BAB V
KOMPUTASI TERDISTRIBUSI PADA DATASET BESAR MENGGUNAKAN PYSPARK

5.1. Tujuan Pembelajaran

Setelah mengikuti praktikum yang dijelaskan didalam modul praktikum ini, mahasiswa diharapkan mampu:

1. Menulis dan mengeksekusi job PySpark untuk melakukan komputasi terdistribusi pada dataset besar.
2. Memahami dasar-dasar pemrograman RDD (*Resilient Distributed Datasets*) dan DataFrame di PySpark.

5.2. Pendahuluan

Komputasi terdistribusi merupakan pendekatan dalam pemrosesan data yang memanfaatkan banyak komputer atau *node* yang bekerja secara paralel. Dengan cara ini, pekerjaan besar yang sulit ditangani oleh satu mesin dapat dibagi menjadi bagian-bagian kecil yang dikerjakan secara bersamaan. Konsep ini menjadi fondasi dalam pengolahan data skala besar atau big data. *Apache Spark* hadir sebagai salah satu *framework* komputasi terdistribusi yang populer, dirancang untuk menangani pemrosesan data dalam jumlah besar dengan kecepatan tinggi. Spark memiliki keunggulan dibandingkan framework pendahulunya, seperti Hadoop MapReduce, karena mampu melakukan komputasi in-memory dan mendukung berbagai API untuk pemrograman data.

PySpark adalah antarmuka Python untuk Apache Spark yang memudahkan pengguna dalam menulis kode dengan sintaks Python. Dengan PySpark, pengembang dapat memanfaatkan kekuatan Spark tanpa harus menggunakan bahasa Scala atau Java. Hal ini menjadikan PySpark sebagai salah satu alat penting dalam analisis data modern, khususnya ketika berhadapan dengan dataset yang sangat besar. Salah satu kemampuan utama PySpark adalah menjalankan job untuk pemrosesan data dalam skala besar. Job PySpark biasanya terdiri dari serangkaian transformasi dan aksi pada data yang terdistribusi di cluster. Penulisan job yang baik memungkinkan data diproses secara efisien dengan memanfaatkan paralelisme yang ditawarkan Spark. Dalam praktiknya, job PySpark dapat berupa perhitungan statistik sederhana, pembersihan data, hingga analisis kompleks pada dataset yang sangat besar. Proses ini melibatkan pemahaman alur kerja Spark, mulai dari pembacaan data, penerapan transformasi, hingga pengumpulan hasil akhir.

RDD (*Resilient Distributed Datasets*) adalah abstraksi data dasar di Spark yang merepresentasikan kumpulan data terdistribusi yang bersifat fault-tolerant. Dengan RDD, data dapat diproses melalui transformasi dan aksi, sehingga sangat fleksibel untuk berbagai macam operasi. Namun, penggunaan RDD murni memerlukan pemahaman lebih mendalam tentang detail teknis pemrograman paralel. Untuk memudahkan analisis data, Spark juga menyediakan DataFrame, yaitu representasi data terstruktur dengan skema yang mirip tabel. DataFrame memungkinkan pengguna untuk bekerja dengan data menggunakan operasi deklaratif, mirip SQL. Dengan cara ini, pengolahan data menjadi lebih sederhana dan efisien, terutama untuk analisis data tabular.

Perbedaan utama antara RDD dan DataFrame terletak pada tingkat abstraksi dan optimasi. RDD memberikan fleksibilitas tinggi tetapi membutuhkan lebih banyak kode, sedangkan DataFrame memanfaatkan Spark SQL Catalyst Optimizer untuk meningkatkan performa. Dengan memahami keduanya, mahasiswa dapat memilih pendekatan yang sesuai untuk setiap kasus penggunaan. RDD menyediakan dua jenis operasi utama: transformasi dan aksi. Transformasi menghasilkan RDD baru dari RDD yang ada, sementara aksi menghasilkan nilai atau data ke driver program. Pemahaman tentang perbedaan keduanya sangat penting untuk menulis kode PySpark yang efisien. Pada sisi lain, DataFrame menawarkan operasi yang lebih sederhana, seperti filter, select, groupBy, dan join. Operasi-operasi ini mempermudah pengguna untuk mengolah data tabular tanpa perlu menuliskan transformasi manual seperti pada RDD. Dengan cara ini, analisis data dapat dilakukan lebih cepat.

Komputasi terdistribusi pada PySpark menghasilkan output yang perlu dianalisis untuk mendapatkan insight yang bermakna. Analisis ini tidak hanya sebatas mengeksekusi job, tetapi juga menafsirkan hasilnya agar sesuai dengan kebutuhan bisnis atau penelitian. Dalam praktiknya, hasil komputasi dapat berupa ringkasan statistik, pola dalam data, atau dataset baru yang siap digunakan untuk tahap analisis lanjutan. Kemampuan mahasiswa dalam membaca dan memahami output ini menjadi kunci keberhasilan dalam memanfaatkan PySpark.

5.3. Persiapan Lingkungan

Sebelum melakukan Praktikum mengenai Komputasi Terdistribusi Pada Dataset Besar Menggunakan Pyspark, pastikan perangkat sudah memenuhi syarat minimum berikut:

1. Perangkat Keras

- Sistem Operasi Windows 10/11

- Laptop/PC dengan minimal RAM 8 GB (lebih disarankan 16 GB untuk simulasi dataset besar).
- Prosesor multi-core (disarankan ≥ 4 core).
- Penyimpanan 20 GB kosong untuk instalasi Spark, dan dataset.
- Koneksi internet untuk mengunduh dataset dan library.

2. Perangkat Lunak

- Python 3.11 atau lebih baru.
- Apache Spark 3.x (3.5.6 atau lebih)
- IDE/Editor: Jupyter Notebook, VSCode, atau PyCharm.
- Instal Library

```
Pip install findspark
```

3. Dataset

- Dataset yaitu HR_DATA_MNC (ukuran > 200MB) : https://s.id/dataset_HR_MNC

5.4. Langkah Praktikum

a. Inisialisasi *SparkSession*

Inisialisasi *SparkSession* adalah langkah awal yang wajib dilakukan ketika bekerja dengan PySpark, karena *SparkSession* berfungsi sebagai gerbang utama untuk berinteraksi dengan *Apache Spark*. Melalui *SparkSession*, pengguna dapat membuat, mengakses, dan mengelola berbagai komponen Spark seperti RDD, DataFrame, maupun SQL API. Proses inisialisasi biasanya dilakukan dengan memanggil SparkSession.builder, kemudian ditambahkan konfigurasi seperti appName() untuk memberi nama aplikasi, master() untuk menentukan mode eksekusi (misalnya *local* atau *cluster*), serta config() untuk menyesuaikan parameter tambahan. Setelah itu, *method* getOrCreate() dipanggil untuk membuat objek *SparkSession* baru atau menggunakan yang sudah ada, sehingga siap dipakai untuk menjalankan job komputasi terdistribusi pada dataset besar. Kode python untuk inisialisasi SparkSession dapat dilihat sebagai berikut :

```
import findspark
findspark.init()

from pyspark.sql import SparkSession

# Membuat SparkSession
spark = SparkSession.builder \
    .appName("HRDataAnalysis") \
    .getOrCreate()

print("Spark Version:", spark.version)
```

b. Membaca Dataset

Membaca dataset di PySpark berarti mengimpor data dari berbagai sumber (seperti file CSV, JSON, Parquet, atau database) ke dalam struktur data terdistribusi yang menyerupai tabel pada sistem basis data. Kode python untuk membaca dataset pada PySpark dapat dilihat sebagai berikut :

```
# Membaca dataset CSV
df = spark.read.csv("HR_DATA_MNC.csv", header=True,
inferSchema=True)

# Menampilkan 5 data teratas
df.show(5)

# Melihat skema DataFrame
df.printSchema()
```

Hasil dari pembacaan dataset tersebut dapat dilihat sebagai berikut :

```
+----------+-----------+---------------+----------+--------------------+-----------+--------------------+-----------------+----------------+--------+--
-------+----------+
|Unnamed: 0|Employee_ID|      Full_Name|Department|           Job_Title| Hire_Date|            Location|Performance_Rating|Experience_Years|  Status|Wo
rk_Mode|Salary_INR|
+----------+-----------+---------------+----------+--------------------+-----------+--------------------+-----------------+----------------+--------+--
-------+----------+
|         0| EMP0000001|  Joshua Nguyen|        IT|   Software Engineer|2011-08-10|  Isaacland, Denmark|                5|              14|Resigned|
On-site|   1585363|
|         1| EMP0000002| Julie Williams| Marketing|      SEO Specialist|2018-03-02|Anthonyside, Cost...|                2|               7|  Active|
On-site|    847686|
|         2| EMP0000003|Alyssa Martinez|        HR|          HR Manager|2023-03-20|Port Christinapor...|                1|               2|  Active|
On-site|   1430084|
|         3| EMP0000004|Nicholas Valdez|        IT|   Software Engineer|2023-10-12|Port Shelbycheste...|                1|               1|  Active|
On-site|    990689|
|         4| EMP0000005| Joel Hendricks|Operations|Logistics Coordin...|2024-12-09|Lake Kimberly, Pa...|                5|               0|  Active|
On-site|    535082|
+----------+-----------+---------------+----------+--------------------+-----------+--------------------+-----------------+----------------+--------+--
-------+----------+
only showing top 5 rows

root
 |-- Unnamed: 0: integer (nullable = true)
 |-- Employee_ID: string (nullable = true)
 |-- Full_Name: string (nullable = true)
 |-- Department: string (nullable = true)
 |-- Job_Title: string (nullable = true)
 |-- Hire_Date: date (nullable = true)
 |-- Location: string (nullable = true)
 |-- Performance_Rating: integer (nullable = true)
 |-- Experience_Years: integer (nullable = true)
 |-- Status: string (nullable = true)
 |-- Work_Mode: string (nullable = true)
 |-- Salary_INR: integer (nullable = true)
```

c. Pengelolaan Data Menggunakan RDD (*Resilient Distributed Datasets*)

Pengelolaan data menggunakan RDD (*Resilient Distributed Datasets*) dalam *PySpark* merupakan konsep dasar pemrograman terdistribusi yang memungkinkan pengguna untuk bekerja dengan kumpulan data besar secara paralel di berbagai *node* dalam *cluster*. RDD bersifat *immutable* (tidak dapat diubah setelah dibuat) dan *fault-tolerant* (tahan terhadap kegagalan), sehingga jika terjadi kerusakan pada partisi data, *Spark* dapat merekonstruksi kembali data tersebut dari sumber atau dari tahapan transformasi sebelumnya. RDD menyediakan dua jenis operasi utama, yaitu transformasi (misalnya map, filter, flatMap, reduceByKey) yang menghasilkan RDD baru, dan aksi (misalnya collect, count, first, saveAsTextFile) yang mengembalikan hasil komputasi ke driver program atau menyimpannya ke penyimpanan eksternal. Dengan memanfaatkan RDD, pengguna dapat mengelola data dalam skala besar secara efisien, baik untuk pemrosesan batch maupun analisis data yang kompleks.

- **Operasi Transformasi Pada RDD**
  - Transformasi map

    Transformasi map pada RDD di Apache Spark adalah operasi transformasi yang bersifat *element-wise*, yaitu menerapkan sebuah fungsi ke setiap elemen RDD untuk menghasilkan RDD baru dengan jumlah elemen yang sama tetapi dengan nilai yang telah diubah sesuai fungsi yang diberikan. Transformasi ini bersifat *lazy evaluation*, artinya eksekusi sebenarnya tidak langsung dilakukan hingga ada aksi (*action*) yang memicu perhitungan. Sebagai contoh mengubah Salary pada dataset dalam format ribuan.

```
# Mengonversi DataFrame ke RDD
rdd = df.rdd

# Transformasi: map - mengambil elemen
print("\na) Transformasi MAP - mengambil gaji dan merubah
menjadi format ribuan")
salary_rdd = rdd.map(lambda row: (row['Full_Name'],
row['Salary_INR'] / 1000))

# Tampilkan hasil
salary_samples = salary_rdd.take(5)
print("Hasil transformasi MAP (5 sample):")
for name, salary_in_k in salary_samples:
    print(f"   {name}: Rp. {salary_in_k}")
```

Hasil dari pembacaan dataset tersebut dapat dilihat sebagai berikut :

```
a) Transformasi MAP - Mengubah gaji menjadi format ribuan
Hasil transformasi MAP (5 sample):
   Joshua Nguyen: Rp. 1585.363
   Julie Williams: Rp. 847.686
   Alyssa Martinez: Rp. 1430.084
   Nicholas Valdez: Rp. 990.689
   Joel Hendricks: Rp. 535.082
```

- Transformasi filter

Transformasi filter pada RDD di *Apache Spark* adalah operasi transformasi yang digunakan untuk menyaring elemen-elemen RDD berdasarkan kondisi atau predikat tertentu yang ditentukan oleh pengguna. Filter akan menghasilkan RDD baru yang hanya berisi elemen-elemen dari RDD asal yang memenuhi kondisi logis yang diberikan, misalnya hanya memilih angka genap dari kumpulan bilangan atau memilih data dengan nilai tertentu dari sebuah dataset. Sebagai contoh melakukan filter gaji karyawan lebih dari 1 juta.

```
# Mengonversi DataFrame ke RDD
rdd = df.rdd

# Transformasi: filter - menyaring data
print("b) Transformasi FILTER - Menyaring karyawan dengan
gaji 1 juta")
filtered_rdd = rdd.filter(lambda row: row['Salary_INR'] >
1000000)

# Menampilkan Hasil
print("10 data pertama hasil filtering (menggunakan
take()):")
first_10 = filtered_rdd.take(10)
for i, employee in enumerate(first_10):
    print(f"{i+1}. {employee}")
```

Hasil dari pembacaan dataset tersebut dapat dilihat sebagai berikut :

```
b) Transformasi FILTER - Menyaring karyawan dengan gaji > 1000000
10 data pertama hasil filtering (menggunakan take()):
1. Row(Unnamed: 0=0, Employee_ID='EMP0000001', Full_Name='Joshua Nguyen', Department='IT', Job_Title='Software Engineer', Hire_Date=datetime.date(2011,
8, 10), Location='Isaacland, Denmark', Performance_Rating=5, Experience_Years=14, Status='Resigned', Work_Mode='On-site', Salary_INR=1585363)
2. Row(Unnamed: 0=2, Employee_ID='EMP0000003', Full_Name='Alyssa Martinez', Department='HR', Job_Title='HR Manager', Hire_Date=datetime.date(2023, 3, 2
0), Location='Port Christinaport, Saudi Arabia', Performance_Rating=1, Experience_Years=2, Status='Active', Work_Mode='On-site', Salary_INR=1430084)
3. Row(Unnamed: 0=6, Employee_ID='EMP0000007', Full_Name='Julie Wright', Department='Finance', Job_Title='Finance Manager', Hire_Date=datetime.date(201
6, 4, 4), Location='Karenfort, South Africa', Performance_Rating=2, Experience_Years=9, Status='Active', Work_Mode='On-site', Salary_INR=1383891)
4. Row(Unnamed: 0=8, Employee_ID='EMP0000009', Full_Name='Cathy Thompson', Department='Finance', Job_Title='Financial Analyst', Hire_Date=datetime.date
(2018, 5, 29), Location='South Catherine, Belize', Performance_Rating=4, Experience_Years=7, Status='Resigned', Work_Mode='Remote', Salary_INR=1138452)
5. Row(Unnamed: 0=9, Employee_ID='EMP0000010', Full_Name='Maria Yu MD', Department='IT', Job_Title='Software Engineer', Hire_Date=datetime.date(2015, 1
0, 8), Location='Brownport, Yemen', Performance_Rating=4, Experience_Years=9, Status='Active', Work_Mode='Remote', Salary_INR=1543102)
6. Row(Unnamed: 0=11, Employee_ID='EMP0000012', Full_Name='Kevin Lowe', Department='Sales', Job_Title='Account Manager', Hire_Date=datetime.date(2024,
7, 2), Location='East Kent, Qatar', Performance_Rating=3, Experience_Years=1, Status='Resigned', Work_Mode='On-site', Salary_INR=1111759)
7. Row(Unnamed: 0=14, Employee_ID='EMP0000015', Full_Name='Megan Hernandez', Department='Sales', Job_Title='Account Manager', Hire_Date=datetime.date(2
016, 1, 14), Location='New Royberg, Thailand', Performance_Rating=5, Experience_Years=9, Status='Active', Work_Mode='Remote', Salary_INR=1158764)
8. Row(Unnamed: 0=24, Employee_ID='EMP0000025', Full_Name='Amanda Miller', Department='IT', Job_Title='IT Manager', Hire_Date=datetime.date(2021, 7, 3
1), Location='Tammyview, Barbados', Performance_Rating=2, Experience_Years=4, Status='Active', Work_Mode='Remote', Salary_INR=2153098)
9. Row(Unnamed: 0=25, Employee_ID='EMP0000026', Full_Name='Craig Kelly', Department='IT', Job_Title='Software Engineer', Hire_Date=datetime.date(2025,
1, 15), Location='Martinmouth, Switzerland', Performance_Rating=1, Experience_Years=0, Status='Resigned', Work_Mode='Remote', Salary_INR=1201087)
10. Row(Unnamed: 0=29, Employee_ID='EMP0000030', Full_Name='Kimberly Russo', Department='Finance', Job_Title='Finance Manager', Hire_Date=datetime.date
(2021, 8, 15), Location='New Hollyhaven, Vietnam', Performance_Rating=4, Experience_Years=4, Status='Active', Work_Mode='Remote', Salary_INR=1508236)
```

- Transformasi FlatMap

Transformasi *flatMap* pada RDD di *Apache Spark* adalah operasi yang mirip dengan map, namun dengan perbedaan utama bahwa setiap elemen input dapat dipetakan menjadi nol, satu, atau lebih elemen output, kemudian semua hasil tersebut diratakan (*flatten*) menjadi satu RDD baru. Dengan kata lain, jika *map* menghasilkan satu output untuk setiap input, maka *flatMap* memungkinkan kita menghasilkan koleksi atau daftar elemen dari satu input, yang kemudian digabung menjadi satu kumpulan tanpa *nested structure*. Transformasi ini sangat berguna, misalnya, pada kasus pemrosesan teks ketika setiap baris *string* ingin dipecah menjadi kata-kata individual; setiap baris dipetakan menjadi *array* kata, lalu semua array tersebut digabung menjadi satu RDD berisi seluruh kata. Karena sifatnya yang fleksibel, *flatMap* sering digunakan untuk operasi *parsing*, eksplorasi data, dan pembuatan struktur data yang lebih terperinci dari data mentah. Sebagai contoh memisahkan nama depan dan nama belakang.

```
# Mengonversi DataFrame ke RDD
rdd = df.rdd

print("c) Transformasi FlatMap - Memecah Full_Name menjadi
kata-words individual:")
name_words_rdd = rdd.flatMap(lambda row:
row['Full_Name'].split())
print("Kata-words dalam nama (10 pertama):")
name_words_rdd.take(10)
```

Hasil dari pembacaan dataset tersebut dapat dilihat sebagai berikut :

```
c) Transformasi FlatMap - Memecah Full_Name menjadi kata-words individual:
Kata-words dalam nama (10 pertama):
['Joshua',
 'Nguyen',
 'Julie',
 'Williams',
 'Alyssa',
 'Martinez',
 'Nicholas',
 'Valdez',
 'Joel',
 'Hendricks']
```

- Transformasi *ReduceByKey*

Transformasi *reduceByKey* pada RDD di *Apache Spark* adalah operasi yang digunakan untuk menggabungkan nilai-nilai yang memiliki key yang sama dengan menggunakan fungsi agregasi yang ditentukan pengguna.

Transformasi ini bekerja pada pasangan key-value ((K, V)) dan secara efisien melakukan pengelompokan berdasarkan key, lalu mengurangi nilai-nilai yang terkait menggunakan fungsi reduce (misalnya penjumlahan, penggabungan, atau operasi lain yang bersifat asosiatif dan komutatif). Sebagai contoh menampilkan total gaji karyawan berdasarkan departemen kerja.

```
# Mengonversi DataFrame ke RDD
rdd = df.rdd

# Transformasi: reduceByKey - mengurangi berdasarkan key
print("d) Transformasi REDUCEBYKEY - Menghitung total gaji
per departemen")
# Pastikan ada kolom 'departemen'
dept_salary_rdd = rdd.map(lambda row: (row['Department'],
row['Salary_INR']))
total_dept_salary = dept_salary_rdd.reduceByKey(lambda a, b:
a + b)

# Menampilkan sample data (misalnya 5 departemen pertama)
print("Sample 5 departemen pertama:")
sample_results = total_dept_salary.take(5)
for dept, total_salary in sample_results:
    print(f"   {dept}: Rp.{total_salary:,.2f}")
```

Hasil dari pembacaan dataset tersebut dapat dilihat sebagai berikut :

```
d) Transformasi REDUCEBYKEY - Menghitung total gaji per departemen
Sample 5 departemen pertama:
   HR: Rp.118,361,234,839.00
   Operations: Rp.226,459,565,534.00
   Sales: Rp.317,207,725,524.00
   R&D: Rp.79,844,824,817.00
   IT: Rp.679,092,203,055.00
```

- Transformasi *Distinct*

Transformasi distinct pada RDD di Spark adalah operasi yang digunakan untuk menghilangkan elemen duplikat dari sebuah RDD sehingga hanya menyisakan nilai yang unik. Saat metode ini dipanggil, Spark akan melakukan shuffle data di seluruh partisi untuk membandingkan elemen dan memastikan tidak ada nilai yang sama tersisa, sehingga hasil akhirnya berupa RDD baru yang berisi elemen-elemen unik.

```
# Mengonversi DataFrame ke RDD
rdd = df.rdd

# Transformasi: reduceByKey - Menampilkan Data Berbeda
berdasarkan key
print("e) Transformasi Distinct - Menampilkan Data yang
berbeda disetiap baris key")
job_title = rdd.map(lambda x: x.Job_Title).distinct()
print("Job Title unik:", job_title.collect())
```

Hasil dari pembacaan dataset tersebut dapat dilihat sebagai berikut :

```
Job Title unik: ['Finance Manager', 'Account Manager', 'Marketing Executive', 'Data Analyst', 'CTO', 'Lab Technician', 'HR Director', 'Operations Execu
tive', 'Sales Director', 'Research Scientist', 'SEO Specialist', 'HR Manager', 'Financial Analyst', 'Content Strategist', 'Operations Director', 'Brand
Manager', 'Logistics Coordinator', 'Software Engineer', 'HR Executive', 'Accountant', 'Talent Acquisition Specialist', 'Sales Executive', 'Product Deve
loper', 'Business Development Manager', 'DevOps Engineer', 'Supply Chain Manager', 'IT Manager', 'CFO', 'Innovation Manager']
```

- Transformasi *Join*

Transformasi Join pada RDD di Spark adalah operasi yang digunakan untuk menggabungkan dua RDD berdasarkan key yang sama, mirip dengan operasi join pada tabel di basis data relasional. Kedua RDD yang akan digabung harus berupa pasangan kunci-nilai (key-value pair), di mana kunci digunakan sebagai acuan untuk mencocokkan data. Hasil dari operasi ini adalah RDD baru yang berisi pasangan kunci beserta nilai dari kedua RDD yang memiliki kunci yang sama.

```
# Repartisi RDD untuk distribusi yang lebih baik
rdd = df.rdd.repartition(4)  # Sesuaikan dengan jumlah core

print(f"Jumlah partisi RDD: {rdd.getNumPartitions()}")

# join → contoh join Employee_ID dengan Status dan Departemen
id_status = rdd.map(lambda x: (x.Employee_ID, x.Status))
id_dept = rdd.map(lambda x: (x.Employee_ID, x.Department))

joined = id_status.join(id_dept)

print("Sample hasil join (10 data pertama):")
sample_results = joined.take(10)
for result in sample_results:
    print(f"  {result}")
```

Hasil dari pembacaan dataset tersebut dapat dilihat sebagai berikut :

```
Sample hasil join (10 data pertama):
  ('EMP0000037', ('Active', 'Marketing'))
  ('EMP0000191', ('Active', 'Marketing'))
  ('EMP0000233', ('Active', 'Sales'))
  ('EMP0000272', ('Active', 'Finance'))
  ('EMP0000277', ('Terminated', 'Sales'))
  ('EMP0000313', ('Active', 'Operations'))
  ('EMP0000320', ('Active', 'R&D'))
  ('EMP0000440', ('Active', 'IT'))
  ('EMP0000471', ('Active', 'Operations'))
  ('EMP0000474', ('Active', 'Sales'))
```

- Operasi Aksi Pada RDD

Operasi aksi dalam Spark RDD adalah operasi yang memicu eksekusi komputasi pada seluruh RDD dan mengembalikan nilai hasil ke driver program atau menyimpannya ke sistem eksternal, berbeda dengan transformasi yang bersifat *lazy evaluation*. Ketika aksi dipanggil, Spark menjalankan semua transformasi yang tertunda dalam *lineage* RDD untuk menghasilkan hasil akhir, membuat aksi menjadi titik di mana komputasi sebenarnya terjadi. Beberapa contoh aksi utama termasuk collect() yang mengembalikan semua elemen RDD ke driver sebagai array, count() yang menghitung jumlah elemen dalam RDD, take(n) yang mengambil n elemen pertama, reduce(func) yang menggabungkan elemen menggunakan fungsi asosiatif, foreach(func) yang menerapkan fungsi ke setiap elemen untuk efek samping seperti menyimpan ke database, dan saveAsTextFile(path) yang menyimpan RDD sebagai file teks.

```
# 1. count() - ex: Menghitung total karyawan
total_employees = rdd.count()
print(f"Total karyawan: {total_employees}")


# 2. take() - ex: Mengambil 8 karyawan pertama
first_three = rdd.take(8)
print("8 karyawan pertama:")
for emp in first_three:
    print(emp[2])


# 3. collect() - ex: Mengambil semua data
all_employees = rdd.collect()
print(f"Data semua {len(all_employees)} karyawan terkumpul")


# 4. first() - ex: Mengambil karyawan pertama
first_employee = rdd.first()
print(f"Karyawan pertama: {first_employee[2]}")


# 5. reduce() - ex: Menghitung total gaji semua karyawan
total_salary = rdd.map(lambda x: x['Salary_INR']).reduce(lambda
a, b: a + b)
print(f"Total gaji semua karyawan: {total_salary:,} INR")


# 6. countByValue() - ex: Menghitung karyawan per department
dept_count = rdd.map(lambda x: x['Department']).countByValue()
```

```
print("Jumlah karyawan per department:")
for dept, count in dept_count.items():
    print(f"{dept}: {count}")


# 7. max() - ex: Mencari gaji tertinggi
max_salary = rdd.map(lambda x: x['Salary_INR']).max()
print(f"Gaji tertinggi: {max_salary:,} INR")


# 8. min() - ex: Mencari gaji terendah
min_salary = rdd.map(lambda x: x[11]).min()
print(f"Gaji terendah: {min_salary:,} INR")


# 9. takeOrdered() - ex: Mengambil 3 karyawan dengan gaji
tertinggi
top_salaries = rdd.takeOrdered(3, key=lambda x: -
x['Salary_INR'])
print("3 karyawan dengan gaji tertinggi:")
for emp in top_salaries:
    print(f"{emp['Full_Name']}: {emp['Salary_INR']:,} INR")


# 10. saveAsTextFile() - Menyimpan data ke file
rdd.map(lambda x:
f"{x['Full_Name']},{x['Department']},{x['Salary_INR']}").saveAsT
extFile("hr_data_output")
```

d. Pengelolaan Data Menggunakan DataFrame

DataFrame pada Spark merupakan pendekatan inti dalam pemrosesan data terdistribusi yang menyediakan antarmuka pemrograman tingkat tinggi yang abstrak, efisien, dan mudah digunakan untuk memanipulasi dataset berskala besar. DataFrame Spark, yang dibangun di atas RDD (*Resilient Distributed Dataset*), mengorganisir data ke dalam bentuk tabel terstruktur dengan nama kolom dan tipe data, sehingga memungkinkan pengguna untuk melakukan operasi SQL-like (seperti select, filter, groupBy, agg) serta transformasi kompleks (join, pivot, window function) dengan kode yang ringkas dan dioptimalkan oleh mesin eksekusi *Catalyst Optimizer* untuk kinerja yang tinggi, baik pada data yang tersimpan di berbagai sumber (seperti CSV, Parquet, database) seperti dataset HR_Data_MNC, yang berisi informasi karyawan, sehingga memudahkan tugas-tugas analitik seperti menghitung rata-rata gaji per

departemen, mengidentifikasi tingkat attrition, atau menganalisis distribusi demografi karyawan. Contoh penggunaan Dataframe dapat dilihat sebagai berikut :

```
# Select: ex - Ambil kolom 'Full Name', 'Department', dan
'Salary_INR'
df_select = df.select("Full_Name", "Department", "Salary_INR")
df_select.show(5)


# Filter: ex - Karyawan dengan gaji di atas 500 ribu dan Departemen
'IT'
df_filter = df.filter((df.Salary_INR > 500000) & (df.Department ==
"IT"))
df_filter.show(5)


# Agregasi: ex - Hitung rata-rata gaji dan jumlah status per
departemen
from pyspark.sql import functions as F
df_agg = df.groupBy("Department").agg(
    F.avg("Salary_INR").alias("avg_salary"),
    F.sum(F.when(df.Status == "Active",
1).otherwise(0)).alias("total_status")
)
df_agg.show()


# Join: ex - Gabungkan dengan dataset lain (misal:
df_department_info)
df_join = df.join(df_department_info, on="Department", how="left")
df_join.show(5)
```

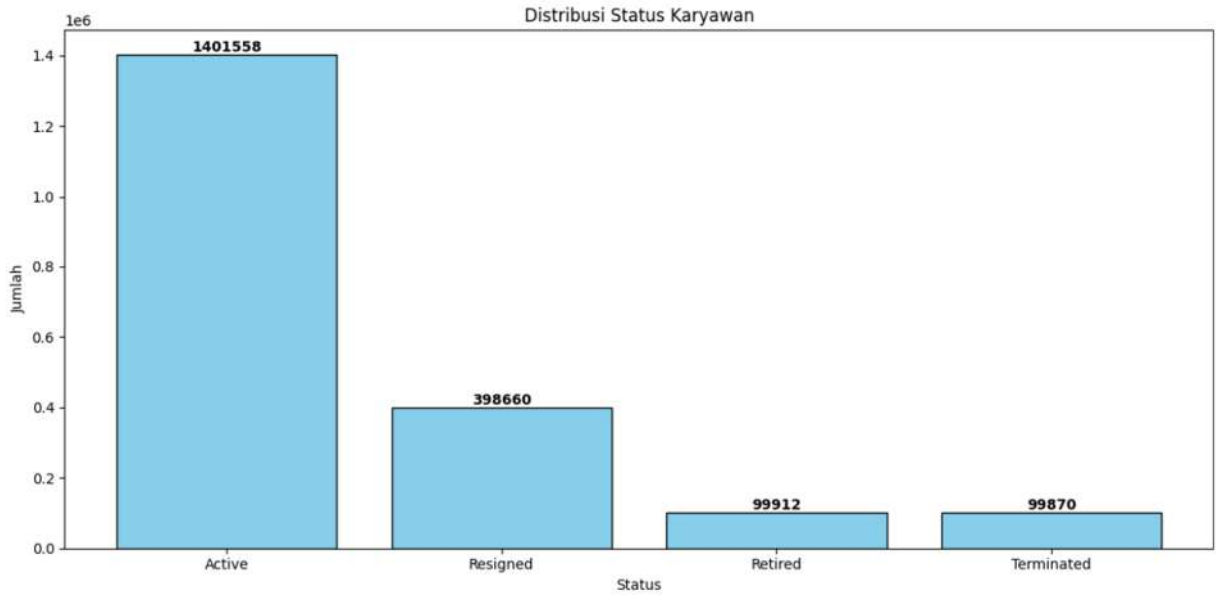
Contoh analisis data menggunakan RDD pada spark yaitu mengetahui distribusi status karyawana, maka dapat dilihat melalui kode python berikut :

```
import matplotlib.pyplot as plt
import pandas as pd
import findspark
findspark.init()

from pyspark.sql import SparkSession

# Membuat SparkSession
spark = SparkSession.builder \
    .appName("HRDataAnalysis") \
    .getOrCreate()

# Membaca dataset CSV
```

```
df = spark.read.csv("HR_Data_MNC.csv", header=True, inferSchema=True)

# 3. Konversi DataFrame ke RDD
rdd = df.rdd

# Repartisi RDD untuk distribusi yang lebih baik
rdd = df.rdd.repartition(4)  # Sesuaikan dengan jumlah core

# 4. Ambil kolom Status
status_rdd = rdd.map(lambda row: (row["Status"], 1))

# 5. Hitung jumlah masing-masing Status dengan reduceByKey
status_jumlah = status_rdd.reduceByKey(lambda a, b: a + b)

# 6. Ambil hasil ke driver
hasil = status_jumlah.collect()

# 7. Tampilkan distribusi
for status, jumlah in hasil:
    print(f"{status}: {jumlah}")

# 8. Visualisasi
status_df = pd.DataFrame(hasil, columns=["Status", "Jumlah"])

plt.figure(figsize=(12,6))
bars = plt.bar(status_df["Status"], status_df["Jumlah"],
color="skyblue", edgecolor="black")

plt.title("Distribusi Status Karyawan")
plt.xlabel("Status")
plt.ylabel("Jumlah")

# Tambahkan nilai di atas batang
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + 0.5,   # posisi
teks
             str(height), ha='center', va='bottom', fontsize=10,
fontweight="bold")

plt.tight_layout()
plt.show()
```

Hasil analisis menunjukkan hasil :

```
Active: 1401558
Resigned: 398660
Retired: 99912
Terminated: 99870
```



5.5. Tugas Praktikum

a. Bagaimana distribusi "Work_Mode" (On-site, Remote)?
b. Berapa jumlah karyawan di setiap departemen?
c. Berapa rata-rata gaji per Departemen?
d. Jabatan apa yang memiliki rata-rata gaji tertinggi?
e. Berapa rata-rata gaji di berbagai Departemen berdasarkan Jabatan?
f. Berapa banyak karyawan yang Mengundurkan Diri & Dipecat di setiap departemen?
g. Bagaimana variasi gaji berdasarkan tahun pengalaman?
h. Berapa rata-rata peringkat kinerja berdasarkan departemen?
i. Negara mana yang memiliki konsentrasi karyawan tertinggi?
j. Apakah ada korelasi antara peringkat kinerja dan gaji?
k. Bagaimana jumlah perekrutan berubah dari waktu ke waktu (per tahun)?
l. Bandingkan gaji karyawan Remote vs. On-site, apakah ada perbedaan yang signifikan?
m. Temukan 10 karyawan dengan gaji tertinggi di setiap departemen.
n. Identifikasi departemen dengan tingkat attrition (persentase pengunduran diri) tertinggi.

5.6. Referensi

1. Apache Spark Documentation. https://spark.apache.org/docs/latest/
2. TutorialsPoint – PySpark Tutorial. https://www.tutorialspoint.com/pyspark/index.htm
3. IBM Developer – Big Data Processing with PySpark. https://developer.ibm.com/articles/bd-using-apache-spark-pyspark/