KINETIK, Vol. 2, No. 2, Mei 2017, Hal. 89-98
ISSN : 2503-2259
E-ISSN : 2503-22677

Perbaikan Mekanisme Load Balancing untuk Komputasi Klaster pada Kondisi Dinamis

Mohammad Zarkasi*[1], Waskitho Wibisono[2]
[1,2]Institut Teknologi Sepuluh Nopember Surabaya
zarkasi09@mhs.if.its.ac.id[*1], waswib@if.its.ac.id[2]

Abstrak

Salah satu permasalahan yang sering terjadi pada lingkungan komputasi klaster adalah terjadinya beban yang tidak seimbang yang dapat menurunkan Quality of Service (QoS). Oleh sebab itu, dibutuhkan metode load balancing yang handal dalam pendistribusian beban. Pada beberapa kasus, metode load balancing dinamis gagal berperan sebagai metode load balancing yang optimal jika lingkungan implementasi tidak sesuai dengan lingkungan yang diasumsikan saat metode tersebut dikembangkan. Dalam penelitian ini diusulkan metode load balancing adaptif dengan menggunakan algoritma reinforcement learning (RL) yang mampu beradaptasi terhadap perubahan lingkungan. Hasil pengujian menunjukkan bahwa kinerja dari metode load balancing yang diusulkan lebih efisien dibandingkan dengan metode load balancing dinamis baik pada kondisi lingkungan tidak mengalami beban dengan peningkatan total waktu eksekusi sebesar 45% maupun pada saat lingkungan mengalami beban yang tidak seimbang dengan peningkatan waktu eksekusi sebesar 21%.

Kata kunci: *Job, Load Balancing Adaptif, Load Balancing Dinamis, Task, Waktu Eksekusi*

Abstract

One of the problems that often occur in cluster computing environment is the unbalanced load which can lower the Quality of Service (QoS). Therefore, it needs a reliable load balancing method for load distribution. In some cases, dynamic load balancing methods fail to give optimal result if the environment of implementation is different compared to assumed environment when the method were developed. In this research, an adaptive load balancing method is proposed using reinforcement learning (RL) algorithm which is able to adapt to environmental changes. The result from experiments show that the performance of proposed load balancing method is more efficient that dynamic load balancing both when the environment condition is not subjected to loads with improvement of total execution time by 45% and when the environment condition are subjected to unbalanced loads with improvement of total execution time by 21%.

Keywords: *Adaptive Load Balancing, Dynamic Load Balancing, Execution Time, Job, Task*

1. Pendahuluan

Adanya teknologi jaringan komunikasi memungkinkan dua komputer untuk saling terhubung satu sama lain. Dengan semakin tingginya kecepatan pengiriman yang dapat dilakukan oleh teknologi komunikasi saat ini memungkinkan komputer-komputer untuk saling berbagi sumber daya, seperti CPU, *memory*, dan media penyimpanan [1]. Selain itu, permasalahan yang perlu diselesaikan telah menjadi lebih kompleks dan dengan skala yang lebih besar untuk dikerjakan oleh sebuah komputer tunggal [2]. Oleh karena itu, diperlukan kolaborasi dari beberapa komputer yang saling berbagi sumber daya untuk menyediakan aplikasi yang lebih unggul dibandingkan sistem tunggal.

Komputasi klaster adalah salah satu aplikasi yang memanfaatkan kemajuan di bidang jaringan komunikasi. Komputasi klaster terdiri dari *client*, *server* dan *node*. *Client* berperan sebagai bagian yang membuat *job* yang terdiri dari beberapa *task*, *server* berperan menerima *job* dari *client*, mendistribusikan *task* ke *node* dan mengirimkan hasil kembali ke *client*, dan *node* berperan sebagai pengeksekusian *task*. Tidak jarang di antara *node-node* yang tergabung dalam sistem komputasi klaster mengalami beban yang tidak seimbang yang dapat menurunkan *Quality*

Zarkasi, M. (2017). Perbaikan Mekanisme Load Balancing untuk Komputasi Klaster pada Kondisi Dinamis. KINETIK, 2(2), 89-98. doi:http://dx.doi.org/10.22219/kinetik.v2i2.154
Makalah dikirim 25 Januari 2017; Revisi 22 Maret 2017; Diterima 6 April 2017

of Service (QoS). Untuk menjaga nilai QoS, maka diperlukan metode Load Balancing yang handal untuk mendistribusikan beban.

Pada [3] dan [4], berdasarkan tipe informasi yang digunakan untuk pengambilan keputusan, metode Load Balancing dibagi menjadi empat kategori, yaitu metode Statis, metode Dinamis, metode Hybrid dan metode Adaptif. Metode Load Balancing Statis berjalan dengan melakukan perhitungan porsi beban untuk tiap-tiap *node* hanya saat *job-job* akan dieksekusi seperti yang ditunjukkan pada [5]. Metode dinamis berjalan dengan melakukan perhitungan porsi beban untuk tiap-tiap *node* selama *job-job* sedang dieksekusi oleh sistem. Metode Hybrid merupakan metode yang menggabungkan antara metode Statis dan metode Dinamis. Sedangkan, metode Adaptif mendistribusikan *task* dalam kuantitas yang berubah-ubah yang dikarenakan hasil adaptasi yang dilakukan terhadap kondisi lingkungan.

Metode Load Balancing Dinamis unggul atas metode Statis apabila terdapat *node* yang mengalami penurunan kemampuan untuk memberikan sumber daya komputasi selama proses eksekusi berjalan. Namun, metode Dinamis gagal berperan sebagai metode Load Balancing yang bagus jika lingkungan implementasi tidak sesuai dengan lingkungan yang diasumsikan saat metode tersebut dikembangkan. Hal ini menjadikan metode Adaptif sebagai solusi untuk metode Load Balancing yang mampu beradaptasi terhadap perubahan lingkungan [4] [6] dan [7] menunjukkan keunggulan metode Adaptif sebagai metode Load Balancing dan *routing* yang ditujukan pada lingkungan yang mengalami tingkat perubahan yang sangat tinggi dari waktu ke waktu, seperti pada lingkungan *cloud* dan *wireless sensor network* (WSN)

Mohammadpour [2] melakukan penelitian metode Distribusi Dinamis dengan mengelompokkan sebuah *task* ke dalam kategori CPU *bound*, *memory bound,* dan *network bound* berdasarkan frekuensi *task-task* yang telah dieksekusi. Wu [8] mengusulkan penggunaan *multi-agent reinforcement learning* (RL) sebagai metode Load Balancing Adaptif pada lingkungan komputasi *grid* berdasarkan pada tabel utilitas yang dipelajari secara *online* untuk mengestimasi efisiensi sumber daya komputasi. Wirth [9] mengusulkan penggunaan metode Monte Carlo untuk menyederhanakan permasalahan *preference-based reinforcement learning* (PBRL) sehingga dapat meniadakan kebutuhan terhadap *parameterizable policies*.

Reinforcement learning (RL) berawal dari suatu gagasan sederhana tentang agen yang menginginkan sesuatu, yang menyesuaikan tingkah lakunya untuk memaksimalkan sinyal khusus dari lingkungannya [10]. Tidak seperti *machine learning* pada umumnya, agen RL tidak diberikan informasi *action* yang harus dilakukan, tetapi harus menermukan sendiri *action* yang menghasilkan *reward* tertinggi dengan cara mencobanya. Tantangan yang muncul di dalam RL dan tidak pada *machine learning* yang lain adalah adanya *trade-off* antara eksplorasi dan eksploitasi. Untuk mendapatkan hasil yang baik, agen harus melakukan eksploitasi dengan memilih *action* pada suatu *state* yang memiliki *reward* yang tinggi. Tetapi, untuk menemukan *action* seperti itu, agen harus mencoba *action* pada suatu *state* secara acak agar memiliki pengetahuan *action* yang lebih luas.

Pada penelitian ini diusulkan metode Adaptif Load Balancing menggunakan RL untuk komputasi klaster pada kondisi *node* mengalami beban yang tidak seimbang. *State* RL didasarkan pada kuantitas *task* yang diberikan pada eksekusi sebelumnya. Sedangkan, *reward* didasarkan pada perbandingan antara waktu eksekusi lokal terhadap rata-rata waktu eksekusi global.

2. Metode Penelitian

Metode Load Balancing yang diusulkan pada penelitian ini adalah menggunakan algoritma RL sebagai agen pembelajaran. Agen mengubah dan menyesuaikan jumlah *task* yang didistribusikan ke suatu *node* agar diperoleh beban yang seimbang. Gambaran umum metode Load Balancing yang diusulkan ditunjukkan pada Gambar 1.

Proses metode Load Balancing dimulai ketika *server* menerima hasil eksekusi *task* dari *node*. Selain hasil eksekusi, disertakan pula *feedback* yang berupa rangkuman berjalannya proses eksekusi di *node*, waktu yang digunakan untuk proses eksekusi dan waktu yang digunakan untuk proses pengiriman *task* dan hasil eksekusi.

Selanjutnya, *feedback* tersebut dialihkan ke agen RL. Proses perhitungan *reward* ini ditunjukkan pada Gambar . Setiap hasil eksekusi dan *feedback* yang diterima merupakan hasil dari *action* a_t yang diambil pada *state* s_t pada proses sebelumnya. Agen menghitung nilai *reward r* berdasarkan pada rasio total waktu eksekusi pada suatu *node* terhadap rata-rata total waktu eksekusi global. Jika total waktu eksekusi pada suatu *node* lebih kecil daripada rata-rata total

waktu eksekusi global, maka agen memberikan *reward* positif, dan apabila total waktu eksekusi pada suatu *node* lebih besar daripada rata-rata total waktu eksekusi global, maka agen memberikan *reward* negatif. Nilai *reward* yang didapatkan kemudian disimpan pada *state-action* yang bersesuaian $(s_t, a_t) = r$.

Setelah selesai menghitung *reward*, agen melanjutkan ke tahap selanjutnya, yaitu menentukan *state* s_{t+1} berdasarkan *feedback* yang diterima seperti yang ditunjukkan pada Gambar 3. Pada penelitian ini, *state* direpresentasikan dengan rasio jumlah *task* yang disimpan oleh *feedback* terhadap jumlah maksimal *task*. Jumlah maksimal *task* merupakan total jumlah *task* yang terdapat di dalam suatu *job*. Proses penentuan *state* s_{t+1} ini ditunjukkan pada Gambar 3. Seperti yang ditunjukkan pada Gambar 3, terdapat 10 kemungkinan *state* yang dihasilkan. Misalkan, *state 0* jika jumlah *task* sebelumnya adalah berkisar antara $0 \leq n \leq \frac{TotTask}{10}$, *state 1* jika jumlah *task* sebelumnya adalah berkisar antara $\frac{TotTask}{10} \leq n \leq \frac{2*TotTask}{10}$, dan seterusnya.

Setelah proses penentuan *state* s_{t+1} dilakukan, selanjutnya adalah memilih *action* terbaik untuk *state* tersebut. Proses pemilihan *action* ini menggunakan algoritma ε-*greedy* yang di tunjukkan pada Gambar 4. Pada proses ini terdapat *trade off* antara melakukan eksploitasi *action* terbaik atau melakukan eksplorasi *action* secara acak. Eksploitasi bertujuan untuk memaksimalkan total *reward* yang diharapkan dengan memilih *action* yang memiliki catatan *reward* yang tinggi. Sedangkan, eksplorasi bertujuan untuk menemukan *action-action* yang memiliki kemungkinan *reward* yang lebih baik dibandingkan dengan *action-action* yang pernah diambil pada proses sebelumnya. Suatu angka dipilih secara acak. Apabila angka tersebut lebih besar daripada nilai *epsilon*, maka agen melakukan proses eksploitasi *action-action* terbaik. Sebaliknya, agen melakukan proses eksplorasi untuk mencari kemungkinan *action* yang lebih baik. *Action* yang dipilih merepresentasikan nilai perubahan jumlah *task* yang akan dikirimkan ke *node* untuk dieksekusi.

Untuk menguji kinerja metode Load Balancing yang diusulkan, dilakukan pengujian dengan mengimplementasikan dengan topologi yang ditunjukkan pada Gambar . Pengujian dilakukan menggunakan sebuah komputer dengan CPU Intel Core i3 1,8 GHz, *memory* 6 GB menggunakan sistem operasi Ubuntu 16.04. Pengujian dilakukan menggunakan teknologi *container* yang berjalan pada sistem operasi Linux. *container* yang digunakan adalah Docker v1.12. *Framework* komputasi klaster yang digunakan sebagai pengujian adalah *Java Parallel Processing Framework* (JPPF) yang dibangun diatas bahasa Java.

Pengujian menggunakan sebuah *client*, sebuah *server* dan empat buah *node*. Masing-masing elemen dijalankan dengan batasan hanya dapat menggunakan 50% sumber daya CPU. *Client* dijalankan hanya menggunakan sumber daya pada *core 0*, *server* dijalankan hanya menggunakan sumber daya pada *core 1* dan dua *node* pertama dijalankan hanya menggunakan sumber daya pada *core 2* dan dua *node* terakhir dijalankan hanya menggunakan sumber daya pada *core 3*.

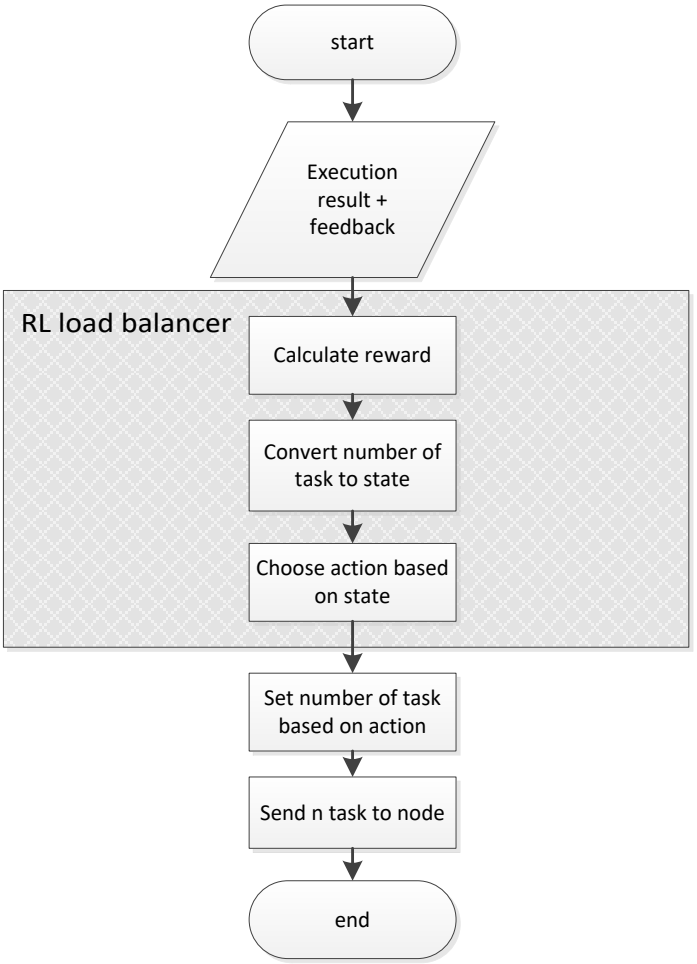
Pengujian dilakukan dengan dua skenario. Skenario pertama dilakukan pada kondisi normal, yaitu *node-node* yang tergabung ke dalam sistem tidak mengalami beban CPU dan *throughput* jaringan sebesar 10 Mbps untuk semua *node*. Pada skenario kedua dilakukan pada kondisi dinamis, yaitu salah satu *node* mengalami beban CPU yang berubah-ubah dan *throughput* jaringan yang berubah-ubah. Kasus uji yang digunakan sebagai *job* yang akan diselesaikan dengan komputasi klaster adalah proses deskripsi *cipheredtext* MD5 menjadi *plaintext*. Tiap *job* berisi 100 *cipheredtext* dan dipecah menjadi 100 *task* untuk didistribusikan ke masing-masing *node.* Total jumlah *job* yang digunakan pada masing-masing skenario adalah sebanyak 25.000 *job*.

Untuk memberikan beban CPU pada suatu *node*, diberikan beban komputasi sebesar 0% dan 50% yang dijalankan secara bergantian masing-masing selama satu menit. Untuk memberikan beban CPU sebesar 50%, sebuah proses dekripsi *cipheredtext* MD5 menjadi *plaintext* dijalankan secara lokal pada suatu *node*.

Untuk memberikan beban pada jaringan pada suatu *node*, *throughput* jaringan pada suatu *node* diatur pada 10 Mbps dan 2 Mbps yang diterapkan masing-masing selama satu menit secara bergantian.

Uji coba dilakukan dengan membandingkan kinerja mekanisme Load Balancing Adaptif yang diusulkan dengan mekanisme Load Balancing Dinamis yang terdapat di dalam JPPF. Untuk mengukur kinerja, digunakan tolok ukur sebagai berikut:

1. Rata-rata waktu eksekusi per *job*, merupakan rata-rata waktu yang diperlukan untuk menyelesaikan suatu *job*.
2. Total waktu eksekusi, merupakan total waktu yang digunakan untuk menyelesaikan seluruh *job* yang diberikan.



Gambar 1. Diagram Metode Load Balancing yang Diusulkan

```
algorithm calculate_reward is
    input:  feedback with total execution time T
    output: reward

    count ← 0

    for each node in connected_node do
        total_global_time += total_execution_time in node
        increment count

    avg_global_time ← total_global_time / count

    if T < global_time then
        reward +1
    else
        reward -1
    end

    return reward
```

Gambar 2. Pseudocode Perhitungan Reward

```
algorithm get_state is
    input:  feedback with number of previous task nTask
    output: state s

    s ← 10 * nTask / total_task_in_job

    return s
```

Gambar 3. Pseudocode Penentuan State Berdasarkan Feedback Jumlah Task

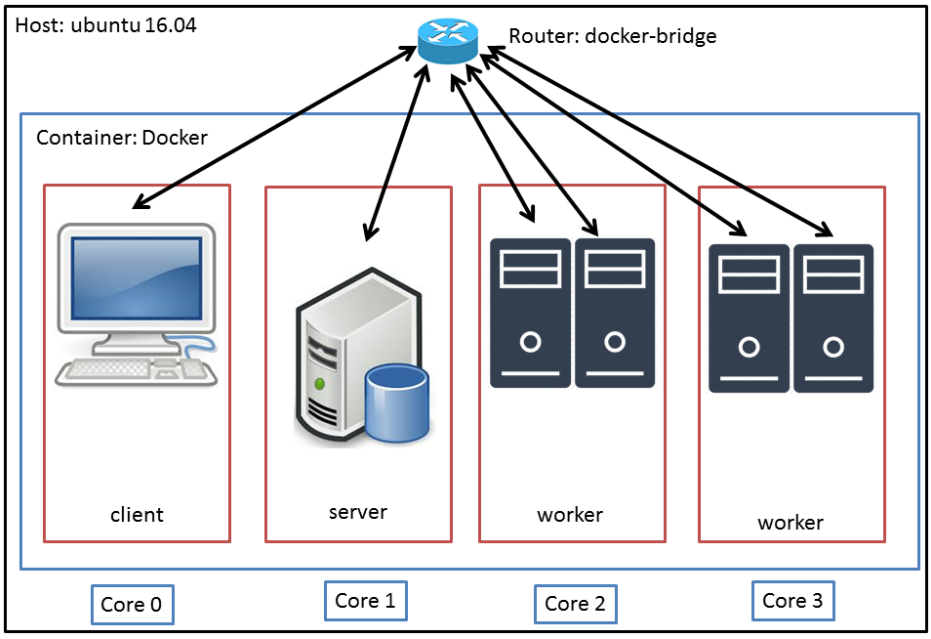
```
algorithm choose_action is
    input:  state s
    output: action a

    rr <-- pick random number

    if rr > epsilon then
        choose best action a on state s
    else
        choose random action
    end

    return a
```

Gambar 4. Psedocode Pemilihan Action Menggunakan Aalgoritma ε-greedy



Gambar 5. Topologi Pengujian

3. Hasil Penelitian dan Pembahasan

Pengujian dilakukan dengan dua skenario. Pada masing-masing skenario, diuji dengan 25.000 *job* dengan metode Load Balancing Dinamis dan metode Load Balancing Adaptif. Skenario pertama dijalankan dengan kondisi lingkungan normal, yaitu seluruh *node* diatur dengan *throughput* jaringan 10 Mbps dan beban CPU sebesar 0%. Sedangkan, skenario kedua dijalankan dengan kondisi lingkungan dinamis, yaitu salah satu *node* mendapat *throughput* yang berubah-ubah antara 10 Mbps dan 2 Mbps secara bergantian masing-masing selama satu menit dan mendapat beban CPU sebesar 0% dan 50% secara bergantian masing-masing selama satu menit.

3.1 Skenario 1

Pada skenario ini, lingkungan dalam kondisi normal, yaitu seluruh *node* diatur memiliki *throughput* sebesar 10 Mbps dan beban CPU sebesar 0%. Metode Load Balancing Dinamis dan metode Load Balancing Adaptif masing-masing dijalankan dengan 25.000 *job*.

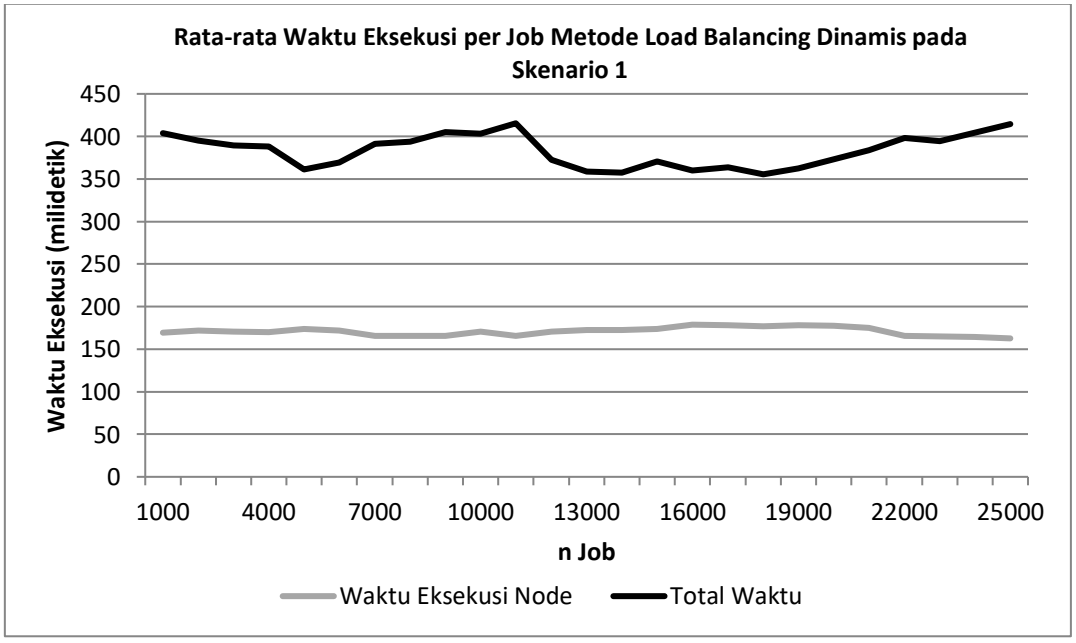
Gambar menunjukkan rata-rata waktu eksekusi per *job* pada skenario pertama. Gambar 6(a) menunjukkan rata-rata waktu eksekusi per *job* menggunakan metode Load Balancing Dinamis. Rata-rata waktu eksekusi yang berjalan pada sisi *node* menggunakan metode *load balancing* dinamis adalah sekitar 180 milidetik. Sedangkan, secara keseluruhan, rata-rata waktu eksekusi per *job* yang menggunakan metode Load Balancing Dinamis adalah sekitar 383 milidetik.

Sedangkan, pada hasil uji coba menggunakan metode Load Balancing Adaptif yang ditunjukkan pada Gambar 6(b), rata-rata waktu eksekusi yang berjalan pada sisi *node* berkisar antara 180 milidetik. Jadi, secara keseluruhan rata-rata waktu eksekusi per *job* yang menggunakan metode Load Balancing Adaptif berkisar antara 208 milidetik.

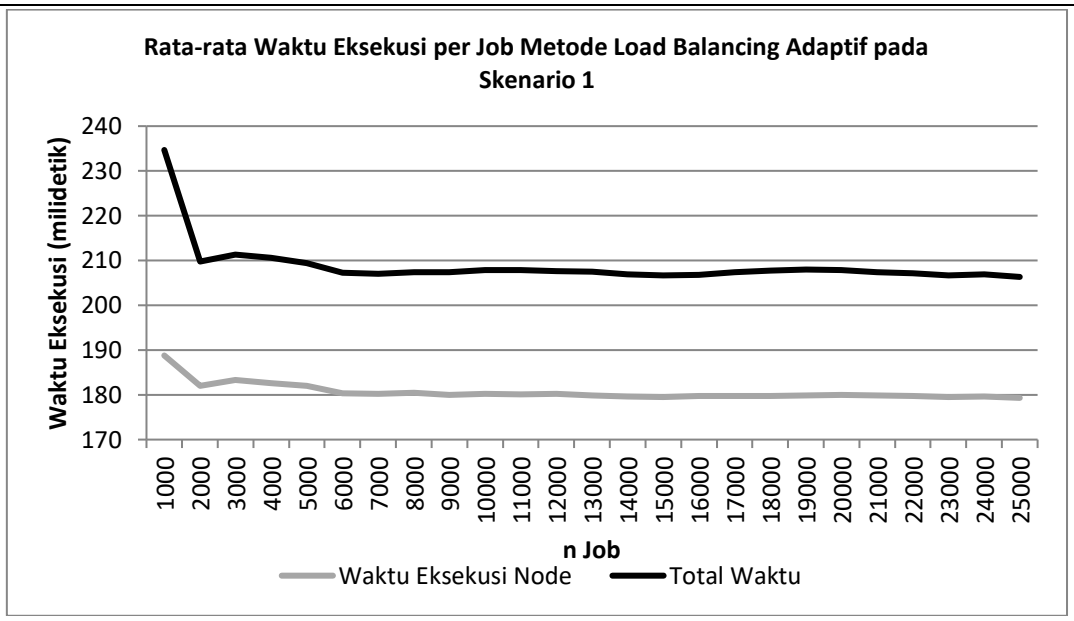
Dilihat dari segi rata-rata waktu eksekusi secara keseluruhan, metode Load Balancing Adaptif memberikan waktu rata-rata yang lebih kecil. Hal ini membuktikan bahwa metode Load Balancing yang diusulkan terbukti lebih baik dibandingkan dengan metode Load Balancing Dinamis pada kondisi normal dengan seluruh *node* mendapatkan *throughput* jaringan sebesar 10 Mbps dan beban CPU 0%.

Pada hasil uji coba menggunakan metode Load Balancing Dinamis yang ditunjukkan pada Gambar 6(a), terdapat perbedaan sekitar 212 milidetik antara rata-rata waktu eksekusi pada *node* dan rata-rata waktu eksekusi secara keseluruhan. Sedangkan pada hasil uji coba menggunakan metode Load Balancing Adaptif yang ditunjukkan pada Gambar 6(b), terdapat perbedaan sekitar 28 milidetik antara rata-rata waktu eksekusi pada *node* dan rata-rata waktu eksekusi secara keseluruhan. Hal ini menunjukkan metode Load Balancing Adaptif melakukan penggunaan sumber daya jaringan secara lebih efisien dibandingkan dengan metode Load Balancing Dinamis pada kondisi normal.

Selain perbedaan antara rata-rata waktu eksekusi pada *node* dan rata-rata waktu eksekusi per *job* secara keseluruhan, dari hasil uji coba pada skenario pertama didapatkan data adanya *node idle*, yaitu *node* yang tidak mendapatkan *task* pada eksekusi suatu *job*. Hal ini menyebabkan waktu eksekusi menjadi lebih tinggi karena dikerjakan oleh sumber daya yang lebih sedikit. Data frekuensi *node idle* ditunjukkan pada Tabel 1. Pada uji coba menggunakan metode Load Balancing Dinamis terjadi 5.034 kemunculan *node idle* selama proses eksekusi 25.000 *job*. Sedangkan pada uji coba menggunakan metode Load Balancing Adaptif, terjadi 59 kemunculan *node idle* selama proses eksekusi 25.000 *job*.



(a)



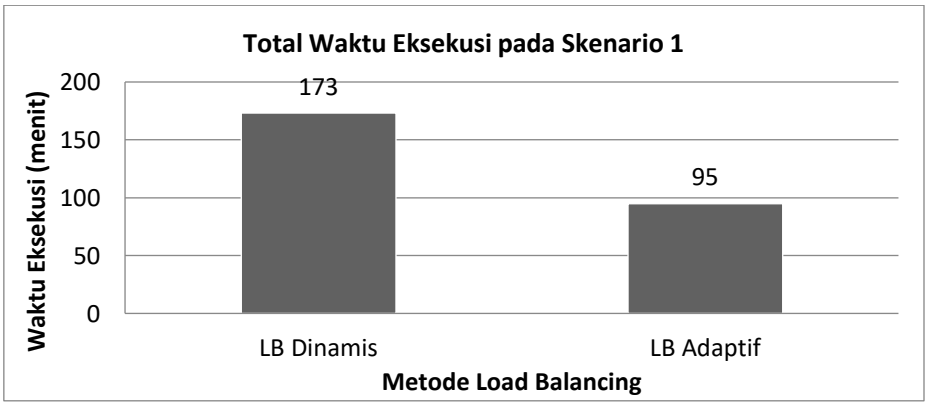
(b)

Gambar 6 (a)(b). Grafik Rata-rata Waktu Eksekusi Per Job Pada Skenario 1

Tabel 1. Tabel Frekuensi Node Idle Pada Skenario 1

Metode *Load Balancing*	Frekuensi *node idle*
Dinamis	5.034
Adaptif	59

Dari sisi total waktu yang digunakan untuk mengeksekusi seluruh 25.000 *job*, metode Load Balancing lebih unggul dibandingkan metode Load Balancing Dinamis seperti yang ditunjukkan pada Gambar 7. Pada uji coba menggunakan metode Load Balancing Dinamis, total waktu yang digunakan untuk mengeksekusi seluruh 25.000 *job* adalah selama 173 menit. Sedangkan, pada uji coba menggunakan metode Load Balancing Adaptif, total waktu yang digunakan untuk mengeksekusi seluruh 25.000 *job* adalah selama 95 menit. Total waktu yang digunakan untuk mengeksekusi 25.000 *job* dengan menggunakan metode Load Balancing Adaptif pada skenario pertama adalah 45% lebih cepat dibandingkan dengan menggunakan metode Load Balancing Dinamis.



Gambar 7. Grafik Total Waktu Eksekusi Pada Skenario 1

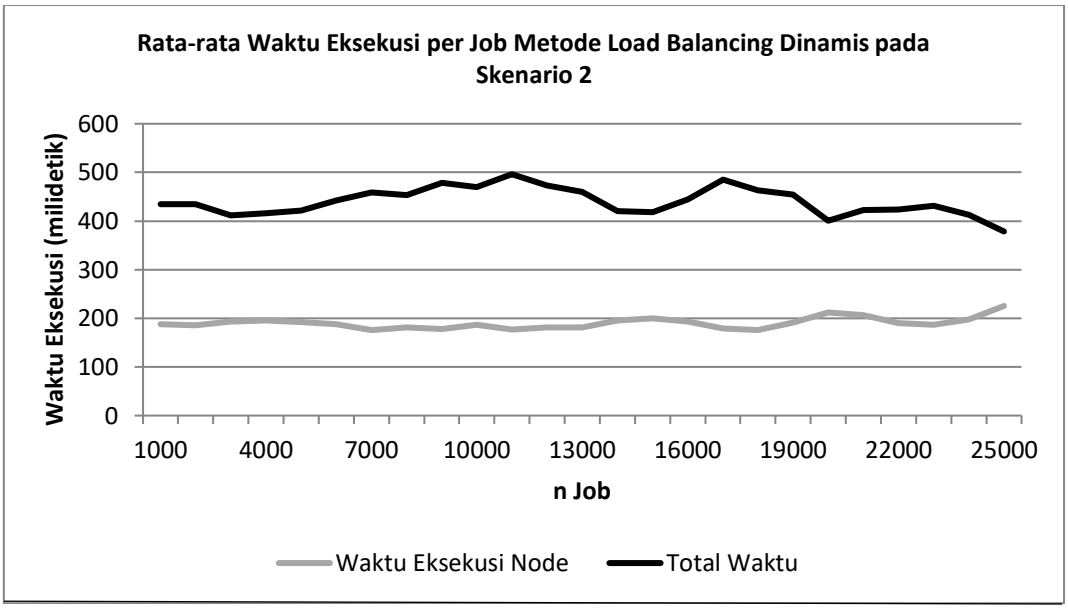
3.2. Skenario 2

Pada skenario 2, disimulasikan sistem berada dalam kondisi dinamis, yaitu salah satu *node* diatur mengalami beban yang berubah-ubah selama proses eksekusi *job* berjalan. Beban

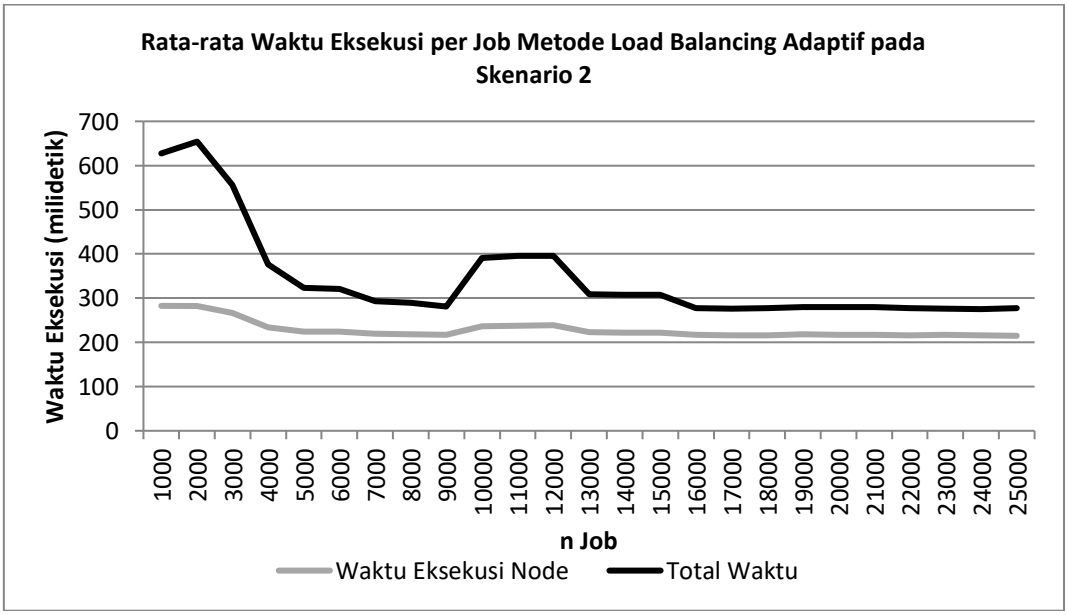
yang diberikan kepada salah satu *node* berupa *throughput* jaringan yang berubah-ubah dan beban CPU yang berubah-ubah. Jumlah *job* yang dieksekusi adalah 25.000 *job*.

Pada skenario ini, salah satu *node* mengalami perubahan *throughput* jaringan yang berubah antara 10Mbps dan 2Mbps yang diterapkan selama satu menit secara bergantian selama proses eksekusi *job* berjalan. Selain itu, *node* ini juga diberikan beban CPU sebesar 0% dan 50% yang dijalankan selama satu menit secara bergantian selama proses eksekusi *job* berjalan. Beban CPU diimplementasikan dengan mengeksekusi proses deskripsi *cipheredtext* MD5 menjadi *plaintext* yang dijalankan secara lokal pada suatu *node*.

Gambar 8 menunjukkan rata-rata waktu eksekusi per *job* pada skenario kedua. Gambar 8(a) menunjukkan rata-rata waktu eksekusi per *job* menggunakan metode Load Balancing Dinamis. Rata-rata waktu eksekusi yang berjalan pada sisi *node* menggunakan metode Load Balancing Dinamis adalah selama 180 milidetik. Sedangkan, secara keseluruhan rata-rata waktu eksekusi per *job* adalah selama 440 milidetik.



(a)



(b)

Gambar 8 (a)(b). Grafik Rata-rata Waktu Eksekusi Per Job Pada Skenario 2

Pada hasil uji coba menggunakan metode Load Balancing Adaptif yang ditunjukkan pada Gambar 8(b), rata-rata waktu eksekusi yang berjalan pada sisi *node* adalah selama 228 milidetik. Sedangkan, secara keseluruhan, rata-rata waktu eksekusi per *job* adalah selama 344 milidetik.

Dilihat dari segi rata-rata waktu eksekusi secara keseluruhan, metode Load Balancing Adaptif memberikan rata-rata waktu eksekusi yang lebih kecil. Hal ini membuktikan bahwa, pada kondisi lingkungan dinamis dengan salah satu *node* mengalami *throughput* jaringan yang berubah-ubah antara 10Mbps dan 2Mbps dan mengalami beban CPU berubah-ubah antara 0% dan 50%, metode Load Balancing Adaptif yang diusulkan terbukti lebih baik dibandingkan dengan metode Load Balancing Dinamis

Pada hasil uji coba metode Load Balancing Dinamis yang ditunjukkan Gambar 8(a), terdapat perbedaan sekitar 249 milidetik antara rata-rata waktu eksekusi pada sisi *node* dan rata-rata waktu eksekusi secara keseluruhan. Sedangkan hasil uji coba metode Load Balancing Adaptif yang ditunjukkan pada Gambar 8(b), terdapat perbedaan sekitar 115 milidetik antara rata-rata waktu eksekusi pada sisi *node* dan rata-rata waktu eksekusi secara keseluruhan. Hal ini menunjukkan bahwa pada saat kondisi lingkungan dinamis, metode Load Balancing Adaptif melakukan penggunaan sumber daya jaringan secara lebih efisien dibandingkan dengan metode Load Balancing Dinamis.

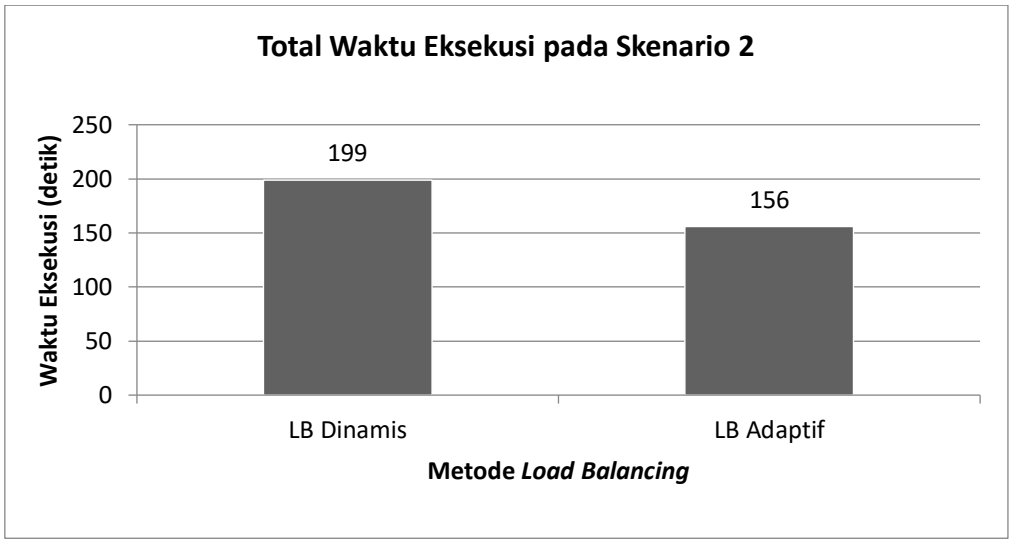
Selain perbedaan antara rata-rata waktu eksekusi pada *node* dan rata-rata waktu eksekusi secara keseluruhan, dari hasil uji coba didapatkan data adanya *node idle* yang ditunjukkan pada

Tabel 2. Pada uji coba menggunakan metode Load Balancing Dinamis terjadi 4.982 kemunculan *node idle*. Sedangkan, saat menggunakan metode Load Balancing Adaptif terjadi 408 kemunculan *node idle*.

Tabel 2. Tabel Frekuensi Node Idle pada Skenario 2

Metode *Load Balancing*	Frekuensi *Node Idle*
Dinamis	4.982
Adaptif	408

Dari sisi total waktu yang digunakan untuk mengeksekusi seluruh 25.000 *job*, metode Load Balancing Adaptif lebih unggul dibandingkan dengan metode Load Balancing Dinamis seperti yang ditunjukkan Gambar 9. Saat uji coba menggunakan metode Load Balancing Dinamis, total waktu yang digunakan untuk mengeksekusi seluruh 25.000 *job* adalah selama 199 menit. Sedangkan, pada uji coba menggunakan metode Load Balancing Adaptif menggunakan RL, total waktu yang digunakan untuk mengeksekusi seluruh 25.000 *job* adalah selama 156 menit. Total waktu yang digunakan untuk mengeksekusi 25.000 *job* menggunakan metode Load Balancing pada skenario kedua adalah 21% lebih cepat dibandingkan dengan menggunakan metode Load Balancing Dinamis.



Gambar 9. Grafik Total Waktu Eksekusi pada Skenario 2

4. Kesimpulan

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

1. Perbaikan mekanisme Load Balancing dengan menggunakan algoritma RL untuk komputasi klaster terbukti dapat meningkatkan kinerja komputasi klaster baik pada lingkungan dalam kondisi normal maupun dalam kondisi dinamis.
2. Hasil uji coba pengukuran rata-rata waktu eksekusi per *job* pada kondisi normal menunjukkan bahwa penggunaan metode Load Balancing Adaptif dengan menggunakan algoritma RL diperoleh rata-rata waktu eksekusi per *job* adalah selama 208 milidetik, sedangkan dengan menggunakan metode Load Balancing Dinamis diperoleh rata-rata waktu eksekusi per *job* adalah selama 383 milidetik, terjadi peningkatan sebesar 45%.
3. Hasil uji coba pengukuran rata-rata waktu eksekusi per *job* pada kondisi dinamis menunjukkan bahwa penggunaan metode Load Balancing Adpatif dengan menggunakan algoritma RL diperoleh rata-rata waktu eksekusi per *job* adalah selama 344 milidetik, sedangkan dengan menggunakan metode Load Balancing Dinamis diperoleh rata-rata waktu eksekusi per *job* adalah selama 440 milidetik, terjadi peningkatan sebesar 21%.
4. Hasil pengukuran total waktu eksekusi pada kondisi normal menunjukkan bahwa penggunaan metode Load Balancing Adaptif dengan menggunakan algoritma RL diperoleh total waktu eksekusi selama 95 menit, sedangkan dengan menggunakan metode Load Balancing Dinamis diperoleh total waktu eksekusi selama 173 menit, terjadi peningkatan sebesar 45%.
5. Hasil pengukuran total waktu eksekusi pada kondisi dinamis menunjukkan bahwa penggunaan metode Load Balancing Adaptif dengan menggunakan algoritma RL diperoleh total waktu eksekusi selama 156 menit, sedangkan dengan menggunakan metode Load Balancing Dinamis diperoleh total waktu eksekusi selama 199 menit, terjadi peningkatan sebesar 21%.

Referensi

[1] F. Berman, R. Wolski, And S. Figueira, “Application-Level Scheduling On Distributed Heterogeneous Networks,” In Proceedings Of The 1996 Acm/Ieee Conference On Supercomputing, 1996.

[2] P. Mohammadpour And M. Sharifi, “A Self-Training Algorithm For Load Balancing In Cluster Computing,” Fourth International Networked Computing and Advanced Information Management, Vol. 1, 2008.

[3] D. Mahato, A. Maurya, And A. Tripathi, “Dynamic And Adaptive Load Balancing In Transaction Oriented Grid Service,” In 2nd International Conference On Green High Performance Computing, 2016.

[4] H. Desai And R. Oza, “A Study Of Dynamic Load Balancing In Grid Environment,” In International Conference On Wireless Communications, Signal Processing And Networking, 2016.

[5] A. Radulescu And A. Van Gemund, “Low-Cost Task Scheduling For Distributed-Memory Machines,” IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 6, Pp. 648–658, 2002.

[6] A. Tchernykh And J. Cortés-Mendoza, “Adaptive Energy Efficient Distributed Voip Load Balancing In Federated Cloud Infrastructure,” In IEEE 3rd International Conference On Cloud Networking, 2014.

[7] S. Li, S. Zhao, X. Wang, K. Zhang, And L. Li, “Adaptive And Secure Load-Balancing Routing Protocol For Service-Oriented Wireless Sensor Networks,” IEEE Systems Journal, Vol. 8, No. 3, Pp. 858–867, 2014.

[8] J. Wu, X. Xu, P. Zhang, And C. Liu, “A Novel Multi-Agent Reinforcement Learning Approach For Job Scheduling In Grid Computing,” Future Generation Computer Systems, Vol. 25, No. 5, Pp. 430–439, 2011.

[9] C. Wirth And J. Fürnkranz, “Epmc: Every Visit Preference Monte Carlo For Reinforcement Learning.,” In Proceedings Of The 5th Asian Conference On Machine Learning, 2013, Vol. 29, Pp. 483–497.

[10] R. Sutton And A. Barto, Reinforcement Learning: An Introduction. Cambridge, Massachusetts: The Mit Press, 2012.