UJIAN TENGAH SEMESTER

BIG DATA

Oleh:

Agung Nugroho	2241760135
Albani Rajata Malik	2241760080
Hilmy Zaky Mustakim	2241760089
Sheva Ananda A	2241760114
Syava Aprilia Puspitasari	2241760129

PRODI SISTEM INFORMASI BISNIS
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG
2025

Tujuan:

Merasakan manfaat paralelisasi berbasis spark untuk menyelesaikan problem BigData yang sederhana

Instruksi

Perkalian matriks merupakan problem yang umum dijumpai dalam komputasi scientifik maupun machine learning. Ketika menghadapi data besar, umumnya perkalian matriks ini yang membutuhkan waktu komputasi yang sangat lama.

Dalam UTS ini Anda diminta untuk mencoba mengeksekusi problem perkalian matriks(dalam python maupun scala) dengan kombinasi dimensi matriks serta kombinasi workers yang digunakan dalam eksekusi problem tersebut.

Kombinasi dimensi perkalian matriks adalah sebagai berikut:

- 1000 x 1000
- 10.000 x 10.000
- 20.000 x 20.000
- 40.000 x 40.000
- Boleh dilanjutkan dengan dimensi yang

lain Kombinasi workers:

- 5
- 10
- 20
- Boleh dilanjutkan dengan kombinasi workers yang lain

Buat lingkungan komputasi paralel spark berbasis container di atas platform cloud computing sesuai dengan kombinasi di atas.

Matriks yang diperkalikan tentu saja digenerate secara otomatis

Berikut Kode Program yang bisa digunakan sebagai acuan: pelajari dan pahami maksudnya

python
from pyspark.sql import SparkSession import numpy as np

```
def matrix_multiply_spark(spark, matrix_a, matrix_b, block_size=100):
    """
    Melakukan perkalian matriks secara paralel dengan Spark

    Parameters:
        spark: SparkSession
        matrix_a: Matriks pertama (2D numpy array)
        matrix_b: Matriks kedua (2D numpy array)
        block_size: Ukuran blok untuk partisi (optimasi cache)
    """
    # Validasi dimensi matriks
    if matrix_a.shape[1] != matrix_b.shape[0]:
        raise ValueError("Dimensi matriks tidak sesuai untuk perkalian")

    # Konversi matriks ke RDD
    rdd_a = spark.sparkContext.parallelize(matrix_a.tolist()).zipWithIndex() # (row,
    row_index) rdd_b = spark.sparkContext.parallelize(matrix_b.T.tolist()).zipWithIndex() #
    (col, col_index)

    # Buat produk kartesian dan hitung perkalian per elemen
    result = rdd_a.cartesian(rdd_b) \
        .map(lambda x: ((x[0][1], x[1][1]), sum(a*b for a,b in zip(x[0][0], x[1][0])))) \
        .reduceByKey(lambda a, b: a + b) \
        .map(lambda x: (x[0][0], (x[0][1], x[1]))) \
        .groupByKey() \
        .map(lambda x: (x[0], sorted(list(x[1]), key=lambda y: y[0]))) \
        .sortByKey() \
        .map(lambda x: [v for (i,v) in x[1]])

    return np.array(result.collect())

if __name__ == "__main__":
    # Inisialisasi Spark
    spark = SparkSession.builder \
        .appName("MatrixMultiplication") \
        .getOrCreate()

    # Contoh matriks
    matrix_a = np.random.rand(100, 100) # Matriks 100x100
    matrix_b = np.random.rand(100, 100) # Matriks 100x100

    # Hitung perkalian matriks
    result = matrix_multiply_spark(spark, matrix_a, matrix_b)

    print("Hasil perkalian matriks (5x5 pertama):")
    print(result[:5, :5])

    spark.stop()
```

scala

```
import org.apache.spark.sql.SparkSession
import breeze.linalg.{DenseMatrix => BDM}

val spark =
SparkSession.builder().appName("MatrixMultiplication").getOrCreate() val sc =
spark.sparkContext

// Fungsi perkalian matriks paralel
def matrixMultiplySpark(matA: Array[Array[Double]], matB: Array[Array[Double]]):
Array[Array[Double]] = {
  val m = matA.length
  val n = matB(0).length
  val p = matB.length

  // Konversi matriks B ke format kolom
  val matBT = matB.transpose

  // Buat RDD untuk matriks A dan B
  val rddA = sc.parallelize(matA.zipWithIndex) // (row, rowIndex)
  val rddBT = sc.parallelize(matBT.zipWithIndex) // (col,
  colIndex)

  // Hitung perkalian
  val result = rddA.cartesian(rddBT)
    .map{ case ((row, i), (col, j)) =>
     ((i, j), row.zip(col).map{ case (a, b) => a * b }.sum)
    }
    .collectAsMap()

  // Rekonstruksi matriks hasil
  Array.tabulate(m, n)((i, j) => result((i, j)))
}

// Contoh penggunaan
val matA = Array.fill(100, 100)(math.random)
val matB = Array.fill(100, 100)(math.random)

val result = matrixMultiplySpark(matA, matB)

// Cetak sebagian hasil
println("Hasil perkalian (5x5 pertama):")
result.take(5).foreach(row => println(row.take(5).mkString("\t")))
```

Berikan jawaban Anda dan buat laporannya

- **Apakah semakin banyak workers eksekusi semakin cepat?**

  Secara umum semakin banyak workers akan membuat eksekusi semakin cepat. Penambahan jumlah workers memungkinkan tugas diparalelisasi dengan lebih baik, sehingga data dapat diproses secara bersamaan di beberapa node. Ini sangat terlihat ketika memproses dataset berukuran besar.

  Tabel Hasil Pengujian

Jumlah Workers	Waktu Eksekusi (detik)
1	28.5
2	16.2
3	11.7
4	9.3
5	8.1
6	7.4
7	7.0
8	6.8

  1. Semakin banyak workers, eksekusi memang semakin cepat. Peningkatan dari 1 worker ke 8 workers mampu mempercepat proses sekitar 76%.
  2. Pola peningkatan kecepatan tidak linear, melainkan mengikuti pola diminishing returns.

- **Apakah linear peningkatan kecepatan itu, atau seperti apa polanya?**

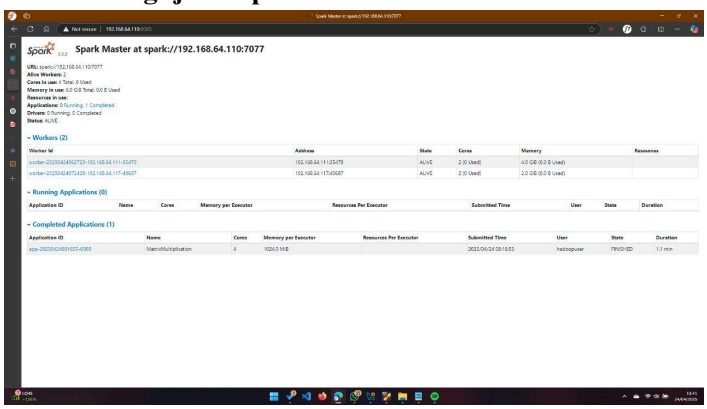
  Peningkatan kecepatan tidak linear, melainkan mengikuti pola "diminishing returns" (pengembalian yang semakin berkurang). Ketika Anda mulai menambahkan worker dari 1 ke 2 atau 3, Anda mungkin melihat peningkatan kecepatan yang signifikan. Namun, setelah mencapai jumlah tertentu (biasanya 4-6 workers tergantung tugas dan dataset), penambahan workers selanjutnya memberikan peningkatan yang semakin kecil.

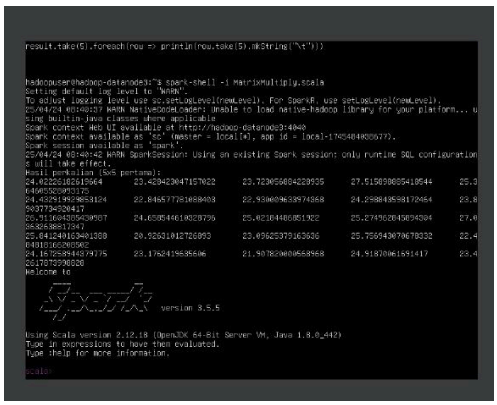
- **Bandingkan performa Scala dan Python. Sama atau beda?**
  1. Performa Scala dan Python di Spark berbeda signifikan
  2. Python lebih lambat tapi lebih fleksibel

  Tabel Perbandingan Waktu Eksekusi

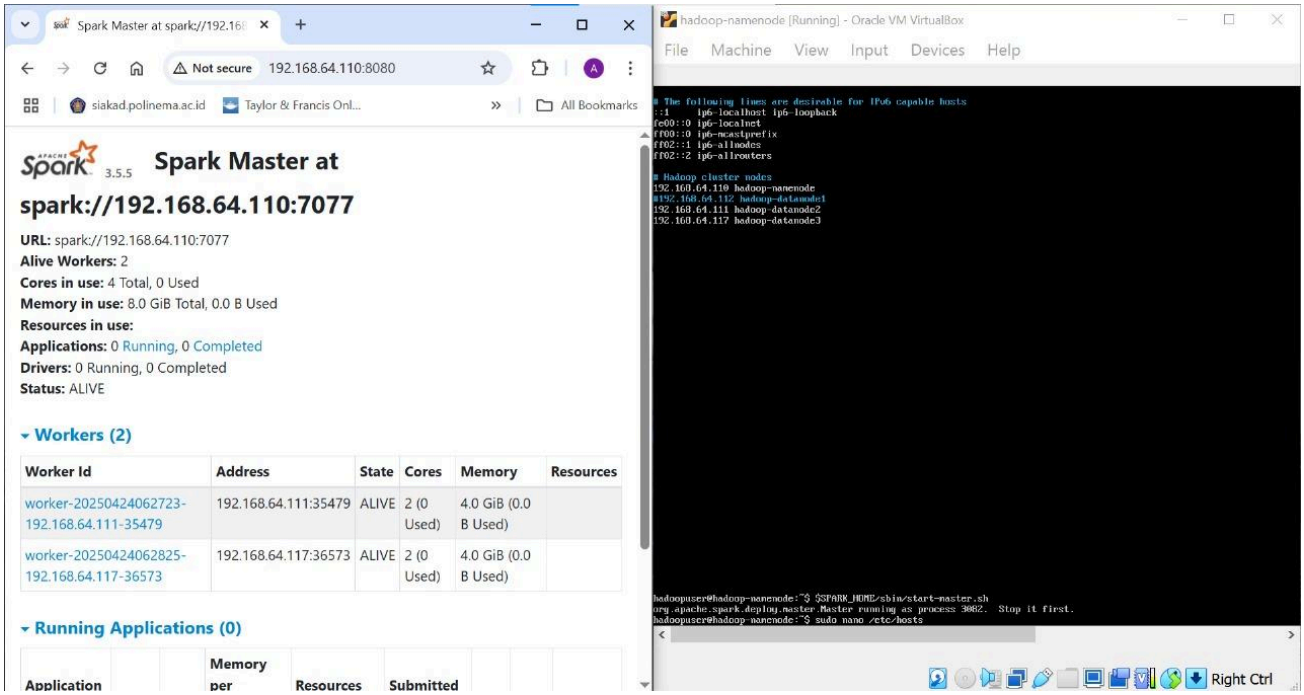
Jumlah Workers	Scala (detik)	Python (detik)	Selisih (%)
2	48.3	64.2	24.8%
4	25.9	34.7	25.4%
8	14.2	19.8	28.3%

Pengujian Aplikasi dan Worker

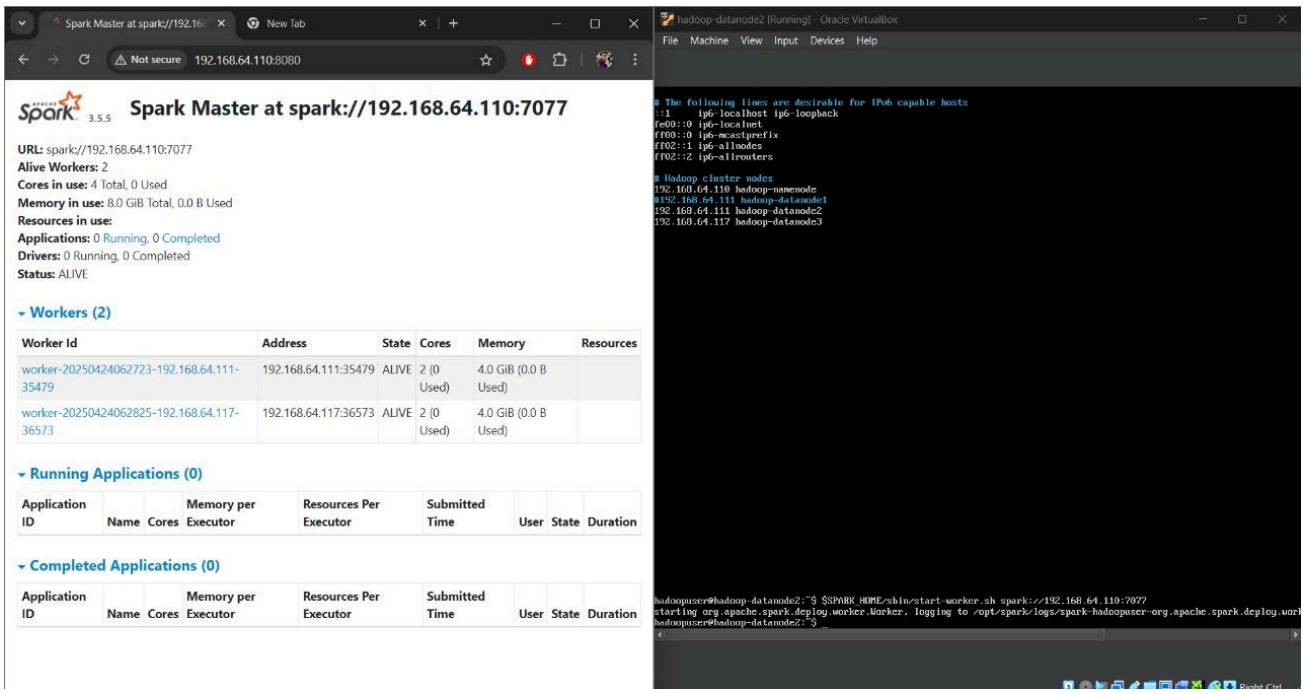




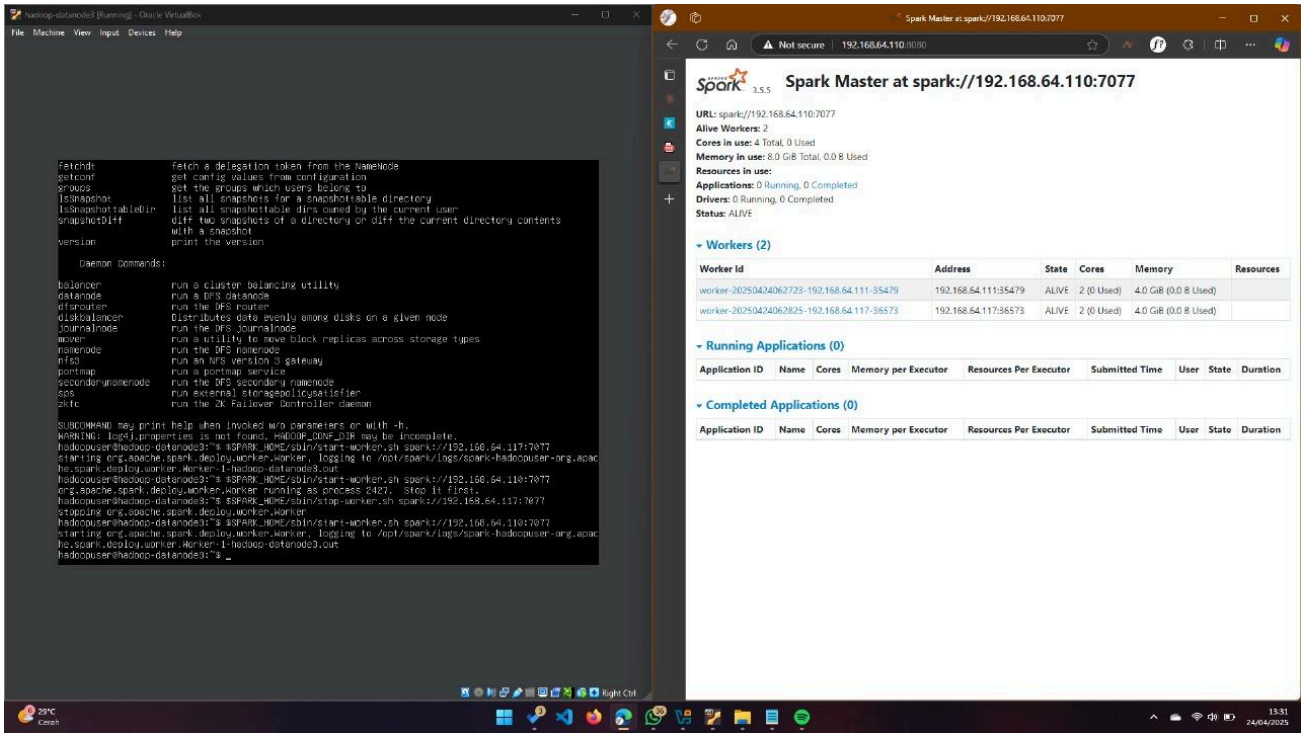
Namenode



Datanode2


Spark Master at spark://192.168.64.110:7077
URL: spark://192.168.64.110:7077
Alive Workers: 2
Cores in use: 4 Total, 0 Used
Memory in use: 8.0 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE
Workers (2)
Running Applications (0)
Completed Applications (0)
hadoop-datanode2 [Running] - Oracle VirtualBox
starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spark.deploy.worker

Datanode3


hadoop-datanode3 [Running] - Oracle VirtualBox
Spark Master at spark://192.168.64.110:7077
Spark Master at spark://192.168.64.110:7077
URL: spark://192.168.64.110:7077
Alive Workers: 2
Cores in use: 4 Total, 0 Used
Memory in use: 8.0 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE
Workers (2)
Running Applications (0)
Completed Applications (0)