Sesi 24
Spark (2)

Spark Library dan Pemrograman dengan Spark

Garis Besar

- Mempelajari SparkContext
- Menjalankan Spark dengan berbagai bahasa pemrograman
- Mencoba berbagai contoh Spark
- Pass fungsi ke Spark
- Membuat aplikasi Spark
- Pasang ke cluster
- Library Spark – SparkSQL, SparkStreaming, MLlib, dan GraphX
- Mempelajari Spark cluster
- Konfigurasi Spark properties, environment variable, logging
- Monitor Spark dengan webUI, metric dan instrumentasi eksternal
- Performance tuning

Spark Application Programming

SparkContext

- Main entry point ke fungsionalitas Spark
- Koneksi ke Spark cluster
- Create RDD, accumulator, broadcast variable pada cluster
- Spark Shell SparkContext, sc terinisialisasi otomatis
- Spark app harus import classes dan konversi implisit buat SparkContext object

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
```

Link Spark dengan Scala

- Spark app butuh beberapa dependencies
- Versi harus kompatibel (contoh Spark 1.1.1 ↔ Scala 2.10)
- Write Spark app → tambahkan Maven dependency di Spark
  - Spark tersedia lewat Maven Central:

```
groupId = org.apache.spark
artifactID = spark-core_2.10
version = 1.1.1
```

- Akses HDFS cluster → tambahkan dependency di hadoop-client untuk versi HDFS-nya

```
groupID = org.apache.hadoop
artifactID = hadoop-client
version = <your-hdfs-version>
```

Link Spark dengan Python

- Spark 1.1.1 bekerja dengan Python 2.6 atau lebih (tidak Python 3)
- Spark 2.4.3 (terbaru sekarang) bekerja dengan Python 2.7+ dan 3.4+
- Gunakan standar CPython interpreter, C library seperti NumPy bisa digunakan
- Run Spark app di Python `bin/spark-submit` script di direktori home Spark
  - Load Spark Java/Scala library
  - Mengizinkan submit app ke cluster
- Akses HDFS link PySpark ke versi HDFS
- Impor beberapa kelas Spark
  from pyspark import SparkContext, SparkConf

Link Spark dengan Java

- Support Java 6+, Java 8 support Lambda expression Jika pakai versi lama bisa
  `org.apache.spark.api.java.function`
- Tambah dependency di Spark
  - Tersedia lewat Maven Central
  ```
  groupId = org.apache.spark
  artifactID = spark-core_2.10
  version = 1.1.1
  ```
- Akses HDFS cluster tambah dependency
  ```
  groupID = org.apache.hadoop
  artifactID = hadoop-client
  version = <your-hdfs-version>
  ```
- Import beberapa kelas Spark
  ```
  import org.apache.spark.api.java.JavaSparkContext
  import org.apache.spark.api.java.JavaRDD
  import org.apache.spark.SparkConf
  ```

Inisialisasi Spark - Scala

- Sebelum inisialisasi Spark build SparkConf object berisi informasi app. Contoh:

```
val conf = new SparkConf().setAppname(appName).setMaster(master)
```

- `appName` parameter nama app di Cluter UI
- `master` parameter URL Spark, Mesos, atau YARN cluster (bisa pakai local keyword string local mode)
  - di testing, bisa passing “local” untuk run Spark
  - local[16] alokasi 16 core
  - production mode jangan hardcode `master` path di program pakai command argumen `spark-submit`
- Selanjutnya, buat SparkContext object pass sebagai parameter

```
new SparkContext (conf)
```

Inisialisasi Spark - Python

- Sebelum inisialisasi Spark build SparkConf object berisi informasi app. Contoh:
  `conf = SparkConf().setAppName(appName).setMaster(master)`
- `appName` parameter nama app di Cluter UI
- `master` parameter URL Spark, Mesos, atau YARN cluster (bisa pakai local keyword string local mode)
  - di testing, bisa passing “local” untuk run Spark
  - production mode jangan hardcode `master` path di program pakai command argumen `spark-submit`
- Selanjutnya, buat SparkContext object pass sebagai parameter
  `sc = SparkContext(conf=conf)`

Inisialisasi Spark - Java

- Sebelum inisialisasi Spark build SparkConf object berisi informasi app. Contoh:

```
SparkConf conf = new SparkConf().setAppname(appName).setMaster(master)
```

- `appName` parameter nama app di Cluter UI
- `master` parameter URL Spark, Mesos, atau YARN cluster (bisa pakai local keyword string local mode)
  - di testing, bisa passing “local” untuk run Spark
  - production mode jangan hardcode `master` path di program pakai command argumen `spark-submit`
- Selanjutnya, buat SparkContext object pass sebagai parameter

```
JavaSparkContext sc = new JavaSparkContext(conf);
```

Passing Function ke Spark

Passing Function ke Spark

- Spark API bergantung passing function di driver program jalan di cluster
- Spark driver info ke worker cara poses data
- Tiga metode
  - Anonymous function syntax – berguna untuk 1x pakai, tidak perlu define eksplisit
    ```
    (x: Int) => x + 1
    ```
    *parameter atau argumen => body function
  - Static method di global singleton object – create global object define function ketika driver butuh function tsb kirim object-nya ke worker
    ```
    object MyFunction{

       def func1 (s: String): String = {...}
    }
    myRdd.map(MyFunctions.func1)
    ```

- Passing by reference – pass reference ke method di sebuah class instance , menghindari kirim keseluruhan object copy function ke variabel local

  Contoh:

```
val field = “Hello”
```

  Hindari:

```
def doStuff(rdd: RDD[String]):RDD[String] = {rdd.map(x =>
field + x)}
```

  Pertimbangkan:

```
def doStuff(rdd: RDD[String]):RDD[String] = {
val field_ = this.field
rdd.map(x => field_ + x)}
```

*Bayangkan jika, log file besar pass by ref greater value (saving storage, tidak pass semua file)

Aktivitas Kelas

Programming the business logic

- Spark's API available in Scala, Java, or Python.
- Create the RDD from an external dataset or from an existing RDD.
- Transformations and actions to process the data.
- Use RDD persistence to improve performance
- Use broadcast variables or accumulators for specific use cases

```scala
package org.apache.spark.examples

import org.apache.spark._


object HdfsTest {

  /** Usage: HdfsTest [file] */
  def main(args: Array[String]) {
    if (args.length < 1) {
      System.err.println("Usage: HdfsTest <file>")
      System.exit(1)
    }
    val sparkConf = new SparkConf().setAppName("HdfsTest")
    val sc = new SparkContext(sparkConf)
    val file = sc.textFile(args(0))
    val mapped = file.map(s => s.length).cache()
    for (iter <- 1 to 10) {
      val start = System.currentTimeMillis()
      for (x <- mapped) { x + 2 }
      val end = System.currentTimeMillis()
      println("Iteration " + iter + " took " + (end-start) + " ms")
    }
    sc.stop()
  }
}
```

Running Spark - Example

- Spark sample tersedia di direktori examples, website, dll
- Run examples:
  `./bin/run-example SparkPi` (SparkPi contoh nama app)
- Di Python:
  `./bin/spark-submit examples/src/main/python/pi.py`

Run Spark Standalone App

Run Spark Standalone App

- Define dependencies package app build tool (Ant, sbt, atau Maven)
- Contoh:
  - Scala `simple.sbt`
  - Java `pom.xml` pakai Maven
  - Python `--py-files` argumen

Scala using SBT :	Java using Maven:
./simple.sbt	./pom.xml
./src	./src
./src/main	./src/main
./src/main/scala	./src/main/java
./src/main/scala/SimpleApp.scala	./src/main/java/SimpleApp.java

- Create JAR package berisi app code Scala dan Java
- Set .py atau .zip Python
- Submit ke Spark cluster `spark-submit` di `$SPARK_HOME/bin`

Submit App ke Cluster

- Use spark-submit under the $SPARK_HOME/bin directory

```
./bin/spark-submit \
--class <main-class> \
--master <master-url> \
--deploy-mode <deploy-mode> \
--conf <key>=<value> \
... # other options
<application-jar> \
[application-arguments]
```

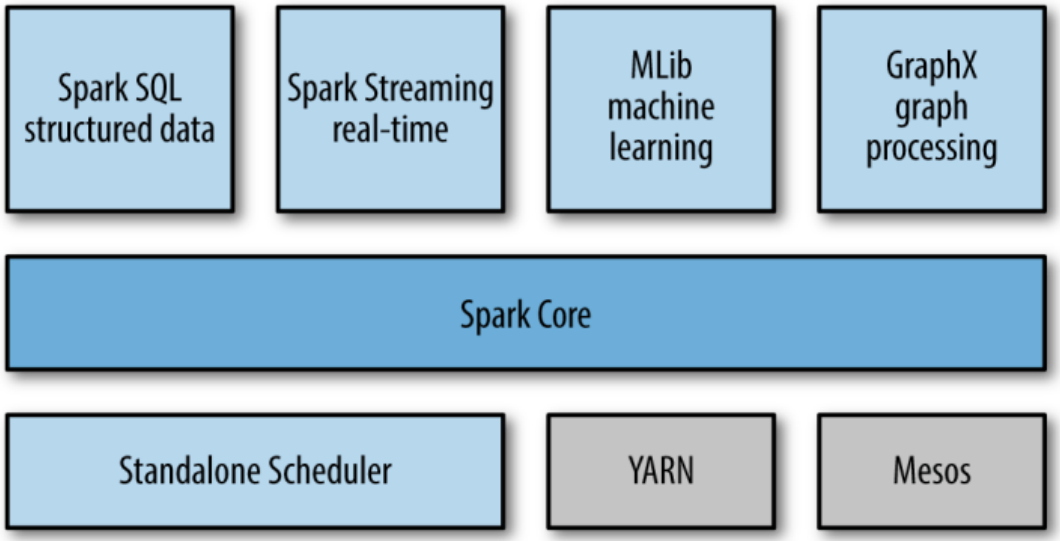
- `spark-submit --help` will show you the other options
- Example of running an application locally on 8 cores:

```
./bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master local[8] \
/path/to/examples.jar \
100
```

Library Spark

Spark Libraries

- Ekstensi core Spark API
- Peningkatan core library juga dapat peningkatannya
- Overhead kecil untuk digunakan



Spark SQL (1)

- Relational query
  - SQL
  - HiveQL
  - Scala
- SchemaRDD RDD khusus Spark SQL
  - row object
  - schema (deskripsi tipe data tiap kolom)
    Tabel di relational database tradisional
  - Dibuat dari
    Existing RDD
    Parquet file
    JSON dataset
    HiveQL query data di Hive
- Support Scala, Java, dan Python

Spark SQL (2)

- SQLContext – dibuat dari SparkContext
  Scala:
```
val sc: SparkContext //an existing SparkContext
val sqlContext = new org.apache.spark.sql.SQLContext (sc)
```
  Java:
```
JavaSparkContext sc = ...; //an existing JavaSparkContext
JavaSQLContext sqlContext = new
org.apache.spar.sql.api.java.JavaSQLContext(sc);
```
  Python:
```
from pyspark.sql import SQLContext sqlContext =
SQLContext(sc)
```
- Impor library untuk konversi RDD ke SchemaRDD
  - Scala: `import sqlContext.createSchemaRDD`
  - Python dan Java tidak perlu library

Spark SQL (3)

- Sumber data SchemaRDD:
  - Refleksi untuk infer skema RDD – ringkas dan bagus ketika sudah tahu skema ketika membuat Spark app
  - Programming interface – construct skema dan apply ke RDD. More control ketika belum tahu skema RDD hingga runtime

Spark SQL – Infer Skema dengan Refleksi

- Case class di Scala define skema tabel, argumen read dengan refleksi dan menjadi nama kolom
```
case class Person(name: String, age: Int)
```
- Create RDD object Person
```
val people =
sc.textFile(“examples/src/main/resources/people.txt”).mpa(_.sp
lit(“,”)).map(p => Person(p(0), p(1).trim.toInt))
```
- Register RDD sebagai tabel –
```
people.registerTempTable(“people”)
```
- Run SQL dengan sql method dari SQLContext
```
val teenagers = sqlContext.sql(“SELECT name FROM people WHERE
age >= 13 AND <= 19”)
```
- Hasil query SchemaRDD. Operasi RDD normal juga jalan
```
teenager.map(t => “Name: “ + t(0)).collect().foreach(println)
```

Spark SQL – Programming Interface (1)

- Digunakan ketika tidak bisa define case classes sebelumnya
  Contoh: ketika struktur record di-encode dalam string/text dataset akan di-parse dan field akan diproyeksikan berbeda untuk user yang berbeda juga
- Create RDD – `val people = sc.textFile(...)`
- Tiga langkah membuat SchemaRDD
  1. Create RDD baris dari RDD original – buat SchemaString
  `val schemaString = “name age”`
  2. Create skema representasi dari StructType yang match struktur baris step 1 map ke StructField
  `val schema = StructType( schemaString.split(“ “).map(fieldName => StructField(fieldName, StringType, true)))`

Spark SQL – Programming Interface (2)

3. Apply skema ke objek RDD baris dengan method applySchema

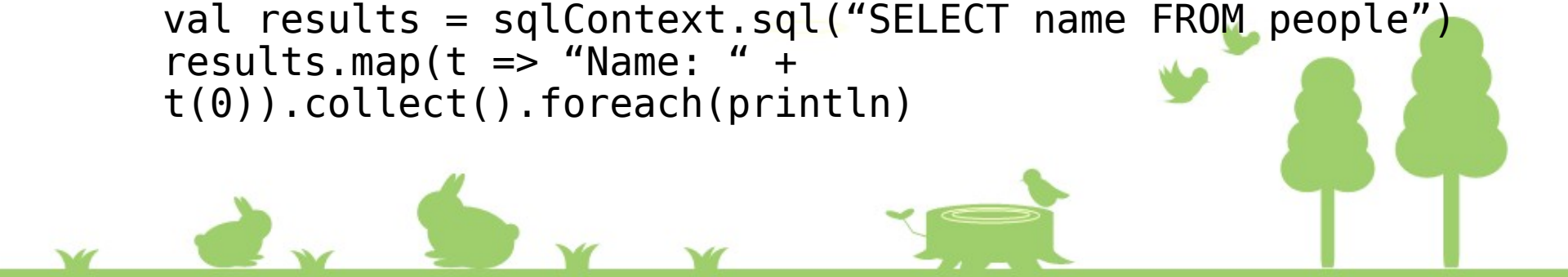
```
val rowRDD = people.map(_.split(“,”)).map(p => Row(p(0), p(1).trim))
val peopleSchemaRDD = sqlContext,applySchema(rowRDD, schema)
```

- Register peopleSchemaRDD sebagai tabel

```
peopleSchemaRDD.registerTempTAble(“people”)
```

- Run sql dengan sql method

```
val results = sqlContext.sql(“SELECT name FROM people”)
results.map(t => “Name: “ + t(0)).collect().foreach(println)
```



Spark Streaming (1)

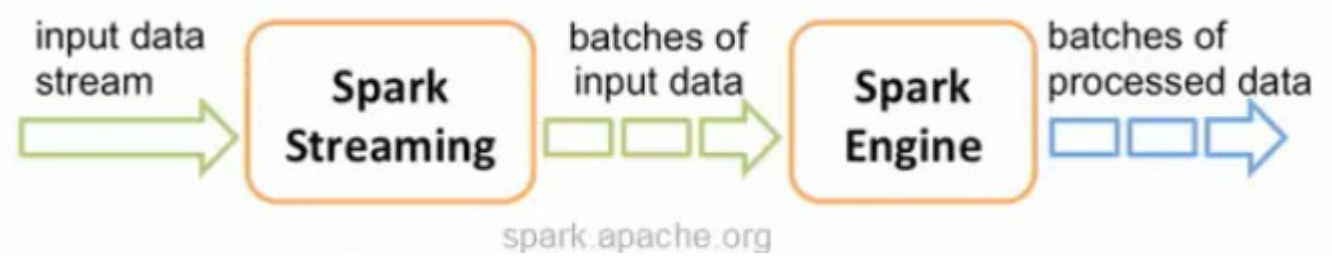
- Scalabel, high-throughput, fault-tolerant, stream process live data stream
- Terima input data live dibagi batch-batch kecil diproses dan return sebagai batches
- Write Stream app DStream – sequence RDD
- Support Scala dan Java
  Python Spark 1.2+

Spark Streaming (2)

- Terima data dari
  - Kafka
  - Flume
  - HDFS/S3
  - Kinesis
  - Twitter
- Publish data ke
  - HDFS
  - Database
  - Dashboard

Spark Streaming – Internal (1)

- Input stream (DStream) Spark streaming
- Dipecah ke batch-batch
- Dimasukkan ke Spark engine processing
- Hasil stream batch-batch



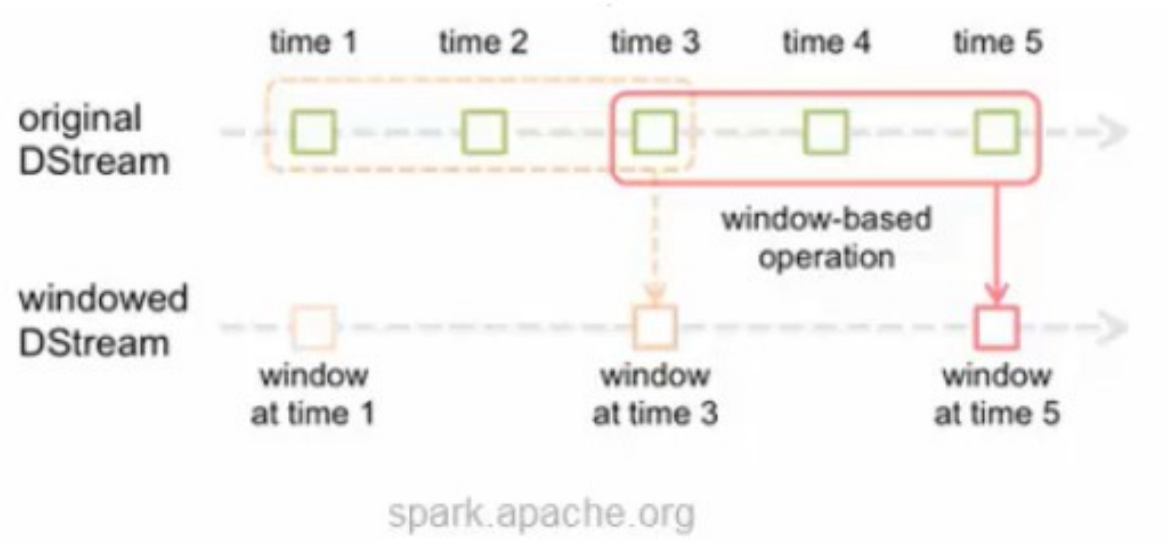
Spark Streaming – Windowed Computation (1)

- Support sliding window – windowed computation
  ketika window slide di source DStream, source RDD yang jatuh di dalam window dikombinasikan dan dioperasikan produce result RDD
  - window length durasi window
  - sliding interval interval window operation yang beroperasi

  Keduanya harus ada di multiple batch interval dari source DStream

Spark Streaming – Windowed Computation (2)

- Window length 3, sliding interval 2
- Perspektif berbeda contoh: generate word count 30 detik data, tiap 10 detik
  - reduceByKeyandWindow



Spark Streaming – Use Case (1)

- Hitung jumlah kata yang datang dari TCP socket
- Impor class Spark Streaming dan beberapa konversi implisit

```
import orang.apache.spark._
import orang.apache.spark.streaming._
import orang.apache.spark.streaming.StreamingContext._
```

- Create StreamingContext object

```
val conf = new
SparkConf().setMaster(“local[2]”).setAppName(“NetworkWordCo
unt”)
val ssc = new StreamingContext(conf, Seconds(1))
```

- Create Dstream

```
val lines = ssc.socketTextStream(“localhost”, 9999)
```

- Split lines ke kata-kata

```
val words = lines.flatMap(_.split(“ “))
```

Spark Streaming – Use Case (2)

- Hitung kata-kata

```
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_+_)
```

- Print ke console

```
wordCounts.print()
```

Spark Streaming - Note

- No real processing hingga

```
ssc.start() //start computation
ssc.awaitTermination() //wait for the computation to terminate
```

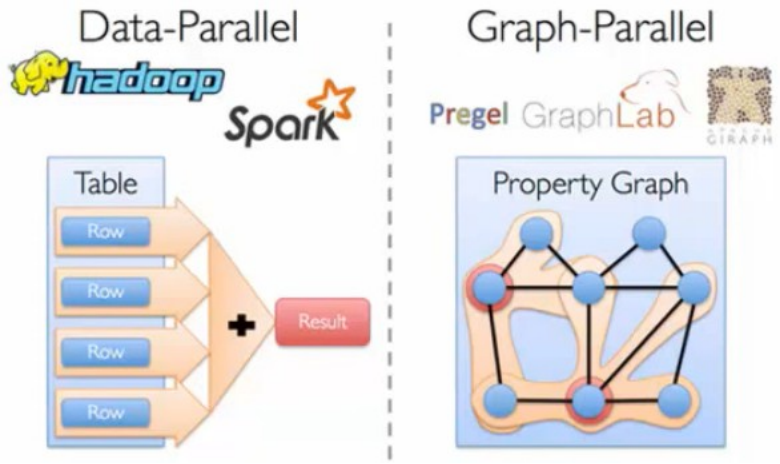
- Code dan app contoh UseCase NetworkWordCount example
- Run example
  - run netcat untuk start data stream
  - di terminal berbeda, run app

```
./bin/run-example streaming.NetworkWordCount localhost 9999
```

MLlib

- Machine Learning library:
  - classification
  - regression
  - clustering
  - collaborative filtering
  - dimensionality reduction

GraphX



- Untuk graph processing
  - graph dan komputasi paralel graph
  - sosial media dan modeling bahasa
- Optimasi proses view data graph dan koleksi (RDD) tanpa perpindahan data dan duplikasi
- Skenario spesifik tidak akan efisien, jika diproses dengan model data-paralel
- Komputasi paralel graph – eksekusi algoritma graph lebih cepat
  - Giraph
  - GraphLab

Graph Inherent Challenge

- Constructing graph
- Modifikasi struktur
- Expressing computation

- Challenges ini butuh span beberapa graph
- Seringkali diperlukan pindah antar tabel dan graph view tergantung objective app dan business requirement