BUKU TESIS - KI2501

Sistem Pendistribusian Beban Berbasis Task Profiling pada Klaster Komputer

Thiar Hasbiya Ditanaya

NRP. 5116201048

DOSEN PEMBIMBING

Royyana M Ijtihadie, S.Kom., M.Kom., Ph.D
NIP: 197708242006041001

Bagus Jati Santoso, S.Kom., Ph.D
NIP: 2017198611051

PROGRAM MAGISTER

RUMPUN MATA KULIAH KOMPUTASI BERBASIS JARINGAN

DEPARTEMEN INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

SURABAYA, 2017

[Halaman ini sengaja dikosongkan]

BUKU TESIS - KI2501

Task Profiling - Based Distribution Provisioning System on Computer Cluster

Thiar Hasbiya Ditanaya

NRP. 5116201048

SUPERVISOR

Royyana M Ijtihadie, S.Kom., M.Kom., Ph.D
NIP: 197708242006041001

Bagus Jati Santoso, S.Kom., Ph.D
NIP: 2017198611051

MAGISTER PROGRAMME

NET CENTRIC COMPUTING

INFORMATICS DEPARTMENT

FACULTY OF INFORMATION TECHNOLOGY AND COMMUNICATION

SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY

SURABAYA

2017

[Halaman ini sengaja dikosongkan]

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:
Thiar Hasbiya Ditanaya
Nrp. 5116201048

Dengan judul :
Sistem Pendistribusian Beban Berbasis Request Profiling pada Klaster Komputer / Request Profilng - Based Distribution Provisioning System on Computer Cluster

Tanggal Ujian : 4-1-2018
Periode Wisuda : 2017 Gasal

Disetujui oleh:

Royyana Muslim I, S.Kom, M.Kom, Ph.D
NIP. 197708242006041001
(Pembimbing 1)

Bagus Jati Santoso, S. Kom., Ph. D.
NIP. 2017198611051
(Pembimbing 2)

Prof.Ir.Supeno Djanali, M.Sc, Ph.D
NIP. 194806191973011001
(Penguji 1)

Tohari Ahmad, S.Kom, MIT, Ph.D
NIP. 197505252003121002
(Penguji 2)

Dr.Eng. Radityo Anggoro, S.Kom, M.Sc
NIP. 1984101620081210002
(Penguji 3)

Dekan Fakultas Teknologi Informasi dan Komunikasi,

DR. Agus Zainal Arifin, S.Kom., M.Kom.
NIP. 19720809 199512 1 00 1

[Halaman ini sengaja dikosongkan]

Sistem Pendistribusian Beban Berbasis Task Profiling pada Klaster Komputer

Nama Mahasiswa : Thiar Hasbiya Ditanaya
NRP : 5116 201 048
Pembimbing : Royyana M Ijtihadie, S.Kom., M.Kom., Ph.D
Bagus Jati Santoso, S.Kom., Ph.D

ABSTRAK

Pengguna internet semakin bertambah setiap tahunnya. Dengan semakin banyaknya layanan online berbasis web dapat mempermudah *task* sehari-hari. Beberapa layanan seperti toko online, pemesanan tiket online dan media sosial memiliki pengguna yang banyak dan terus bertambah setiap tahunnya. Bertambahnya pengguna tersebut menuntut pengembangan sistem menjadi lebih handal dalam menangani permintaan setiap pengguna. Penanganan permintaan pengguna yang banyak dalam waktu bersamaan membutuhkan algoritma penyeimbang beban. Algoritma penyeimbang beban akan mendistribusikan *task* ke komputer pekerja(*workers*) yang sesuai.

Untuk meningkatkan mutu layanan waktu respons harus di minimalkan. Pemilihan *worker* yang tepat dapat meminimalkan waktu respons. *Worker* yang kelebihan beban karena mengerjakan *task* yang tidak sesuai, akan memperlambat waktu penyelesaian *task*. Kriteria pemilihan *workers* dapat berdasarkan ketersediaan sumber daya komputer dan waktu terselesaikan *task* (*task completion*).

Oleh karena itu, pada penelitian ini diusulkan peningkatan algoritma penyeimbang beban dengan mempertimbangkan kebutuhan sumber daya *task* sebagai dasar pengelompokan *task*.

Kata kunci*: Penyeimbang beban, distribusi task*

[Halaman ini sengaja dikosongkan]

Task Profiling – Based Distribution Provisioning System on Computer Cluster

Student Name : Thiar Hasbiya Ditanaya

NRP : 5116 201 048

Supervisor : Royyana M Ijtihadie, S.Kom., M.Kom., Ph.D
Bagus Jati Santoso, S.Kom., Ph.D

ABSTRACT

Internet users are growing every year. With the increasing number of web based online services can simplify the daily work. Some services such as online stores, online ticketing and social media have growing users every day. Increasing the user demands system development to be more reliable in handling each user's request. Handling multiple user requests at the same time requires load balancing algorithms. The load balancing algorithm will distribute the work to the appropriate worker's computer

To improve the service quality response time should be minimized. Choosing the right computer worker can minimize response time. Computer workers who are overloaded for doing unsuitable work will slow down the completion time. The criteria for selecting workers can be based on the availability of computer resources and time completion of the job (task completion).

Therefore, in this research, it is proposed to increase load balancing algorithm by considering the need of job resource as the basis of job grouping.

Keywords*: load balancer, task distribution*

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Bismillahirrohmanirohim.

Alhamdulillahirabil'alamin, segala puji bagi Allah Subhanahu Wata'alla, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis bisa menyelesaikan Tugas Akhir yang berjudul "***Sistem Pendistribusian Beban Berbasis Task Profiling pada Klaster Komputer***" dengan baik dan tepat waktu.

Dalam pelaksanaan dan pembuatan Tugas Akhir ini tentunya sangat banyak bantuan-bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Allah Subhanahu Wa Ta'ala atas limpahan rahmat, bimbingan serta hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir ini dengan baik.
2. Kedua orang tua penulis, yang telah memberikan dukungan moral, spiritual dan material, semangat, perhatian, selalu setia dan sabar dalam menghadapi curhatan dari penulis, serta selalu memberikan doa yang tiada habisnya yang dipanjatkan untuk penulis
3. Bapak Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D. dan Bapak Bagus Jati Santoso, S.Kom., Ph.D selaku dosen pembimbing pertama dan kedua, atas bimbingan, arahan, serta bantuan ide untuk menyelesaikan Tesis ini.
4. Bapak Waskitho Wibisono, S.Kom., M.Eng, Ph.D, Dr.Eng. selaku ketua program studi pasca sarjana jurusan Teknik Informatika ITS dan Ibu Dr.Eng. Chastine Fatichah, S.Kom, M.Kom selaku sekretaris prodi S2.
5. Pak Yudi, Pak Sugeng, Mas Jumali dan segenap staf Tata Usaha yang telah memberikan segala bantuan dan kemudahan kepada penulis selama menjalani kuliah di Teknik Informatika ITS.
6. Seluruh teman Teknik Informatika ITS angkatan 2014 dan teman pasca sarjana 2016, selama bersama kalian sadar atau tidak telah membentuk karakter dan kepribadian penulis.
7. Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu yang telah membantu terselesaikannya Tugas Akhir ini.

Kesempurnaan tentu sangat jauh tercapai pada Tesis ini, maka penulis mengharapkan saran dan kritik yang membangun dari pembaca.

Surabaya, Juli 2017

Thiar Hasbiya Ditanaya

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

Pada Bab ini akan dijelaskan mengenai beberapa hal dasar dalam pembuatan proposal penelitian yang meliputi latar belakang, perumusan masalah, tujuan, manfaat, kontribusi penelitian, dan batasan masalah.

1.1 Latar Belakang

Pengguna internet semakin bertambah setiap tahunnya. Dengan semakin banyaknya layanan online berbasis web dapat mempermudah *task* sehari-hari. Beberapa layanan seperti toko online, pemesanan tiket online dan media sosial memiliki pengguna yang banyak dan terus bertambah setiap tahunnya. Bertambahnya pengguna tersebut menuntut pengembangan sistem menjadi lebih handal dalam menangani permintaan setiap pengguna. Penanganan permintaan pengguna yang banyak dalam waktu bersamaan membutuhkan sistem pendistribusian beban. Sistem pendistribusian beban akan berjalan menggunakan algoritma penyeimbang beban.

Algoritma Penyeimbang beban adalah mekanisme yang digunakan untuk mendistribusikan beban *task* yang datang secara bersamaan ke komputer pekerja (*worker)* yang tepat. Algoritma penyeimbang beban *round robin* secara sederhana mendistribusikan *task* bergantian kepada setiap *worker*, pengembangan algoritma *round robin* yaitu *weighted round robin* menambahkan pembobotan dalam menentukan *worker* yang akan diberi beban dan *least connections* akan memberi beban pada *worker* yang memiliki paling sedikit *task* yang sedang ditangani (Kumari & Saxena, 2016).

Seiring meningkatnya pengguna, mutu layanan juga harus ditingkatkan. Peningkatan mutu layanan dapat diukur melalui waktu respons (Al-Moayed & Hollunder, 2010). Semakin cepat permintaan pengguna terlayani semakin baik layanan yang diberikan (Nah, 2003). Kebutuhan tersebut menuntut algoritma penyeimbang beban terus berkembang. Algoritma penyeimbang beban diharapkan mampu menangani lebih banyak pengguna dengan jumlah komputer yang terbatas, sehingga mutu layanan dapat ditingkatkan dengan tetap menekan biaya perawatan.

Demi meningkatkan mutu layanan, dikembangkan banyak algoritma penyeimbang beban, salah satunya *Improved Weighted Round Robin* (*IWRR*) algoritma penyeimbang beban berbasis *round robin* dengan pembobotan, dikembangkan dengan memperhitungkan kemampuan setiap *worker* untuk menangani sejumlah *task* (Devi & Uthariaraj, 2016). Algoritma penyeimbang beban menggunakan pembobotan juga di aplikasikan pada lingkungan *Software-defined Networking* (*SDN*) (Sabiya, 2016). Algoritma penyeimbang beban menggunakan pendekatan heuristis mengalokasikan sumber daya ke *worker* dengan cara memindahkan *task* dari *worker* yang kelebihan beban ke *worker* yang belum kelebihan beban (Zalavadiya & Vaghela, 2016) (Reda, 2015) (Neelakantan, 2012) (Soni, et al., 2015). Kelemahan pendekatan *heuristic*, sumber daya di setiap komputer di pantau setiap waktu sehingga membutuhkan ekstra *bandwith* dan komputasi.

Pendekatan lain dilakukan dengan meminimalkan waktu respons. Waktu respons yang kecil akan meningkatkan mutu layanan. Setiap *task* akan di jalankan pada setiap komputer untuk mendapatkan waktu respons dari masing-masing komputer, pemilihan dilakukan dengan penyesuaian waktu respons terhadap *threshold* yang telah ditentukan (Sharma & Peddoju, 2014) (Jena & Ahmad, 2013).

Dari algoritma penyeimbang beban yang ada pada umumnya masih menggunakan pendekatan dari sisi ketersediaan dan kapasitas sumber daya pada *worker* saja. Padahal beban yang harus dikerjakan oleh *worker* kadang tidak sesuai dengan jenis *task* yang seharusnya dikerjakan oleh *worker*. Setiap *task* memiliki kebutuhan sumber daya yang berbeda (Bautista, et al., 2010). *Task* dapat membutuhkan sumber daya komputasi tinggi, sumber daya memori yang besar atau keduanya. Ada juga *task* yang tidak membutuhkan sumber daya komputasi tinggi maupun sumber daya memori besar.

Proses pemilihan *worker* berdasarkan pengelompokan jenis *task* pernah dilakukan. Pengelompokan dilakukan berdasarkan prioritas waktu terselesaikannya *task* (Sharma & Peddoju, 2014). Pengelompokan dengan mempertimbangkan kebutuhan sumber daya tiap *task* diharapkan mampu memilih *worker* dengan lebih tepat. Untuk mendapatkan pengelompokan *task*, digunakan proses *profiling* tiap *task*. Proses *profiling* menentukan karakteristik *task* menggunakan pengukuran penggunaan memori, komputasi dan penggunaan fungsi (Andelin, et al., 1986)

Oleh karena itu penelitian ini dirancang untuk mengembangkan algoritma penyeimbang beban yang dapat mendistribusikan *task* berdasarkan kebutuhan sumber daya tiap *task*. Pendekatan yang membutuhkan informasi tambahan seperti ketersediaan sumber daya dan kebutuhan sumber daya harus melakukan koleksi data terkait sumber daya pada setiap *worker*. Koleksi data membutuhkan biaya komputasi tambahan. Koleksi data jika dilakukan terlalu sering akan menghabiskan biaya komputasi yang cukup besar. Biaya komputasi tambahan harus dapat di tekan agar tidak memberatkan sistem.

1.2 Perumusan Masalah

Rumusan masalah yang diangkat dalam penelitian ini adalah sebagai berikut.

1. Bagaimana merancang pengelompokan *task* berdasarkan jenis *task?*
2. Bagaimana mendistribusikan *task* berdasarkan rata-rata kebutuhan sumber daya?
3. Bagaimana melakukan pemilihan *worker* yang tepat?
4. Bagaimana mengevaluasi kinerja algoritma penyeimbang beban?

1.3 Tujuan

Tujuan yang akan dicapai dalam pembuatan tesis ini adalah untuk peningkatan algoritma penyeimban beban yang sudah ada dengan cara mendistribusikan *task* berdasarkan pengelompokkan *task* ke *worker* yang tepat, diharapkan metode ini mampu meningkatkan mutu layanan dengan meminimalkan waktu respons.

1.4 Manfaat

Manfaat dari penelitian ini adalah memberikan alternatif algoritma penyeimbang beban yang lebih efektif guna meningkatkan mutu layanan.

1.5 Kontribusi Penelitian

Penelitian ini mengembangkan algoritma penyeimbang beban dengan metode pengelompokan *task* berdasarkan kebutuhan sumber daya agar *task* dapat di distribusikan ke *worker* yang sesuai dengan bebannya. Metode ini diharapkan mengurangi potensi worker yang kelebihan beban dan tidak mampu menyelesaikan *task*nya.

1.6 Batasan Masalah

Batasan masalah pada penelitian ini adalah:

1. Metode dikembangkan menggunakan platform virtualisasi
2. *Task* yang dikerjakan tidak mengalami perubahan kebutuhan sumber daya
3. Lingkungan pengujian adalah sistem operasi linux didalam virtualisasi tanpa pengaturan lebih lanjut pada aspek *scheduling*.

1.7 Hipotesis Awal

Dengan menggunakan algoritma *task profiling* dapat menurunkan waktu terselesaikan (completion time) dan tetap menjaga prosentase task terselesaikan (success rate) tetap maksimal pada lingkungan dengan *task* yang kebutuhan sumber dayanya beragam.

BAB 2
KAJIAN PUSTAKA

Pada bab ini akan dijelaskan tentang pustaka yang terkait dengan landasan penelitian. Pustaka yang terkait adalah seputar Penyeimbang beban, kriteria beban dan kriteria klaster.

2.1 *Task*

Task adalah sekumpulan instruksi yang harus dijalankan oleh komputer. *Task* yang sedang dijalankan oleh sebuah komputer kemudian akan menjadi beban komputasi bagi komputer tersebut. Sekumpulan *task* bisa bersifat homogen tingkat bebannya, bisa juga heterogen. Sangat dimungkinkan sekumpulan *task* memiliki beban komputasi yang berbeda (Bautista, et al., 2010).

Beban komputasi *task* yang berbeda akan mempengaruhi performa dari kluster komputer. Keberagaman beban komputasi *task* akan menyebabkan ketidakseimbangan pada komputer-komputer dalam kluster komputer. Task dengan kebutuhan komputasi tinggi harus dikerjakan dengan komputer yang memiliki sumber daya tinggi juga. Jika *task* dengan kebutuhan komputasi tinggi dikerjakan oleh komputer dengan spesifikasi rendah, maka komputer akan mengalami kegagalan.

2.2 Sumber Daya

Sumber daya adalah kumpulan komponen dalam komputer yang tersedia baik berwujud fisik maupun virtual yang digunakan untuk melakukan *task*. Terdapat banyak jenis sumber daya pada komputer, yang paling utama adalah sumber daya komputasi dan sumber daya memori.

- Sumber daya komputasi
  Sumber daya untuk melakukan komputasi, di tunjukkan sebagai penggunaan CPU dalam *ticks*. Komputasi dapat berjalan semakin cepat seiring bertambahnya inti pada CPU.
- Sumber daya memori
  Sumber daya memori digunakan untuk menyimpan data secara sementara. Beberapa komputasi membutuhkan sumber daya memori untuk menyimpan data seperti pada komputasi pengurutan, pencarian dan pengelompokan data.

2.3 Waktu Respons

Waktu respons adalah merupakan salah satu aspek pengukuran kualitas layanan. Waktu respons yang cukup besar mampu menurunkan kenyamanan pengguna dalam menggunakan layanan. Waktu respons lebih dari 10 detik akan mengurangi kenyamanan pengguna, waktu respons yang dapat diterima pengguna antara 0 – 2 detik (Nah, 2003).

Waktu respons sangat ditentukan oleh kemampuan komputer dalam menyelesaikan *task*. Komputer dengan kemampuan komputasi tinggi mampu menyelesaikan *task* dengan lebih cepat. Kecepatan proses komputer ditentukan dengan banyaknya *ticks* pada *CPU* komputer.

2.4 Profiling

Profiling adalah pengukuran yang digunakan pada komputer atau program komputer untuk keperluan analisis dan forensik. *Profiling* pada perangkat lunak dilakukan dengan mengamati, mengumpulkan dan menganalisa data untuk mengkarakterisasi sebuah program atau *task* (Diep, et al., 2006). Beberapa metode digunakan untuk melakukan *profiling* diantaranya dengan menggunakan standar performa hardware untuk melakukan pengukuran perangkat lunak (Eklov, et al., 2014).

Penerapan profiling dilakukan untuk mendapatkan data kebutuhan sumber daya tiap *task*. Kebutuhan sumber daya tiap *task* kemudian digunakan sebagai parameter karakterisasi *task*. *Task* yang sudah terkarakterisasi kemudian dapat dikelompokkan berdasarkan kesamaan karakternya. Kelompok-kelompok *task* yang dihasilkan nantinya dapat digunakan dalam berbagai macam keperluan komputasi.

2.5 Penyeimbang Beban

Penyeimbang beban adalah sebuah mekanisme untuk menangani beban komputasi yang sangat besar dengan mendistribusikan *task* ke beberapa komputer. Penyeimbang beban terdiri dari beberapa komponen seperti server, *worker* dan peralatan jaringan (Salchow, 2015). Beberapa komputer yang menangani *task* tergabung ke dalam sebuah klaster komputer (Bader & Pennington, 2001). Klaster komputer adalah sekumpulan *worker* yang terkoneksi satu sama lain yang bertugas mengerjakan sekumpulan *task*. Penyeimbang beban harus membagi beban secara tepat dan menghindari adanya *worker* yang kelebihan beban (Beloglazov & Buyya, 2013).

Algoritma Penyeimbang beban adalah kerangka berpikir yang dikembangkan untuk membantu permasalahan dalam pemilihan komputer yang akan melakukan *task* pada sistem penyeimbang beban (Salchow, 2015). Algoritma penyeimbang beban paling sederhana adalah ***roundrobbin***, yang bekerja dengan cara membagi beban satu persatu secara bergantian kepada setiap komputer pada klaster komputer (Kumari & Saxena, 2016).

2.5.1 A Round-Robin based Load balancing (Kumari & Saxena, 2016)

Menggunakan pendekatan *round robin* dengan membagi *task* secara merata. Beban *task* didistribusikan secara rata ke setiap *worker* tanpa memperhitungkan sumber daya yang tersedia pada setiap komputer. Distribusi beban akan dilakukan dengan cepat, karena pemilihan komputer dilakukan dengan cepat secara bergantian. *Round robin* sangat cocok digunakan pada sistem dengan *worker* yang identik dan memiliki konfigurasi yang serupa.

2.5.2 Min Min Algorithm (Kaur & Luthra, 2014)

Algoritma penjadwalan min min bertujuan untuk menyelesaikan *task* tersingkat terlebih dahulu. Algoritma min-min pertama kali akan menghitung waktu eksekusi dan penyelesaian setiap task. Kemudian akan mengurutkan setiap *task* berdasarkan waktu selesai tersingkat. *Task* akan dikerjakan berdasarkan waktu tersingkat. Setiap worker hanya akan mengerjakan satu *task* dalam satu waktu.

2.5.3 Max Min Algorithm (Kaur & Luthra, 2014)

Algoritma penjadwalan max min bertujuan untuk menyelesaikan *task* terlama terlebih dahulu. Algoritma max-min pertama kali akan menghitung waktu eksekusi dan penyelesaian setiap task. Kemudian akan mengurutkan setiap *task* berdasarkan waktu selesai terlama. *Task* akan dikerjakan berdasarkan waktu terlama. Setiap worker hanya akan mengerjakan satu *task* dalam satu waktu.

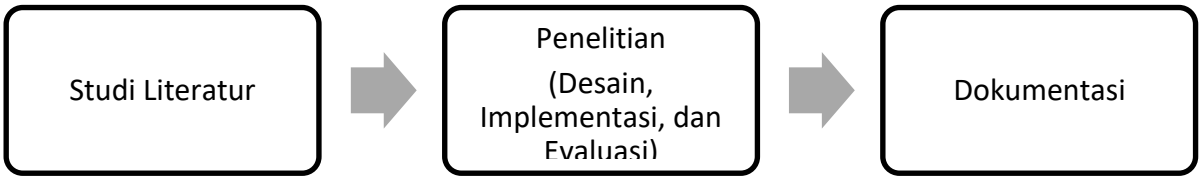
Tabel 2.1 Perbandingan Metriks Setiap Metode

Metode	Parameter pendekatan	
	Worker	*Task*
A Round-Robin based Load balancing (Kumari & Saxena, 2016)	Tidak ada	Tidak ada
Min Min Algorithm (Kaur & Luthra, 2014)	Waktu penyelesaian	Waktu eksekusi
Max Min Algorithm (Kaur & Luthra, 2014)	Waktu penyelesaian	Waktu eksekusi

[Halaman ini sengaja dikosongkan]

BAB 3
METODOLOGI PENELITIAN

Bab ini akan memaparkan tentang metodologi penelitian yang digunakan pada penelitian ini, yang terdiri dari (1) studi literatur, (2) penelitian, dan (3) dokumentasi. Ilustrasi alur metodologi penelitian dapat dilihat pada Gambar 3.1.



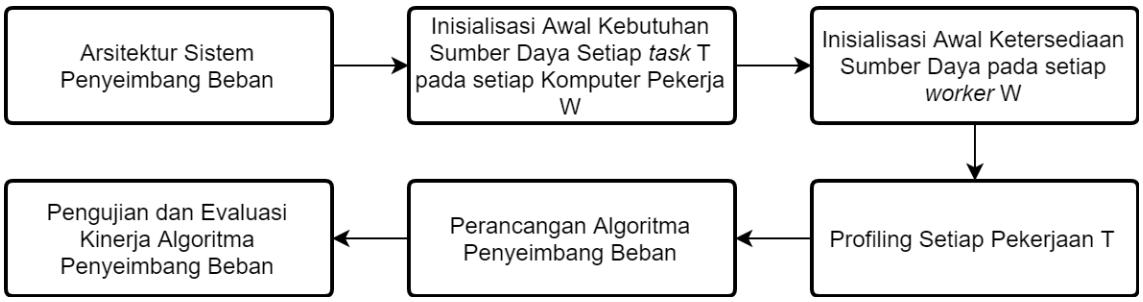
Gambar 3.1 Alur Metodologi Penelitian

Penjelasan tahapan metode penelitian pada Gambar 3.1 akan diterangkan secara terperinci pada subbab berikut.

3.1 Studi Literatur

Penelitian diawali dengan melakukan kajian yang berkaitan dengan topik penelitian. Referensi yang digunakan dalam penelitian ini berasal dari jurnal, konferensi, dan buku yang berkaitan dengan algoritma penjadwalan dan Penyeimbang beban. Berdasarkan studi literatur yang telah dilakukan, terdapat informasi sebagai berikut.

1. Waktu respons dari layanan sangat penting untuk menjaga kepuasan pengguna.
2. *Worker* yang kelebihan beban atau kehabisan sumber daya akan mengalami kegagalan dalam menyelesaikan *task* atau keterlambatan dalam mengembalikan respons.
3. *Task* harus didistribusikan kepada komputer yang tepat, untuk mendapatkan waktu respons terbaik.
4. Belum adanya algoritma penyeimbang beban yang mengelompokkan *task* berdasarkan kebutuhan sumber daya.



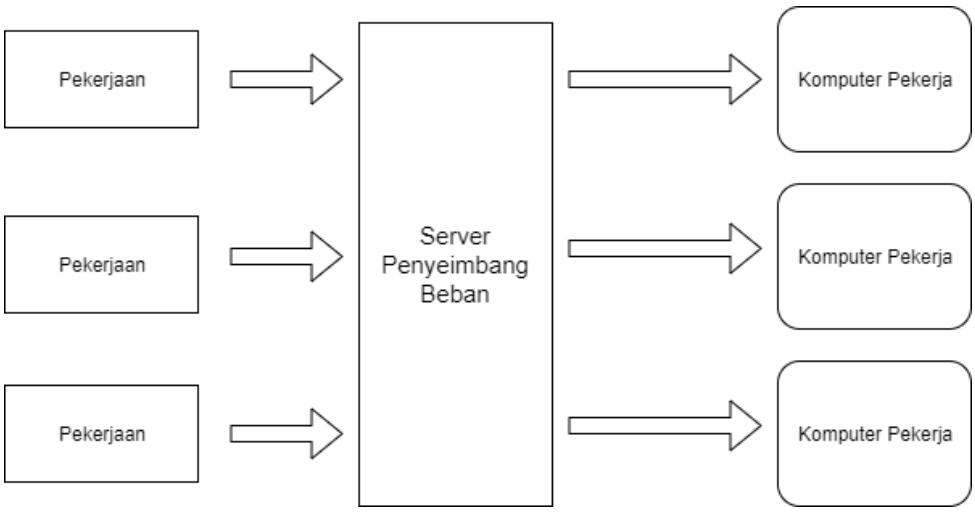
Gambar 3.2 Alur Pembuatan Algoritma Penyeimbang Beban

3.2 Penelitian

Alur sistem dalam pembuatan algoritma Penyeimbang beban dapat dilihat pada Gambar 3.2. Alur pembuatan kerangka kerja ini dibagi menjadi enam tahap, yaitu (1) Desain arsitektur sistem penyeimbang beban, (2) Inisialisasi kebutuhan sumber daya *task*, (3) Inisialisasi ketersediaan sumber daya *worker*, (4) Profiling *task*, (5) Perancangan algoritma penyeimbang beban (6) Pengujian dan Evaluasi. Penjelasan tentang kegiatan dalam bab Penelitian akan dibahas pada subbab berikut.

3.2.1 Desain Arsitektur Sistem Penyeimbang Beban

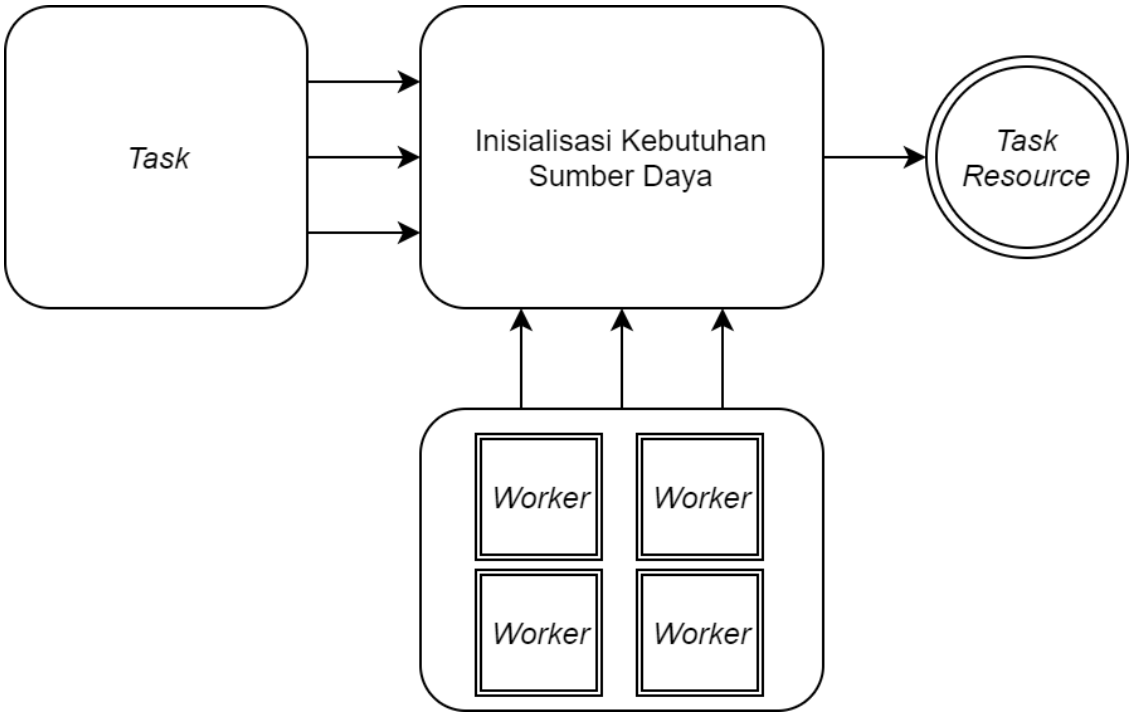
Arsitektur sistem penyeimbang beban terdiri dari server penyeimbang beban dan *worker*. Server penyeimbang beban akan menerima *task* dari pengguna kemudian akan mendistribusikan *task* kepada *worker* yang sesuai. Desain arsitektur sitem penyeimbang beban dapat dilihat pada gambar 3.3.



Gambar 3.3 Desain Sistem Penyeimbang Beban

3.2.2 Inisialisasi kebutuhan sumber daya tiap *task*

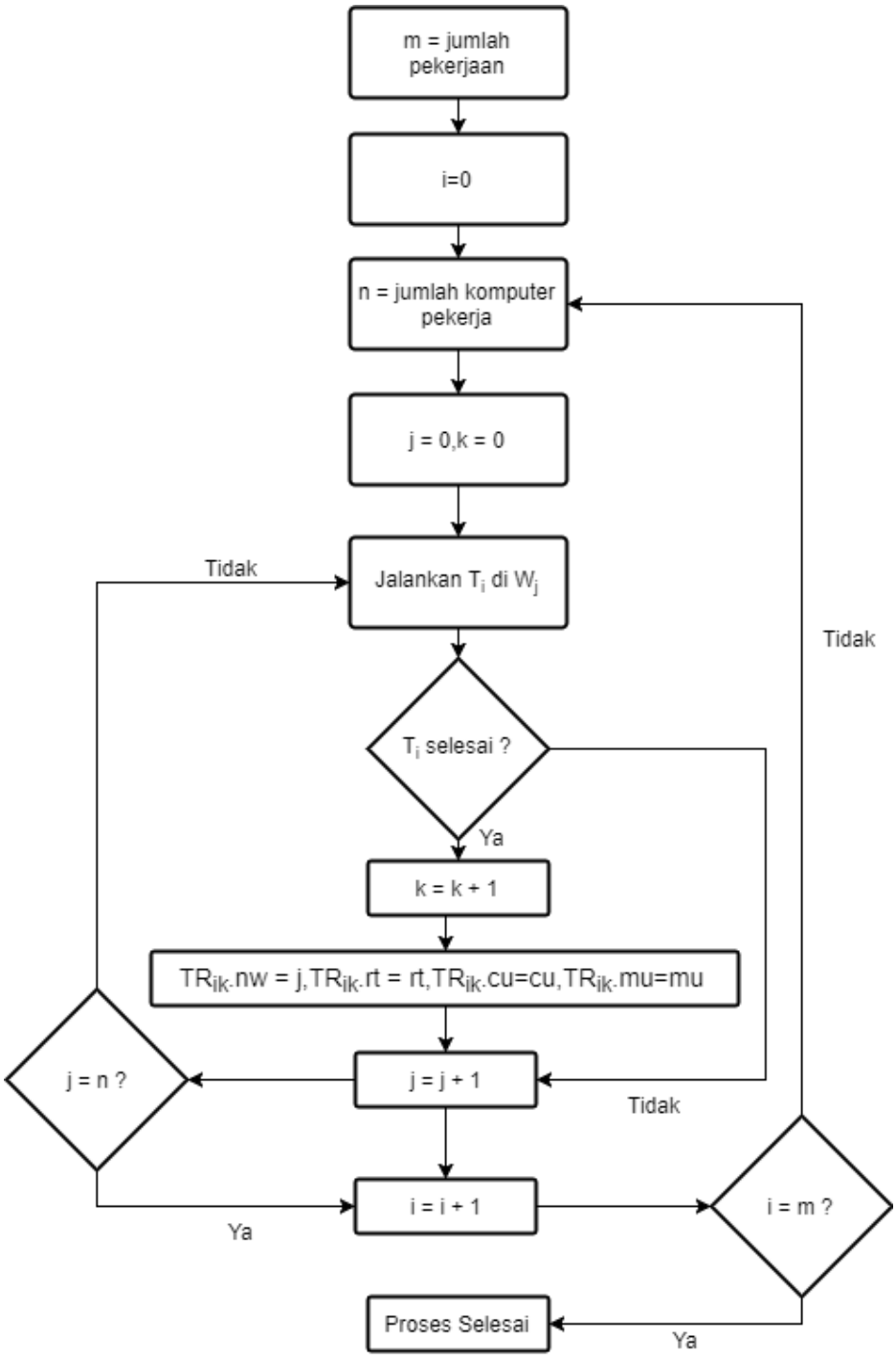
Proses inisialisasi bobot *task* dilakukan untuk menentukan bobot setiap *task* pada setiap komputer. Proses ini juga akan menentukan waktu respons *task* pada setiap komputer. Inisialisasi dilakukan dengan menjalankan setiap *task* pada setiap *worker*, untuk mendapatkan kebutuhan sumber daya dan waktu respons di setiap *worker*. Variabel yang diperoleh pada proses ini nantinya akan digunakan pada proses *profiling task* pada tahap (4) dan penyeimbang beban pada tahap (5). Proses inisialisasi kebutuhan sumber daya di tunjukan pada Gambar 3.4



Gambar 3.4 Proses Inisialisasi Kebutuhan Sumber Daya

Inisialisasi kebutuhan sumber daya dilakukan dengan menjalankan setiap *task* pada setiap *worker* kemudian melakukan pengukuran kebutuhan memori, kebutuhan komputasi cpu dan waktu selesainya *task* pada setiap *worker*. Proses ini akan menghasilkan *Task Resource* (*TR)* yang menyimpan informasi kebutuhan sumber daya memori *(mu)* dan komputasi *(cu)* tiap *task* pada setiap *worker* dan tipe *task* yang akan di tentukan pada proses *profiling task* pada tahap (2). Task Resource juga menyimpan waktu respon (rt) dan nomor worker (nw) yang digunakan sebagai parameter pengurutan

sebelum dilakukan proses *profiling task* pada tahap (4). Diagram alur dijelaskan pada Gambar 3.5.



Gambar 3.5 Diagram Alur Inisialisasi Bobot *Task*

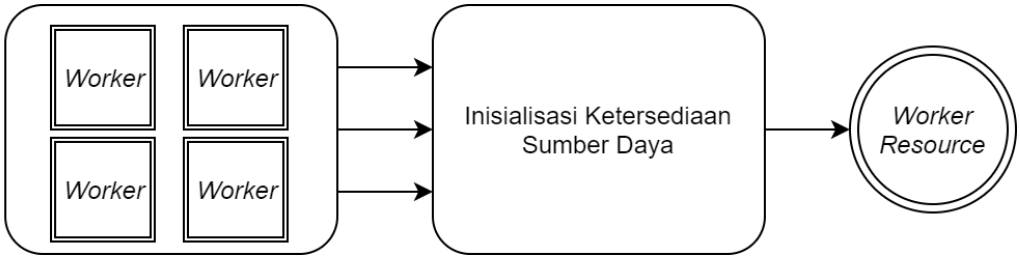
Task $T_{1...m}$ adalah pilihan *task* yang dapat di jalankan pada sistem penyeimbang beban. Untuk setiap *task* $T_{1...m}$ akan dijalankan pada *worker* $W_{1...n}$,Kemudian akan didapat kan kebutuhan sumber daya *task TR*. Untuk setiap kebutuhan sumber daya *task* T_i di komputer W_j yang akan kita sebut dengan TR_{ij} pada *i*=1…m dan j=1…n, akan didapatkan variabel variabel berikut:

- nomor *worker nw*,
- waktu respons *rt*
- Kebutuhan sumber daya komputasi *cu* dalam *millisceonds*
- Kebutuhan sumber daya memori *mu* dalam *megabytes*

Jika pada saat menjalankan T_i di komputer W_j *task* T_i tidak dapat terselesaikan maka kebutuhan sumber daya TR_{ij} tidak akan di simpan, hal ini menadakan komputer W_j tidak mampu menjalankan *task* T_i sehingga tidak dibutuhkan referensi kebutuhan sumber daya TR_{ij}.

3.2.3 Inisialisasi ketersediaan sumber daya

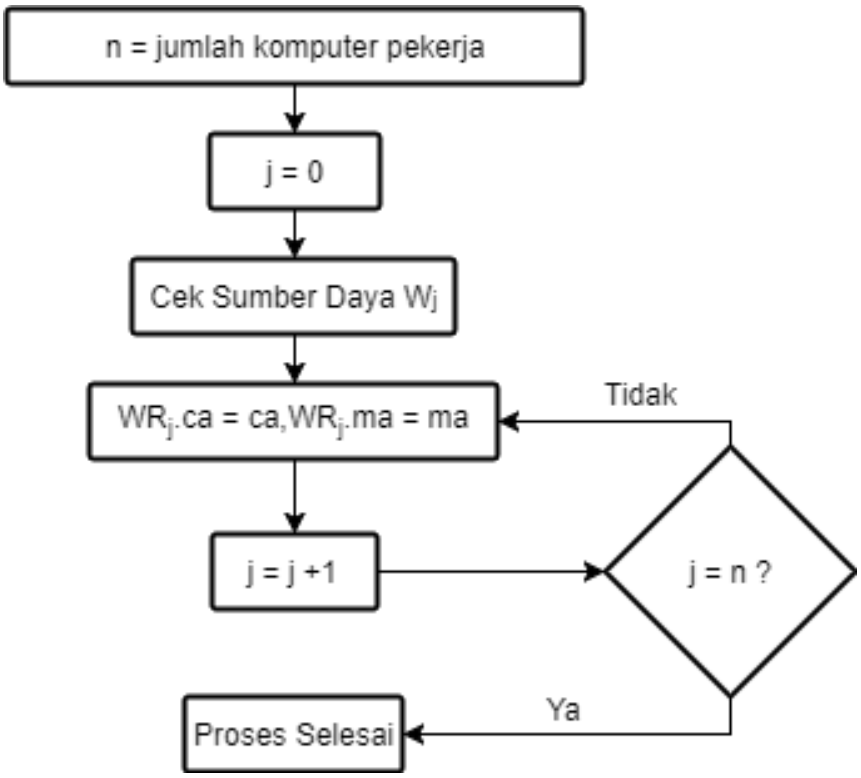
Ketersediaan sumber daya WR_i pada komputer W_i akan di cari terlebih dahulu, untuk melakukan pengelompokkan *task* dan memilih *worker* pada saat menjalankan algoritma penyeimbang beban. Proses inisialisasi ketersediaan sumber daya di tunjukan pada Gambar 3.6



Gambar 3.6 Proses Inisialisasi Ketersediaan Sumber Daya

Inisialisasi ketersediaan sumber daya dilakukan dengan melakukan koleksi data pada setiap *worker* kemudian dilakukan pengukuran ketersediaan memori dan ketersediaan komputasi cpu. Proses ini akan menghasilkan *Worker Resource* (*WR)* yang menyimpan informasi ketersediaan sumber daya memori (*ma)* dan komputasi *(cu) setiap worker. Worker Resource (WR)* akan di gunakan sebagai parameter proses *profiling task*

pada tahap (2). Diagram alur inisialisasi ketersediaan sumber daya dijelaskan dengan Gambar 3.7.



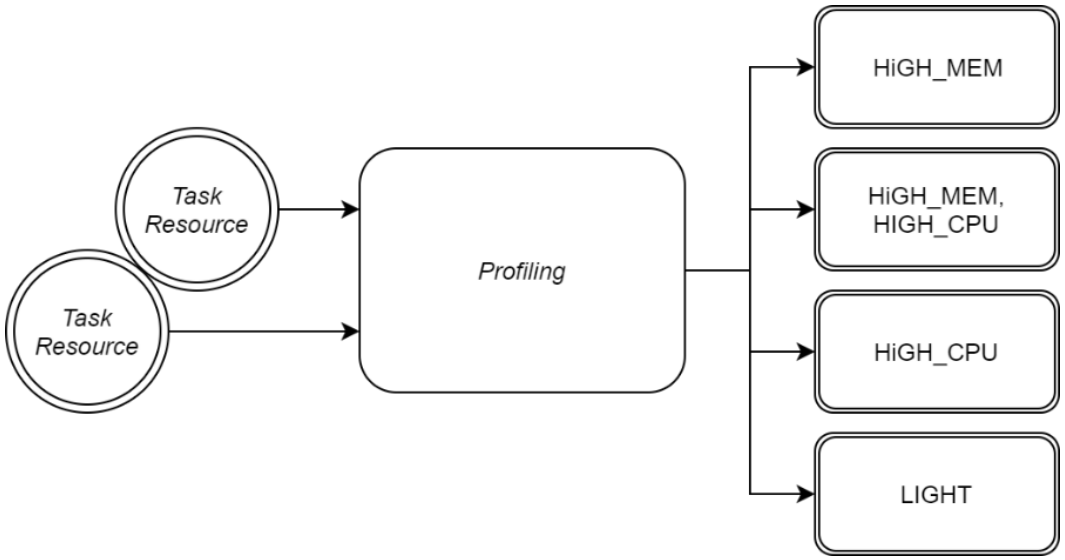
Gambar 3.7 Diagram Alur Inisialisasi Ketersediaan Sumber Daya

Dari diagram alur tersebut akan ditentukan variabel variabel berikut:

- Ketersediaan sumber daya komputasi *ca* berupa jumlah prosesor
- Ketersediaan sumber daya memori *ma* dalam *milliseconds*

3.2.4 Profiling *task*

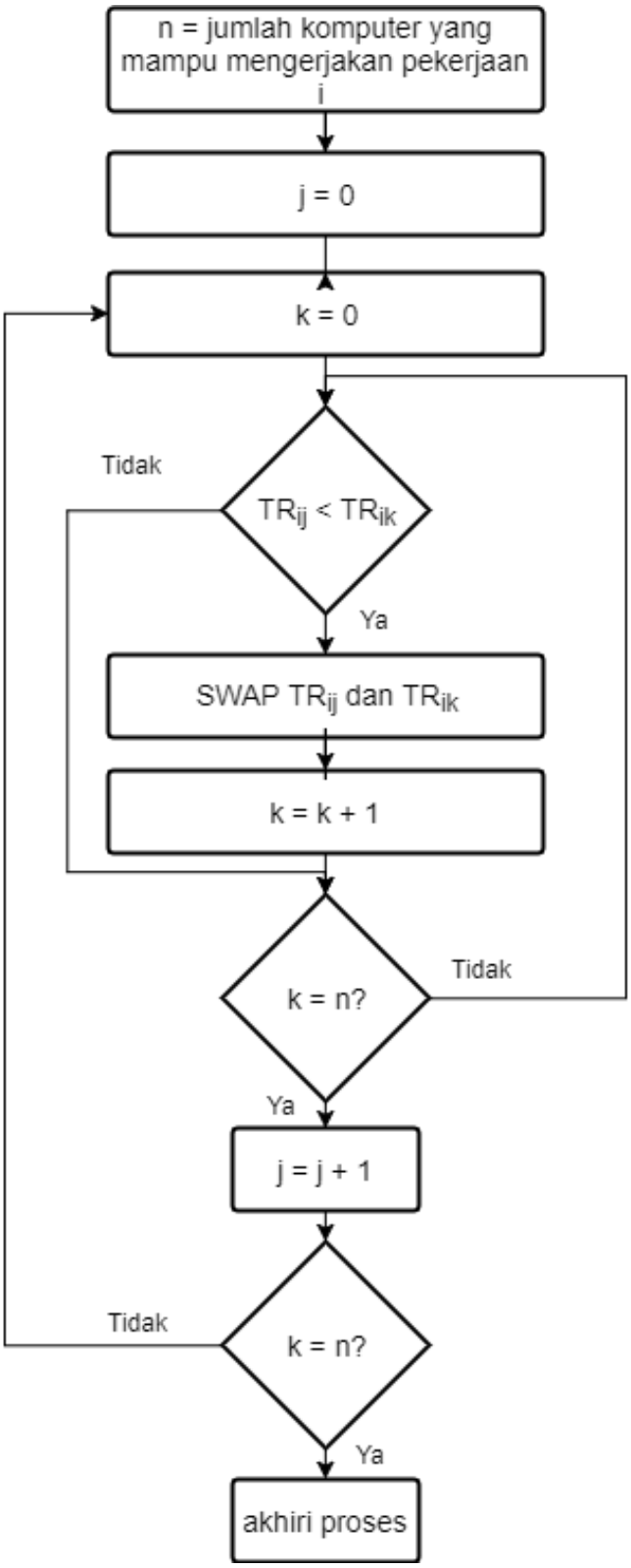
Setelah mendapatkan kebutuhan sumber daya *task TR* dan ketersediaan sumber daya *WR*, kemudian kebutuhan sumber daya *TR* akan dikelompokkan menjadi HIGH_MEM, HIGH_CPU, dan LIGHT melalui proses profiling. Pengelompokan ini dilakukan dengan memberi label pada setiap TR_i,i=1…m. Proses *profiling task* ditunjukkan pada Gambar 3.8.



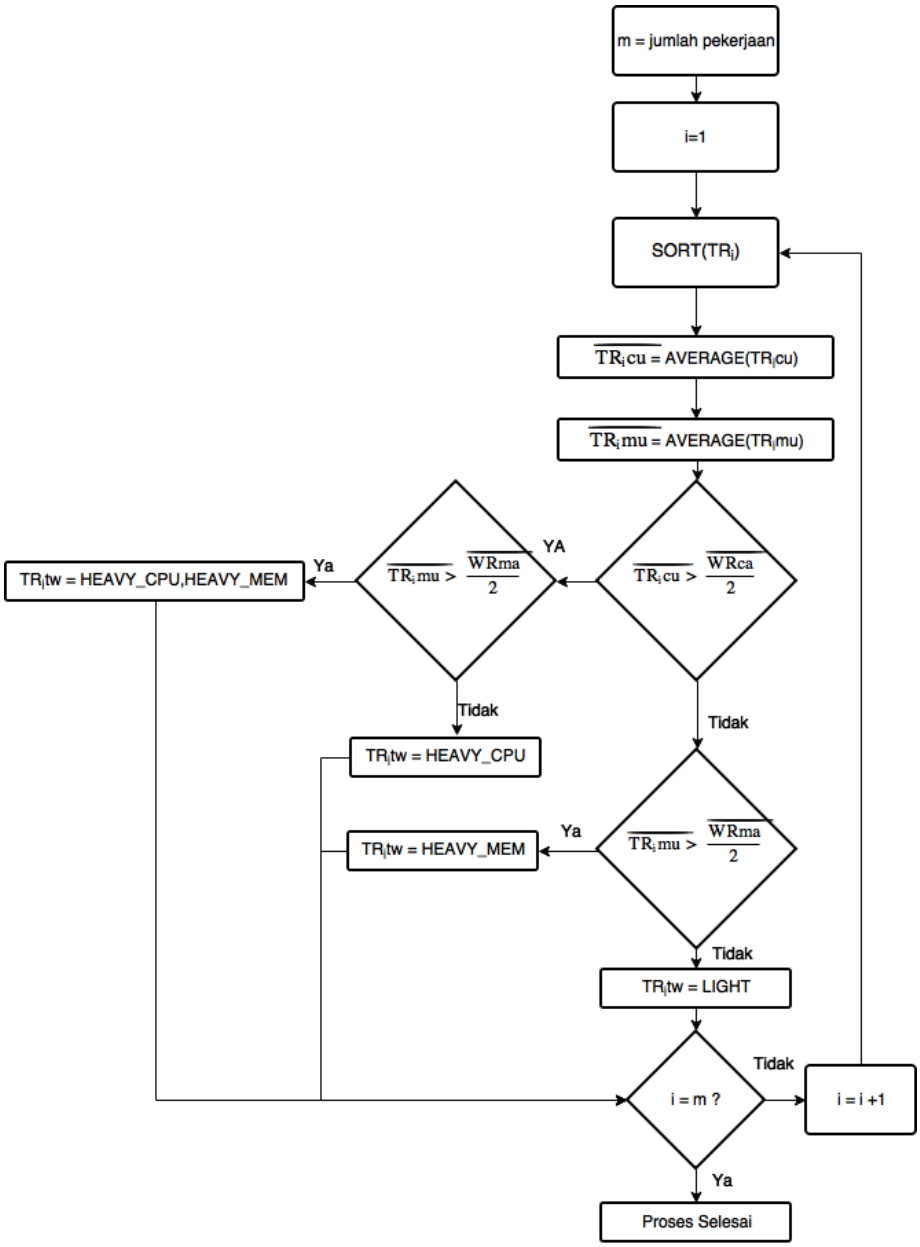
Gambar 3.8 Proses Profiling Task

Proses *profiling task* dilakukan dengan analisa data *Task Resource(TR)* dan *Worker Resource (WR)* yang di dapat menggunakan pengukuran standar pada semua *task* terhadap setiap *worker*. *Profiling task* bertujuan menghasilkan pengelompokan *task* menjadi HIGH_MEM, HIGH_CPU, dan LIGHT. Pengelompokan dilakukan dengan memberi label pada *Task Resource (TR)*. Label pada *Task Resource (TR)* akan digunakan sebagai referensi proses penyeimbang beban pada tahap (5). Diagram alur pengelompokan *task* di tunjukkan pada Gambar 3.9.

Untuk membantu penentuan pilihan pada algoritma penyeimbang beban, setiap kebutuhan sumber daya TR_i harus di urutkan berdasarkan waktu respons di setiap komputer W_j. Sehingga terurut berdasarkan *worker* yang menyelesaikan *task* denganlebih cepat. Pengurutan di tunjukan melalui diagram alur pada Gambar 3.10



Gambar 3.9 Diagram Alur Pengurutan



Gambar 3.10 Diagram Alur Pengelompokkan Task

Selanjutnya untuk setiap kebutuhan sumber daya TR_i akan ditentukan variabel variabel berikut:

- Rata-rata kebutuhan waktu komputasi $\overline{TR_\iota cu}$
- Rata-rata kebutuhan memori $\overline{TR_\iota mu}$
- Kelompok *task tw*

Rata-rata kebutuhan komputasi *task* $\overline{TR_\iota cu}$ pada persamaan (3.1) ditentukan dengan menghitung rata-rata kebutuhan komputasi *TRicu*.

$$\overline{TR_\iota cu} = \frac{\sum_{j=0}^{m} TR_{ij}cu}{m} \quad (3.1)$$

Rata-rata kebutuhan memori *task* $\overline{TR_\iota mu}$ pada persamaan (3.2) ditentukan dengan menghitung rata-rata kebutuhan memori *TRimu.*

$$\overline{TR_\iota mu} = \frac{\sum_{j=0}^{m} TR_{ij}mu}{m} \quad (3.2)$$

Setelah di dapatkan rata-rata kebutuhan komputasi $\overline{TR_\iota cu}$ dan rata-rata kebutuhan memori $\overline{TR_\iota mu}$, pengelompokkan dilakukan dengan membandingkan nilai rata-rata kebutuhan komputasi $\overline{TR_\iota cu}$ terhadap *threshold* waktu pengerjaan dan membandingkan nilai rata-rata kebutuhan memori $\overline{TR_\iota mu}$ dengan rata-rata ketersediaan sumber daya memori $\overline{WRma}$. Penggunaan nilai *threshold* adalah 8 second, karena meurut penelitian maksimal waktu respon yang dapat ditoleransi oleh pengguna adalah 8 second (Nah, 2003)

$$\overline{WRma} = \frac{\sum_{i=0}^{n} WR_i ma}{n} \quad (3.3)$$

Jika $\overline{TR_\iota cu} > threshold$, maka pengelompokan $TR_i tw$ adalah HIGH_CPU,

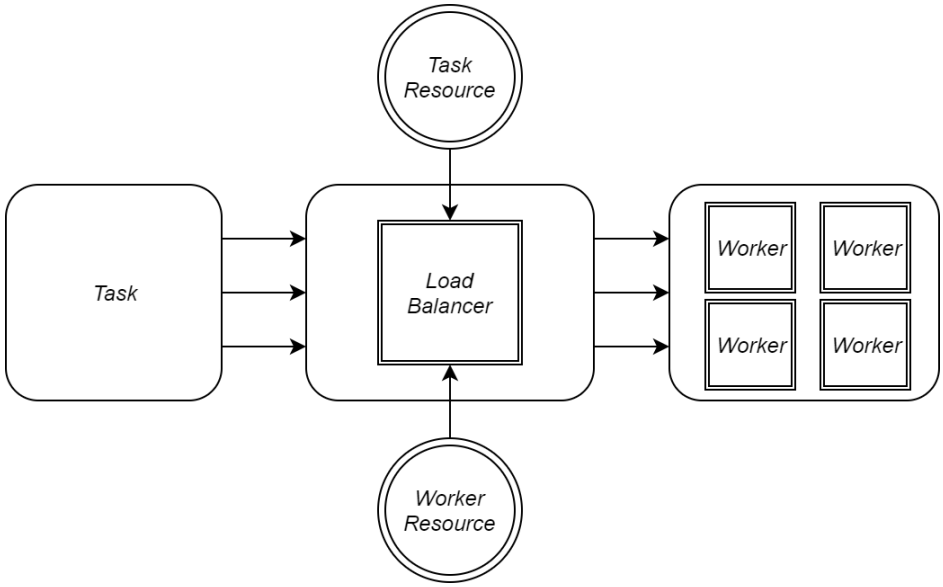
Jika $\overline{TR_\iota mu} > \frac{\overline{WRma}}{2}$, maka pengelompokan $TR_i tw$ adalah HIGH_MEM,

Jika $\overline{TR_\iota cu} > threshold$ dan $\overline{TR_\iota mu} > \frac{\overline{WRma}}{2}$, maka pengelompokan $TR_i tw$ adalah HIGH_CPU dan HIGH_MEM,

Jika $\overline{TR_\iota cu} < threshold$ dan $\overline{TR_\iota mu} < \frac{\overline{WRma}}{2}$, maka pengelompokan $TR_i tw$ adalah LIGHT.

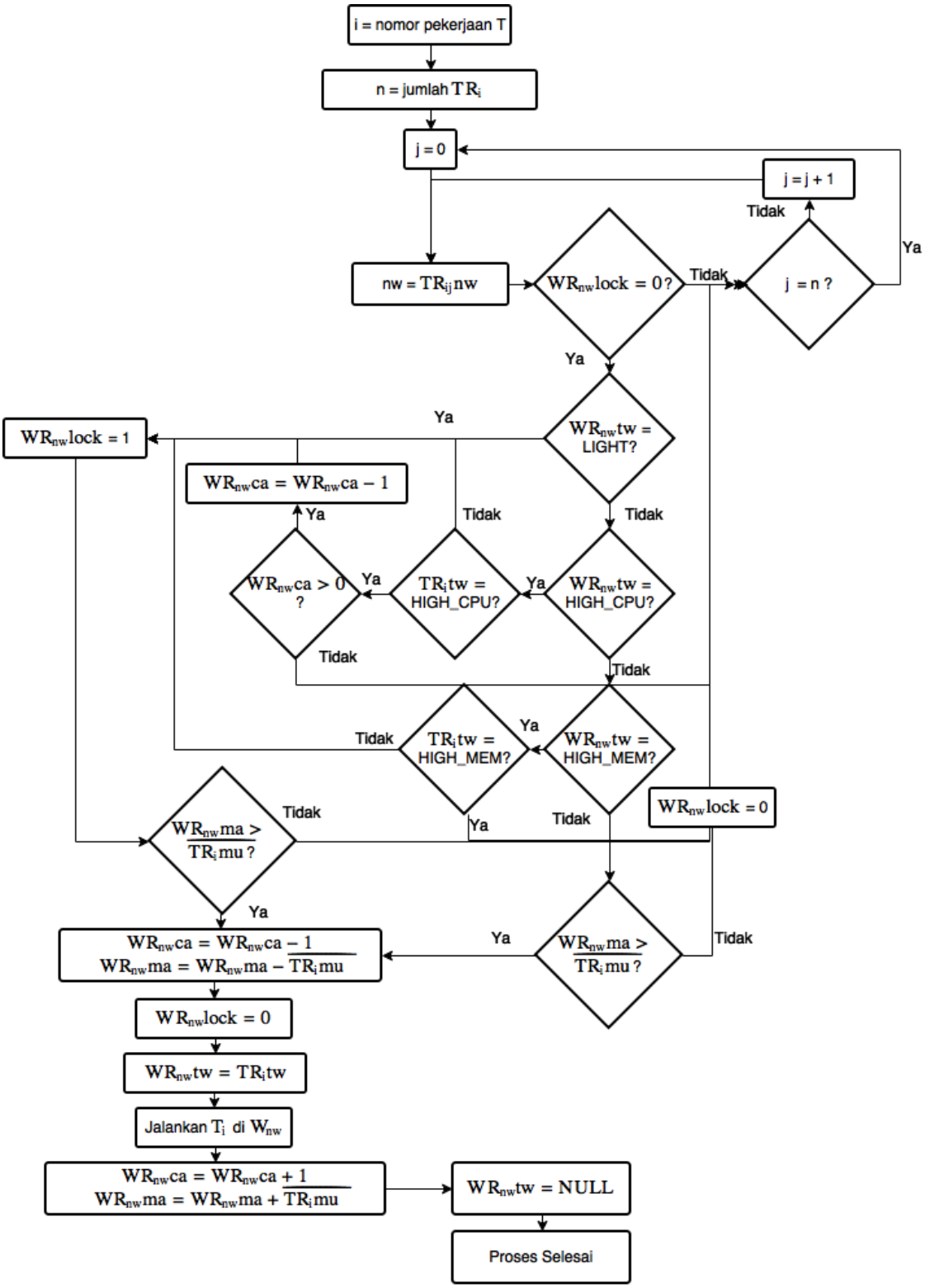
3.2.5 Algoritma Penyeimbang Beban

Algoritma penyeimbang beban yang diusulkan memanfaatkan informasi informasi yang di dapat dari tahap tahap sebelumnya seperti *Task Resource* (*TR)* dan *Worker Resource* (*WR).* Algoritma akan dijalankan setiap *task* baru datang. Proses berjalannya algoritma pembagian beban tunjukkan pada Gambar 3.11.



Gambar 3.11 Proses Algoritma Penyeimbang Beban

Task yang akan di proses oleh klaster komputer akan di terima terlebih dahulu oleh komputer *load balancer*. Komputer *load balancer* akan menggunakan informasi *Task Resource (TR)* dan *Worker Resource (WR)* yang di dapatkan pada tahap-tahap sebelumnya sebagai referensi untuk menentukan *worker* yang akan di berikan *task.* Informasi *Worker Resource (WR)* akan diperbarui setiap *worker* menerima *task* untuk menjaga informasi tetap releven. Diagram alur algoritma penyeimbang beban dijelaskan pada Gambar 3.12.



Gambar 3.12 Diagram Alur Algoritma Penyeimbang Beban

Untuk *task* T_i yang akan dikerjakan pertama-tama akan di ambil referensi kebutuhan sumber daya TR_i dan referensi ketersediaan sumber daya WR. Kemudian akan dipilih *worker* W melalui kebutuhan sumber daya TR_i dengan dipilih yang waktu respons nya paling kecil, sehingga nomor komputer $nw = min\{TR_{ij}rt | j = \{0 \ldots . k\}$ dimana *k* adalah jumlah total komputer yang mampu menangani *task* T_i. Komputer yang boleh dipilih adalah komputer yang tidak sedang terkunci $WR_i lock = 0$. Komputer yang terkunci menandakan sedang ditinjau sebagai kandidat pekerja oleh *task* lain. Setelah di dapatkan nomor komputer *nw* yang sesuai akan dilakukan pengecekan kelompok *task*. Pengecekan kelompok *task* tw bertujuan untuk memastikan *worker* tidak sedang mengerjakan *task* lain dengan kelompok yang membutuhkan komputasi yang sama tingginya. Ketika *task* T_i di putuskan untuk dikerjakan pada komputer W_j maka ketersediaan komputer WR_j akan di tandai sedang mengerjakan kelompok *task* $TR_{ij}tw$, sehingga $WR_j tw = TR_{ij}tw$ hingga *task* terselesaikan.

Jika $= nw,\ TR_{ij}tw = HIGH\_MEM\ dan\ WR_j tw = HIGH\_MEM$, maka komputer W_j tidak dapat digunakan sebagai kandidat untuk mengerjakan *task* T_i dan harus mencari komputer lain

Jika $= nw,\ TR_{ij}tw = HIGH\_MEM\ dan\ WR_j tw = HIGH\_CPU$, maka komputer W_j dapat digunakan sebagai kandidat, Karena $TR_{ij}tw \neq WR_j tw$, menandakan *task* T_i memiliki kelompok yang berbeda dengan *task* yang sedang dikerjakan di komputer W_j.

Jika $WR_j tw = HIGH\_CPU$, sedangkan $\overline{WR_J ca} = 0$, maka komputer W_j tidak dapat digunakan karena komputer W_j tidak memiliki prosesor yang tersedia. Semua pekerjaan dengan *task weight(tw) HIGH_CPU* hanya boleh dikerjakan pada komputer W jika komputer W memiliki prosesor yang sedang tidak bekerja(*idle).*

Jika $j = nw,\ TR_{ij}tw = LIGHT$, maka komputer W_j dapat digunakann sebagai kandidat, Karena sedang tidak mengerjakan *task* yang termasuk dalam kelompok *task* tinggi.

Setelah kandidat komputer W_j terpilih, maka akan diperiksa ketersediaan sumber daya $WR_jca\ dan\ WR_jma$ memeuhi rata-rata kebutuhan sumber daya $\overline{TR_jcu}\ dan\ \overline{TR_jmu}$.

Jika $WR_jca < \overline{TR_jcu}\ atau\ WR_jma < \overline{TR_jmu}$ maka komputer W_j tidak dapat digunakan dan algoritma akan memeriksa komputer selanjutnya.

Jika $WR_jca > \overline{TR_jcu}\ dan\ WR_jma > \overline{TR_jmu}$ maka komputer W_j akan mengerjakan *task* T_i.

Sebelum *task* dijalankan maka:

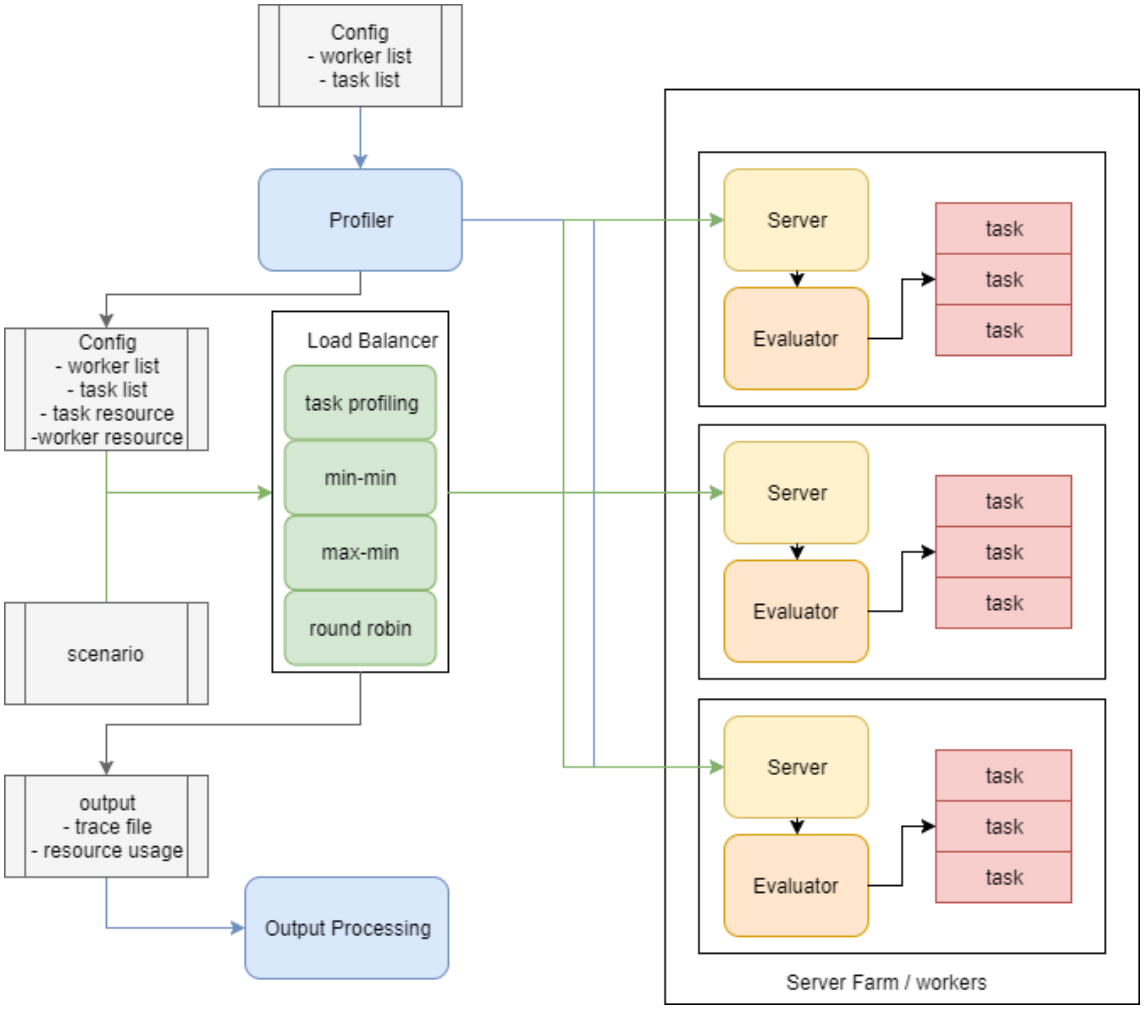
- kunci komputer W_j harus dibuka sehingga $W_jlock = 0$ agar komputer W_j dapat diperiksa oleh *task* yang datang selanjutnya.
- Ketersediaan sumber daya WR harus diperbarui dengan pengurangan$WR_jca - 1$, jika task adalah HIGH_CPU $dan\ WR_jma - \overline{TR_jmu}$ pada setiap task.
- Tandai komputer W_j sedang mengerjakan kelompok *task* T_i, ubah $WR_jtw = TR_{ij}tw$.

Ketika *task* selesai dikerjakan maka:

- ketersediaan sumber daya WR harus diperbarui dengan penambahan $WR_jca + 1$ jika task adalah HIGH_CPU $dan\ WR_jma + \overline{TR_jmu}$ pada setiap task agar referensi ketersediaan sumber daya WR tetap sesuai digunakan sebagai referensi oleh *task-task* selanjutnya.
- Kosongkan tanda kelompok *task* yang sedang dikerjakan oleh komputer W_j, ubah $WR_jtw = NULL$.

3.3 Implementasi

Implementasi akan dilakukan menggunakan python2.7 dan pustaka psutil. Rancangan implementasi sistem terdiri dari profiler, evaluator, server dan load balancer. Gambar arsitektur sistem pengujian ditunjukan pada gambar 3.13.



Gambar 3.13 Arsitektur Sistem Pengujian

Penjelasan sistem pengujian sebagai berikut:

- Config berisi alamat ip dari worker yang digunakan, daftar *task* unik yang akan dijalankan
- Profiler adalah subsistem yang bertugas melakukan profiling setiap *task* ke semua worker. Hasil profiler akan memperbarui data pada config yang kemudian akan digunakan oleh load balancer.

- Load balancer adalah subsistem untuk menjalankan pengujian sesuai skenario yang digunakan. Load balancer memiliki empat algoritma yaitu round robin, min-min, max-min dan *task profiling*. Load balancer menggunakan referensi data config yang dihasilkan oleh profiler pada proses profiling sebelumnya. Hasil pengujian berupa output yang terdiri dari *trace file* dan *resource usage*.
- *trace file* adalah catatan *event* yang berlangsung pada saat simulasi berupa waktu, *worker, task* dan *event*. Sedangkan *resource usage* adalah penggunaan sumber daya pada setiap *task* dijalankan.
- Output Processing akan memproses *trace file* dan *resource usage* untuk mendapatkan analisa data.

3.4 Pengujian

Pengujian di lakukan pada platform virtualisasi, virtualisasi menggunakan proxmox. Spesifikasi komputer host proxmox yang digunakan sebagai berikut

- Cpu 8 Core dengan spesifikasi 8 x Intel(R) Core(TM) i7-2700K CPU @ 3.50GHz (1 Socket)
- RAM 16 GB
- HDD 100 GB
- Kernel Version Linux 4.13.4-1-pve #1 SMP PVE 4.13.4-25 (Fri, 13 Oct 2017 08:59:53 +0200)
- PVE Manager Version pve-manager/5.1-35/722cc488

Untuk setiap mesin virtual yang di buat memiliki spesifikasi sebagai berikut

- Nama = kucing1
    - Jumlah Prosesor = 1
    - Memori = 1024 MB
    - HDD = 32G
- Nama = kucing2
    - Jumlah Prosesor = 1
    - Memori = 1024 MB
    - HDD = 32G

- Nama = kucing3
    - Jumlah Prosesor = 1
    - Memori = 1024 MB
    - HDD = 32G
- Nama = kucing4
    - Jumlah Prosesor = 4
    - Memori = 512 MB
    - HDD = 32G
- Nama = kucing5
    - Jumlah Prosesor = 1
    - Memori = 2048 MB
    - HDD = 32G
- Nama = kucing6
    - Jumlah Prosesor = 4
    - Memori = 2048 MB
    - HDD = 32G

Pengujian dilakukan dengan mengukur rata-rata waktu respons, rata-rata penggunaan memori, rata-rata waktu cpu beroperasi dan total waktu penyelesaian semua *task.* Parameter pengukuran tersebut digunakan untuk meningkatkan mutu layanan (Nah, 2003) (Al-Moayed & Hollunder, 2010). Pengujian dilakukan menggunakan *worker* dengan spesifikasi sama dan dengan spesifikasi berbeda, pengujian juga dilakukan menggunakan *task* dengan kebutuhan sumber daya sama dan kebutuhan sumber daya yang berbeda. Pengujian dilakukan menjadi 4 macam skenario, dijelaskan pada Tabel 3.1.

Tabel 3.1 Daftar Skenario Pengujian

Nama	Worker			Task	
Worker seragam dan task seragam (wsts)	Nama	Memori	Prosesor	Nama	Keterangan
	Kucing1	1024 MB	1	write_high	Operasi tulis 20000000 baris
	Kucing2	1024 MB	1		
	Kucing3	1024 MB	1		
Worker seragam dan task berbeda (wstd)	Nama	Memori	Prosesor	Nama	Keterangan
	Kucing1	1024 MB	1	write_high	Operasi tulis 20000000 baris
	Kucing2	1024 MB	1	read_high	Operasi baca 15000000 baris
	Kucing3	1024 MB	1	read_moderate	operasi baca 5000000 baris
Worker berbeda dan task seragam (wdts)	Nama	Memori	Prosesor	Nama	Keterangan
	Kucing4	512 MB	4	write_high	Operasi tulis 20000000 baris
	Kucing5	2048 MB	1		
	Kucing6	2048 MB	4		
Worker berbeda dan task berbeda (wdtd)	Nama	Memori	Prosesor	Nama	Keterangan
	Kucing4	512 MB	4	write_high	Operasi tulis 20000000 baris
	Kucing5	2048 MB	1	read_high	Operasi baca 15000000 baris
	Kucing6	2048 MB	4	read_moderate	operasi baca 5000000 baris

A. Worker seragam dan task seragam(wsts)

Pengujian dilakukan sebanyak 5 step dimana masing masing step jumlah task akan bertambah secara linier. Detail pengujian di jelaskan pada tabel 3.2

Tabel 3.2 Skenario wsts

Step	Worker			Task	
wsts-1.1/ Step 1	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing1	1024 MB	1	write_high	15
	Kucing2	1024 MB	1		
	Kucing3	1024 MB	1		
wsts-1.2/ Step 2	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing1	1024 MB	1	write_high	30
	Kucing2	1024 MB	1		
	Kucing3	1024 MB	1		
wsts-1.3/ Step 3	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing1	1024 MB	1	write_high	45
	Kucing2	1024 MB	1		
	Kucing3	1024 MB	1		
wsts-1.4/ Step 4	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing1	1024 MB	1	write_high	60
	Kucing2	1024 MB	1		
	Kucing3	1024 MB	1		
wsts-1.5/ Step 5	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing1	1024 MB	1	write_high	75
	Kucing2	1024 MB	1		
	Kucing3	1024 MB	1		

B. Worker seragam dan task berbeda (wstd)

Pengujian dilakukan sebanyak 5 step dimana masing masing step jumlah task akan bertambah secara linier. Detail pengujian di jelaskan pada tabel 3.3

Tabel 3.3 Skenario wstd

Step	Worker			Task	
wsts-1.1/ Step 1	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing1	1024 MB	1	write_high	5
	Kucing2	1024 MB	1	read_high	5
	Kucing3	1024 MB	1	read_moderate	5
wsts-1.2/ Step 2	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing1	1024 MB	1	write_high	10
	Kucing2	1024 MB	1	read_high	10
	Kucing3	1024 MB	1	read_moderate	10
wsts-1.3/ Step 3	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing1	1024 MB	1	write_high	15
	Kucing2	1024 MB	1	read_high	15
	Kucing3	1024 MB	1	read_moderate	15
wsts-1.4/ Step 4	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing1	1024 MB	1	write_high	20
	Kucing2	1024 MB	1	read_high	20
	Kucing3	1024 MB	1	read_moderate	20
wsts-1.5/ Step 5	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing1	1024 MB	1	write_high	25
	Kucing2	1024 MB	1	read_high	25
	Kucing3	1024 MB	1	read_moderate	25

C. Worker berbeda dan task seragam (wdts)

Pengujian dilakukan sebanyak 5 step dimana masing masing step jumlah task akan bertambah secara linier. Detail pengujian di jelaskan pada tabel 3.4

Tabel 3.4 Skenario wdts

Step	Worker			Task	
wsts-1.1/ Step 1	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing4	512 MB	4	write_high	15
	Kucing5	2048 MB	1		
	Kucing6	2048 MB	4		
wsts-1.2/ Step 2	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing4	512 MB	4	write_high	30
	Kucing5	2048 MB	1		
	Kucing6	2048 MB	4		
wsts-1.3/ Step 3	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing4	512 MB	4	write_high	45
	Kucing5	2048 MB	1		
	Kucing6	2048 MB	4		
wsts-1.4/ Step 4	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing4	512 MB	4	write_high	60
	Kucing5	2048 MB	1		
	Kucing6	2048 MB	4		
wsts-1.5/ Step 5	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing4	512 MB	4	write_high	75
	Kucing5	2048 MB	1		
	Kucing6	2048 MB	4		

D. Worker berbeda dan task berbeda (wdtd)

Pengujian dilakukan sebanyak 5 step dimana masing masing step jumlah task akan bertambah secara linier. Detail pengujian di jelaskan pada tabel 3.5

Tabel 3.5 Skenario wstd

Step	Worker			Task	
wsts-1.1/ Step 1	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing4	512 MB	4	write_high	5
	Kucing5	2048 MB	1	read_high	5
	Kucing6	2048 MB	4	read_moderate	5
wsts-1.2/ Step 2	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing4	512 MB	4	write_high	10
	Kucing5	2048 MB	1	read_high	10
	Kucing6	2048 MB	4	read_moderate	10
wsts-1.3/ Step 3	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing4	512 MB	4	write_high	15
	Kucing5	2048 MB	1	read_high	15
	Kucing6	2048 MB	4	read_moderate	15
wsts-1.4/ Step 4	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing4	512 MB	4	write_high	20
	Kucing5	2048 MB	1	read_high	20
	Kucing6	2048 MB	4	read_moderate	20
wsts-1.5/ Step 5	Nama	Memori	Prosesor	Nama	Jumlah
	Kucing4	512 MB	4	write_high	25
	Kucing5	2048 MB	1	read_high	25
	Kucing6	2048 MB	4	read_moderate	25

3.5 Evaluasi

Pada tahap ini akan dilakukan evaluasi untuk menguji kinerja dari algoritma Penyeimbang beban. Selain melakukan evaluasi terhadap waktu respons, evaluasi akan dilakukan pada beberapa parameter sebagai berikut

- Waktu Eksekusi

  Evaluasi akan dilakukan pada rata-rata waktu eksekusi setiap *task*.

- Waktu Terselesaikan

  Waktu semua task terselesaikan digunakan sebagai tolok ukur utama evaluasi kinerja. Semakin pendek waktu terselesaikan semua pekerjaan maka semakin baik juga algoritma penyeimbang beban
- Utilisasi

  Utilisasi di gunakan sebagai tolok ukur keadilan dari algoritma penyeimbang beban, utilisasi antar *worker* diharapkan seimbang untuk menghindari komputer yang kelebihan beban
- Prosentase *task* terselesaikan

  Prosentase jumlah *task* yang terselesaikan menunjukan kehandalan sistem penyeimbang beban dalam melayani permintaan *task*. Diharapkan prosentase *task* terselesaikan mencapai 100%
- Ketepatan pemilihan *worker*

  Evaluasi ini diperlukan untuk mengetahui algoritma penyeimbang beban telah berjalan dengan benar mendistribusikan *task* ke komputer yang tepat.

3.6 Penyusunan Buku Tesis

Penyusunan buku tesis dilakukan sebagai dokumentasi terhadap serangkaian penelitian yang dikerjakan agar dapat dijadikan sebagai bahan pembelajaran, referensi maupun sebuah perbaikan penelitian di masa depan yang berhubungan dengan perbaikan mekanisme *update* informasi.

[Halaman ini sengaja dikosongkan]

BAB 4
HASIL DAN PEMBAHASAN

Pada bab 4 dijelaskan tahapan analisis kinerja sistem pendistribusian beban berdasarkan hasil uji coba dengan 4 macam skenario uji yang telah dijelaskan pada sub bab 3.4. Evaluasi dilakukan pada semua algoritma penyeimbang beban terhadap 4 macam skenario uji. Arsitektur yang digunakan seperti yang sudah dijelaskan pada sub bab 3.3. Parameter perbandingan yang digunakan adalah waktu eksekusi, waktu terselesaikan, utilisasi, prosentase task terselesaikan dan ketepatan pemilihan worker. Sebelum melakukan ujicoba untuk setiap skenario akan dilakukan profiling *task* pada setiap task. Profiling *task* akan dilakukan pertama kali di setiap akan memulai skenario untuk mendapatkan TR dan WR seperti pada sub bab 3.2.4. Hasil profiling *task* pada ditunjukkan pada tabel 4.1 dan tabel 4.2

Tabel 4.1 Parameter TR pada Profiling Task

No	Nama Task	Waktu cpu	Penggunaan memori	Pengelompokan
1	write_high	8,67	53,3 MB	HIGH_CPU
2	read_high	0,83	442,67 MB	HIGH_MEMORY
3	read_moderate	0,04	138 MB	LIGHT

Tabel 4.2 Parameter WR pada Profiling Task

No	Nama Worker	Cpu Prosesor	Total Memori	Memori Tersedia
1	kucing1	1	1024MB	556 MB
2	kucing2	1	1024 MB	750 MB
3	kucing3	1	1024 MB	802 MB

Ketiga tipe *task* dipilih dalam ujicoba karena ketiga *task* sudah mewakili 3 pengelompokan pada profiling *task*. Pada keempat skenario yang akan dijalankan

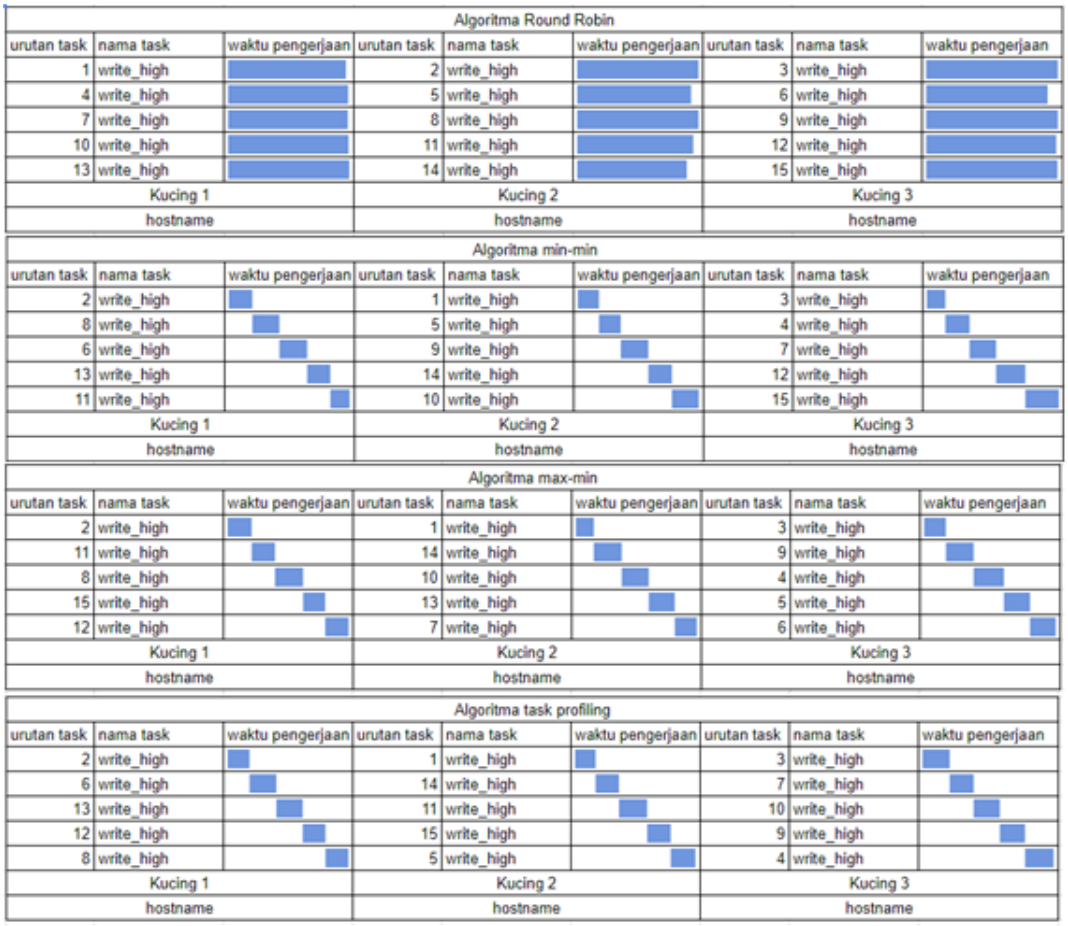
penggunaan waktu cpu dan memori akan berbeda tetapi pengelompokan *task* tetap sama. Total memori yang tersedia pada setiap *worker* adalah 1024 MB, tetapi yang tersedia dan dapat digunakan tidak mencapai 1024 MB. Hal ini disebabkan karena kebutuhan sistem operasi pada memori.

Hasil pengujian tiap tiap skenario akan dijelaskan dan di evaluasi satu persatu. Pengujian akan membandingkan kinerja algoritma Round Robin, Min-Min, Max-Min dan algoritma penyeimbang *task* profiling yang di ajukan.

A. Worker Seragam Task Seragam (wsts)

   Pengujian dilakukan pada lingkungan *worker* dan *task* yang seragam. *Task* yang digunakan adalah *task* write_high, karena termasuk dalam kategori *HIGH_CPU*. *Task* dengan kategori *HIGH_CPU* mampu menunjukan perbedaan yang baik pada setiap algoritma dalam hal waktu terselesaikan dan distribusi *task* ke setiap *worker* dibandingkan dengan *task* dengan kategori *HIGH_MEMORY* atau *LIGHT*. Distribusi beban pada skenario wsts dijelaskan pada tabel 4.3

Tabel 4.3 Distribusi Beban Skenario WSTS



Algoritma Round Robin								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
1	write_high		2	write_high		3	write_high	
4	write_high		5	write_high		6	write_high	
7	write_high		8	write_high		9	write_high	
10	write_high		11	write_high		12	write_high	
13	write_high		14	write_high		15	write_high	
Kucing 1			Kucing 2			Kucing 3		
hostname			hostname			hostname		

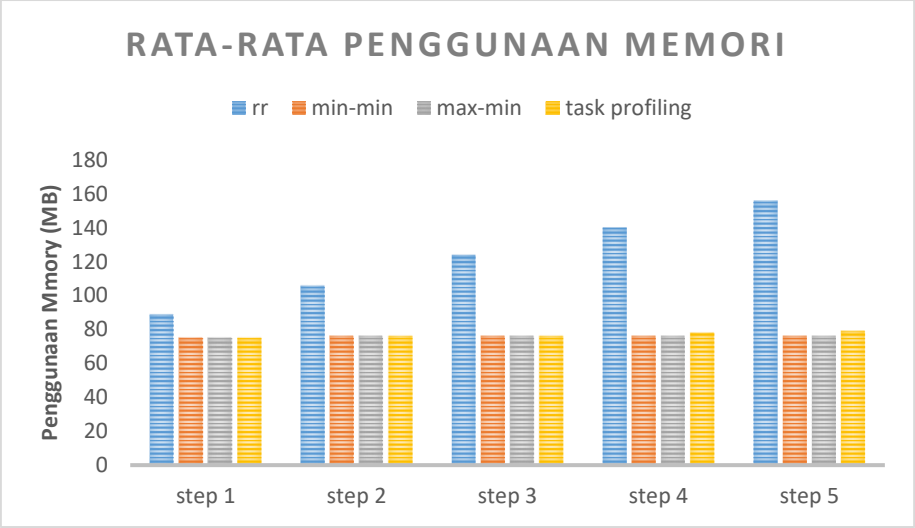
Algoritma min-min								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
2	write_high		1	write_high		3	write_high	
8	write_high		5	write_high		4	write_high	
6	write_high		9	write_high		7	write_high	
13	write_high		14	write_high		12	write_high	
11	write_high		10	write_high		15	write_high	
Kucing 1			Kucing 2			Kucing 3		
hostname			hostname			hostname		

Algoritma max-min								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
2	write_high		1	write_high		3	write_high	
11	write_high		14	write_high		9	write_high	
8	write_high		10	write_high		4	write_high	
15	write_high		13	write_high		5	write_high	
12	write_high		7	write_high		6	write_high	
Kucing 1			Kucing 2			Kucing 3		
hostname			hostname			hostname		

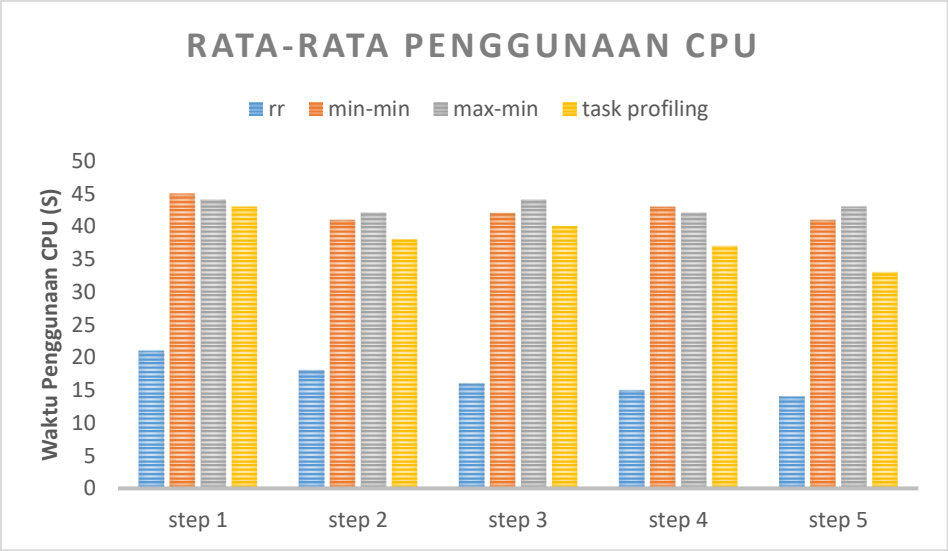
Algoritma task profiling								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
2	write_high		1	write_high		3	write_high	
6	write_high		14	write_high		7	write_high	
13	write_high		11	write_high		10	write_high	
12	write_high		15	write_high		9	write_high	
8	write_high		5	write_high		4	write_high	
Kucing 1			Kucing 2			Kucing 3		
hostname			hostname			hostname		

Distribusi beban pada scenario wsts menunjukkan setiap algoritma membagi *task* secara rata ke semua *worker*. Algoritma round robin menjalankan semua *task* secara bersamaan sedangkan algoritma min-min, max-min dan *task profiling* menjalankan secara bergantian agar *worker* tidak kelebihan beban. Pada skenario wsts *task profiling* berjalan seperti min-min dan max-min karena *worker* dan *task* seragam.

Pada skenario wsts utilisasi memori dan utilisasi cpu ditunjukkan pada gambar 4.1 dan gambar 4.2



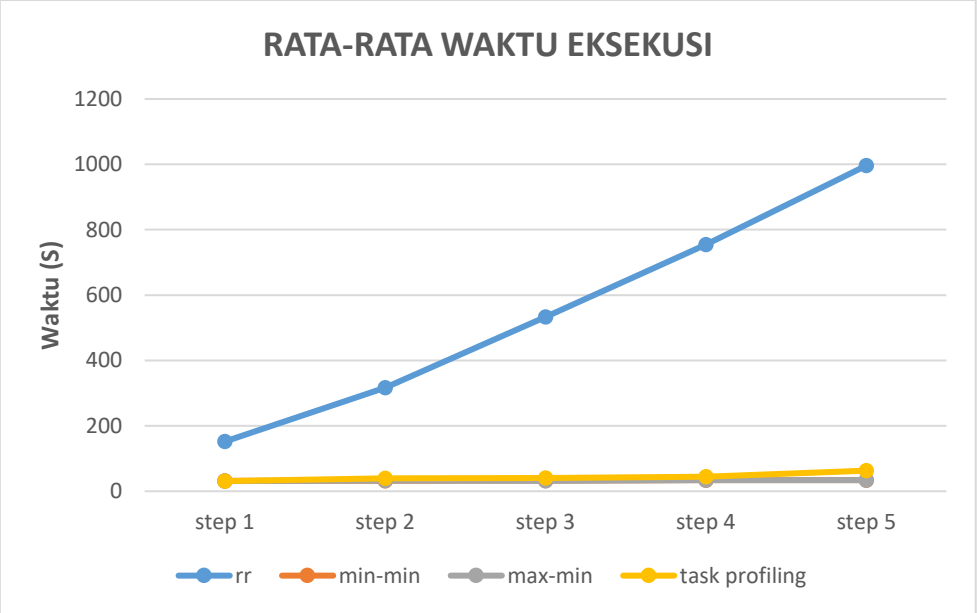
Gambar 4.1 Utilisasi Memori Skenario WSTS



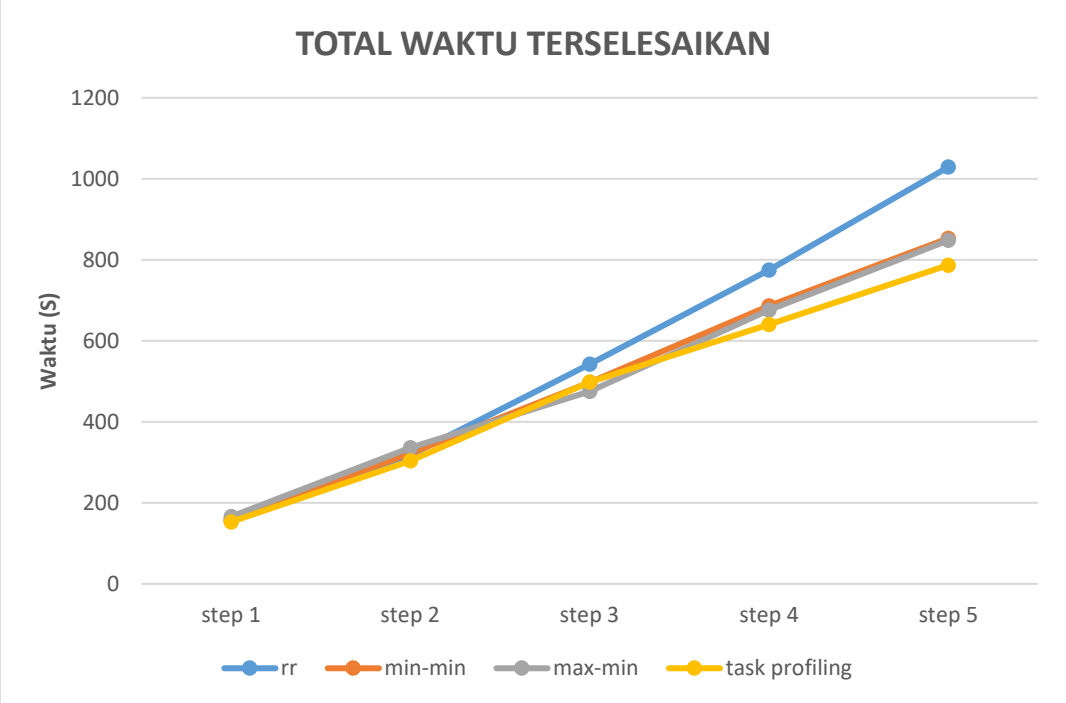
Gambar 4.2 Utilisasi CPU Skenario WSTS

Pada skenario wsts prosentase *task* terselesaikan pada keempat algoritma mencapai 100%. Karena *task* yang dijalankan memiliki kebutuhan sumber daya yang sama.

Pada skenario wsts waktu eksekusi dan waktu *task* terselesaikan ditunjukkan pada gambar 4.3 dan gambar 4.4



Gambar 4.3 Rata-Rata Waktu Eksekusi Skenario WSTS



Gambar 4.4 Total Waktu Terselesaikan Task Skenario WSTS

B. Worker Seragam Task Berbeda (wstd)

Sesuai dengan tabel 3.3, pengujian dilakukan pada lingkungan *worker* yang seragam dengan task yang berbeda. *Task* yang digunakan adalah ketiga macam *task*

dengan jumlah sesuai tabel 3.3. Distribusi beban pada skenario wsts dijelaskan pada tabel 4.4

Tabel 4.4 Distribusi Beban Skenario WSTD

Algoritma Round Robin								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
1	read_high		2	read_high		3	read_moderate	
4	read_moderate		5	read_moderate		6	write_high	
7	write_high		8	write_high		9	read_high	
10	write_high		11	read_high		12	write_high	
13	read_moderate		14	read_moderate		15	read_high	
Kucing 1			Kucing 2			Kucing 3		
hostname			hostname			hostname		

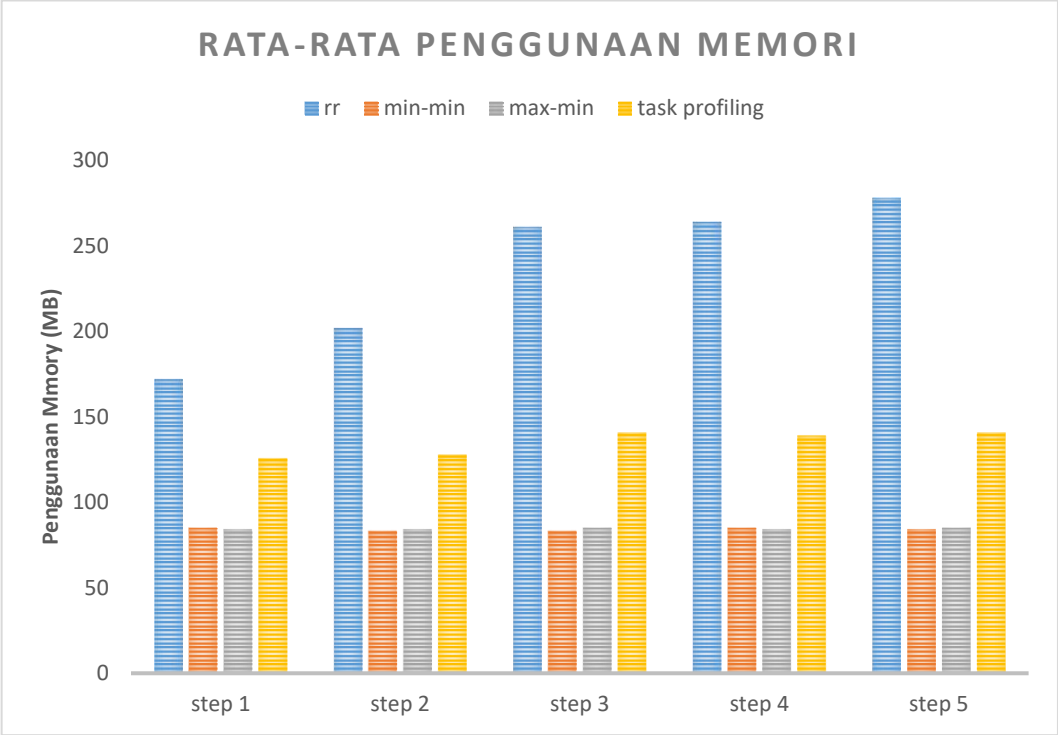
Algoritma min-min								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
2	read_moderate		4	read_moderate		5	read_moderate	
9	read_high		13	read_high		10	read_high	
14	write_high		8	write_high		11	write_high	
3	read_moderate		15	write_high		7	read_moderate	
12	read_high		6	write_high		1	read_high	
Kucing 1			Kucing 2			Kucing 3		
hostname			hostname			hostname		

Algoritma max-min								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
2	read_moderate		3	write_high		1	read_high	
10	read_high		12	write_high		5	read_high	
13	read_high		7	read_high		9	read_moderate	
6	write_high		4	read_moderate		14	read_moderate	
8	write_high		15	write_high		11	read_moderate	
Kucing 1			Kucing 2			Kucing 3		
hostname			hostname			hostname		

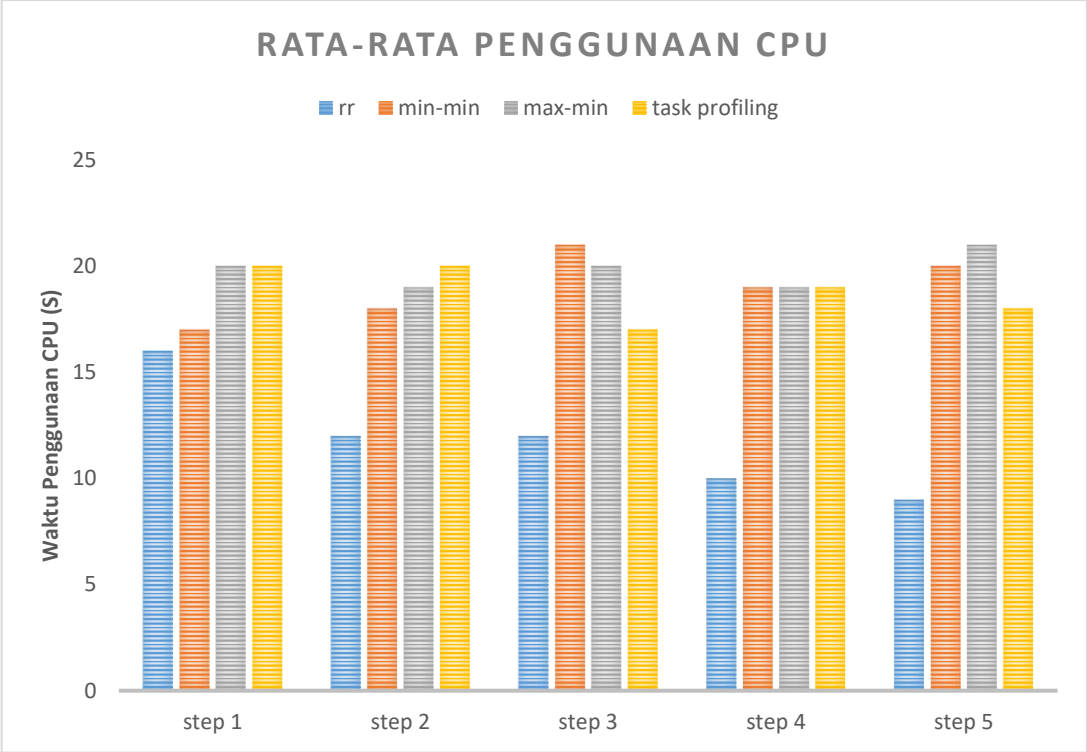
Algoritma task profiling								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
1	read_moderate		4	write_high		2	write_high	
5	read_high		6	read_moderate		3	read_high	
7	write_high		8	read_moderate		9	write_high	
14	write_high		10	read_moderate		15	read_high	
12	read_high		11	read_moderate				
Kucing 1			Kucing 2			Kucing 3		
hostname			hostname			hostname		

Pada pengujian skenario wstd terlihat perbedaan antara min-min dan max-min adalah pada *task* yang dijalankan. Algoritma min-min mengerjakan task dengan waktu eksekusi terpendek sedangkan algoritma max-min mengerjakan *task* dengan waktu eksekusi terpanjang. Tetapi algoritma min-min dan max-min hanya mengerjakan satu pekerjaan dalam satu waktu. Pada algoritma *task profiling* beberapa *task* dapat dikerjakan secara bersamaan berdasarkan kriteria hasil profiling. Sehingga *worker* dapat memproses banyak *task* dengan lebih optimal dan tetap tidak membebani *worker*. Sedangkan algoritma round robin tetap berjalan seperti skenario wsts.

Pada skenario wstd utilisasi memori dan utilisasi cpu ditunjukkan pada gambar 4.5 dan gambar 4.6

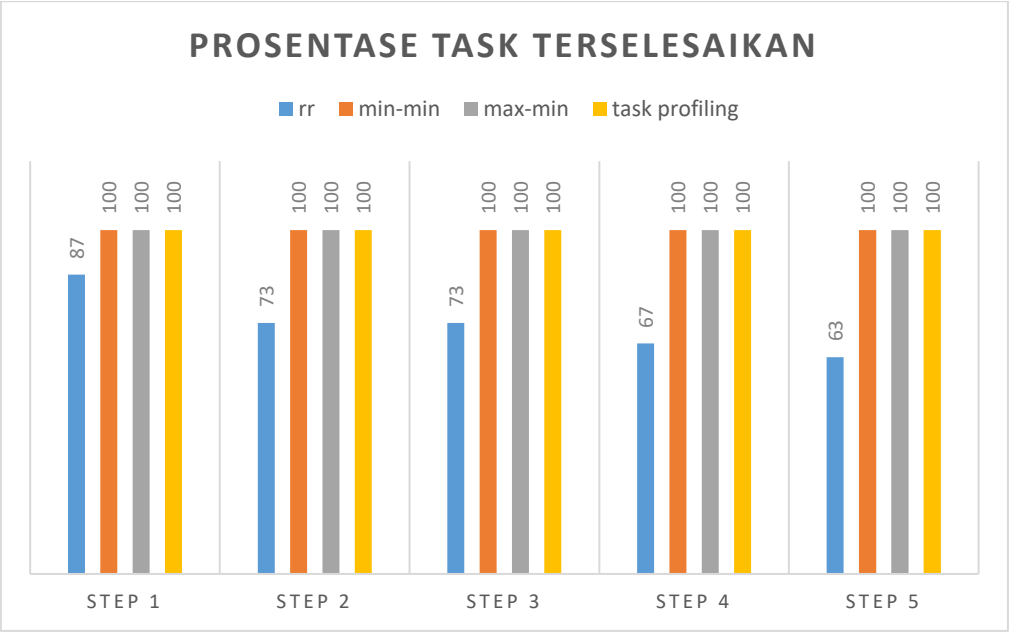


Gambar 4.5 Utilisasi Memori Skenario WSTD



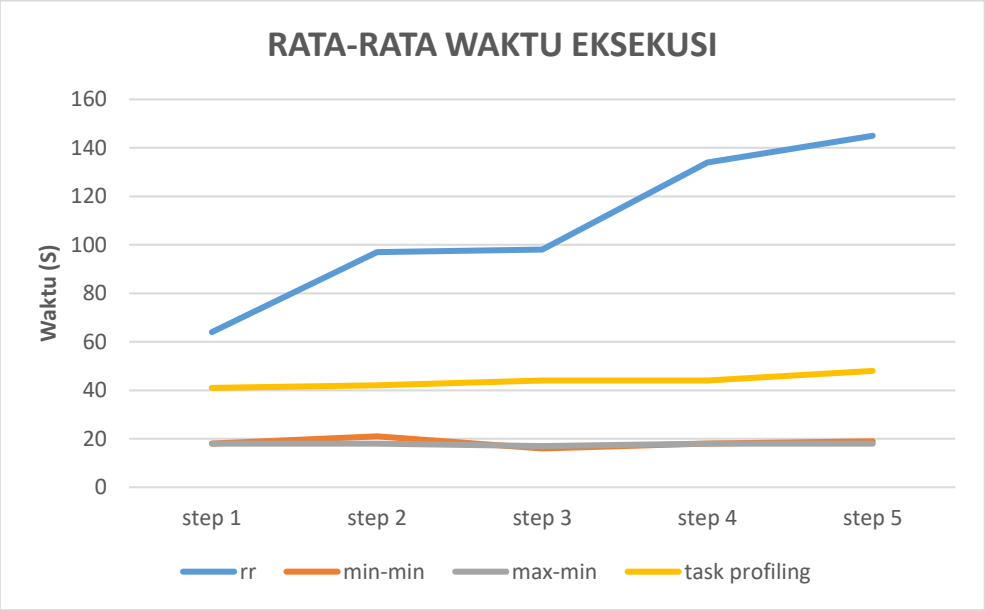
Gambar 4.6 Utilisasi CPU Skenario WSTD

Pada skenario wstd prosentase *task* terselesaikan ditunjukkan pada gambar 4.10

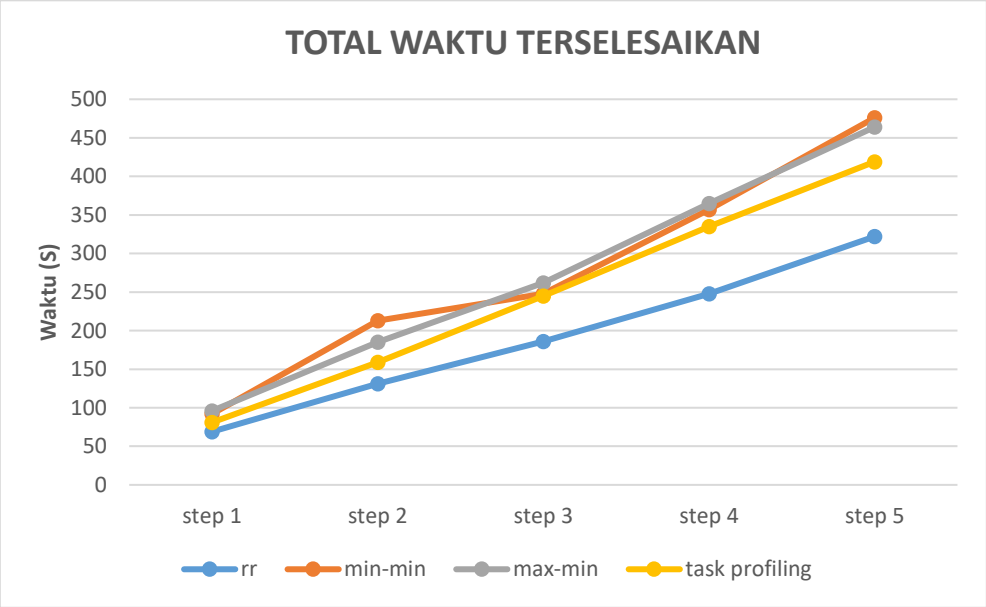


Gambar 4.7 Prosentase Task Terselesaikan Skenario WSTD

Pada skenario wstd waktu eksekusi dan waktu *task* terselesaikan ditunjukkan pada gambar 4.11 dan gambar 4.12



Gambar 4.8 Rata-Rata Waktu Eksekusi Skenario WSTD



Gambar 4.9 Total Waktu Terselesaikan Task Skenario WSTD

C. Worker Berbeda Task Seragam (wdts)

Sesuai dengan tabel 3.4, pengujian dilakukan pada lingkungan *worker* yang berbeda. Setelah scenario dijalankan distribusi beban pada skenario wdts dijelaskan pada gambar 4.13

Tabel 4.5 Distribusi Beban Skenario WDTS

Algoritma Round Robin								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
1	write_high		2	write_high		3	write_high	
4	write_high		5	write_high		6	write_high	
7	write_high		8	write_high		9	write_high	
10	write_high		11	write_high		12	write_high	
13	write_high		14	write_high		15	write_high	
Kucing 4			Kucing 5			Kucing 6		
hostname			hostname			hostname		

Algoritma Min Min								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
2	write_high		3	write_high		1	write_high	
11	write_high		5	write_high		9	write_high	
8	write_high		14	write_high		4	write_high	
			6	write_high		15	write_high	
			12	write_high		10	write_high	
			13	write_high			write_high	
Kucing 4			Kucing 5			Kucing 6		
hostname			hostname			hostname		

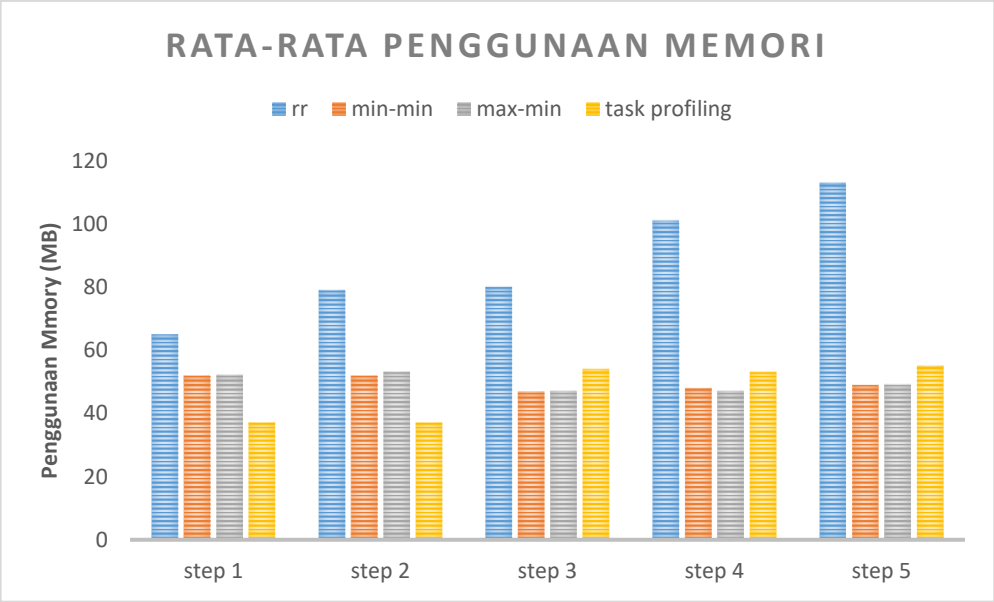
Algoritma Max Min								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
2	write_high		3	write_high		1	write_high	
11	write_high		9	write_high		8	write_high	
5	write_high		10	write_high		4	write_high	
			15	write_high		7	write_high	
			13	write_high		6	write_high	
			14	write_high		12	write_high	
Kucing 4			Kucing 5			Kucing 6		
hostname			hostname			hostname		

Algoritma task profiling								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
1	write_high		6	write_high		2	write_high	
12	write_high		15	write_high		3	write_high	
7	write_high		10	write_high		4	write_high	
						5	write_high	
						8	write_high	
						11	write_high	
						13	write_high	
						14	write_high	
						9	write_high	
Kucing 4			Kucing 5			Kucing 6		
hostname			hostname			hostname		

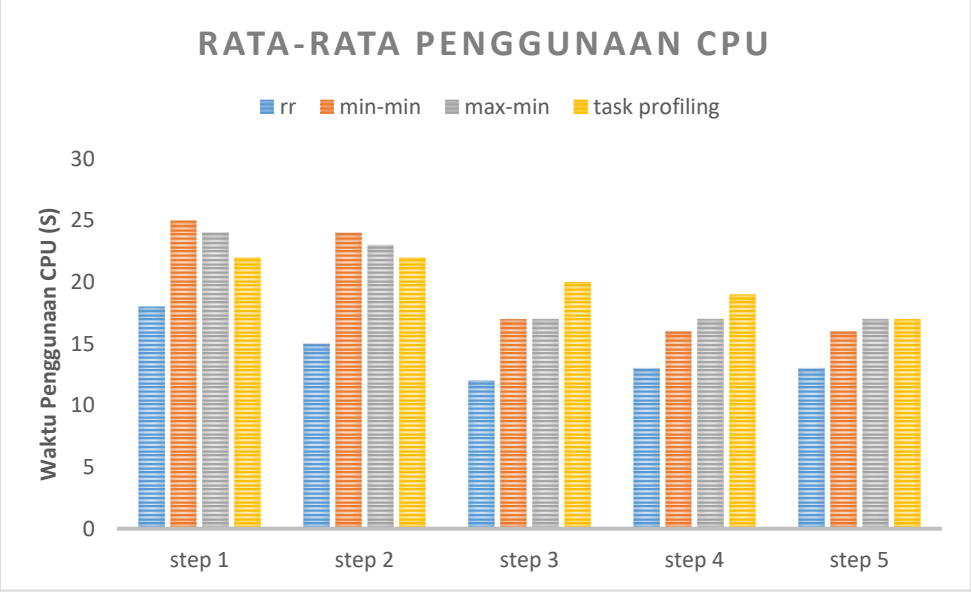
Pada skenario wdts algoritma min-min, max-min, dan *task profiling* mendistribusikan *task* lebih banyak pada *worker* dengan spesifikasi lebih tinggi. Tetapi terdapat perbedaan dengan algoritma *task profiling*. Algoritma *task proiling* mampu menjalankan *task* secara bersamaan berdasarkan hasil profiling.

Worker kucing6 dengan 4 buah prosesor mampu menjalankan maksimal 4 task *write_high* yang termasuk dalam kategori *HIGH_CPU* secara bersamaan.

Pada skenario wdts utilisasi memori dan utilisasi cpu ditunjukkan pada gambar 4.14 dan gambar 4.15



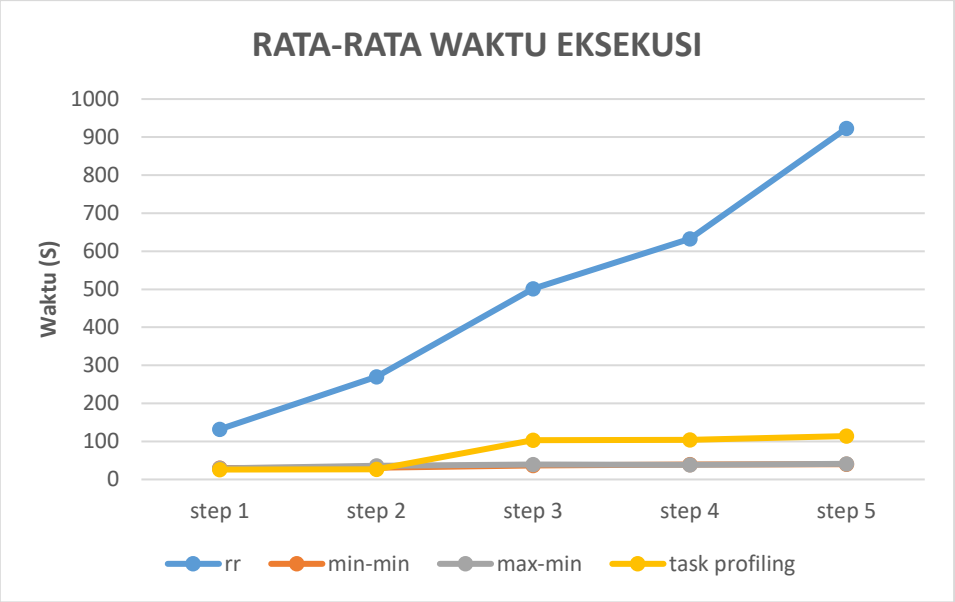
Gambar 4.10 Utilisasi Memori Skenario WDTS



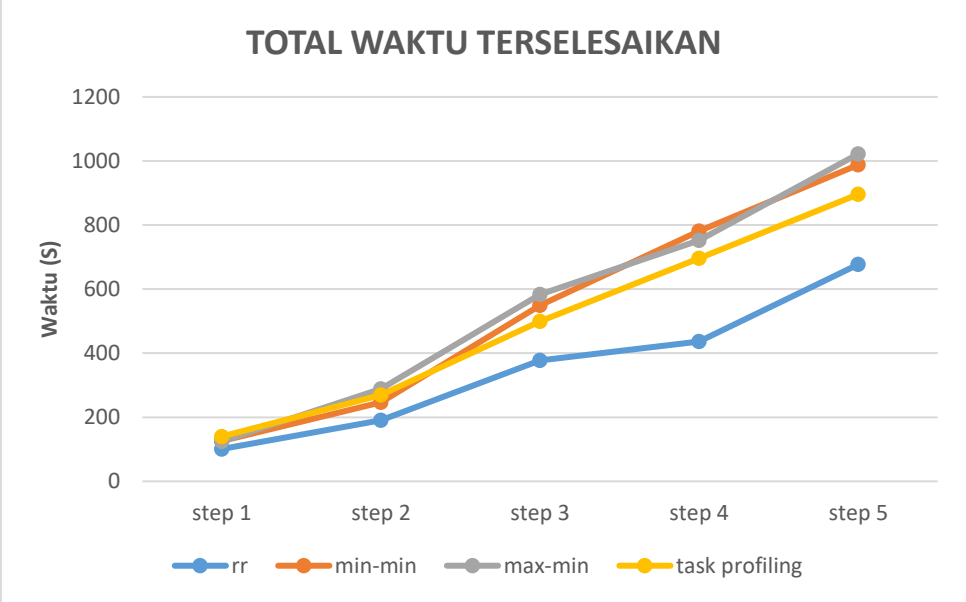
Gambar 4.11 Utilisasi CPU Skenario WDTS

Pada skenario wdts prosentase *task* terselesaikan pada keempat algoritma menunjukkan angka 100% sama halnya pada skenario wsts, hal ini dikarenakan *task* yang dikerjakan adalah *task* yang kebutuhabn sumber dayanya seragam.

Pada skenario wsts waktu eksekusi dan waktu *task* terselesaikan ditunjukkan pada gambar 4.17 dan gambar 4.18



Gambar 4.12 Rata-Rata Waktu Eksekusi Skenario WDTS



Gambar 4.13 Total Waktu Terselesaikan Task Skenario WDTS

D. Worker Berbeda Task Berbeda (wdtd)

Sesuai dengan tabel 3.5, pengujian dilakukan pada lingkungan *worker* yang berbeda dan dengan *task* berbeda. Distribusi beban uji coba skenario wdtd dijelaskan pada gambar 4.19

Tabel 4.6 Distribusi Beban Skenario WDTD

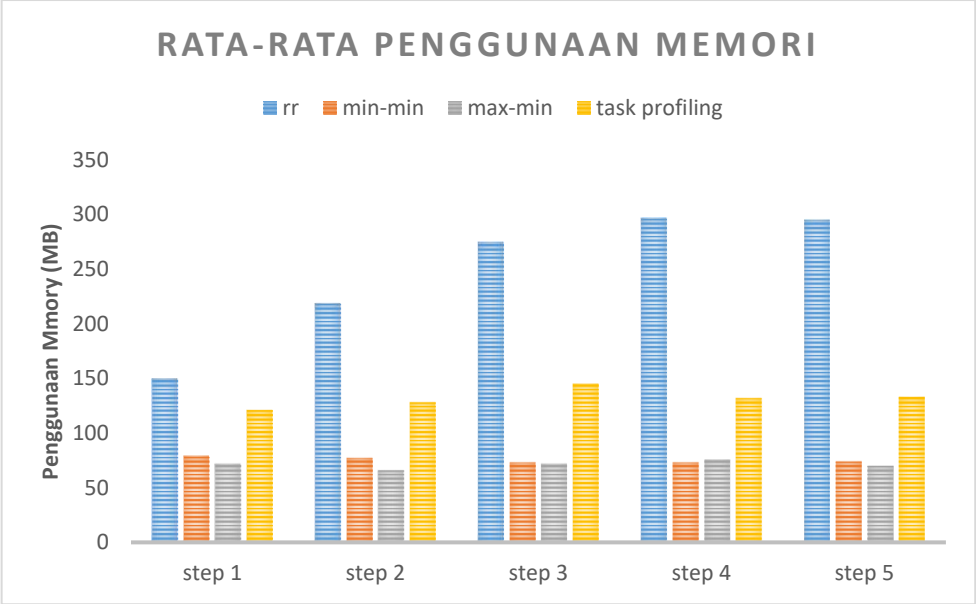
Algoritma Round Robin								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
1	read_high		2	read_moderate		3	read_high	
4	read_moderate		5	read_high		6	write_high	
7	write_high		8	write_high		9	write_high	
10	write_high		11	read_moderate		12	read_high	
13	read_moderate		14	read_moderate		15	read_high	
Kucing 4			Kucing 5			Kucing 6		
hostname			hostname			hostname		

Algoritma min-min								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
9	write_high		2	read_moderate		7	read_high	
6	read_moderate		14	read_high		8	read_high	
13	write_high		5	write_high		3	read_moderate	
15	read_high		4	read_high		1	write_high	
11	read_moderate					10	read_moderate	
12	write_high							
Kucing 4			Kucing 5			Kucing 6		
hostname			hostname			hostname		

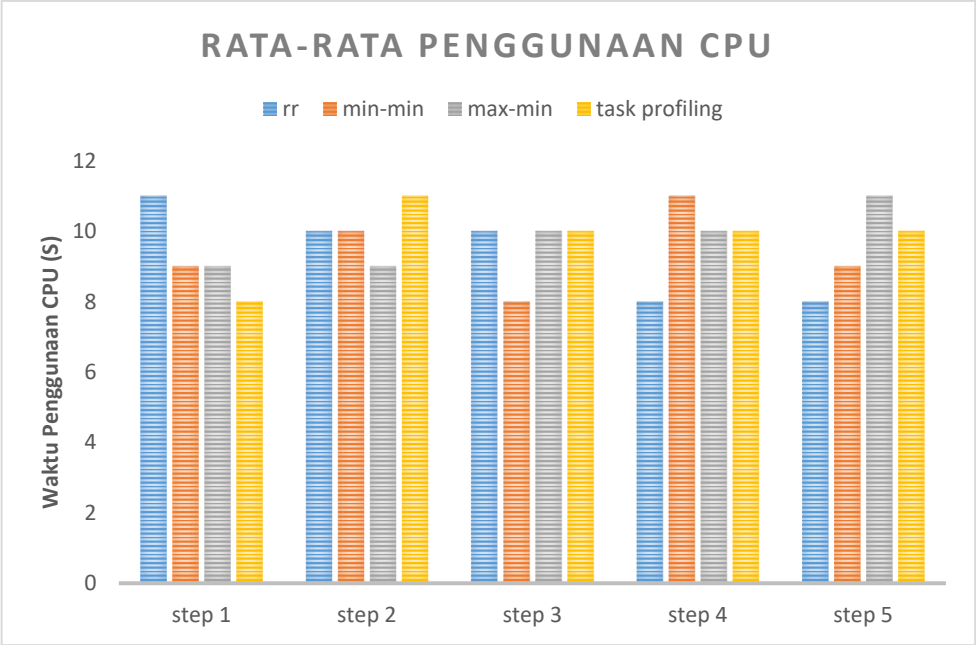
Algoritma max-min								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
3	read_high		1	read_moderate		2	write_high	
5	write_high		14	write_high		8	read_high	
11	read_high		6	read_moderate		9	read_moderate	
15	write_high		13	read_high		7	write_high	
4	read_moderate					10	read_high	
12	read_moderate							
Kucing 4			Kucing 5			Kucing 6		
hostname			hostname			hostname		

Algoritma task profiling								
urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan	urutan task	nama task	waktu pengerjaan
2	write_high		1	read_moderate		6	read_high	
5	write_high		3	read_moderate		12	read_high	
7	write_high		4	read_moderate		15	read_high	
9	write_high		8	read_moderate				
			13	write_high				
			14	read_moderate				
			11	read_high				
			10	read_high				
Kucing 4			Kucing 5			Kucing 6		
hostname			hostname			hostname		

Dengan perubahan lingkungan *worker* kinerja algoritma *task profiling* akan beradaptasi. Pada skenario wdtd utilisasi memori dan utilisasi cpu ditunjukkan pada gambar 4.20 dan gambar 4.21

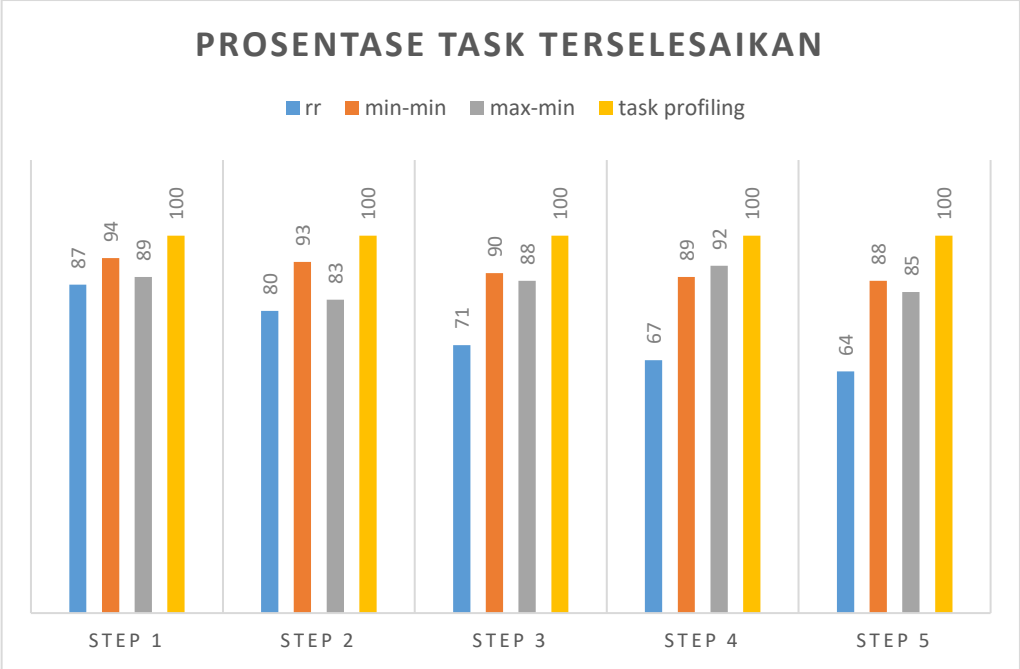


Gambar 4.14 Utilisasi Memori Skenario WDTD



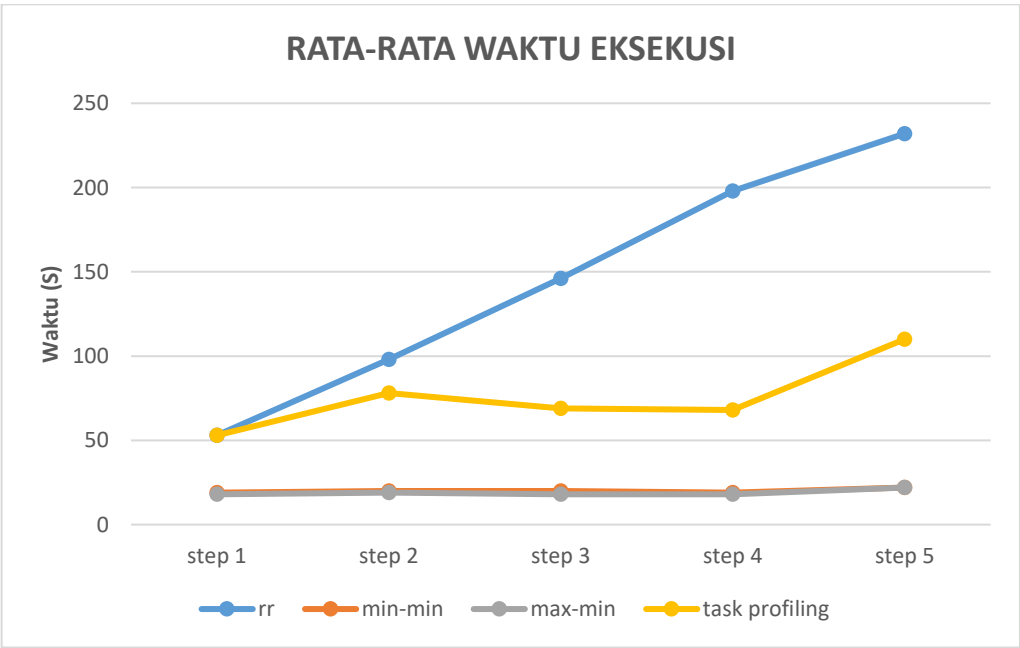
Gambar 4.15 Utilisasi CPU Skenario WDTD

Pada skenario wdtd prosentase *task* terselesaikan ditunjukkan pada gambar 4.22

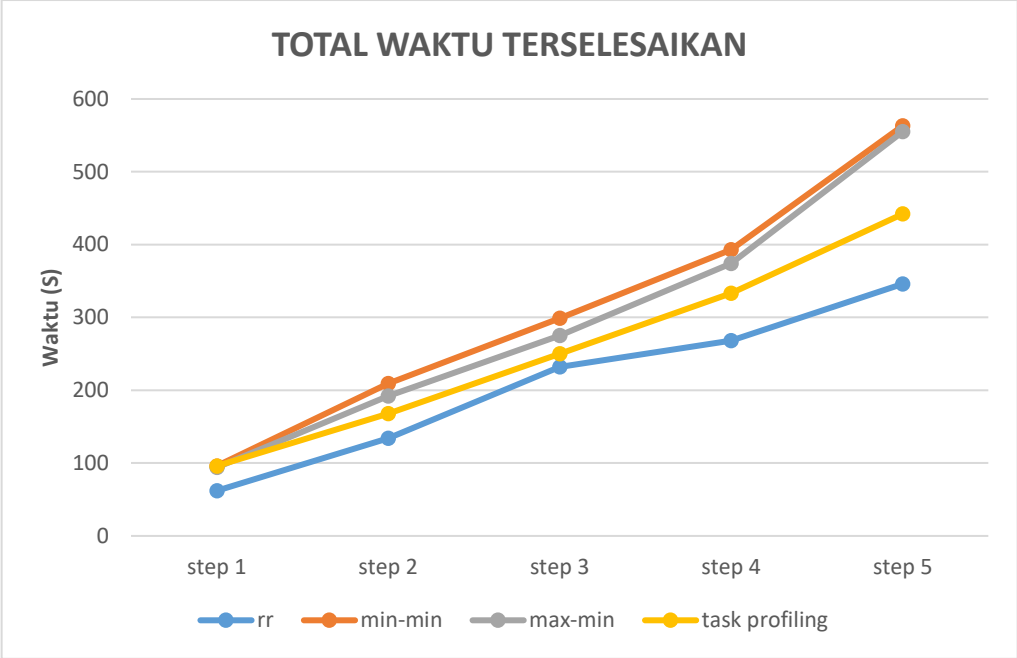


Gambar 4.16 Prosentase Task Terselesaikan Skenario WDTD

Pada skenario wdtd waktu eksekusi dan waktu *task* terselesaikan ditunjukkan pada gambar 4.23 dan gambar 4.24



Gambar 4.17 Rata-Rata Waktu Eksekusi Skenario WDTD



Gambar 4.18 Total Waktu Terselesaikan Task Skenario WDTD

Dari empat skenario diatas didapatkan hasil sebagai berikut:

- Pada keempat skenario utilisasi memori dari algoritma min-min, max-min dan task profiling rata-rata lebih rendah dari pada algoritma roundrobin. Hal ini disebabkan karena algoritma roundrobin menjalankan semua *task* bersamaan, sehinga rata-rata penggunaan memori lebih tinggi. Sedangkan pada utilisasi cpu, algoritma round robin lebih rendah. Hal ini disebabkan karena round robin mengeksekusi semua *task* secara bersamaan, sehingga rata-rata penggunaan waktu cpu lebih rendah. Utiliasi memori dan cpu keempat algoritma tidak berubah secara drastis dan cenderung stabil pada setiap stepnya.
- Pada skenario wsts dan wdts keempat algoritma mampu menyelesaikan semua task, tanpa adanya *task* yang tidak terselesaikan. Karena beban *task* sama sehingga setiap *task* mampu diselesaikan oleh keempat algoritma, sebaliknya pada skenario wstd dan wdtd, *task* yang dijalankan memiliki kebutuhan sumber daya yang berbeda. Sehingga pada algoritma round robin, max-min dan min-min prosentase *task* terselesaikan tidak maksimal.

- Pada skenario wdtd algoritma *task profiling* mampu menyelesaikan semua *task* dengan tingkat keberhasilan maksimal, dibandingkan dengan ketiga algoritma lainnya.
- Pada keempat skenario terlihat rata-rata waktu eksekusi algoritma round robin paling besar dari ketiga algoritma lainnya dan rata-rata waktu eksekusi algoritma *task profiling* lebih besar daripada algoritma min-min dan max-min. Tetapi total waktu terselesaikan algoritma *task profiling* lebih rendah dibandingkan dengan algoritma min-min dan max-min.
- Algoritma *task profiling* bekerja optimal pada skenario wdtd. Karena hasil ujicoba menunjukkan, total waktu terselesaikan dan rata-rata waktu eksekusi algoritma *task profiling* cukup baik dan prosentase keberhasilan *task* maksimal dibandingkan ketiga algoritma lainnya

[Halaman ini sengaja dikosongkan]

BAB 5
KESIMPULAN DAN SARAN

Pada bab ini akan dibahas mengenai kesimpulan yang dapat diambil dari serangkaian penelitian yang telah dilakukan dan saran yang dapat diambil untuk pengembangan penelitian.

5.1 Kesimpulan

Pengujian dan analisis yang telah dilakukan menghasilkan beberapa kesimpulan penelitian, yaitu:

1. Algoritma *task profiling* dapat mengelompokkan *task* berdasarkan kebutuhan sumber daya menggunakan metode *profiling*.
2. Algoritma *task profiling* dapat mendistribusikan *task* berdasarkan kebutuhan sumber daya ke *worker* yang sesuai dan mampu mengerjakannya.
3. Algoritma *task profiling* dapat memilih *worker* berdasarkan ketersediaan sumber daya pada *worker* tersebut, ketika *worker* memiliki sumber daya yang cukup untuk mengerjakan sebuah *task* maka *task* akan dikerjakan oleh *worker* tersebut. Tetapi jika *worker* tidak memiliki sumber daya yang cukup, *task* akan dikerjakan oleh *worker* yang lain.
4. Algoritma task profiling telah di evaluasi menggunakan uji coba pada lingkungan mesin virtual dengan jenis *task* dan spesifikasi *worker* yang beragam. Evaluasi lebih lanjut dilakukan dengan menganalisa rata-rata waktu respon, total waktu terselesaikan dan prosentase keberhasilan *task*.

5.2 Saran

Saran yang dapat diberikan dari hasil uji coba dan evaluasi adalah sebagai berikut:

1. Perlu adanya studi lebih lanjut dalam penentuan batas waktu komputasi tiap *task* yang digunakan untuk menentukan pengelompokan HIGH_CPU pada profiling *task*.

2. Perlu dilakukan studi lebih lanjut pada lingkungan *task* dan *worker* yang berubah secara dinamis pada saat penggunaan algoritma distribusi beban *task profiling* untuk menentukan kapan profiling harus dijalankan kembali.
3. Perlu dilakukan studi lebih lanjut dalam penentuan klasifikasi *task*. Parameter-parameter lain seperti bandwith, I/O dan storage dapat digunakan sebagai dasar penentuan klasifikasi

DAFTAR PUSTAKA

Al-Moayed, A. & Hollunder, B., 2010. *Quality of Service Attributes in Web Services.* Nice, France, s.n., pp. 367-372.

Andelin, J. et al., 1986. *Electronic Record Systems and Individual Chapter 5: Computer Profiling,* Washington, DC: U.S. Government Printing Office,.

Bader, D. A. & Pennington, R., 2001. Cluster Computing: Applications. *The International Journal of High Performance Computing Applications,,* 15(2), pp. 181-185.

Bautista, D. et al., 2010. *Balancing Task Resource Requirements in Embedded Multithreaded Multicore Processors to Reduce Power Consumption.* Pisa, Italy, s.n., pp. 200-204.

Beloglazov, A. & Buyya, R., 2013. Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers under Quality of Service Constraints. *IEEE Transactions on Parallel and Distributed Systems,* 24(7), pp. 1366 - 1379.

Devi, D. C. & Uthariaraj, V. R., 2016. Load Balancing in Cloud Computing Environment Using Improved Weighted Round Robin Algorithm for Nonpreemptive Dependent Tasks. *Hindawi Publishing Corporation,* pp. 1-14.

Diep, M., Cohen, M. & Elbaum, S., 2006. *Probe Distribution Techniques to Profile Events in Deployed Software.* Raleigh, NC, USA, s.n.

Eklov, D., Nikoleris, N. & Hagersten, E., 2014. *A Software Based Profiling Method for Obtaining Speedup Stacks on Commodity Multi-Cores.* Monterey, CA, USA, 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).

Jena, S. R. & Ahmad, Z., 2013. Response Time Minimization of Different Load Balancing Algorithms in Cloud Computing Environment. *International Journal of Computer Applications,* Volume 69, pp. 22-27.

Kaur, R. & Luthra, P., 2014. Load Balancing in Cloud System using Max Min and Min. *International Journal of Computer Applications,* pp. 31-34.

Kumari, P. & Saxena, M., 2016. A Round-Robin based Load balancing approach for Scalable demands and maximized Resource availability. *International Journal Of Engineering And Computer Science,* 5(8), pp. 17375-17380.

Nah, F. F.-H., 2003. *A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait?*. Tampa, FL, USA, s.n., pp. 1-37.

Neelakantan, P., 2012. Load Balancing in Distributed Systems using Diffusion Technique. *International Journal of Computer Applications*, 39(4), pp. 1-7.

Reda, N. M., 2015. An Improved Sufferage Meta-Task Scheduling Algorithm in Grid Computing Systems. *International Journal of Advanced Research*, 3(10), pp. 123 - 129.

Sabiya, J. S., 2016. Weighted Round-Robin Load Balancing Using Software Defined Networking. *International Journal of Advanced Research in Computer Science and Software Engineering*, 6(6), pp. 621-625.

Salchow, ., 2015. *Load Balancing 101: Nuts and Bolts*, s.l.: s.n.

Sharma, A. & Peddoju, S. K., 2014. *Response Time Based Load Balancing in Cloud Computing*. Kanyakumari District, India, IEEE, pp. 1287-1293.

Soni, A., Vishwakarma, G. & Jain, Y. K., 2015. A Bee Colony based Multi-Objective Load Balancing Technique for Cloud Computing Environment. *International Journal of Computer Applications*, 114(4), pp. 19-25.

Walker, R., 2016. *Examining Load Average*, s.l.: Linux Journal.

Zalavadiya, K. & Vaghela, D., 2016. Honey Bee Behavior Load Balancing of Tasks in Cloud Computing. *International Journal of Computer Applications*, 139(1), pp. 16-19.

BIODATA PENULIS

Thiar Hasbiya Ditanaya, Akrab dipanggil Thiar lahir di Muara Bulian pada tanggal 14 maret 1994. Penulis adalah anak pertama dari dua bersaudara. Penulis menempuh pendidikan formal di SDN Wates III Mojokerto, SMPN 2 Mojokerto, SMAN 1 Sooko Mojokerto dan S1 Teknik Informatika FTIF ITS. Penulis adalah pengembang aplikasi berbasis web. Sebagai seorang pengembang, penulis juga harus selalu berkembang. Beberapa aplikasi berbasis web yang pernah di kembangkan penulis adalah web penerimaan peserta didik baru sidoarjo dan web kenaikan pangkat online dinas pendidikan surabaya. Penulis adalah asisten Lab pada Laboratorium Arsitektur dan Jaringan Komputer. Selama menjadi asisten Lab penulis beberapa kali menjadi asisten dosen dan praktikum pada mata kuliah sistem operasi, jaringan komputer, keamanan informasi jaringan dan komputasi awan. Penulis senang mempelajari hal baru terkait teknologi informasi. Beberapa topik yang sedang di dalami adalah topik terkait Aplikasi Berbasis Web, Komputasi Awan, dan Teknologi Pada Jaringan. Penulis dapat dihubungi melalui surat elektronik thiar.hasbiya@gmail.com