Vol.6 No.2 Tahun 2025
Jurnal Kecerdasan Buatan, Komputasi dan Teknologi Informasi
https://ejournal.unuja.ac.id/index.php/core
E-ISSN : 2774-7875 and P-ISSN : 2775-0124

Optimasi *Task Scheduling* dengan *Enhanced Whale Optimization* pada *Cloud* dan MEC

Ahmad Caesar Oktavio[1], Rico Rahmat Hendriyanto[2], Muhaimin Yahya[3], Moch Yasin[4]
[1,2,3,4] Fakultas Sains dan Teknologi, Program Studi Sistem, Universitas Islam Negeri Sunan Ampel, Surabaya, Indonesia

Info Artikel

Riwayat Artikel
Diterima: 22-10-2025
Disetujui: 27-12-2025

Kata Kunci
Penjadwalan Tugas;
Optimasi;
Enhanced Whale Optimization Algorithm;
Cloud Computing;
MEC;

moch.yasin@uinsa.ac.id

ABSTRAK

Integrasi *Cloud Computing* dan *Mobile Edge Computing* (MEC) menghadapi tantangan krusial dalam penjadwalan tugas (*task scheduling*), di mana distribusi beban yang tidak efisien sering menyebabkan tingginya latensi dan kehilangan paket data. Selain itu, algoritma konvensional sering kali terjebak dalam optima lokal sehingga gagal memberikan solusi penjadwalan terbaik. Penelitian ini bertujuan untuk mengatasi masalah tersebut dengan menerapkan metode *Enhanced Whale Optimization Algorithm* (EWOA). Untuk menguji efektivitasnya, serangkaian simulasi dilakukan menggunakan prototipe media uji berbasis *web* yang memvisualisasikan pembagian beban kerja di antara *server* heterogen. Kinerja algoritma dievaluasi berdasarkan parameter *throughput*, *packet loss*, dan latensi. Hasil simulasi menunjukkan bahwa EWOA mampu menyalurkan beban secara merata, menghasilkan *throughput* 100% dan *packet loss* nol. Metode ini terbukti memiliki kinerja yang lebih efisien dan stabil dibandingkan alokasi acak, *Round Robin*, dan alokasi statis. Hal ini mengindikasikan bahwa penerapan logika EWOA merupakan alternatif yang efektif untuk meningkatkan pemanfaatan sumber daya dan kinerja sistem dalam lingkungan *Cloud* dan MEC.

1. PENDAHULUAN

Perkembangan komputasi modern bergeser dari model *cloud* terpusat menuju paradigma terdistribusi *Mobile Edge Computing* (MEC). Meskipun *cloud computing* skalabel, arsitektur terpusatnya menyebabkan latensi tinggi bagi aplikasi IoT dan waktu-nyata akibat beban data masif yang ditransfer ke pusat data [1]. MEC mengatasi kendala ini dengan menempatkan sumber daya di tepi jaringan, sehingga mengurangi latensi dan menyediakan *bandwidth* tinggi [2]. Integrasi *Cloud*-MEC menciptakan lingkungan hibrida yang kuat: tugas sensitif latensi dieksekusi di *edge*, sementara tugas intensif komputasi dialihkan ke *cloud*.

Efisiensi lingkungan *Cloud*-MEC sangat bergantung pada penjadwalan tugas (*task scheduling*), sebuah masalah optimasi NP-hard yang kompleks [3]. Kompleksitas ini meningkat akibat heterogenitas perangkat *edge* dan beban kerja dinamis yang sulit diprediksi. Selain itu, penjadwalan ini bersifat multi-objektif karena harus menyeimbangkan pengalaman pengguna dan keuntungan penyedia layanan [4]. Hal ini menuntut *trade-off* cermat untuk meminimalkan *makespan*, menekan biaya, dan memaksimalkan utilisasi sumber daya.

Untuk menangani kompleksitas tersebut, algoritma metaheuristik seperti *Particle Swarm Optimization* (PSO) dan *Genetic Algorithm* (GA) sering digunakan. Namun, pendekatan ini memiliki kelemahan fundamental, yaitu kecepatan konvergensi yang rendah dan kecenderungan terjebak dalam optima lokal [5] [6]. Untuk mengatasi keterbatasan ini, penelitian ini mengusulkan *Enhanced Whale Optimization Algorithm* (EWOA). EWOA disempurnakan dengan pemetaan kaotik dan faktor konvergensi non-linear untuk meningkatkan kemampuan pencarian global dan menghindari konvergensi prematur, sehingga menghasilkan

Commented [O1]: Your article exceeds the specified limit. Try to omit some unimportant material/issues from your article so that it is not too long. If you look at the guidelines, there is a specified page limit.

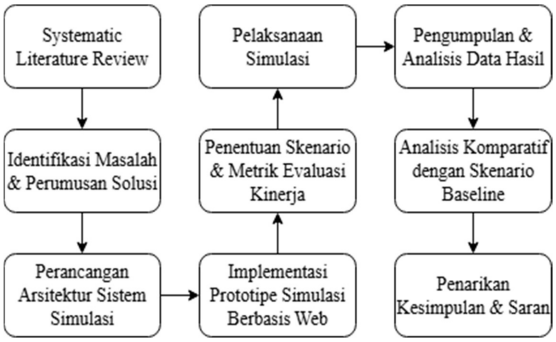
Commented [ca2R1]: done

solusi penjadwalan yang lebih unggul [7].

2. METODE

Bab ini menguraikan metodologi penelitian yang digunakan untuk merancang, mengimplementasikan, dan mengevaluasi kinerja logika penjadwalan tugas yang terinspirasi dari *Enhanced Whale Optimization Algorithm* (EWOA). Proses penelitian diawali dengan *Systematic Literature Review* (SLR), sebuah metode sistematis untuk mengumpulkan dan mensintesis temuan dari studi-studi relevan, yang bertujuan untuk mengidentifikasi kelemahan pada algoritma penjadwalan yang ada, seperti kecenderungan terjebak dalam optima lokal. Berdasarkan identifikasi masalah tersebut, solusi yang dirumuskan adalah pengembangan sebuah prototipe simulasi berbasis web untuk menguji efektivitas pendekatan EWOA. Salah satu bentuk pengembangan algoritma ini adalah *Modified Whale Optimization Algorithm* (MWOA), yang terbukti efisien untuk mekanisme *load balancing* dengan tujuan mengurangi kelemahan yang ada pada algoritma optimasi lain seperti PSO, GOA, dan GWO [8].

Selanjutnya, arsitektur sistem simulasi tersebut dirancang dan diimplementasikan menggunakan teknologi *web* standar (HTML, CSS, JavaScript) untuk merepresentasikan lingkungan server yang heterogen. Tahap validasi dilakukan melalui eksperimen simulasi, di mana kinerja pendekatan EWOA dievaluasi menggunakan skenario pengujian yang telah ditentukan dan dibandingkan secara langsung dengan skenario baseline (alokasi statis). Kinerja diukur berdasarkan metrik kunci seperti *Packet Loss*, Latensi, dan *Throughput*. Keseluruhan alur metodologi penelitian ini, mulai dari tahap awal hingga penarikan kesimpulan, dirangkum secara visual pada Gambar 1.



Gambar 1. Diagram Alir Metodologi Penelitian

Alur penelitian diawali dengan Tinjauan Pustaka Sistematis (SLR) untuk membangun landasan teoritis dan mengidentifikasi masalah pada algoritma penjadwalan yang ada. Berdasarkan temuan tersebut, dilakukan perumusan solusi berupa adaptasi algoritma EWOA. Tahap selanjutnya adalah perancangan arsitektur sistem simulasi berbasis *web*, yang kemudian diimplementasikan menjadi sebuah prototipe fungsional. Setelah itu, skenario pengujian dan metrik evaluasi kinerja ditetapkan untuk mengukur efektivitas algoritma. Simulasi dijalankan sesuai skenario, dan data hasilnya dikumpulkan untuk dianalisis secara kuantitatif. Analisis ini mencakup perbandingan antara skenario yang dioptimalkan dengan skenario dasar (tidak dioptimalkan) untuk memvalidasi keunggulan pendekatan yang diusulkan. Terakhir, kesimpulan dirumuskan dari hasil analisis untuk menjawab tujuan penelitian dan memberikan saran untuk pengembangan di masa depan.

2.1. Tinjauan Pustaka Sistematis (SLR)

Penelitian ini menerapkan *Systematic Literature Review* (SLR) untuk mengidentifikasi celah penelitian (*research gap*) pada algoritma penjadwalan tugas di lingkungan terdistribusi. Hasil analisis awal mengonfirmasi bahwa banyak algoritma metaheuristik konvensional seringkali tidak efisien akibat terjebak dalam optima lokal [5]. Untuk mengatasi kendala tersebut, tren penelitian saat ini mengarah pada pengembangan algoritma hibrida dan adaptif. Pendekatan seperti hibridisasi PSO dengan *Grasshopper Optimization Algorithm* (HPSO-GOA) [5] dan hibridisasi *Differential Evolution* dengan PSO [3] terbukti efektif meningkatkan eksplorasi global. Selain itu, strategi parameter dinamis juga banyak diterapkan untuk menghindari konvergensi prematur, misalnya pada *Hybrid Differential Evolution* (HDE) [6], *Improved Discrete PSO* (IDPSO) [10], serta NTSA-PSO yang memanfaatkan fungsi Copula [12].

Secara spesifik terkait *Whale Optimization Algorithm* (WOA), beberapa studi telah melakukan peningkatan signifikan. Pengembangan *Improved WOA* (IWC) untuk *cloud* dengan faktor konvergensi non-linear telah dilakukan [11], sementara EWOA dengan pemetaan kaotik diusulkan untuk lingkungan *edge computing* [7]. Di sisi lain, studi mengenai lingkungan MEC dan 5G menyoroti pentingnya metrik kinerja seperti latensi dan *throughput*. Hal ini telah dibahas dalam survei komprehensif mengenai MEC [2], pengembangan algoritma sadar biaya (*Cost-Aware Genetic*) [1], serta analisis komparatif algoritma penjadwalan pada jaringan 5G [13]. Berdasarkan sintesis tersebut, meskipun banyak algoritma berfokus pada optimasi *makespan* dan energi, penanganan fluktuasi beban secara dinamis di lingkungan *Cloud*-MEC yang heterogen masih menjadi tantangan terbuka. Temuan ini memvalidasi urgensi penelitian ini untuk menerapkan EWOA yang dirancang khusus dengan mekanisme adaptif guna menyeimbangkan beban dan mencegah *packet loss* secara lebih efisien dibandingkan metode konvensional.

2.2. Desain dan Arsitektur Sistem Simulasi

Untuk menguji hipotesis penelitian, sebuah prototipe simulasi kustom dikembangkan menggunakan teknologi web standar. Pendekatan ini dipilih untuk menyediakan representasi visual yang interaktif dan mudah dipahami mengenai proses penyeimbangan beban. Dalam merancang sebuah simulasi, penting untuk melakukan Konfigurasi Lingkungan: Untuk menyelaraskan tujuan Anda, konfigurasikan platform simulasi. Biasanya, ini mencakup pengaturan hirarki perangkat *cloud*, *fog*, dan *edge*, fitur jaringan seperti latensi, *bandwidth*, dan sumber daya komputasi seperti memori, CPU [14]. Sejalan dengan prinsip tersebut, arsitektur sistem simulasi ini dibagi menjadi tiga komponen utama:

1. Antarmuka Pengguna (*Frontend*)
   Dibangun menggunakan HTML dan CSS, antarmuka ini berfungsi sebagai dasbor kontrol dan visualisasi. Pengguna dapat memasukkan parameter simulasi (jumlah pengguna dan volume data), mengamati diagram jaringan secara visual, dan melihat panel metrik kinerja akhir.
2. Logika Simulasi (*Backend*)
   Inti dari simulasi diimplementasikan menggunakan JavaScript. Skrip ini bertanggung jawab untuk menangani input pengguna, menjalankan logika algoritma penjadwalan, menghitung metrik kinerja, dan memperbarui antarmuka secara dinamis.
3. Komponen Jaringan Virtual
   Lingkungan simulasi merepresentasikan jaringan yang terdiri dari sekelompok *server* heterogen. Setiap *server* didefinisikan dengan karakteristik kapasitas dan latensi dasar yang telah ditentukan sebelumnya, seperti yang dirinci pada Tabel 2.

Konfigurasi *server* yang disajikan pada Tabel 1 dirancang secara spesifik untuk merepresentasikan lingkungan komputasi yang heterogen, yang merupakan karakteristik umum dari arsitektur Cloud-MEC di dunia nyata.

Tabel 1. Konfigurasi *Server Virtual* pada Lingkungan Simulasi

No.	Nama *Server*	Kapasitas Maksimum (KB)	Latensi Dasar (ms)
1.	*Server* A	500	20
2.	*Server* B	700	15
3.	*Server* C	400	30

Heterogenitas ini sengaja dibuat dengan memberikan setiap *server* properti yang berbeda dalam hal kapasitas pemrosesan (kemampuan menampung beban) dan latensi dasar (kecepatan respons awal). *Server* B, misalnya, memiliki kapasitas terbesar namun latensi dasarnya bukan yang terendah, sementara *Server* C memiliki kapasitas terkecil namun latensi dasarnya paling tinggi. Pengaturan ini menciptakan skenario pengujian yang realistis dan menantang bagi algoritma penjadwalan, yang harus membuat keputusan cerdas dengan mempertimbangkan *trade-off* antara kapasitas dan kecepatan untuk setiap tugas yang masuk.

2.3. Tahapan Pelaksanaan Simulasi

Eksperimen dalam penelitian ini dilakukan melalui prototipe simulasi berbasis *web* yang telah dirancang secara kustom dan dikembangkan pada server lokal. Pelaksanaan simulasi mengikuti serangkaian tahapan yang terstruktur untuk memastikan data yang dihasilkan akurat dan dapat direplikasi. Protokol eksperimen dalam satu siklus simulasi berjalan sebagai berikut:

1. Inisialisasi Lingkungan Simulasi
   Sesi simulasi diawali dengan antarmuka yang menampilkan keadaan awal sistem, sebagaimana direpresentasikan pada Gambar 2. Pada tahap ini, seluruh *server* virtual (*Server* A, B, dan C) ditampilkan dengan beban awal 0 KB, dan semua panel metrik kinerja (KPI) juga menunjukkan nilai dasar nol.
2. Konfigurasi Skenario Beban Kerja
   Pengguna mengatur parameter beban kerja melalui panel kontrol yang tersedia. Dua input utama yang ditentukan adalah:
   a. Jumlah Pengguna: Merepresentasikan total jumlah tugas yang akan dijadwalkan.
   b. Volume Data per Pengguna: Mendefinisikan ukuran beban komputasi untuk setiap tugas individual.
3. Inisiasi Proses Penjadwalan
   Simulasi dieksekusi ketika pengguna mengaktifkan tombol "Mulai Simulasi". Tindakan ini secara langsung memicu logika algoritma penjadwalan yang tertanam pada skrip *backend*.
4. Eksekusi Algoritma EWOA
   Setelah terpicu, logika EWOA yang disederhanakan berjalan secara iteratif untuk setiap tugas yang telah dikonfigurasi. Proses ini meliputi:
   a. Untuk setiap tugas, algoritma mengidentifikasi server dengan beban terendah saat ini *(lowest current load*) di antara klaster yang tersedia.
   b. Tugas tersebut kemudian dialokasikan ke server yang paling optimal (paling kosong), dan status beban pada server tersebut diperbarui secara dinamis.
   c. Proses ini diulang secara sekuensial hingga seluruh tugas dari semua pengguna berhasil didistribusikan.
5. Kalkulasi Metrik Kinerja
   Segera setelah tugas terakhir selesai dialokasikan, sistem secara otomatis melakukan kalkulasi kuantitatif untuk metrik kinerja akhir berdasarkan kondisi final setiap *server*. Perhitungan ini meliputi:
   a. Total *Packet Loss,* dihitung menggunakan Persamaan (1).
   b. Total Latensi Transmisi, dihitung sebagai agregat latensi dari setiap *server* menggunakan Persamaan (2).
   c. Total *Throughput* Jaringan, dihitung sebagai jumlah data yang berhasil diproses oleh semua *server* menggunakan Persamaan (3).

Commented [O3]: The spacing in your article needs attention, as some paragraphs are inconsistent.
Commented [ca4R3]: done
Commented [O5]: The spacing in your article needs attention, as some paragraphs are inconsistent.
Commented [ca6R5]: done

6. Visualisasi dan Pengumpulan Hasil:
   Hasil akhir dari proses penjadwalan dan kalkulasi metrik kemudian ditampilkan kembali pada antarmuka. Beban akhir pada setiap *server* diperbarui secara visual, dan *dashboard* KPI menampilkan nilai numerik yang dihasilkan, seperti dicontohkan pada Gambar 3. Data kuantitatif inilah yang kemudian dikumpulkan untuk dianalisis lebih lanjut pada tahap analisis komparatif.

2.4. Implementasi Algoritma dan Metrik Kinerja

Tahap selanjutnya dalam metodologi ini adalah implementasi praktis dari algoritma yang diusulkan dan penetapan metrik kuantitatif untuk mengevaluasi kinerjanya. Implementasi ini berpusat pada logika EWOA yang disederhanakan, yang dirancang untuk mendistribusikan tugas secara efisien di antara sumber daya *server virtual*. Untuk mengukur efektivitas dari logika ini, serangkaian metrik kinerja utama (KPI) didefinisikan. Pemilihan metrik ini sejalan dengan tujuan umum dari optimasi penjadwalan, di mana penjadwal tugas sering kali menggunakan serangkaian algoritma optimasi untuk mencapai tujuan yang berbeda seperti memaksimalkan utilisasi sumber daya, meningkatkan efisiensi energi, menyeimbangkan beban kerja, dan meminimalkan waktu penyelesaian [10]. Oleh karena itu, penelitian ini berfokus pada tiga KPI fundamental: *Packet Loss*, Latensi Transmisi, dan *Throughput* Jaringan, yang secara kolektif memberikan gambaran komprehensif tentang kualitas layanan dan efisiensi sistem.

2.4.1. Algoritma yang Diuji

Untuk mengevaluasi kinerja penjadwalan secara komprehensif, penelitian ini mengimplementasikan dan membandingkan beberapa kategori algoritma. Pendekatan utama yang diuji adalah logika penjadwalan yang diusulkan, yaitu implementasi EWOA yang disederhanakan, yang dirancang untuk mendistribusikan beban secara cerdas dan adaptif. Untuk mengukur efektivitasnya secara objektif, algoritma ini dihadapkan pada serangkaian algoritma pembanding yang merepresentasikan pendekatan penjadwalan non-metaheuristik dan fundamental. Pemilihan algoritma pembanding ini bertujuan untuk menciptakan sebuah baseline atau tolok ukur kinerja yang valid, sehingga keunggulan dari pendekatan yang diusulkan dapat divalidasi secara empiris.

2.4.2. Implementasi Logika EWOA yang Disederhanakan

Mengingat tujuan penelitian ini adalah untuk demonstrasi konsep, implementasi algoritma tidak mencakup seluruh kompleksitas matematis dari EWOA. Sebaliknya, penelitian ini mengadopsi pendekatan yang terinspirasi oleh prinsip inti dari *Whale Optimization Algorithm* (WOA). Dalam penelitian terkait, dinyatakan bahwa secara umum, kami pertama-tama memetakan skema penjadwalan tugas kami ke model penjelajahan paus, dan dengan demikian kami bisa mendapatkan solusi yang mendekati optimal menggunakan algoritma WOA [11]. Sejalan dengan ide tersebut, simulasi ini menggunakan pendekatan greedy yang mencerminkan tujuan optimisasi WOA: menemukan sumber daya (*server*) yang paling optimal untuk setiap tugas. Logika ini bekerja sebagai berikut: untuk setiap tugas yang masuk dari pengguna, algoritma secara iteratif mengidentifikasi server dengan beban terendah saat ini (*lowest current load*) dan segera menugaskan tugas tersebut ke *server* itu. Proses ini diulang untuk semua tugas, memastikan total beban didistribusikan se-merata mungkin.

2.4.3. Metrik Kinerja (KPI)

Kinerja sistem dievaluasi berdasarkan tiga metrik utama yang relevan dengan kualitas layanan jaringan. Penggunaan metrik seperti *makespan* dan *throughput* adalah hal yang umum, di mana sebuah studi menyatakan bahwa hasil eksperimen kami menunjukkan bahwa metode hibrida ini secara signifikan mengungguli algoritma mandiri tradisional dalam mengurangi Makespan, yang merupakan ukuran penting dari waktu penyelesaian tugas [10]. Metrik yang digunakan dalam penelitian ini dihitung secara matematis sebagai berikut:

1. *Packet Loss* (Kehilangan Paket)
   Mengukur jumlah data yang ditolak karena *server* melebihi kapasitasnya. Metrik ini adalah indikator utama kemacetan jaringan.

$$PacketLoss = max(0, Beban - Kapasitas)$$

Dimana:

PacketLoss	:	Jumlah data yang hilang (dalam KB).
max()	:	Fungsi matematika yang mengembalikan nilai terbesar dari dua argumen yang diberikan. Tanda koma (,) berfungsi sebagai pemisah antara argumen.
Beban	:	Total beban data yang masuk ke sebuah *server* (dalam KB).
Kapasitas	:	Kapasitas maksimum *server* tersebut (dalam KB).

2. Latensi Transmisi

Model latensi yang disederhanakan di mana latensi total meningkat secara proporsional terhadap rasio beban pada *server*. Ini memodelkan bagaimana *server* yang lebih sibuk membutuhkan waktu lebih lama untuk merespons.

$$Latensi = Latensi_{Dasar} + \left(\frac{Beban}{Kapasitas}\right) \times 10$$

Dimana:

Latensi	:	Total waktu tunda transmisi untuk sebuah *server* (dalam ms).
Latensi_Dasar	:	Waktu tunda awal yang dimiliki *server* tanpa ada beban (dalam ms).
Beban	:	Total beban data yang sedang diproses oleh *server* (dalam KB).
Kapasitas	:	Kapasitas maksimum *server* (dalam KB).
10	:	Faktor pengali konstan yang digunakan dalam model simulasi untuk merepresentasikan peningkatan latensi akibat beban.

3. *Throughput* Jaringan

Merepresentasikan jumlah total data yang berhasil diproses oleh semua *server* dalam klaster. Ini adalah indikator efisiensi pemrosesan jaringan secara keseluruhan.

$$Throughtput = \Sigma min(Beban_{server}, Kapasitas_{server})$$

Dimana:

Throughput	:	Total data yang berhasil diproses oleh semua *server* di jaringan (dalam KB).
Σ	:	Simbol sigma yang berarti penjumlahan total dari semua *server*.
min()	:	Fungsi matematika yang mengembalikan nilai terkecil dari dua argumen yang diberikan. Tanda koma (,) berfungsi sebagai pemisah antara argumen.
Beban_server	:	Beban data yang dialokasikan ke satu *server* spesifik.
Kapasitas_server	:	Kapasitas maksimum dari *server* spesifik tersebut.

2.5. Skenario Pengujian

Untuk mengevaluasi efektivitas algoritma yang diusulkan, serangkaian skenario pengujian dirancang. Melakukan analisis komparatif sangat penting dalam studi simulasi, di mana berbagai algoritma dievaluasi dan kinerjanya dibandingkan satu sama lain untuk menentukan pendekatan yang paling unggul [13]. Berdasarkan prinsip ini, dua skenario utama digunakan:

1. Skenario Teroptimalkan (EWOA)

   Skenario ini menjalankan simulasi menggunakan logika EWOA yang disederhanakan. Sebuah kasus uji standar dijalankan dengan 10 pengguna, di mana setiap pengguna menghasilkan 50 KB data, sehingga total beban jaringan adalah 500 KB.
2. Skenario Tidak Teroptimalkan (*Baseline*)

   Untuk analisis komparatif, hasil dari skenario teroptimalkan dibandingkan dengan skenario dasar di mana semua lalu lintas (beban) diarahkan ke satu *server* tunggal (*Server* A) tanpa adanya mekanisme penyeimbangan beban. Skenario ini dirancang

untuk menunjukkan dampak dari kemacetan pada satu titik dan memvalidasi pentingnya algoritma penjadwalan yang cerdas.

3. HASIL DAN PEMBAHASAN

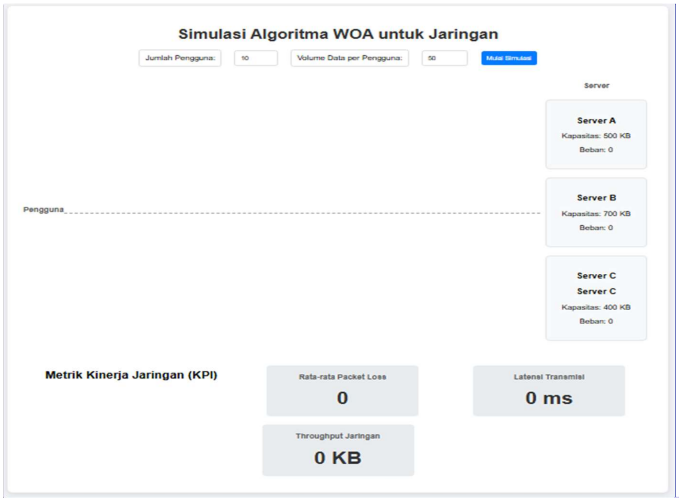
Analisis kinerja dilakukan dengan membandingkan logika distribusi beban berbasis *Enhanced Whale Optimization Algorithm* (EWOA) yang disederhanakan dengan beberapa algoritma pembanding fundamental untuk memvalidasi efektivitasnya secara komprehensif. Pendekatan perbandingan semacam ini sangat penting, karena hasil eksperimental menunjukkan bahwa algoritma yang diusulkan secara signifikan mengungguli algoritma lain dan berhasil mengatasi kekurangan dari WOA klasik [15].

3.1. Pengaturan Skenario dan Eksekusi Simulasi

Simulasi dijalankan dalam lingkungan berbasis *web* yang dirancang khusus untuk penelitian ini, memungkinkan visualisasi interaktif dari proses penyeimbangan beban. Sesuai dengan metodologi yang dijelaskan di Bab 2, arsitektur simulasi terdiri dari antarmuka pengguna (*frontend*), logika simulasi (*backend*), dan komponen jaringan *virtual* dengan tiga *server* heterogen (*Server* A, B, dan C). Untuk analisis awal, sebuah skenario pengujian utama dijalankan dengan parameter sebagai berikut:

1. Jumlah Pengguna: 10
2. Volume Data per Pengguna: 50 KB
3. Total Beban Jaringan: 10 pengguna × 50 KB/pengguna = 500 KB

Inti dari simulasi ini terletak pada implementasi logika EWOA yang disederhanakan. Sebagaimana dijelaskan dalam penelitian terkait, pendekatan WOA dapat diadaptasi di mana secara umum, kami pertama-tama memetakan skema penjadwalan tugas kami ke model penjelajahan paus, dan dengan demikian kami bisa mendapatkan solusi yang mendekati optimal [11]. Alih-alih mengimplementasikan model matematika WOA secara penuh, simulasi ini menggunakan pendekatan *greedy* yang mencerminkan tujuan optimisasi EWOA: menemukan sumber daya (*server*) yang paling optimal untuk setiap tugas. Logikanya bekerja dengan mengidentifikasi *server* dengan beban terendah saat ini (*lowest current load*) dan menugaskan tugas berikutnya ke *server* tersebut. Proses ini diulang untuk semua pengguna, memastikan bahwa total beban didistribusikan seefisien mungkin. Keadaan awal dari skenario simulasi ini digambarkan pada Gambar 2.



Gambar 2. Antarmuka Pengguna yang Menampilkan Parameter Input untuk Simulasi

Commented [O7]: The image size needs to be adjusted so that it can be read and understood.

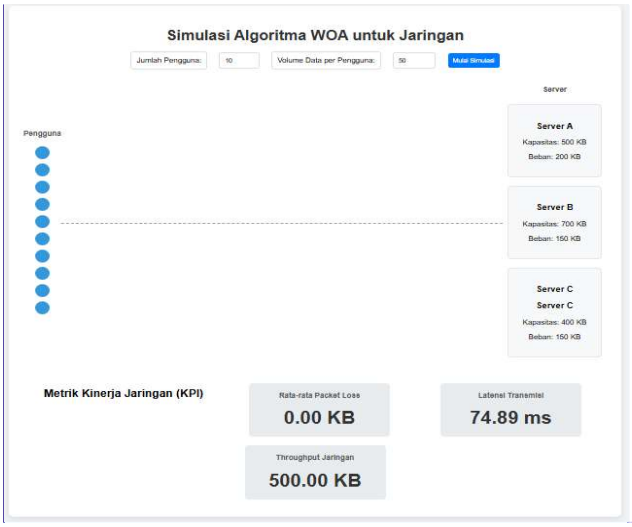
Commented [ca8R7]: done

Gambar 2 menampilkan kondisi awal dari antarmuka simulasi sebelum eksekusi dimulai. Panel kontrol di bagian atas menunjukkan parameter input yang telah ditetapkan, yaitu 10 pengguna dengan masing-masing menghasilkan volume data sebesar 50 KB. Di sisi kanan, klaster *server* yang terdiri dari *Server* A, B, dan C ditampilkan bersama kapasitas heterogen mereka, dengan beban awal (Beban) pada setiap server adalah 0 KB. Di bagian bawah, dasbor Metrik Kinerja Jaringan (KPI) menunjukkan semua metrik Rata-rata *Packet Loss*, Latensi Transmisi, dan *Throughput* Jaringan berada pada nilai dasar nol. Tampilan ini secara efektif merepresentasikan keadaan sistem yang siap menerima beban kerja sebelum algoritma penjadwalan diterapkan.

3.2. Analisis Kinerja Skenario Teroptimalkan

Setelah menjalankan simulasi dengan total beban 500 KB, algoritma EWOA yang disederhanakan berhasil mendistribusikan beban ke tiga server. Alokasi beban yang dihasilkan adalah: 200 KB pada *Server* A, 150 KB pada *Server* B, dan 150 KB pada *Server* C. Distribusi ini berhasil menghindari kelebihan beban pada *server* mana pun, karena alokasi beban pada setiap *server* berada di bawah kapasitas maksimumnya (*Server* A: 500 KB, B: 700 KB, C: 400 KB).

Metrik kinerja akhir (KPI) yang dihasilkan dari skenario ini, seperti yang ditampilkan pada dasbor, ditunjukkan pada Gambar 3 dan dianalisis secara rinci di bawah ini.



Gambar 3. Dashboard KPI yang menunjukkan hasil setelah simulasi dijalankan.

Gambar 3 merepresentasikan hasil akhir dari simulasi setelah algoritma EWOA yang disederhanakan selesai mengeksekusi seluruh tugas. Visualisasi menunjukkan bahwa 10 pengguna (direpresentasikan oleh titik-titik biru) telah berhasil dilayani. Beban total sebesar 500 KB telah didistribusikan secara cerdas ke tiga *server*: *Server* A menampung 200 KB, *Server* B menampung 150 KB, dan *Server* C menampung 150 KB. Yang terpenting, *dashboard* KPI di bagian bawah telah diperbarui dengan hasil kuantitatif dari simulasi, menampilkan nilai akhir untuk Rata-rata *Packet Loss* (0.00 KB), Latensi Transmisi (74.89 ms), dan *Throughput* Jaringan (500.00 KB). Gambar ini memberikan bukti visual dan numerik yang jelas mengenai kinerja algoritma dalam skenario pengujian ini.

1. *Packet Loss* (Kehilangan Paket): 0.00 KB

Commented [O9]: The image size needs to be adjusted so that it can be read and understood.

Commented [ca10R9]: done

Analisis: Hasil ini menunjukkan efektivitas sempurna dari algoritma dalam mencegah kemacetan jaringan. Karena beban pada setiap *server* (A: 200/500 KB, B: 150/700 KB, C: 150/400 KB) tidak pernah melebihi kapasitasnya, tidak ada paket data yang hilang. Ini adalah tujuan utama dari algoritma penjadwalan yang efisien, karena kehilangan paket merupakan masalah kritis yang dapat menurunkan kualitas layanan secara drastis, terutama untuk aplikasi waktu-nyata [13].

2. *Throughput* Jaringan: 500.00 KB
   Analisis: Total throughput jaringan sama persis dengan total volume data yang dihasilkan (500 KB). Ini mengindikasikan bahwa sistem berhasil memproses 100% data yang masuk. Hasil ini merupakan konsekuensi langsung dari tidak adanya packet loss dan menunjukkan bahwa kapasitas gabungan dari klaster *server* dimanfaatkan secara optimal tanpa ada data yang terbuang.
3. Latensi Transmisi: 74.89 ms
   Analisis: Total latensi transmisi adalah hasil penjumlahan latensi dari setiap *server*, yang dihitung berdasarkan bebannya masing-masing:
   a. Latensi A = 20 + (200 / 500) × 10 = 24.00 ms
   b. Latensi B = 15 + (150 / 700) × 10 ≈ 17.14 ms
   c. Latensi C = 30 + (150 / 400) × 10 = 33.75 ms
   d. Total Latensi = 24.00 + 17.14 + 33.75 = 74.89 ms
      Distribusi beban yang cerdas menghasilkan kontribusi latensi yang seimbang dari semua *server*. Hal ini sangat penting karena mencegah satu *server* pun menjadi *bottleneck* (penghambat) yang dapat meningkatkan latensi keseluruhan secara drastis.

Commented [O11]: some fonts are not the same, check all of them in your article

Commented [ca12R11]: done

3.3. Analisis Komparatif Kinerja Algoritma

Untuk memvalidasi efektivitas logika EWOA yang disederhanakan, kinerja algoritma ini dibandingkan secara langsung dengan tiga algoritma pembanding fundamental: Alokasi Statis, Round Robin, dan Alokasi Acak. Perbandingan ini dirancang untuk mengukur sejauh mana pendekatan cerdas EWOA mampu memberikan keunggulan dibandingkan metode-metode non-optimasi dan klasik.

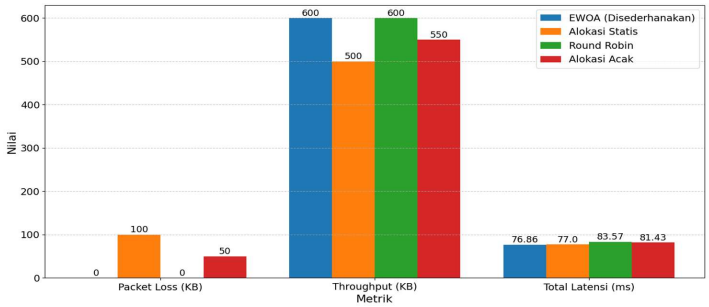
Untuk menyoroti perbedaan kinerja secara signifikan, seluruh algoritma diuji menggunakan skenario beban kerja yang lebih menuntut, yaitu 12 pengguna yang masing-masing mengirimkan 50 KB data, sehingga menciptakan total beban jaringan sebesar 600 KB. Hasil kuantitatif dari eksekusi simulasi untuk keempat algoritma disajikan pada Tabel 3.

Tabel 3. Analisis Komparatif Kinerja pada Beban 600 KB

No.	Metrik	EWOA (Disederhanakan)	Alokasi Statis	Round Robin	Alokasi Acak
1.	*Packet Loss*	0 KB	100 KB	0 KB	50 KB
2.	*Throughput*	600 KB	500 KB	600 KB	550 KB
3.	Total Latensi	76.86 ms	77.00 ms	83.57 ms	81.43 ms

Data yang disajikan pada Tabel 3 secara kuantitatif menunjukkan perbedaan kinerja yang signifikan antara algoritma yang diuji. Algoritma Alokasi Statis secara jelas gagal menangani beban kerja yang diberikan, terbukti dengan adanya *packet loss* sebesar 100 KB dan *throughput* yang terbatas pada 500 KB. Hal ini disebabkan oleh ketidakmampuannya mendistribusikan beban, sehingga menyebabkan satu *server* mengalami kelebihan muatan. Di sisi lain, Alokasi Acak, meskipun lebih baik, masih menunjukkan ketidakstabilan dengan menghasilkan packet loss sebesar 50 KB, yang menegaskan bahwa distribusi tanpa strategi tidak dapat diandalkan.

Algoritma Round Robin berhasil mengatasi masalah *packet loss* dengan menyebarkan tugas secara merata, namun kinerjanya terhambat oleh latensi tertinggi (83.57 ms). Ini menunjukkan bahwa distribusi merata tanpa mempertimbangkan kondisi *server* (beban dan kapasitas) tidak cukup untuk mencapai efisiensi waktu respons yang optimal. Sebaliknya, EWOA yang diusulkan tidak hanya berhasil mencapai nol *packet loss* dan *throughput* maksimal (600 KB), tetapi juga mencatatkan latensi terendah (76.86 ms). Hasil ini secara empiris membuktikan bahwa kemampuan EWOA untuk secara dinamis memilih server dengan beban terendah memberikan keunggulan superior di semua metrik kinerja.



Gambar 4. Grafik Perbandingan Kinerja Algoritma Penjadwalan

Visualisasi pada Gambar 4 secara nyata mengkonfirmasi keunggulan pendekatan EWOA yang diusulkan. Pada metrik *Packet Loss*, balok diagram untuk EWOA dan Round Robin berada pada nilai nol, menunjukkan tidak ada data yang hilang. Sebaliknya, Alokasi Statis menunjukkan balok tertinggi (100 KB), yang secara visual merepresentasikan kegagalan total dalam menangani kelebihan beban.

Pada metrik *Throughput*, balok EWOA dan Round Robin mencapai ketinggian maksimal, yaitu 600 KB, yang menandakan pemrosesan data 100%. Hal ini kontras dengan Alokasi Statis yang baloknya terlihat jelas lebih pendek (500 KB), menunjukkan adanya data yang tidak berhasil diproses.

Namun, keunggulan utama EWOA paling menonjol pada metrik Total Latensi. Meskipun Round Robin berhasil memproses semua data, balok latensinya adalah yang tertinggi di antara semua algoritma. Sebaliknya, balok latensi untuk EWOA secara visual adalah yang terpendek (76.86 ms), membuktikan bahwa algoritma ini tidak hanya efektif dalam mencegah kemacetan tetapi juga paling efisien dalam hal waktu respons.

Perbedaan ketinggian balok diagram di setiap metrik ini memberikan bukti visual yang kuat bahwa kemampuan adaptif EWOA dalam memilih sumber daya secara dinamis adalah faktor kunci yang membuatnya superior. Analisis komparatif ini, baik secara numerik maupun visual, menegaskan bahwa algoritma penyeimbangan beban yang cerdas seperti EWOA sangat krusial untuk menjaga integritas, efisiensi, dan kualitas layanan jaringan di bawah beban kerja yang tinggi.

3.4. Diskusi Implikasi dan Keterbatasan

Berdasarkan hasil analisis komparatif, implementasi EWOA yang disederhanakan menunjukkan beberapa implikasi dan keunggulan signifikan dibandingkan pendekatan penjadwalan fundamental lainnya:

1. Implikasi dan Keunggulan
    a. Pemanfaatan Sumber Daya yang Optimal
    Hasil perbandingan menegaskan bahwa EWOA secara signifikan lebih unggul dalam meningkatkan utilisasi semua sumber daya server yang tersedia. Tidak seperti Alokasi Statis yang menyebabkan satu server terbebani, atau Round Robin dan Alokasi Acak yang distribusinya kurang efisien, EWOA mampu mencegah satu server pun menjadi terbebani sementara yang lain tidak aktif.
    b. Peningkatan Kinerja Jaringan

Kemampuan EWOA untuk menyeimbangkan beban secara cerdas terbukti secara drastis mengurangi packet loss hingga nol dan memaksimalkan throughput. Keunggulan ini sangat menonjol jika dibandingkan dengan Alokasi Statis dan Alokasi Acak yang keduanya mengalami kehilangan paket. Peningkatan kinerja ini secara langsung berdampak pada peningkatan Kualitas Layanan (QoS) bagi pengguna akhir.

c. Skalabilitas dan Ketahanan
Pendekatan algoritmik EWOA pada dasarnya dapat diskalakan. Meskipun hanya diuji pada tiga server, logikanya yang dinamis dalam memilih server berbeban terendah dapat beradaptasi seiring penambahan lebih banyak server. Hal ini membuat jaringan lebih tahan terhadap lonjakan lalu lintas dibandingkan dengan metode kaku seperti Alokasi Statis.

d. Visualisasi yang Efektif
Antarmuka berbasis web memberikan umpan balik visual yang jelas dan langsung mengenai hasil dari setiap algoritma yang diuji. Ini menjadikannya alat yang sangat baik tidak hanya untuk analisis tetapi juga untuk tujuan edukasi, memungkinkan perbandingan langsung antara pendekatan penjadwalan yang cerdas dan yang naif.

2. Keterbatasan Penelitian

a. Penyederhanaan Algoritma
Simulasi ini menggunakan pendekatan *greedy* sebagai proksi untuk EWOA. Implementasi EWOA yang sebenarnya jauh lebih kompleks, melibatkan fase-fase seperti *encircling prey*, *bubble-net attack*, dan *search for prey*, yang secara komputasi lebih intensif tetapi dapat menghasilkan solusi yang lebih optimal di lingkungan yang sangat dinamis [11].

b. Kondisi Jaringan Statis
Simulasi ini tidak memperhitungkan dinamika jaringan dunia nyata seperti *jitter*, variasi *bandwidth*, atau mobilitas pengguna yang dapat mengubah kondisi jaringan secara konstan. Tantangan seperti mobilitas dan migrasi tugas merupakan area penelitian aktif di lingkungan MEC [2].

c. Overhead Komputasi Diabaikan
Logika penjadwalan ini tidak memperhitungkan *overhead* komputasi dari algoritma itu sendiri. Dalam skenario dunia nyata, proses pengambilan keputusan akan mengonsumsi sebagian kecil sumber daya dan menambahkan latensi.

4. KESIMPULAN DAN SARAN

Berdasarkan hasil simulasi dan analisis komparatif yang telah dilakukan, penelitian ini menunjukkan bahwa implementasi logika penjadwalan yang terinspirasi dari EWOA terbukti sangat efektif dalam mendistribusikan beban kerja di antara server-server heterogen. Algoritma yang diusulkan berhasil mencapai kinerja optimal dengan nol packet loss dan 100% throughput, yang menandakan semua tugas berhasil diproses tanpa kehilangan data akibat kemacetan. Analisis komparatif juga menegaskan keunggulan kinerja EWOA secara signifikan dibandingkan dengan algoritma pembanding fundamental seperti Alokasi Statis, Alokasi Acak, dan Round Robin, terutama dalam hal efisiensi latensi di bawah beban kerja yang tinggi.v Meskipun penelitian ini berhasil menunjukkan efektivitas pendekatan EWOA yang disederhanakan, terdapat beberapa area yang dapat dikembangkan untuk penelitian di masa depan. Sebagai saran, direkomendasikan untuk mengimplementasikan versi penuh dari algoritma EWOA untuk mengeksplorasi solusi yang lebih optimal. Selain itu, pengujian pada lingkungan jaringan yang dinamis (misalnya dengan variasi bandwidth atau jitter) akan memberikan wawasan yang lebih dalam mengenai ketahanan algoritma. Penelitian selanjutnya juga dapat diperkaya dengan menganalisis overhead komputasi dari setiap algoritma dan

menambahkan algoritma pembanding dari rumpun metaheuristik lain seperti PSO atau GA untuk validasi kinerja yang lebih komprehensif.

5. DAFTAR PUSTAKA

[1] T. S. Nikoui, A. Balador, A. M. Rahmani, dan Z. Bakhshi, "Cost-Aware Task Scheduling in Fog-Cloud Environment," dalam *2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, Tehran, Iran: IEEE, Jun 2020, hlm. 1–8. doi: 10.1109/RTEST49666.2020.9140118.

[2] A. Filali, A. Abouaomar, S. Cherkaoui, A. Kobbane, dan M. Guizani, "Multi-Access Edge Computing: A Survey," *IEEE Access*, vol. 8, hlm. 197017–197046, 2020, doi: 10.1109/ACCESS.2020.3034136.

[3] S. A. Malleswaran dan K. Parthiban, "A Novel Task Scheduling Algorithm In A Cloud Computing Environment Using Hybrid DE-PSO Algorithm," vol. 9, no. 02, 2020.

[4] S. Chen *dkk.*, "QoS-aware placement of interdependent services in energy-harvesting-enabled multi-access edge computing," *Future Gener. Comput. Syst.*, vol. 174, hlm. 108009, Jan 2026, doi: 10.1016/j.future.2025.108009.

[5] S. Bansal, B. S. Singla, dan H. Aggarwal, "Workflow task scheduling in a cloud-fog environment: a hybrid PSO-GOA approach," *Int. J. Syst. Assur. Eng. Manag.*, Jan 2025, doi: 10.1007/s13198-024-02638-8.

[6] M. Abdel-Basset, R. Mohamed, W. Abd Elkhalik, M. Sharawi, dan K. M. Sallam, "Task Scheduling Approach in Cloud Computing Environment Using Hybrid Differential Evolution," *Mathematics*, vol. 10, no. 21, hlm. 4049, Okt 2022, doi: 10.3390/math10214049.

[7] L. Han, S. Zhu, H. Zhao, dan Y. He, "An enhanced whale optimization algorithm for task scheduling in edge computing environments," *Front. Big Data*, vol. 7, hlm. 1422546, Okt 2024, doi: 10.3389/fdata.2024.1422546.

[8] K. J. Rajashekar dan Channakrishnaraju, "Dynamic Load Balancing in Distributed Storage Systems using Modified Whale Optimization Techniques," *J. Syst. Manag. Sci.*, vol. 13, no. 1, Feb 2023, doi: 10.33168/JSMS.2023.0130.

[9] M. J. Page *dkk.*, "The PRISMA 2020 statement: an updated guideline for reporting systematic reviews," *BMJ*, hlm. n71, Mar 2021, doi: 10.1136/bmj.n71.

[10] A. M. Abdulghani, "Hybrid Task Scheduling Algorithm for Makespan Optimisation in Cloud Computing: A Performance Evaluation," Okt 2024.

[11] X. Chen *dkk.*, "A WOA-Based Optimization Approach for Task Scheduling in Cloud Computing Systems," *IEEE Syst. J.*, vol. 14, no. 3, hlm. 3117–3128, Sep 2020, doi: 10.1109/JSYST.2019.2960088.

[12] Z. Wu dan J. Xiong, "A Novel Task-Scheduling Algorithm of Cloud Computing Based on Particle Swarm Optimization:," *Int. J. Gaming Comput.-Mediat. Simul.*, vol. 13, no. 2, hlm. 1–15, Apr 2021, doi: 10.4018/IJGCMS.2021040101.

[13] M. S. Abdulrahman dan B. M. Omran, "Comparative Performance Evaluation of Scheduling Algorithms using 5G-Air-Simulator," *J. Eng.*, vol. 31, no. 5, hlm. 206–223, Mei 2025, doi: 10.31026/j.eng.2025.05.12.

[14] "iFogSim2 Simulation Ideas," PHD Services. Diakses: 5 Oktober 2025. [Daring]. Tersedia pada: https://phdservices.org/ifogsim2-simulation/

[15] J. Wei, Y. Gu, B. Lu, dan N. Cheong, "RWOA: A novel enhanced whale optimization algorithm with multi-strategy for numerical optimization and engineering design problems," *PLOS One*, vol. 20, no. 4, hlm. e0320913, Apr 2025, doi: 10.1371/journal.pone.0320913.