JEPIN
(Jurnal Edukasi dan Penelitian Informatika)
ISSN(e): 2548-9364 / ISSN(p) : 2460-0741

Vol. 7
No. 1
April 2021

Optimasi Algoritma *K-Means Clustering* dengan *Parallel Processing* menggunakan *Framework R*

Mastura Diana Marieska[1], Suci Lestari[2], Calvin Mahendra[3], Nabila Rizky Oktadini[4], Muhammad Ali Buchari[5]

Distributed System Research Group, Fakultas Ilmu Komputer, Universitas Sriwijaya
Jalan Palembang Prabumulih KM 32 Palembang Sumatera Selatan

[1]mastura.diana@ilkom.unsri.ac.id
[2]09021281722038@students.unsri.ac.id
[3]09021381722115@students.unsri.ac.id
[4]nabila@unsri.ac.id
[5]ali_buchari@ilkom.unsri.ac.id

Abstrak—***Parallel processing* sering digunakan untuk melakukan optimasi execution time terhadap algoritma data mining. Pada penelitian ini, *parallel processing* digunakan untuk melakukan optimasi pada algoritma clustering K-Means. Implementasi algoritma K-means dilakukan dengan memanfaatkan package yang tersedia pada *framework R*. Algoritma K-Means dijalankan secara serial dan parallel. Untuk mendapatkan persentase optimasi, maka dilakukan perbandingan antara execution time pada *parallel processing* dan execution time pada serial processing. Penelitian ini menggunakan dataset Boston Housing yang umum digunakan pada data mining. Skenario pengujian dibedakan berdasarkan jumlah *core* dan jumlah centroid. Hasil pengujian menunjukkan bahwa *parallel processing* untuk tiap skenario memiliki execution time yang lebih kecil daripada serial processing. Optimasi yang dihasilkan cukup signifikan, yakni bernilai 20% hingga 52%. Optimasi tertinggi didapatkan pada jumlah *core* terbanyak dan jumlah centroid terbesar.**

Kata kunci*— *Parallel Processing, Clustering, K-means, Framework R, Execution Time

I. PENDAHULUAN

Teknik *parallel processing* berkembang pesat seiring dengan berkembangnya teknologi *multicore processor*. Dengan adanya lebih dari satu *core* pada prosesor, maka prosesor dapat menjalankan lebih dari satu proses yang terpisah pada waktu bersamaan [1]. Hal ini meningkatkan performansi prosesor secara signifikan. Tidak perlu *hardware* khusus untuk mengimplementasikan strategi paralel karena hampir semua komputer modern telah memiliki lebih dari satu prosesor.

Dukungan *hardware* saja tidak cukup untuk dapat mengembangkan strategi paralel dengan baik. Diperlukan juga dukungan *software*. Beberapa bahasa pemrograman telah memiliki kemampuan paralel. Salah satu *framework* yang mendukung implementasi paralel adalah R. R merupakan *framework* untuk *data analytics* yang memudahkan programmer dalam *data manipulation*, perhitungan data, dan *graphical display* [2]. R dilengkapi dengan banyak *library* untuk analisis data yang dikembangkan secara *open source*.

Mengembangkan solusi paralel tidaklah mudah. Menambah *core* pada prosesor tidak serta merta membuat program berjalan lebih cepat [3]. Programmer perlu mengidentifikasi apakah suatu algoritma data mining berpotensi untuk dioptimasi dengan *parallel processing*. Salah satu indikasi algoritma dapat dioptimalkan dengan *parallel processing* adalah sedikitnya *dependency* antar data yang diproses secara bersamaan [4]. Apabila suatu algoritma memiliki tingkat keterkaitan yang tinggi maka pemrosesan terhadap data yang terkait tersebut perlu dilakukan secara serial.

Pada penelitian ini, *parallel processing* digunakan untuk melakukan optimasi terhadap algoritma K-Means. Algoritma K-Means merupakan algoritma untuk mengelompokkan sejumlah node menjadi beberapa *cluster*. Pengelompokan ini dilakukan dengan cara menghitung jarak antara node dengan pusat dari tiap *cluster*. Tiap node dikelompokkan pada *cluster* yang memiliki jarak terdekat. Pada K-Means, proses untuk mencari jarak pada masing-masing node dapat dilakukan terpisah. Tidak ada keterkaitan antar data dalam menghitung jarak. Proses pada tiap node dapat dilakukan secara bersamaan, sehingga dapat disimpulkan bahwa algoritma K-Means cocok untuk diimplementasikan secara paralel.

Dataset yang digunakan pada penelitian ini adalah data Boston Housing, yaitu data mengenai *housing market* di kota Boston, Amerika Serikat yang dikumpulkan oleh Statlib Library of Carnegie Mellon University [5]. Dataset ini sering dipakai pada penelitian mengenai *data mining* seperti pada penelitian prediksi *housing prices* [6] dan *regression* [7]. Algoritma K-Means diimplementasikan pada *framework* R menggunakan *library*. Dataset Boston Housing akan menjadi input pada program R untuk kemudian diproses menggunakan algoritma K-Means.

Submitted 28-11-2020; Revised 07-04-2021; Accepted 16-04-2021

Untuk mengamati seberapa besar pengaruh *parallel processing*, maka pada penelitian ini akan dilakukan pula pemrosesan data dengan *serial/sequential processing*. *Execution time* pada *serial processing* akan digunakan sebagai nilai dasar (*base*). *Execution time* pada *parallel processing* akan dihitung berdasarkan jumlah *core* yang digunakan, yaitu 2, 3, dan 4 *core*. Dengan membandingkan *execution time* pada *serial processing* dengan *parallel processing* maka dapat dianalisis seberapa banyak optimasi yang dihasilkan oleh *parallel processing.*

Telah banyak penelitian yang dilakukan untuk membandingkan *execution time* antara *serial processing* dengan *parallel processing*. Pada penelitian [8], dilakukan perbandingan *execution time* antara serial dan paralel dalam kasus *stemming* bahasa Indonesia. Hasilnya, solusi paralel berhasil mengurangi *execution time* hingga 98,45%. Penelitian [9] mengenai *multicore parallel* pada *geospatial visualizations* menghasilkan kesimpulan bahwa *parallel processing* lebih cepat sampai dengan 4x daripada *serial processing*

Parallel processing tidak selalu menghasilkan performansi yang lebih baik daripada serialnya. Seperti pada penelitian [10], solusi paralel dengan Graphical Proessing Unit (GPU) menghasilkan *execution time* yang lebih besar daripada *serial processing* pada CPU. Sehingga disimpulkan untuk jumlah data yang relatif kecil sebaiknya menggunakan *serial processing* pada CPU. Pada penelitian [11] dilakukan survey teoritis mengenai variasi *parallel package* yang tersedia pada R di tahun 2014. Penelitian mengenai implementasi paralel menggunakan *framework* R juga telah banyak dilakukan. Penelitan [12] menggunakan *parallel processing* pada R untuk *regression task*. Solusi paralel dengan *package* pada R berhasil memperoleh *execution time* terkecil.

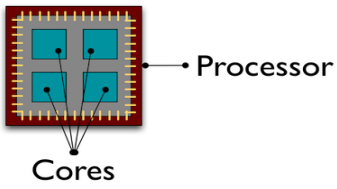
Berdasarkan analisis terhadap penelitian terkait, dapat dibuat hipotesis bahwa secara umum *parallel processing* dapat memperkecil *execution time*. Namun terdapat kasus tertentu dimana *serial processing* lebih cepat daripada *parallel processing*. Penelitian ini bertujuan untuk melakukan optimasi *execution time* terhadap algoritma *clustering* K-means. Selain itu, akan dihitung seberapa besar optimasi yang dihasilkan jika dibandingkan dengan *serial processing*.

II. METODE PENELITIAN

Pada penelitian ini dilakukan implementasi algoritma K-Means pada *framework* R dengan memanfaatkan package terkait *data mining* dan *multicore parallel*. Tersedianya *multicore processor* pada *hardware* yang digunakan memungkinkan pemrosesan data dilakukan secara bersamaan pada tiap *core*. Telah dilakukan analisis terhadap dataset dan library yang tepat untuk dipakai penelitian ini.

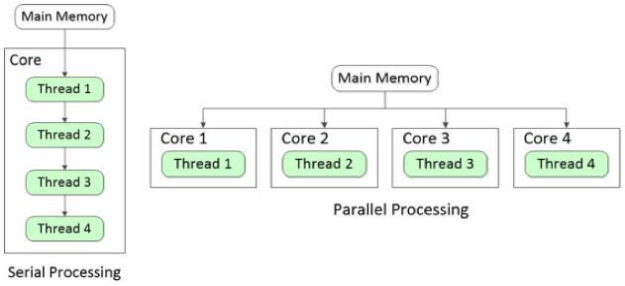
A. *Parallel Processing*

Sebuah prosesor merupakan inti dari tiap komputer. Komputer klasik memiliki hanya satu prosesor dengan 1 *core* di dalamnya. Seiring berkembangnya teknologi prosesor, prosesor masa kini dapat memiliki lebih dari satu *core*, atau yang disebut dengan *multicore*. Penambahan *core* ini menjadikan komputer mampu melakukan lebih dari 1 task dalam waktu yang sama. Dengan demikian performansi dari komputer dapat meningkat Suatu komputer yang memiliki satu prosesor yang terdiri atas 4 *core* dapat melakukan 4 komputasi dalam waktu bersamaan. Keempat *core* tersebut terkoneksi dengan *main memory* yang sama. Berikut ilustrasi struktur prosesor dan *core*.



Gambar. 1 *Multicore processor* [13]

Thread adalah *lightweight process*, yaitu bagian dari proses yang memiliki *creation cost* yang lebih ringan dan tidak memiliki alokasi memori sendiri [14]. Pada pemrosesan serial, suatu *thread* dapat dieksekusi setelah *thread* sebelumnya selesai dijalankan dan telah memperoleh hasil (*return*). Konsep dasar dari implementasi paralel yang membedakan dengan implementasi secara serial adalah *thread* pada paralel dapat dijalankan secara bersamaan pada satu waktu. Oleh karena itu, data yang digunakan pada tiap *thread* harus tidak saling berkaitan satu sama lain sehingga dapat diproses secara terpisah. Apabila data yang digunakan merupakan output dari proses sebelumnya maka kedua proses ini tidak dapat dijalankan secara paralel. Berikut adalah ilustrasi pemrosesan serial menggunakan 1 *core* dan paralel menggunakan 4 *core*.

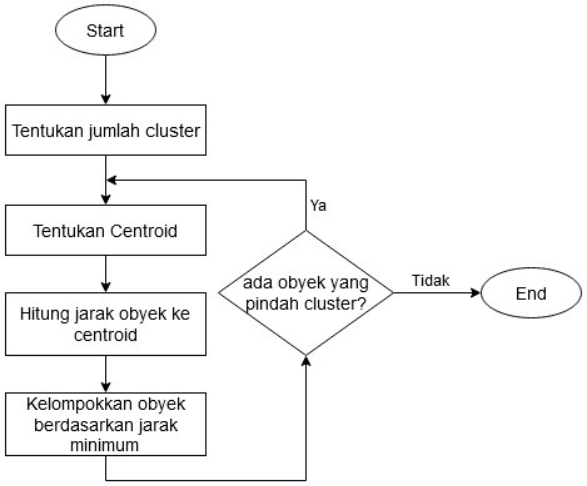


Gambar. 2 Skema serial processing dan *parallel processing*

B. *K-Means Clustering*

Data clustering merupakan teknik pengelompokan data ke dalam *cluster*, sedemikian sehingga data pada satu *cluster* memiliki kemiripan yang lebih tinggi dibanding data pada *cluster* lain [15]. *K-Means* merupakan salah satu metode *data clustering* non hirarki yang mempartisi data ke *cluster* yang bersesuaian [16]. Metode ini mempartisi data ke dalam *cluster* sehingga data yang memiliki karakteristik yang sama dikelompokkan ke dalam satu *cluster* yang sama dan data yang mempunyai karakteristik yang berbeda dikelompokkan ke dalam *cluster* yang lain.

Alur proses *K-Means clustering* dapat dilihat pada Gambar 3.



Gambar. 3 Alur proses *k-means* [16]

Pada penelitian ini digunakan dataset Boston Housing sebagai data yang akan diproses dengan algoritma K-Means. Dataset Boston Housing disediakan oleh perpustakaan StatLib yang dikelola oleh Universitas Carnegie Mellon. Dataset ini dibuat pada tahun 1993 dan telah menjadi dataset *well known* di kalangan peneliti [5] Data harga rumah di wilayah Boston terdiri atas 14 fitur, yaitu 13 variabel input dan 1 variabel output. Terdapat 506 baris data yang dikumpulkan oleh U.S Census Service terkait kebutuhan perumahan saat itu [6]. Dimensi dari dataset ini adalah 506x14.

C. Framework dan Library

R adalah bahasa pemrograman dan *framework* untuk *data analytics*. Bahasa R merupakan pengembangan dari bahasa S yang dikembangkan di AT&T Bell Laboratories oleh Rick Becker, John Chambers dan Allan Wilks [2]. R merupakan *framework* yang dapat dijalankan pada berbagai arsiektur komputer, seperti Intel, PowerPC, Alpha, dan Sparc. Pengembangan R saat ini dilakukan oleh *core team* yang terdiri dari sejumlah orang dari berbagai institusi di seluruh dunia [17]. *Framework* R juga dilengkapi GUI yang memudahkan pengguna untuk membuat, menjalankan, dan plot grafik. Salah satu *framework* R yang dilengkapi dengan GUI adalah RStudio.

Keunggulan dari *framework* R adalah tersedianya banyak *library* yang dikembangkan oleh komunitas di seluruh dunia. *Library* ini berbentuk *package* yang bersifat *free* dan open *source*. *Library* tersedia pada *repository* R yaitu CRAN. CRAN memiliki banyak *mirror server* di berbagai universitas dunia. Saat ini tersedia 16.385 *packages* pada R. *Package* pada R dapat berupa penambahan fitur, implementasi algoritma *data analytics*, maupun fitur pembuatan grafik dan visualisasi. Diperlukan eksplorasi dan analisis untuk menentukan *package* yang tepat untuk digunakan pada suatu penelitian. Pada penelitian ini digunakan *package* untuk implementasi *parallel processing*, implementasi *data clustering* menggunakan K-Means, dan pengaksesan dataset Boston Housing.

1) Library parallel: diperkenalkan oleh Luke Tierney dan tim R-*Core* untuk mendukung komputasi *parallel* di R. Versi pertama dari *library* ini disertakan dalam R versi 2.14.0 dan telah menjadi bagian dari inti dalam *package* R sejak saat itu [18]. *Library* parallel menggunakan *coarse-grained parallelization* menggunakan *fork thread* pada sistem operasi. Terdapat 3 method pada pada *library parallel* yang digunakan di penelitian ini, yaitu `detectCores()`, `lapply()`, dan `mclapply()`.

- `detectCores()` berfungsi untuk mendapatkan jumlah *core* dalam prosesor. Method ini mengembalikan nilai angka jumlah *core* yang terdeteksi. Berikut kode program untuk mendapatkan jumlah *core* dan ditampung pada variable `numCores`.

```
> numCores <- detectCores()
```

- `lapply()` berfungsi untuk melakukan operasi pada objek list dan mengembalikan objek list dengan panjang yang sama. Method ini mengembalikan list hasil eksekusi fungsi input. Berikut kode program untuk menjalankan `lapply()` dan menampung hasilnya pada variabel `res`.

```
> res <- lapply (data, fx)
```

Argumen :
`data` : Vektor atau objek yang ingin diolah
`fx` : Fungsi diterapkan ke tiap elemen data

- `mclapply()` untuk menjalankan komputasi paralel pada mesin dengan beberapa *core* atau CPU. Dengan fungsi ini, tugas dapat dibagikan pada seluruh *core* secara merata sehingga bisa berjalan bersamaan. Method `mclapply()` merupakan versi paralel dari `lapply()`, yang mengembalikan list dengan panjang yang sama dengan list masukan. Berbeda dgn `lapply()`, pada `mclapply()` terdapat argumen tambahan yaitu jumlah *core* yang ingin digunakan. Berikut kode program untuk menjalankan `mclapply()` dan menampung hasilnya pada variabel `res`.

```
> res<-mclapply (data, fx, mc.cores=4)
```

Argumen :
`data` : Vektor atau objek yang ingin diolah
`fx` : Fungsi diterapkan ke tiap elemen data
`mc.cores` : Jumlah *cores* yang digunakan

2) Library MASS: dikembangkan oleh Venables dan Ripley sebagai materi suplemen dari buku mereka yang berjudul "Modern Applied Statistics with S" (4th edition, 2002) [19]. Library MASS berisi fungsi dan dataset yang digunakan sebagai contoh dan latihan pada buku tersebut, salah satunya adalah dataset Boston Housing. Berikut adalah contoh penggunaan dataset Boston Housing yang menjadi salah satu parameter masukan pada suatu fungsi:

```
> kmeans(Boston, 5, nstart=nstart)
```

III. HASIL DAN PEMBAHASAN

Perbandingan *execution time* dilakukan terhadap *serial processing* dengan *parallel processing* dalam menjalankan algoritma K-Means untuk dataset Boston Housing. Algoritma K-Means dipilih karena dalam pencarian jarak antar node tidak ada keterkaitan data, sehingga dapat dilakukan pemrosesan secara terpisah. Data Boston Housing digunakan pada penelitian ini karena mudah didapat dan merupakan contoh dataset yang umum digunakan untuk penelitian di bidang *data mining*. Implementasi kedua program, baik serial maupun paralel dilakukan menggunakan bahasa R dengan menggunakan aplikasi RStudio. Kedua progam akan dijalankan pada komputer yang memiliki *multicore processor*. Tabel I merupakan spesifikasi *hardware* dan *software* dari komputer yang digunakan.

TABEL I
SPESIFIKASI *HARDWARE* DAN *SOFTWARE*

No	Hardware/Software	Spesifikasi
1	Processor	Intel(R) Core(TM) i5-5200U
2	Jumlah core pada processor	4
3	RAM	4 GB
4	Sistem Operasi	Ubuntu 20.04
5	RStudio	1.3.1073

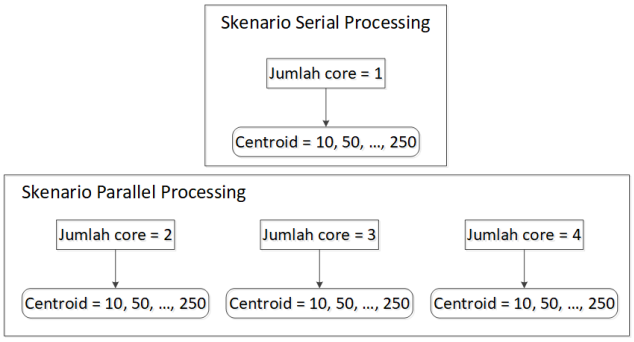
Agar dapat melalukan pengukuran *execution time*, dibuatlah sebuah proram uji yang menggunakan *library* parallel dan MASS. Pada program uji ini diimplementasikan suatu fungsi yang memanggil method `kmeans()` dengan masukan berupa dataset Boston Housing. Pencatatan *execution time* dilakukan dengan memakai fitur dasar R yaitu `system.time` yang menghitung berapa lama suatu fungsi dijalankan dalam satuan detik. Berikut adalah kode program uji menggunakan bahasa R.

```
Library (parallel)
library (MASS)

starts <- rep (1, 100)
fx <- function(nstart) kmeans(Boston, 5,
nstart=nstart)
numCores <- detectCores()

system.time(resserial <- lapply(starts, fx))
system.time(resparallel <- mclapply(starts,
fx, mc.cores = numCores))
```

Skenario pengujian dibedakan berdasarkan jumlah *centroid* untuk melihat keterkaitan jumlah *centroid* dengan optimasi paralel yang dihasilkan. Jumlah *centroid* yang digunakan adalah 10, 50, 100, 150, 200, dan 250. Dalam menjalankan suatu program, terdapat kemungkinan program tersebut akan diinterupsi oleh sistem operasi jika terdapat task lain dengan prioritas yang lebih tinggi. Untuk itu perhitungan *execution time* dilakukan sebanyak 5 kali dan dihitung nilai rata-ratanya/ mean. Gambar 4 adalah alur pengujian untuk membandingkan execution time serial processing dengan *parallel processing*.



Gambar. 4 Skenario pengujian

Skenario uji yang pertama adalah serial processing untuk 6 jenis centroid. Serial processing dijalankan hanya pada satu *core* dengan menggunakan method `lapply()`. *Execution time* diukur dalam satuan detik. Berikut adalah *execution time* untuk *serial processing* dengan n adalah jumlah centroid.

TABEL II
SKENARIO SERIAL *PROCESSING*

n	Percobaan ke					Mean
	1	2	3	4	5	
10	0,123	0,119	0,114	0,119	0,119	0,119
50	0,275	0,273	0,277	0,270	0,266	0,272
100	0,426	0,423	0,419	0,418	0,435	0,424
150	0,537	0,542	0,537	0,541	0,550	0,541
200	0,631	0,632	0,641	0,625	0,648	0,635
250	0,686	0,694	0,686	0,678	0,681	0,685

Dari hasil pengujian pada *serial processing*, dapat dilihat bahwa semakin bertambahnya jumlah *centroid*, maka bertambah pula *execution time*. Hal ini sesuai dengan logika dari algoritma K-Means, bahwa semakin banyak jumlah *centroid* maka semakin banyak jarak antar node yang harus dihitung. Selisih antar percobaan tidak terlalu besar. Misalnya pada jumlah centroid 50, selisih *execution time* terbesar adalah 11 *milisecond*. Dapat dikatakan bahwa method `lapply()` dijalankan secara konsisten karena tidak terdapat lonjakan *execution time*.

Setelah didapatkan nilai *execution time* untuk *serial processing* sebagai dasar perbandingan, maka selanjutnya dijalankan scenario uji untuk *parallel processing*. *Parallel processing* dijalankan menggunakan method `mclapply()` pada jumlah *core* yang berbeda-beda, yaitu 2, 3, dan 4 *core*.

TABEL III
SKENARIO *PARALLEL PROCESSING* 2 *CORE*

n	Percobaan ke					Mean
	1	2	3	4	5	
10	0,097	0,096	0,093	0,095	0,093	0,095
50	0,188	0,190	0,188	0,186	0,186	0,188
100	0,268	0,266	0,269	0,272	0,273	0,270
150	0,341	0,345	0,346	0,338	0,342	0,342
200	0,396	0,394	0,401	0,391	0,398	0,396
250	0,435	0,432	0,433	0,434	0,431	0,433

Jika dibandingkan dengan *serial processing*, maka pada *paralel processing* dengan 2 *core* ini telah terjadi optimasi *execution time*. Pada jumlah *centroid* 10, *execution time* pada *parallel processing* lebih cepat 24 *milisecond* atau 20%. Tidak terjadi lonjakan nilai *execution time* yang signifikan, sehingga dapat disimpulkan bahwa sama halnya dengan `lapply()`, method `mclapply()` juga dijalankan secara konsisten.

TABEL IV
SKENARIO *PARALLEL PROCESSING* 3 *CORE*

n	Percobaan ke					Mean
	1	2	3	4	5	
10	0,096	0,092	0,092	0,095	0,091	0,093
50	0,178	0,173	0,172	0,180	0,180	0,177
100	0,257	0,261	0,260	0,259	0,258	0,259
150	0,325	0,323	0,324	0,323	0,321	0,323
200	0,369	0,374	0,377	0,371	0,381	0,374
250	0,397	0,414	0,411	0,403	0,408	0,407

Execution time pada *parallel processing* dengan 3 *core* secara umum lebih kecil daripada *execution time* dengan 2 *core*, namun perbedaannya tidak terlalu besar. Selisih terbesar antara 2 *core* dan 3 *core* dihasilkan pada jumlah *centroid* terbanyak yaitu 120 *centroid*. Sedangkan pada jumlah *centroid* terkecil selisih *execution time* hanya 2 *milisecond*. Dapat disimpulkan bahwa menambah jumlah *core* tidak selalu menghasilkan optimasi yang besar.

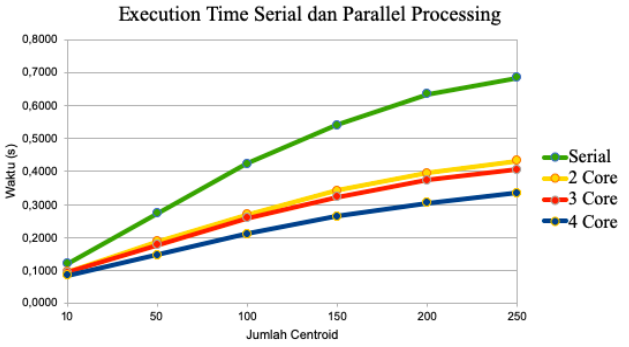
TABEL V
SKENARIO *PARALLEL PROCESSING* 4 *CORE*

n	Percobaan ke					Mean
	1	2	3	4	5	
10	0,085	0,085	0,088	0,078	0,082	0,084
50	0,146	0,147	0,146	0,146	0,145	0,146
100	0,209	0,210	0,209	0,208	0,214	0,210
150	0,262	0,263	0,263	0,267	0,262	0,263
200	0,368	0,306	0,308	0,299	0,302	0,317
250	0,331	0,333	0,330	0,331	0,327	0,330

Dapat dilihat pada Tabel V, bahwa *execution time parallel processing* dengan 4 *core* menghasilkan optimasi yang paling besar dibandingkan dengan jumlah *core* lainnya. Jika dibandingkan dengan 3 *core* maka dihasilkan optimasi paling besar sebanyak 73 *milisecond* yaitu pada jumlah *centroid* 250. Selanjutnya untuk analisis keseluruhan skenario, maka dibuat tabel dan grafik *execution time* tiap skenario dan tiap jumlah *centroid*. Tabel VI dan Gambar 5 merupakan perbandingan rata-rata *execution time* serial dan *parallel processing* untuk tiap jumlah *centroid*.

TABEL VI
PERBANDINGAN *EXECUTION TIME*

n	Jumlah *Core*			
	1 (Serial)	2	3	4
10	0,119	0,095	0,093	0,084
50	0,272	0,188	0,177	0,146
100	0,424	0,270	0,259	0,210
150	0,541	0,342	0,323	0,263
200	0,635	0,396	0,374	0,317
250	0,685	0,433	0,407	0,330



Gambar. 5 Grafik hasil pengujian serial dan *parallel processing*

Dengan menjadikan *execution time* pada *serial processing* sebagai dasar perbandingan, maka dapat dilakukan analisis optimasi. Pada jumlah *centroid* paling kecil, yaitu 10, optimasi *execution time* yang dihasilkan oleh *parallel processing* dengan 2 *core* adalah sebesar 20% Jika dijalankan dengan 4 *core* maka menghasilkan optimasi sebesar 29%. Semakin tinggi jumlah *centroid*, maka optimasi yang dihasilkan juga semakin besar. Penurunan *execution time* terjadi secara signifikan mulai dari jumlah centroid 100 atau lebih. Pada jumlah *centroid* 100, optimasi yang dihasilkan jika menggunakan 4 *core* adalah 50%. Optimasi tertinggi dihasilkan pada jumlah *centroid* terbesar, yaitu 250. *Parallel processing* pada jumlah *centroid* 250 menggunakan 4 *core* menghasilkan optimasi terbesar yaitu 52 %,

Jumlah *centroid* berpengaruh besar pada *execution time*. Semakin banyak jumlah *centroid*, maka semakin banyak waktu yang diperlukan untuk menjalankan algoritma K-Means. Hal ini disebabkan karena semakin banyak jumlah *centroid*, maka semakin banyak pula nilai jarak yang harus dihitung antar data dengan masing-masing *centroid*. Dari hasil pengujian, dapat dilihat bahwa *execution time* pada *parallel processing* untuk tiap kasus selalu lebih kecil daripada *execution time* pada *serial processing*. Dengan demikian maka dapat diambil kesimpulan bahwa implementasi strategi paralel berhasil melakukan optimasi dalam menjalankan algoritma K-Means.

IV. KESIMPULAN

Pada penelitian ini telah dilakukan optimasi algoritma *clustering* K-Means dengan menjalankan algoritma tersebut secara paralel pada *framework* R. Untuk mengukur seberapa tinggi optimasi yang berhasil dilakukan, maka dilakukan perbandingan *execution time* antara *serial processing* menggunakan method lapply dan *paralel processing* menggunakan method mclapply. Dari hasil percobaan didapatkan bahwa *parallel processing* dengan jumlah *core* lebih dari 2 memperoleh *execution time* lebih kecil dari pada *serial processing*. Optimasi tertinggi didapatkan pada saat *parallel processing* dijalankan pada *core* yang paling banyak, yaitu 4 *core*. Dengan demikian dapat disimpulkan bahwa makin banyak jumlah *core* maka makin kecil *execution time*.

UCAPAN TERIMA KASIH / ACKNOWLEDGMENT

Penelitian ini dibiayai oleh Universitas Sriwijaya pada Skema Hibah Sains Teknologi dan Seni tahun 2020.

REFERENSI

[1] B. Parhami, "Parallel Processing with Big Data," *Encycl. Big Data Technol.*, pp. 1–7, 2018.

[2] R. Harris, "An Introduction to R," *Quant. Geogr. Basics*, vol. 2, pp. 250–286, 2018.

[3] P. Pacheco, *An Introduction to Parallel Programming*. 2011.

[4] G. Xie *et al.*, "Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems," *IEEE Trans. Serv. Comput.*, vol. 13, no. 5, pp. 871–886, 2020.

[5] P. Kasap and B. Şeyda, "The Review of Attributes Influencing Housing Prices using Data Mining Methods," vol. 4531, pp. 155–165.

[6] A. Ali, "The Boston Housing Dataset Analysis University of Cumberlands," no. September, 2020.

[7] M. Hittmeir, A. Ekelhart, and R. Mayer, "Utility and Privacy Assessments of Synthetic Data for Regression Tasks," *Proc. - 2019 IEEE Int. Conf. Big Data, Big Data 2019*, pp. 5763–5772, 2019.

[8] N. Yusliani, R. Primartha, and M. Diana, "Multiprocessing Stemming: A Case Study of Indonesian Stemming," *Int. J. Comput. Appl.*, vol. 182, no. 40, pp. 15–19, 2019.

[9] C. Ledur, D. Griebler, I. Manssour, and L. G. Fernandes, "A High-Level DSL for Geospatial Visualizations with Multi-core Parallelism Support," *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 298–304, 2017.

[10] M. D. Marieska, M. R. P. Sufa, A. Widianto, N. Yusliani, and R. I. Heroza, "Performance Analysis of Parallel Processing on GPU for Simple Mathematical Computations," vol. 172, no. Siconian 2019, pp. 294–299, 2020.

[11] E. Mahdi, "A Survey of R Software for Parallel Computing," *Am. J. Appl. Math. Stat.*, vol. 2, no. 4, pp. 224–230, 2014.

[12] L. S. Riza, M. A. Ashari, and R. Megasari, "The Implementation of Gradient Descent Based Methods Using Parallel Computing in R for Regression Tasks," *Proceeding - 2018 Int. Symp. Adv. Intell. Informatics Revolutionize Intell. Informatics Spectr. Humanit. SAIN 2018*, pp. 37–42, 2019.

[13] M. Jones, "No Title," 2017. .

[14] V. M. Steen and A. S. Tanenbaum, *Distributed Systems*, 3rd ed. 2017.

[15] M. Kalra, N. Lal, and S. Qamar, "K-Mean Clustering Algorithm Approach for Data Mining of Heterogeneous Data," *Lect. Notes Networks Syst.*, vol. 10, no. January, pp. 61–70, 2018.

[16] Yudi Agusta, "K-Means – Penerapan, Permasalahan dan Metode Terkait," *J. Sist. dan Inform.*, vol. 3, no. Februari, pp. 47–60, 2007.

[17] H. Crc, *Data Mining with R-Kumar*. .

[18] R. September, "Package ' parallel ,'" pp. 1–14, 2020.

[19] A. Brian, B. Venables, D. M. Bates, D. Firth, and M. B. Ripley, "Package ' MASS ,'" 2020.

Korespondensi : Mastura Diana Marieska