PROSISI: Prosiding Seminar Nasional Saintek
Vol.1 No.1, 2025, Hal. 470-479
ISSN XXXX-XXXX

Implementasi *Autoscaling* Reaktif Berbasis Fuzzy *Logic* Mamdani pada *Platform* Docker Swarm

Henni Endah Wahanani[1*], Bayu Setiawan[1]

[1] Universitas Pembangunan Nasional "VETERAN" Jawa Timur
[*]E-mail: henniendah@upnjatim.ac.id, khususpby@gmail.com

Abstrak

Mekanisme autoscaling pada Docker Swarm umumnya bergantung pada pendekatan reaktif yang menggunakan formula berbasis target utilitas. Pendekatan ini memiliki keterbatasan karena keputusan scaling hanya bergantung pada nilai utilitas saat ini dan belum mempertimbangkan arah perubahan beban kerja. Penelitian ini mengimplementasikan autoscaling reaktif berbasis Fuzzy Logic Mamdani pada Docker Swarm dengan dua variabel input, yaitu CPU usage dan CPU trend, serta aturan inferensi linguistik untuk menghasilkan keputusan scaling yang lebih gradual dan kontekstual. Pengujian dilakukan pada klaster Docker Swarm dengan 1 manager dan 2 worker menggunakan profil beban kerja dari dataset BitBrains GWA-T-12 selama 24 jam. Hasil pengujian menunjukkan bahwa pendekatan fuzzy menurunkan rata-rata response time dari 108,40 ms menjadi 87,04 ms (19,7%), meningkatkan rata-rata RPS dari 24,75 req/s menjadi 26,86 req/s (8,5%), serta mereduksi scaling event dari 237 menjadi 113 kali (52,3%), dengan konsekuensi rata-rata replika yang sedikit lebih tinggi.

Kata kunci : *autoscaling*, Docker Swarm, *fuzzy logic*, kontainer, Mamdani.

Abstract

Autoscaling mechanisms on Docker Swarm generally rely on reactive approaches based on utilization targets. This approach has limitations because scaling decisions depend only on current utilization and do not consider the direction of workload changes. This study implements Mamdani Fuzzy Logic-based reactive autoscaling on Docker Swarm using two input variables, CPU usage and CPU trend, along with linguistic inference rules to produce more gradual and contextual scaling decisions. Testing was conducted on a Docker Swarm cluster with 1 manager and 2 workers using workload profiles from the BitBrains GWA-T-12 dataset over 24 hours. The results show that the fuzzy approach reduces average response time from 108.40 ms to 87.04 ms (19.7%), increases average RPS from 24.75 req/s to 26.86 req/s (8.5%), and decreases scaling events from 237 to 113 times (52.3%), with a slightly higher average replica count.

Keywords: *autoscaling, container, Docker Swarm, fuzzy logic, Mamdani.*

PENDAHULUAN

Seiring dengan meningkatnya penggunaan layanan digital, kebutuhan terhadap mekanisme pengelolaan sumber daya server menjadi semakin penting. Intensitas akses yang tinggi menyebabkan pola permintaan menjadi dinamis dan sulit diprediksi. Tanpa pengelolaan sumber daya yang memadai, lonjakan beban dapat meningkatkan waktu respons hingga menyebabkan kegagalan layanan. Sebaliknya, penyediaan sumber daya secara berlebihan dapat menimbulkan pemborosan biaya operasional. Oleh karena itu, autoscaling menjadi mekanisme penting untuk menjaga elastisitas layanan ketika beban kerja berubah [1][2].

Kontainerisasi menjadi salah satu pendekatan utama dalam pengelolaan aplikasi modern karena mampu menjalankan aplikasi dalam lingkungan terisolasi dengan kebutuhan sumber daya yang lebih ringan dibandingkan virtualisasi tradisional [3]. Docker Swarm merupakan platform orkestrasi kontainer yang sederhana dan sesuai untuk klaster berskala kecil hingga menengah. Namun, berbeda dengan Kubernetes yang menyediakan Horizontal Pod Autoscaler (HPA), Docker Swarm tidak

menyediakan fitur autoscaling bawaan sehingga proses penyesuaian kapasitas layanan umumnya dilakukan secara manual atau melalui mekanisme tambahan [3][4].

Pendekatan autoscaling reaktif menghitung jumlah replika berdasarkan rasio penggunaan CPU aktual terhadap target utilitas pada saat pengukuran [5][6]. Meskipun formula tersebut mempertimbangkan besar deviasi dari target, keputusan scaling tetap bergantung pada nilai utilitas saat ini dan belum mempertimbangkan arah perubahan beban. Akibatnya, sistem sulit membedakan apakah penggunaan CPU tinggi sedang meningkat, stabil, atau menurun. Dalam kondisi beban yang fluktuatif, kekurangan konteks temporal tersebut dapat memicu perubahan replika berulang dan mengurangi stabilitas layanan [7][8].

Fuzzy logic memungkinkan keputusan autoscaling dibuat secara gradual melalui kategori linguistik dan derajat keanggotaan, bukan keputusan biner [9]. Silva et al. [2] menunjukkan bahwa Fuzzy Logic Mamdani mampu menghasilkan latensi lebih rendah dibandingkan HPA berbasis *threshold* pada *Platform* Kubernetes. Penelitian autoscaling kontainer juga telah dikaji melalui kontrol *feedback* [10], metrik autoscaling [7][8], *elastic deployment* [11], dan metode prediktif [12]. Namun, penerapan inferensi fuzzy Mamdani pada Docker Swarm masih belum banyak dibahas, terutama untuk autoscaling reaktif berbasis CPU usage dan CPU trend.

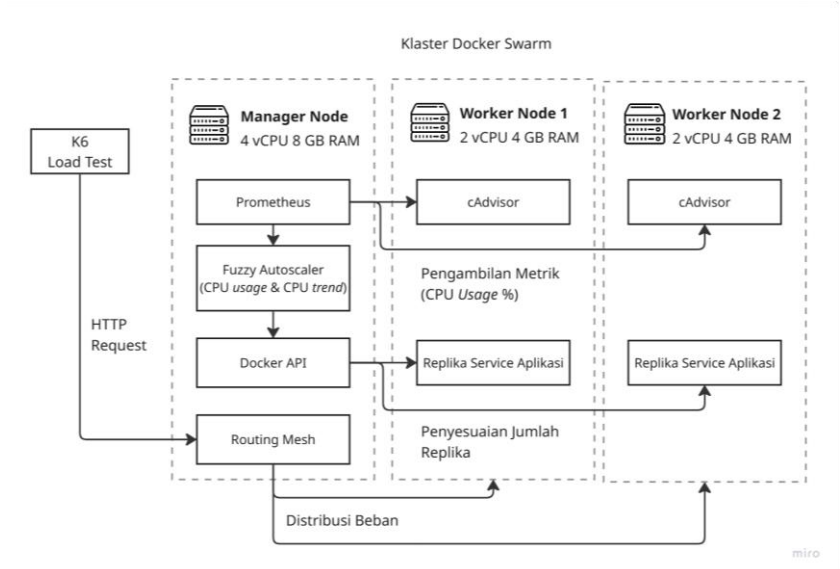
Berdasarkan permasalahan tersebut, penelitian ini mengimplementasikan autoscaling reaktif berbasis Fuzzy Logic Mamdani pada Docker Swarm. Sistem dirancang menggunakan dua variabel input, yaitu CPU usage dan CPU trend, serta sembilan aturan linguistik untuk menghasilkan keputusan scaling yang gradual, kontekstual, dan mudah dijelaskan. Metode Mamdani dipilih karena menghasilkan output linguistik yang intuitif dan dapat ditelusuri melalui aturan inferensi [13]. Kinerja sistem fuzzy dievaluasi terhadap autoscaling reaktif berbasis rasio HPA menggunakan metrik response time, Requests Per Second (RPS), rata-rata replika, dan jumlah scaling event.

METODE PENELITIAN

Penelitian ini merancang dan mengevaluasi autoscaling reaktif berbasis Fuzzy Logic Mamdani pada klaster Docker Swarm. Tahapan penelitian meliputi perancangan arsitektur autoscaler, perancangan sistem inferensi fuzzy, *preprocessing* profil beban kerja, kalibrasi beban k6, dan pengujian komparatif terhadap mekanisme reaktif berbasis rasio HPA.

Arsitektur dan Lingkungan Pengujian

Arsitektur sistem mengintegrasikan Docker Swarm, cAdvisor, Prometheus, modul autoscaler, dan aplikasi target. cAdvisor mengumpulkan metrik CPU kontainer, Prometheus menarik metrik setiap 5 detik, sedangkan modul autoscaler membaca metrik melalui PromQL. Nilai CPU usage dan CPU trend diproses oleh sistem inferensi fuzzy untuk menentukan jumlah replika, lalu keputusan scaling dikirimkan ke Docker Swarm melalui Docker API. Arsitektur sistem ditunjukkan pada Gambar 1.



Gambar 1. Arsitektur sistem

Lingkungan pengujian menggunakan tiga node VPS DigitalOcean berbasis Ubuntu 22.04 LTS, terdiri atas satu manager dan dua worker. Aplikasi target berupa layanan web Flask yang menjalankan hashing SHA-256 sebanyak 50.000 iterasi untuk setiap request sehingga menghasilkan beban CPU-bound. Konfigurasi utama eksperimen dijelaskan pada Tabel 1.

Tabel 1. Konfigurasi lingkungan pengujian

Parameter	Nilai
Topologi	1 Manager (4 vCPU, 8 GB RAM) 2 Worker (2 vCPU, 4 GB RAM)
Sistem Operasi	Ubuntu 22.04 LTS
Monitoring	Prometheus v3.2.1, cAdvisor v0.51.0, Grafana v11.5.2
Autoscaler	Python 3.11 dan Docker API
Load Generator	Grafana k6 v0.49.0
Aplikasi Target	SHA 256 50.000 iterasi/*requests*
Replika Kontainer	Minimal 1, Maksimal 10
Konfigurasi Autoscaling	Evaluasi 15 detik, Interval CPU 5 menit, *scale down* 300 detik
Batas CPU	0.5 core Per Replika

CPU *usage* yang digunakan sebagai input autoscaler diperoleh dari metrik Prometheus dan cAdvisor, kemudian dinyatakan sebagai rata-rata utilisasi CPU aktual layanan terhadap batas CPU per replika. Nilai ini digunakan sebagai metrik CPU aktual pada autoscaler fuzzy dan baseline reaktif sehingga kedua skenario menggunakan sumber metrik yang sama.

Sistem Inferensi Fuzzy Mamdani

Autoscaler fuzzy menggunakan dua *input*, yaitu CPU *usage* dan CPU *trend*, serta satu *output* berupa tindakan scaling. Proses inferensi mengikuti tahapan Mamdani yang meliputi fuzzifikasi, evaluasi aturan, agregasi, dan defuzzifikasi centroid [13]. CPU *usage* merepresentasikan rata-rata penggunaan CPU terhadap batas CPU per replika, sedangkan CPU *trend* merepresentasikan arah perubahan CPU *usage* antar siklus evaluasi autoscaler. CPU trend dihitung menggunakan pendekatan *first difference* antar-observasi berurutan sebagaimana ditunjukkan pada Persamaan (1)[5].

$$CPU\ trend = CPU_t - CPU_{t-1} \quad (1)$$

dengan CPU_t adalah rata-rata CPU *usage* pada siklus saat ini CPU_{t-1} adalah rata-rata CPU usage pada siklus sebelumnya. Autoscaler berjalan setiap 15 detik, sedangkan nilai CPU usage diperoleh dari *query* Prometheus dengan window 5 menit. Dengan demikian, CPU *trend* merepresentasikan perubahan antar-siklus dari rata-rata CPU jangka pendek.

Fungsi keanggotaan CPU usage mempertimbangkan target CPU 70% pada baseline HPA dan temuan Peng et al. [14] bahwa VM *non-intensive* umumnya memiliki rata-rata penggunaan CPU di bawah 30%. CPU usage dibagi menjadi *Low, Medium, dan High.* CPU *trend* dibagi menjadi *Falling, Stable, dan Rising,* dengan rentang *stable* -10 hingga +10 poin persentase sebagai *deadband*. Variabel *output* berupa tindakan scaling pada rentang -3 hingga +5 replika. Konfigurasi variabel fuzzy disajikan pada Tabel 2.

Tabel 2. Konfigurasi variabel dan kategori fuzzy

Variabel	Kategori	Rentang
CPU *usage*	Low	0-50%
CPU *usage*	Medium	30-80%
CPU *usage*	High	70-100%
CPU *trend*	Falling	-100 hingga 0
CPU *trend*	Stable	-10 hingga +10
CPU *trend*	Rising	0 hingga +100
Scaling Action	Scale Down	-3 hingga 0
Scaling Action	Maintain	-1 hingga +1
Scaling Action	Scale Up Small	0 hingga +3
Scaling Action	Scale Up Large	+2 hingga +5

Mengacu pada karakteristik Fuzzy Logic Mamdani yang menggunakan aturan linguistik [4], rule base disusun berdasarkan hubungan antara tingkat CPU usage, arah CPU trend, dan tindakan scaling. CPU usage menunjukkan kebutuhan kapasitas, sedangkan CPU trend menunjukkan arah perubahan beban. Secara umum, CPU rendah diarahkan ke scale down atau maintain, CPU menengah dengan trend meningkat diarahkan ke scale up kecil, dan CPU tinggi dengan trend meningkat diarahkan ke scale up besar untuk mengurangi risiko overload. Berdasarkan prinsip tersebut, kombinasi tiga kategori CPU usage dan tiga kategori CPU trend menghasilkan sembilan aturan eksplisit sebagaimana ditunjukkan pada Tabel 3.

Tabel 3. Basis Aturan Fuzzy Logic Mamdani

Aturan	CPU *usage*	CPU *trend*	Tindakan Scaling
R1	Low	Falling	Scale Down
R2	Low	Stable	Scale Down
R3	Low	Rising	Maintain
R4	Medium	Falling	Scale Down
R5	Medium	Stable	Maintain
R6	Medium	Rising	Scale Up Small
R7	High	Falling	Maintain
R8	High	Stable	Scale Up Small
R9	High	Rising	Scale Up Large

Evaluasi aturan menggunakan operator AND minimum, kemudian seluruh output aturan diagregasikan. Defuzzifikasi dilakukan dengan metode centroid sebagaimana ditunjukkan pada Persamaan (2).

$$output = \frac{\int x \cdot \mu(x)\, dx}{\int \mu(x)\, dx} \tag{2}$$

dengan x adalah nilai pada domain output tindakan scaling, $\mu(x)$ adalah derajat keanggotaan hasil agregasi aturan fuzzy pada nilai x, dan output adalah nilai keputusan scaling hasil defuzzifikasi centroid. Nilai output dibulatkan ke bilangan bulat terdekat dan ditambahkan ke jumlah replika aktif. Jumlah replika akhir dibatasi pada rentang minimum 1 dan maksimum 10 replika menggunakan Persamaan (3).

$$desiredReplica = min(max(currentReplicas + round(output),1),10) \tag{3}$$

Dengan *currentReplicas* adalah jumlah replika aktif sebelum keputusan scaling diterapkan, *round*(output) adalah hasil pembulatan keputusan fuzzy ke bilangan bulat terdekat, sedangkan fungsi *min* dan *max* digunakan untuk membatasi jumlah replika akhir pada rentang 1 hingga 10 replika.

Dataset dan *Preprocessing*

Beban kerja pengujian menggunakan dataset BitBrains GWA-T-12 yang berisi data penggunaan sumber daya dari 1.250 Virtual Machine (VM) pada infrastruktur data center selama Agustus hingga September 2013 [15]. Preprocessing dilakukan melalui seleksi VM dengan rata-rata CPU >= 30%, agregasi temporal 5 menit, resampling, dan interpolasi linear untuk mengisi nilai yang hilang. Ambang CPU >= 30% digunakan agar profil yang diproses merepresentasikan beban aktif dan tidak didominasi oleh VM dengan penggunaan CPU rendah.

Konversi CPU dataset ke jumlah VU dilakukan melalui kalibrasi langsung pada infrastruktur pengujian. Nilai CPU dataset kemudian dipetakan ke jumlah VU menggunakan interpolasi linear dan dibatasi pada rentang 2 hingga 10 VU. Profil hasil preprocessing digunakan sebagai sumber beban yang sama pada kedua skenario agar perbandingan S1 dan S2 dilakukan pada kondisi beban yang setara.

Skenario Pengujian dan Metrik Evaluasi

Pengujian dilakukan pada dua skenario, yaitu S1 sebagai autoscaling reaktif berbasis rasio HPA dan S2 sebagai autoscaling berbasis Fuzzy Logic Mamdani. Pada S1, jumlah replika dihitung dengan mengadaptasi formula inti HPA sebagaimana ditunjukkan pada Persamaan (4) [3].

$$desiredReplica = \frac{cpuAktual}{targetCPU} \; x \; currentReplicas \quad (4)$$

Dengan cpuAktual adalah rata-rata CPU *usage* aktual/saat ini, targetCPU adalah target utilitas CPU sebesar 70%, dan *currentReplicas* adalah jumlah replika aktif. Hasil perhitungan dibatasi pada rentang 1 hingga 10 replika. Kedua skenario menggunakan sumber beban, durasi profil, replika awal, batas replika, interval evaluasi, window CPU, stabilization window, dan sumber metrik yang sama. Setiap skenario dijalankan secara terpisah dari kondisi awal yang sama agar hasil evaluasi hanya dipengaruhi oleh perbedaan mekanisme keputusan scaling. S1 menggunakan rasio HPA, sedangkan S2 menggunakan inferensi Fuzzy Logic Mamdani.

Evaluasi kinerja menggunakan empat metrik utama. Rata-rata response time digunakan untuk mengukur durasi rata-rata request HTTP yang dicatat oleh k6. RPS digunakan untuk mengukur throughput berupa jumlah request yang berhasil diproses per detik. Rata-rata jumlah replika digunakan sebagai indikator efisiensi sumber daya. Total scaling event digunakan sebagai indikator stabilitas autoscaler, yaitu jumlah eksekusi perubahan replika baik scale up maupun scale down.

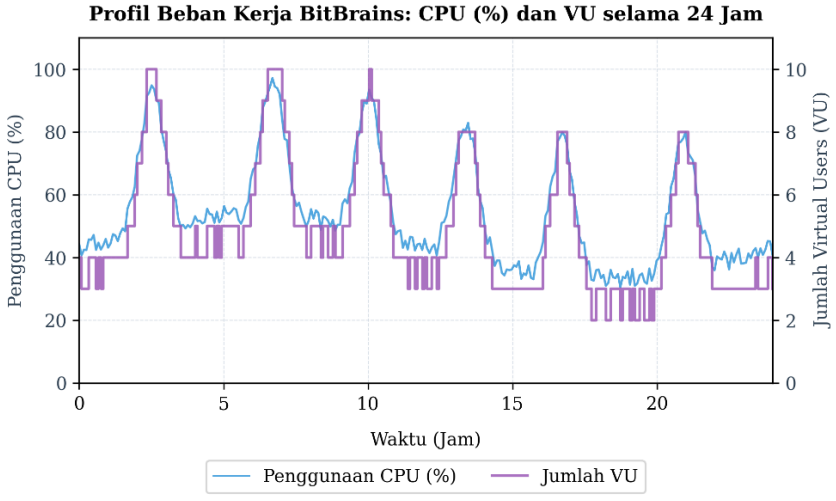
HASIL DAN PEMBAHASAN

Bagian ini menyajikan hasil komparatif antara autoscaling reaktif berbasis rasio HPA (S1) dan autoscaling berbasis Fuzzy Logic Mamdani (S2). Kedua skenario dijalankan menggunakan profil beban yang sama sehingga perbedaan hasil dapat dikaitkan dengan mekanisme pengambilan keputusan scaling.

Hasil *Preprocessing* dan Kalibrasi Beban

Hasil preprocessing dataset BitBrains menghasilkan satu time series CPU yang dikonversi menjadi profil Virtual User (VU) k6. Profil beban kerja diambil dari tanggal 4 September 2013 pukul 00:00 hingga 23:55 dan terdiri dari 288 tahap berdurasi 5 menit. Dengan demikian, profil tersebut merepresentasikan pengujian selama 24 jam sebagaimana ditunjukkan pada Gambar 2.

Hasil kalibrasi langsung pada infrastruktur pengujian menunjukkan bahwa 1 VU menghasilkan CPU sekitar 16,9%, 3 VU menghasilkan 38,4%, 5 VU menghasilkan 56,5%, dan 10 VU menghasilkan 95,1%. Berdasarkan hasil kalibrasi tersebut, nilai CPU dataset dipetakan ke jumlah VU menggunakan interpolasi linear dan dibatasi pada rentang 2 hingga 10 VU. Penggunaan profil yang sama pada S1 dan S2 memastikan bahwa perbandingan tidak dipengaruhi oleh perbedaan intensitas beban, melainkan oleh perbedaan mekanisme keputusan scaling.



Gambar 2. Profil Beban Kerja dan Hasil Pemetaan

Gambar 2 memperlihatkan pola beban yang berubah sepanjang periode pengujian. Kondisi seperti ini relevan untuk mengevaluasi *autoscaler* reaktif karena sistem harus merespons kenaikan dan penurunan permintaan tanpa menimbulkan perubahan replika yang terlalu sering. Pada S1, perubahan replika ditentukan oleh rasio CPU aktual terhadap target CPU. Pada S2, keputusan scaling tidak hanya mempertimbangkan posisi CPU *usage* terhadap kategori *Low, Medium, dan High,* tetapi juga arah perubahannya melalui CPU *trend*.

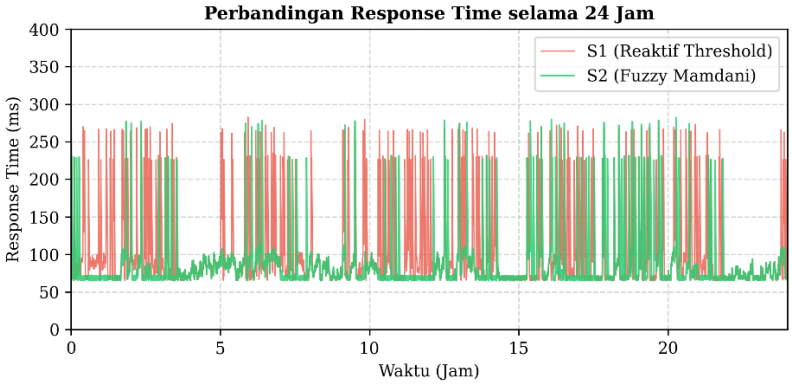
Perbandingan Kinerja

Hasil pengujian komparatif disajikan pada Tabel 4. Secara umum, autoscaling fuzzy menghasilkan response time yang lebih rendah, RPS yang lebih tinggi, dan scaling event yang lebih sedikit dibandingkan baseline reaktif. Namun, pendekatan fuzzy menggunakan rata-rata replika yang sedikit lebih tinggi.

Tabel 4. Hasil Pengujian Skenario

Metrik	S1	S2
Rata-rata *Response Time*	108,40 ms	87,04 ms
Rata-rata RPS (*request per second*)	24,75 req/s	26,75 req/s
Rata-rata Replika	4,49	4,85
Total *Scaling Event*	237	113

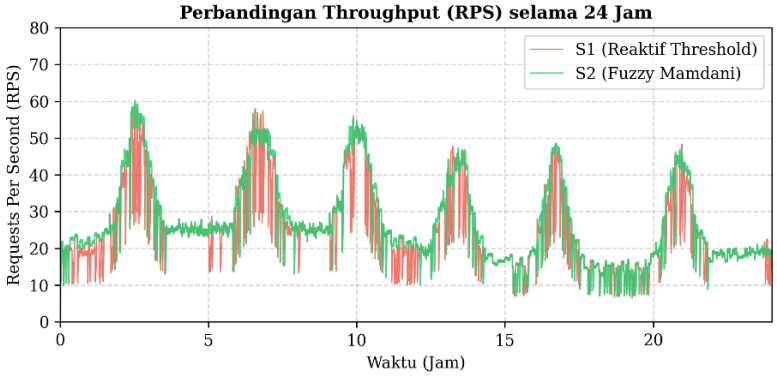
Pada metrik response time, S2 menghasilkan rata-rata 87,04 ms, lebih rendah dibandingkan S1 sebesar 108,40 ms. Penurunan ini setara dengan 19,7% atau 21,36 ms. Grafik response time 24 jam ditunjukkan pada Gambar 3.



Gambar 3. Perbandingan *Response Time*

Penurunan response time dapat dijelaskan oleh dua faktor utama. Pertama, fuzzy autoscaler menghasilkan scaling event yang lebih sedikit sehingga potensi penalti latensi transien akibat inisiasi kontainer dan konvergensi routing mesh Docker Swarm dapat berkurang. Kedua, rata-rata replika pada S2 sedikit lebih tinggi sehingga kapasitas layanan lebih antisipatif ketika terjadi peningkatan beban. Kombinasi kedua faktor tersebut membantu mengurangi risiko antrean *request* pada aplikasi target. Hal ini sejalan dengan studi Casalicchio [8] yang menunjukkan bahwa pemilihan metrik dan mekanisme autoscaling berpengaruh langsung terhadap kinerja aplikasi kontainer yang CPU *intensive*.

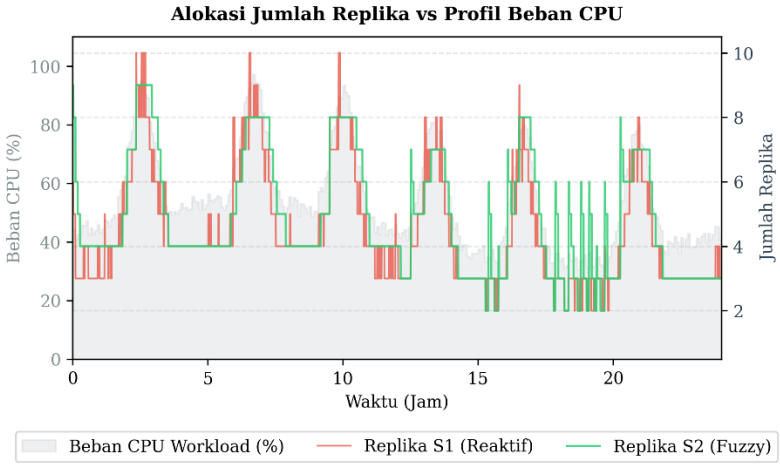
Pada metrik *throughput* atau RPS, S2 menghasilkan rata-rata RPS sebesar 26,86 req/s, lebih tinggi dibandingkan S1 sebesar 24,75 req/s. Peningkatan ini setara dengan 8,5%. Grafik RPS 24 jam ditunjukkan pada Gambar 4.



Gambar 4. Perbandingan RPS

Peningkatan RPS berkaitan dengan *response time* yang lebih rendah. Ketika request dapat diselesaikan lebih cepat, virtual user pada k6 dapat menyelesaikan siklus request-sleep dalam waktu lebih singkat sehingga throughput transaksi per detik meningkat. Dengan demikian, pendekatan fuzzy tidak hanya menurunkan latency, tetapi juga meningkatkan kemampuan sistem dalam memproses request pada sumber daya klaster yang sama. Peningkatan throughput ini tidak muncul karena profil beban yang berbeda, sebab kedua skenario memakai profil VU yang sama, tetapi karena kapasitas layanan pada S2 lebih mampu mengikuti perubahan beban.

Perbedaan alokasi replika disajikan pada Gambar 5. S1 menggunakan rata-rata 4,49 replika, sedangkan S2 menggunakan rata-rata 4,85 replika. Kenaikan rata-rata replika pada S2 sekitar 8,02% dan menjadi *tradeoff* dari peningkatan kualitas layanan.



Gambar 5. Perbandingan Jumlah Replika

Pendekatan fuzzy cenderung memberikan respons *scaling* yang lebih gradual karena mempertimbangkan derajat keanggotaan dan arah CPU trend. Ketika CPU usage tinggi tetapi trend menurun, sistem dapat mempertahankan replika. Sebaliknya, ketika CPU usage menengah tetapi trend meningkat, sistem dapat menambah replika dalam jumlah kecil sebagai tindakan antisipatif. Perilaku ini berbeda dari baseline reaktif yang hanya menghitung replika berdasarkan rasio CPU aktual terhadap target utilitas pada saat evaluasi.

Kenaikan rata-rata replika pada S2 sebesar 0,36 replika menunjukkan adanya tambahan kapasitas yang digunakan oleh pendekatan fuzzy. Tambahan ini menjadi konsekuensi dari keputusan scaling yang lebih antisipatif. Dengan tambahan kapasitas tersebut, S2 mampu menghasilkan response time yang lebih rendah dibandingkan baseline reaktif. Pengaruh pendekatan fuzzy terhadap stabilitas perubahan replika dibahas pada bagian berikutnya.

Analisis Stabilitas *Scaling*

Stabilitas autoscaler dievaluasi menggunakan jumlah scaling event, yaitu jumlah perubahan replika yang terjadi selama periode pengujian. Setiap perubahan jumlah replika, baik scale up maupun scale down, dihitung sebagai satu event. Metrik ini digunakan karena autoscaler yang terlalu sering mengubah replika dapat

menimbulkan fluktuasi kapasitas dan overhead orkestrasi pada Docker Swarm. Rincian scaling event ditunjukkan pada Tabel 5.

Tabel 5. Perbandingan *Scaling Event*

Metrik	S1	S2
Scale Up	124	53
Scale Down	113	60
Total	237	113

S2 menghasilkan total scaling event sebanyak 113 kali, sedangkan S1 menghasilkan 237 kali. Dengan demikian, fuzzy autoscaler mereduksi scaling event sekitar 52,3%. Jika dilihat lebih rinci, scale up event turun dari 124 menjadi 53 kali, sedangkan scale down event turun dari 113 menjadi 60 kali. Penurunan pada dua arah scaling ini menunjukkan bahwa fuzzy autoscaler tidak hanya lebih hati-hati ketika menambah replika, tetapi juga lebih stabil ketika mengurangi replika.

Penurunan scaling event menunjukkan bahwa penggunaan CPU trend dan deadband pada kategori Stable membantu meredam fluktuasi jangka pendek. Sistem tidak langsung mengubah replika hanya karena perubahan kecil di sekitar target, tetapi mempertimbangkan apakah beban sedang naik, stabil, atau turun. Pada baseline S1, penggunaan fungsi rasio dan pembulatan ke atas membuat perubahan CPU di sekitar target lebih mudah menghasilkan perubahan replika. Pada S2, nilai CPU yang berada di daerah transisi dapat menghasilkan derajat keanggotaan pada lebih dari satu kategori, sehingga keputusan akhir menjadi lebih gradual.

Penurunan scaling event juga penting dalam konteks Docker Swarm. Setiap perubahan jumlah replika dapat memicu proses penjadwalan kontainer, inisiasi container baru, update service state, dan konvergensi routing mesh. Jika event terjadi terlalu sering, sistem berpotensi mengalami osilasi dan overhead orkestrasi. Oleh karena itu, pengurangan event pada S2 menunjukkan bahwa fuzzy autoscaler lebih stabil dalam menghadapi profil beban fluktuatif. Temuan ini mendukung pandangan bahwa evaluasi autoscaling kontainer tidak cukup hanya melihat latency atau throughput, tetapi juga perlu mempertimbangkan stabilitas perubahan kapasitas [7][15].

Pembahasan Hasil

Hasil penelitian menunjukkan bahwa autoscaling reaktif berbasis Fuzzy Logic Mamdani menghasilkan peningkatan kualitas layanan dibandingkan autoscaling reaktif berbasis rasio HPA. Keunggulan utama pendekatan fuzzy terletak pada tiga aspek. Pertama, keputusan scaling bersifat gradual karena setiap nilai input dapat memiliki derajat keanggotaan pada lebih dari satu kategori. Kedua, CPU trend memberikan konteks temporal sehingga sistem dapat membedakan kondisi beban yang sedang naik, stabil, atau turun. Ketiga, *rule base* fuzzy menghasilkan keputusan yang lebih mudah dijelaskan secara linguistik, misalnya kondisi CPU *usage* tinggi dan CPU *trend* meningkat menghasilkan tindakan *Scale Up Large*.

Hasil ini sejalan dengan temuan Silva et al. [2] yang menunjukkan bahwa fuzzy logic Mamdani dapat meningkatkan kinerja autoscaling dibandingkan baseline berbasis *threshold* pada Kubernetes. Temuan ini juga konsisten dengan penelitian autoscaling kontainer yang menekankan bahwa pemilihan metrik dan mekanisme keputusan berpengaruh langsung terhadap response time dan stabilitas elastisitas [7][8][16]. Perbedaannya, penelitian ini menempatkan pendekatan fuzzy pada Docker Swarm, yaitu platform yang tidak menyediakan autoscaling bawaan. Dengan demikian, kontribusi penelitian tidak hanya terletak pada penggunaan fuzzy logic, tetapi juga pada adaptasi mekanisme autoscaling yang dapat diterapkan pada lingkungan Docker Swarm berskala kecil hingga menengah.

Dibandingkan penelitian Docker Swarm berbasis AmRP oleh Huang et al. [4], pendekatan pada penelitian ini tetap berada dalam kategori reaktif karena tidak menggunakan model prediksi beban. Keunggulannya adalah rancangan lebih sederhana dan mudah dijelaskan melalui rule base. Keterbatasannya, sistem belum memiliki kemampuan prediktif untuk mengantisipasi lonjakan yang sangat mendadak sebelum perubahan CPU terdeteksi. Dengan demikian, pendekatan fuzzy Mamdani lebih sesuai diposisikan sebagai peningkatan terhadap autoscaling reaktif, bukan sebagai pengganti penuh bagi metode prediktif.

Meskipun demikian, penelitian ini memiliki beberapa keterbatasan. Fungsi keanggotaan dan *rule base* masih dirancang manual berdasarkan literatur dan karakteristik sistem, sehingga belum tentu optimal untuk seluruh jenis beban kerja. Pengujian juga dilakukan pada satu aplikasi target, satu profil beban kerja utama dari

dataset BitBrains, dan klaster Docker Swarm berskala kecil. Selain itu, efisiensi sumber daya masih direpresentasikan melalui rata-rata replika dan belum mencakup analisis biaya operasional secara langsung.

SIMPULAN

Berdasarkan hasil penelitian, autoscaling reaktif berbasis Fuzzy Logic Mamdani berhasil diimplementasikan pada Docker Swarm menggunakan dua variabel input, yaitu CPU usage dan CPU trend, serta sembilan aturan inferensi linguistik. Dibandingkan autoscaling reaktif berbasis rasio HPA, pendekatan fuzzy menurunkan rata-rata response time sebesar 19,7%, meningkatkan rata-rata RPS sebesar 8,5%, dan mereduksi total scaling event sebesar 52,3%. Peningkatan kualitas layanan tersebut diperoleh dengan konsekuensi rata-rata replika yang sedikit lebih tinggi, yaitu 4,85 dibandingkan 4,49 pada baseline reaktif. Keunggulan utama pendekatan fuzzy terletak pada keputusan scaling yang lebih gradual, kontekstual terhadap arah perubahan beban, dan dapat dijelaskan melalui aturan linguistik. Penelitian selanjutnya dapat mengeksplorasi optimasi fungsi keanggotaan secara otomatis, penambahan variabel input seperti *response time* dan memori, pengujian pada profil beban yang lebih beragam, serta perbandingan dengan metode inferensi Sugeno.

DAFTAR PUSTAKA

[1] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, dan P. Merle, "Autonomic Vertical Elasticity of Docker Containers with ELASTICDOCKER," dalam *Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, IEEE, 2017, hlm. 472–479. doi: 10.1109/CLOUD.2017.67.

[2] S. N. Silva, M. A. S. de S. Goldbarg, L. M. D. da Silva, dan M. A. C. Fernandes, "Application of Fuzzy Logic for Horizontal Scaling in Kubernetes Environments within the Context of Edge Computing," *Future Internet*, vol. 16, no. 9, hlm. 316, 2024, doi: 10.3390/fi16090316.

[3] Kubernetes, "Horizontal Pod Autoscaling," 2026. [Daring]. Tersedia pada: https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/

[4] E. H. Mamdani dan S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *Int. J. Man. Mach. Stud.*, vol. 7, no. 1, hlm. 1–13, 1975.

[5] Q. Huang, S. Wang, dan Z. Ding, "Autoscaling Method for Docker Swarm Towards Bursty Workload," *Computing and Informatics*, vol. 42, no. 5, hlm. 1037–1059, 2023, doi: 10.31577/cai_2023_5_1037.

[6] R. J. Hyndman dan G. Athanasopoulos, *Forecasting: Principles and Practice*, 3 ed. Melbourne: OTexts, 2021. [Daring]. Tersedia pada: https://otexts.com/fpp3/

[7] T. T. Nguyen, Y. J. Yeom, T. Kim, D. H. Park, dan S. Kim, "Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration," *Sensors*, vol. 20, no. 16, hlm. 4621, 2020, doi: 10.3390/s20164621.

[8] E. Casalicchio dan V. Perciballi, "Auto-Scaling of Containers: The Impact of Relative and Absolute Metrics," dalam *Proceedings of the 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, IEEE, 2017, hlm. 207–214. doi: 10.1109/FAS-W.2017.149.

[9] E. Casalicchio, "A Study on Performance Measures for Auto-Scaling CPU-Intensive Containerized Applications," *Cluster Comput.*, vol. 22, hlm. 995–1006, 2019, doi: 10.1007/s10586-018-02890-1.

[10] D. Maia, F. Correia, dan P. Queiroz, "Configurational Patterns of Container Orchestration," dalam *Proceedings of the 29th European Conference on Pattern Languages of Programs (EuroPLoP 2024)*, New York, NY, USA: ACM, 2024. doi: 10.1145/3698322.3698342.

[11] L. Baresi, S. Guinea, A. Leva, dan G. Quattrocchi, "A Discrete-Time Feedback Controller for Containerized Cloud Applications," dalam *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*, ACM, 2016, hlm. 217–228. doi: 10.1145/2950290.2950328.

[12] M. Nardelli, V. Cardellini, dan E. Casalicchio, "Multi-Level Elastic Deployment of Containerized Applications in Geo-Distributed Environments," dalam *Proceedings of the 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, IEEE, 2018, hlm. 1–8. doi: 10.1109/FiCloud.2018.00009.

[13] N. M. Dang-Quang dan M. Yoo, "Deep Learning-Based Autoscaling Using Bidirectional Long Short-Term Memory for Kubernetes," *Applied Sciences*, vol. 11, no. 9, hlm. 3835, 2021, doi: 10.3390/app11093835.

[14] X. Peng, B. Pernici, dan M. Vitali, "Virtual Machine Profiling for Analyzing Resource Usage of Applications," dalam *Services Computing - SCC 2018*, vol. 10969, dalam Lecture Notes in Computer Science, vol. 10969. , Cham: Springer, 2018, hlm. 103–118. doi: 10.1007/978-3-319-94376-3_7.

[15] S. Shen, V. van Beek, dan A. Iosup, "GWA-T-12 BitBrains," 2015. [Daring]. Tersedia pada: http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains

[16] F. Zhang, X. Tang, X. Li, S. U. Khan, dan Z. Li, "Quantifying Cloud Elasticity with Container-Based Autoscaling," *Future Generation Computer Systems*, vol. 98, hlm. 672–681, 2019, doi: 10.1016/j.future.2018.09.009.