Jurnal Indonesia : Manajemen Informatika dan Komunikasi
https://journal.stmiki.ac.id

E - ISSN: 2723-7079 | P - ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS

https://doi.org/10.35870/jimik.v5i3.999

Optimisasi *Web Service REST API* Menggunakan *Load Balancer* dan *Cache* dengan Algoritma *Round Robin* (Studi Kasus: Madani Infosphere)

Fadida Zanetti Junaedy [1], Untung Surapati [2*]

[1,2*] Program Studi Teknik Informatika, Sekolah Tinggi Ilmu Komputer Cipta Karya Informatika, Kota Jakarta Timur, Daerah Khusus Ibukota Jakarta, Indonesia.

Email: fadida@stikomcki.ac.id [1], kisuro2003@gmail.com [2*]

Histori Artikel:
Dikirim 25 Juli 2024; *Diterima dalam bentuk revisi* 10 Agustus 2024; *Diterima* 20 Agustus 2024; *Diterbitkan* 20 September 2024.

Abstrak

Di era digital saat ini, penggunaan Web Service REST API semakin meningkat, menjadi fondasi utama dalam pengembangan aplikasi web karena kesederhanaan dan skalabilitasnya. Namun, peningkatan permintaan dan kompleksitas data sering kali menyebabkan penurunan performa Web Service, terutama dalam hal waktu respons dan ketersediaan layanan. Penelitian ini berfokus pada optimasi kinerja Web Service Madani Infosphere, sebuah platform informasi yang dikembangkan oleh organisasi nirlaba MADANI Berkelanjutan. Platform ini sebelumnya menggunakan arsitektur server tunggal yang rentan terhadap masalah kinerja. Untuk mengatasi hal ini, penelitian ini menerapkan teknologi load balancer dengan algoritma round robin untuk mendistribusikan beban kerja secara merata ke beberapa server dan menggunakan cache untuk menyimpan data yang sering diakses. Hasil penelitian menunjukkan bahwa penggunaan load balancer dengan algoritma round robin efektif dalam mengatasi potensi kelebihan beban server, membuat kinerja Web Service lebih stabil dan responsif. Selain itu, implementasi cache berhasil mengurangi beban kerja server dan mempercepat waktu respons. Dengan demikian, optimasi ini dapat meningkatkan efisiensi dan ketersediaan layanan Web Service Madani Infosphere.

Kata Kunci: Web Service; REST API; Load Balancer; Cache; Round Robin.

Abstract

In today's digital era, the use of REST API Web Services is increasing, becoming a major foundation in web application development due to its simplicity and scalability. However, increased demand and data complexity often lead to a decrease in Web Service performance, especially in terms of response time and service availability. This research focuses on the performance optimisation of Madani Infosphere Web Service, an information platform developed by the non-profit organisation MADANI Berkelanjutan. This platform previously used a single server architecture that was prone to performance issues. To overcome this, this research applies load balancer technology with a round robin algorithm to distribute the workload evenly across multiple servers and uses a cache to store frequently accessed data. The results show that the use of load balancer with round robin algorithm is effective in overcoming potential server overload, making Web Service performance more stable and responsive. In addition, the cache implementation successfully reduces server workload and speeds up response time. Thus, this optimisation can improve the efficiency and availability of Madani Infosphere Web Services.

Keyword: Web Service; REST API; Load Balancer; Cache; Round Robin.

Jurnal Indonesia : Manajemen Informatika dan Komunikasi
https://journal.stmiki.ac.id
E-ISSN: 2723-7079 | P-ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

1. Pendahuluan

Manusia dan Alam untuk Indonesia Berkelanjutan (MADANI Berkelanjutan) adalah organisasi nirlaba yang berfokus pada penanganan krisis iklim melalui penelitian dan advokasi. MADANI Berkelanjutan memiliki sebuah *platform* bernama Madani Infosphere, sebuah sistem informasi terintegrasi yang menyediakan berbagai pengetahuan dan data terkait kerja-kerja organisasi. Madani Infosphere menggunakan REST API sebagai *Web Service* untuk menyediakan data dan layanan kepada pengguna. Di era digital saat ini, meningkatnya penggunaan *Web Service* menjadi tantangan tersendiri bagi pengelola platform digital. *Web Service* memungkinkan aplikasi pada platform yang berbeda untuk bertukar data bisnis (Monson-Haefel, 2024). Salah satu arsitektur *Web Service* yang populer adalah REST (*Representational State Transfer*), REST merupakan seperangkat aturan yang digunakan pada standar HTTP dan URI (*Uniform Resource Identifier*) (Bojinov, 2016). Pertumbuhan pengguna dan volume data yang semakin meningkat sering kali menyebabkan performa *Web Service* menurun, terutama dalam hal waktu respon dan ketersediaan layanan. Permasalahan utama dalam penelitian ini adalah bagaimana mengoptimalkan kinerja *Web Service* Madani Infosphere, sebuah platform yang menyediakan berbagai layanan informasi. Platform ini masih menggunakan arsitektur server tunggal, yang menyebabkan atau berpotensi menimbulkan masalah serius terkait kinerja *Web Service*. Solusi yang diharapkan dari penelitian ini adalah optimasi *Web Service* dengan penerapan teknologi *load balancer* dan *cache* dengan menggunakan algoritma *round robin*. Optimasi berarti menemukan solusi terbaik di antara banyak solusi layak yang tersedia bagi kita. Solusi yang layak adalah solusi yang memenuhi semua kendala dalam masalah optimasi. Solusi terbaik dapat berupa meminimalkan biaya suatu proses atau memaksimalkan efisiensi suatu sistem(Rajesh Kumar Arora, 2015).

Load balancer adalah perangkat yang mendistribusikan lalu lintas jaringan atau aplikasi saat melewati sekelompok server (Muhammad Arigoh Waluyo *et al.*, 2023). Dengan mendistribusikan permintaan yang masuk di antara beberapa server atau komponen, sebuah sistem dapat lebih efektif menangani beban berat dan mempertahankan ketersediaan yang tinggi (Kumar & Singh, 2024). Algoritma *round robin* digunakan sebagai salah satu metode *load balancer*. *Round Robin* mendistribusikan permintaan yang masuk atau lalu lintas jaringan secara merata di sekelompok server atau sumber daya *backend* (H. Ali, 2024). *Cache* adalah komponen atau mekanisme yang digunakan untuk menyimpan data atau instruksi yang sering diakses lebih dekat ke prosesor, mengurangi latensi dalam mengambil informasi dari penyimpanan yang lebih lambat atau sumber daya eksternal (Kumar & Singh, 2024). Dengan menyimpan informasi yang sering diakses ini dalam *cache*, akses berikutnya ke data yang sama dapat dilayani dengan lebih cepat, sehingga menghasilkan kinerja yang lebih baik (Kumar & Singh, 2024). Dalam beberapa tahun terakhir, berbagai penelitian telah menyoroti pentingnya penerapan metode load balancer dan cache untuk meningkatkan kinerja server dan sistem informasi. Penelitian menunjukkan bahwa load balancing secara signifikan dapat meningkatkan *throughput* dan skalabilitas sistem, dengan hasil uji coba yang menunjukkan adanya peningkatan *throughput* dari 7236,6 bps menjadi 11473,72 bps dan peningkatan dalam menangani peningkatan beban akibat pertumbuhan jumlah pengguna dan permintaan koneksi (Nuraini, 2022b, 2022a).

Penelitian lain menemukan bahwa *load balancer* pada aplikasi berhasil mengoptimalkan kinerja sistem. Aplikasi dapat berjalan dengan lancar dan memberikan hasil pemesanan yang cepat dengan menggunakan algoritma *round robin* (Sulaksono & Giovanni, 2020). Penelitian lebih lanjut menunjukkan bahwa penggunaan load balancer dengan algoritma round robin dapat berjalan dengan baik dengan memberikan bobot pada setiap server secara merata (Cerdas Kurniawan *et al.*, 2023; Sofyan *et al.*, 2022). Selain itu, penelitian menunjukkan bahwa teknik load balancing sangat efektif untuk kondisi beban kerja yang tinggi dan dapat membantu menjaga kestabilan sistem serta memberikan performa yang optimal (Wijaya *et al.*, 2023). Lebih lanjut, analisis dampak positif dari *load balancing* terhadap performa server, khususnya dalam *throughput*, menunjukkan hasil yang baik (Hadi Prayitno & Trianto, 2024). Demonstrasi bahwa kombinasi *load balancing*, *database clustering*, dan *caching* secara efektif meningkatkan performa *web server* dalam kondisi trafik dan volume data yang

Jurnal Indonesia : Manajemen Informatika dan Komunikasi
https://journal.stmiki.ac.id
E-ISSN: 2723-7079 | P-ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

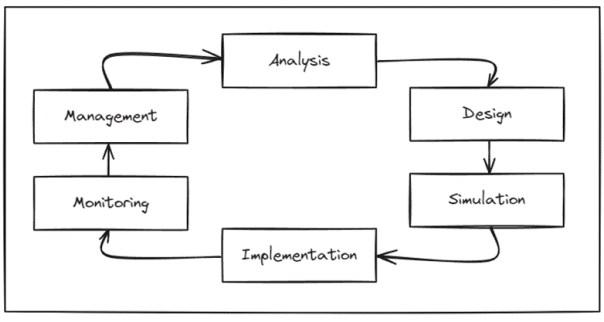
tinggi (Aemy & Rahmatulloh, 2024). Penelitian lain menemukan bahwa *load balancing* secara signifikan meningkatkan kinerja situs *web e-learning* dengan mengurangi waktu respons dan secara efisien menangani permintaan pengguna yang meningkat (Riskiono & Pasha, 2020). Penelitian lain mengonfirmasi bahwa metode penyeimbangan beban atau penerapan *load balancer* pada server web Nginx secara efektif mencegah kerusakan server dengan mendistribusikan beban kerja yang berat di beberapa server (Chyrvon *et al.*, 2023). Penelitian-penelitian tersebut menunjukkan bahwa teknologi load balancing dan caching memiliki potensi yang besar untuk mengatasi masalah performa pada *Web Service*. Namun, masih terdapat celah dalam literatur yang membahas integrasi kedua teknologi tersebut dengan menggunakan algoritma round robin dalam konteks peningkatan performa layanan web seperti Madani Infosphere. Penelitian ini berusaha untuk mengisi kesenjangan tersebut dengan mengintegrasikan kedua teknologi tersebut dan menguji keefektifan nya dalam konteks tertentu.

Tujuan dari penelitian ini adalah untuk mengoptimalkan performa *Web Service* Madani Infosphere dengan mengimplementasikan *load balancer* menggunakan algoritma round robin dan implementasi *cache*. Diharapkan dengan metode ini, response time dapat dikurangi dan throughput serta ketersediaan layanan dapat ditingkatkan, sehingga Madani Infosphere dapat memberikan layanan yang lebih responsif dan stabil kepada para penggunanya. Penelitian ini diharapkan dapat memberikan kontribusi yang signifikan dalam bidang optimasi *Web Service*, khususnya dalam konteks penggunaan kombinasi load balancer dan cache. Selain itu, hasil penelitian ini diharapkan dapat menjadi referensi bagi peneliti lain yang ingin mengimplementasikan solusi serupa untuk mengatasi permasalahan performa pada *Web Service*. Dengan demikian, penelitian ini tidak hanya memberikan solusi praktis bagi Madani Infosphere, tetapi juga menambah wawasan baru dalam ranah optimasi *Web Service*.

2. Metode Penelitian

2.1 Penerapan Metodologi

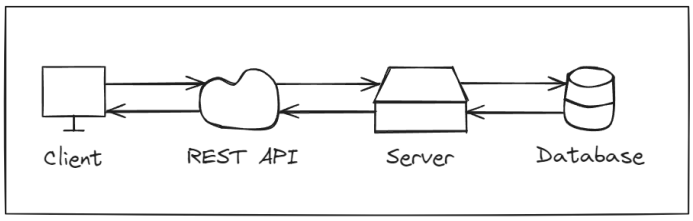
Penelitian ini menggunakan metode NDLC (*Network Development Life Cycle*). NDLC merupakan sebuah metode yang bergantung pada proses pembangunan sebelumnya. Dalam melakukan penerapan dari optimasi *Web Service* yaitu di mana sebuah pendekatan sistematis yang digunakan dalam pengembangan sistem atau jaringan komputer yang termasuk di dalam nya infrastruktur secara keseluruhan termasuk server dan aplikasi. Pendekatan ini membantu dalam perencanaan, perancangan, implementasi, dan pemeliharaan jaringan yang efektif dan efisien. Metode NDLC terdiri dari beberapa tahap. Berikut tahapan metodologi penelitian yang dapat dilihat pada gambar 1.



Gambar 1. *Network Development Life Cycle* (NDLC)

1) Analysis
Pada tahap ini, peneliti menganalisis permasalahan yang ada pada *Web Service* Madani Infosphere terkait dengan arsitektur dan performa. Berikut merupakan rancangan sistem *Web Service* Madani Infosphere yang sudah berjalan.

Jurnal Indonesia : Manajemen Informatika dan Komunikasi
https://journal.stmiki.ac.id
E-ISSN: 2723-7079 | P-ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

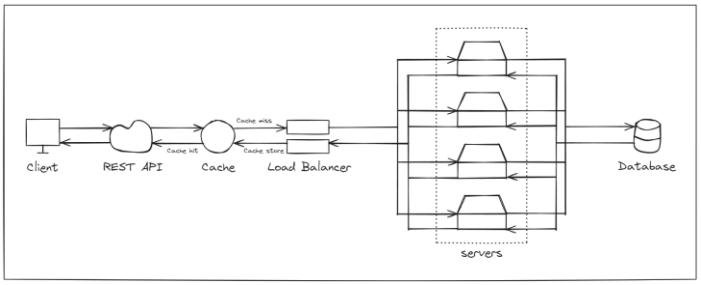


Gambar 2. Rancangan Sistem Madani Infosphere

Pada gambar 2, sistem masih menggunakan single server sehingga semua permintaan yang masuk langsung ditangani oleh satu server.

2) *Design*

Tahap ini peneliti akan membuat desain sistem dari informasi atau data yang telah didapat sebelumnya, peneliti merancang kebutuhan teknis yang terperinci seperti perangkat lunak, teknik, dan algoritma yang digunakan. Desain ini dapat dilihat pada gambar 3.



Gambar 3. Rancangan Optimasi Sistem Madani Infosphere

3) *Simulation*

Tahap ini melibatkan pengembangan sistem menggunakan *framework express*. Pemilihan *express* dipilih karena *REST API* yang sudah ada dibuat dengan *express*, dan peneliti akan mengembangkan dengan alat yang sama. Tahap ini bertujuan untuk menguji efektivitas dan efisiensi dari design sistem yang telah dirancang pada tahap sebelumnya.

4) *Implementation*

Tahap ini di mana design sistem yang telah disimulasikan akan diimplementasikan dalam lingkungan produksi. Proses ini melibatkan konfigurasi perangkat lunak yang dibutuhkan, serta pengaturan hosting tempat dimana service akan di deploy.

5) *Monitoring*

Pada Tahap ini sistem akan terus dimonitor secara berkala untuk memastikan bahwa semua komponen berjalan sesuai dengan yang diharapkan.

6) *Management*

Tahap ini melibatkan pengelolaan rutin terhadap sistem yang telah diimplementasikan. Ini mencakup pemeliharaan rutin, penanganan masalah yang mungkin muncul, serta penyesuaian dan peningkatan sesuai dengan kebutuhan yang berkembang. Manajemen juga mencakup pemantauan tren dan evaluasi performa sistem secara keseluruhan untuk memastikan bahwa tujuan optimasi terus tercapai.

2.2 Rancangan Pengujian

Metode pengujian yang akan dilakukan meliputi uji beban pada server menggunakan Apache HTTP server benchmarking tool. Pengujian ini dilakukan pada tahap simulation untuk mengetahui apakah rancangan sistem yang sudah didesain sebelumnya dapat berfungsi dengan baik sebelum di diimplementasikan. Peneliti menguji efektivitas dan efisiensi sistem *load balancer* dan *cache* yang telah dirancang dan disimulasikan. Selain itu, peneliti juga akan mengukur *throughput* dan *response time server*

Jurnal Indonesia : Manajemen Informatika dan Komunikasi
https://journal.stmiki.ac.id
E-ISSN: 2723-7079 | P-ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

sebelum dan setelah penerapan *load balancer* dan *cache*. Tujuannya adalah untuk memastikan bahwa sistem yang dirancang mampu menangani beban lalu lintas yang tinggi tanpa mengalami bottleneck. Adapun tahapan pengujian adalah sebagai berikut:

a) Persiapan
Mempersiapkan script Apache HTTP *server benchmarking tool* yang telah di konfigurasi dengan parameter yang sesuai, dan memastikan *load balancer* dan *cache* pada server backend telah di konfigurasi dengan benar.

b) Uji Beban
Menyusun skema pengujian beban pada server, selanjutnya menjalankan script ab untuk mensimulasikan beban *traffic* pada server. Selanjutnya peneliti mencatat dan memperhatikan nilai throughput dan response time yang dihasilkan oleh ab.

c) Analisis Hasil
Peneliti mencatat dan memperhatikan nilai throughput dan response time yang dihasilkan oleh ab. Membandingkan nilai *throughput* dan *response time* yang diperoleh sebelum dan setelah penerapan *load balancer* dan *cache*. Apabila ada potensi bottleneck, peneliti akan melakukan penyesuaian sistem yang diperlukan.

Pengujian akan dilakukan dengan memperhatikan beberapa dua variabel utama untuk mengevaluasi efektivitas dan efisiensi sistem yang diusulkan. Variabel yang akan diuji meliputi:

1) *Request per Second*
Metrik ini menunjukkan berapa banyak permintaan individu yang dapat ditangani server dalam satu detik. Metrik ini merupakan ukuran throughput. Angka yang lebih tinggi menandakan bahwa server mampu menangani lebih banyak permintaan per unit waktu, menunjukkan kinerja yang lebih baik di bawah beban.

$$Request\ per\ Second = \frac{Complete\ request}{Time\ taken\ for\ tests}$$

2) *Time per Request*
Metrik ini mengukur waktu rata-rata yang dibutuhkan untuk menyelesaikan satu permintaan. Metrik ini memberikan indikasi latensi.

$$Time\ per\ Request = \frac{Time\ taken\ for\ tests}{Complete\ requests}$$

Time per Request sejalan dengan *Response Time*, karena ini menggambarkan seberapa cepat server merespons permintaan.

3. Hasil dan Pembahasan

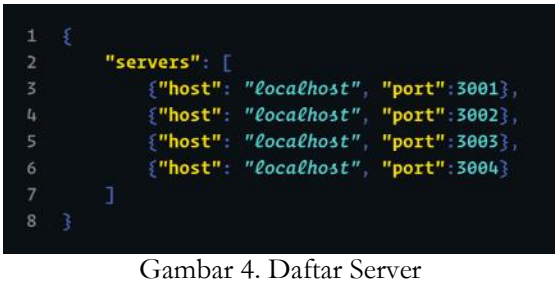
3.1 Hasil

Pada penelitian ini menggunakan metode pengembangan NDLC (*Network Development Life Cycle*). NDLC terdiri dari beberapa tahap yaitu *Analysis*, *Design*, *Simulation*, *Implementation*, *Monitoring*, *Management*. Setelah penerapan dengan metode pengembangan NDLC, dilakukan pengujian dengan tahap Persiapan, Uji Beban, dan Analisis Hasil.

Jurnal Indonesia : Manajemen Informatika dan Komunikasi
https://journal.stmiki.ac.id
E-ISSN: 2723-7079 | P-ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

3.1.1 *Network Development Life Cycle* (NDLC)

1) *Analysis*
Madani Infosphere saat ini masih menggunakan single-server untuk menangani semua permintaan dari pengguna. Kondisi ini dapat menyebabkan penurunan waktu respons ketika banyak pengguna melakukan permintaan secara bersamaan, dan juga berpotensi menyebabkan *overload* pada server.
2) *Design*
Peneliti merancang sistem optimasi untuk Madani Infosphere dengan menggunakan beberapa server (multi server) dan menerapkan load balancer dengan algoritma *round robin* untuk mendistribusikan beban secara merata. Selain itu, peneliti juga menerapkan *cache* untuk penyimpanan data sementara, sehingga ketersediaan data tetap terjaga.
3) *Simulation*
Pada tahap awal simulasi, peneliti membuat daftar server yang akan digunakan. Pada gambar 4, terlihat penggunaan 4 server dengan port yang berbeda.



Gambar 4. Daftar Server

Selanjutnya, server-server tersebut diaktifkan dengan cara melakukan looping, seperti yang terlihat pada gambar 5, sehingga semua server dapat digunakan.

Gambar 5. Multi Server

Dengan adanya beberapa server, klien hanya perlu mengakses satu port server saja. Selanjutnya, peneliti membuat fungsi *load balancer* untuk menangani semua permintaan yang masuk sebelum didistribusikan ke masing-masing server, seperti yang terlihat pada gambar 6.

Gambar 6. *Load Balancer Handler*

Jurnal Indonesia : Manajemen Informatika dan Komunikasi
https://journal.stmiki.ac.id
E-ISSN: 2723-7079 | P-ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

Pada fungsi *load balancer* di atas, peneliti menggunakan algoritma *round robin* untuk pendistribusian beban yang merata. Selanjutnya, pada gambar 7, peneliti membuat *middleware cache* untuk menangani permintaan yang masuk sebelum diproses oleh *load balancer*.

```
const NodeCache = require("node-cache")

const cache = new NodeCache()

const cacheMiddleware = async (req, res, next) => {
    try {
        if (req.method !== "GET") {
            console.error("Can't cache non-GET method!")
            return next()
        }

        const key = req.originalUrl
        const cachedResponse = cache.get(key)

        if (cachedResponse) {
            console.log(`Cache hit for ${key}`)
            return res.send(cachedResponse)
        } else {
            console.log(`Cache miss for ${key}`)
            res.originalSend = res.send.bind(res)
            res.send = (body) => {
                res.originalSend(body)
                cache.set(key, body, 300)
            }
        }

        next()
    } catch (error) {
        console.error("An error occurred in the cache middleware:", error)
        next(error)
    }
}

module.exports = cacheMiddleware
```

Gambar 7. Cache Middleware

Pada kode di atas, cache akan mengecek metode HTTP yang masuk. *Cache* hanya akan menerima metode HTTP GET untuk menyimpan data. Setelah itu, dicek apakah ada *cache* dengan kunci tertentu. Jika ada, data langsung dikembalikan ke klien (*cache hit*). Jika tidak, permintaan dilanjutkan ke load balancer (*cache miss*). Setelah semua komponen dibuat, peneliti membuat *server load balancer* untuk menampung semua komponen yang telah dibuat. Server load balancer dibuat menggunakan Express, seperti yang terlihat pada gambar 8.

```
const express = require("express")
const cacheMiddleware = require("./middlewares/cache-middleware.js")

const loadBalancer = express()
loadBalancer.use(cacheMiddleware)
loadBalancer.use((req, res) => loadBalancerHandler(req, res))
loadBalancer.listen(8080, err => {
    err ?
    console.log("Failed to listen to port 8080") :
    console.log("Load Balancer is listen to port 8080")
})
```

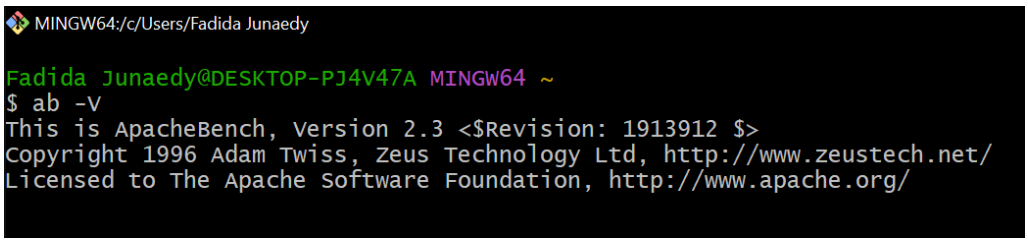
Gambar 8. Penerapan *Load Balancer*

4) *Implementation*
Pada tahap ini, sistem akan di *deploy* ke *hosting*. Sebelum proses *hosting* dilakukan, peneliti melakukan beberapa konfigurasi, seperti menyesuaikan host yang digunakan pada tahap simulasi (pengembangan), di mana localhost diubah menjadi domain api.madani-infosphere.id.
5) *Monitoring*
Pada tahap pengembangan, peneliti menambahkan fungsi *logging* untuk mencatat aktivitas pada server. *Logging* membantu memastikan bahwa setiap permintaan dapat berjalan dengan baik, yang mengindikasikan bahwa setiap komponen yang diterapkan berfungsi sebagaimana mestinya.
6) *Management*
Setelah menyelesaikan semua proses dan tahapan yang telah dilakukan, peneliti sekarang harus aktif memantau kinerja layanan web, memastikan semua tujuan tercapai, dan siap melakukan penyesuaian jika dibutuhkan di masa mendatang. Ini meliputi memonitor *traffic server*, performa *load balancer*, dan efektivitas *caching* untuk memastikan bahwa sistem tetap responsif dan efisien. Selain itu, peneliti perlu mempertimbangkan untuk melakukan skalabilitas jika terjadi peningkatan permintaan pengguna atau perubahan kebutuhan sistem.

Jurnal Indonesia : Manajemen Informatika dan Komunikasi
https://journal.stmiki.ac.id
E-ISSN: 2723-7079 | P-ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

3.1.2 Pengujian

1) Persiapan



Gambar 9. Apache Benchmark Version

Pada pengujian ini, peneliti menggunakan Apache Benchmark versi 2.3. Apache Benchmark (ab) adalah alat uji beban HTTP yang sangat sederhana namun kuat, yang dirancang untuk memberikan gambaran tentang kinerja server web.

2) Uji Beban

Uji beban dilakukan menggunakan Apache Benchmark untuk mengukur *Request Per Time* dan *Time Per Request server* sebelum dan sesudah optimasi sistem dengan *load balancer* dan *cache*. Pengujian dilakukan dengan mensimulasikan 1000 hingga 10000 permintaan ke server dengan *concurrency* 100 hingga 1000, yang mengacu pada jumlah permintaan yang dikirimkan secara bersamaan oleh klien. *Concurrency* ini mengukur kemampuan server untuk menangani banyak permintaan secara simultan, sehingga memberikan gambaran yang lebih akurat tentang kinerja server dalam kondisi beban tinggi. Untuk detail lebih lanjut, dapat dilihat pada tabel 1.

Tabel 1. Skema Pengujian

Pengujian ke-	Concurrency	Total Request
1	100	1000
2	200	2000
3	300	3000
4	400	4000
5	500	5000
6	600	6000
7	700	7000
8	800	8000
9	900	9000
10	1000	10000

3) Analisis Hasil

Berdasarkan skema pengujian yang telah dilakukan, berikut adalah hasil pengujian sistem sebelum dan sesudah penerapan load balancer dan cache.

Tabel 2. Hasil rata-rata pengujian sebelum optimasi

Pengujian ke-	*Concurrency*	Total *Request*	*Time taken for test* (s)	*Request per Second*	*Time per Request* (ms)
1	100	1000	13.18	75.90	1317.49
2	200	2000	25.37	78.84	2536.78
3	300	3000	39.26	76.41	3926.39
4	400	4000	54.44	73.50	5442.23

Jurnal Indonesia : Manajemen Informatika dan Komunikasi
https://journal.stmiki.ac.id
E-ISSN: 2723-7079 | P-ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

5	500	5000	66.91	74.73	6690.58
6	600	6000	91.87	65.31	9187.41
7	700	7000	105.99	66.05	10598.46
8	800	8000	111.02	72.06	11101.92
9	900	9000	130.86	68.78	13085.66
10	1000	10000	143.60	69.64	14359.66
		Rata-rata	78.25	72.12	7824.66

Tabel 3. Hasil rata-rata pengujian setelah optimasi

Pengujian ke-	*Concurrency*	Total *Request*	*Time taken for test* (s)	*Request per Second*	*Time per Request* (ms)
1	100	1000	2.83	354.04	282.45
2	200	2000	4.85	412.33	485.05
3	300	3000	6.71	447.23	670.79
4	400	4000	9.89	404.63	988.55
5	500	5000	15.45	323.72	1544.55
6	600	6000	15.47	387.95	1546.60
7	700	7000	17.39	402.63	1738.56
8	800	8000	19.91	401.74	1991.35
9	900	9000	23.45	383.80	2344.98
10	1000	10000	23.74	421.17	2374.33
		Rata-rata	13.97	393.92	1396.72

Hasil Setelah implementasi load balancer dan cache, terjadi peningkatan kinerja sistem secara signifikan. Berdasarkan tabel hasil pengujian pada Tabel 3.2 dan Tabel 3.3, terlihat bahwa rata-rata permintaan yang ditangani per detik (*Request per Second*) meningkat drastis dari rata-rata 72.12 menjadi 393.92. Ini menunjukkan bahwa sistem mampu menangani lebih banyak permintaan dalam waktu yang sama setelah optimasi dilakukan. Selain itu, data menunjukkan penurunan signifikan pada waktu yang dibutuhkan untuk menangani setiap permintaan (*Time per Request*). Rata-rata waktu yang dibutuhkan turun dari 7824.66ms menjadi 1396.72ms, mengindikasikan bahwa sistem menjadi lebih efisien dan responsif terhadap permintaan klien setelah penerapan cache.

3.2 Pembahasan

Penerapan *load balancer* dengan algoritma *round robin* dan penggunaan *cache* pada *web service* Madani Infosphere terbukti meningkatkan kinerja platform secara signifikan. Penggunaan algoritma *round robin* untuk *load balancing* telah terbukti efektif dalam berbagai penelitian sebelumnya. Menurut Cerdas Kurniawan *et al.* (2023), algoritma *round robin* mampu mendistribusikan permintaan secara merata di antara server, sehingga mengurangi risiko kelebihan beban pada satu server tertentu dan meningkatkan stabilitas sistem. Hal ini sejalan dengan hasil penelitian ini, di mana peningkatan rata-rata permintaan yang dapat ditangani oleh server per detik meningkat dari 72,12 menjadi 393,92 setelah penerapan optimasi ini. Selain itu, penggunaan *cache* sebagai bagian dari optimasi kinerja *web service* juga memberikan dampak positif. Aemy dan Rahmatulloh (2024) menunjukkan bahwa implementasi *cache* dalam sistem terdistribusi, seperti penggunaan *Redis cache*, dapat mengurangi beban pada server dengan menyimpan data yang sering diakses. Pada penelitian ini, penerapan *cache* mampu menurunkan waktu rata-rata yang dibutuhkan untuk menyelesaikan setiap permintaan dari 7824,66 ms menjadi 1396,72 ms. Penurunan ini mengindikasikan peningkatan efisiensi dalam penanganan permintaan klien, membuat sistem lebih responsif dan stabil.

Jurnal Indonesia : Manajemen Informatika dan Komunikasi
https://journal.stmiki.ac.id
E-ISSN: 2723-7079 | P-ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

Penelitian ini juga mengkonfirmasi temuan Muhammad Arigoh Waluyo *et al.* (2023) yang menggarisbawahi efektivitas algoritma *round robin* dalam *load balancing* untuk server *web* dengan meningkatkan kemampuan sistem dalam menangani beban permintaan yang tinggi. Hal ini relevan dengan hasil penelitian di mana algoritma *round robin* memastikan distribusi beban kerja yang merata, menjaga ketersediaan layanan terutama ketika volume permintaan meningkat secara signifikan. Algoritma ini efektif karena bekerja dengan prinsip mendistribusikan setiap permintaan secara bergilir, yang sederhana namun cukup untuk kasus-kasus dengan spesifikasi server yang serupa. Namun, penting untuk mempertimbangkan bahwa meskipun algoritma *round robin* efektif dalam mendistribusikan beban, algoritma ini tidak mempertimbangkan kapasitas masing-masing server. Seperti yang diungkapkan oleh Riskiono dan Pasha (2020), algoritma ini mungkin tidak memberikan distribusi optimal ketika server memiliki kapasitas berbeda-beda. Dalam situasi ini, penggunaan algoritma yang lebih adaptif seperti *weighted round robin* atau *least connection* mungkin lebih sesuai. Algoritma-algoritma ini dapat memperhitungkan kapasitas server dan jumlah koneksi aktif, yang menghasilkan distribusi beban yang lebih optimal dan efisien. Penggunaan *cache* dalam penelitian ini juga memerlukan evaluasi lebih lanjut terkait strategi yang diterapkan. Menurut Chyrvon *et al.* (2023), strategi *cache* yang baik harus mempertimbangkan faktor-faktor seperti frekuensi akses data dan kebutuhan konsistensi data. Dalam skenario di mana data mengalami perubahan yang dinamis atau memerlukan pembaruan berkala, strategi *cache* yang lebih canggih seperti invalidasi otomatis atau pembaruan periodik mungkin diperlukan untuk memastikan data yang disajikan tetap relevan dan akurat bagi pengguna.

Secara keseluruhan, penelitian ini memperkuat temuan-temuan sebelumnya tentang pentingnya *load balancing* dan *caching* dalam meningkatkan kinerja *web service*. Sebagai contoh, Hadi Prayitno dan Trianto (2024) menunjukkan bahwa *load balancing* dapat meningkatkan throughput dan stabilitas sistem secara signifikan, terutama dalam konteks implementasi di lingkungan dengan beban tinggi. Demikian pula, penelitian oleh Wijaya *et al.* (2023) mengonfirmasi bahwa teknik *load balancing* sangat efektif untuk kondisi beban kerja yang tinggi dan dapat membantu menjaga kestabilan sistem. Keberhasilan optimasi ini menggarisbawahi bahwa teknologi *load balancer* dan *cache* merupakan solusi yang efektif untuk meningkatkan performa *web service*, terutama di tengah peningkatan jumlah pengguna dan permintaan. Namun, untuk mencapai efisiensi yang lebih tinggi, diperlukan evaluasi dan penyesuaian berkelanjutan terhadap konfigurasi sistem dan algoritma yang digunakan, seperti yang diusulkan oleh Kumar dan Singh (2024) dalam konteks desain sistem di lingkungan *cloud*.

4. Kesimpulan

Berdasarkan penelitian yang telah dilakukan mengenai optimasi kinerja *Web Service* REST API menggunakan *load balancer* dan *cache* dengan algoritma round robin pada platform Madani Infosphere, dapat disimpulkan beberapa poin penting sebagai berikut: Berdasarkan penelitian yang telah dilakukan mengenai optimasi kinerja *Web Service* REST API menggunakan *load balancer* dan *cache* dengan algoritma *round robin* pada platform Madani Infosphere, dapat disimpulkan beberapa poin penting sebagai berikut:

1) Penggunaan *load balancer* dengan algoritma round robin berhasil mendistribusikan beban kerja secara merata ke beberapa server. Hal ini efektif dalam mengatasi potensi kelebihan beban server (*overload*), sehingga kinerja *Web Service* Madani Infosphere menjadi lebih stabil dan responsif.
2) Implementasi *cache* pada *Web Service* Madani Infosphere mampu mengurangi beban kerja server dengan menyimpan data yang sering diakses. Dengan demikian, server tidak perlu memproses permintaan yang sama berulang kali, sehingga waktu response dapat dipersingkat dan efisiensi kinerja sistem meningkat.

E-ISSN: 2723-7079 | P-ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

5. Ucapan Terima Kasih

Peneliti ingin menyampaikan rasa terima kasih yang tulus kepada semua pihak yang telah memberikan dukungan dan bantuan selama proses penyusunan jurnal ini. Penghargaan yang mendalam diberikan kepada dosen pembimbing atas bimbingan dan saran yang berharga, rekan-rekan sejawat atas dukungan moral dan intelektual, serta keluarga yang selalu memberikan doa dan semangat yang tiada henti. Peneliti juga ingin mengucapkan terima kasih kepada pihak Madani karena telah memberikan kesempatan untuk melakukan penelitian ini. Semua kontribusi ini sangat berarti dalam penyelesaian jurnal ini, dan peneliti berharap hasil karya ini dapat memberikan manfaat bagi perkembangan ilmu pengetahuan dan masyarakat luas.

6. Daftar Pustaka

Aemy, N., & Rahmatulloh, A. Implementasi HALB dan Klaster MongoDB dengan Penyimpanan Cache Redis dalam Sistem Terdistribusi. *JUSTIN (Jurnal Sistem dan Teknologi Informasi)*, *12*(2), 265-270. DOI: http://dx.doi.org/10.26418/justin.v12i2.73413.

Arora, R. K. (2015). *Optimization: algorithms and applications*. CRC press.

Bojinov, V. (2016). *RESTful Web API Design with Node. js*. Packt Publishing Ltd.

Chyrvon, A., Lisovskyi, K., & Kyryndas, N. (2023). the Main Methods of Load Balancing on the Nginx Web Server. *Collection of scientific papers «ΛΌГОΣ»*, (May 26, 2023; Boston, USA), 146-151. DOI: https://doi.org/10.36074/logos-26.05.2023.04.

Nuraini, R. (2022). Implementasi Metode Load balancing Sebagai Upaya Meningkatkan Kinerja Server. *Journal of Information System Research (JOSH)*, *3*(4), 507-514. DOI: https://doi.org/10.47065/josh.v3i4.1792.

Riskiono, S. D., & Pasha, D. (2020). Analisis Metode Load Balancing Dalam Meningkatkan Kinerja Website E-Learning. *Jurnal TeknoInfo*, *14*(1), 22-26. DOI: https://doi.org/10.33365/jti.v14i1.466.

Setiawan, Y., & Farosh, G. M. (2023). Analisis Load Balancing Round Robin dan Fault Detection pada Software Defined Network Berbasis P4. *Indonesian Journal of Computer Science*, *12*(2).

Singh, I. (2002). *Designing enterprise applications with the J2EE platform*. Addison-Wesley Professional.

Sofyan, A. R., & Kusuma, S. D. Y. (2022). Implementasi load balancing web server menggunakan Haproxy pada virtual server Direktorat SMK Kemendikbudristek. *Jurnal Pendidikan Tambusai*, *6*(2), 9669-9682. DOI: https://doi.org/10.31004/jptam.v6i2.3954.

Sulaksono, D. H., & Giovanni, A. R. (2020). IMPLEMENTASI LOAD BALANCING MENGGUNAKAN ANTRIAN ROUND ROBIN DENGAN STUDI KASUS E-SHOP. *Jurnal Riset Inovasi Bidang Informatika Dan Pendidikan Informatika (KERNEL)*, *1*(2).

Waluyo, M. A., Antony, F., & Setiawan, C. (2023). Implementasi Load Balancing Web Server dengan Haproxy Menggunakan Algoritma Round Robin. *Journal of Intelligent Networks and IoT Global*, *1*(1), 46-52. DOI: https://doi.org/10.36982/jinig.v1i1.3074.

E - ISSN: 2723-7079 | P - ISSN: 2776-8074
Vol. 5 No. 3 (2024) | September
OPEN ACCESS
https://doi.org/10.35870/jimik.v5i3.999

Wartono, W., Rudiansyah, R., Prayitno, M. H., & Trianto, J. (2024). ANALISIS PERFORMA LOAD BALANCING TERHDAP THROUGHPUT PADA KLUSTER SERVER UNTUK MENDUKUNG SMART CITY. *Jurnal Teknoinfo*, *18*(1), 173-181. DOI: https://doi.org/10.33365/jti.v18i1.3442.

Wijaya, H., Abdurrohim, I., Tugiyono, J., & Rumandan, R. J. (2023). Implementasi Metode Load Balancing Untuk Optimalisasi Performa Server Pada Jaringan Internet. *Journal of Information System Research (JOSH)*, *5*(1), 252-260. DOI: https://doi.org/10.47065/josh.v5i1.4386.