Perbandingan Kinerja Infrastruktur Pararel Dalam Pemrosesan Data Dengan Menggunakan *Apache Spark*

Prayogi Kardani[1], I Made Suartana[2]

[1,2] Teknik Informatika, Fakultas Teknik, Universitas Negeri Surabaya
[1]prayogi.19055@mhs.unesa.ac.id
[2]madesuartana@unesa.ac.id

***Abstrak*—Perkembangan data yang pesat memerlukan infrastruktur pemrosesan yang efektif. Apache Spark, platform komputasi data, mampu memproses data besar melalui infrastruktur paralel. Evaluasi terhadap kinerja Spark diperlukan, mengacu pada penelitian sebelumnya yang menunjukkan peningkatan efisiensi. Penelitian ini membandingkan kinerja infrastruktur paralel Spark dalam pemrosesan data besar. Apache Spark diimplementasikan dalam sebuah cluster dengan 1 master dan 2 worker nodes untuk memproses data besar secara paralel. Penelitian ini mengevaluasi kinerja Apache Spark dalam pemrosesan data besar menggunakan cluster berkonfigurasi 1 master dan 2 worker nodes. Eksperimen menghasilkan temuan bahwa pada tahap counting, aggregation, dan filtering, cluster dengan 2 worker nodes menunjukkan peningkatan efisiensi yang signifikan, dengan waktu eksekusi lebih cepat dibandingkan dengan konfigurasi lainnya. Analisis penggunaan CPU menjelaskan bahwa cluster dengan satu master dan dua client mencapai penggunaan CPU yang lebih efisien, terutama pada worker nodes. Ditemukan bahwa penggunaan CPU pada master node tetap rendah, sementara worker nodes, khususnya pada mode cluster dengan dua client, dapat mengoptimalkan penggunaan CPU dengan tingkat yang lebih tinggi.**

***Kata Kunci*— *Apache Spark, Big Data, Dataset, Cluster,* Pemrosesan Data.**

I. PENDAHULUAN

Pada era digital saat ini, penggunaan data semakin meningkat seiring dengan perkembangan teknologi informasi. Data yang dihasilkan dari berbagai aplikasi seperti *e-commerce*, sistem manajemen inventori, dan sensor IoT semakin besar dan kompleks. Hal ini menyebabkan pengolahan data yang dilakukan secara tradisional tidak efektif dan membutuhkan waktu yang lama. Oleh karena itu, diperlukan infrastruktur paralel untuk mempercepat proses pemrosesan data besar [8].

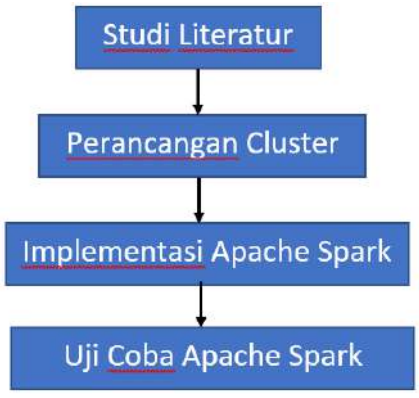
Apache Spark merupakan salah satu platform komputasi data yang dapat digunakan untuk memproses data besar dengan skala yang lebih besar dan kompleks. Infrastruktur paralel menggunakan Apache Spark dapat memproses data secara cepat dan efisien dengan menggunakan *multiple node* pada sebuah cluster. Namun, kinerja dari infrastruktur paralel menggunakan Apache Spark perlu dievaluasi untuk mengetahui kecepatan dan efisiensi pemrosesan data [5].

Tujuan penelitian ini adalah untuk melakukan perbandingan kinerja infrastruktur paralel menggunakan Apache Spark dalam memproses data besar. Penelitian ini akan memperhitungkan analisis kinerja infrastruktur paralel menggunakan Apache Spark. Diharapkan hasil penelitian ini dapat memberikan pemahaman yang lebih baik mengenai kinerja infrastruktur paralel menggunakan Apache Spark dan dapat digunakan sebagai dasar dalam pengembangan aplikasi yang memerlukan pemrosesan data besar. Penelitian ini juga diharapkan dapat memberikan kontribusi bagi pengembangan teknologi informasi khususnya dalam pengolahan data besar menggunakan infrastruktur paralel menggunakan Apache Spark.

Dari latar belakang di atas, maka peneliti ingin melakukan penelitian terkait Bagaimana kinerja infrastruktur paralel menggunakan Apache Spark dalam memproses data besar.

II. METODOLOGI PENELITIAN

A. *Tahapan Penelitian*



Gbr. 1 Tahapan Penelitian

Pada gambar 1 terdapat beberapa tahapan, pertama tahap Studi Literatur dimana pada tahap tersebut yang harus dilakukan yaitu menentukan topik penelitian yang akan diambil. Dalam hal ini, topik penelitian adalah "Infrastruktur Paralel Menggunakan Apache Spark". Setelah topik penelitian ditentukan, selanjutnya mencari referensi atau literatur yang relevan dengan topik penelitian. Referensi dapat dicari melalui buku, jurnal, artikel, website, atau sumber informasi lainnya yang terpercaya.

Beberapa referensi yang ditemukan terdapat pada latar belakang penelitian terdahulu. Setelah informasi terkumpul, selanjutnya menyusun kerangka teori. Kerangka teori adalah

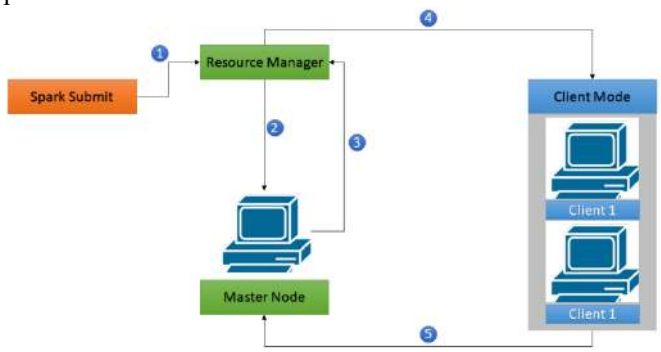
rangkaian konsep atau model yang menjelaskan topik penelitian secara lebih terperinci. Lalu menulis laporan studi literatur. Laporan ini harus mencakup semua informasi yang telah terkumpul dan harus disusun dengan cara yang sistematis dan terstruktur.

Studi literatur akan membantu peneliti untuk memperoleh pemahaman yang lebih baik tentang konsep-konsep dasar yang terkait dengan Apache Spark dan teknologi infrastruktur paralel yang digunakan. Tahap ini juga akan membantu peneliti untuk memilih dan merancang model infrastruktur paralel yang tepat.

B. *Perancangan Cluster*

Perancangan *cluster* adalah proses merancang dan membangun infrastruktur paralel menggunakan Apache Spark untuk memproses data secara terdistribusi pada beberapa node atau komputer dalam sebuah jaringan. Tujuan dari perancangan ini adalah untuk meningkatkan kecepatan pemrosesan data dan memperluas kapasitas penyimpanan data yang dapat diakses oleh sistem, dengan memanfaatkan kemampuan Apache Spark dalam memproses data secara paralel.

Dalam arsitektur ini, terdapat satu mesin yang berperan sebagai master dan dua mesin yang berperan sebagai *client* atau *worker*. master Spark sebagai pengelola cluster dapat memiliki kontrol penuh terhadap lingkungan distribusi, memastikan ketersediaan, keamanan, dan kinerja yang optimal untuk pemrosesan data besar.

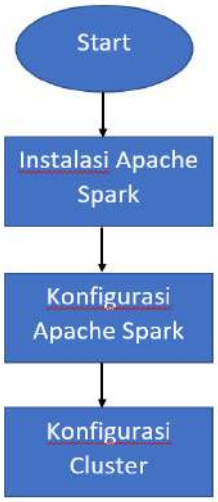


Gbr. 2 Topologi Cluster

Gambar 2 menjelaskan secara rinci bagaimana aplikasi Spark beroperasi dalam mode cluster, termasuk proses memulai mode cluster, mengajukan permintaan sumber daya, dan menjalankan tugas pada executor yang berjalan di client yang dialokasikan. Dalam mode cluster, driver dimulai pada node master, sementara Executor client menjalankan tugas yang diinstruksikan oleh master dan mengirimkan hasil kembali ke node master. Berikut penjelasan secara detailnya:

1) Dalam mode cluster, perintah spark-submit akan mengirimkan permintaan sesi Spark ke manajer sumber daya.
2) Manajer sumber daya, setelah menerima permintaan sesi Spark, memulai node master yang akan bertanggung jawab atas pengelolaan eksekusi pada cluster.
3) Setelah node master dimulai, master mengajukan permintaan kepada manajer sumber daya untuk mendapatkan lebih banyak perangkat yang akan berfungsi sebagai node client. Tujuan dari permintaan ini adalah untuk mendapatkan sumber daya tambahan guna menjalankan tugas secara terdistribusi.
4) Manajer sumber daya mengalokasikan dua node client untuk sesi Spark. Setiap node client ini berfungsi sebagai tempat di mana executor Spark dapat berjalan.
5) Setelah alokasi dua node client diperoleh, node master memulai executor pada setiap node client. Executor ini akan menjalankan tugas yang diberikan oleh master. Dengan dua node client, aplikasi Spark dapat menjalankan tugas secara lebih efisien, meningkatkan kinerja dan kemampuan pemrosesan data.

C. *Implementasi Apache Spark*



Gbr. 3 Implementasi Apache Spark

Pada gambar 3 menjelaskan tentang tahapan selanjutnya yaitu mengimplementasikan Apache Spark. Pada tahapan ini hal pertama yang dilakukan adalah instalasi Apache Spark dan perangkat lunak lainnya. Apache Spark ini berfungsi untuk memproses data besar. Dalam Tahapan ini setelah semua perangkat lunak telah terinstall dan Apache Spark telah berhasil digunakan. Selanjutnya akan membuat server Apache Spark yang dinamakan master node. Master node yang berhasil dikoneksikan selanjutnya, Buat file konfigurasi Spark, yang biasanya disebut "spark-env.sh". File ini akan digunakan untuk mengatur konfigurasi Spark, seperti jumlah *worker* yang akan digunakan. Edit file "spark-env.sh" menggunakan editor teks, dan tambahkan “export SPARK_WORKER_INSTANCES=2” untuk mengatur jumlah *worker*. Apache Spark diimplementasikan pada sebuah *cluster* yang terdiri dari 1 master dan 2 *client* atau *worker*.

Kemudian dilakukan pemrosesan sesuai dengan tujuan penelitian. Proses pemrosesan data dilakukan dengan menggunakan Apache Spark pada *cluster* yang telah diimplementasikan.

D. *Uji Coba Apache Spark*

Pada tahap uji coba kali ini. Data yang digunakan adalah data yang diambil dari sumber terbuka seperti *Kaggle*. Salah satu data yang akan diambil adalah dataset produk *e-commerce*. Proses load data pada *Apache Spark* nanti data besar yang di download dari internet akan ditempatkan pada local data storage dan akan diproses menggunakan *Apache Spark*.

Pengukuran Kinerja ini akan dibagi dalam tiga Skenario. Yang pertama akan diproses single node atau master saja. Lalu yang kedua akan diproses master dengan dua *client/slave node*. Yang ketiga data akan diproses oleh master dan satu *client/slave node*.

Pengukuran kinerja masing – masing skenario dilakukan menggunakan dua metrik yaitu waktu pemrosesan, dan persentase penggunaan CPU.

III. HASIL DAN PEMBAHASAN

A. *Deskripsi Data Penelitian*

Data yang digunakan dalam penelitian ini mencakup data harga item dari Mimovrste dalam rentang waktu 2021 hingga 2023. Mimovrste merupakan sebuah toko online yang berbasis di Slovenia. Mereka menyediakan berbagai produk elektronik, peralatan rumah tangga, pakaian, dan barang-barang lainnya. Mimovrste dikenal sebagai salah satu toko online terkemuka di Slovenia dan menyediakan layanan pengiriman ke berbagai wilayah. Data ini memiliki ukuran total sekitar 94,81 gigabyte. Penggunaan data berukuran 94.81 gigabyte ini dikarenakan untuk melihat kinerja pemrosesan data diperlukan data yang besar. Tetapi tidak ada ketentuan untuk ukuran data sebenarnya.

Sumber data ini diperoleh dari platform Kaggle. Yang dapat diakses secara publik pada halaman web Kaggle. Data diambil dari Kaggle, diunduh dari sumbernya, dan kemudian diimpor ke dalam sistem penelitian untuk persiapan dan analisis lebih lanjut. Berikut dataset sebelum saya proses menggunakan apache spark.

Gbr. 4 Dataset Sebelum Proses

Data yang dijelaskan di gambar 4, merupakan data harga item dari Mimovrste selama periode 2021-2023. Dalam konteks dataset yang besar ini, penerapan infrastruktur paralel seperti Apache Spark mungkin diperlukan untuk memastikan kinerja dan efisiensi pengolahan data yang optimal.

Gbr. 5 Dataset Show

Pada gambar 5 adalah gambar tampilan data saat sudah berhasil diproses pada apache spark. Gambar ini untuk memastikan bahwa data yang sudah di load tidak berubah isinya. Data yang di tampilkan adalah 20 data baris pertama yang ada pada dataset.

B. Konfgurasi Cluster Apache Spark

Pada bagian ini, saya melakukan konfigurasi cluster Apache Spark yang terdiri dari satu node master dan dua node client. Konfigurasi seperti ini sering digunakan untuk mendistribusikan pemrosesan data dan komputasi dalam lingkungan yang lebih besar. Sebelum melakukan konfigurasi cluster perlu mengetahui spesifikasi dari masing-masing perangkat yang akan digunakan untuk perbandingan kinerja nantinya. Spesifikasi perangkat dapat dilihat di tabel 1.

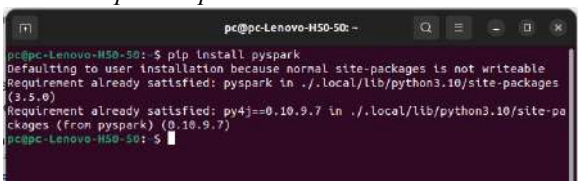
TABEL I

SPESIFIKASI NODE CLUSTER

	Master	Client 1	Client 2
Processor	I7-4970	I3-2120	I3-2120
Ram Memory	16 GB	6 GB	6 GB
Disk Capacity	256 GB	256 GB	256 GB

Dalam penjelasan ini, ada beberapa langkah-langkah yang perlu diambil untuk mengatur cluster Spark dengan konfigurasi tersebut.

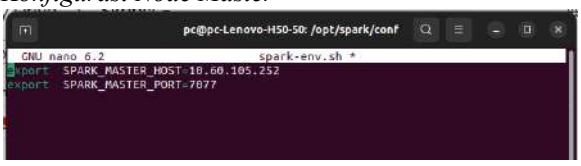
1) Instalasi Apache Spark



Gbr. 6 Instalasi Apache Spark

Langkah pertama seperti pada gambar 6 yang menjelaskan cara menginstal Apache Spark di semua node yang akan menjadi bagian dari cluster. Spark diunduh dan diinstal menggunakan versi 3.5.0 pada semua node. Selanjutnya, memastikan Spark diatur dalam PATH sistem agar perintahnya dapat diakses dengan mudah.

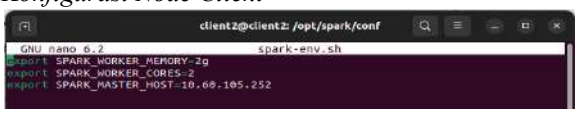
2) Konfigurasi Node Master



Gbr. 7 Konfigurasi Node Master

Pada gambar 7 menjelaskan tentang cara mengkonfigurasi node master. Node master adalah otak dari cluster. Konfigurasi node master melibatkan mengatur master URL dan konfigurasi lainnya. Edit file spark-env.sh pada node master untuk mengatur variabel lingkungan seperti SPARK_MASTER_HOST dan SPARK_MASTER_PORT.

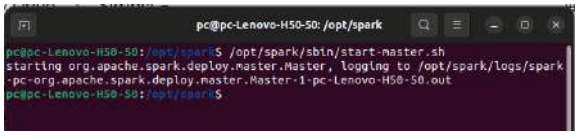
3) *Konfigurasi Node Client*



Gbr. 8 Konfigurasi Node Client

Gambar 8 menjelaskan tentang cara konfigurasi pada node client, selanjutnya mengkonfigurasi file spark-env.sh dan spark-defaults.conf. Di dalam spark-env.sh, menentukan SPARK_WORKER_CORES untuk mengatur jumlah core yang akan digunakan oleh setiap worker node. Di dalam spark-defaults.conf, mengatur konfigurasi khusus seperti penggunaan memory dan konfigurasi executor.

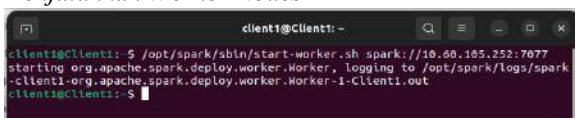
4) *Inisialisasi Cluster*



Gbr. 9 Start Node Master

Pada gambar 9 menjelaskan tentang inisialisasi node master dengan cara menjalankan perintah start-master.sh untuk memulai master Spark. Ini akan memberikan URL yang dapat digunakan oleh worker nodes untuk terhubung ke master.

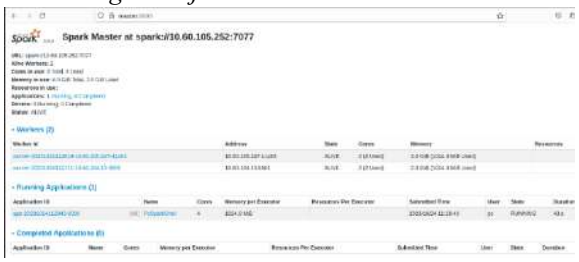
5) *Menjalankan Worker Nodes*



Gbr. 10 Start Client Node

Pada gambar 10 menjelaskan tentang menjalankan worker nodes pada dua node client dengan cara menjalankan perintah start-worker.sh dengan memberikan URL master yang diberikan oleh node master. Misalnya: start-worker.sh spark://master-node:7077.

6) *Monitoring Manajemen Cluster*



Gbr. 11 Monitoring Cluster

Pada gambar 11 menjelaskan tentang monitoring manajemen cluster setelah cluster diinisialisasi, selanjutnya dapat menggunakan Spark submit atau Pyspark untuk mengirimkan tugas pemrosesan data ke cluster. Kinerja cluster dapat dipantau dan melihat tugas yang berjalan pada antarmuka web Spark yang disediakan oleh node master.

Monitoring kerja cluster menggunakan Dashboard web pada “master:8080” untuk memantau dan mengelola sumber daya cluster. Ini akan membantu dalam alokasi sumber daya yang efisien.

Penting untuk selalu memastikan bahwa cluster Spark terkonfigurasi dengan benar dan sumber daya yang cukup untuk menangani tugas pemrosesan data. Selain itu, pemantauan dan pengelolaan cluster secara teratur baik untuk menjaga kinerja yang optimal. Dengan konfigurasi ini, dapat mendistribusikan beban pemrosesan data secara lebih efisien di seluruh cluster Spark Anda.

C. *Perbandingan Kinerja*

Pada perbandingan kinerja akan dilakukan pemrosesan data dengan tiga skenario yang berbeda. Pada seknario yang pertama data akan diproses menggunakan cluster yang berisi satu master dan dua client. Skenario kedua data akan diproses menggunakan master saja. Skenario yang ketiga data akan diproses menggunakan cluster satu master dan satu client.

Pemrosesan data menggunakan perintah Spark-Submit pada cluster Apache Spark. Ada beberapa perintah pada pemrosesan data yang akan dijadikan perbandingan nantinya. Yang pertama Operasi perhitungan (menghitung jumlah baris pada dataset). Operasi yang kedua yaitu operasi agregasi (menemukan harga rata-rata). Yang terakhir yaitu operasi pemfilteran (menyaring produk yang sedang promosi). Ketiga operasi tersebut dapat dijalankan dengan menggunakan kode menggunakan Bahasa pemrograman python. Berikut kode yang akan saya jalankan pada cluster Apache Spark dijelaskan pada gambar 12.

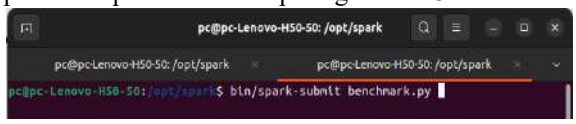
```
import time
import logging
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

spark = SparkSession.builder.appName("CSVProcessing").getOrCreate()

logging.basicConfig(filename="processing.log", level=logging.INFO)

csv_file_path = "data/mimcdump-dataset.csv"
start_time = time.time()
df = spark.read.csv(csv_file_path, header=True, sep=";")
end_time = time.time()
loading_time = end_time - start_time
logging.info(f"Loading time: {loading_time} seconds")

start_time = time.time()
row_count = df.count()
print("Row count:", row_count)

end_time = time.time()
counting_time = end_time - start_time

#logging.info (f"Counting time: {counting_time} seconds")

Aggregation Operation (e.g., find the average price)
start_time = time.time()

avg_price = df.agg({"price": "avg"}).collect()[0][0]
print("Average Price:", avg_price)
end_time = time.time()
aggregation_time = end_time - start_time

logging.info(f"Aggregation time: {aggregation_time} seconds")
start_time = time.time()
promotional_products =
df.filter(col("in_promotion") == "1")
promotional_products.show()end_time = time.time()
filtering_time = end_time - start_time

# Log filtering time
logging.info(f"Filtering time: {filtering_time} seconds")
user_input = input("Press Enter to stop the application...")
if user_input:
    # Stop the Spark session
     spark.stop()
```

Gbr. 12 Kode Node Cluster

Kode pemrosesan data disimpan dalam file berjudul benchmark.py. dan diproses langsung menggunakan perintah Spark-Submit. Dan dapat dipantau langsung pada dashboard Spark master “master:8080”.

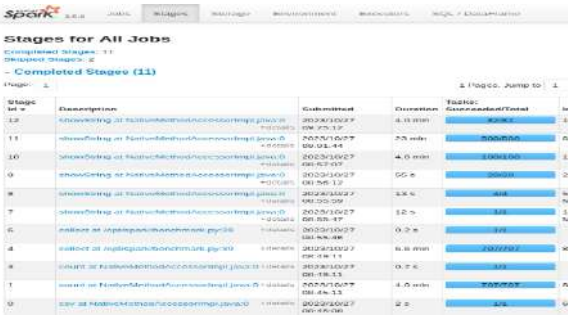
1) *Spark Cluster 1 Master 2 Client*

File benchmark.py dijalankan dengan menggunakan perintah Spark-Submit.Seperti gambar 13.



Gbr. 13 Spark-Submit Cluster

Data diproses pada cluster dan dapat dilihat perkembangan pemrosesan datanya pada dashboard Apache Spark Master seperti pada gambar 14.

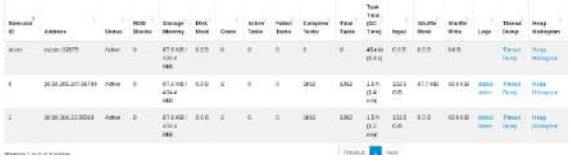


Gbr. 14 Dashboard Spark Cluster

Proses pemrosesan data pada cluster selesai dan dapat dilihat waktu yang dibutuhkan cluster untuk memproses tiga perintah pada data besar berukuran 94.81 gb secara detailnya yang dijelaskan pada gambar 15.

Gbr. 15 Duration Cluster

Beban executor pada cluster dapat dilihat pada dashboard executor cluster Apache Spark seperti pada gambar 16.



Gbr. 16 Executor Cluster

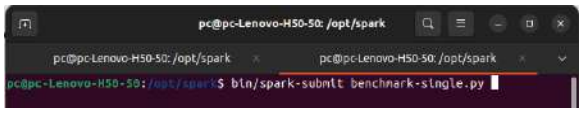
2) *Spark Master Node*

Pada skenario yang kedua yaitu pemrosesan data pada spark master atau single node. Disini pada cluster spark yang bekerja hanyalah master nodenya saja. Pada skenario ini, kode pada file benchmark.py, ditambahkan perintah agar yang bekerja hanyalah master node saja. File di simpan ulang dengan mengganti nama menjadi benchmark-single.py. Perintah yang ditambahkan pada file benchmark-single.py dapat dilihat pada gambar 17.

```
# Initialize a Spark session
spark =
SparkSession.builder.appName("SingleCoreSp
ark").master("local[2]").getOrCreate()
```

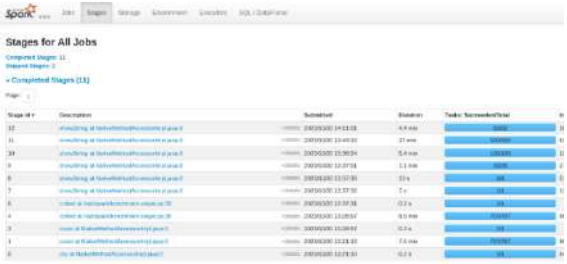
Gbr. 17 Kode Master Node

Pemrosesan data pada skenario kedua dapat diproses langsung menggunakan perintah Spark-Submit. Seperti gambar 18.



Gbr. 18 Spark-Submit Master

Data diproses pada cluster dan dapat dilihat perkembangan pemrosesan datanya pada dashboard Apache Spark Master Node "localhost:4040" seperti pada gambar 19.



Gbr. 19 Dashboard Spark Mode

Proses pemrosesan data pada master node selesai dan dapat dilihat waktu yang dibutuhkan Master Node untuk memproses tiga perintah pada data besar berukuran 94.81 gb, secara detailnya dapat dilihat pada gambar 20.

Gbr. 20 Duration Master Node

Selain itu juga dapat dilihat beban executor Master pada dashboard executor seperti gambar 21.

Gbr. 21 Executor Master Node

3) Spark Cluster 1 Master 1 Client

Pada skenario yang ketiga yaitu pemrosesan data pada spark menggunakan cluster spark 1 master dan 1 client. Disini pada cluster spark yang bekerja hanyalah 1 client nodenya saja. Pada skenario ini, kode pada file benchmark.py, ditambahkan perintah agar yang bekerja hanyalah 1client node saja. File di simpan ulang dengan mengganti nama menjadi benchmark-singleclient.py. Kode yang ditambahkan pada file benchmark-singleclient.py dapat dilihat pada gambar 22.

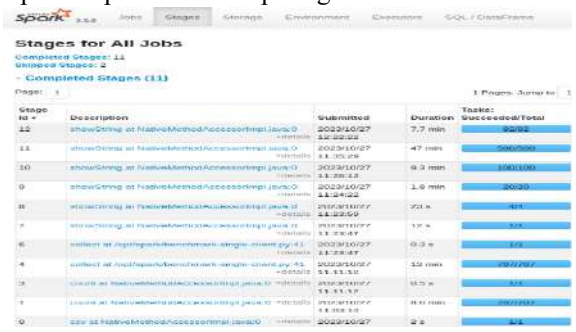
```
# Initialize a Spark session
spark =
SparkSession.builder.appName("SingleCoreSpark").mast
er("local[2]").getOrCreate()
```

Gbr. 22 Kode Satu Client Node

File benchmark-singleclient.py dijalankan dengan menggunakan perintah Spark-Submit seperti gambar 23.

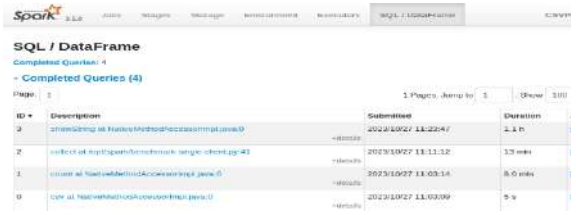
Gbr. 23 Spark-Submit Client

Data diproses pada cluster dan dapat dilihat perkembangan pemrosesan datanya pada dashboard Apache Spark Master seperti gambar 24.



Gbr. 24 Dashboard Spark Client Node

Proses pemrosesan data pada cluster selesai dan dapat dilihat waktu yang dibutuhkan cluster untuk memproses tiga perintah pada data besar berukuran 94.81 gb secara detailnya dapat dilihat pada gambar 25.



Gbr. 25 Duration Client Node

Beban executor pada cluster dapat dilihat pada dashboard executor cluster Apache Spark seperti pada gambr 26.

Gbr. 26 Executor Client Node

D. Hasil Eksperimen

Dalam tahap ini, Saya akan menyajikan hasil eksperimen yang mencakup tabel perbandingan antara tiga skenario yang berbeda dalam pemrosesan data menggunakan Apache Spark. Eksperimen pemrosesan data dilakukan lima kali tiap skenarionya. Hasil eksperimen ini dirancang untuk memberikan gambaran yang jelas tentang perbandingan kinerja dan efisiensi antara pengolahan data pada cluster satu master node dua client node, satu node menggunakan master node, dan cluster menggunakan 1 client node.

1) Waktu Eksekusi

Waktu eksekusi didapat dari dashboard Apache Spark. Dapat dilihat secara detail informasi berapa lama Cluster memproses data besar menggunakan Apache Spark. Setiap skenario nya di lakukan percobaan sebanyak lima kali. Berikut hasil percobaan tiap skenarionya:

a) Cluster 1 Master 2 Client

Pada Cluster satu master dan dua client ini dilakukan lima kali pemrosesan data. Data yang selesai di proses, waktu pemrosesannya akan terlihat jelas pada dashboard Apache Spark. Berikut hasil dari kelima percobaan Apache spark menggunakan cluster satu master dan dua client:

TABEL II

WAKTU EKSEKUSI CLUSTER

Proses Data	Percobaan Ke-					Rata-rata
	1	2	3	4	5	
Counting Time	4	4.1	4	4	4	4 menit
Aggregation Time	6.6	6.1	6.2	6.3	6.5	6.3 menit
Filtering Time	33	34	34	34	35	34 menit

Pada tabel 2 menjelaskan hasil rata-rata waktu eksekusi yang dibutuhkan untuk memproses data menggunakan cluster satu master dan dua client Apache Spark. Rata rata diambil dari penjumlahan tiap node dan dibagi dengan lima.

b) Master Node

Pada master node ini juga dilakukan lima kali pemrosesan data. Data yang selesai di proses, waktu pemrosesannya akan terlihat jelas pada dashboard Apache Spark. Berikut hasil dari kelima percobaan Apache spark menggunakan master node:

TABEL III

WAKTU EKSEKUSI MASTER

Proses Data	Percobaan Ke-					Rata-rata
	1	2	3	4	5	
Counting Time	7.6	7.8	7.6	7.6	7.8	7.7 menit
Aggregation Time	8.7	8.6	8.6	8.5	8.5	8.6 menit
Filtering Time	41	37	38	38	37	38.2 menit

Pada tabel 3 menjelaskan hasil rata-rata waktu eksekusi yang dibutuhkan untuk memproses data menggunakan master node Apache Spark. Rata rata diambil dari penjumlahan tiap node dan dibagi dengan lima.

c) Cluster 1 Master 1 Client

Pada Cluster satu master dan satu client ini dilakukan lima kali pemrosesan data. Data yang selesai di proses, waktu pemrosesannya akan terlihat jelas pada dashboard Apache Spark. Berikut hasil dari kelima percobaan Apache spark menggunakan cluster satu master dan satu client:

TABEL IV

WAKTU EKSEKUSI SATU CLIENT

Proses Data	Percobaan Ke-					Rata-rata
	1	2	3	4	5	
Counting Time	8.0	8.0	8.0	7.9	8.0	8.0 menit
Aggregation Time	13	13	13	13	13	13 menit
Filtering Time	66	66	66	66	66	66 menit

Pada tabel 4 menjelaskan hasil rata-rata waktu eksekusi yang dibutuhkan untuk memproses data menggunakan cluster satu master dan satu client Apache Spark. Rata rata diambil dari penjumlahan tiap node dan dibagi dengan lima.

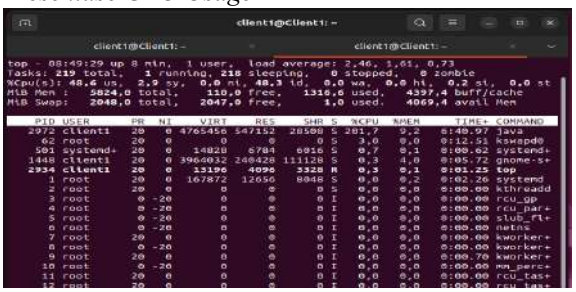
TABEL V

RATA-RATA WAKTU EKSEKUSI

	Cluster 1 master 2 Client	Cluster 1 master	Cluster 1 Client
Counting Time	4 menit	7.7 menit	8 menit
Aggregation Time	6.3 menit	8.6 menit	13 menit
Filtering Time	34 menit	38 menit	66 menit

Pada tabel 5 menjelaskan hasil rata-rata waktu eksekusi yang dibutuhkan untuk memproses data menggunakan apache spark. Dapat diketahui bahwa perbandingan waktu pemrosesan data pada 3 skenario yang berbeda. Pemrosesan data menggunakan cluster Spark lebih cepat daripada pemrosesan data single node menggunakan master node dan cluster dengan satu client node Apache Spark.

2) Presentase CPU Usage



Gbr. 27 CPU Usage

Pada gambar 27 menjelaskan presentase CPU usage yang diperoleh dari fungsi ubuntu pada terminal. Jalankan fungsi top atau htop untuk mengetahui berapa presentase CPU yang dipakai saat memproses data menggunakan Apache Spark.

TABEL VI

PRESENTASE CPU USAGE

Kategori	Ke-	Master Node	Client Node 1	Client Node 2
Cluster Satu Master Dua Client	1	1.3 %	48.6 %	48.3 %
	2	1.0 %	48.4 %	48.5 %
	3	1.1 %	48.6 %	48.2 %
	4	1.5 %	47.9 %	48.1 %
	5	1.1 %	48.6 %	48.7 %
Master Node	1	13.2 %	0%	0%
	2	13.1 %	0%	0%
	3	13.3 %	0%	0%
	4	12.9 %	0%	0%
	5	13.0 %	0%	0%
Cluster Satu Master Satu Client	1	1.2 %	32.2 %	0%
	2	1.0 %	33.4 %	0%
	3	1.0 %	35.4 %	0%
	4	1.0 %	34.1 %	0%
	5	1.0 %	36.2 %	0%

Pada tabel 6 menjelaskan persentase penggunaan CPU yang dibutuhkan untuk memproses data menggunakan apache spark. Penggunaan CPU pada pemrosesan data ini diambil 5 kali pada setiap skenrio nya dapat diketahui bahwa perbandingan penggunaan CPU pada 3 skenario yang berbeda.

TABEL VII

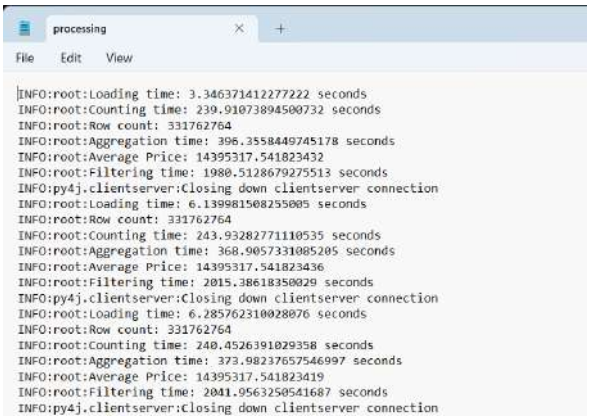
RATA-RATA CPU USAGE

	Cluster 1 master 2 Client	Cluster 1 master	Cluster 1 Client
Master Node	1.2 %	13.1 %	1.1 %
Client Node 1	48.4 %	0%	34.3 %
Client Node 2	48.4 %	0 %	0%

Pada tabel 7 menjelaskan rata – rata persentase CPU Usage. Rata rata diambil dari hasil pemrosesan dibagi dari banyaknya percobaan yaitu lima kali percobaan. Pemrosesan data menggunakan cluster Spark lebih memaksimalkan ram daripada pemrosesan data single node menggunakan master node dan cluster node dengan 1 client node Apache Spark.

3) Hasil Pemrosesan Data

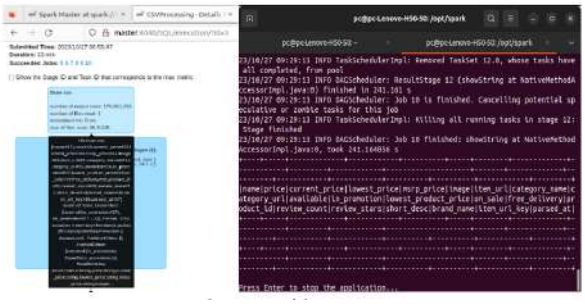
Data yang diproses menggunakan Apache Spark sesuai dengan kode yang ada tersimpan dalam file "processing.log". File tersebut berisi hasil dari perintah row count dan average price, Sedangkan filtering time dapat dilihat di terminal ubuntu dan juga dashboard apache spark. Pada file tersebut juga tertera berapa waktu yang dibutuhkan masing-masing perintah pemrosesan.



Gbr. 28 Processing.log file

Pada gambar 28 menjelaskan hasil dari pemrosesan data besar menggunakan apache spark. Dapat dilihat bahwa hasil yang diberikan konsisten. Pada jumlah baris yang dihitung tetap pada 331.762.764 baris. Dan pada rata-rata harga menunjukkan angka yang sama di tiap pemrosesan dan tiap skenario nya yaitu 14.395.317,54.

Pada perintah filtering mencari pada file csv pada kolom in_promotion. Mencari berapa produk yang termasuk produk promosi. Dan dari hasil penelitian saya tidak ditemukannya produk promosi pada dataset tersebut. Dan dari perintah filter tadi dikembalikan lagi untuk mencari filter pada kolom lain dengan menampilkan kolom yang tersedia pada dataset tersebut.



Gbr. 29 Filtering

Pada gambar 29 menjelaskan hasil dari proses filtering. Yang didapat dari proses ini dari tiap lima kali percobaan tiap skenarionya tetap sama dan tidak berubah.

IV. KESIMPULAN

Bedasarkan hasil dari analisis, implementasi, dan pengujian yang sesuai dengan rancangan sebelumnya dapat diambil kesimpulan bahwa Hasil percobaan menunjukkan kinerja infrastruktur Spark dalam pemrosesan data. Rata-rata waktu eksekusi pada eksekusi (counting time) Spark cluster dengan 2 worker adalah 4 menit, sedangkan pada 1 worker node dan master node masing-masing memerlukan waktu 8 menit dan 7.7 menit. Pada eksekusi (aggregation time), Spark cluster dengan 2 worker memerlukan waktu 6.3 menit, sementara pada

1 worker node dan master node masing-masing membutuhkan waktu 13 menit dan 8.6 menit. Untuk eksekusi (filtering time), Spark cluster dengan 2 worker membutuhkan waktu 34 menit, sedangkan pada 1 worker node dan master node masing-masing memerlukan waktu 66 menit dan 38 menit.

Dalam mode cluster dengan satu master dan dua client, penggunaan CPU pada master node cenderung rendah, sekitar rata-rata 1.2% dalam lima kali percobaan. Pada konfigurasi cluster satu master dan satu client, master node tetap menunjukkan penggunaan CPU yang rendah, sekitar rata-rata 1.1% pada 5 kali percobaan. Ketika master node memproses data secara mandiri, penggunaan CPU pada perangkat master dalam lima kali percobaan hanya sekitar 13.1%, meskipun persentasenya masih rendah, namun optimal sesuai dengan spesifikasi perangkat master yang lebih unggul dibandingkan dengan perangkat client. Dengan demikian, dapat disimpulkan bahwa penggunaan CPU lebih optimal ketika cluster satu master dan dua client beroperasi.

REFERENSI

[1] Jeremia, D., Novianus, H., & Gunawan, A. (2022). *Platform Big Data Analytic Berbasis Apache Spark Bagi Pemula Dalam Menyusun Data Analysis Workflow.*

[2] Sitepu, H., Zefanya, C., & Hutagalung, M. (2018). *Analisis Big Data Berbasis Stream Processing Menggunakan Apache Spark. Jurnal Telematika.*

[3] Budi Cahyono, E. (2019). *Pengukuran Performa Apache Spark Dengan Library H2O Menggunakan Benchmark Hibench Berbasis Cloud Computing Measuring Performance Apache Spark With Library H2O Using Benchmark Hibench Based Cloud Computing.*

[4] Rakhmat Purnomo, Wowon Priatna, Tri Dharma Putra (2021) *View of Implementasi Big Data Analytical Untuk Perguruan Tinggi Menggunakan Machine Learning*.

[5] Wiyaja. (2018). *Apache Spark: Perangkat Lunak Analisis Terpadu untuk Big Data*. http://www.teknologi-bigdata.com/2018/08/apache-spark-perangkat-lunak-analisis-bid-data.html

[6] Wali. (2023). *Penerapan dan Implementasi Big Data di Berbagai Sektor*. PT. Sonpedia Publishing Indonesia.

[7] Cahyo. (2018). *Studi dan Implementasi Apache Spark MLlib untuk Analisis Big Data*. https://repository.unpar.ac.id

[8] Muttaqina. (2020) *Analisa Komputasi Paralel Mengurutkan Data dengan Metode Radix dan Selection*. Jurnal Komputasi. Universitas Lampung, Bandar Lampung.

[9] Umam. (2018). *Data Mining dan Big Data Analytics : Teori dan Implementasi Menggunakan Phyton & Apache Spark.* Penebar Media Pustaka.

[10] Wicaksono. (2021). *Sistem Komputasi Paralel Menggunakan GPU Berbasis Nvidia Cuda*. Majalah Ilmiah Pengempangan Rekayasa dan Sosial. Jurnal Polines, 17(2), 158-164.

[11] Febriani. (2018). *Studi dan Perbandingan Apache Spark SQL dan Hive Dalam Konteks Analisis Big Data*. UNPAR Institutional Repository.

[12] Indra. (2021). *Analisis Big Data dengan Metode Exploratory Data Analysis (EDA) dan Metode Visualisasi Menggunakan Jupyter Notebook*. Jurnal Sistem Informasi dan Ilmu Komputer Prima, 4(2).

[13] Wibawa. (2022). *Komparasi Kecepatan Hadoop Mapreduce dan Apache Spark Dalam Mengolah Data Teks*. Jurnal Ilmiah Matriks, 24(1).

[14] Amron. (2020). *Pengembangan Platform Pengolahan Data Sensor Internet of Things Berjenis Streaming dengan Komputasi Terdistribusi Menggunakan Spark Streaming*. Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 4(7), 2102-2110.

[15] Maryanto. (2022). *Perancangan Komputasi Paralel Determinan dan Balikan Matriks Menggunakan Teknik Analisis Geometri Komputasional*. Jurnal Teknik Informatika, 3(1), 91-98.