MapReduce w Apache Spark I

20 grudnia 2018

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.
- 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).

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

W tym zadaniu należy wykonać 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 Zapytania do danych MSDC *bonus 5p.⋆*

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

Za powyższe zadanie można otrzymać punkty bonusowe, po 1 punkcie za każde poprawnie wykonane zapytanie.