ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - IF184802

PEMANFAATAN ALGORITMA HEURISTIK UNTUK PENGATURAN BEBAN PADA KOMPUTASI AWAN

RALDO KUSUMA
NRP 05111640000026

Dosen Pembimbing I
Bagus Jati Santoso, S.Kom., Ph.D.

Dosen Pembimbing II
Ary Mazharuddin S, S.Kom., M.Comp.Sc., Ph.D.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - IF184802

PEMANFAATAN ALGORITMA HEURISTIK UNTUK PENGATURAN BEBAN PADA KOMPUTASI AWAN

RALDO KUSUMA
NRP 05111640000026

Dosen Pembimbing I
Bagus Jati Santoso, S.Kom., Ph.D.

Dosen Pembimbing II
Ary Mazharuddin S, S.Kom., M.Comp.Sc., Ph.D.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - IF184802

UTILIZATION OF HEURISTIC ALGORITHM FOR LOAD BALANCING IN CLOUD COMPUTING

RALDO KUSUMA
NRP 05111640000026

Supervisor I
Bagus Jati Santoso, S.Kom., Ph.D.

Supervisor II
Ary Mazharuddin S, S.Kom., M.Comp.Sc., Ph.D.

DEPARTMENT OF INFORMATICS ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

PEMANFAATAN ALGORITMA HEURISTIK UNTUK PENGATURAN BEBAN PADA KOMPUTASI AWAN

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur Jaringan dan Komputer
Program Studi S1 Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh :
RALDO KUSUMA
NRP: 05111640000026

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Bagus Jati Santoso, S.Kom., Ph.D. .....
NIP: 198611252018031001 (Pembimbing 1)

Ary Mazharuddin S, S.Kom., M.Comp.Sc., Ph.D .....
NIP: 198106202005011003 (Pembimbing 2)

SURABAYA
Januari 2020

(Halaman ini sengaja dikosongkan)

PEMANFAATAN ALGORITMA HEURISTIK UNTUK PENGATURAN BEBAN PADA KOMPUTASI AWAN

Nama	:	**RALDO KUSUMA**
NRP	:	**05111640000026**
Departemen	:	**Teknik Informatika FTEIC**
Pembimbing I	:	**Bagus Jati Santoso, S.Kom., Ph.D.**
Pembimbing II	:	**Ary Mazharuddin S, S.Kom., M.Comp.Sc., Ph.D.**

Abstrak

Komputasi awan menjadi salah satu alternatif dalam pembuatan layanan berbasis web. Semakin banyak pengguna internet menyebabkan lalu lintas pengguna layanan web menjadi tinggi yang tentu akan memerlukan infrastruktur yang baik dalam mengelola server di komputasi awan. Load balancing merupakan salah satu tantangan utama dalam komputasi awan dalam mendistribusikan beban kerja di beberapa node untuk memastikan bahwa tidak ada sumber daya tunggal yang kewalahan atau kurang dimanfaatkan.

Load balancing adalah teknik atau cara untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal, memaksimalkan throughput, memperkecil waktu tanggap dan menghindari overload pada salah satu jalur koneksi.

Untuk menyelesaikan permasalahan dalam infrastruktur di komputasi awan diperlukan sebuah algoritma heuristik yang sesuai untuk pembagian beban pada komputasi awan. Hal ini perlu dilakukan sehingga semua server memiliki beban yang seimbang. Dengan demikian server dapat berjalan optimal dan menghindari ada sumber daya tunggal yang kewalahan atau kurang dimanfaatkan.

Dari hasil uji coba, algoritma heuristik dapat membagi beban task secara seimbang ke resource yang disediakan. Algoritma heuristik juga mampu membagi task dengan resource yang memiliki spesifikasi berbeda dan task dengan berat yang berbeda.

***Kata-Kunci**: Load balancing, Heuristic algorithm*

UTILIZATION OF HEURISTIC ALGORITHM FOR LOAD BALANCING IN CLOUD COMPUTING

Name	:	**RALDO KUSUMA**
NRP	:	**05111640000026**
Department	:	**Informatics Engineering ELECTICS**
Supervisor I	:	**Bagus Jati Santoso, S.Kom., Ph.D.**
Supervisor II	:	**Ary Mazharuddin S, S.Kom., M.Comp.Sc., Ph.D.**

Abstract

Cloud computing is an alternative in making web-based services. More internet users causes high web service user traffic which will certainly require good infrastructure in managing servers in cloud computing. Load balancing is one of the main challenges in cloud computing in distributing workload across multiple nodes to ensure that no single resource is overwhelmed or underused.

Load balancing is a technique or way to distribute traffic loads on two or more connection lines in a balanced way, so that traffic can run optimally, maximize throughput, minimize response times and avoid overloading on one of the connection lines.

To solve problems in infrastructure in cloud computing we need a heuristic algorithm that is suitable for sharing the burden on cloud computing. This needs to be done so that all servers have a balanced load. Thus the server can run optimally and avoid having a single resource that is overwhelmed or underused.

From the results of the trial, the heuristic algorithm can divide the balance equally into existing resources. The heuristic algorithm is also able to divide tasks with resources that have different specifications and tasks with different weight.

Keywords: *Load balancing, Heuristic algorithm*

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul **Pemanfaatan Algoritma Heuristik untuk Pengaturan Beban pada Komputasi Awan**. Pengerjaan tugas akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan tugas akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Departemen Informatika ITS. Dengan tugas akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesainya tugas akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Keluarga penulis yang selalu menyemangati.
3. Bapak Bagus Jati Santoso, S.Kom., Ph.D selaku pembimbing I yang telah membantu, membimbing dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya tugas akhir ini.
4. Bapak Ary Mazharuddin S, S.Kom., M.Comp.Sc., Ph.D. selaku pembimbing II yang juga telah membantu, membimbing dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya tugas akhir ini serta selaku Koordinator TA dan segenap dosen Departemen Informatika yang telah memberikan ilmu dan pengalamannya..
5. Ibu Dr.Eng. Chastine Fatichah, S.Kom., M.Kom selaku Kepala Departemen Informatika ITS pada masa

pengerjaan tugas akhir.

6. Teman-teman Administrator Laboratorium Arsitektur dan Jaringan Komputer.
7. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa tugas akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya 2020

Raldo Kusuma

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir yang meliputi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari tugas akhir yang dikerjakan.

1.1 Latar Belakang

Komputasi awan semakin berkembang dengan sangat cepat. Banyak organisasi besar seperti Amazon, Yahoo, dan Google menawarkan layanan cloud dan memiliki banyak pengguna. Generasi berikutnya dari komputasi awan akan berkembang pada seberapa efektif infrastruktur yang digunakan dan sumber daya yang tersedia. *Load balancing* merupakan salah satu tantangan utama dalam komputasi awan dalam mendistribusikan beban kerja di beberapa node untuk memastikan bahwa tidak ada sumber daya tunggal yang kewalahan atau kurang dimanfaatkan.

Load balancing adalah teknik atau cara untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal, memaksimalkan *throughput*, memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi.

Dari permasalahan di atas dibutuhkan algoritma yang digunakan untuk pengaturan beban menggunakan algoritma heuristik. Tugas akhir ini akan menerapkan algoritma heuristik pada pengaturan beban di komputasi awan sehingga hasil dari tugas akhir ini diharapkan dapat menghasilkan algoritma yang tepat untuk memecahkan permasalahan di atas secara optimal dan diharapkan dapat memberikan kontribusi pada ilmu pengetahuan dan teknologi informasi

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana mengimplementasikan algoritma yang dibangun pada permasalahan pembagian beban pada komputasi awan dengan memanfaatkan algoritma heurisik ?
2. Bagaimana memformulasikan *fitness function* untuk diterapkan pada algoritma heuristik pada pembagian beban di komputasi awan?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki batasan antara lain:

1. Implementasi dilakukan dengan bahasa pemrogaman *Python*.
2. Virtualisasi dilakukan dengan menggunakan *Docker*.

1.4 Tujuan

Tujuan pembuatan tugas akhir ini antara lain:

1. Melakukan implementasi algoritma yang dibangun pada permasalahan pembagian beban pada komputasi awan dengan memanfaatkan algoritma heuristik pada pembagian beban.
2. Memformulasikan *fitness function* yang diterapkan pada algoritma heuristik pada pembagian beban di komputasi awan.

1.5 Manfaat

Manfaat yang diharapkan dari tugas akhir ini adalah dapat mendesain dan mengimplementasikan algoritma yang tepat dengan pemanfaatan algoritma heuristik pada pengaturan beban di komputasi awan serta diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.6 Metodologi

Tahap-tahap yang dilakukan dalam pengerjaan tugas akhir ini adalah:

1. **Penyusunan Proposal Tugas Akhir**
   Proposal tugas akhir ini berisi mengenai deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Subbab metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

2. **Studi Literatur**
   Pada tahap ini dilakukan pencarian informasi dan studi literatur yang diperlukan untuk penyelesaian persoalan yang akan dikerjakan. Informasi didapatkan dari materi-materi yang berhubungan dengan algoritma yang

digunakan dalam pengerjaan tugas akhir ini. Materi-materi tersebut didapatkan dari *paper*, internet, maupun buku acuan.

3. **Analisis dan Desain Perangkat Lunak**
   Pada tahap ini penulis akan menganalisa masalah yang ada dalam desain algoritma heuristik pada pengaturan beban di komputasi awan dalam mendistribusikan beban kerja pada node untuk memastikan bahwa tidak ada sumber daya tunggal yang kewalahan atau kurang dimanfaatkan.

4. **Implementasi Perangkat Lunak**
   Implementasi dari desain algoritma akan dibuat dalam bentuk program konsol yang dibangun dengan menggunakan bahasa pemrograman Python.

5. **Uji Coba dan Evaluasi**
   Pengujian dilakukan dengan beberapa cara, antara lain:
   (a) **Pengujian Waktu Eksekusi**
   Pengujian ini berfokus pada berapa lama waktu yang diperlukan untuk menyelesaikan task yang dieksekusi dengan algoritma heuristik pada pengaturan beban di komputasi awan dalam mendistribusikan beban kerja pada node.
   (b) **Pengujian Penggunaan Resource** Pengujian ini akan berfokus pada seberapa besar *resource* yang digunakan saat algoritma utama dijalankan.

6. **Penyusunan Buku Tugas Akhir**
   Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi perangkat lunak yang sudah dibuat. Sistematika penulisan buku tugas akhir adalah sebagai berikut:

(a) Pendahuluan
    i. Latar Belakang
    ii. Rumusan Masalah
    iii. Batasan Masalah
    iv. Tujuan
    v. Manfaat
    vi. Metodologi
    vii. Sistematika Penulisan
(b) Tinjauan Pustaka
(c) Analisis dan Perancangan
(d) Implementasi
(e) Pengujian dan Evaluasi
(f) Kesimpulan dan Saran
(g) Daftar Pustaka

1.7 Sistematika Penulisan

Penulisan buku tugas akhir ini memiliki tujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir. Secara garis besar, buku tugas akhir ini terdiri dari beberapa bagian seperti berikut:

1. **Bab I Pendahuluan**
   Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan tugas akhir. Selain itu, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

2. **Bab II Tinjauan Pustaka**
   Bab ini berisi tentang penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan tugas akhir ini.

3. **Bab III Desain dan Perancangan**
   Bab ini membahas desain dan perancangan sistem yang akan dibangun untuk menyelesaikan permasalahan Pemanfaatan Algoritma Heuristik untuk Pengaturan Beban pada Komputasi Awan

4. **Bab IV Implementasi**
   Bab ini membahas implementasi dari desain dan perancangan sistem yang telah dibuat pada bab sebelumnya.

5. **Bab V Pengujian dan Evaluasi**
   Bab ini membahas tahap-tahap uji coba yang dilakukan pada sistem yang dibuat. Kemudian kinerja sistem akan dievaluasi berdasarkan hasil uji coba yang didapatkan.

6. **Bab VI Penutup**
   Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan serta saran untuk pengembangan aplikasi di kemudian hari.

BAB II

TINJAUAN PUSTAKA

Bab ini menjelaskan teori-teori yang berkaitan dengan pemanfaatan Algoritma Heuristik untuk pengaturan beban pada komputasi awan yang diajukan untuk tugas akhir ini. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap perangkat lunak yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 Komputasi Awan

Cloud Computing (dalam bahasa Indonesia disebut komputasi awan) adalah proses pengolahan daya komputasi (baik CPU, RAM, Network Speeds, Software, OS maupun Storage) melalui jaringan (biasanya lewat internet). Jadi transfer data yang terjadi bukan secara fisik dan sumber daya komputasi yang dimiliki berada di lokasi pengguna yang memakai layanannya.

NIST dengan tanggung jawabnya untuk membangun standar dan pedoman, menyatakan bahwa karakteristik komputasi awan adalah[5]:

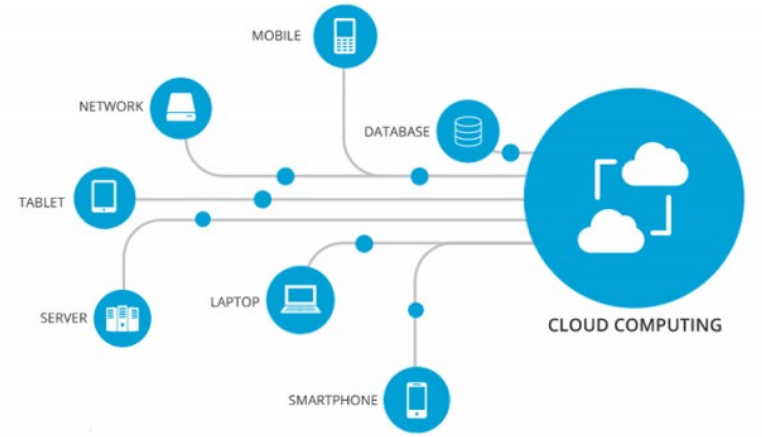
1. ***On-demand self-service***
   Saat ada pengguna yang menginginkan layanan dan sumber daya, maka cloud akan menyediakan layanan atas permintaan. Penyediaan layanan cloud memiliki kemampuan dan kontrol untuk melakukan yang serupa dengan server dan penyimpanan, yang dapat dilakukan oleh konsumen secara sepihak, tanpa interaksi manusia dengan setiap penyedia layanan.
2. ***Network-Access***
   Dalam model ini, set koneksi terhubung ke klien heterogen seperti ponsel dan workstation yang menyediakan kemampuan melalui mekanisme standar.
3. ***Resource pooling***
   Pemasok menggunakan model multi-penyewa yang

menyediakan kumpulan sumber daya, alokasi sumber daya secara fisik dan praktis disediakan untuk konsumen.

4. ***Rapid elasticity***
   Jumlah sumber daya di *cloud* saat ini dapat dengan cepat disediakan atau dirilis sesuai permintaan dengan elastisitas lengkap.
5. ***Measured service***
   Sistem berbasis *cloud* mempelajari kemampuan untuk mengoptimalkan sumber daya dan mengelola penggunaan sumber daya. Ini dapat dipantau, dikendalikan, dan dilaporkan, dengan memberikan tingkat transparansi tertentu.

Cloud Computing merupakan istilah dari *Cloud* diartikan sebagai internet dan *Computing* diartikan sebagai komputer. Definisi dari *Cloud Computing* adalah sebuah proses pengolahan daya komputasi melalui jaringan internet yang memiliki fungsi agar dapat menjalankan program melalui komputer yang telah terkoneksi satu sama lain pada waktu yang sama.

Komputasi Awan merupakan sebuah teknologi yang menjadikan internet sebagai pusat server untuk mengelola data dan juga aplikasi pengguna. *Cloud Computing* memudahkan penggunanya untuk menjalankan program tanpa harus menginstall aplikasi terlebih dahulu dan memudahkan pengguna untuk mengakses data dan informasi melalui internet seperti pada Gambar 2.1.



Gambar 2.1: Komputasi Awan[1]

Teknologi *Cloud Computing* ini menjadikan internet sebagai pusat server dalam mengelelola data. Sistem ini memudahkan pengguna untuk login ke internet agar mendapatkan akses untuk menjalankan program atau aplikasi tanpa harus menginstall aplikasi tersebut.

Karena tidak perlu melakukan installasi pada aplikasi, maka untuk media penyimpanan data dari pengguna juga disimpan secara virtual sehingga tidak akan terbebani dengan penggunaan memori yang ada di komputer. Peritah – perintah yang digunakan oleh pengguna tadi selanjutnya akan dilanjutkan ke server aplikasi.

Setelah perintah diterima oleh sever aplikasi, maka data akan diproses yang akhirnya pengguna akan menerima halaman yang telah diperbaharui sesuai dengan perintah yang telah diberikan sebelumnya. Contoh dari *Cloud Computing* adalah Yahoo, PDF Gmail, Google Drive.

Perintah yang diberikan dalam penggunaan aplikasi tersebut akan langsung terintegrasi secara langsung dengan sistem *Cloud*

Computing yang ada di komputer. Pengguna hanya memerlukan jaringan internet agar dapat menjalankan aplikasi tersebut tanpa perlu melakukan instalasi.

Penggunaan *Cloud Computing* memiliki banyak manfaat, antara lain sebagai berikut:

1. **Media Penyimpanan Terpusat pada Server**
   Teknologi *Cloud Computing* memudahkan pengguna untuk menyimpan data secara terpusat di satu server sesuai layanan yang sudah di sediakan oleh *Cloud Computing*. Selain itu, dari segi infrastruktur pengguna tidak tidak perlu lagi menyediakannya seperti data center, media penyimpanan, sudah tersedia secara virtual oleh *Cloud Computing*.
2. **Keamanan Data**
   Dalam penerapan teknologi *Cloud Computing* penyedia *Cloud Computing* telah menyediakan jaminan data sehingga data tidak mudah *corrupt* atau rusak , platform teknologi, jaminan ISO. Tentunya dengan *Cloud Computing* akan membuat data dan informasi Anda bisa lebih aman terjaga dibandingkan metode konventional yang digunakan oleh kebanyakan orang saat ini.
3. **Lebih Murah dan Tahan Lama**
   Cloud Computing tidak memerlukan media penyimpanan *storage* pada *hard disk* eksternal karena sudah ada media penyimpanan terpusat pada server. Karena semua produk *hardware* atau fisik memiliki masa pemakaian dan setelah masa pemakaian tersebut biasanya akan terjadi beberapa kerusakan dan berfungsi tidak optimal dan sering terjadi *error*.[1]

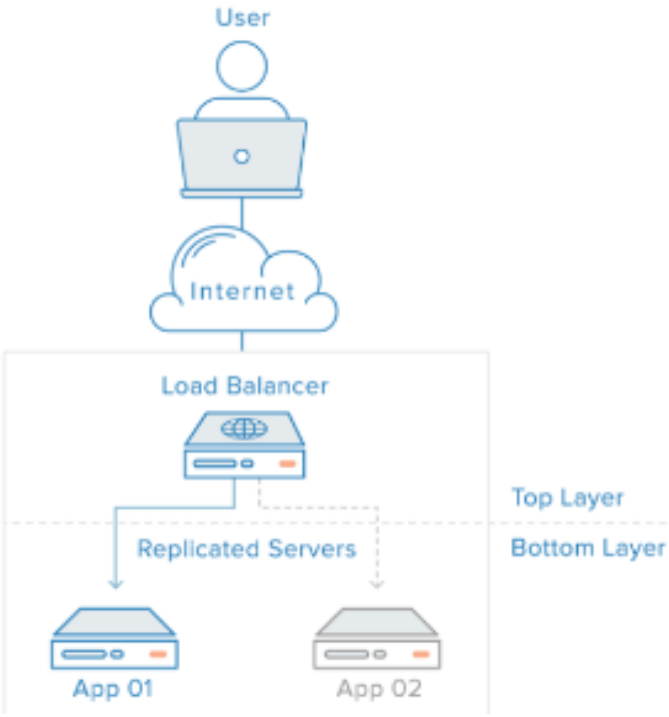
2.2 *Load Balancing*

Load balancing adalah komponen *key* dari infrastruktur yang biasa digunakan untuk meningkatkan kinerja dan keandalan situs web, aplikasi, *database* dan layanan lainnya dengan mendistribusikan beban kerja ke beberapa server. Infrastruktur web tanpa load balancing mungkin terlihat seperti Gambar 2.2.

Gambar 2.2: Tanpa *Load Balancing*[2]

Dalam contoh ini, user terhubung langsung ke server web, di *yourdomain.com*. Jika server web tunggal ini macet, user tidak lagi dapat mengakses situs web. Selain itu, jika banyak pengguna mencoba mengakses server secara bersamaan dan tidak dapat menangani beban, mereka mungkin mengalami waktu yang lambat atau mungkin tidak dapat terhubung sama sekali.

Kegagalan ini bisa dikurangi dengan memasukkan *Load balancer* dan setidaknya satu server web tambahan di *backend*. Biasanya, semua server *backend* akan memasok konten yang identik sehingga pengguna menerima konten yang konsisten tanpa menanggapi server mana yang meresponsnya.



Gambar 2.3: Menggunakan *Load Balancing*[2]

Pada Gambar 2.3, *user* mengakses Load balancer, yang meneruskan permintaan pengguna ke server *backend*, yang kemudian merespons langsung permintaan user[2].

Load balancer membagi *task* sejumlah *resource* yang disediakan. *Load balancer* membagi beban dengan algoritma yang bervariasi. Algoritma pembagian beban yang baik mampu membagi *task* dengan seimbang ke semua *resource* yang tersedia. Sehingga *resource* tidak ada yang *underload* atau *overload*.

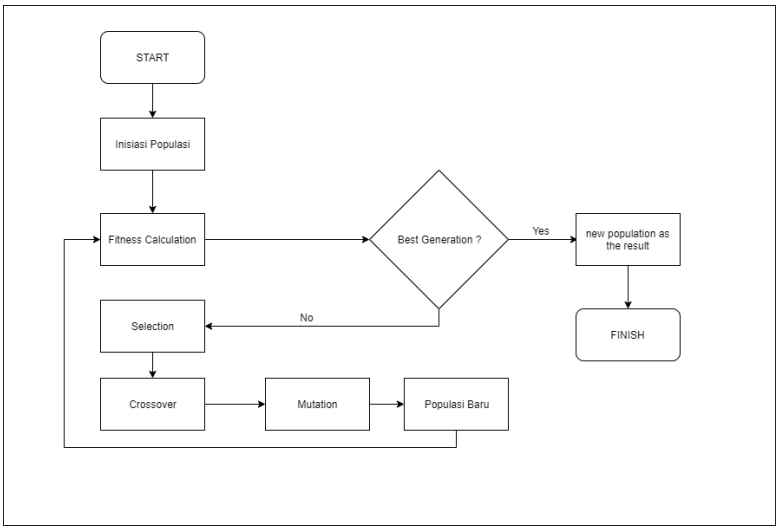
2.3 Python

Python adalah bahasa pemrograman interpretatif, interaktif dan berorientasi objek. Python menggabungkan modul, pengecualian, penulisan secara dinamis, tipe data dinamis yang sangat tinggi dan kelas. Python memiliki antarmuka ke banyak *system call* dan pustaka di berbagai sistem dan dapat diperluas ke bahasa pemrograman C atau C++. Python dapat berjalan pada berbagai sistem operasi seperti Unix, Linux, Mac Os dan Windows.

Python adalah bahasa pemrograman tingkat tinggi yang dapat diterapkan pada berbagai masalah. Bahasa ini dilengkapi pustaka yang besar untuk melakukan pemrosesan *string*, protokol internet, rekayasa perangkat lunak dan antarmuka sistem operasi. [6]

2.4 *Genetic Algorithm*

Genetic Algorithm (GA) adalah teknik pencarian dalam bidang komputasi untuk menemukan solusi benar atau pendekatan untuk masalah optimasi dan pencarian. GA memiliki tiga operasi: *selection* (pemilihan kromosom dua orang tua dari populasi), *crossover* (melakukan *crossover* probabilitas orang tua untuk membangun keturunan baru), *mutation*, diterima (mengambil keturunan baru sebagai hasil terbaik) dan menggesernya[5].

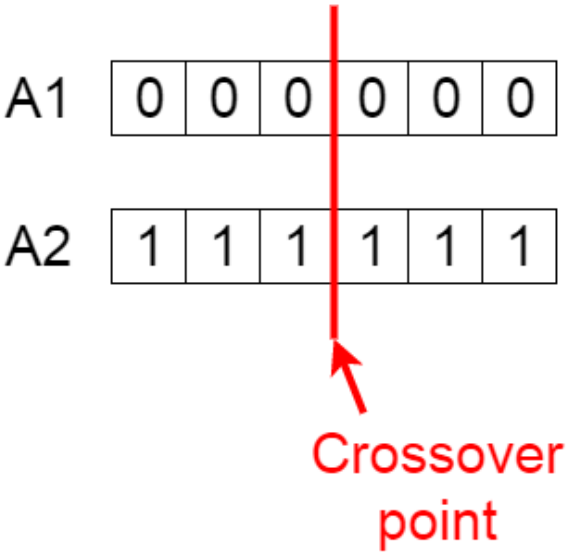


Gambar 2.4: Flowchart *Genetic Algorithm*

Pada Gambar 2.4 merupakan alur normal *Genetic Algorithm*. Dimana *Genetic Algorithm* melalui proses *fitness calculation*, *selection*, *crossover*, dan *mutation*.

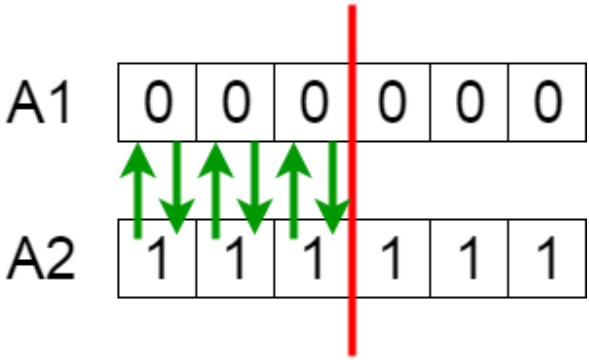
1. ***Fitness Calculation***
   Fitness Calculation menentukan seberapa cocok seorang individu (kemampuan individu untuk bersaing dengan individu lain). Ini memberikan nilai *fitness* untuk setiap individu. Probabilitas bahwa seseorang akan dipilih untuk reproduksi didasarkan pada nilai *fitness*-nya.
2. ***Selection***
   Tahap *selection* adalah memilih individu yang paling cocok dan membiarkan mereka meneruskan gen mereka ke generasi berikutnya. Dua pasang individu (orang tua) dipilih berdasarkan nilai *fitness* mereka. Individu dengan nilai fitness tinggi memiliki lebih banyak kesempatan untuk dipilih untuk lanjut ke tahap berikutnya.

3. ***Crossover***
   Crossover adalah fase paling signifikan dalam algoritma genetika. Untuk setiap pasangan orang tua yang akan dikawinkan, titik crossover dipilih secara acak dari dalam gen. Misalnya, anggap titik crossover adalah 3 seperti yang ditunjukkan pada Gambar 2.5.



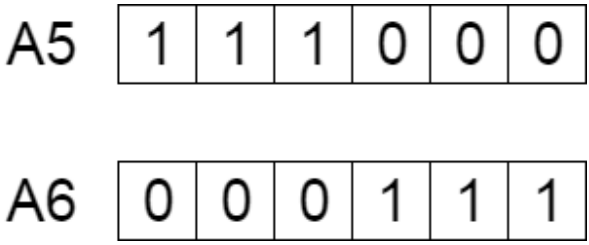
Gambar 2.5: *Crossover Point*[3]

Offspring diciptakan dengan menukar gen orang tua di antara mereka sendiri sampai titik *crossover* tercapai seperti pada Gambar 2.6



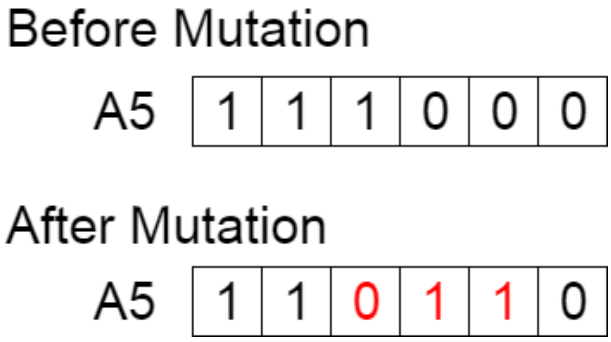
Gambar 2.6: Pengukaran Gen[3]

Offspring baru ditambahkan ke populasi seperti pada Gambar 2.7.



Gambar 2.7: *Offspring* Baru[3]

4. ***Mutation***
   Pada *offspring* baru tertentu yang terbentuk, beberapa gen mereka dapat mengalami mutasi dengan probabilitas acak yang rendah. Ini menyiratkan bahwa beberapa bit dalam string bit dapat dibalik seperti pada Gambar 2.8.



Gambar 2.8: *Mutation*[3]

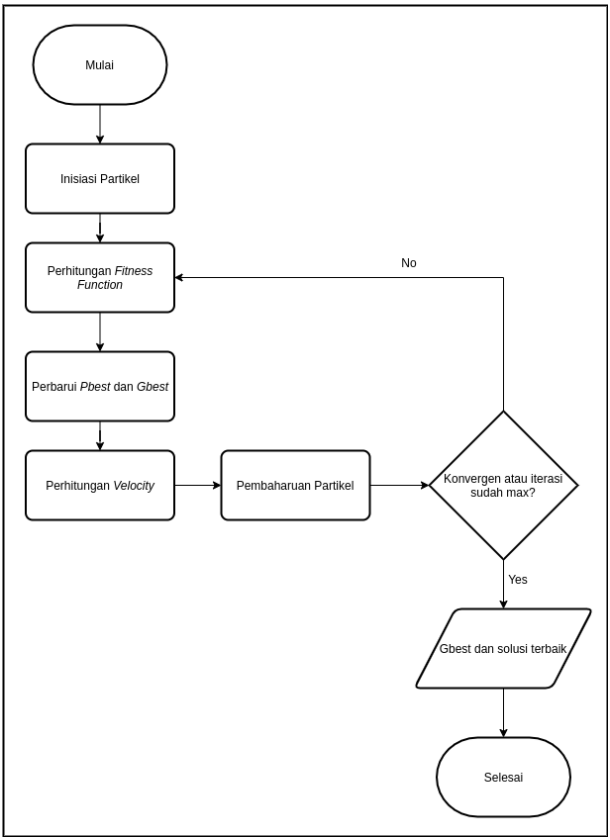
Mutasi terjadi untuk mempertahankan keragaman dalam populasi dan mencegah konvergensi prematur.

5. ***Termination***
   Algoritma berakhir jika populasi telah konvergen (tidak menghasilkan keturunan yang secara signifikan berbeda dari generasi sebelumnya). Kemudian dikatakan bahwa algoritma genetika telah menyediakan serangkaian solusi untuk masalah kita[3].

2.5 *Particle Swarm Optimization*

Particle Swarm Optimization (PSO) adalah algoritma yang mengoptimalkan masalah dengan secara iteratif mencoba meningkatkan solusi kandidat terkait dengan ukuran kualitas yang diberikan. Algoritma ini memecahkan masalah dengan memiliki populasi solusi kandidat, di sini dijuluki partikel, dan memindahkan partikel-partikel ini di dalam ruang pencarian sesuai dengan rumus matematika sederhana atas posisi dan kecepatan partikel. Setiap pergerakan partikel dipengaruhi oleh posisi lokalnya yang terbaik, tetapi juga dipandu ke posisi yang terbaik di ruang pencarian, yang diperbarui menjadi posisi yang

lebih baik, yang ditemukan oleh partikel lain.



Gambar 2.9: Flowchart *Particle Swarm Optimization*

Pada Gambar 2.9 merupakan alur normal *Particle Swarm Optimization*. Dimana partikel-partikel melalui proses *fitness function*, perbarui *Pbest dan Gbest*, perhitungan *velocity* dan pembaharuan partikel.

1. ***Fitness Calculation***

   Pada fase ini dilakukan perhitungan nilai *fitness* tiap

partikel dengan menggunakan *fitness function*. Nilai *fitness* menentukan seberapa baik nilai partikel tersebut.

2. **Perbarui *Pbest* dan *Gbest***

   Nilai *Pbest* merupakan nilai *fitness* terbaik yang dimiliki sebuah partikel. Pada setiap iterasi diperiksa apakah nilai *fitness* pada partikel lebih baik dari pada nilai *Pbest* yang ada. Apabila nilai *Pbest* lebih baik maka *Pbest* tidak diubah. Namun apabila nilai *fitness* lebih baik dari pada *Pbest* maka nilai *fitness* tersebut menjadi *Pbest* yang baru dari partikel tersebut.

   Nilai *Gbest* merupakan nilai *fitness* terbaik yang diambil dari nilai *fitness* pada setiap partikel. Pada setiap iterasi dilihat apakah nilai *fitness* pada setiap partikel ada yang lebih baik dari pada *Gbest*. Apabila nilai *Gbest* lebih baik maka nilai *Gbest* tetap. Apabila nilai *fitness* dari sebuah partikel ada yang lebih baik dari pada nilai *Gbest* maka nilai *Gbest* tersebut akan diperbarui.

   Nilai *Gbest* dan *Pbest* ini nanti yang akan menentukan kemana arah pergerakan partikel. Partikel-partikel akan selalu bergerak menuju ke solusi terbaik.

3. **Perhitungan *Velocity***

   Pada tahap ini dilakukan perhitungan *velocity* yang nantinya akan memperbarui posisi partikel. Perhitungan *velocity* dilakukan dengan rumus perhitungan *velocity*.

   Perhitungan *Velocity*

   $$v_{i,d} = v_{i-1,d} + c_1 * r_1 * (p_b - p_{i,d}) + c_2 * r_2 * (g_b - p_{i,d})$$

   Perhitungan *velocity* dilakukan pada setiap dimensi di dalam partikel. Perubahan posisi partikel dengan *velocity* bertujuan untuk mendapatkan solusi terbaik. Nilai *velocity* berbeda setiap dimensi di partikel. *Velocity* terus berubah setiap iterasi apabila belum mencapai solusi terbaik.

4. **Pembaharuan Partikel**
   Pada tahap ini dilakukan pembaharuan partikel dengan menambahkan posisi partikel dengan *velocity* yang didapat dari tahap perhitungan *velocity*. Dengan pembaharuan partikel maka posisi partikel akan bergerak.
5. ***Termination***
   Algoritma berakhir jika populasi telah konvergen (tidak terjadi perubahan yang signifikan). Kemudian dikatakan bahwa algoritma *Particle Swarm Optimization* telah menyediakan serangkaian solusi untuk masalah kita.

2.6 *Docker*

Docker adalah aplikasi *open source* untuk menyatukan file-file yang dibutuhkan sebuah *software* sehingga menjadi menjadi satu kesatuan yang lengkap dab berfungsi. Data pengaturan dan file pendukung disebut sebagai *image*. Selanjutnya kumpulan *image* digabung dalam satu wadah yang disebut *Container*.

Docker merupakan solusi dari permasalahan yang kerap dialami para *developer* untuk mengembangkan aplikasi mereka agar bisa berjalan fleksibel di berbagai lingkungan.

Kemampuan yang dimiliki *Docker* yaitu mampu menjalankan berbagai macam aplikasi dengan konfigurasi sistem yang berbeda-beda, meskipun masih dalam satu perangkat komputer atau server.

Berikut ini adalah fitur *Docker* yang bisa digunakan sesuai dengan kebutuhan.

1. *Docker Engine*, digunakan untuk membangun Docker images dan membuat kontainer *Docker*.
2. *Docker Hub*, *registry* yang digunakan untuk berbagai macam *Docker images*.
3. *Docker Compose*, digunakan untuk mendefinisikan aplikasi menggunakan banyak kontainer *Docker*.
4. *Docker* untuk *Mac*, memungkinkan menjalankan kontainer

Docker pada *Mac*.

5. *Docker* untuk *Linux*, memungkinkan menjalankan kontainer *Docker* pada *Linux*.
6. *Docker* untuk *Windows*, memungkinkan menjalankan kontainer *Docker* pada *Windows*.

Pengertian umum kontainer (*container*) merupakan alat untuk mempermudah mengemas dan mendistribusikan suatu hal dari satu tempat ke tempat lain.

Sedangkan kita berbicara mengenai teknologi informasi khususnya *Linux*, kontainer diartikan sebagai sistem yang dapat digunakan untuk memberikan sistem yang terisolasi (isolated environment) pada level *OS* yang dijalankan pada satu induk *linux kernel*.

Docker dengan kontainer miliknya memberikan solusi terhadap permasalahan fleksibilitas untuk menjalankan aplikasi pada lingkungan yang berbeda.

Biasanya jika ingin melakukan percobaan sebuah aplikasi membutuhkan lingkungan yang sama persis agar bisa berjalan dengan baik. Jadi jika menggunakan cara yang tidak terintegrasi tentunya akan membutuhkan waktu lama, karena harus melakukan beberapa konfigurasi sehingga server lokal dengan server *live* harus sama konfigurasinya.

Jadi dengan menggunakan Docker, pemindahan dari komputer lokal developer ke server *live* seperti *vps server* untuk menjalankan aplikasi tidak perlu banyak melakukan perubahan. Ada pun keunggulan lain dari penggunaan Docker adalah sebagai berikut:

1. **Konfigurasi Sederhana**
   Kelebihan pertama *Docker* adalah konfigurasi yang diterapkan cukup sederhana dan bisa disesuaikan dengan kebutuhan aplikasi yang sedang dikembangkan. Hanya dengan beberapa baris kode, Docker sudah bisa membuat lingkungan (*environment*) sendiri yang berbeda dengan

lingkungan server utama. Ini akan memisahkan kebutuhkan infrastruktur dari lingkungan aplikasi

2. ***Platform Multi-Cloud***
   Tidak hanya berjalan pada satu *platform cloud* saja, *Docker* bisa dijalankan di berbagai *platform cloud* sehingga membuat Docker cukup fleksibel. Ini adalah kelebihan dan manfaat *Docker* yang bisa menjadi alasan utama *developer* menggunakan *Docker*.
   Penyedia layanan *cloud* besar di dunia beberapa sudah mulai menyediakan dukungan *Docker* pada layanan mereka. Ini tentu saja menanggapi permintaan pasar yang sudah mulai melirik *Docker* untuk menjadi arsitektur utama dalam pengembangan sistem mereka.
   Kontainer *Docker* yang bisa berjalan di berbagai layanan *cloud* memungkinkan sebuah aplikasi bisa di porting antar lingkungan dengan mudah. Sama seperti virtual machine dengan file vdi yang dijalankannya. Hanya saja Docker lebih ringan dan tidak terlalu menggunakan banyak alokasi memori untuk menjalankannya.
3. **Standarisasi Lingkungan dan Kontrol Versi**
   Saat melakukan *upgrade* komponen biasanya seluruh lingkungan akan dipecah. Jika terjadi masalah pada proses tersebut, *Docker* mempunysi fitur agar bisa langsung melakukan *rollback* ke versi sebelumnya melalui *image Docker*. Contoh penggunaan *Docker* ini bekerja lebih cepat dibandingkan dengan *Virtual Machine*. Hal ini dikarenakan Docker melakukan standarisasi lingkungan dengan memastikan konsistensi di beberapa proses pengembangan dan pembaruan versi.
4. **Dapat Melakukan Pengujian dan Distribusi Aplikasi Secara Terus Menerus**
   Fleksibilitas berjalan pada segala macam sistem operasi membuat aplikasi yang menggunakan Docker dapat

dilakukan pengujian dengan mudah dilakukan. Tanpa memempersiapkan konfigurasi yang terkadang membutuhkan waktu yang sangat lama. Dengan begitu, segala proses pengujian kemudian perbaikan dapat dilakukan dengan cepat.

5. **Isolasi**
   Sumber daya pada setiap aplikasi dipastikan terisolasi secara terpisah oleh *Docker*. Jadi *user* dapat menyesuaikan kebutuhan pada setiap aplikasi tanpa harus mempengaruhi konfigurasi pada aplikasi yang lain.
   Sebagai contoh dua aplikasi A dan B membutuhkan *webserver* versi pertama. Setelah proses update ternyata aplikasi A hanya bisa berjalan menggunakan versi *webserver* kedua sehingga harus dilakukan perubahan pada *webserver*. Padahal pada aplikasi B membutuhkan *webserver* versi pertama. Tentu saja ini akan menimbulkan masalah di salah satu aplikasi yang berjalan pada server yang sama.
6. **Keamanan**
   Docker memastikan aplikasi yang berjalan tidak bisa mempengaruhi kontainer atau bahkan memberikan kontrol penuh atas manajemen dan arus lalu lintas. Pengamanan lain yang dilakukan *Docker* adalah dengan mengatur *OS host* mount point dengan *read-only* jadi tidak bisa merubah apapun konfigurasi disana kecuali yang mempunyai akses penuh.[7]

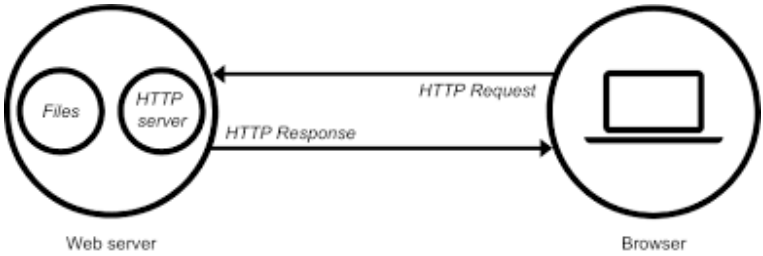
2.7 Web Server

Web server adalah perangkat lunak yang berfungsi sebagai penerima permintaan yang dikirimkan melalui *browser* kemudian memberikan tanggapan permintaan dalam bentuk halaman situs web atau lebih umumnya dalam dokumen *HTML*.

Namun, web server dapat mempunyai dua pengertian berbeda, yaitu sebagai bagian dari perangkat keras (*hardware*) maupun sebagai bagian dari perangkat lunak (*software*).

Jika merujuk pada hardware, web server digunakan untuk menyimpan semua data seperti *HTML* dokumen, gambar, file *CSS stylesheets*, dan file *JavaScript*. Sedangkan pada sisi *software*, fungsi web server adalah sebagai pusat kontrol untuk memproses permintaan yang diterima dari *browser*.

Jadi sebenarnya semua yang berhubungan dengan *website* biasanya juga berhubungan dengan web server, karena tugas web server adalah mengatur semua komunikasi yang terjadi antara browser dengan server untuk memproses sebuah website.



Gambar 2.10: Cara Kerja Web Sever[4]

Cara kerja webserver dapat dilihat pada Gambar 2.10. Seperti penjelasan sebelumnya, saat mengambil halaman website, *browser* mengirimkan permintaan ke server yang kemudian diproses oleh web server. *HTTP request* dikirimkan ke web server. Sebelum memproses *HTTP request*, web server juga melakukan pengecekan terhadap keamanan. Pada web server, *HTTP request* diproses dengan bantuan *HTTP server*. *HTTP* server merupakan perangkat lunak yang bertugas menerjemahkan *URL* (alamat situs web) serta *HTTP* (protokol yang digunakan *browser* untuk menampilkan halaman *website*). Kemudian web server mengirimkan *HTTP response* ke browser

dan memprosesnya menjadi halaman situs web.

Pada saat web server menerima *HTTP request* dari *browser*, jika diperlukan web server akan mengirimkan *query* ke *database* untuk memenuhi permintaan *HTTP request* yang dikirimkan oleh *browser*.

Selain berfungsi sebagai komunikasi penghubung dengan situs web dan memproses *HTTP request* yang dikirimkan oleh *browser*, secara umum beberapa fungsi web server adalah sebagai berikut:

1. Memastikan semua modul yang dibutuhkan tersedia dan siap digunakan.
2. Membersihkan penyimpanan, *cache*, dan *module* yang tidak terpakai.
3. Melakukan pemeriksaan keamanan terhadap *HTTP request* yang dikirimkan *browser*[4].

2.8 Flask

Flask adalah kerangka aplikasi web Python yang ringan. Flask dirancang untuk memulai membuat web dengan cepat dan mudah, dengan kemampuan untuk membuat aplikasi web sampai tingkat yang rumit. Flask dibuat dengan terintegrasi dengan modul *Werkzeug* dan *Jinja*. Flask termasuk salah satu kerangka aplikasi web Python yang populer.

Flask didesain tidak memiliki depedensi dan tata letak kerangka aplikasi, dengan demikian pengembang memiliki kebebasan untuk mengatur kerangka aplikasinya sendiri serta menambahkan modul yang diperlukan sesuai kebutuhan. Flask memiliki berbagai ekstensi yang dikembangkan oleh komunitas sehingga dapat menambahkan berbagai fungsi dengan mudah[8].

(Halaman ini sengaja dikosongkan)

BAB III

ANALISIS DAN PERANCANGAN

Bab ini menjelaskan tentang analisis dan perancangan mengenai sistem yang akan dibuat.

3.1 Analisis Sistem

Analisis sistem terbagi menjadi dua bagian, yaitu analisis permasalahan yang diangkat pada tugas akhir ini dan deskripsi umum sistem yang dibangun.

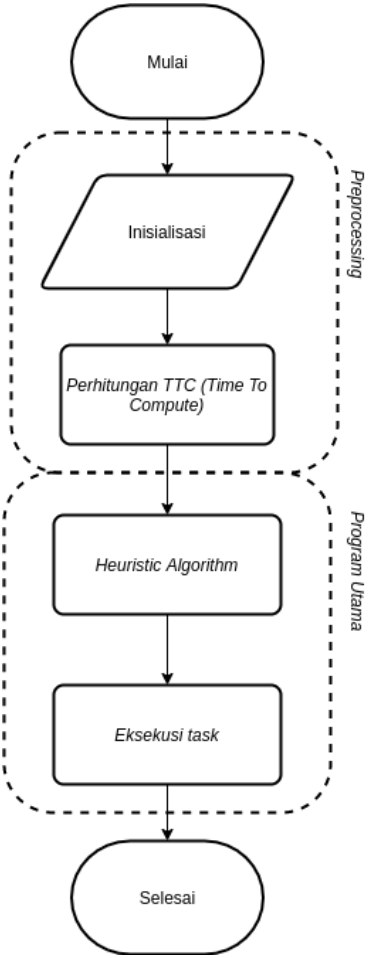
3.1.1 Analisis Permasalahan

Permasalahan yang diangkat pada tugas akhir ini adalah dimana diberikan sejumlah *task* atau *job* dan terdapat sejumlah *virtual machine*. Untuk membagi beban yang *task* agar sesuai diperlukan sebuah algoritma heuristik supaya *task* dapat terbagi dengan seimbang ke *virtual machine* yang telah disediakan sehingga tidak ada *resource* yang *overload* ataupun *underload*.

3.1.2 Deskripsi Umum Sistem

Secara garis besar fitur yang diimplementasikan pada tugas akhir ini terdiri dari dua proses, yaitu *preprocessing* untuk membantu meringankan perhitungan pada proses utama dan proses utama di mana pemetaan pengguna akan dilakukan.

Alur kerja sistem pada penelitian ini dapat dilihat pada diagram alur pada Gambar 3.1.



Gambar 3.1: Diagram Alur Deskripsi Umum Sistem

Pada Gambar 3.1 secara umum memiliki 2 bagian yaitu *preprocessing* dan program utama. Dimana *preprocessing* menghitung *Time to Compute*(TTC) dan program utama merupakan algoritma heuristik. Pada program utama dilakukan

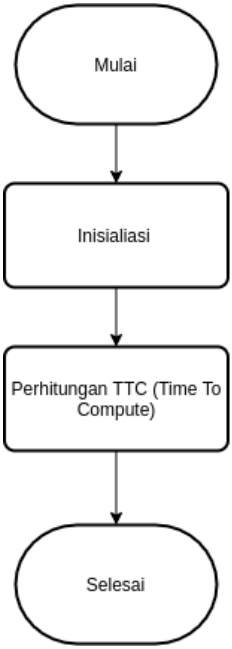
pemrosesan *task* dengan *heuristic algorithm* dimana pada tugas akhir ini algoritma yang digunakan *Genetic Algorithm* dan *Particle Swarm Optimization* dan eksekusi *task* yang mengeksekusi *task* hasil dari *heuristic algorithm* ke *resource* yang tersedia.

3.2 Perancangan Sistem

Perancangan sistem menjelaskan perancangan proses pada sistem, yaitu *preprocessing* dan proses utama.

3.2.1 *Preprocessing*

Pada tahap ini dilakukan perhitungan *Time to Compute*(TTC) yang merupakan perhitungan waktu untuk menyelesaikan sebuah job yang dilakukan oleh sebuah *virtual machine* atau *server*. Tahap *preprocessing* mempunyai alur seperti terlihat pada Gambar 3.2.



Gambar 3.2: Diagram Alur Preprocessing

Pada Gambar 3.2 merupakan alur preprocessing. Pada tahap ini terdapat insialisai awal dan dilakukan perhitungan *Time to Compute(TTC)* yang nantinya akan digunakan pada proses utama.

3.2.1.1 Inisialisasi

Pada tahap ini dilakukan ini inisialisasi jumlah total *job* atau *task*, jumlah *resource* dan bentuk data dengan isi random. Pada *Genetic Algorithm* bentuk data dapat dilihat seperti Tabel 3.1.

j_1	j_3	$*$	j_4	$*$	j_2	j_5

Tabel 3.1: Bentuk Data pada *Genetic Algorithm*

Pada Tabel 3.1 merupakan bentuk data(*chromosome*) yang akan digunakan pada *Genetic Algorithm*. Pada data j_i merupakan indeks task yang diinisiasi. Pada data * mengartikan bahwa *job* setelahnya dieksekusi oleh *resource* selanjutnya.

Untuk algoritma *Particle Swarm Optimization* memiliki bentuk data pada Tabel 3.2.

j_1	j_2	j_3	j_4	j_5	j_6
1	1	2	0	1	1

Tabel 3.2: Bentuk Data pada *Particle Swarm Optimization*

Pada Tabel 3.2 merupakan bentuk data(partikel) yang akan digunakan pada *Particle Swarm Optimization*. Data berupa *array* dengan indeks yang merupakan indeks dari *task* dan *value* dari *array* tersebut merupakan dimana *job* itu di alokasikan. Misal pada j_1 dialokasikan pada r_1.

3.2.1.2 Perhitungan TTC

Pada tahap ini dilakukan perhitungan *Time to Compute*(TTC) dengan rumus *Time to Compute(TTC)*.

$$TTC[i][j] = S_j / CP_i$$

TTC	Time to Compute
S_j	Size of job j
CP_i	Computer Power of Resource i

Sehingga setelah perhitungan diperoleh sebuah matriks seperi pada Tabel 3.3.

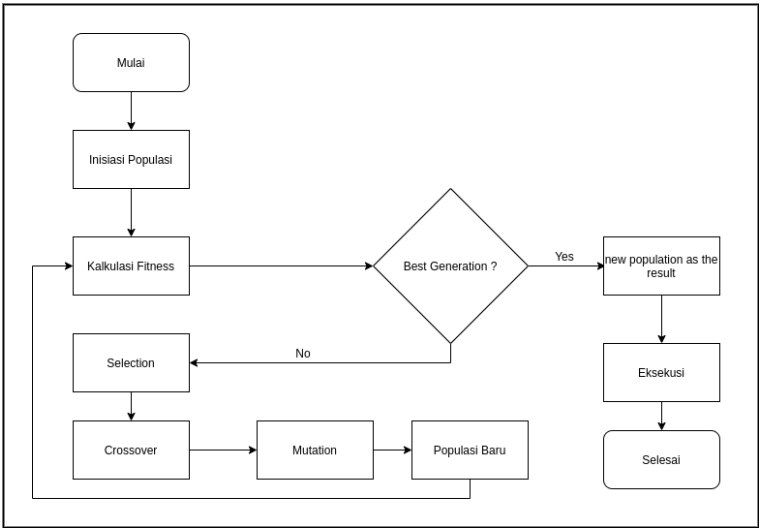
	j_1	j_2	...	j_n
r_1	$ttc_{1,1}$	$ttc_{1,2}$	...	$ttc_{1,n}$
r_2	$ttc_{2,1}$	$ttc_{2,2}$	...	$ttc_{2,n}$
r_3	$ttc_{3,1}$	$ttc_{3,2}$	...	$ttc_{3,n}$
...	...	...	...	...
r_m	$ttc_{m,1}$	$ttc_{m,2}$	...	$ttc_{m,n}$

Tabel 3.3: Matriks TTC

Pada Tabel 3.3 akan terbentuk sebuah matriks yang berisi *time to compute* setiap *job* dengan *resource*. Matriks ini nantinya akan digunakan untuk perhitungan di *fitness function*.

3.2.2 *Genetic Algorithm*

Proses utama pada *Genetic Algorithm* memiliki alur pada Gambar 3.3.



Gambar 3.3: Diagram Alur Proses Utama *Genetic Algorithm*

Pada Gambar 3.3 merupakan diagram *Genetic Algorithm* yang digunakan untuk pembagian beban. Dimana *Genetic Algorithm* melalui proses *fitness calculation*, *selection*, *crossover*, dan *mutation*. Dimana pada *fitness calculation* dilakukan perhitungan *fitness* pada setiap *chromosome* di sebuah populasi. Lalu pada tahap *selection* dipilih *chromosome* terbaik pada populasi. Pada *crossover* dilakukan persilangan pada chromosome terpilih pada tahap *selection*. Pada tahap *mutation* dilakukan mutasi dengan sebuah operasi seperti *move* yang dilakukan pada gen acak pada sebuah *chromosome*. Dan yang terakhir *task* dieksekusi ke *resource* yang tersedia.

3.2.2.1 Inisiasi Populasi

Pada tahap ini dilakukan inisiasi jumlah populasi yang akan diproses dengan *Genetic Algorithm*.

3.2.2.2 Fitness Calculation

Pada tahap ini dilakukan perhitungan dengan menggunakan *Fitness Function* pada rumus dibawah dengan penjelasan pada Tabel 3.4.

$$CT[j_i] = w + c$$

$$AverageCT = totalCT/n$$

$$F = \lambda * maxCT(CT) + (1 - \lambda) * averageCT$$

w	Waiting Time
c	Time to Compute Job
CT	Completion Time
$totalCT$	Total Completion Time
n	Total Job
$maxCT$	Maximum Completion Time
$averageCT$	Average Completion Time
λ	Minimizing factor(0-1)

Tabel 3.4: Penjelasan *Fitness function Genetic Algorithm*

3.2.2.3 *Selection*

Pada tahap ini dilakukan pemilihan *chromosome* yang paling baik. Pemilihan dilakukan dengan melihat hasil dari perhitungan *fitness function*. Semakin kecil hasil dari *fitness function* semakin baik. Hal ini dikarenakan semakin kecil hasil perhitungan *fitness function* maka semakin baik pembagian *task*.

3.2.2.4 *Crossover*

Pada tahap ini dilakukan *crossover* atau penyilangan pada *chromosome* yang terpilih dari tahap *selection*. Penyilangan dilakukan seperti berikut.

Parent

Parent 1	j_1	j_2	$*$	j_3	j_4	$*$	j_5
Parent 2	j_2	$*$	j_1	j_4	$*$	j_5	j_3

Setelah *Crossover*

Child 1	j_1	j_2	$*$	j_1	j_4	$*$	j_5	j_3
Child 2	j_2	$*$	j_3	j_4	$*$	j_5	null	

Selection dilakukan dengan menyilangkan 2 *chromosome* yang berbeda. Dengan penyilangan 2 *chromosome* akan didapatkan 2 *child* yang akan menjadi *chromosome* baru. Penyilangan dilakukan dengan batasan *gen* '*' pada setiap *chromosome* lalu disilangkan setiap *chromosome*.

Setelah dilakukan *crossover*, dikarenakan *chromosome Child 1* dan *Child 2* tidak seimbang dengan terdapat *job* yang sama(diulang) pada *Child 1* dan ada job yang hilang dari *Child 2* maka diperlukan *rebalancing chromosome*.

Setelah *Rebalancing*

Child 1	j_1	j_2	$*$	j_4	$*$	j_5	j_3
Child 2	j_2	$*$	j_3	j_4	$*$	j_5	j_1

3.2.2.5 *Mutation*

Pada tahap ini dilakukan *mutaion* dengan memindahkan *job* ke *resource* lain. Pemilihan job yang akan dipindah dilakukan secara *random* dan dipindahkan ke *resource* lain yang dipilih juga secara *random*. *Mutation* dilakukan seperti berikut.

Sebelum *Mutation*

Child 1	j_1	j_2	$*$	j_4	$*$	j_5	j_3

Setelah *Mutation* j_1 dipindahkan ke *resource* terakhir

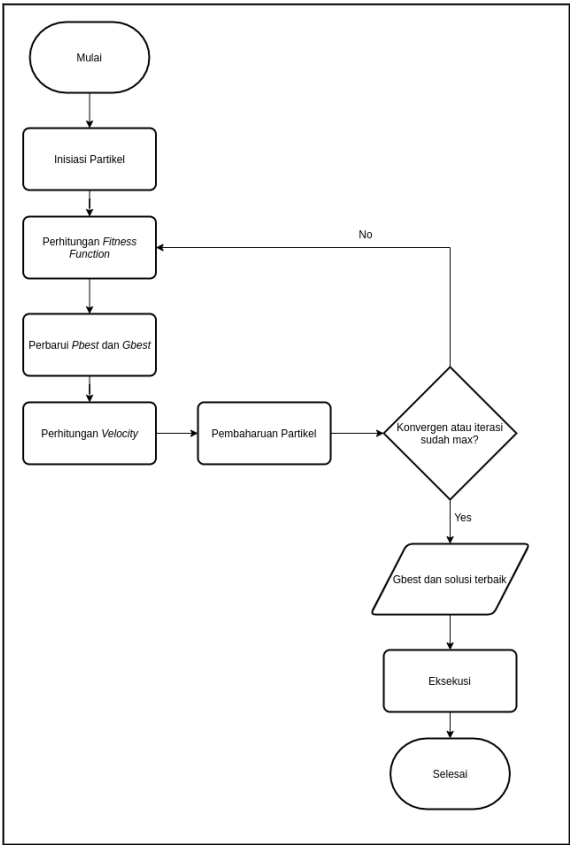
Child 1	j_2	$*$	j_4	$*$	j_5	j_3	j_1

3.2.2.6 Eksekusi

Pada tahap ini dipilihlah *chromosome* terbaik yang memiliki hasil *fitness function*. Lalu *job* dieksekusi ke *resource* yang telah dialokasikan ke job itu.

3.2.3 *Particle Swarm Optimization*

Proses utama pada *Particle Swarm Optimization* memiliki alur pada Gambar 3.4.



Gambar 3.4: Proses Utama *Particel Swarm Optimization*

Pada Gambar 3.4 merupakan alur *Particle Swarm Optimization* untuk pembagian beban. Dimana partikel-partikel melalui proses *fitness function*, perbarui *Pbest dan Gbest*, perhitungan *velocity* dan pembaharuan partikel. Dimana pada *fitness calculation* dilakukan perhitungan *fitness* pada setiap *chromosome* di sebuah populasi. Lalu di perbarui *Pbest* dan *Gbest*. Pada perhitungan *velocity* dilakukan perhitungan *velocity* pada setiap dimensi di partikel. Lalu pada pembaharuan partikel setiap dimensi pada partikel diperbarui dengan *velocity* untuk mendapat posisi partikel baru. Dan terakhir akan dieksekusi pembagian beban ke *resource* yang telah dialokasikan.

3.2.3.1 Inisiasi Partikel

Pada tahap ini diinisiai jumlah partikel yang akan diproses dengan algoritma *Particle Swarm Optimization*.

3.2.3.2 Perhitungan *Fitness Function*

Pada tahap ini dilakukan perhitungan *fitness function* pada setiap partikel dengan rumus yang sama dengan *fitness function genetic algorithm* dengan rumus dibawah dengan penjelasan pada Tabel 3.5.

$$CT[j_i] = w + c$$

$$AverageCT = totalCT/n$$

$$F = \lambda * maxCT(CT) + (1 - \lambda) * averageCT$$

w	Waiting Time
c	Time to Compute Job
CT	Completion Time
$totalCT$	Total Completion Time
n	Total Job
$maxCT$	Maximum Completion Time
$averageCT$	Average Completion Time
λ	Minimizing factor(0-1)

Tabel 3.5: Penjelasan *Fitness function Particle Swarm Optimization*

Pada tahap ini dilakukan perhitungan *fitness function*. Perhitungan dilakukan pada setiap partikel di setiap iterasi. Semakin kecil hasil perhitungan *fitness function* maka semakin baik partikel itu.

3.2.3.3 Perbarui *Pbest* dan *Gbest*

Pbest merupakan nilai terbaik dari sebuah partikel. Setiap partikel memiliki *Pbest*. *Pbest* dari partikel terus diperbarui setiap iterasi. Apabila pada iterasi tersebut sebuah partikel memiliki hasil *fitness function* lebih baik dari sebelumnya, maka *Pbest* kan diperbarui. Namun, apabila hasil *fitness function* dari *Pbest* lebih baik dari pada hasil *fitness function* partikel maka *Pbest* dari partikel tersebut tidak berubah.

Gbest merupakan nilai terbaik dari semua partikel. *Gbest* melihat hasil *fitness function* dari semua partikel. Apabila terdapat sebuah partikel yang memiliki *fitness function* lebih baik maka *Gbest* akan berubah menjadi partikel tersebut. Namun apabila tidak ada hasil *fitness function* yang lebih baik maka nilai *Gbest* tidak berubah.

3.2.3.4 Perhitungan *Velocity*

Pada tahap ini dilakukan perhitungan *velocity* yang nantinya akan memperbarui posisi partikel. Perhitungan *velocity* dilakukan dengan rumus perhitungan *velocity* dengan penjelasan pada Tabel 3.6.

Perhitungan *Velocity*

$$v_{i,d} = v_{i-1,d} + c_1 * r_1 * (pbest - p_{i,d}) + c_2 * r_2 * (gbest - p_{i,d})$$

$v_{i,d}$	*Velocity* yang baru
$v_{i-1,d}$	*Velocity* sebelumnya
c_1	*Learning rates* untuk *pbest*
c_2	*Learning rates* untuk *gbest*
$pbest$	*Particle Best*
$gbest$	*Global Best*
$p_{i,d}$	Nilai partikel *i* pada dimensi *d*
r_1	Nilai *random* 1
r_2	Nilai *random* 2

Tabel 3.6: Penjelasan Rumus *Velocity*

Perhitungan *velocity* dilakukan pada setiap dimensi di dalam partikel. Perubahan posisi partikel dengan *velocity* bertujuan untuk mendapatkan solusi terbaik. Nilai *velocity* berbeda setiap dimensi di partikel. *Velocity* terus berubah setiap iterasi apabila belum mencapai solusi terbaik.

3.2.3.5 Pembaharuan Partikel

Setelah mendapatkan *velocity* tiap dimensi di tiap partikel. Lalu posisi partikel diperbarui dengan menjumlahkan dimensi tiap partikel dengan *velocity* tiap dimensi pada tiap partikel. Sehingga dengan ini posisi tiap partikel akan berubah menuju solusi yang terbaik.

3.2.3.6 Eksekusi

Lalu setelah melalui proses *particle swarm optimization* hingga nilai *gbest konvergen* atau telah mencapai iterasi maksimal maka akan didapatkan partikel terbaik yang membagi beban *job* untuk dieksekusi. Sehingga *job* dapat dieksekusi ke *resource* yang telah dialokasikan ke job itu.

(Halaman ini sengaja dikosongkan)

BAB IV

IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan sebelumnya. Pada bagian implementasi ini juga akan dijelaskan mengenai fungsi-fungsi yang digunakan dalam program tugas akhir ini dan disertai dengan pseudocode masing-masing fungsi utama. Implementasi dilakukan dalam bahasa pemrograman Python.

4.1 Lingkungan Implementasi Perangkat Lunak

Lingkungan implementasi adalah lingkungan di mana fitur temu kembali informasi dibangun. Lingkungan implementasi dibagi menjadi dua yaitu perangkat keras dan perangkat lunak.

4.1.1 Perangkat Keras

Spesifikasi perangkat keras yang digunakan dalam tugas akhir ini dapat dilihat pada Tabel 4.1.

Tabel 4.1: Tabel Perangkat Keras Implementasi Sistem

Spesifikasi	Deskripsi
Tipe	MSI GL62 6QF
Prosesor	Intel® Core™ i7-6700HQ CPU @ 2.60GHz × 8
Memori (RAM)	16 GB

4.1.2 Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan dalam tugas akhir ini dapat dilihat pada Tabel 4.2.

Tabel 4.2: Tabel Perangkat Lunak Implementasi Sistem

Spesifikasi	**Deskripsi**
Sistem Operasi	Ubuntu 18.04.2 LTS
Bahasa Pemrograman	Python 3.6.9
Text Editor	Visual Studio Code 1.43.2
Virtualization	Docker 19.03.8

4.2 Implementasi Program *Preprocessing*

Pada tahap ini akan menjelaskan mengenai implementasi algoritma Perhitungan *TTC* untuk tiap *job* dan *resource*.

4.2.1 Perhitungan *TTC*

Algorithm 1: Algoritma Perhitungan *TTC*

for r *in list_resources* **do**
    $C = getComputerResource(r)$;
    for j *in list_jobs* **do**
        $S = getSizeofJob(j)$;
        $TTC_{r,j} = S/C$;
    end
end

Pseudocode 4.1: Algoritma Perhitungan *TTC*

Dari perhitungan *TTC* akan menghasilkan sebuah matriks *TTC* yang akan dipergunakan pada proses utama. Semakin kecil hasil perhitungan *Time to Compute* maka semakin baik.

4.3 Implementasi Proses Utama

Algoritma pada proses utama diimplementasikan sesuai dengan rancangan sistem yang sudah dibahas pada bab desain dan perancangan. Tujuan dari proses utama adalah menemukan solusi terbaik untuk pembagian beban. Terdapat dua solusi yang ditawarkan untuk menyelesaikan permasalahan, yaitu solusi dengan menggunakan *Genetic Algorithm* dan solusi dengan menggunakan *Particle Swarm Optimization*.

4.3.1 Pembagian Beban Menggunakan *Genetic Algorithm*

Algorithm 2: Algoritma Utama *Genetic Algorithm*

```
InitiatePopulation;
while generation < max_generation do
    calculateFitness(population);
    parents ← selection(parents);
    offsprings ← crossover(parents);
    offsprings ← rebalance(Offsprings);
    offsprings ← mutation(Offspring);
    newPopulation = parents + offsprings;
end
Execute_Best_Chromosome;
```

Pseudocode 4.2: Algoritma Utama *Genetic Algorithm*

Pada proses utama terdapat *calculateFitness* yang dilakukan pada setiap *chromosome* yang nantinya dipergunakan untuk memilih *parent* terbaik. Semakin kecil hasil *calculateFitness* maka semakin baik *chromosome* tersebut.

Algorithm 3: Algoritma *Calculate Fitness*

```
for c in population do
    for r in resources do
        for j in job_list do
            w = waitingTime(j, r);
            c = getTTC(j, r);
            CT_j = w + c;
        end
    end
    maxCT = max(CT);
    totalCT = sum(CT;
    averageCT = totalCT/n;
    Fitness_c = λ * maxCT + (1 − λ) * averageCT;
end
```

Pseudocode 4.3: Algoritma *Calculate Fitness Genetic Algorithm*

Pada proses utama terdapat *selection* dimana pada proses tersebut dilakukan pemilihan *chromosome* terbaik berdasarkan hasil dari *calculateFitness*.

Algorithm 4: Algoritma *Selection*

```
for parent in parent_size do
    parents ← bestChromosome(population, fitness);
    deleteBestChromosome(population, fitness);
end
```

Pseudocode 4.4: Algoritma *Selection*

Setelah dilakukan *selection* dilakukan *crossover* untuk mendapatkan *offspring* dari *parent* parent yang telah didapatkan dari *selection*. *Crossover* dilakukan dengan menyilangkan $parent_i$ dengan $parent_{i+1}$. *Crossover* dilakukan dari indeks 0 dari $parent_i$ dan dari indeks * ke indeks terakhir $parent_{i+1}$.

Algorithm 5: Algoritma *Crossover*

```
for offspring in offspring_size do
    for i = 0 in parent_size do
        crossoverPoint1 = getIndex(parent_i, *);
        crossoverPoint2 = getIndex(parent_{i+1}, *);
        parent1 ← getArray(parent_i, 0, crossoverPoint1);
        parent2 ← getArray(parent_{i+1}, crossoverPoint2, −1);
        offspring ← parent1 + parent2;
    end
end
```

Pseudocode 4.5: Algoritma *Crossover*

Setelah dilakukan *crossover* dilakukan *rebalancing*. *Rebalancing*perlu dilakukan untuk menyeimbangkan *chromosome*. Hal ini dilakukan karena ada kemungkinan *job* yang hilang atau terdapat *job* yang ganda karena melalui tahap *crossover*.

Algorithm 6: Algoritma *Rebalancing*

```
for chromosome in population do
    fixDulpicatedJob(chromosome);
    fixMissingJob(Chromosome);
end
```

Pseudocode 4.6: Algoritma *Rebalancing*

Lalu dilakukan *mutation* setelah tahap *crossover* yang menghasilkan *offspring*. Dimana pada tahap *mutation* dilakukan *move* pada salah satu *gen* yang merupakan *job* ke *resource* lain yang dipilih secara acak.

Algorithm 7: Algoritma *Mutation*

```
for offspring in offspring_size do
    job_idx = randomJobIndex();
    new_resource_idx = randomResIndex();
    offspring ← move(offspring, job_idx, new_resource, idx);
end
```

Pseudocode 4.7: Algoritma *Mutation*

Setelah tahap *mutation* maka terbentuk lah populasi baru yang nanti akan melalui proses *calculateFitness* lagi dan seterusnya. Hingga pada akhirnya hasil *fitness* konvergen atau mencapai batas generasi. Dan pada akhirnya akan mendapatkan solusi terbaik.

Lalu setelah didapatkan solusi terbaik, maka *task-task* akan dieksekusi sesuai *resource* yang dialokasikan pada solusi terbaik hasil dari *Genetic Algorithm*.

Algorithm 8: Algoritma Eksekusi

```
r = 0;
for g in chromosome do
    if g == "*" then
        r = r + 1;
    end
    if g! = "*" then
        execute(job_g, resource_r);
    end
end
```

Pseudocode 4.8: Algoritma Eksekusi *Genetic Algorithm*

4.3.2 Pembagian Beban Menggunakan *Particle Swarm Optimization*

Algorithm 9: Algoritma Utama *Particle Swarm Optimization*

```
InitiateParticles;
while iteration < max_iteration do
    calculateFitness(particles);
    pbest ← getPbest(particles);
    gbest ← getGbest(particles);
    velocity ← calculateVelocity(particles, gbest, pbest);
    particles ← updateParticles(particles, velocity);
end
Execute_Best_Particle;
```

Pseudocode 4.9: Algoritma Utama *Particle Swarm Optimization*

Pada proses utama terdapat *calculateFitness* yang dilakukan pada setiap *particle* yang nantinya dipergunakan untuk memilih

particle terbaik untuk update *pbest* dan *gbest*. Algoritma *calculateFitness* adalah sebagai berikut.

Algorithm 10: Algoritma *Calculate Fitness*

```
for c in particles do
    for r in resources do
        for j in job_list do
            w = waitingTime(j, r);
            c = getTTC(j, r);
            CT_j = w + c;
        end
    end
    maxCT = max(CT);
    totalCT = sum(CT;
    averageCT = totalCT/n;
    Fitness_c = λ * maxCT + (1 − λ) * averageCT;
end
```

Pseudocode 4.10: Algoritma *Calculate Fitness Particle Swarm Optimization*

Setelah dilakukan *calculateFitness* maka akan didapatkan *fitness* pada setiap partikel. Setiap partikel memiliki nilai *Pbest* yang merupakan nilai terbaik setiap partikel. Setiap setelah dilakukan *calculateFitness* dilakukan pembaharuan *Pbest* pada setiap partikel. Apabila hasil *calculateFitness* lebih baik dari pada *Pbest* maka akan *Pbest* akan diperbarui. Apabila nilai *Pbest* lebih baik dari pada hasil *calculateFitness* maka nilai *Pbest* akan tetap.

Algorithm 11: Algoritma *Update Pbest*

```
for i = 0 in particleSize do
    if fitness(particle_i) < Pbest_i then
        Pbest_i ← fitness(particle_i);
    end
end
```

Pseudocode 4.11: Algoritma *Update Pbest Particle Swarm Optimization*

Setiap iterasi memiliki nilai *Gbest*. *Gbest* merupakan nilai *calculateFitness* dari semua partikel. Apabila ada hasil

calculateFitness di sebuah lebih baik dari pada *Gbest* maka nilai *Gbest* diperbarui. Namun apabila nilai *Gbest* lebih baik dari pada nilai *calculateFitness* partikel, maka nilai *Gbest* tetap.

Algorithm 12: Algoritma *Update Gbest*

```
for i = 0 in particleSize do
    if fitness(particle_i) < Gbest then
        Gbest ← fitness(particle_i);
    end
end
```

Pseudocode 4.12: Algoritma *Update Gbest Particle Swarm Optimization*

Pada algoritma *Particle Swarm Optimization* memiliki fungsi *calculateVelocity* yang dipergunakan untuk *updateParticles*.

Algorithm 13: Algoritma *Calculate Velocity*

```
for p = 0 to particles do
    for d = 0 to particles_p do
        r_1 = random(0, 1);
        r_1 = random(0, 1);
        c = 0.2;                                          ▷ value 0-1
        c = 0.2;                                          ▷ value 0-1
        velocity_{p,d} = velocity_{p-1,d} + c_1 * r_1 * (pbest_{p,d} −
          particle_{p,d}) + c_2 * r_2 * (gbest_{p,d} − particle_{p,d}));
    end
end
```

Pseudocode 4.13: Algoritma *Calculate Velocity Particle Swarm Optimization*

Setelah dilakukan *calculateVelocity* maka akan dilakukan *updateParticles*. Tahapan *updateParticles* dilakukan dengan memperbarui setiap dimensi pada setiap partikel.

Algorithm 14: Algoritma *Update Particles*

```
for p = 0 to particles do
    for d = 0 to particles_p do
        particles_{p,d} = particles_{p,d} + velocity_{p,d};
    end
end
```

Pseudocode 4.14: Algoritma *Update Particles Particle Swarm Optimization*

Setelah dilakukan *updateParticles* maka akan dilakukan kembali ke tahapan *calculateFitness*. Hal ini dilakukan terus sampai hasil *calculateFitness* konvergen atau sudah mencapai maksimal iterasi. Apabila sudah konvergen atau sudah mencapai maksimal iterasi maka telah didapatkan solusi terbaik hasil dari *Particle Swarm Optimization*. Solusi terbaik ini nantinya akan berupa pembagian beban ke *resource* yang ada.

Lalu setelah didapatkan solusi terbaik, maka *task-task* akan dieksekusi sesuai *resource* yang dialokasikan pada solusi terbaik hasil dari *Particle Swarm Optimization*.

Algorithm 15: Algoritma Eksekusi

$r = 0$;
for $i = 0$ *in* $particle$ **do**
  $execute(job_i, resource_{particle_i})$;
end

Pseudocode 4.15: Algoritma Eksekusi *Particle Swarm Optimisation*

(Halaman ini sengaja dikosongkan)

BAB V

PENGUJIAN DAN EVALUASI

Bab ini akan membahas tahap pengujian dan evaluasi sistem yang sudah dibangun berdasarkan desain dan perancangan pada bab sebelumnya. Hasil yang didapatkan dari tahap pengujian akan dievaluasi sehingga dapat didapatkan kesimpulan untuk bab selanjutnya.

5.1 Lingkungan Pengujian

Pada proses pengujian perangkat lunak, dibutuhkan suatu lingkungan pengujian yang sesuai dengan standar kebutuhan. Lingkungan pengujian dalam tugas akhir ini adalah sama untuk setiap kasus. Spesifikasi dari lingkungan pengujian dapat dilihat pada Tabel 5.1.

Tabel 5.1: Lingkungan Pengujian

Spesifikasi	**Deskripsi**
Prosesor	Intel® Core™ i7-6700HQ CPU @ 2.60GHz × 8
Arsitektur Prosesor	64 bit
Sistem Operasi	Ubuntu 18.04.2 LTS
Memori (RAM)	16 GB

5.2 Jenis Data Pengujian

Pada pengujian ini dataset yang digunakan adalah data buatan. Jumlah *task*, jumlah *resource Computer Power*, dan *Size of Job* telah ditentukan. Pada *Genetic Algorithm* jumlah generasi, jumlah *chromosome* di populasi, dan jumlah parent sudah ditentukan. Pada *Particle Swarm Optimization* jumlah partikel, dan iterasik maksimal telah ditentukan

5.3 Skenario Pengujian

Skenario uji coba akan dibedakan menjadi dua jenis, yaitu uji coba fungsionalitas dan uji coba performa. Uji coba fungsionalitas bertujuan untuk menguji apakah sistem yang dibuat sudah berhasil menjawab kebutuhan fungsional. Sedangkan uji coba performa digunakan untuk mengetahui perbandingan dua solusi yang ditawarkan untuk menyelesaikan permasalahan pada tugas akhir ini.

5.3.1 Uji Coba Fungsioanlitas

Uji coba fungsionalitas adalah tahap pengujian untuk memeriksa apakah aplikasi dapat digunakan sesuai dengan kebutuhan fungsional. Kebutuhan fungsional dapat dilihat pada Tabel 5.2.

Tabel 5.2: Kebutuhan Fungsional

Kode	Deskripsi Kebutuhan
F01	Membagi beban *task* dengan variasi *Size Job*
F02	Membagi beban *task* dengan variasi *Computer Power*

5.3.2 Uji Coba Performa

Skenario uji coba performa dilakukan dengan cara menguji masing-masing algoritma dan mencatat waktu yang digunakan dan *memory* yang digunakan. Pengujian dilakukan 10 kali tiap skenario dengan nilai *default* pada Tabel 5.3.

Parameter	Nilai Default
Task	240
Maksimal Generasi/Iterasi	150
Maksimal *Chromosome*/Partikel	16

Tabel 5.3: Nilai Default Skenario

5.3.2.1 Parameter Pengujian

Pada uji coba performa akan dilakukan beberapa skenario pengujian dengan parameter yang berbeda-beda seperti yang dapat dilihat pada daftar berikut:

1. Skenario Variasi Jumlah *Task*

Kode	Jumlah *Task*
A01	210
A02	240
A03	270
A04	300

Tabel 5.4: Skenario Variasi Jumlah *Task*

2. Skenario Variasi Jumlah *Chromosome*/Partikel

Kode	*Chromosome*/Partikel
B01	8
B02	16
B03	24
B04	32

Tabel 5.5: Skenario Variasi Jumlah *Chromosome*/Partikel

3. Skenario Variasi Maksimal Generasi/Iterasi

Kode	Maksimal Generasi/Iterasi
C01	50
C02	100
C03	150
C04	200

Tabel 5.6: Skenario Variasi Maksimal Generasi/Iterasi

5.4 Analisis Hasil Uji Coba

Bagian ini akan menjelaskan hasil uji coba dan analisis terhadap hasil tersebut. Terdapat dua hasil uji coba yang akan ditampilkan, yaitu hasil uji coba fungsionalitas dan hasil uji coba performa.

5.4.1 Hasil Uji Coba Fungsionalitas

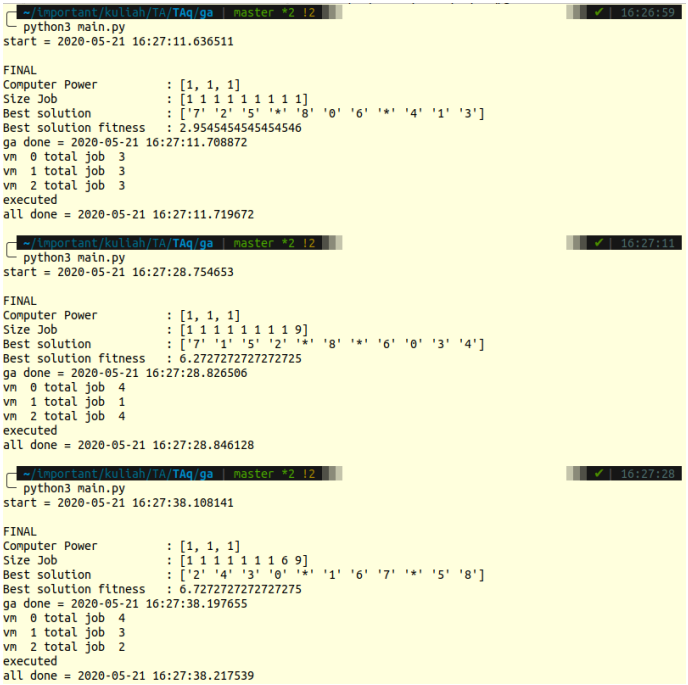
Skenario uji coba fungsionalitas dilakukan berdasarkan skenario pada Tabel 5.2.

1. ***Genetic Algorithm***

Kode	Keterangan
F01	Berhasil
F02	Berhasil

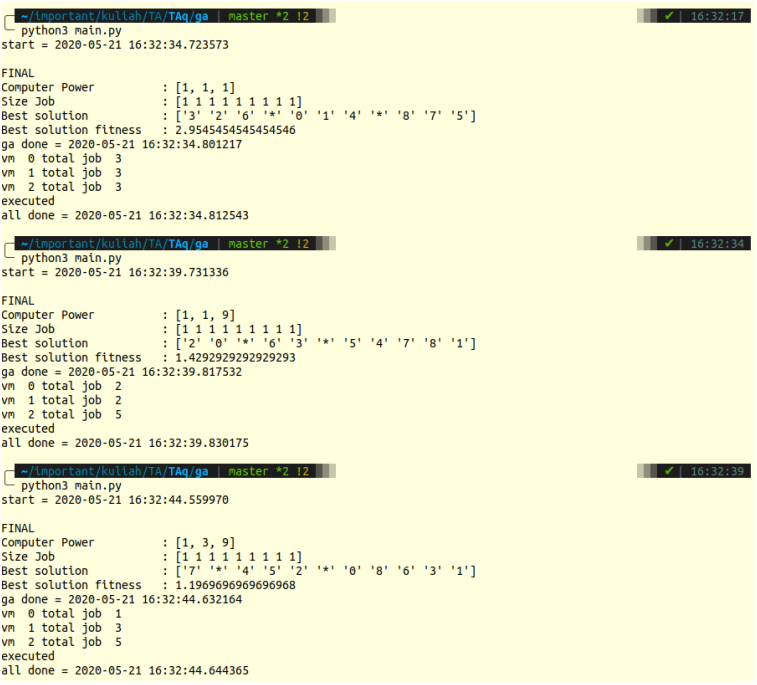
Tabel 5.7: Hasil Uji Coba Kebutuhan Fungsional *Genetic Algorithm*

Pada Tabel 5.7 merupakan hasil uji coba fungsional *Genetic Algorithm*. Dari tabel tersebut dapat dilihat bahwa *Genetic Algorithm* dapat membagi beban *task* ke *resource* yang ada.



Gambar 5.1: Hasil Uji Coba F01 *Genetic Algorithm*

Pada Gambar 5.1 dapat dilihat bahwa *Genetic Algorithm* mampu membagi beban *task* atau *job* dengan *size job* yang bervariasi. Dapat dilihat bahwa *Genetic Algorithm* dapat berhasil membagi beban *task* secara seimbang dengan *size job* yang sama, misalkan terdapat *job* dengan *size job* yang semua sama makan semua *resource* akan mendapat jumlah *job* seimbang. Lalu dengan *size job* yang berbeda, *task* juga dapat dibagi dengan seimbang ke *resouce* yang ada, misalkan pada gambar tersebut terdapat 1 *job* dengan *size job* 9 maka *job* tersebut dialokasikan ke *resource* sendiri tanpa ditambahkan *job* lain.



Gambar 5.2: Hasil Uji Coba F02 *Genetic Algorithm*

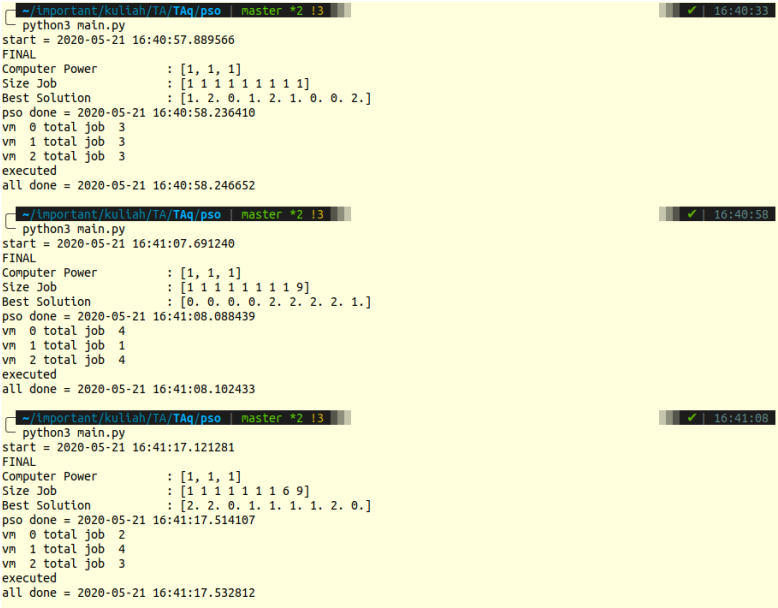
Pada Gambar 5.2 dapat dilihat bahwa *Genetic Algorithm* mampu membagi beban *task* atau *job* dengan *computer power* yang bervariasi. Dapat dilihat bahwa *Genetic Algorithm* dapat berhasil membagi beban *task* secara seimbang dengan *computer power* yang sama, misalkan teradapat 3 *resource* dengan *computer power* yang sama maka *task* akan terbagi seimbang. Lalu dengan *computer power* yang berbeda, *task* juga dapat dibagi dengan seimbang ke *resouce* yang ada, misalkan pada gambar tersebut terdapat *resource* dengan *computer power* lebih besar maka *resource* tersebut akan mendapat beban *task* lebih besar dari pada *resource* yang lain.

2. ***Particle Swarm Optimization***

Kode	Keterangan
F01	Berhasil
F02	Berhasil

Tabel 5.8: Hasil Uji Coba Kebutuhan Fungsional *Particle Swarm Optimization*

Pada Tabel 5.8 merupakan hasil uji coba fungsional *Particle Swarm Optimization*. Dari tabel tersebut dapat dilihat bahwa *Particle Swarm Optimization* dapat membagi beban *task* ke *resource* yang ada.



Gambar 5.3: Hasil Uji Coba F01 *Particle Swarm Optimization*

Pada Gambar 5.3 dapat dilihat bahwa *Particle Swarm Optimization* mampu membagi beban *task* atau *job* dengan *size job* yang bervariasi. Dapat dilihat bahwa *Particle Swarm Optimization* dapat berhasil membagi beban *task* secara seimbang dengan *size job* yang sama, misalkan terdapat *job* dengan *size job* yang semua sama makan semua *resource* akan mendapat jumlah *job* seimbang. Lalu dengan *size job* yang berbeda, *task* juga dapat dibagi dengan seimbang ke *resouce* yang ada, misalkan pada gambar tersebut terdapat 1 *job* dengan *size job* 9 maka *job* tersebut dialokasikan ke *resource* sendiri tanpa ditambahkan *job* lain.

```
~/important/kuliah/TA/TAq/pso | master *2 !3                    ✔ | 16:41:17
  python3 main.py
start = 2020-05-21 16:41:59.782350
FINAL
Computer Power          : [1, 1, 1]
Size Job                : [1 1 1 1 1 1 1 1 1]
Best Solution           : [1. 1. 1. 0. 2. 0. 2. 0. 2.]
pso done = 2020-05-21 16:42:00.146501
vm  0 total job  3
vm  1 total job  3
vm  2 total job  3
executed
all done = 2020-05-21 16:42:00.157589

~/important/kuliah/TA/TAq/pso | master *2 !3                    ✔ | 16:42:00
  python3 main.py
start = 2020-05-21 16:42:05.536241
FINAL
Computer Power          : [1, 1, 9]
Size Job                : [1 1 1 1 1 1 1 1 1]
Best Solution           : [1. 2. 2. 0. 2. 2. 1. 0. 2.]
pso done = 2020-05-21 16:42:05.945907
vm  0 total job  2
vm  1 total job  2
vm  2 total job  5
executed
all done = 2020-05-21 16:42:05.959169

~/important/kuliah/TA/TAq/pso | master *2 !3                    ✔ | 16:42:05
  python3 main.py
start = 2020-05-21 16:42:10.589927
FINAL
Computer Power          : [1, 3, 9]
Size Job                : [1 1 1 1 1 1 1 1 1]
Best Solution           : [1. 0. 1. 2. 2. 1. 2. 2. 2.]
pso done = 2020-05-21 16:42:10.986338
vm  0 total job  1
vm  1 total job  3
vm  2 total job  5
executed
all done = 2020-05-21 16:42:10.998998
```

Gambar 5.4: Hasil Uji Coba F02 *Particle Swarm Optimization*

Pada Gambar 5.4 dapat dilihat bahwa *Particle Swarm Optimization* mampu membagi beban *task* atau *job* dengan *computer power* yang bervariasi. Dapat dilihat bahwa *Particle Swarm Optimization* dapat berhasil membagi beban *task* secara seimbang dengan *computer power* yang sama, misalkan teradapat 3 *resource* dengan *computer power* yang sama maka *task* akan terbagi seimbang. Lalu dengan *computer power* yang berbeda, *task* juga dapat dibagi dengan seimbang ke *resouce* yang ada, misalkan pada gambar tersebut terdapat *resource* dengan *computer power* lebih besar maka *resource* tersebut akan mendapat beban *task* lebih besar dari pada *resource* yang lain.

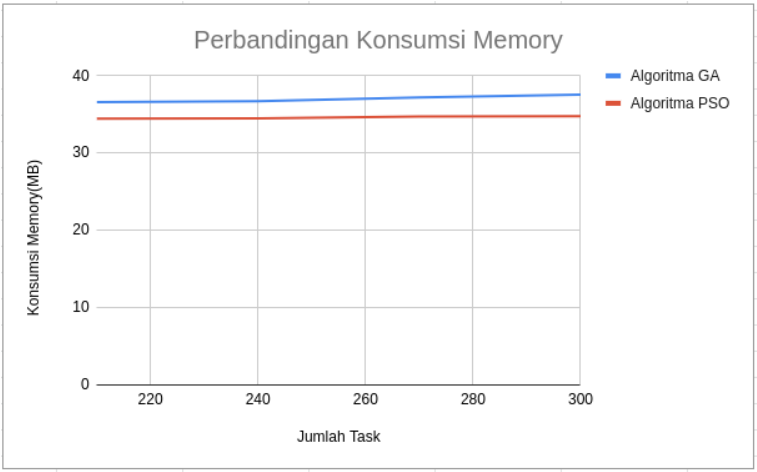
5.4.2 Hasil Uji Coba Performa

Uji Coba dilakukan pada algoritma *Particle Swarm Optimization* dan *Genetic Algorithm* berdasarkan Tabel 5.4, Tabel 5.5 dan Tabel 5.6. Performa yang dilihat adalah waktu yang dibutuhkan untuk menjalankan program dan penggunaan memori.

1. Skenario Variasi Jumlah *Task*

Gambar 5.5: Grafik Perbandingan Waktu Eksekusi dengan Jumlah *Task*

Pada Gambar 5.5 merupakan grafik hasil uji performa *Genetic Algorithm* dan *Particle Swarm Optimization* terhadap waktu eksekusi pada skenario variasi jumlah *task*. Dapat dilihat pada grafik tersebut *Genetic Algorithm* memiliki waktu eksekusi lebih cepat dibandingkan *Particle Swarm Optimization*. Waktu eksekusi pada *Genetic Algorithm* lebih stabil dan lebih cepat karena *Genetic Algorithm* lebih cepat konvergen sehingga cepat menemukan solusi terbaik. Sedangkan pada *Particle Swarm optimization* waktu eksekusi lebih lama dan semakin banyak maka semakin tinggi juga waktu eksekusi.



Gambar 5.6: Grafik Perbandingan Penggunaan *Memory* dengan Jumlah *Task*

Pada Gambar 5.6 merupakan grafik hasil uji performa *Genetic Algorithm* dan *Particle Swarm Optimization* terhadap penggunaan *memory* pada skenario variasi jumlah *task*. Pada gambar grafik tersebut dapat dilihat bahwa *Genetic Algorithm* memiliki penggunaan *memory* lebih tinggi dari pada penggunaan *memory Particle Swarm Optimization*. Dapat dilihat pada grafik algoritma *Genetic Algorithm* dan *Particle Swarm Optimization* memiliki penggunaan *memory* cenderung stabil pada variasi *task*.

Task	GA time(s)	PSO time(s)
210	3.688	11.099
240	3.825	14.27
270	5.165	16.452
300	6.219	19.333

Tabel 5.9: Perbandingan Waktu Eksekusi dengan Jumlah *Task*

Pada Tabel 5.9 terlihat di *Genetic Algorithm* dan *Particle Swarm Optimization* semakin banyak jumlah *task* semakin tinggi juga waktu eksekusi. Dan dilihat bahwa *Genetic Algorithm* memiliki waktu eksekusi lebih sedikit dibanding kan *Particle Swarm Optimization*.

Task	GA Memory(MB)	PSO Memory(MB)
210	36.59	34.451
240	36.699	34.508
270	37.18	34.752
300	37.55	34.795

Tabel 5.10: Perbandingan Penggunaan Memori dengan Jumlah *Task*

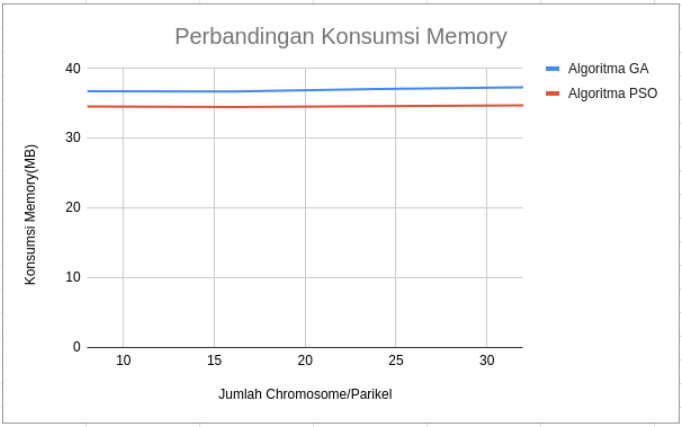
Pada Tabel 5.10 terlihat bahwa rata-rata penggunaan memori pada *Genetic Algorithm* lebih tinggi dari pada penggunaan memori *Particle Swarm Optimization*. Dan dapat dilihat juga bahwa penggunaan memori dipengaruhi oleh jumlah *task*.

Pada skenario ini dapat disimpulkan bahwa jumlah *task* mempengaruhi waktu eksekusi dan penggunaan memori pada kedua algoritma. Hal ini dikarenakan semakin banyak jumlah *task* semakin banyak juga komputasi yang dilakukan.

2. Skenario Variasi Jumlah *Chromosome*/Partikel

Gambar 5.7: Grafik Perbandingan Waktu Eksekusi dengan Jumlah *Chromosome*/Partikel

Pada Gambar 5.7 merupakan grafik hasil uji performa *Genetic Algorithm* dan *Particle Swarm Optimization* terhadap waktu eksekusi pada skenario variasi jumlah *chromosome*/partikel. Dapat dilihat pada grafik tersebut *Genetic Algorithm* memiliki waktu eksekusi lebih cepat dibandingkan *Particle Swarm Optimization*. Waktu eksekusi pada *Genetic Algorithm* lebih stabil dan lebih cepat karena *Genetic Algorithm* lebih cepat konvergen sehingga cepat menemukan solusi terbaik. Sedangkan pada *Particle Swarm optimization* waktu eksekusi lebih lama dan semakin banyak maka semakin tinggi juga waktu eksekusi.



Gambar 5.8: Grafik Perbandingan Penggunaan Memori dengan Jumlah *Chromosome*/Partikel

Pada Gambar 5.8 merupakan grafik hasil uji performa *Genetic Algorithm* dan *Particle Swarm Optimization* terhadap penggunaan *memory* pada skenario variasi jumlah *chromosome*/partikel. Pada gambar grafik tersebut dapat dilihat bahwa *Genetic Algorithm* memiliki penggunaan *memory* lebih tinggi dari pada penggunaan *memory Particle Swarm Optimization*. Dapat dilihat pada grafik algoritma *Genetic Algorithm* dan *Particle Swarm Optimization* memiliki penggunaan *memory* cenderung stabil pada variasi *chromosome*/partikel.

Chromosome/Partikel	GA Time(s)	PSO Time(s)
8	2.806	6.415
16	2.963	13.523
24	5.964	19.277
32	6.472	25.688

Tabel 5.11: Perbandingan Waktu Eksekusi dengan Jumlah *Chromosome*/Partikel

Pada Tabel 5.11 terlihat di *Genetic Algorithm* dan *Particle Swarm Optimization* semakin banyak jumlah *Chromosome*/Partikel semakin tinggi juga waktu eksekusi. Dan dilihat bahwa *Genetic Algorithm* memiliki waktu eksekusi lebih sedikit dibanding kan *Particle Swarm Optimization*.

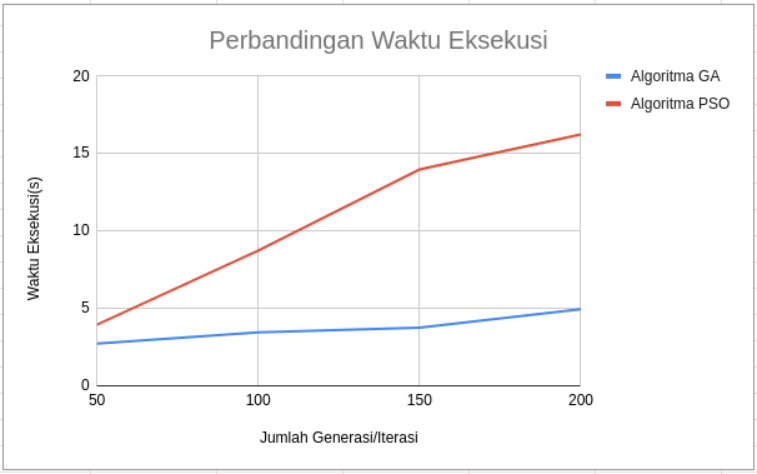
Chromosome/Partikel	GA Memory(MB)	PSO Memory(MB)
8	36.774	34.567
16	36.711	34.501
24	37.081	34.632
32	37.306	34.742

Tabel 5.12: Perbandingan Penggunaan Memori dengan Jumlah *Chromosome*/Partikel

Pada Tabel 5.12 terlihat bahwa rata-rata penggunaan memori pada *Genetic Algorithm* lebih tinggi dari pada penggunaan memori *Particle Swarm Optimization*. Dan dapat dilihat juga bahwa penggunaan memori dipengaruhi oleh jumlah *Chromosome*/Partikel.

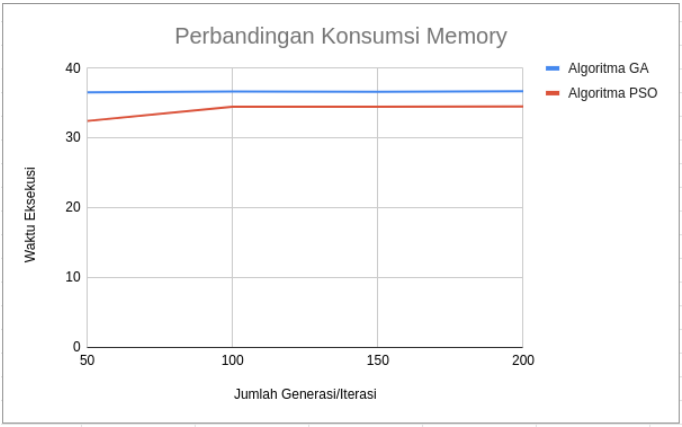
Pada skenario ini dapat disimpulkan bahwa jumlah *Chromosome*/Partikel mempengaruhi waktu eksekusi dan penggunaan memori pada kedua algoritma. Hal ini dikarenakan semakin banyak jumlah *Chromosome*/Partikel semakin banyak juga komputasi yang dilakukan.

3. Skenario Variasi Maksimal Generasi/Iterasi



Gambar 5.9: Perbandingan Waktu Eksekusi dengan Maksimal Generasi/Iterasi

Pada Gambar 5.9 merupakan grafik hasil uji performa *Genetic Algorithm* dan *Particle Swarm Optimization* terhadap waktu eksekusi pada skenario variasi maksimal generasi/iterasi. Dapat dilihat pada grafik tersebut *Genetic Algorithm* memiliki waktu eksekusi lebih cepat dibandingkan *Particle Swarm Optimization*. Waktu eksekusi pada *Genetic Algorithm* lebih stabil dan lebih cepat karena *Genetic Algorithm* lebih cepat konvergen sehingga cepat menemukan solusi terbaik. Sedangkan pada *Particle Swarm optimization* waktu eksekusi lebih lama dan semakin banyak maka semakin tinggi juga waktu eksekusi.



Gambar 5.10: Perbandingan Penggunaan Memori dengan Maksimal Generasi/Iterasi

Pada Gambar 5.10 merupakan grafik hasil uji performa *Genetic Algorithm* dan *Particle Swarm Optimization* terhadap penggunaan *memory* pada skenario variasi maksimal generasi/iterasi. Pada gambar grafik tersebut dapat dilihat bahwa *Genetic Algorithm* memiliki penggunaan *memory* lebih tinggi dari pada penggunaan *memory Particle Swarm Optimization*. Dapat dilihat pada grafik algoritma *Genetic Algorithm* dan *Particle Swarm Optimization* memiliki penggunaan *memory* cenderung stabil pada variasi maksimal generasi/iterasi.

Generasi/Iterasi	GA Time(s)	PSO Time(s)
50	2.727	3.952
100	3.448	8.739
150	3.747	13.975
200	4.938	16.239

Tabel 5.13: Perbandingan Waktu Eksekusi dengan Maksimal Generasi/Iterasi

Pada Tabel 5.13 terlihat di *Genetic Algorithm* dan *Particle Swarm Optimization* semakin banyak maksimal generasi/iterasi semakin tinggi juga waktu eksekusi. Dan dilihat bahwa *Genetic Algorithm* memiliki waktu eksekusi lebih sedikit dibanding kan *Particle Swarm Optimization*.

Generasi/Iterasi	GA Memory(MB)	PSO Memory(MB)
50	36.571	32.452
100	36.659	34.504
150	36.623	34.506
200	36.731	34.513

Tabel 5.14: Perbandingan Penggunaan Memori dengan Maksimal Generasi/Iterasi

Pada Tabel 5.14 terlihat bahwa rata-rata penggunaan memori pada *Genetic Algorithm* lebih tinggi dari pada penggunaan memori *Particle Swarm Optimization*. Dan dapat dilihat juga bahwa penggunaan memori dipengaruhi oleh maksimal generasi/iterasi.

Pada skenario ini dapat disimpulkan bahwa maksimal generasi/iterasi mempengaruhi waktu eksekusi dan penggunaan memori pada kedua algoritma. Hal ini dikarenakan semakin banyak jumlah maksimal generasi/iterasi semakin banyak juga komputasi yang mungkin dilakukan.

BAB VI

KESIMPULAN DAN SARAN

Bab ini akan memaparkan kesimpulan yang bisa diambil selama pengerjaan tugas akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1 Kesimpulan

1. Terdapat dua algoritma yang diajukan untuk menyelesaikan permasalahan, yaitu algoritma dengan menggunakan *Genetic Algorithm* dan algoritma dengan menggunakan *Particle Swarm Optimization*. Proses pembagian beban dengan menggunakan *Genetic Algorithm* dilakukan dengan menginisiasi populasi dimana setiap *chromosome* di dalam populasi mencari solusi pembagian terbaik untuk pembagian beban. Setiap proses seperti *crossover* dan *mutation* hingga pada akhirnya nilai *fitness* konvergen atau generasi sudah mencapai maksimal. Untuk pembagian beban dengan menggunakan algoritma *Partikel Swarm Optimization* dilakukan dengan menginisiai sejumlah partikel dimana setiap partikel akan bergerak untuk mencari solusi terbaik pada pembagian beban. Setiap partikel akan diperbarui setiap iterasi dengan *velocity* yang berbeda-beda tiap dimensinya sehingga partikel akan terus bergerak sampai nilai *fitness* konvergen atau iterasi telah maksimal.
   Dari hasil uji coba, dapat dilihat bahwa *Genetic Algorithm* memiliki waktu eksekusi lebih baik dari pada *Particle Swarm Optimization*. Hal ini dikarenakan *Genetic Algorithm* dapat menghasilkan nilai konvergen lebih cepat dari pada *Particle Swarm Optimization*.

Dari Hasil uji coba, penggunaan memori pada *Particle Swarm Optimization* lebih sedikit dari pada *Genetic Algorithm*. Hal ini dikarenakan pada *Genetic Algorithm* memiliki lebih banyak komputasi dibandingkan *Particle Swarm Optimization*.

2. *Fitness function* yang tepat diperlukan untuk membagi beban *task* yang seimbang. *Fitness function* yang tepat yang akan membawa partikel atau *chromosome* menuju solusi terbaik.

6.2 Saran

1. Menerapkan metode threading agar proses yang dilakukan dapat berjalan secara paralel sehingga dapat meningkatkan performa dalam segi waktu eksekusi program.
2. Menemukan *fitness function* yang lebih baik supaya hasil lebih valid.

DAFTAR PUSTAKA

[1] "Mengenal Apa itu Cloud Computing," 9 Juni 2020. [Daring]. Tersedia pada: https://idcloudhost.com/mengenal-apa-itu-cloud-computing-defenisi-fungsi-dan-cara-kerja/. [Diakses: 9 Juni 2020].

[2] "Arti Load Balancing," 9 Juni 2020. [Daring]. Tersedia pada: https://www.jagoanhosting.com/blog/belajar-banyak-tentang-server-kamu-harus-tahu-arti-load-balancing/. [Diakses: 9 Juni 2020].

[3] "Introduction to Genetic Algorithms," 9 Juni 2020. [Daring]. Tersedia pada: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8b
[Diakses: 9 Juni 2020].

[4] "Apa Itu Web Server," 9 Juni 2020. [Daring]. Tersedia pada: https://www.niagahoster.co.id/blog/web-server-adalah/. [Diakses: 9 Juni 2020].

[5] M. Lagwal dan N. Bhardwaj, "Load balancing in cloud computing using genetic algorithm," in *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 2017, hal. 560–565.

[6] "General Python FAQ," 7 Juni 2020. [Daring]. Tersedia pada: https://docs.python.org/3/faq/general.html#what-is-python. [Diakses: 7 Juni 2020].

[7] "Apa Itu Docker," 9 Juni 2020. [Daring]. Tersedia pada: https://www.niagahoster.co.id/blog/docker-tutorial/. [Diakses: 9 Juni 2020].

[8] "Flask," 7 Juni 2020. [Daring]. Tersedia pada: https://pypi.org/project/Flask/. [Diakses: 7 Juni 2020].

[9] K. Thirumoorthy, B. Selvakumar, dan N. Umakanth, "Genetic algorithm based task deployment approach for load balancing

in cloud environment," *International Journal of Pure and Applied Mathematics*, vol. 118, no. 20, hal. 371–377, 2018.

LAMPIRAN A

KODE SUMBER

```
import numpy
import random
import urllib.request
import threading

def init_pop(pop_size,job_size,total_gen):
    for i in range(pop_size):
        array=numpy.random.choice(range(total_gen),total_gen
    ,replace=False)
        array=numpy.where(array>job_size-1,"*",array)
        if i==0:
            new_population=array
        else:
            new_population=numpy.vstack((new_population,
    array))
    return new_population

def calculate_ttc(vm_size,job_size,cp,sj):
    ttc=numpy.empty((vm_size,job_size))
    for r in range(vm_size):
        for j in range(job_size):
            ttc[r][j]=sj[j]/cp[r]
    return ttc

def cal_pop_fitness(pop,pop_size,total_gen,ttc,vm_size):
    fitness=numpy.zeros(pop_size)
    # loop per chromosome
    for i in range(pop_size):
        job_pair=[[] for z in range(vm_size)]
        CT=[0 for i in range(total_gen)]
        r=0
        # split in pair
        for g in range(total_gen):
            if pop[i][g]=="*":
                r=r+1
                continue
            else:
                job_pair[r].append(pop[i][g])
        # calculate completion time
```

```
38          for r in range(len(job_pair)):
39              if len(job_pair[r]) == 0:
40                  continue
41              else:
42                  for j in range(len(job_pair[r])):
43                      w=0
44                      if j==0:
45                          w=0
46                      if j>0:
47                          w=get_waiting_time(job_pair[r],j,CT)
48                      c=ttc[r][int(job_pair[r][j])]
49                      CT[int(job_pair[r][j])]=w+c
50          # calculate fitness
51          maxCT=max(CT)
52          totalCT=0
53          for item in CT:
54              totalCT=totalCT+item
55          avgCT=totalCT/len(CT)
56          lam=0.5
57          f=lam*maxCT+(1-lam)*avgCT
58          fitness[i]=f
59      return fitness
60
61  def get_waiting_time(job,index,ct):
62      wt=0
63      for i in range(index):
64          wt=wt+ct[int(job[i])]
65      return wt
66
67  def select_mating_pool(pop, fitness_old, num_parents):
68      fitness=numpy.array(fitness_old)
69      parents = numpy.empty((num_parents, pop.shape[1]),dtype=
        object)
70      for parent_num in range(num_parents):
71          min_fitness_idx = numpy.where(fitness == numpy.min(
        fitness))
72          min_fitness_idx = min_fitness_idx[0][0]
73          parents[parent_num] = pop[min_fitness_idx]
74          fitness[min_fitness_idx] = 99999999999
75
76      return parents
```

```
77
78 def crossover(parents, offspring_size):
79     offspring_temp=[[] for i in range(offspring_size[0])]
80     for k in range(offspring_size[0]):
81         parent1_idx = k%parents.shape[0]
82         parent2_idx = (k+1)%parents.shape[0]
83
84         crossover1 = numpy.where(parents[parent1_idx] == "*"
       )
85         crossover_point1=int(crossover1[0][0])
86
87         crossover2 = numpy.where(parents[parent2_idx] == "*"
       )
88         crossover_point2=int(crossover2[0][0])
89         offspring_temp[k][0:crossover_point1] = parents[
       parent1_idx, 0:crossover_point1]
90         offspring_temp[k][crossover_point1:] = parents[
       parent2_idx, crossover_point2:]
91     return offspring_temp
92
93 def rebalance(chr,vm_size,job_size,total_gen):
94     rebalanced=numpy.empty((len(chr),total_gen),dtype=object
       )
95     for i in range(len(chr)):
96         chr_temp=numpy.array(chr[i])
97         for j in range(job_size):
98             check=numpy.where(chr_temp == str(j))
99             if(len(check[0])>1):
100                chr_temp=numpy.delete(chr_temp,int(check
       [0][-1]))
101            elif (len(check[0])==0):
102                chr_temp=numpy.append(chr_temp,str(j))
103        rebalanced[i]=chr_temp
104    return rebalanced
105
106
107 def mutation(offspring_crossover,job_size,vm_size):
108    for idx in range(offspring_crossover.shape[0]):
109        # move mutation
110        gen=random.randrange(0,job_size-1)
111        gen_idx=numpy.where(offspring_crossover[idx]==str(
```

```
        gen))
112         rsc_idx=numpy.where(offspring_crossover[idx]=="*")
113         r_num=0
114         for rsc in range((len(rsc_idx[0])-1),-1,-1):
115             if (int(gen_idx[0][0]) > int(rsc_idx[0][rsc])):
116                 r_num=rsc_idx[0][rsc]
117                 break
118 
119         new_idx=random.randrange(0,job_size+vm_size-1)
120         # * diawal
121         if (r_num==0):
122             new_idx=random.randrange(int(rsc_idx[0][0])+1,(
        job_size+vm_size-1))
123         # * di akhir
124         elif (r_num==int(rsc_idx[0][-1])):
125             new_idx=random.randrange(0,(int(rsc_idx[0][-1]))
        )
126         # * di tengah
127         else:
128             # print("RNUM TENGAH")
129             r_idx=numpy.where(rsc_idx[0]==r_num)
130             while(new_idx > int(r_num) and new_idx <=
        rsc_idx[0][(int(r_idx[0][0])+1)]):
131                 new_idx=random.randrange(0,(job_size+vm_size
        -1))
132         offspring_crossover_temp=offspring_crossover[idx]
133         offspring_crossover_temp=numpy.delete(
        offspring_crossover_temp,gen_idx)
134         offspring_crossover_temp=numpy.insert(
        offspring_crossover_temp,new_idx,str(gen))
135         offspring_crossover[idx]=offspring_crossover_temp
136     return offspring_crossover
137 
138 def requestgan(port):
139     urlnya="http://localhost:"+str(port)
140     try:
141         with urllib.request.urlopen(urlnya) as response:
142             html = response.read()
143     except Exception as e:
144         print(e)
145 
```

```
146 def execute(solution,vm_size,total_gen,size_perjob):
147     job_pair=[[] for i in range(vm_size)]
148     r=0
149     # split in pair
150     for g in range(total_gen):
151         if solution[g]=="*":
152             r=r+1
153             continue
154         else:
155             job_pair[r].append(solution[g])
156     port=5000
157     for i in range(len(job_pair)):
158         print("vm ",i,"total job ",len(job_pair[i]))
159 
160     # execute
161     r=0
162     t=[None for i in range(total_gen*4)]
163     ti=0
164     for i in range(total_gen):
165         port=5000+r
166         if (solution[i]=="*"):
167             r=r+1
168         else:
169             for s in range(int(size_perjob[int(solution[i])
    ])):
170                 t[ti] = threading.Thread(target=requestgan,
    args=(port,))
171                 t[ti].start()
172                 ti=ti+1
173 
174     for k in range(total_gen):
175         if(t[k] is not None):
176             t[k].join()
177     print("executed")
```

Kode Sumber 1.1: Kode Sumber Fungsi GA

```
1 import numpy
2 import ga
3 from datetime import datetime
4 
5 start = datetime.now()
```

```
print("start =", start)
job_size=300 #banyak job
vm_size=3 #>jobsize = vm
total_gen=job_size+(vm_size-1)
pop_size=8
cp=[1,1,1]
size_perjob=numpy.random.randint(low=1,high=2,size=job_size
    +1)
# size_perjob=numpy.array([1,1,1,1,1,1,1,1,1])
new_population = ga.init_pop(pop_size,job_size,total_gen)
ttc = ga.calculate_ttc(vm_size,job_size,cp,size_perjob)
num_generation=50
num_parents_mating = 4

best_count=0
temp_best=0
for generation in range(num_generation):
    fitness = ga.cal_pop_fitness(new_population,pop_size,
    total_gen,ttc,vm_size)

    parents=ga.select_mating_pool(new_population, fitness,
    num_parents_mating)
    offspring_crossover = ga.crossover(parents,
    offspring_size=(pop_size-parents.shape[0], total_gen))

    offspring_crossover=ga.rebalance(offspring_crossover,
    vm_size,job_size,total_gen)

    offspring_mutation=ga.mutation(offspring_crossover,
    job_size,vm_size)

    new_population[0:parents.shape[0], :] = parents
    new_population[parents.shape[0]:, :] =
    offspring_mutation
    best_match_idx = numpy.where(fitness == numpy.min(
    fitness))

    if best_count==int(num_generation/5):
        break
    if (temp_best!=fitness[best_match_idx[0][0]]):
        temp_best=fitness[best_match_idx[0][0]]
```

```
39          best_count=0
40      else:
41          best_count=best_count+1
42
43
44
45  fitness = ga.cal_pop_fitness(new_population,pop_size,
        total_gen,ttc,vm_size)
46  best_match_idx = numpy.where(fitness == numpy.min(fitness))
47  print("\nFINAL")
48  print("Computer Power \t\t:",cp)
49  print("Size Job \t\t:",size_perjob)
50  print("Best solution \t\t:", new_population[best_match_idx
        [0][0], :])
51  print("Best solution fitness \t:", fitness[best_match_idx
        [0][0]])
52  gadone = datetime.now()
53  print("ga done =", gadone)
54  run=ga.execute(new_population[best_match_idx[0][0], :],
        vm_size,total_gen,size_perjob)
55  alldone = datetime.now()
56  print("all done =", alldone)
```

Kode Sumber 1.2: Kode Sumber Program Utama GA

```
1  import numpy
2  import threading
3  import urllib.request
4
5  def init_particle(job_size,vm_size,particle_size):
6      for i in range(particle_size):
7          array=numpy.random.uniform(low=0.0,high=0.999,size=
       job_size)
8          if i==0:
9              new_particle=array
10         else:
11             new_particle=numpy.vstack((new_particle, array))
12     return new_particle
13
14 def calculate_ttc(vm_size,job_size,cp,sj):
15     ttc=numpy.empty((vm_size,job_size))
16     for r in range(vm_size):
```

```
        for j in range(job_size):
            ttc[r][j]=sj[j]/cp[r]
    return ttc

def get_waiting_time(job,index,ct):
    wt=0
    for i in range(index):
        wt=wt+ct[int(job[i])]
    return wt

def cal_fitness(particle,ttc,particle_size,vm_size):
    fitness=numpy.zeros(particle_size)
    for i in range(particle_size):
        norm_particle=numpy.array(particle[i])
        norm_particle=numpy.multiply(norm_particle,vm_size)
        norm_particle=numpy.floor(norm_particle)

        # split per job
        job_pair=[[] for z in range(vm_size)]
        for j in range(len(norm_particle)):
            job_pair[int(norm_particle[j])].append(j)

        # calculate completion time
        CT=[0 for x in range(len(norm_particle))]
        for r in range(len(job_pair)):
            if len(job_pair[r]) == 0:
                continue
            else:
                for j in range(len(job_pair[r])):
                    w=0
                    if j==0:
                        w=0
                    if j>0:
                        w=get_waiting_time(job_pair[r],j,CT)
                    c=ttc[r][int(job_pair[r][j])]
                    CT[int(job_pair[r][j])]=w+c

        # calculate fitness
        maxCT=max(CT)
        totalCT=0
        for item in CT:
```

```
58              totalCT=totalCT+item
59          avgCT=totalCT/len(CT)
60          lam=0.5
61          f=lam*maxCT+(1-lam)*avgCT
62          fitness[i]=f
63      return fitness
64
65  def update_pbest(pbest,particles,old_fitness,fitness,
      particle_size):
66      new_pbest=numpy.array(pbest)
67      for i in range(particle_size):
68          if fitness[i] < old_fitness[i]:
69              new_pbest[i]=particles[i]
70      return new_pbest
71
72  def update_velocity(pbest,gbest,velocity,particles,
      particle_size):
73      new_velocity=numpy.array(velocity)
74      c=[0.2,0.2]
75      for i in range(len(particles)):
76          for d in range(len(particles[i])):
77              r=numpy.random.uniform(low=0.0,high=1.0,size=2)
78              r=numpy.around(r,3)
79              new_velocity[i][d] = velocity[i][d] + c[0]*r[0]
      * (pbest[i][d] - particles[i][d]) + c[1]*r[1] * (gbest[d
      ] - particles[i][d])
80      new_velocity=numpy.around(new_velocity,3)
81      return new_velocity
82
83  def update_particles(particles,velocity):
84      for i in range(len(particles)):
85          for d in range(len(particles[i])):
86              if (particles[i][d]+velocity[i][d]) < 0:
87                  particles[i][d]=0
88              elif (particles[i][d]+velocity[i][d]) >= 1:
89                  particles[i][d]=0.999
90              else:
91                  particles[i][d]=particles[i][d]+velocity[i][
      d]
92      return particles
93
```

```
94
95 def requestgan(port):
96     urlnya="http://localhost:"+str(port)
97     try:
98         with urllib.request.urlopen(urlnya) as response:
99             html = response.read()
100    except Exception as e:
101        print(Exception)
102
103 def execute(gbest,vm_size,job_size,size_perjob):
104    norm_particle=numpy.array(gbest)
105    norm_particle=numpy.multiply(norm_particle,vm_size)
106    norm_particle=numpy.floor(norm_particle)
107
108    # split in pair
109    job_pair=[[] for i in range(vm_size)]
110    for j in range(len(norm_particle)):
111        job_pair[int(norm_particle[j])].append(j)
112
113    for i in range(len(job_pair)):
114        print("vm ",i,"total job ",len(job_pair[i]))
115
116    # execute
117    ti=0
118    t=[None for i in range(job_size*4)]
119    for n in range(len(norm_particle)):
120        port=5000+int(norm_particle[n])
121        for s in range(int(size_perjob[n])):
122            t[ti] = threading.Thread(target=requestgan, args
       =(port,))
123            t[ti].start()
124            ti=ti+1
125    for k in range(job_size):
126        if(t[k] is not None):
127            t[k].join()
128    print("executed")
129    return 1
```

Kode Sumber 1.3: Kode Sumber Fungsi PSO

```
1 import psoku
2 import numpy
```

```
3
4  from datetime import datetime
5
6
7  if __name__ == "__main__":
8      start = datetime.now()
9      print("start =", start)
10     job_size=300
11     vm_size=3
12     particle_size=8
13     velocity=numpy.zeros([particle_size,job_size])
14     cp=[1,1,1]
15     size_perjob=numpy.random.randint(low=1,high=2,size=
       job_size+1)
16     # size_perjob=numpy.array([1,1,1,1,1,1,1,1,1])
17     particles=psoku.init_particle(job_size,vm_size,
       particle_size)
18     particles=numpy.around(particles,3)
19     fitness=numpy.random.randint(low=999999,high=1000000,
       size=particle_size)
20     ttc = psoku.calculate_ttc(vm_size,job_size,cp,
       size_perjob)
21     max_iteration=200
22     gbest_count=0
23     temp_best=0
24     for i in range(max_iteration):
25         old_fitness=numpy.array(fitness)
26         fitness=psoku.cal_fitness(particles,ttc,
       particle_size,vm_size)
27
28         if i==0:
29             pbest=numpy.array(particles)
30         else:
31             pbest=psoku.update_pbest(pbest,particles,
       old_fitness,fitness,particle_size)
32         gbest_idx = numpy.where(fitness == numpy.min(fitness
       ))
33         gbest=numpy.array(particles[int(gbest_idx[0][0])])
34         velocity=psoku.update_velocity(pbest,gbest,velocity,
       particles,particle_size)
35         particles=psoku.update_particles(particles,velocity)
```

```
36
37          if gbest_count==(max_iteration):
38              break
39          if (temp_best!=fitness[gbest_idx[0][0]]):
40              temp_best=fitness[gbest_idx[0][0]]
41              gbest_count=0
42          else:
43              gbest_count=gbest_count+1
44
45          norm_gbest_particle=numpy.array(gbest)
46          norm_gbest_particle=numpy.multiply(
        norm_gbest_particle,vm_size)
47          norm_gbest_particle=numpy.floor(norm_gbest_particle)
48      print("FINAL")
49      print("Computer Power \t\t:",cp)
50      print("Size Job \t\t:",size_perjob)
51      print("Best Solution \t\t:",norm_gbest_particle)
52      psodone = datetime.now()
53      print("pso done =", psodone)
54      execute=psoku.execute(gbest,vm_size,job_size,size_perjob
        )
55      alldone = datetime.now()
56      print("all done =", alldone)
```

Kode Sumber 1.4: Kode Sumber Program Utama PSO

BIODATA PENULIS

Raldo Kusuma, akbrab dipanggil Raldo lahir di Blora pada tanggal 28 Januari 1998. Penulis merupakan anak kedua dari 2 bersaudara. Penulis memiliki hobi antara lain menonton film, bermain basket dan bermain game. Selama berkuliah di Departemen Informatika ITS, penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah Jaringan Komputer (2018),Sistem Operasi (2019-Gasal), dan Sistem Operasi (2019-Genap). Selama menempuh perkuliahan penulis juga aktif di kegiatan organisasi dan kepanitiaan diantaranya menjadi Staf Departemen Dalam Negeri HMTC ITS, Staf Ahli Departemen Dalam Negeri HMTC ITS, Staf REEVA Schematics 2017, dan Staf Ahli PERKAPTRANS Schematics 2018.