluruh *backend server* tidak berada dalam tempat atau jaringan yang sama agar tidak terjadi kegagalan jika terjadi masalah pada koneksi di jaringan tersebut.
2. Penelitian selanjutnya menerapkan *failover* pada arsitektur server *load balancer* jika server load balancer mengalami masalah.
3. Melakukan penelitian lanjutan menggunakan berbagai macam arsitektur agar dapat dibandingkan hasil dan ketersediaan layanan pada setiap arsitektur yang berbeda.

DAFTAR PUSTAKA

[1] I. G. L. P. E. Supramana, Prismana, "Implementasi Load Balancing Pada Web Server Dengan Menggunakan Apache," *J. Manaj. Inform.*, vol. 5, pp. 117–125, 2016.

[2] E. Ray, *Pengembangan Aplikasi Monitoring Server Berbasis Mobile Web Dengan Sistem Notifikasi Universitas Islam Negeri Syarif Hidayatullah Jakarta*. 2015.

[3] C. Kopparapu, *Load Balancing Servers , Firewalls , and Caches*. .

[4] G. Triono, "Implementasi Load Balancing Dengan Menggunakan Algoritma Round Robin Pada Kasus," pp. 169–176, 2015.

[5] V. Pimentel and B. G. Nickerson, "Communicating and displaying real-time data with WebSocket," *IEEE Internet Comput.*, vol. 16, no. 4, pp. 45–53, 2012.