Apache Spark – opis projektu

Ogólny opis projektu

Celem projektu jest praktyczne wykorzystanie Apache Spark do implementacji procesów przetwarzających dane *Big Data*.

W ramach projektu będziemy przetwarzali dwa powiązane ze sobą zbiory danych.

Projekt będzie składał się z **trzech części**, zależnych od wykorzystywanego języka programowania. Każda z tych części będzie dokonywała tego samego przetwarzania, jednak używać będzie do tego celu innego API lub jego wariantu.

Projekt może być zrealizowany w oparciu o ***misję główną*** – zadanie o wyższym poziomie skomplikowania, oceniane na podstawie (1) wykorzystania właściwego API, (2) zgodności z oczekiwanym wynikiem oraz (3) wydajności.
Za misję główną można zdobyć maksymalną liczbę przewidzianych punktów dla każdej z części.

Alternatywnie projekt może być zrealizowany w oparciu o ***misje poboczne*** – dwa zadania o niskim poziomie skomplikowania, oceniane na oceniane na podstawie (1) wykorzystania właściwego API, (2) poprawności implementacji. Za misje poboczne można zdobyć maksymalnie 50% punktów przewidzianych dla każdej z części.

Każda z części dla ***misji głównych*** umieszcza wynik przetwarzania w innym miejscu docelowym i innym formacie.

Wyniki uzyskane dla ***misji pobocznych*** mają być "wyświetlane" w notatniku.

	Scala	**Python**	**Miejsce docelowe dla wyników misji głównych**
Część 1	*Spark Core – RDD*	*Spark Core – RDD*	*Katalog hdfs* `/tmp/ouput1` ***Scala**: pliki z elementami serializowanymi za pomocą mechanizmów Javy (saveAsObjectFile)* ***Python**: pliki w formacie SequenceFile z danymi serializowanymi w formacie Pickle (saveAsPickleFile)*
Część 2	*Spark SQL – DataFrame*	*Spark SQL – DataFrame*	*Tabela Delta Lake* `output2`
Część 3	*Spark SQL - DataSet*	*Spark SQL – Pandas API on Spark*	*plik* `/tmp/output3.json` *w lokalnym systemie plików w formacie json (na poziomie wierszy)*

Implementacja projektu ma mieć postać notatnika *JupyterLab* opartego o interpreter (*kernel*) zależny od wykorzystywanego języka programowania. Podczas ostatecznych testów notatnik musi działać w oparciu o środowisko *Apache Spark* funkcjonujące w trybie YARN na klastrze *Dataproc* uruchomionym za pomocą polecenia zawartego w jednej z późniejszych sekcji tego dokumentu.

- Scala: *Apache Toree*
- Python: *Python* 3

Tworząc implementację projektu należy wykorzystać przygotowany do tego celu szablon notatnika. W szablonie nie wolno zmieniać paragrafów, które zawierają komentarze wyznaczające miejsca implementacji określonych części projektu oraz zawierające dodatkowe instrukcje dotyczące projektu.

W szablonie można edytować paragrafy zawierające fragmenty kodu przeznaczonego do uzupełnienia jeśli wynika to z instrukcji zawartych w notatniku. Można także dodawać nowe paragrafy, zarówno takie, które zawierają kod, jak i paragrafy *markdown*, które ten kod komentują.

Szablon zawiera dodatkowe instrukcje dotyczące projektu, do których należy się dostosować podczas implementacji.

Zestawy danych

Wszystkie zestawy danych pobieramy ze strony
`http://www.cs.put.poznan.pl/kjankiewicz/bigdata/projekty`
niezależnie od ich oryginalnego źródła pochodzenia.

Pobieranie danych źródłowych, rozpakowanie plików i ładowanie ich do zasobnika nie należą do projektu.
Operacje te należy wykonać wcześniej.
W założeniu projektu dane źródłowe mają znajdować się już w zasobniku w dowolnym katalogu w podkatalogach:

- `datasource1` – pierwszy zbiór danych
- `datasource4` – drugi zbiór danych

Opis zestawów danych

Opisy poszczególnych zbiorów dostępne są w oddzielnych dokumentach.

Kryteria oceny

Misja główna

Każda z części jest tak samo punktowana (po 10 punktów)

- Fundamentalnym elementem warunkującym ocenę jest czystość implementacji – trzymanie się wskazanego API. Odstępstwo ma prawo się pojawić tylko i wyłącznie jeśli zostało wymienione na liście wyjątków.
- Zgodność z oczekiwanym wynikiem (o ile oczekiwany wynik nie zostanie uznany za błędny) - 6 pkt.
  W przypadku gdy oczekiwany wynik zostanie uznany za błędny, wówczas ocena może być cząstkowa i wynika z poziomu poprawności rozwiązania w stosunku do treści zadania
- Wydajność rozwiązania – 4 pkt.
  - Najgorsza wydajność – poniżej 10 percentyli – 0 pkt
  - Słaba wydajność – poniżej 20 percentyli – 1 pkt
  - Średnia wydajność – poniżej 30 percentyli – 2 pkt
  - Dobra wydajność – poniżej 50 percentyli – 3 pkt
  - Najlepsza wydajność – 50% percentyli i więcej – 4 pkt.

Ocena wydajności będzie wyznaczana w oparciu o miary w sposób następujący:

```
shuffleReadBytes + shuffleWriteBytes + inputBytes
```

Misje poboczne

Każda z części jest tak samo punktowana (po 5 punktów – 50% maksymalnej liczby punktów)

- Fundamentalnym elementem warunkującym ocenę jest czystość implementacji – trzymanie się wskazanego API. Odstępstwo ma prawo się pojawić tylko i wyłącznie jeśli zostało wymienione na liście wyjątków.
- Ocena każdego z zadań wynika z poziomu poprawności rozwiązania w stosunku do treści zadania
  - Zadanie 1 – max 2 pkt
  - Zadanie 2 – max 3 pkt.

Lista wyjątków

RDD – API – brak

DataFrame API – brak

Pandas API on Spark – brak

Dataset API

- `join` – dopuszczalne jest wykorzystanie transformacji `join` (która tworzy obiekt `DataFrame`). Warunkiem jest natychmiastowa (bezpośrednio po metodzie `join`) konwersja otrzymanego wyniku do typu `Dataset`. Alternatywa to implementacja `join` za pomocą `map` lub `flatMap` – problemem może być jednak w takim przypadku wydajność.

Sposób uruchomienia środowiska dla ostatecznych testów oraz wykonywania recenzji

Klaster Dataproc

```
gcloud dataproc clusters create ${CLUSTER_NAME} \
  --enable-component-gateway --bucket ${BUCKET_NAME} \
  --region ${REGION} --subnet default \
  --master-machine-type n1-standard-4 --master-boot-disk-size 50 \
  --num-workers 2 --worker-machine-type n1-standard-2 --worker-boot-disk-size 50 \
  --image-version 2.1-debian11 --optional-components JUPYTER \
  --project ${PROJECT_ID} --max-age=3h
```