2025

MODUL PRAKTIKUM 4

PYSPARK FOR BIG DATA ANALYTICS

OPTIMIZED VERSION

Disusun Oleh:
Ryan F F Hakim
Silvi Nurinsan

Dipersembahkan Untuk:
Sains Data

Mata Kuliah
Analisis Big Data

APA ITU APACHE SPARK

Apache Spark adalah sebuah *framework* komputasi terdistribusi *open-source* yang dirancang untuk pemrosesan data skala besar dengan cepat dan efisien. Spark dikembangkan untuk meningkatkan kecepatan dan kesederhanaan analisis data besar dengan menyediakan antarmuka yang mudah digunakan dan kemampuan bawaan untuk pemrosesan data terdistribusi. Spark dapat memproses data dari berbagai sumber seperti Hadoop Distributed File System (HDFS), Apache Cassandra, Apache HBase, dan S3.

Spark dimulai pada tahun 2009 sebagai proyek penelitian di AMPLab UC Berkeley dengan tujuan menciptakan *framework* baru yang dioptimalkan untuk pemrosesan iteratif cepat seperti pembelajaran mesin dan analisis data interaktif, sambil tetap mempertahankan skalabilitas dan toleransi kesalahan dari Hadoop MapReduce.

MESIN PEMROSESAN DATA TERDISTIBUSI DAN GENERAL-PURPOSE

Apache Spark didefinisikan sebagai mesin analitik terpadu untuk pemrosesan data skala besar 5. Karakteristik utamanya sebagai mesin pemrosesan data terdistribusi yang cepat dan serbaguna (*general-purpose*) dapat dijabarkan sebagai berikut:

- Terdistribusi: Spark bekerja di atas klaster komputer, memungkinkan pemrosesan dataset masif secara paralel. Ini berarti tugas-tugas pemrosesan dipecah dan didistribusikan ke banyak mesin (node) yang bekerja bersama.
- Cepat: Kecepatan Spark terutama berasal dari kemampuannya melakukan komputasi *in-memory* (dalam memori). Dengan menyimpan data sementara di memori alih-alih di disk (seperti yang dilakukan Hadoop MapReduce), Spark secara signifikan mengurangi latensi operasi baca/tulis, membuatnya hingga 100 kali lebih cepat untuk pemrosesan *in-memory* dan 10 kali lebih cepat di disk.
- General-Purpose: Spark adalah mesin serbaguna yang mendukung berbagai jenis beban kerja, termasuk pemrosesan batch, kueri interaktif, analitik *real-time*, pembelajaran mesin, dan pemrosesan graf. Fleksibilitas ini memungkinkan pengguna untuk membangun aplikasi yang menggabungkan berbagai jenis pemrosesan dalam satu *framework*.

KEUNGGULAN SPARK

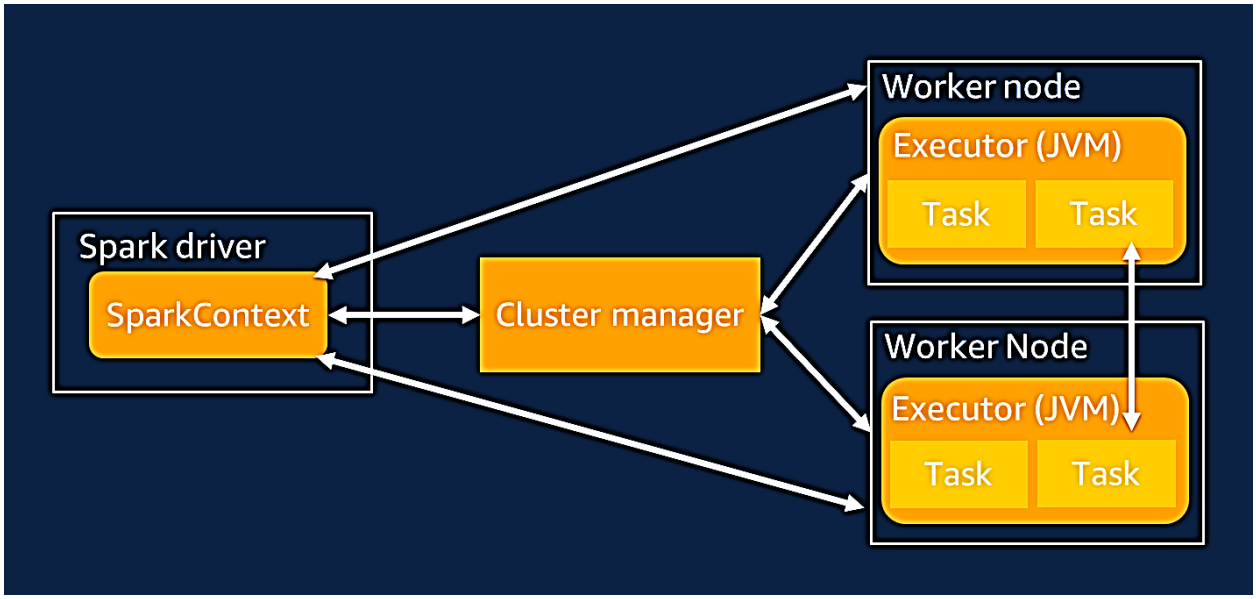
Apache Spark menawarkan beberapa keunggulan signifikan dibandingkan teknologi pemrosesan data besar lainnya, terutama Hadoop MapReduce:

- Kecepatan: Seperti yang telah disebutkan, kemampuan pemrosesan *in-memory* Spark menghasilkan kinerja yang jauh lebih tinggi, terutama untuk algoritma iteratif yang umum dalam pembelajaran mesin dan analisis data interaktif.
- Kemudahan Penggunaan: Spark menyediakan API tingkat tinggi dalam bahasa pemrograman populer seperti Java, Scala, Python, dan R. Hal ini memudahkan pengembang untuk menulis aplikasi pemrosesan data paralel dengan kode yang lebih ringkas dibandingkan dengan MapReduce.
- Mesin Terpadu: Spark menggabungkan berbagai kemampuan pemrosesan data (SQL, streaming, ML, graf) dalam satu platform. Ini menyederhanakan pengembangan dan pemeliharaan aplikasi data yang kompleks.
- Pemrosesan Aliran Real-time: Spark Streaming memungkinkan pemrosesan data aliran *real-time* atau *near real-time* dengan menyerap data dalam *mini-batch* dan melakukan transformasi RDD (Resilient Distributed Dataset) pada *mini-batch* tersebut.
- Toleransi Kesalahan (Fault Tolerance): Spark mencapai toleransi kesalahan melalui konsep RDD. RDD adalah kumpulan data terdistribusi yang tidak dapat diubah (immutable) dan dapat diproses secara paralel. Jika sebuah partisi RDD hilang karena kegagalan node, Spark dapat merekonstruksinya menggunakan *lineage* (informasi silsilah operasi yang menghasilkan RDD tersebut).
- Skalabilitas: Spark dapat berjalan secara mandiri, di atas Apache Mesos, Kubernetes, atau yang paling umum, di atas Hadoop YARN, memungkinkannya untuk diskalakan dari satu node hingga ribuan node.
- Analitik Tingkat Lanjut: Selain operasi "Map" dan "Reduce" dasar, Spark mendukung kueri SQL, pembelajaran mesin, pemrosesan aliran, dan algoritma graf, menjadikannya platform yang kuat untuk analitik tingkat lanjut.

KOMPONEN EKOSISTEM SPARK

Ekosistem Apache Spark terdiri dari beberapa komponen utama yang dibangun di atas Spark Core. Komponen-komponen ini dapat digunakan secara bersamaan dalam satu aplikasi, memberikan fleksibilitas yang besar.

Berikut adalah visualisasi sederhana interaksi komponen-komponen utama dalam Apache Spark:



ARSITEKTUR DASAR SPARK

Apache Spark memiliki arsitektur *master-slave* yang dirancang untuk komputasi terdistribusi. Komponen-komponen utamanya adalah:

1. Driver Program:
    - Ini adalah proses utama yang menjalankan fungsi main() dari aplikasi Spark Anda dan membuat SparkContext.
    - SparkContext bertanggung jawab untuk mengoordinasikan proses-proses di klaster.
    - Driver mengubah kode pengguna menjadi *Directed Acyclic Graph* (DAG) dari tugas-tugas dan membaginya menjadi *stages* (tahapan).
    - Driver menjadwalkan tugas-tugas ke *executors*.
    - Driver harus dapat dijangkau melalui jaringan oleh *worker nodes* karena ia mendengarkan koneksi masuk dari *executors*.
2. Cluster Manager:
    - Ini adalah komponen eksternal yang bertanggung jawab untuk mengakuisisi sumber daya pada klaster untuk aplikasi Spark.
    - Spark mendukung beberapa jenis *cluster manager*, termasuk:
        - Standalone: Manajer klaster sederhana yang disertakan dengan Spark.
        - Apache Mesos: Manajer klaster umum yang juga dapat menjalankan aplikasi lain.
        - Hadoop YARN (Yet Another Resource Negotiator): Manajer sumber daya yang umum digunakan dalam ekosistem Hadoop.
        - Kubernetes: Sistem orkestrasi kontainer *open-source*.

3. Executor:
    - Ini adalah proses yang berjalan pada *worker nodes* di klaster.
    - *Executors* bertanggung jawab untuk menjalankan tugas-tugas (komputasi) yang diberikan oleh *driver* dan menyimpan data dalam memori atau disk.
    - Setiap aplikasi Spark memiliki *executor processes* sendiri yang berjalan selama durasi aplikasi dan menjalankan tugas dalam banyak *threads*.
    - Hasil dari komputasi dikembalikan ke *driver*.

Alur Kerja Dasar:

1. Pengguna mengirimkan aplikasi Spark menggunakan skrip spark-submit.
2. Driver program dimulai dan SparkContext dibuat.
3. SparkContext terhubung ke *cluster manager*.
4. *Cluster manager* mengalokasikan sumber daya dan meluncurkan *executors* pada *worker nodes*.
5. Driver mengirimkan kode aplikasi (file JAR atau Python) dan tugas-tugas ke *executors*.
6. *Executors* menjalankan tugas-tugas dan mengembalikan hasilnya ke *driver*.

Abstraksi Data Utama:

- Resilient Distributed Dataset (RDD): Abstraksi data fundamental di Spark. RDD adalah kumpulan elemen yang tidak dapat diubah (immutable), terpartisi, dan dapat dioperasikan secara paralel di seluruh node klaster. RDD bersifat toleran terhadap kesalahan (*fault-tolerant*) karena kemampuannya untuk dibangun kembali dari *lineage* (silsilah transformasi).
- DataFrame dan Dataset: Abstraksi tingkat lebih tinggi yang dibangun di atas RDD. DataFrame mengorganisir data ke dalam kolom bernama, mirip tabel relasional, dan menyediakan optimasi melalui Catalyst optimizer. Dataset (terutama di Scala/Java) adalah ekstensi dari DataFrame API yang menyediakan keamanan tipe (*type safety*).

APA ITU PYSPARK

PySpark adalah API Python untuk Apache Spark. Ini memungkinkan Anda untuk memanfaatkan kekuatan pemrosesan data terdistribusi Spark menggunakan bahasa pemrograman Python yang populer dan mudah dipelajari.

Berikut poin-poin penting tentang PySpark:

- Antarmuka ke Spark: PySpark menyediakan *binding* Python ke *core engine* Spark, memungkinkan pengembang Python untuk menulis aplikasi Spark.
- Integrasi dengan Ekosistem Python: PySpark terintegrasi dengan baik dengan pustaka Python lainnya yang umum digunakan dalam ilmu data, seperti Pandas dan NumPy, meskipun penting untuk diingat bahwa operasi terdistribusi utama dilakukan pada struktur data Spark (RDD, DataFrame).
- Menggunakan Py4J: Di balik layar, PySpark menggunakan pustaka bernama Py4J yang memungkinkan Python berinteraksi dengan objek Java Virtual Machine (JVM), karena Spark sendiri ditulis dalam Scala dan berjalan di JVM.
- Mendukung Semua Fitur Spark: PySpark mendukung semua fitur utama Spark, termasuk Spark SQL (untuk bekerja dengan data terstruktur menggunakan kueri SQL dan DataFrame), Spark Streaming (untuk pemrosesan data aliran), MLlib (untuk pembelajaran mesin), dan GraphX (untuk pemrosesan graf).
- DataFrame API: Sama seperti Spark Scala/Java, PySpark menyediakan DataFrame API yang kuat untuk manipulasi data terstruktur secara efisien.

KAPAN MENGGUNAKAN PYSPARK (VS PANDAS)

Pilihan antara PySpark dan Pandas sangat bergantung pada ukuran data dan lingkungan komputasi yang dimiliki.

Gunakan Pandas ketika:

- Ukuran Data Kecil hingga Sedang: Pandas sangat efisien untuk dataset yang dapat dimuat sepenuhnya ke dalam memori satu mesin (biasanya hingga beberapa gigabyte).
- Operasi pada Mesin Tunggal: Pandas dirancang untuk berjalan pada satu mesin dan memanfaatkan CPU tunggal (meskipun beberapa operasi dapat menggunakan *multithreading* secara internal).
- Analisis Data Eksploratif Cepat dan Manipulasi Data Interaktif: Sintaks Pandas yang intuitif dan banyaknya fungsi bawaan membuatnya sangat baik untuk eksplorasi data cepat, pembersihan data, dan transformasi pada dataset yang lebih kecil.
- Integrasi Erat dengan Pustaka Python Lainnya: Jika alur kerja Anda sangat bergantung pada pustaka Python lain yang tidak terdistribusi (misalnya, beberapa implementasi Scikit-learn tertentu, visualisasi dengan Matplotlib/Seaborn secara langsung pada keseluruhan dataset).
- Kurva Pembelajaran yang Lebih Landai: Bagi mereka yang sudah familiar dengan Python, Pandas umumnya lebih mudah dipelajari dan digunakan untuk tugas-tugas analisis data standar.

Gunakan PySpark ketika:

- Ukuran Data Besar (Big Data): PySpark dirancang untuk menangani dataset yang sangat besar (puluhan gigabyte, terabyte, atau bahkan petabyte) yang tidak muat dalam memori satu mesin.
- Pemrosesan Terdistribusi: Jika Anda perlu memproses data secara paralel di banyak mesin dalam sebuah klaster untuk mempercepat komputasi.
- Skalabilitas: Aplikasi PySpark dapat dengan mudah diskalakan dengan menambahkan lebih banyak node ke klaster.
- Toleransi Kesalahan: Spark menyediakan toleransi kesalahan bawaan, yang berarti jika satu node gagal selama pemrosesan, Spark dapat memulihkan pekerjaan yang hilang.
- Operasi Pembelajaran Mesin Skala Besar: MLlib Spark memungkinkan pelatihan model pembelajaran mesin pada dataset yang sangat besar secara terdistribusi.
- Pemrosesan Aliran Data (Streaming): Jika Anda perlu memproses aliran data *real-time* atau *near real-time*.

- Kueri SQL pada Data Besar: Spark SQL memungkinkan Anda menjalankan kueri mirip SQL pada dataset terdistribusi yang besar.

Perbandingan Performa dan Konsumsi Memori:

- Performa:
    - Untuk dataset kecil, Pandas seringkali lebih cepat karena tidak ada *overhead* dari komputasi terdistribusi yang dimiliki PySpark.
    - Untuk dataset besar, PySpark secara signifikan mengungguli Pandas karena kemampuannya untuk memproses data secara paralel di banyak node. Benchmark menunjukkan bahwa PySpark bisa 100x lebih cepat dari sistem tradisional untuk tugas tertentu.
- Konsumsi Memori:
    - Pandas memuat seluruh dataset ke dalam memori, yang bisa menjadi masalah untuk data besar dan menyebabkan kesalahan OutOfMemoryError.
    - PySpark memproses data dalam partisi-partisi terdistribusi dan dapat melakukan operasi *out-of-core* (data tidak harus sepenuhnya dimuat ke memori), sehingga lebih efisien dalam penggunaan memori untuk dataset besar. PySpark juga menggunakan evaluasi *lazy*, yang berarti transformasi tidak dieksekusi sampai sebuah *action* dipanggil, ini membantu dalam optimasi.

LATIHAN PYSPARK DENGAN GOOGLE COLAB

DATASETS

Untuk Dataset dapat diunduh pada link berikut:

Tree.csv

https://github.com/RyanHakim24/Lab_SainsData_BigData/blob/99335643aaf3baed033aac812c54dbdfc6e4e874/Dataset_Praktikum/Tree.csv

Environment.csv

https://github.com/RyanHakim24/Lab_SainsData_BigData/blob/99335643aaf3baed033aac812c54dbdfc6e4e874/Dataset_Praktikum/Tree_Environment.csv

INSTALL JAVA DAN PYSPARK

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!pip install -q pyspark
```

SET ENVIRONMENT VARIABLES

```
import os

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/usr/local/lib/python3.11/dist-packages/pyspark"
```

INISIALISASI SPARKSESSION

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local[*]").appName("Colab PySpark").getOrCreate()
```

CONTOH PENGGUNAAN PYSPARK

```
data = [("Alice", 25), ("Bob", 30), ("Charlie", 35)]
df = spark.createDataFrame(data, ["Name", "Age"])
df.show()
```

```
df2 = spark.range(500).toDF('number')
df2.select(df2['number'] + 10)
df2.show()
```

```
spark.range(2).collect()
```

LOAD DATASET

```
trees_df = spark.read.csv('/content/Tree.csv', header=True, inferSchema=True)

trees_df.show()
```

```
environment_df = spark.read.csv('/content/Tree_Environment.csv', header=True, inferSchema=True)

environment_df.show()
```

BASIC DATAFRAME OPERATIONS

```
# 1. Menampilkan 5 baris pertama
trees_df.show(5, truncate=False)
```

```
# 2. Filter pohon yang tingginya >30 meter dan berumur <70 tahun
tall_young_trees = trees_df.filter((trees_df.Height_m > 30) & (trees_df.Age_years < 70))
tall_young_trees.show()
```

```
# 3. Hitung jumlah pohon per status kesehatan
health_counts = trees_df.groupBy("Health_Status").count()
health_counts.show()
```

DATA CLEANING

```
# 1. Cek missing values
from pyspark.sql.functions import col, sum

missing_values = trees_df.select([sum(col(c).isNull().cast("int")).alias(c) for c in trees_df.columns])
missing_values.show()
```

```python
# 2. Isi missing value di Height_m dengan nilai median
from pyspark.sql.functions import median

median_height = trees_df.select(median("Height_m")).first()[0]
trees_df = trees_df.na.fill(median_height, subset=["Height_m"])

trees_df.show()
```

```python
# 2.1 Isi missing value di Health_Status dengan nilai modus
from pyspark.sql.functions import mode

modus_health_status = trees_df.select(mode("Health_Status")).first()[0]
trees_df = trees_df.na.fill(modus_health_status, subset=["Health_Status"])

trees_df.show()
```

```python
# 3. Hapus data duplikat berdasarkan Tree_ID
trees_df = trees_df.dropDuplicates(["Tree_ID"])

trees_df.show()
```

JOIN OPERATIONS

```python
# 1. Lakukan inner join antara trees_df dan environment_df
joined_df = trees_df.join(environment_df, on="Tree_ID", how="inner")

joined_df.show()
```

```python
# 2. Hitung rata-rata diameter per tipe tanah
avg_diameter_by_soil = joined_df.groupBy("Soil_Type").avg("Diameter_cm")
avg_diameter_by_soil.show()
```

```python
# 3. Temukan pohon Pinaceae yang memiliki status 'Healthy'
sick_pine_trees = joined_df.filter((col("Family") == "Pinaceae") & (col("Health_Status") == "Healthy"))

sick_pine_trees.show()
```

DATE FUNCTION

```python
from pyspark.sql.functions import to_date, datediff, current_date, year

# 1. Hitung berapa tahun sejak pohon ditanam

# 1.1 Konversi string ke Date
trees_df = trees_df.withColumn(
    "Planted_Date",
    to_date(col("Planted_Date"), "M/d/yyyy")
)

trees_with_age = trees_df.withColumn(
    "Years_Since_Planted",
    year(current_date()) - year(col("Planted_Date"))
)

trees_with_age.select("Tree_ID", "Planted_Date", "Years_Since_Planted").show()
```

```python
# 2. Membandingkan Age_years dengan Years_Since_Planted
descrepancy = trees_with_age.filter(col("Age_years") != col("Years_Since_Planted"))

descrepancy.select("Tree_ID", "Age_years", "Years_Since_Planted").show()
```

ADVANCED AGGREGATIONS

```python
# 1. Menghitung total carbon sequestration per family pohon
carbon_per_family = joined_df.groupBy("Family").sum("Carbon_Sequestration_kg_yr")

carbon_per_family.show()
```

```python
# 2. Menemukan 3 lokasi dengan rata-rata tinggi pohon tertinggi
from pyspark.sql.functions import desc

top_locations = joined_df.groupBy("Location").avg("Height_m")

top_locations = top_locations.orderBy(col("avg(Height_m)").desc()).limit(3)

top_locations.show()
```

```python
# 3. Buat pivot table: rata-rata diameter per Soil_Type dan Evergreen
pivot_table = joined_df.groupBy("Soil_Type").pivot("Evergreen").avg("Diameter_cm")

pivot_table.show()
```

KODE LENGKAP

Untuk kode lengkap praktikum dapat diakses berikut:

https://colab.research.google.com/drive/1bby02N9cPlcTZyGbtlC22x6eW0rV9lTF?usp=sharing