SP03_l1_23-Spark-RDD

November 4, 2023

1 Wprowadzenie

RDD to podstawowa (niskopoziomowa) abstrakcja danych w Sparku.

1.1 Podstawy

1.1.1 Co oznacza RDD

- ***Resilient*** – odporne na błędy, RDD powstają bezpośrednio lub pośrednio z trwałych źródeł danych. Dzięki temu, że znany jest graf transformacji, awaria węzła i utrata części danych RDD może być zniwelowana przez ponowne wyliczenie utraconej porcji.
- ***Distributed*** – poszczególne porcje (partycje) danych rezydują na wielu węzłach klastra.
- ***Dataset*** – RDD to partycjonowana kolekcja prostych wartości lub wartości o bardziej złożonej strukturze (np. rekordów).

1.1.2 Cechy

- RDD rezydują w pamięci w takim zakresie i przez taki długi okres czasu jak to tylko możliwe.
- RDD są niezmienne (tylko do odczytu), transformacje niczego nie zmieniają a jedynie tworzą nowe RDD.
- Przetwarzanie RDD jest leniwe – transformacje są ewaluowane dopiero gdy akcja uruchomi graf transformacji.
- W zależności od potrzeb istnieje możliwość zapisania trwałej postaci pośrednich RDD w pamięci (preferowane i domyślne) lub na dysku.
- RDD przetwarzane są równolegle przez wiele węzłów.
- RDD jest partycjonowany i rozpraszany pomiędzy węzły klastra.

1.2 Kiedy używać RDD

- Gdy chcemy wykonać podstawowych (niskopoziomych?) transformacji i akcji na danych;
- Dane są niestrukturalne np.: pliki lub strumienie tekstowe;
- Gdy podczas przetwarzania wolimy bardziej oprzeć się na konstrukcjach programowania funkcyjnego niż na wyrażeniach przetwarzania dziedzinowego;
- Nie zależy nam na uwzględnianiu schematów danych jakie występują w formacie kolumnowym, gdzie odwołujemy się do danych za pomocą nazw lub kolumn
- Możemy pozwolić sobie na rezygnację z optymalizacji dostępnych w *Spark SQL*, wykorzystywanych podczas przetwarzania danych strukturalnych i semi-strukturalnych.

1.3 Dokumentacja

https://spark.apache.org/docs/latest/api/python/reference/pyspark.html

[1]:
```
# w przypadku korzystania z kernela Python
from pyspark import SparkContext, SparkConf
```

[3]:
```
# w przypadku korzystania z kernela Python
conf = SparkConf().setAppName("Spark - RDD").setMaster("local")
sc = SparkContext(conf=conf)
```

[3]:
```
# w przypadku korzystania z kernela Python
# w przypadku korzystania z klastra Hadoop
conf = SparkConf().setAppName("Spark - RDD").setMaster("yarn")
sc = SparkContext(conf=conf)
```

[4]:
```
sc
```

[4]:
```
<SparkContext master=local appName=Spark - RDD>
```

2 Ładowanie danych

2.1 Z elementów kolekcji

Za pomocą metody `parallelize` obiektu `SparkContext`. Kolekcja danych zostaje rozproszona na węzły klastra. Jej elementy trafiają do poszczególnych partycji obiektu `RDD`.

[5]:
```
ravenPoe = [
"Once upon a midnight dreary, while I pondered, weak and weary, ",
"Over many a quaint and curious volume of forgotten lore ",
"While I nodded, nearly napping, suddenly there came a tapping, ",
"As of some one gently rapping, rapping at my chamber door. ",
"'Tis some visitor,' I muttered, 'tapping at my chamber door ",
"Only this and nothing more.'"]

distRavenPoe = sc.parallelize(ravenPoe)
distRavenPoe.count()
```

[5]:
```
6
```

2.2 Z zewnętrznych źródeł danych

- `textFile` - pliki tekstowe - każda linia oddzielnym rekordem w `RDD`
- `wholeTextFiles` - zbiór małych plików, każdy z nich odzielnym rekordem (nazwa pliku, zawartość)
- `pickleFile` - pliki w prostym formacie zserializowanych obiektów Pythona (*pickled Python objects*)
- `sequenceFile` - pliki *SequenceFiles*

[6]:
```
%%sh
# w przypadku korzystania z klastra Hadoop (yarn)
hadoop fs -rm -r raven.pickle
```

[7]:
```
# w przypadku uruchamiania Sparka lokalnie
rm raven.pickle
```

```
rm: cannot remove 'raven.pickle': No such file or directory
```

[8]:
```
distRavenPoe.saveAsPickleFile("raven.pickle")
```

[7]:
```
%%sh
# w przypadku korzystania z klastra Hadoop
hadoop fs -ls raven.pickle
```

[10]:
```
%%sh
# w przypadku uruchamiania Sparka lokalnie
ls raven.pickle
```

```
part-00000
_SUCCESS
```

[11]:
```
tmp = sc.pickleFile("raven.pickle")
tmp.collect()
```

[11]:
```
['Once upon a midnight dreary, while I pondered, weak and weary, ',
 'Over many a quaint and curious volume of forgotten lore ',
 'While I nodded, nearly napping, suddenly there came a tapping, ',
 'As of some one gently rapping, rapping at my chamber door. ',
 "'Tis some visitor,' I muttered, 'tapping at my chamber door ",
 "Only this and nothing more.'"]
```

2.3 Operacje

2.3.1 Transformacje

Przekształcają obiekt RDD w nowy obiekt RDD

- *lazy* – wyliczane tylko wówczas gdy wymaga tego wykonywana akcja (wzrost wydajności)
- Tworzą graf przetwarzania *DAG*
- Przeliczane za każdym razem gdy wykorzystywana jest akcja
- Możliwe jest zachowanie ich rezultatu (za pomocą metod `persist` lub `cache`) w celu ich ponownego wykorzystania bez przeliczania

[12]:
```
lineLengths = distRavenPoe.map(lambda s: len(s))
```

2.3.2 Akcje

Zwracają wyniki do programu sterownika (*driver*), ewentualnie wysyłają/zapisują wyniki do - plików zewnętrznych - tabel zarządzanych przez Sparka - innych zewnętrznych systemów (np. *Apache Kafka*, *GFS*)

[13]:
```
lineLengths.reduce(lambda a, b: a + b)
```

[13]:
```
329
```

3 Transformacje

3.1 Wprowadzenie

- `map(func)` – zwraca nowy RDD będący wynikiem działania `func` na każdym elemencie danych (1->1)
- `filter(func)` – zwraca nowy RDD posiadający tylko te elementy, dla których `func` daje wartość true
- `flatMap(func)` – podobnie jak `map` (1->0..n) przy czym wyniki mające złożoną postać są dekomponowane na elementy składowe
- `sample(withReplacement, fraction, seed)` – pobiera losowy zbiór danych z podanym prawdopodobieństwem
- `union|intersection(otherDataset)` – tworzy sumę|cześć wspólną zbiorów dwóch RDD
- `distinct([numTasks])` – zwraca unikalne wartości RDD

3.2 Przykłady 1

[14]:
```
distRavenPoe.first()
```

[14]:
```
'Once upon a midnight dreary, while I pondered, weak and weary, '
```

[15]:
```
distRavenPoe.filter(lambda line: "tapping" in line).first()
```

[15]:
```
'While I nodded, nearly napping, suddenly there came a tapping, '
```

[16]:
```
distRavenPoe.map(lambda line: len(line)).first()
```

[16]:
```
63
```

[17]:
```
distRavenPoe.map(lambda line: line.split(" ")).first()
```

[17]:
```
['Once',
 'upon',
 'a',
 'midnight',
 'dreary,',
 'while',
 'I',
```

```
 'pondered,',
 'weak',
 'and',
 'weary,',
 '']
```

[18]:
```
distRavenPoe.flatMap(lambda line: line.split(" ")).first()
```

[18]:
```
'Once'
```

3.3 Przykłady 2

[19]:
```
distRavenPoe.flatMap(lambda line: line.split(" ")).count()
```

[19]:
```
62
```

[20]:
```
wordsRavenPoe = distRavenPoe.flatMap(lambda line: line.split(" "))
```

[21]:
```
wordsRavenPoe.sample(False, .5).count()
```

[21]:
```
31
```

[22]:
```
wordsRavenPoe.map(lambda word: (word,1)).reduceByKey(lambda a, b: a + b).first()
```

[22]:
```
('Once', 1)
```

[23]:
```
wordsRavenPoe.filter(lambda word: word.endswith("y")).intersection(
    wordsRavenPoe.filter(lambda word: word.startswith("m"))).collect()
```

[23]:
```
['many', 'my']
```

3.4 Przykłady 3

[24]:
```
dreamPoe = ["Take this kiss upon the brow!",
"And, in parting from you now,",
"Thus much let me avow --",
"You are not wrong, who deem",
"That my days have been a dream;",
"Yet if hope has flown away",
"In a night, or in a day,",
"In a vision, or in none,",
"Is it therefore the less gone?",
"All that we see or seem",
"Is but a dream within a dream."]

distDreamPoe = sc.parallelize(dreamPoe)
```

```
distDreamPoe.count()
```

[24]:
```
11
```

[25]:
```
wordsDreamPoe = distDreamPoe.flatMap(lambda line: line.split(" "))

wordsDreamCount = wordsDreamPoe.map(lambda word: (word,1)).reduceByKey(lambda␣
↪a, b: a + b)
```

[26]:
```
wordsRavenCount =  wordsRavenPoe.map(lambda word: (word,1)).reduceByKey(lambda␣
↪a, b: a + b)
```

[27]:
```
wordsRavenCount.sortByKey().take(3)
```

[27]:
```
[('', 5), ("'Tis", 1), ("'tapping", 1)]
```

[28]:
```
wordsRavenCount.join(wordsDreamCount).collect()
```

[28]:
```
[('upon', (1, 1)), ('a', (3, 6)), ('my', (2, 1)), ('this', (1, 1))]
```

[29]:
```
wordsRavenByLength = wordsRavenPoe.map(lambda word: (len(word),word))

wordsDreamByLength = wordsDreamPoe.map(lambda word: (len(word),word))

wordsRavenByLength.cogroup(wordsDreamByLength).sortByKey().take(3)
```

[29]:
```
[(0,
  (<pyspark.resultiterable.ResultIterable at 0x7f6de5bf8610>,
   <pyspark.resultiterable.ResultIterable at 0x7f6de5bf8640>)),
 (1,
  (<pyspark.resultiterable.ResultIterable at 0x7f6de5bf86a0>,
   <pyspark.resultiterable.ResultIterable at 0x7f6de5bf8700>)),
 (2,
  (<pyspark.resultiterable.ResultIterable at 0x7f6de5bf8760>,
   <pyspark.resultiterable.ResultIterable at 0x7f6de5bf87c0>))]
```

4 Akcje

4.1 Wprowadzenie

- `reduce(func)` – agreguje składowe wykorzystując funkcję `func`, która dostaje dwa argumenty i wyznacza dla nich pojedynczą wartość. Funkcja powinna być komutatywna i łączna – tylko wtedy może być poprawnie przetwarzana równolegle
  - `func(a,b) = func(b,a)`
  - `func(a,func(b,c)) = func(func(a,b),c)`
- `collect()` – zwraca tablicę zawierającą wszystkie składowe RDD

- `count()` – daje w wyniku liczbę składowych
- `first()` – daje w wyniku pierwszą ze składowych
- `take(n)` – daje w wyniku pierwsze n składowych
- `takeSample(withReplacement,num,[seed])` – daje w wyniku przykładowe `num` składowych
- `takeOrdered(n,[ordering])` – daje w wyniku pierwszych `n` składowych RDD uwzględniając ich aktualny porządek lub funkcję porządkującą.

4.2 Przykłady

[30]:
```
wordsRavenPoe.takeSample(False,3)
```

[30]:
```
['some', 'while', 'some']
```

[31]:
```
wordsRavenPoe.takeOrdered(3)
```

[31]:
```
['', '', '']
```

[32]:
```
wordsRavenPoe.takeOrdered(10, key=lambda x: len(x))
```

[32]:
```
['', '', '', '', '', 'a', 'I', 'a', 'I', 'a']
```

[33]:
```
wordsDreamCount.takeOrdered(3, key=lambda x: -x[1])
```

[33]:
```
[('a', 6), ('in', 3), ('or', 3)]
```

[34]:
```
wordsRavenPoe.map(lambda word: (word,1)).countByKey()
```

[34]:
```
defaultdict(int,
            {'Once': 1,
             'upon': 1,
             'a': 3,
             'midnight': 1,
             'dreary,': 1,
             'while': 1,
             'I': 3,
             'pondered,': 1,
             'weak': 1,
             'and': 3,
             'weary,': 1,
             '': 5,
             'Over': 1,
             'many': 1,
             'quaint': 1,
             'curious': 1,
             'volume': 1,
             'of': 2,
             'forgotten': 1,
```

```
 'lore': 1,
 'While': 1,
 'nodded,': 1,
 'nearly': 1,
 'napping,': 1,
 'suddenly': 1,
 'there': 1,
 'came': 1,
 'tapping,': 1,
 'As': 1,
 'some': 2,
 'one': 1,
 'gently': 1,
 'rapping,': 1,
 'rapping': 1,
 'at': 2,
 'my': 2,
 'chamber': 2,
 'door.': 1,
 "'Tis": 1,
 "visitor,'": 1,
 'muttered,': 1,
 "'tapping": 1,
 'door': 1,
 'Only': 1,
 'this': 1,
 'nothing': 1,
 "more.'": 1})
```

[35]:
```
wordsRavenPoe.map(lambda word: (word,1)).countByKey()["my"]
```

[35]: 2

[36]:
```
wordsRavenPoe.foreach(lambda x: print(x))
```

5 Redukcja, agregacja

5.1 Python - lokalnie

- Redukcja i agregacja to jedne z najważniejszych operacji podczas przetwarzania danych
- Do wykonywania operacji redukcji i agregacji mamy w Pythonie mnóstwo bibliotek, typów, method

5.2 Spark - zdalnie

- analogiczne mechanizmy dostępne są dla RDD
- różnica polega na tym, że są one wykonywane

  - na poszczególnych węzłach - wyliczając agregacje częściowe, a następnie
  - po przesłaniu cząstkowych wyników do określonych węzłów, są dokonywane agragacje ostateczne
- takie podejście wyklucza zastosowanie niektórych rozwiązań znanych z wariantów lokalnych

5.3 Metody dla RDD

- `aggregate(zeroValue: U, seqOp: Callable[[U, T], U], combOp: Callable[[U, U], U]) → U`
  - wartość zerowa
  - możliwy inny typ wyniku `U` niż typ składowych przetwarzanego RDD `T`
  - przetwarzany równolegle
- `fold(zeroValue: T, op: Callable[[T, T], T]) → T`
  - wartość zerowa
  - ten sam typ wyniku
  - przetwarzany równolegle
- `reduce(f: Callable[[T, T], T]) → T`
  - `f` powinna być komutatywna i łączna – tylko wtedy może być poprawnie przetwarzana równolegle
  - ten sam typ wyniku
  - przetwarzany równolegle

6 Ćwiczenie

[37]:
```
x = sc.range(start=10, end=99, step=2)
```

[ ]:
```
# wylicz sumę liczb
x.
```

[ ]:
```
# wylicz wartość maksymalną
x.
```

[ ]:
```
# wyznacz wartość średnią (wykorzystaj dwa polecenia)
y = x.

y[0]/y[1]
```

6.1 Rozwiązania

[41]:
```
# wylicz sumę liczb
x.reduce(lambda a, b: a + b)
```

[41]: 2430

[42]:
```
# wylicz wartość maksymalną
x.reduce(lambda a, b: a if a>b else b)
```

[42]: 98

[43]:
```
# wyznacz wartość średnią (wykorzystaj dwa polecenia)
y = x.map(lambda l: (l,1)).reduce(lambda a, b: (a[0]+b[0], a[1]+b[1]))

y[0]/y[1]
```

[43]: 54.0

[ ]: