Terakreditasi SINTA Peringkat 2
Surat Keputusan Dirjen Penguatan Riset dan Pengembangan Ristek Dikti No. 10/E/KPT/2019
masa berlaku mulai Vol. 1 No. 1 tahun 2017 s.d Vol. 5 No. 3 tahun 2021

Terbit online pada laman web jurnal: **http://jurnal.iaii.or.id**

JURNAL RESTI
(Rekayasa Sistem dan Teknologi Informasi)
Vol. 4 No. 6 (2020) 1190-1197 ISSN Media Elektronik: 2580-0760

Implementasi Algoritma Load Balancing PLBA Komputasi Grid pada Lab Environment Menggunakan PVM3

Taufiq Odhi Dwi Putra[1], Wisnu Widiarto[2], Wiharto[3]
[123]Informatika, Universitas Sebelas Maret Surakarta
[1]taufiq_0dhi_dwi_putra@student.uns.ac.id, [2]wisnu.widiarto@staff.uns.ac.id*, [3]wiharto@staff.uns.ac.id

Abstract

*Load balancing is one of the main parts of scheduling Grid resources. One of the load balancing models on Grid resources is the hierarchical model. This model has the advantage that it requires minimal communication costs between one resource and another. The PLBA load balancing algorithm uses a hierarchical model with dynamically obtained threshold values, so that it can adjust conditions at a time, both the state of the resource, the state of the computer network, and the state of the recipient or client. PVM3 is a software system capable of optimizing heterogeneous resources, so that resources can work in parallel. Resources can also complete tasks well, even though they are very large and complex tasks. This research has implemented the PLBA load balancing algorithm, with the aim of optimizing Grid resources. This research has also developed the PLBA load balancing algorithm by changing the arguments for NPEList, so that resources can be grouped more optimally. The PLBA load balancing algorithm has been successfully developed by modifying the arguments for NPEList, so that the running time required to complete the given tasks is shorter, because resources can be grouped more optimally. This has been shown by the shorter average running time when using the modified NPEList argument (0.75 * threshold1 <= ALCi <= 1.25 * threshold1) is shorter, than using the NPEList argument in previous research (ALCi = threshold1). Comparison of the average running time has been obtained as follows : (82513.63740 : 67837.71720); (63869.92450 : 50722.17210); (858,96710 : 207,33680); (321.88000 : 126.89100); (768.54560 : 468.27190); (780.22770 : 279.43730).*

Keywords: Grid Computing, Load Balancing, Hierarchical Load Balancing Model, Virtual Machine, PVM3

Abstrak

Load balancing adalah salah satu bagian utama pada penjadwalan sumber daya Grid. Salah satu model *load balancing* pada sumber daya Grid yaitu model hierarki. Model ini memiliki kelebihan yaitu memerlukan biaya komunikasi yang sedikit, antara sumber daya satu dengan sumber daya lainnya. Algoritma *load balancing* PLBA menggunakan model hierarki dengan nilai threshold yang diperoleh secara dinamis, sehingga dapat menyesuaikan keadaan pada suatu waktu, baik keadaan sumber daya, keadaan jaringan komputer, dan keadaan penerima atau client. PVM3 adalah sistem perangkat lunak yang mampu mengoptimalkan sumber daya yang heterogen, sehingga sumber daya dapat bekerja secara parallel. Sumber daya juga bisa menyelesaikan tasks dengan baik, meskipun tasks tersebut merupakan tasks yang sangat besar dan kompleks. Penelitian ini mengimplementasikan algoritma *load balancing* PLBA, dengan tujuan untuk mengoptimalkan sumber daya Grid. Penelitian ini juga mengembangkan algoritma *load balancing* PLBA dengan mengubah argument untuk NPEList, sehingga sumber daya dapat dikelompokkan dengan lebih optimal. Algoritma *load balancing* PLBA berhasil dikembangkan dengan memodifikasi argument untuk NPEList sehingga *running time* yang diperlukan untuk menyelesaikan tasks yang diberikan lebih singkat, karena sumber daya dapat dikelompokkan dengan lebih optimal. Hal ini ditunjukkan dengan rata – rata *running time* pada saat menggunakan argument NPEList yang dimodifikasi (0,75*threshold1 <= ALCi <= 1,25*threshold1) lebih singkat, daripada menggunakan argument NPEList pada penelitian sebelumnya (ALCi = threshold1). Perbandingan rata-rata *running time* tersebut adalah sebagai berikut: (82513.63740 : 67837.71720); (63869.92450 : 50722.17210); (858.96710 : 207.33680); (321.88000 : 126.89100); (768.54560 : 468.27190); (780.22770 : 279.43730).

Kata kunci: Grid Computing, Load Balancing, Hierarchical Load Balancing Model, Virtual Machine, PVM3

Diterima Redaksi: 08-11-2020 | Selesai Revisi: 27-12-2020 | Diterbitkan Online: 30-12-2020

1. Pendahuluan

Grid merupakan suatu teknologi yang banyak digunakan suatu perusahaan atau instansi untuk memenuhi kebutuhan komputasi mereka, karena didukung oleh perkembangan teknologi jaringan di mana memungkinkan antara komputer satu dengan komputer lainnya dapat bekerja untuk menyelesaikan suatu proses yang kompleks secara optimal [1,2,3]. Perkembangan ilmu pengetahuan di bidang teknologi informasi membuat kebutuhan komputasi semakin besar. Hal ini disebabkan oleh semakin kompleks proses yang harus diselesaikan karena semakin banyak user yang menggunakan dan kebutuhan user yang dinamis. Oleh karena itu banyak perusahaan atau instansi yang mengandalkan teknologi komputasi grid untuk memenuhi kebutuhan komputasi mereka dikarenakan keterbatasan resource atau sumber daya yang dimiliki.

Salah satu komponen dalam komputasi Grid yaitu *load balancing*. *Load balancing* merupakan proses untuk membagi load atau proses ke komputer–komputer yang terhubung [4,5]. Komponen ini merupakan salah satu komponen penting dalam grid, karena untuk load atau proses harus didistribusikan ke komputer–komputer sesuai dengan keadaan dan spesifikasi dari komputer-komputer tersebut, sehingga sumber daya yang ada dapat dioptimalkan secara maksimal[6,7].

Terdapat beberapa algoritma *load balancing* yang dikembangkan berdasarkan beberapa pendekatan, antara lain Tree based approach, Estimation based approach, Optimization based approach, Agent based approach, Hybrid based approach, Neighbor based approach, Partitioning based approach [8,9]. Masing-masing algoritma mempunyai kelebihan dan kekurangan serta digunakan untuk menyelesaikan masalah tertentu untuk hasil yang paling optimal.

Penelitian-penelitian yang sudah dilakukan, yaitu membandingkan suatu algoritma *load balancing* dan mengembangkannya, sehingga menghasilkan algoritma baru, serta mencoba membandingkan algoritma-algoritma tersebut dengan mencobanya pada suatu program simulasi. Program simulasi yang sering digunakan yaitu GridSim atau CloudSim[10]. Mengembangkan algoritma *load balancing* dengan pendekatan secara hierarki yang berbasis variable threshold value dan diberi nama algoritma PLBA [11]. Mengintegrasikan algoritma *load balancing* yang dibuat dengan fault-tolerant scheduling yang bernama MinRC dan mengembangkan kemampuan dari algoritma *load balancing* berbasis fault-tolerant dan diberi nama algoritma PD_MinRC [12]. Mengembangkan algoritma *load balancing* berdasarkan fault tolerant dan user deadline dan diberi nama algoritma HLBFT [13]. Mengembangkan algoritma *load balancing* yang dinamis yang menyediakan pengaturan untuk deadline dari tasks yang diberikan dan diberi nama algoritma EGDC [14].

Algoritma *load balancing* PLBA [11], merupakan salah satu algoritma load balancing dengan pendekatan *Tree based approach* dan konsep hierarki yang mengelompokkan sumber daya menjadi 4 kelompok yaitu *overloaded*, normal, *lightly*, dan *underlightly*. Algoritma ini menggunakan suatu nilai threshold untuk mengelompokkan sumber daya menjadi 4 kelompok tersebut, di mana nilai threshold didapatkan secara dinamis.

2. Metode Penelitian

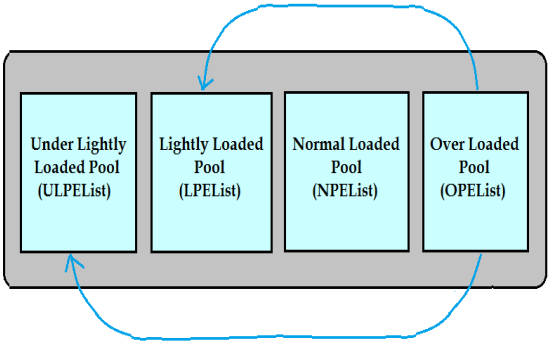
Pada bagian ini dijelaskan tentang metode yang digunakan yaitu algoritma load balancing PLBA dan PVM3 (*Parallel Virtual Machine*) versi 3.

2.1. Algoritma load balancing PLBA

Model load balancing pada algoritma PLBA dapat diklasifikasikan menjadi tiga level, yaitu *resource*, *machine*, dan PE (*Processing Element*) [11]. Algoritma PLBA mengeksekusi secara individu pada setiap level, menurut nilai threshold pada setiap level dan algoritma PE level pada level 2. *Load* sudah dihitung ketika proses datang pada setiap node (komputer) dan terhubung pada masing-masing PE. Proses *load balancing* yang dilakukan tergantung pada *availability* dari daftar node yang memiliki load ringan atau agak ringan dan pada saat yang sama node-node tersebut mengirimkan informasi tentang daftar node ini pada setiap komputer. Untuk melakukan proses load balancing, secara acak memilih suatu proses dikirimkan ke node yang memiliki load tinggi atau node yang memiliki *load* yang rendah. Strategi acak diimplementasikan pada algoritma untuk level *machine*. Akhirnya, proses perpindahan task dilakukan sampai *overload queue* tidak kosong dan prosedur yang sama juga diimplementasikan pada level *machine* dan level *resource* sebagaimana Gambar 1.

Algoritma PLBA mengelompokkan PE menjadi 4 kelompok yaitu OPEList (*Overloaded PE List*), NPEList (Normal PE List), LPEList (*Lightly Loaded PEList*), dan ULPEList (*Under Lighly PE List*). OPEList adalah PE yang dikategorikan sebagai overload, artinya PE memiliki beban proses yang sangat tinggi, sehingga jika ada task yang ingin diproses pada PE tersebut, maka task akan diproses dengan kurang optimal. NPEList adalah PE yang dikategorikan normal, artinya PE memiliki beban proses yang normal. LPEList adalah PE yang dikategorikan ringan, artinya PE memiliki beban proses yang sedikit. ULPEList adalah PE yang dikategorikan sangat ringan, artinya PE memiliki beban proses yang sangat sedikit. Algoritma ini bertujuan untuk memindahkan tasks yang akan diproses pada PE yang

termasuk pada OPEList sebanyak mungkin ke PE yang termasuk pada LPEList atau ULPEList.



Gambar 1. Metode Penelitian Pemilihan PE Load Balance

Algoritma *load balancing* PLBA mengelompokkan PE sesuai yang dijelaskan di atas dengan menggunakan nilai threshold yang dinamis sesuai dengan kapan suatu tasks datang dan harus diproses. Nilai threshold [11] tersebut didapatkan dari persamaan (1).

$$\partial = ALM_i + \sigma \quad (1)$$

Dimana ∂ merupakan nilai threshold pertama, ALMi (*average load machine system*) yaitu rata – rata load pada level machine, dan σ adalah standar deviasi dari load pada level *machine*. Standar deviasi dari load pada level *machine* dapat dinotasikan pada persamaan (2).

$$\sigma = \frac{1}{N}\sum_{i=1}^{N}(ALC_i - ALM)^2 \quad (2)$$

Dimana σ adalah nilai standar deviasi dari load pada level machine, ALM (*average load machine system*) yaitu rata – rata load pada level *machine*, dan ALCi (*average load cluster*) yaitu rata – rata load pada setiap cluster i, dan N adalah banyaknya cluster pada suatu sistem di level *machine*. Nilai ALM dapat dihitung dari persamaan (3).

$$ALM_i = \frac{1}{m}\sum_{i=1}^{m} ALC_i \quad (3)$$

Dimana ALMi (*average load machine system*) yaitu rata – rata load pada level *machine*, ALCi (*average load cluster*) yaitu rata – rata load pada setiap *cluster*, dan m adalah banyaknya *cluster* pada suatu sistem di level *machine*. Nilai ALCi diperoleh dari persamaan (4).

$$ALC_i = \frac{1}{p}\sum_{k=1}^{p} PELoad_{k,i} \quad (4)$$

Dimana ALCi (*average load cluster*) yaitu rata – rata *load* pada setiap cluster, PELoadk,i adalah *load* dari semua PE di *cluster* i, p adalah jumlah PE di dalam suatu cluster. Nilai PELoadk,i dihitung dari persamaan (5).

$$PELoad_{k,i} = \frac{CurrLoad_{PE} - AvgLoad}{Rating} \quad (5)$$

Dimana PELoadk,i adalah load dari semua (PE)k di cluster i, CurrLoadPE adalah load dari PE saat ini, AvgLoad adalah rata – rata load dari PE, dan Rating adalah kapasitas sebenarnya dari PE seperti kecepatan CPU dalam ketentuan rating MIPS (*Million Instructions Per Second*). Nilai AvgLoad dapat dihitung dari persamaan (6).

$$AvgLoad = \sum_{i=1}^{p}\left(\frac{CurrLoad_i}{p}\right) \quad (6)$$

Dimana AvgLoad adalah rata – rata load dari PE, CurrLoadi adalah load dari PE saat ini, dan p adalah jumlah PE di dalam suatu cluster. CurrLoadp dapat dinotasikan pada persamaan (7).

$$CurrLoad_{PE} = \sum_{1}^{g} FileSize \quad (7)$$

Dimana CurrLoadPE adalah load dari PE saat ini, g adalah jumlah semua tasks di dalam suatu PE, dan FileSize adalah workload dari suatu PE. Kemudian terdapat nilai threshold lainnya yang digunakan untuk menentukan apakah terdapat log yang ditambahkan pada log *Overloaded* PE. Nilai threshold yang kedua ini didapatkan dari persamaan (8).

$$\eta = ALR_i + \sigma_1 \quad (8)$$

Dimana η merupakan nilai threshold kedua, ALRi (*Average Load of Resource System*) yaitu rata – rata load pada level resource, dan σ1 adalah nilai standar deviasi untuk threshold kedua. Nilai standar deviasi untuk threshold kedua dapat dinotasikan pada persamaan (9).

$$\sigma_1 = \frac{1}{N}\sum_{i=1}^{N}(ALM_i - ALR)^2 \quad (9)$$

Dimana σ1 adalah nilai standar deviasi untuk threshold kedua, ALMi (*average load machine system*) yaitu rata – rata load pada level machine, ALR (*Average Load of Resource System*) yaitu rata – rata load pada level resource, dan N adalah banyaknya sistem pada level machine. Nilai ALR dapat dihitung dengan persamaan (10).

$$ALR_i = \frac{1}{r}\sum_{i=1}^{r} ALM_i \quad (10)$$

Dimana ALRi (*Average Load of Resource System*) yaitu rata – rata load pada level resource, ALMi (*average load machine system*) yaitu rata – rata load pada level machine, dan r adalah banyaknya sistem pada level machine. Langkah–langkah algoritma load balancing PLBA, pada masing–masing level dijabarkan dalam algoritma 1, algoritma 2, algoritma 3 dan algoritma 4 [11].

Algoritma 1: *Load Balancing Activity at PE level*

Deskripsi: Algoritma ini mengeksekusi pada setiap node (0 <= i <= n)
Input: daftar proses yang siap untuk dieksekusi
Output: Antrian pada daftar PE untuk eksekusi proses

1. Begin
2. For (mengecek semua machine pada masing – masing sumber)
3. Menunggu aktivitas(seperti perubahan load pada setiap machine)
4. If (terdapat aktivitas)
5. Memanggil Algorithm 2 untuk mencari kategori PE (contoh OP, UP, LP, NP)
6. Else ke langkah 2
7. End if
8. End for
9. End

Algoritma 2: *Load calculation algorithm at PE level*

Deskripsi: Algoritma ini mengeksekusi pada setiap PE sampai PE! =0
Input: Ukuran file (sebagai workload pada PE), rating
Output: Menghitung load pada setiap PE

1. Begin
2. Menginisiasi PELoad dan mengecek semua proses
3. If PELoad = 0
4. Ke langkah 9
5. Else
6. Mengecek load untuk semua PE sesuai dengan proses masing-masing
7. Menghitung load saat ini pada PE = $\sum_{1}^{g}(\boldsymbol{FileSize})\boldsymbol{j}$
8. Jumlah PELoad adalah PE = ((Curr_Load)PE – Avg_load)/ Rating)
9. Return PELoad
10. End if
11. End

Algoritma 3: *Load balancing algorithm at machine level*

Deskripsi: Untuk membagi proses load ke dalam daftar PE yang dieksekusi pada semua machine..
Input: nilai threshold, PELoad, Algorithm 2
Output: : Persebaran task sesuai dengan load di semua daftar PE

1. Begin
2. Membuat OPElist, ULPEList, LPEList dan NPEList
3. Menghitung ALM, dan standar deviasi
4. For (semua PE diasosiasikan dengan machine yang sedang memproses/kosong)
5. Memanggil PELoad_Calc_Algo() //Menghitung PELoad (work load dari PE)
6. If (ALCi == Nilai threshold PELoad)
7. Menambahkan PE ini ke NPEList
8. Else if (ALCi > Nilai threshold PELoad)
9. Menambahkan PE ini ke OPEList
10. If (ALCi < 0.5* Nilai threshold PELoad)
11. Menambahkan PE ini ke ULPEList
12. Else Menambahkan PE ini ke LPEList
13. End if
14. End if
15. End if
16. End for
17. Mengurutkan OPEList secara descending, ULPEList dan LPEList secara ascending terhadap loads
18. Memanggil Algorithm 4
19. End

Algoritma 4: job migration algorithm at machine level

Deskripsi: menghitung load untuk proses perpindahan job/tasks.
Input: PE load, Tasks load
Output: Mengirimkan informasi load ke sumber daya pada upper level job migration.

1. Begin
2. For (mengecek PE dari overloaded PE List (descending) dari PE yang dipilih)
3. If (mengecek untuk semua tasks yang diproses atau kosong)
4. Ke langkah 18
5. Else
6. Mengecek untuk semua PE dalam lightly loaded PE list (satu persatu secara ascending)
7. If (menghitung dan mengecek ((PELoad)D + (TaskLoad)¬L < 0.5* Nilai threshold PELoad))
8. Menggeser tasks load dari overload ke lightly loaded PE List
9. Else mengecek untuk semua PE di under lightly loaded PElist (satu persatu secara ascending)
10. If (menghitung dan mengecek ((PELoad)E + (TaskLoad)¬L < 0.5* Nilai threshold PELoad))
11. Menggeser tasks load dari overload ke under lightly loaded PE List
12. Else tasks load dieksekusi di machine semula
13. End if
14. End if
15. End if
16. End for
17. Memanggil Algorithm 3
18. If ((ALCi >= nilai threshold pada level machine) //situasi overload machine
19. Mengirimkan informasi ke level resource untuk perhitungan selanjutnya
20. End if
21. End

2.2. *Parallel Virtual Machine* versi 3 (PVM3)

PVM3 adalah perangkat lunak yang memungkinkan beberapa komputer yang terhubung ke jaringan dapat digunakan sebagai suatu komputer paralel yang besar [15]. Masalah komputasi yang memerlukan sumber daya

yang besar dapat diselesaikan dengan beberapa komputer bekerja secara paralel menggunakan perangkat lunak ini [16]. Virtual machine digunakan untuk mengatur persebaran memory dari berbagai komputer. Host digunakan untuk mencatat komputer-komputer yang terhubung dan dapat digunakan untuk mengeksekusi proses.

PVM mendukung berbagai jenis aplikasi dan arsitektur mesin, serta level jaringan. PVM dapat memungkinkan menjalankan tasks ke mesin dengan arsitektur dan aplikasi yang sesuai untuk solusi dari tasks tersebut. PVM dapat mengkonversi semua data jika antara dua komputer mempunyai *integer point* atau *floating point* yang berbeda. PVM dapat menghubungkan antar *virtual machine* melalui berbagai tipe jaringan.

Sistem PVM terdiri dari 2 bagian. Bagian pertama yaitu daemon, bernama pvmd3 atau terkadang pada arsitektur tertentu bernama pvmd yang membuat *virtual machine* pada komputer. Pvmd3 didesain sehingga pengguna dengan *autentifikasi* yang valid dapat menginstall daemon ini pada komputer. Jika pengguna ingin menjalankan PVM, dia harus membuat *virtual machine* dengan menjalankan pvmd3 terlebih dahulu. PVM dapat dijalankan di terminal atau *command prompt*. Banyak pengguna dapat mengkonfigurasi antar *virtual machine* satu sama lain, dan dapat menjalankan beberapa aplikasi PVM secara simultan. Fitur-fitur yang terdapat pada PVM3 yaitu:

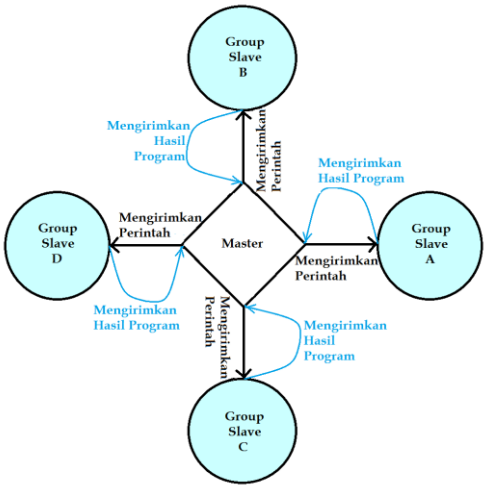
1. *User Interface* yang diupdate dari versi sebelumnya.
2. *Integer Task Identifier*, semua proses yang dijalankan pada PVM3 direpresentasikan sebagai *Integer Task Identifier.*
3. *Process Control*, PVM menyediakan fitur sehingga proses yang dijalankan dapat menjadi PVM task dan dapat dikembalikan menjadi proses normal lagi.
4. *Fault Tolerance*, PVM dapat mendeteksi secara otomatis jika terdapat *host fails* dan menghapus host tersebut dari *virtual machine.*
5. *Dynamic Process Groups*, proses dapat dimiliki oleh beberapa group, dan group dapat dapat mengubahnya secara dinamis pada waktu kapan saja saat komputasi sedang berlangsung.
6. *Signaling*, PVM menyediakan dua cara untuk memberi sinyal kepada PVM tasks lainnya. Cara pertama mengirimkan sinyal UNIX ke task lainnya. Cara kedua yaitu mengirimkan pesan notifikasi dengan tag pengguna tertentu dan aplikasi dapat mengeceknya.
7. *Communication*, PVM menyediakan siklus untuk mengirim dan menerima pesan antara task satu dengan task lainnya.
8. *Multiprocessor Integration*, PVM dapat mengintegrasikan komputer satu dengan komputer lainnya dengan tipe dan aristektur yang berbeda tanpa membebankannya kepada perangkat lunak jaringan seperti TCP/IP.

3. Hasil dan Pembahasan

Hasil percobaan implementasi algoritma load balancing PLBA menggunakan PVM3 pada lab Environment dapat dijabarkan dalam 2 hal: Lab. Environment dan Hasil Percobaan Implementasi Program.

3.1. *Lab Environment*

Implementasi percobaan pada tahap ini dilakukan menggunakan 31 virtual machine dengan Oracle VM Virtualbox versi 6.1.0. virtual machine yang digunakan terdiri dari 16 virtual machine dengan sistem operasi Ubuntu 18.04.4 LTS dan 15 virtual machine dengan sistem operasi Debian 10 Buster di mana virtual machine tidak terinstal desktop environment. Konfigurasi untuk setiap *virtual machine* dibuat sama yaitu 4 CPU, 4 Gb RAM, 20 Gb hardisk, dan network adapter jenis *Bridged adapter* yaitu *network interface* yang mirip seperti salah satu network interface pada komputer yang menjalankannya serta mendapat ip address yang satu network dengan network interface tersebut. Spesifikasi komputer yang digunakan yaitu CPU 3.10 Ghz x4, RAM 8 Gb, hardisk 1 Tb, dengan sistem operasi Windows 7. Komputer yang digunakan berjumlah 24 di mana masing – masing komputer menjalankan satu atau dua virtual machine. Konfigurasi jaringan yang digunakan yaitu menggunakan local area network yang dihubungkan interface Gigabit Ethernet.



Gambar 2. Ilustrasi Pengujian Implementasi Program

3.2. Hasil Percobaan Implementasi Program

Percobaan pada tahap ini yaitu menguji algoritma Load Balancing PLBA dengan menggunakan 3 program yaitu program A, program B, dan Program C. Program A adalah program yang melakukan proses membuat data berupa angka sebanyak 100.000.000 secara acak, kemudian data tersebut diurutkan dengan menggunakan algoritma sorting Merge Sort. Program B adalah program yang melakukan proses membuat data berupa angka sebanyak 1.000.000 secara acak, kemudian data tersebut dijumlahkan semua.

Tabel 1. Running Time (ms) Percobaan Program A

No.	*Argument* NPEList			
	ALC_i = threshold1		0,75*threshold1 <= ALC_i <= 1,25*threshold1	
	25 Hosts	30 Hosts	25 Hosts	30 Hosts
1	85778.08800	66348.00100	67794.60300	45755.21300
2	96164.64900	58215.60500	66758.79100	58376.39000
3	62034.33400	66057.35800	55829.83900	50177.47300
4	97337.94800	65335.72700	67505.78400	47895.97600
5	99177.46700	67202.46200	75569.22400	50991.72300
6	61608.69900	64453.51200	68690.16300	54360.56800
7	63971.62400	66172.19400	68296.71900	52065.33200
8	97271.09000	66456.95100	71692.89700	51716.92600
9	97755.26800	64248.14000	68054.65100	52501.67900
10	64037.20700	54209.29500	68184.50100	43380.44100
Rata2	82513.63740	63869.92450	67837.71720	50722.17210

Tabel 2. Running Time (ms) Percobaan Program B

No.	*Argument* NPEList			
	ALC_i = threshold1		0,75*threshold1 <= ALC_i <= 1,25*threshold1	
	25 Hosts	30 Hosts	25 Hosts	30 Hosts
1	776.15400	306.44500	170.79600	128.55300
2	770.19900	301.69200	186.06200	132.37300
3	1724.4660	308.70500	193.56100	127.45700
4	762.25600	300.88400	171.57000	129.15100
5	750.01200	302.63600	187.84900	120.66900
6	759.49500	342.42000	194.91700	133.54400
7	759.41800	337.97000	190.29000	116.21500
8	769.86900	341.41800	261.29500	124.30900
9	761.01500	342.86100	257.40700	131.14400
10	756.78700	333.76900	259.62100	125.49500
Rata2	858.96710	321.88000	207.33680	126.89100

Tabel 3. Running Time (ms) Percobaan Program C

No.	*Argument* NPEList			
	ALC_i = threshold1		0,75*threshold1 <= ALC_i <= 1,25*threshold1	
	25 Hosts	30 Hosts	25 Hosts	30 Hosts
1	828.16800	759.03800	455.88700	300.04900
2	811.04300	721.03200	440.77200	266.20500
3	772.95800	708.92500	451.21200	232.54900
4	769.47600	715.09400	438.99500	225.48000
5	751.94600	711.28300	452.12100	231.40100
6	755.11900	738.66000	509.80200	381.87200
7	765.20300	773.84800	458.05100	311.25600
8	776.56000	754.07800	484.61300	285.16500
9	743.86000	1215.77200	496.29300	287.86400
10	711.12300	704.54700	494.97300	272.53200
Rata2	768.54560	780.22770	468.27190	279.43730

Program C adalah program yang membuat angka Fibonacci dari ke-1 sampai ke-10.000. Ketiga program tersebut dijalankan secara parallel dengan model membagi data yang harus dibuat dan diproses sesuai dengan jumlah virtual machine yang dihubungkan pada PVM (parallel virtual machine).

Tabel 1 menunjukkan running time menggunakan program A dengan 4 kelompok komputer. Percobaan dilakukan sebanyak 10 kali proses yang dijalankan pada satu kali eksekusi program, dengan running time yaitu:

a) 82513.63740 ms, untuk argument NPEList ALCi = threshold1 dan jumlah hosts 25,
b) 63869.92450 ms, untuk argument NPEList ALCi = threshold1 dan jumlah hosts 30,
c) 67837.71720 ms, untuk argument NPEList 0,75*threshold1 <= ALCi <= 1,25*threshold1 dan jumlah hosts 25,
d) 50722.17210 ms, untuk argument NPEList 0,75*threshold1 <= ALCi <= 1,25*threshold1 dan jumlah hosts 30.

Tabel 2 menunjukkan running time menggunakan program B dengan 4 kelompok komputer. Percobaan dilakukan sebanyak 10 kali proses yang dijalankan pada satu kali eksekusi program, dengan running time yaitu:

a) 858.96710 ms, untuk argument NPEList ALCi = threshold1 dan jumlah hosts 25,
b) 321.88000 ms, untuk argument NPEList ALCi = threshold1 dan jumlah hosts 30,
c) 207.33680 ms, untuk argument NPEList 0,75*threshold1 <= ALCi <= 1,25*threshold1 dan jumlah hosts 25,
d) 126.89100 ms, untuk argument NPEList 0,75*threshold1 <= ALCi <= 1,25*threshold1 dan jumlah hosts 30.

Tabel 3 menunjukkan running time dengan 4 kelompok komputer. Percobaan dilakukan sebanyak 10 kali proses yang dijalankan pada satu kali eksekusi program, dengan running time yaitu:

a) 768.54560 ms, untuk argument NPEList ALCi = threshold1 dan jumlah hosts 25,
b) 780.22770 ms, untuk argument NPEList ALCi = threshold1 dan jumlah hosts 30,
c) 468.27190 ms, untuk argument NPEList 0,75*threshold1 <= ALCi <= 1,25*threshold1 dan jumlah hosts 25,
d) 279.43730 ms, untuk argument NPEList 0,75*threshold1 <= ALCi <= 1,25*threshold1 dan jumlah hosts 30.

Pada Gambar 2 menunjukkan ilustrasi pengujian yang dilakukan di lab. Terdapat satu *virtual machine* yang ditunjuk sebagai *master* dan sisanya yaitu sebanyak 30 *virtual machine* berperan sebagai *slave*. Dari 30 *virtual machine* akan dibagi menjadi 4 kelompok yaitu Group *Slave* A yang terdiri dari 8 *virtual machine*, Group *Slave* B yang terdiri dari 6 *virtual machine*, Group *Slave* C yang terdiri dari 7 *virtual machine*, Group *Slave* C terdiri dari 9 *virtual machine*. Percobaan pada tahap ini hanya *virtual machine* yang berperan sebagai *slave* saja yang mengerjakan proses yang diberikan, *virtual machine* yang berperan sebagai *master* hanya membagi proses dari user dan menggabungkan hasil dari setiap *slave* dan dikirimkan lagi ke user dalam bentuk output sesuai dengan proses yang diberikan.

Berdasarkan hasil pengujian program dan pada Tabel 1, Tabel 2, dan Tabel 3, pertama argument atau parameter untuk NPEList (Normal PEList) yang diusulkan dapat meningkatkan efisiensi program, karena rata - rata running time program baik program A, B, C lebih singkat ketika menggunakan argument NPEList 0,75*threshold1 <= ALCi <= 1,25*threshold1 daripada menggunakan argument NPEList sebelumnya yaitu ALCi = threshold1. Hal ini disebabkan karena ketika menggunakan argument NPEList ALCi = threshold1 maka kondisi tersebut hampir tidak mungkin terpenuhi, sehingga pada semua hasil pengujian menggunakan argument tersebut tidak ada host yang dimasukkan pada NPEList sedangkan ketika menggunakan argument NPEList 0,75*threshold1 <= ALCi <= 1,25*threshold1 maka untuk kelompok NPEList tidak selalu kosong. Beberapa hasil percobaan menunjukkan running time program lebih singkat jika menggunakan argument NPEList ALCi = threshold1, akan tetapi hal ini disebabkan oleh faktor lainnya seperti koneksi jaringan karena *virtual machine* dijalankan di beberapa komputer sehingga terkadang terjadi *bottle neck* di jaringan komputer yang digunakan, dan dapat disebabkan juga karena proses – proses lainnya yang sedang berjalan pada waktu yang sama.

Algoritma *load balancing* PLBA dapat diimplementasikan cukup optimal dengan menggunakan PVM3, karena dapat menggunakan sumber daya yang terdistribusi secara optimal, meskipun sumber daya yang digunakan berjalan pada sistem operasi yang tidak sama semua dan dalam kondisi yang berbeda – beda, yaitu beberapa komputer menjalankan satu *virtual machine* dan beberapa lainnya menjalankan dua *virtual machine* sekaligus. PVM3 dapat digunakan sebagai media komunikasi antara sumber daya satu dengan sumber daya lainnya dalam suatu sumber daya grid, karena dapat menyelesaikan salah satu masalah dari sumber daya grid, yaitu sumber daya yang *heterogen* sehingga perlu adanya sebuah *compatibility layer* antara sumber daya satu dengan sumber daya lainnya. Ketika terdapat host yang masuk pada kelompok OPEList selalu ditambahkan pada log *Overloaded* PE, sehingga nilai threshold2 dapat digantikan dengan argument, jika OPEList tidak NULL maka log Overloaded PE ditambahkan sesuai dengan jumlah hosts yang ada di OPEList.

4. Kesimpulan

Algoritma load balancing PLBA telah berhasil

diimplementasikan pada lab environment, yaitu menggunakan lab komputer sebagai sumber daya Grid, dengan komputer – komputer sebagai sumber daya tersebut. Implementasi algoritma load balancing PLBA menggunakan PVM3, dapat mengoptimalkan sumber daya Grid yang digunakan, untuk menyelesaikan tasks yang diberikan. Algoritma load balancing PLBA juga telah berhasil dioptimalkan, dengan memodifikasi argument untuk NPEList. Hal ini ditunjukkan dengan rata – rata running time dalam menyelesaikan tasks, ketika menggunakan argument NPEList yang dimodifikasi (menggunakan argument NPEList 0,75*threshold1 <= ALCi <= 1,25*threshold1) lebih singkat, daripada ketika menggunakan argument NPEList pada penelitian sebelumnya (menggunakan argument NPEList ALCi = threshold1), karena pengelompokkan PE (Process Element) dengan argument NPEList dimodifikasi lebih optimal, daripada dengan menggunakan argument NPEList pada penelitian sebelumnya. Perbandingan rata-rata running time untuk argument NPEList asal dengan argument NPEList modifikasi adalah sebagai berikut: 82513.63740:67837.71720; 63869.92450:50722.17210; 858.96710 : 207.33680; 321.88000 :126.89100; 768.54560 : 468.27190; 780.22770 : 279.43730.

Daftar Rujukan

[1] Buyya, R. and Murshed, M., 2002, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," Concurrency and Computation: Practice and Experience, vol. 14, no. 13–15, pp. 1175–1220, DOI: 10.1002/cpe.710

[2] Menasce, D.A. and Casalicchio, E., 2004, "A framework for resource allocation in grid computing", In: The IEEE Computer Society's, 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, Volendam, Netherlands, 8-8 Oct. 2004, pp. 259–267, IEEE, DOI: 10.1109/MASCOT.2004.1348280

[3] Prajapati, H.B. and Shah, V.A., 2014, "Scheduling in Grid Computing Environment", In: The Institute of Electrical and Electronics Engineers, Inc., 2014 Fourth International Conference on Advanced Computing & Communication Technologies, Rohtak, India, 8-9 Feb. 2014, pp. 315–324, IEEE, Conference Publishing Services (CPS), DOI: 10.1109/ACCT.2014.32

[4] Goswami, S. and Sarkar, A.D., 2013, "A comparative study of load balancing algorithms in computational grid environment," In: The Institute of Electrical and Electronics Engineers, Inc., 2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation, Seoul, South Korea, 24-25 Sept. 2013, pp. 99–104, IEEE, Conference Publishing Services (CPS), DOI: 10.1109/CIMSim.2013.24

[5] Qilin, M. and Weikang, S., 2015, "A Load Balancing Method Based on SDN," In: ICMTMA 2015, 2015 Seventh International Conference on Measuring Technology and Mechatronics Automation, Nanchang, China, 13-14 June 2015, pp. 18–21, IEEE, DOI: 10.1109/ICMTMA.2015.13

[6] Jiao, Y. and Wang, W., 2010, "Design and Implementation of Load Balancing of Distributed-system-based Web Server," In: ISECS 2010, 2010 Third International Symposium on Electronic Commerce and Security, Guangzhou, 29-31 July 2010, pp. 337–342, IEEE, DOI: 10.1109/ISECS.2010.81

[7] Ohta, S. and Andou, R., 2009, "WWW server load balancing technique based on passive performance measurement," 2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, Pattaya, Chonburi, Thailand, 6-9 May 2009, pp. 884–887, IEEE, DOI: 10.1109/ECTICON.2009.5137187

[8] Patel, D.K., Tripathy, D., and Tripathy, C.R., 2016, "Survey of load balancing techniques for grid," Journal of Network and Computer Applications, vol. 65, pp. 103–119, DOI: 10.1016/j.jnca.2016.02.012

[9] Guan, H., Li, C.K., Cheung, T.Y., Yu, S. and Tong, W., 1996, "Design and implementation of a parallel software for hybrid neural network computation in PVM environment," Proceedings of Third International Conference on Signal Processing (ICSP'96), Beijing, China, 18-18 Oct. 1996, pp. 1421–1424, IEEE, DOI: 10.1109/ICSIGP.1996.566591

[10] Ponciano, J.P. and Anani, N., 2014, "Load balancing in modern network infrastructures — A simulation model", In: CSNDSP, 2014 9th International Symposium on Communication Systems, Networks & Digital Sign (CSNDSP), Manchester, UK, 23-25 July 2014, pp. 841–846, IEEE, DOI: 10.1109/CSNDSP.2014.6923944

[11] Rathore, N. and Chana, I., 2015, "Variable threshold-based hierarchical load balancing technique in Grid," Engineering with Computers., vol. 31, no. 3, pp. 597–615 DOI: 10.1007/s00366-014-0364-z

[12] Balasangameshwara, J. and Raju, N., 2012, "Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost," IEEE Transactions on Computers, vol. 62, no. 5, pp. 990–1003, DOI: 10.1109/TC.2012.44

[13] Nanthiya, D. and Keerthika, P., 2013, "Load balancing GridSim architecture with fault tolerance," In: ICICES, 2013 International Conference on Information Communication and Embedded Systems, Chennai, India, 21-22 Feb. 2013, pp. 425–428, IEEE, DOI: 10.1109/ICICES.2013.6508306

[14] Hao, Y., Liu, G., and Wen, N., 2012, "An enhanced load balancing mechanism based on deadline control on GridSim," Future Generation Computer Systems, vol. 28, no. 4, pp. 657–665, DOI: 10.1016/j.future.2011.10.010

[15] Setyawan, H.H., Widiarto, W. dan Wiharto, 2020, "Implementasi Algoritma Improvised Prioritized Deadline Scheduling Algorithm (IPDSA) pada Grid Environment", Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi), Vol. 4, No. 5, Oktober 2020, pp. 957-963, DOI: doi.org/10.29207/resti.v4i5.2457

[16] Sampath, S., Nanjesh, B.R., Sagar, B.B. and Subbaraya, C.K., 2014, "Performance optimization of PVM based parallel applications using optimal number of slaves," In: ICROIT, 2014 International Conference on Reliability Optimization and Information Technology, Faridabad, India, 6-8 Feb. 2014, pp. 388–392, IEEE, DOI: 10.1109/ICROIT.2014.67983