MapReduce w Apache Spark

30 marca 2019

Opis pliku z zadaniami

Wszystkie zadania na zajęciach będą przekazywane w postaci plików `.pdf`, sformatowanych podobnie do tego dokumentu. Zadania będą różnego rodzaju. Za każdym razem będą one odpowiednio oznaczone:

- Zadania do wykonania na zajęciach oznaczone są symbolem $\triangle$ – nie są one punktowane, ale należy je wykonać w czasie zajęć.
- Punktowane zadania do wykonania na zajęciach oznaczone są symbolem $\diamond$ – należy je wykonać na zajęciach i zaprezentować prowadzącemu, w wypadku nie wykonania zadania w czasie zajęć lub nieobecności, zadania staje się zadaniem do wykonania w domu ($\star$).
- Zadania do wykonania w domu oznaczone są symbolem $\star$ – są one punktowane, należy je dostarczyć w sposób podany przez prowadzącego i w wyznaczonym terminie (zwykle przed kolejnymi zajęciami).
- Zadania programistyczne można wykonywać w dowolnym języku programowania, używając jedynie biblioteki standardowej dostępnej dla tego języka.

1 Pierwsze kroki z Apache Spark △

Treść

Na zajęciach będziemy wykorzystywać Apache Spark, w którym będziemy stosować obliczenia oparte na paradygmacie MapReduce.

Do rozpoczęcia pracy należy wykonać następujące kroki:

1. Zapoznać się z najważniejszymi informacjami dotyczącymi Apache Spark: https://spark.apache.org/docs/latest/index.html
2. Pobrać paczkę instalacyjną ze strony: http://spark.apache.org/downloads.html
3. Rozpakować pobraną paczkę, np.: `tar xvfz spark-2.2.0-bin-hadoop2.7.tar`
4. Rozpocząć pracę z systemem poprzez konsolę: `./bin/spark-shell`

2 Proste zadania w Apache Spark △

Treść

Wykonaj następujące zadania:

- `WordCount`: Należy napisać program tworzący plik zawierający listę słów wraz z liczbą ich wystąpień w dziełach Szekspira (odpowiedni plik został załączony na stronie przedmiotu). Lista słów powinna zostać posortowana zgodnie z liczbą ich wystąpień. Program powinien wykorzystywać następujące instrukcje: `textFile`, `FlatMap`, `map`, `reduceByKey`, `sortBy(_._2)`, `saveAsTextFile`. Adresowanie pól krotek odbywa się w następujący sposób: `nazwa_krotki._1`, gdzie `_1` jest pierwszym polem. Rozwiązanie można znaleźć w materiałach z wykładu. Po uruchomieniu programu należy sprawdzić zapisany wynik i odpowiedzieć na następujące pytania:
  - Jak wyglądają pliki wynikowe?
  - Ile jest takich plików? Dlaczego tak jest?
  - Jak działa `reduceByKey`? Co się stanie, jak zamienimy '`+`' na '`*`'?
- `MatrixVectorMultiplication`: Należy napisać program obliczający iloczyn macierzy i wektora zapisanych w sposób relacyjny (odpowiednie pliki zostały dołączone do strony przedmiotu). Zakładamy, że wektor może zostać rozesłany do wszystkich komputerów w klastrze. Program powinien wykorzystywać następujące instrukcje: `textFile`, `map`, `toInt`, `toDouble`, `collect`, `FlatMap`, `reduceByKey`, `broadcast`, `toDF`, `orderBy`, `show`. Adresowanie elementów listy odbywa się w następujący sposób: `nazwa_listy(i)`, gdzie `i` jest liczbą całkowitą równą lub większą od zera. Rozwiązanie można znaleźć w materiałach z wykładu.
- `ApproximatePI`: Należy napisać program, który przybliża liczbę π metodą Monte Carlo. Program powinien wykorzystywać następujące instrukcje: `parallelize`, `map`, `1 to 10`, `math.random`, `if (.) . else .`, `reduce`.

3 Agregacja danych *5p.*◇

Treść

Dla losowych danych zawierających indeks grupy oraz wartość ciągłą pochodzącą z rozkładu normalnego oblicz statystyki: liczbę przykładów, średnią oraz wariancję. Statystyki należy policzyć dla całego zbioru oraz dla każdej grupy z osobna. Do obliczeń należy wykorzystać operacje `map`, `reduce`, oraz `reduceByKey`. Oblicz statystyki przechodząc jednokrotnie przez zbiór danych.

Obliczenia dla całego zbioru danych wykonaj na dwa sposoby. W pierwszym policz statystyki na pierwotnych danych. W drugim wykorzystaj wyniki policzone dla grup. Sprawdź, czy takie podejście przyśpiesza obliczania. Czy skorzystanie z metody `cache` wpływa na szybkość obliczeń? W celu monitorowania zadań i sprawdzenia czasu ich wykonywania można skorzystać z interfejsu webowego `http://localhost:4040/jobs/`.

W celu wygenerowania danych skorzystaj z poniższego kodu:

```
import scala.util.Random

val m = 1000

val random = Random

val groups = Map((for (i <- 0 to m) yield {
  val mu = random.nextDouble()*10-5
  val std = random.nextDouble()*10-5
        (i -> (mu,std))}): _*)

val n = 1000000

val data = for (i <- 1 to n) yield {val g = random.nextInt(m);
    val (mu, sigma) = groups(g); (g, mu + sigma*random.
    nextGaussian())}

val dataRDD = sc.parallelize(data)

//Check 10 records
data.take(10)
```

4 Mnożenie macierzy *5p.*◇

Treść

Napisz program, który pozwoli na mnożenie dwóch macierzy przez siebie. Zakładamy, że żadna z macierzy nie mieści się w pamięci i obie należy wczytać jako dane wejściowe. Plik `result_mm.txt` zawiera oczekiwany wynik mnożenia macierzy z pliku `M.txt` przez macierz zawartą w pliku `N.txt`.

Format plików wejściowych: `wiersz kolumna wartość`

Podpowiedź:

- Zajrzyj do wykładu :)
- Wczytaj dane podobnie jak w przypadku mnożenia macierzy przez wektor,
- Odpowiednio przemapuj wczytane dane (`map`),
- Do wykonania zadania wystarczy wykorzystać następujące polecenia: `join`, `map`, `reduceByKey`, `sortByKey`, `collect`, `foreach`, `println`.

5 Zapytania do danych MSDC *15p.*◇

Treść

Wykorzystując pliki `.csv`, przygotowane na zajęciach dotyczących przetwarzania danych MSDC w powłoce bash, zapisz w Sparku poniższe zapytania (są to te same zapytania, jak na wcześniejszych zajęciach dotyczących danych MSDC):

- Ranking popularności piosenek,
- Ranking użytkowników ze względu na największą liczbę odsłuchanych unikalnych utworów,
- Artysta z największą liczbą odsłuchań,
- Sumaryczna liczba odsłuchań w podziale na poszczególne miesiące,
- Wszyscy użytkownicy, którzy odsłuchali wszystkie trzy najbardziej popularne piosenki zespołu Queen.

Można wykorzystać następujące instrukcje: `spark.read.csv`, `groupBy`, `count`, `join`, `select`, `orderBy`,`show`, `agg`, `countDistinct`, `col`, `toDF`, `limit`, `filter`.

Dla ułatwienia wykonania zadania poniżej przedstawiony jest kod dla pierwszego zapytania:

```
//Read data
val songs = spark.read.
                  option("delimiter", ",").
                  csv("songs").
                  toDF("song_id", "track_long_id", "
                      song_long_id", "artist", "song")
val facts = spark.read.
                  option("delimiter", ",").
                  csv("facts").
                  toDF("id", "user_id", "song_id", "
                      date_id")

//The most popular songs
facts.groupBy("song_id").
      count.
      join(songs, facts("song_id")===songs("song_id")).
      select("song", "count").
      orderBy(desc("count")).
      show(10)
```

spark–msdc–1.scala