Implementasi Load Balancing NGINX dan Mongodb Cluster serta Mekanisme Redis Caching

Faisal Al Isfahani
Informatic
Siliwangi University
Tasikmalaya
177006047@student.unsil.ac.id

Fuji Nugraha
Informatic
Siliwangi University
Tasikmalaya
177006052@student.unsil.ac.id

Abstract—*Meningkatnya kebutuhan akan informasi menuntut akses yang cepat untuk mendapatkan informasi – informasi terkini, salah satunya yang paling mempengaruhi kecepatan akses suatu alamat website tertentu adalah server penyedia layanan. Server merupakan penyedia layanan sekaligus sebagai pusat pemrosesan data sehingga menuntut kinerja server secara maksimal agar dapat memberikan pelayanan yang maksimal. Salah satu metode yang dapat digunakan yaitu Load balancing atau membagi beban (request) pada web server yang bertujuan untuk meringankan beban yang ditanggung oleh masing masing server, sehingga mampu meningkatkan kinerja server dengan ketersediaan (availability) layanan web server tetap. Namun terdapat pula faktor penting lain, meningkatnya jumlah data yang disertai dengan pengolahan seara terpusat akan berujung tidak optimal. Ketidak optimalan tersebut berupa nilai response time yang tinggi, maka dibutuhkan sebuah arsitektur basis data selain arsitektur terpusat. Salah satu alternative bisa menerapkan metode cluster database untuk meningkatkan ketersediaan basis data dan penerapan mekanisme cache dengan teknologi redis untuk meningkatkan response time terhadap request data serupa. Penelitian ini bertujuan untuk membuat arsitektur sistem terdistribusi dengan menerapkan load balancing web server serta database cluster dan mekanisme caching guna mengatasi permasalahan low availability (rendahnya ketersediaan) pada suatu sistem atau unit kerja.*

Keywords—Load Balancing, Basis data terdistribusi, redis cach, nginx, mongodb cluster

I. PENDAHULUAN

Perkembangan teknologi menjadi faktor utama meningkatnya pengguna layana web. Pesatnya perkembangan teknologi tentu diimbangi dengan bertambahnya pula kebutuhan pengguna yang membuat jumlah traffic website pupular menjadi tinggi. Peningkatan traffic ini tentu sangat memberatkan kinerja server dalam melayani permintaan yang ada. Akibat dari pembebanan kerja ini adalah performa server menjadi menurun dan sering terjadi gangguan pada layanan-ayanan web tersebut. Jika tidak ditangani dengan cepat dan baik tentunya ini akan mengakibatkan suatu sistem atau situs mati/down, yang berarti akan menghambat pengguna dalam mengakses informasi atau resource dalam suatu sistem atau situs [1]. adalah Selain permasalahan traffic yang tinggi, permasalahan lain yang muncul adalah ketersediaan infrastruktur teknologi penyedia atau pengelola data. Data biasanya di kelola dalam jumlah besar oleh server dalam waktu yang cenderung repetitif dan rutin [2].

Salah satu solusi untuk menangani masalah traffic tinggi adalah dengan menggunakan sebuah server yang handal atau memiliki performa yang inggi. Solusi lainnya adalah dengan menerapkan teknologi *load balancing*. Teknologi load balancing dapat membagi beban permintaan ke beberapa web server sehingga teknologi ini memiliki peranan pernting untuk mencapai penggunaan sumber daya bersama yang efektif [3]. Solusi selanjutnya yakni untuk mengatasi permasalahan pada pengolahan data yang sama diterapkan sebuah metode replikasi basis data. Sistem basis data akan dibangun dengan metode cluster dan antara satu node dengan node lain akan saling berhubungan dan saling berbagi resource serta saling membackup ketika salah satu atau lebih dari database core mengalami *down* atau *out of request*. Salah satu DBMS yang memiliki kemampuan basisdata terdistribusi adalah MongoDB. Penerapan teknik ini bbertujuan untuk meningkatkan performansi sistem terutama dalam aspek *availability* data pada sistem basis data [2], [4]. Solusi yang ketiga adalah untuk mengatasi masalah lambatnya response time ketika melakukan request data ke sistem basis data yang memuat lebih dari 1000 data , maka harus di terapkan metode cache dimana ketika request akan data yang sama datang data hasil request sebelumnya akan disimpan kedalam cache sehingga akan mengurangi request yang berlebihan akan data yang sama kedalam sistem basis data [5].

Pada penelitian ini bertujuan untuk membangun suatu arsitektur sistem yang mengimplementasikan pembagi beban server (load balancing) dan sebuah sistem database yang menggunakan metode cluster dan penerapan mekanisme cache dengan teknologi redis. Selain itu juga dilakukan analisa pada performa sistem ketika ditambahkan satu persatu teknologi. Dalam penelitian ini dimulai dengan studi literature, dilanjutkan dengan pengumpulan data, peranacngan sistem, implementasi, pengujian dan analisis hasil.

II. TINJAUAN PUSTAKA

A. Sistem Terdistribusi

Sistem terdistribusi adalah sekumpulan komputer otonom yang terhubung ke suatu jaringan, dimana bagi pengguna sistem terlihat sebagai satu komputer. Dengan menerapkan sistem terdistribusi, komputer dapat melakukkan : Koordinasi Aktifitas dan berbagi sumber daya (hardware, software dan data). Tujuan utama dari sistem terdistribusi adalah *high availability* (ketersediaan tinggi) yang hanya akan terwujud dengan berbagai mekanisme sistem terdistribusi mulai dari distribusi proses, data, method atau class, dll [6], [7].

B. Load Balancing

Penyeimbang beban atau load balancing adalah suatu proses untuk mendistribusian sebuah trafik pada web dari

beberapa server yang di akses melalui jaringan . Load balancing ini dimaksudkan untuk mengurangi beban kerja server dengan cara mendistribusikan beban kerja tersebut ke beberapa server yang tergabung dalam web cluster, sehingga tidak ada server yang overload bahkan hingga down. Walaupun terdapat down pada salah satu web server, dengan menerapkan load balancing ini traffik akan di alihkan pada web server yang available. Hal ini sangat dapat diandalkan terutama untuk mewujutkan suatu sistem yang high availability [1], [2], [8], [9].

C. *Web Server*

Web server adalah sebuah software yang memberikan layanan berbasis data dengan meggunakan protokol HTTP atau HTTPS dari web browser klien untuk request data dan server akan mengirimkan response berisikan resource yang diminta oleh client biasanya dalam bentuk HTML, CSS ataupun JS. Resource tidak hanya berisikan laman web namun juga bisa terdiri dari berkas teks, video, gambar, file dan banyak lagi [2].

Salah satu program dari web server adalah NGINX. NGINX adalah server HTTP yang gratis, opensource yang memiliki performa sangat tinggi. Selain itu NGINX memiliki fungsi lain seperti dapat berfungsi sebagai proxy server, reverse proxy dan juga NGINX dikenal karena memiliki performa yang snagat tinggi [8].

D. *Database Server*

Database server adalah aplikasi komputer yang menyediakan layanan data ke komputer atau program komputer dan memiliki fungsi sebagai tempat penyimpanan data, seperti yang ditetapkan oleh model klien-server. Istilah ini juga merujuk kepada sebuah komputer yang didedikasikan untuk menjalankan program server database [2]. Pada sistem database terdapat teknik untuk menduplikast sebuah kumpulan data yang terdapat pada suatu server dan data tersebut disistribusikan ke server yang lain didalam database cluster, sehingga dengan memanfaatkan metode replikasi ini akan dapat membuat suatu sistem basis data yang high availability dan fault tolerant seperti tujuan awal dari pembangunan arsitektur sistem terdistribusi [10].

III. METODE PENELITIAN

Metode yang digunakan dalam penelitian ini terdiri dari 4 tahap, sebagai berikut :

A. *Analisis dan Pengumpulan data*

Pada tahap ini dilakukan studi pustaka dari beberapa sumber terkait informasi load balancing dengan nginx serta replikasi database mongoDB dan penerapan mekanisme cache dengan teknologi redis. Informasi didapat dari jurnal maupun artikel media online dan juga terdapat pada dokumentasi resmi dari penyedia teknologi terkait

B. *Perancangan Sistem*

Para tahap ini dilakukan perancangan arsitektur sistem sesuai dengan hasil informasi dan pengetahuan yang didapat dari tahap sebelumnya. Rancangan arsitektur yang dibuat meliputi permodelan arsitektur sistem secara keseluruhan.

C. *Implementasi*

Rancangan arsitektur sistem yang telah dibuat sebelunya, pada tahap ini akan diimplementasikan ke dalam aplikasi dengan memanfaatkan teknologi informasi. Implementasi bermulai pada konfigurasi load balancing nginx, dilanjutkan dengan replikasi mongodb dan penerapan redis cache, kemudian pembuatan endpoint untuk request data dari client ke server.

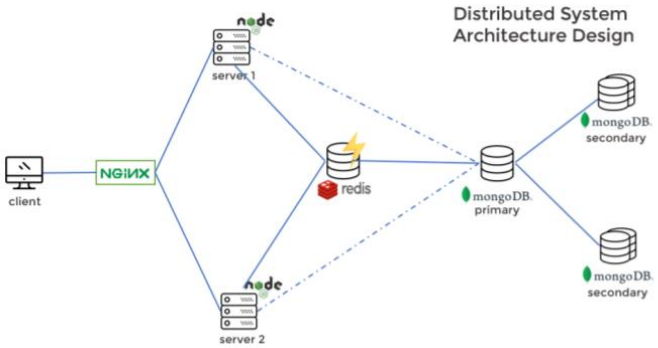
D. *Pengujian dan Analisis*

Pada tahap ini dilakukan pengujian terhadap hasil implementasi dari arsitektur sistem yang telah dibangun. Pengujian yang dilakukan adalah performance testing menggunakan Jmeter, pengujian dibentuk menjadi 2 case, yaitu : perbandingan sebelum dan sesudah pengimplementasian load balancing , kemudian case yang kedua adalah perbandingan antara database cluster yang menggunakan redis cache dan tanpa mekanisme cache.

IV. HASIL DAN PEMBAHASAN

A. *Perancangan arsitektur sistem*

Perancangan arsitektur siste yang akan dibangun untuk server load balancing denagan nginx dan replikasi database mongodb serta penerapan mekanisme redis cache menggunakan tujuh server dimana dua server berperan sebagai webserver, tiga server sebagai database server dan terhubung menjadi sebuah database cluster, satu buah untuk cache server dan satu sisanya sebagai load balancer atau pembagi beban. Perancangan arsitektur sistem dapat dilihat pada Gambar 1.



Gambar 1. Perancangan arsitektur sistem terdistribusi

Gambar 1 merupakan desain arsitektur sistem terdistribusi yang akan dibangun menggunakan load balancer nginx dengan nodejs web server, mongodb untuk database server dan redis untuk caching. Seperti yang terkandung pada gambar load balancer terhubung dengan 2 webserver yang bertanggung jawab sebagai pembagi beban request yang masuk dari user. Kemudian kedua webserver terhubung ke redis cache dan mongodb primary. Fungsi hubungan webserver dengan cache adalah ketika melakukan request data yang pertama kali dilakukan pengecekan adalah pada redis cache apakah ada cache dari request sebelumnya atau apakah dalam cache sedang tersimpan data yang sama dengan request yang sedang dilakukan, jika dirasa tidak ada maka request akan di teruskan ke database server atau mongodb primary. Database cluster terdiri dari 3 buah database server yang jika dilihat dari gambar terdiri dari 1 buah database server primary dan 2 buah database server

secondary yang fungsinya saling membackup dan mendistribusikan data antar satu dengan lainnya. Ketika database server primary down maka database secondary akan menggantikan peran primary hingga database primary kembali pulih.

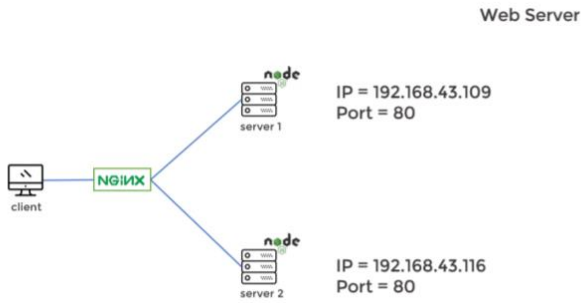
Spesifikasi teknis kebutuhan hardware dan Software bisa dilihat pada tabel 1.

Tabel 1. Spesifikasi kebutuhan hardware dan software

No	Deskripsi	Spesifikasi
Kebutuhan Hardware		
1	Load balancer	
	Spesifikasi	- 2 Ghz dual core processor - 2 GB RAM - Minimum internal storage 1GB
2	Web Server	
	Spesifikasi	- 1,6 GHz CPU - 1,75 GB RAM - 1 x 40 GB of free space
3	Database Server	
	Spesifikasi	- 1,6 GHz CPU - 1,75 GB RAM - 1 x 40 GB of free space
Kebutuhan Software		
4	NGINX	
	OS	Recomended Linux Ubuntu 16.04 LTS++
5	Node.Js	
	OS	Recomended Window 10
6	MongoDB	
	OS	Recomended Window 10
7	Redis	
	OS	Recomended Linux Ubuntu 16.04 LTS++

B. Konfigurasi web server

Konfigurasi webserver dapat dilihat pada gambar dibawah ini :



Gambar 2. Konfigurasi Webserver

Gambar 2 merupakan desain konfigurasi dari webserver dimana terdapat 2 buah webserver dan keduanya menggunakan webserver nodejs, spesifikasi konfigurasi sebagai berikut :

Server 1
Ip : 192.168.43.109
Port : 80
OS : Windows
W.server: Nodejs

Server 2
Ip : 192.168.43.116
Port : 80
OS : Windows
W.server: Nodejs

Load Balancing (NGINX)
Ip : 192.168.43.141
Port : 80
OS : Windows Subsystem Linux (Ubuntu)

C. Konfigurasi Load balancing

Load balancing menggunakan sistem operasi Ubuntu dan berjalan pada windows subsystem linux. Tahapa pertama untuk konfigurai load balancing aitu pembaruan repositori pada server dengan perintah apt –get update. Setelah proses update selesai lanjut ke proses peninginstallan paket nginx dengan perintah apt-get install nginx kemudian dilanjutkan dengan konfigurasi nginx yang terletak pada file /etc/nginx/sites-available/default file konfigurasi terdapat pada gambar berikut :

```
upstream backend {
        server 192.168.43.109:80; #server 1
        server 192.168.43.116:80; #server 2
}

server {
        listen 80;
        server_name 192.168.43.141;
        location / {
                proxy_pass http://backend;
        }
}
```

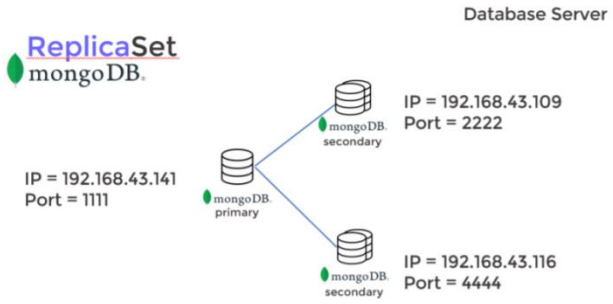
Gambar 3. Konfigurasi Load Balancing

Gambar 3 merupakan konfigurasi loadbalancing nginx. Pada gambar diatas terdapat upstream dengan ip diisi dengan ip web server cluster yaitu web server 1 dan web server 2. Kemudian terdapat server section yang diisi oleh port server load balancing dan ip dari server load balancing. Pemilihan algoritma load balancing terdapat pada upstream

section dimana jika tidak di inputkan algoritma maka akan menggunakan algoritma default yaitu round robbin.

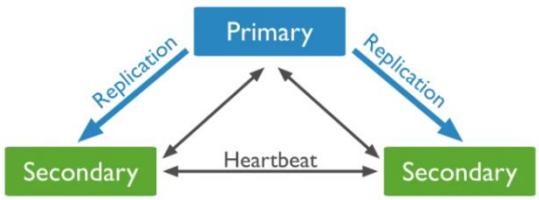
D. Konfigurasi Replikasi Database Mongodb

Berikut adalah rancangan konfigurasi replikasi database :



Gambar 4. Konfigurasi replikasi database

Gambar 4 merupakan rancangan konfigurasi replikasi database mongodb. Konfigurasi replikasi pada mongodb direkomendasikan memiliki 3 member/node, satu node akan menjadi primary dan lainnya akan menjadi secondary, ketika primary node *down*, salah satu node secondary akan otomatis menggantikannnya. Proses ini dinamakan election, proses yang menentukan node mana yang akan di jadikan primary. Rancangan arsitektur diatas merupakan representasi dari arsitektur mongodb cluster seperti berikut :



Gambar 5. Arsitektur mongodb Cluster

Gambar 5. Merupakan arsitektur yang di rekomendasikan oleh mongodb atau arsitektur simple mongodb cluster dimana ketiga node akan saling berkomunikasi melalui jaringan lokal, berikut konfigurasi masing-masing node :

Node1 primary
- Ip address : 192.168.43.141
- Port : 1111

Node2 secondary
- IP address : 192.168.43.109
- Port : 2222

Node 3 secondary
- Ip address : 192.168.43.116
- Port : 4444

Selanjutnya start mongodb server dengan masing masing konfigurasi diatas seperti pada gambar 6.

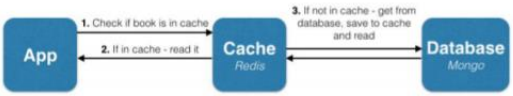
Gambar 6. Memulai server sesuai konfigurasi

Pada gambar 6 dijelaskan konfigurasi database dilakuakan dengan menjalankan masing masing database server sesuai dengan konfigurasi masing masing dengan sintaks mongod –bind_ip %ipadress% --port %port% --dbpath %path% --logpath %logpath%. –bind_ip untuk ip database server, --port untuk alokasi port database server, -dbpath adalah lokasi dari database, --logpath adalah untuk lokasi dari logfile database.

E. Konfigurasi Redis Cache

konfigurasi database server redis dilakukan ketika mengidentifikasikan koneksi yang dilakukan pada server side yaitu di dalam sintaks Node.js , denagn mendefinisikan fungsi modul redis serta membuat redisClient yang menggunakan ip server terkait serta portnya. Maka redisClient siap digunakan untuk melakukan request get data.

Redis bekerja pada memori RAM karena kinerja ram dalam pengolahan data sangat cepat dibanding hardisk, meskipun ukuran dari RAM tidak sebesar hardisk namun pada studi kasus mekanisme cache tidak perlu dibutuhkan resource data yang besar karena cache dari Redis bersifat sementara. Berikut permodelan mekanisme cache server :



Gambar 7. Mekanisme kerja redis cache

Gambar 7 merepresentasikan mekanisme cara kerja cache yang berjalan dalam aplikasi. Aplikasi akan melakukan pengecekan cache redis terlebih dahulu ketika hendak melakukan request data pada basis data, jika terdapat data yang sesuai dengan request pada basis data utama maka yang digunakan data yang ada pada redis cache. Namun ketika tidak menemukan data yang sesuai dengan request pada database , request akan dilakukan ke database utama kemudian hasil request tersebut akan di simpan sementara di dalam redis cache karena mengantisipasi akan adanya request terhadap data yang sama di masa mendatang.

F. Pengujian

Pengujian ini dilakukan untuk mengetahui apakah balancer melakukan tugasnya dalam pemagian traffic atau beban dengan *proper*/baik. Pengujian dilakukan dengan pengaksesan ke IP load balancer secara manual dengan menggunakan 2 browser yaitu chrome dan chrome dengan mode incognito secara bergantian untuk melihat alokasi traffic dari load balancer. Pengujian akan dianggap berhasil ketika IP load balancer di akses akan bergantian mengalokasikan beban ke dua server namun dengan endpoint yang mengembalikan data status server.

First Attempt

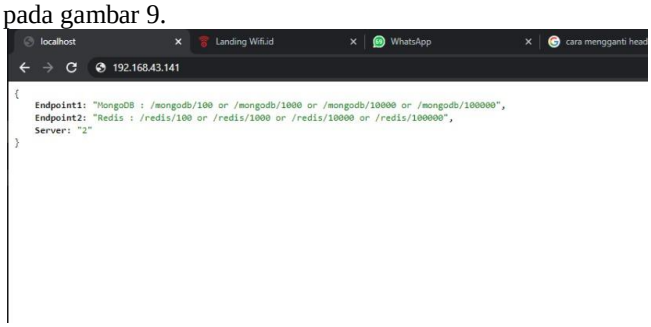
Hasil dari percobaan pertama adalah sebagaimana di tunjukan pada gambar 8.

Gambar 8. Hasil percobaan pertama

Gambar 8 menunjukan hasil dari percobaan pertama mengakses Ip load balancer dan hasilnya adalah endpoint dengan status yang menunjukan bahwa yang sedang di akses adalah server 1.

Second Attempt

Hasil dari percobaan Kedua adalah sebagaimana di tunjukan pada gambar 9.

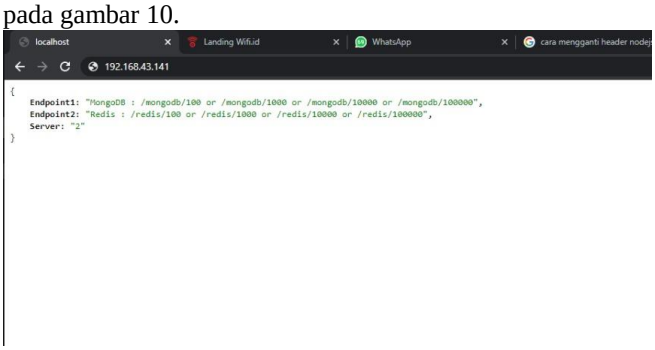


Gambar 9. Hasil percobaan Kedua

Gambar 9 menunjukan hasil dari percobaan kedua mengakses Ip load balancer dan hasilnya adalah endpoint dengan status yang menunjukan bahwa yang sedang di akses adalah server 2.

Third Attempt

Hasil dari percobaan ketiga adalah sebagaimana di tunjukan pada gambar 10.



Gambar 10. Hasil percobaan ketiga

Gambar 10 menunjukan hasil dari percobaan ketiga mengakses Ip load balancer dan hasilnya adalah endpoint dengan status yang menunjukan bahwa yang sedang di akses adalah server 2.

Fourth Attempt

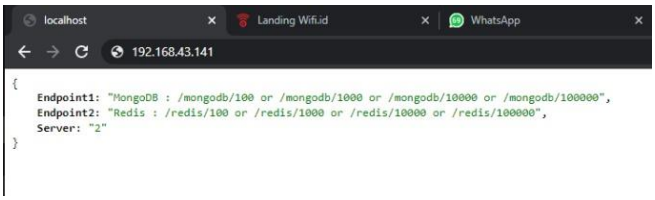
Hasil dari percobaan keempat adalah sebagaimana di tunjukan pada gambar 11.

Gambar 11. Hasil percobaan keempat

Gambar 11 menunjukan hasil dari percobaan keempat mengakses Ip load balancer dan hasilnya adalah endpoint dengan status yang menunjukan bahwa yang sedang di akses adalah server 1.

Pada hasil yang ditunjukan pada gambar 8, 9, 10, dan 11 bisa di tarik kesimpulan bahwa load balancing berjalan dengan semestinya yaitu dengan algoritma default atau RR(Round Robin) membagi beban secara merata dan bergantian tidak dengan memperhitungkan jumlah beban masing masing server. Itu terbukti dari keempat gambar tersebut bahwasanya pengalokasian beban saling bergantian dari server 1 kemudian berpindah ke server 2 selanjutnya kembali server 2 dan terakhir di alokasikan ke server 1.

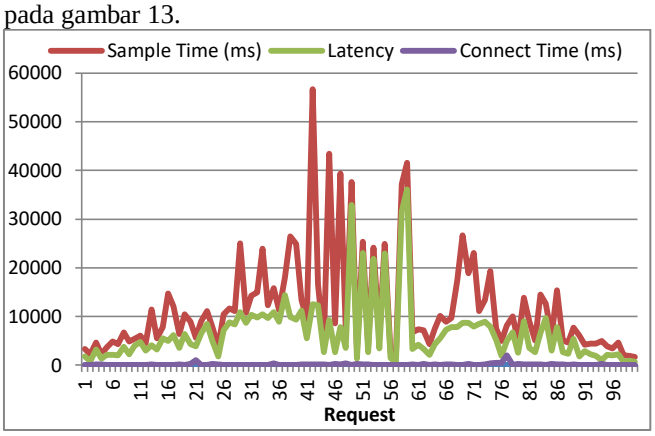
Pengujiain selanjutnya adalah pengujian failover load balancer yaitu pada kasus yang dibangun dalam percobaan kali ini , penulis akan mematikan salah satu server kemudian kembali mengakses ke loadbalancer apakah akan dialokasikan ke server yang tersedia. Pertama tama matikan salah satu server, disini yang di matikan adalah server 1 dengan perintah service nginx stop. Hasil dari pengujian fail over seperti pada gambar 12.



Gambar 12. Hasil Uji Failover

Gambar 12 menunjukan hasil dari pengujian failover bahwasanya ketika mengakses ip dari load balancing akan langsung di alokasikan ke server 2 dikarenakan server 1 sedang down atau dimatikan. Ini menunjukan bahwa loadbalancing bekerja dengan semestinya yaitu menjaga availability walaupun salah satu web server sedang down.

Kemudian akan dilakukan pengujian Load balancer dengan replikasi database tanpa redis cache. Pengujian ini dilakukan dengan bantuan tools Jmeter dengan ip tujuaun yaitu ip load balancer dan endpoint /mongodb/10000 (berisikan 10000 data), thread 10 user ,interval 50 detik dan iterasi sebanyak 10 kali. Hasil dari pengujian bisa dilihat pada gambar 13.



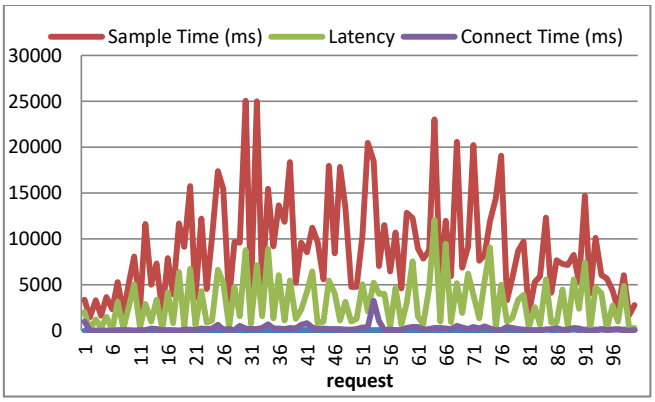
Gambar 13. Hasil pengujian tanpa redis cache

Gambar 13 menunjukan hasil dari 3 parameter yaitu sample time, latency, dan connect time pada request get sebanyak total 100 request dan dapat diambil informasi berupa rata rata nilai tiap parameter, nilai terendah dan tertinggi yang direpresentasikan dalam tabel 2.

Tabel 2. Hasil pengujian tanpa redis cache

No	Deskripsi	Hasil		
		Max	Min	Avg
1	Sample Time (ms)	56712	1617	11692.1
2	Latency	36132	589	6674.26
3	Connect Time(ms)	1992	0	137.78

Kemudian akan dilakukan pengujian untuk sistem yang menerapkan mekanisme redis cache untuk menarik kesimpulan dan untuk komparasi antara 2 hasil. Pengujian menggunakan tools yang sama dengan thread, iterasi, dan interval yang sama namun yang membedakan adalah endpoint tujuanya. Endpoint kali ini adalah /redis/10000 (menggunakan mekanisme redis cache dan request 10 ribu data). Hasil pengujian bisa dilihat pada gambar 14.



Gambar 14. Hasil pengujian penerapan redis cache

Gambar 14 menunjukan hasil uji 3 parameter yatu sample time dalam satuan milisecond(ms), latency, dan connectime dalam satuan milisecond(ms) pada request get sebanyak total 100 request dan dapat diambil informasi yang direpresentasikakan kedalam table sebagai berikut :

Tabel 3 hasil pengujian penerapan redis cache

No	Deskripsi	Hasil		
		Max	Min	Avg
1	Sample Time (ms)	25070	1449	8986.91
2	Latency	12118	104	3064.22
3	Connect Time(ms)	3263	0	234.27

Dilihat dari data yang dimuat pada tabel 2 dan 3 maka dapat disimpulkan sistem yang menerapkan mekanisme cache redis cenderung memiliki nilai nilai yang lebih rendah ketimbang sistem yang tidak menggunakan redis. Ini menunjukan bahwa penggunaan atau penerapan mekanisme redis cache sangat efektive untuk meningkatkan performa suatu sistem. Perbedaan nilai yang terdapat pada kedua sistem terpaut lumayan jauh hampir berselisih di 2 – 3 second untuk sample time, untuk latency penggunaan redis cache nyaris menurunkan nilai setengah dari nilai sistem yang tidak menerapkan mekanisme ini, kemudian pada connect time untuk sistem yang tidak menerapkan mekanisme redis cache lebih unggul. Namun dari performansi yang cenderung di pertimbangkan adalah reponse time dan dari data yang diperoleh sistem yang menerapkan mekanisme cache lebih unggul atau lebih cepat memberikan respon ketimbang sistem yang tidak menerapkan redis cache.

V. KESIMPULAN DAN SARAN

A. *Kesimpulan*

1) Penerapan load balancing terhadap sistem terdstribusi dapat bekerja dengan baik ketika request yang datang berhasil di distribusikan secara merata ke setiap web server yang tergabung dalam cluster secara merata sehingga server tidak mengalami overload karena saling membagi beban satu sama lain. Selain itu dengan penerapan load balancing sistem menjadi bersifat fault

tolerant atau saling membackup satu sama lain ketika salah satunya down.

2) Penerapan cluster mongodb database membuat sistem basis data menjadi high availability, dikarenakan mongodb cluster memiliki metode election dimana ketika primary database down maka secondary database akan menggantikan posisi primary database sehingga proses fungsional yang melibatkan komunikasi dengan database tidak akan terganggu.
3) Penerapan mekanisme redis cache bekerja dengan baik dimana dari hasil pengujian terjadi penurunan response time dari server yang berarti penerapan mekanisme ini sangat meningkatkan efektifitas dan performansi sistem terdistribusi.

VI. DAFTAR PUSTAKA

[1] R. Dani and F. Suryawan, "Perancangan dan Pengujian Load Balancing dan Failover Menggunakan NginX," *Khazanah Inform. J. Ilmu Komput. dan Inform.*, vol. 3, no. 1, p. 43, 2017.

[2] A. Rahmatulloh and F. MSN, "Implementasi Load Balancing Web Server menggunakan Haproxy dan Sinkronisasi File pada Sistem Informasi Akademik Universitas Siliwangi," *J. Nas. Teknol. dan Sist. Inf.*, vol. 3, no. 2, pp. 241–248, 2017.

[3] G. Singh and K. Kaur, "An Improved Weighted Least Connection Scheduling Algorithm for Load Balancing in Web Cluster Systems," pp. 1950–1955, 2018.

[4] E. Purwanto and S. Kom, "Perbandingan Strategi Replikasi Pada Sistem Basis Data Terdistribusi," *J. Inform.*, pp. 1–8, 2012.

[5] A. W. Services, "Database Caching Strategies Using Redis," no. May, 2017.

[6] R. J. Suhatril, "Sistem Terdistribusi," pp. 14–18, 2004.

[7] P. D. P. Silitonga, "Replikasi Basis Data Pada Sistem Pengolahan Data Akademik Univeristas Katolik Santo Thomas," *J. TIME*, vol. III, no. 1, pp. 32–36, 2014.

[8] W. Chen, A. Noertjahyana, and J. Andjarwirawan, "Analisis Perbandingan Kinerja Algoritma Load Balancer NGINX pada Studi Kasus PRS."

[9] R. Julianto, W. Yahya, and S. R. Akbar, "Implementasi Load Balancing Di Web Server Menggunakan Metode Berbasis Sumber Daya CPU Pada Software Defined Networking," *J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya*, vol. 1, no. 9, pp. 904–914, 2017.

[10] M. T. Nunit Prihatoni Siregar, Kemas Rahmat S. W., S.T., M.Eng , Alfian Akbar G., S.T., "Analisis dan Implementasi Basis Data Terdistribusi Horizontal pada MongoDB untuk ClikKB BKKBN Regional Jawa Barat Analysis," in *Cybrarians Journal*, 2015, vol. 2, no. 37, pp. 1–31.