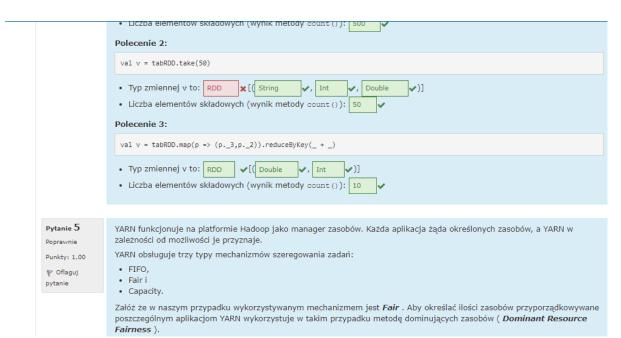
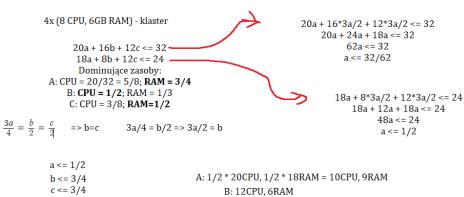
Pytanie 1 Programista aplikacji Spark operuje na obiekcie DataFrame o nazwie gameInfosDF, którego schemat jest następujący: Cześciowo poprawnie |-- _c0: integer (nullable = true) Punkty: 1,00 |-- score_phrase: string (nullable = true) P Oflaguj |-- title: string (nullable = true) |-- url: string (nullable = true) pytanie |-- platform: string (nullable = true) |-- score: double (nullable = true) |-- genre: string (nullable = true) |-- editors_choice: string (nullable = true) |-- release_year: integer (nullable = true) |-- release_month: integer (nullable = true) |-- release_day: integer (nullable = true) Zaimplementował on operację na tym obiekcie z wykorzystaniem polecenia SQL, tworząc w ten sposób wynikowy obiekt result. gameInfosDF.createOrReplaceTempView("gameInfos") val result = spark.sql(""" select platform, count(*) as ile from gameInfos group by platform having count(*) > 100""") Uzupełnij poniższe polecenie wykorzystujące metody obiektu DataFrame tak, aby obiekt result2 utworzony za jego pomocą był pełnym odpowiednikiem obiektu result. import org.apache.spark.sql.functions. x (\$" platform val result2 = gameInfosDF. select ✓(count(\$"platform"). as ✓(count(\$"platform"). as ✓("ile")). √(\$"ile" > 100) where Pytanie 2 Platformy Hive i Pig pozwalają definiować złożone przetwarzanie danych na platformie Hadoop, które wykonywane jest następnie przy użyciu jednego z silników (MapReduce, Tez, Spark). Częściowo poprawnie Hive pozwala definiować to przetwarzanie za pomocą języka SQL, natomiast Pig za pomocą języka Pig Latin. Punkty: 1,00 Polecenia języka Pig Latin bardzo często odpowiadają klazulom w języku SQL. V Usuń flagę Dopasuj do siebie polecenia Pig Latin o odpowiadające im klauzule. ₹ 🗸 COGROUP Brak odpowiednika GROUP BY GROUP FOREACH Brak odpowiednika Y X T X WHERE HAVING 7 🗸 ORDER BY ORDER BY ₹ 🗸 FILTER WHERE





Załóż, że na naszej platformie do dyspozycji YARN są 2 typy zasobów CPU i RAM. Klaster Hadoop składa się z: • 1 - liczba węzłów master (na którym funkcjonuje tylko i wyłącznie resource manager) • 4 - liczba węzłów roboczych, z których każdy dysponuje • 8 CPU o 6 GB RAM Do YARN zgłosiły się 3 aplikacje (A, B, C) żądając następującej wielkości poszczególnych zasobów: • Aplikacja A: 20 CPU, 18 GB RAM · Aplikacja B: 16 CPU, 8 GB RAM • Aplikacja C: 12 CPU, 12 GB RAM Uwzględniając powyższe oblicz przydział zasobów dokonany przez YARN dla poszczególnych aplikacji, a następnie uzupełnij poniższe zdania. • Zasobem dominującym dla aplikacji A jest: oba są tak samo dominujące • Zasobem dominującym dla aplikacji **B** jest: O RAM ○ CPU
 ✓ oba są tak samo dominujące • Zasobem dominującym dla aplikacji C jest: ○ CPU RAM ✔ ○ oba są tak samo dominujące W wyniki alokacji zasobów następujące zasoby zostały $wykorzystane \ w \ 100\%$: CPU ✓ RAM
✓ Zasoby przyznane poszczególnym aplikacjom wyglądają następująco: ✓ CPU, 9
✓ GB RAM Aplikacja A: 10 Aplikacja B: 12 ✓ CPU, 6 ✓ GB RAM Aplikacja C: 9 ✓ CPU, 9 ✓ GB RAM



C: 9CPU, 9RAM

Pytanie 6

Częściowo poprawnie Punkty: 1,00 P Oflaguj pytanie W bazie danych Hive utworzono tabelę TAB za pomocą poniższego polecenia:

```
CREATE TABLE TAB(
B INT,
C INT,
D INT)
PARTITIONED BY (A INT)
CLUSTERED BY(B) INTO 25 BUCKETS
STORED AS ORC;
```

Do tabeli TAB załadowano dane z pliku utworzonego za pomocą poniższego kodu

```
var tab_source = new ListBuffer[(Int,Int,Int,Int)]()
for (a <- (1 to 20); b <- (1 to 100); c <- (1 to 500); d <- (1 to 200))
  tab_source += ((a,b,c,d))
val tab_source_list = tab_source.toList
val tabRDD = sc.parallelize(tab_source_list)

tabRDD.toDF.write.csv("file:///C:\\tab_source.csv")</pre>
```

Zalóż, że kolejność wierszy w tabeli TAB odpowiada implementacji generowania danych w pętli for.

Załóż, że serializowany rozmiar wartości w każdym wierszu każdej kolumny to 10B.

Uzupełnij poniższe zadania:

- Liczba (w milionach) krotek (rekordów) w tabeli TAB wynosi: 200
- Rozmiar (w milionach bajtów) serializowanych źródłowych danych niepodzielonych w żaden sposób (na partycje lub kubełki) wynosi:

Załóż, że tabela TAB w formacie ORC:

Załóż, że tabela TAB w formacie ORC:

- posiada funkcję rozpraszającą wiersze do poszczególnych kubełków jako *klucz mod liczba_kubelkow*
- posiada stopień kompresji (wielkośćDanychPrzedKompresją/wielkośćDanychPoKompresji) serializowanych danych równy 2.
- posiada wielkość paska wynoszącą (w milionach bajtów): 2
- nie jest tabelą wspierającą ACID (nie obsługuje transakcji poleceń DML) nie zawiera dodatkowych katalogów wynikających z tej funkcjonalności

Uzupełnij poniższe zadania:

- Liczba katalogów wewnętrznych (poza głównym katalogiem) dla tabeli TAB wynosi: 20
- Liczba (w milionach) krotek wchodzących w skład pojedynczej partycji i pojedynczego kubełka wynosi: 0,4
- Wielkość (w milionach bajtów) plików wchodzących w skład pojedynczej partycji i pojedynczego kubełka wynosi (uwzględnij kompresję i wielkość serializowanych danych, pomiń wielkość indeksów):
- Liczba plików dla tabeli TAB wynosi: 500

Na tabeli TAB wykonano następujące zapytania:

Zapytanie A:

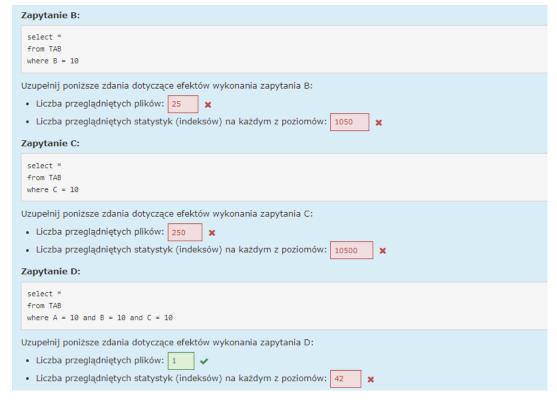
select *
from TAB
where A = 10

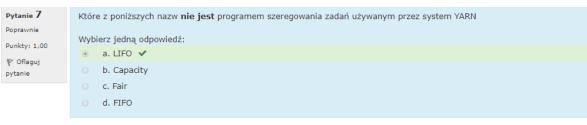
Uzupełnij poniższe zdania dotyczące efektów wykonania zapytania A:

- Liczba przeglądniętych plików: 25

 ✓
- Liczba przeglądniętych statystyk (indeksów) na każdym z poziomów: 0

Zapytanie B:





Pytanie 8
Cześciowo

poprawnie Punkty: 1,00

♥ Oflaguj pytanie Programista Spark korzystając z klastra o następujących parametrach:

- liczba węzłów (wykonawców): 10
- liczba dostępnych rdzeni procesorów na każdym z węzłów: 4

utworzył zmienną RDD o nazwie tabRDD za pomocą poniższego polecenia:

```
var tab_source = new ListBuffer[(Int, Int_Int)]()
for (a <- (1 to 500); b <- (1 to 200);
    tab_source += ((a,b,c))
val tab_source_list = tab_source.toList

val tabRDD = sc.parallelize(tab_source_list)</pre>
ZADANIA
```

 ${\sf Zał\acute{o}\dot{z},\,\dot{z}e\,\,\,''kolejno\acute{s}c''\,\,wierszy\,\,w\,\,utworzonej\,\,w\,\,ten\,\,spos\acute{o}b\,\,zmiennej\,\,jest\,\,nieokre\acute{s}lona}.$

Następnie programista wykonał następującą operację:

```
val aRDD1 = tabRDD.map(p => (p._3,p)).

filter(_.2._3 \% 100 == 0)
```

```
filter(_._2._3 % 100 == 0)
 val aRDD2 = aRDD1.groupByKey().
               mapValues(ps => ps.map(gi => gi. 2)).
                  mapValues(ps => ps.sum/ps.size)
 aRDD2.collect()
Uzupełnij brakujące fragmenty w poniższym tekście.
• Liczba partycji w tabRDD wynosi 40

    Liczba rekordów w pojedynczej partycji tabRDD wynosi 125000

    Liczba partycji w aRDD1 wynosi 40

    Liczba rekordów w pojedynczej partycji aRDD1 wynosi 2500

    Liczba partycji w aRDD2 wynosi 40

    Liczba rekordów w pojedynczej partycji aRDD2 wynosi 1

    Liczba przesyłanych rekordów pomiędzy węzłami klastra: 2500

W związku ze słabą wydajnością powyższego kodu, programista dokonał jego rekonstrukcji i uruchomił poniższą operację:
 val bRDD1 = tabRDD.map(p \Rightarrow (p._3, (p._3, 1))).
                   filter(_._2._1 % 100 == 0)
 \mbox{val bRDD2 = bRDD1.reduceByKey((p1,p2) => (p1.\_1 + p2.\_1, p1.\_2 + p2.\_2)).}
                   mapValues(p => p._1/p._2)
 bRDD2.collect()
Uzupełnij brakujące fragmenty w poniższym tekście.
• Liczba partycji w bRDD2 wynosi 40 🗶

    Liczba rekordów w pojedynczej partycji bRDD2 wynosi 1

    Liczba przesyłanych rekordów pomiędzy węzłami klastra: 10
```

Następnie powyższe programy zostały użyte w poniższym poleceniu uruchamiającym zadanie **MapReduce**:

```
mapred streaming -D mapred.reduce.tasks=1 \
-files Identity.py, IntSum.py, IntSum.py \
-mapper Identity.py \
-combiner IntSum.py \
-reducer IntSum.py \
-input /user/maria_dev/input \
-output /user/maria_dev/output
```

 $W \ katalogu \ / user/maria_dev/input \ znajdowały \ się \ dwa \ identyczne \ pliki, \ których \ zawartość została \ utworzona \ w \ następujący \ sposób:$

```
var tab_source = new ListBuffer[(String, Int)]()
for (a <- (1001 to 1010); b <- (1 to 10))
    tab_source += ((a.toString,b))
val tab_source_list = tab_source.toList
val tabRDD = sc.parallelize(tab_source_list)
tabRDD.toDF.repartition(1).write.csv("plik_wynikowy")</pre>
```

```
tab_source += ((a.toString,b))
 val tab_source_list = tab_source.toList
 val tabRDD = sc.parallelize(tab_source_list)
 tabRDD.toDF.repartition(1).write.csv("plik_wynikowy")
Ostatecznie przed przetwarzaniem format tych plików z csv został zmieniony na tsv.
Zwróć uwagę na następujące fakty:
• podczas uruchamiania nie określono typu klucza, a to oznacza, że przy wszelakim sortowaniu przez mechanizm
   MapReduce wykorzystany zostanie domyślny typ klucza - Text - np. liczby 1, 2, 10 zostaną posortowane: 1, 10, 2
 • Hadoop Streaming dokonuje sortowania tylko raz - po mapowaniu. Agregator łączący (combiner) nie powinien zaburzać
  kolejności. Jeśli tak się stanie, kolejność danych przetwarzanych przez reduktor jest nieokreślona.
 • sortowanie odbywa się tylko i wyłącznie w oparciu o jeden ze składników.
Uzupełnij brakujące wartości odnoszące się do zadania MapReduce uruchomionego za pomocą podanego powyżej polecenia:
• Liczba rekordów utworzonych w wyniku mapowania: 200

    Liczba rekordów przesłanych pomiędzy węzłami: 10

    Liczba rekordów umieszczonych w pliku wynikowym: 10

 • Kolejność wierszy w pliku wynikowym jest:
                                                     □ nieokreślona
   🗹 określona 🗸
 • Jeśli kolejność wierszy jest określona podaj wartość key z pierwszej linii wynikowego pliku (w przeciwnym wypadku podaj
   wartość "x"): 1010
                        ×
 • Jeśli kolejność wierszy jest określona podaj wartość value z pierwszej linii wynikowego pliku (w przeciwnym wypadku
```

podaj wartość "x"): 45