Chmura Obliczeniowa i Technologie Big Data

Moduł „PyPork” realizujący mechanizmy logiki rozmytej dla platformy Apache Pig

Kierownik	dr hab. inż. Dariusz Mrozek, Prof. PŚ
Autorzy	Jan Skowronek Zuzanna Lempa Paweł Kudzia Wojciech Wątroba
Kierunek	Informatyka SSM
Specjalizacja	IDSI
Technologia	Apache Pig

Spis treści

1. Wstęp

Big Data to termin odnoszący się do dużych i różnorodnych zbiorów danych. Przetwarzanie tego typu danych jest zadaniem wyjątkowo trudnym i wymagającym, ale z drugiej strony analiza ogromnych zbiorów okazuje się być wartościowa, ponieważ pozwala zdobyć nowy rodzaj wiedzy. Z tego też powodu na przestrzeni lat powstały narzędzia i rozbudowane platformy programistyczne umożliwiające m.in. efektywne gromadzenie, przechowywanie i przetwarzanie dużych ilości informacji. Warto zaznaczyć, że dane mogą pochodzić z wielu różnych źródeł, co niesie za sobą konieczność użycia specjalistycznego oprogramowania, za pomocą którego można usprawnić proces przetwarzania. Korzyści z używania narzędzi Big Data są niezaprzeczalne, dlatego istotne jest, aby istniała możliwość rozszerzania o nowe funkcjonalności i tak już zaawansowanych platform.

W ramach kursu *„Chmura Obliczeniowa i Technologie Big Data”* należy zrealizować własny moduł rozszerzający dla jednej z przykładowych platform programistycznych stosowanych w dziedzinie Big Data.

2.1. Cel projektu

Celem projektu jest wzbogacenie przykładowej platformy programistycznej o moduł, który umożliwi realizację rozmytego przetwarzania danych. Moduł ten należy opracować dla technologii Apache Pig będącej częścią ekosystemu Hadoop.

2.2. Założenia projektu

Nowy moduł dla Apache Pig powinien dostarczać podstawowe funkcjonalności rozmytego przetwarzania danych, do których należy zaliczyć m.in. rozmyte filtrowanie z wykorzystaniem zmiennych lingwistycznych, grupowanie rozmyte względem zmiennej lingwistycznej, czy złączenia. Bibliotekę należy zaimplementować wykorzystując do tego UDF (*ang. User-defined function*), czyli funkcje rozszerzające dany program lub środowisko.

Ogromną zaletą ekosystemu Hadoop jest to, iż funkcje mogą zostać zaimplementowane w wybranych językach programowania, wspieranych przez aplikację Apache Pig. Jednym z nich jest Python i to właśnie język ten posłużył do wykonania projektu.

2. Stos technologiczny

W niniejszym rozdziale przedstawiony został stos technologiczny, który posłużył do realizacji projektu. Opisane zostały zagadnienia związane z przygotowaniem środowiska testowego, jak i również architektura autorskiej biblioteki *„PyPork”*.

2.1. Przygotowanie środowiska

W celu rozpoczęcia pracy z platformą Apache Pig należy przygotować środowisko, które zazwyczaj przyjmuje postać jakiegoś systemu operacyjnego z rodziny Linux. Z reguły wiele narzędzi programistycznych jest tworzonych właśnie pod ten system. Nie inaczej jest w przypadku technologii Hadoop i Apache Pig, w związku z czym pierwszym krokiem do rozpoczęcia pracy było skonfigurowanie przykładowej maszyny wirtualnej. Autorzy projektu zdecydowali, że użyją jedną z popularnych dystrybucji Linuksa, którą niewątpliwie jest Ubuntu. Instalacja systemu jest prosta i tak właściwie sprowadza się do postępowania według wskazówek wyświetlanych na ekranie. W przypadku, gdy zdecydowano się wybrać wersję desktop Ubuntu, podczas instalacji systemu warto skorzystać z opcji *„minimal”*, dzięki której finalnie zainstalowane zostaną tylko niezbędne pakiety i narzędzia.

Po prawidłowym wdrożeniu systemu na maszynie wirtualnej można przejść do kolejnego kroku, który dotyczy instalacji Hadoop i Apache Pig. Aby zminimalizować mogące się pojawić problemy i przy okazji uprościć ręczną konfigurację, autorzy projektu utworzyli przykładowy skrypt powłoki Bash. Skrypt ten realizuje następujące zadania:

- Instalacja środowiska uruchomieniowego Java,
- Pobranie i rozpakowanie paczek Hadoop i Apache Pig,
- Przeniesienie plików Hadoop i Apache Pig do odpowiednich katalogów,
- Utworzenie zmiennych środowiskowych,
- Sprawdzenie poprawności wykonanej instalacji za pomocą polecenia wypisującego informacje o zainstalowanej wersji danego oprogramowania.

Na listingu 2.1.1. zaprezentowany został skrypt instalacyjny. Należy pamiętać o tym, aby uruchomić ten skrypt z uprawnieniami administratora systemu.

```
#!/bin/bash
sudo apt install openjdk-8-jdk
sudo apt install openjdk-8-jre
java -version

# main directory where Hadoop and Pig will be installed
MAIN_DIRECTORY=/usr/local
cd $MAIN_DIRECTORY

# download hadoop
sudo wget http://apache.claz.org/hadoop/common/hadoop-2.10.1/hadoop-2.10.1.tar.gz
sudo tar xzf hadoop-2.10.1.tar.gz
sudo mkdir hadoop
sudo mv hadoop-2.10.1/* hadoop
sudo rm -r hadoop-2.10.1

# download pig
sudo wget http://apache.claz.org/pig/pig-0.17.0/pig-0.17.0.tar.gz
sudo tar xzf pig-0.17.0.tar.gz
sudo mkdir pig
sudo mv pig-0.17.0/* pig
sudo rm -r pig-0.17.0

# home directories for Hadoop and Pig
MY_HADOOP_HOME=$MAIN_DIRECTORY/hadoop
MY_PIG_HOME=$MAIN_DIRECTORY/pig

# create environmental variables
echo export HADOOP_HOME=$MY_HADOOP_HOME >> ~/.bashrc
echo export HADOOP_INSTALL=$MY_HADOOP_HOME >> ~/.bashrc
echo export HADOOP_MAPRED_HOME=$MY_HADOOP_HOME >> ~/.bashrc
echo export HADOOP_COMMON_HOME=$MY_HADOOP_HOME >> ~/.bashrc
echo export HADOOP_HDFS_HOME=$MY_HADOOP_HOME >> ~/.bashrc
echo export HADOOP_COMMON_LIB_NATIVE_DIR=$MY_HADOOP_HOME/lib/native >> ~/.bashrc
echo export HADOOP_OPTS="-Djava.library.path=$MY_HADOOP_HOME/lib/nativ" >> ~/.bashrc
echo export YARN_HOME=$MY_HADOOP_HOME >> ~/.bashrc

# if you installed java 7 change name here
echo export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64 >> ~/.bashrc

# add Hadoop and Pig to PATH
HADOOP_PATH=$MY_HADOOP_HOME/sbin:$MY_HADOOP_HOME/bin
PIG_PATH=$MY_PIG_HOME/bin

echo export PATH=$PATH:$HADOOP_PATH:$PIG_PATH >> ~/.bashrc

source ~/.bashrc

hadoop version
pig -version

echo "Script completed."
```

Listing 2.1.1. Skrypt instalacyjny

Następnie można zweryfikować poprawności instalacji Apache Pig. Najprostszy sposób polega na uruchomieniu aplikacji w trybie lokalnym korzystając z komendy przedstawionej na listingu 2.1.2.

```
pig -x local
```

Listing 2.1.2. Uruchomienie Apache Pig w trybie lokalnym

Jeśli instalacja odbyła się w sposób prawidłowy, aplikacja powinna uruchomić się w trybie interaktywnym, w którym możliwe jest wpisywanie poleceń języka Pig Latin. Aby zapoznać się z podstawowymi funkcjami można np. wczytać plik CSV i wyświetlić jego zawartość.

2.2. Przykład użycia biblioteki „PyPork”

W celu sprawdzenia poprawności opisywanej biblioteki należy skorzystać z dostarczonego skryptu z katalogu pig_examples. W przykładzie pokazano użycie operacji rozmytych na danych pochodzących z prostego pliku CSV.

Ponadto należy pamiętać również o tym, aby na początku własnych skryptów Pig Latin zarejestrować główny plik biblioteki main_fuzzy.py. Plik ten musi należy zapisać w miejscu, do którego jest łatwy dostęp. Może to być np. główny katalog użytkownika lub jego pulpit.

Rejestracja modułu „PyPork” odbywa się w pliku o rozszerzeniu .pig za pomocą operatora REGISTER w podobny sposób, jak prezentuje to listing 2.2.1. Ponadto należy pamiętać o tym, aby w miejscu, w którym znajduje się skrypt Pig Latin umieszczony został plik konfiguracyjny pattern.json oraz odpowiedni plik CSV z danymi do przetwarzania.

```
REGISTER '/home/tester/Desktop/PyPork/main_fuzzy.py' using jython as fuzzy
```

Listing 2.2.1. Wskazanie lokalizacji skryptu main_fuzzy.py

Uruchomienie skryptu .pig odbywa się z wykorzystaniem komendy z listingu 2.2.2. W oknie konsoli systemowej powinny zostać wypisane zarówno logi generowane przez narzędzie Hadoop oraz wyniki poszczególnych operacji zdefiniowanych w skrypcie języka Pig Latin.

```
pig -x local linguistic.pig
```

Listing 2.2.2. Uruchomienie skryptu linguistic.pig

2.3. Architektura biblioteki „PyPork”

Projekt składa się z kilku plików skryptowych języka Python, w których zdefiniowane zostały funkcje odpowiedzialne za realizację określonych zadań.

W katalogu pig_examples umieszczono przykłady użycia rozszerzenia *„PyPork”*. Można tutaj znaleźć:

- linguistic.pig – zawiera kilka przykładów pokazujących działanie autorskich funkcji,
- user_data.csv – przechowuje analizowane dane,
- pattern.json – plik konfiguracyjny, w którym zdefiniowane są zmienne lingwistyczne.

W głównym katalogu znajduje się skrypt main_fuzzy.py zawierający wszystkie potrzebne funkcje do prawidłowego działania biblioteki. W pliku tym dostępne są m.in. funkcje operujące na wartościach lingwistycznych, definicje funkcji przynależności (np. trójkątna, trapezowa), czy funkcje realizujące działanie rozmytych operatorów AND, OR i NOT. Każda z funkcji została udokumentowana bezpośrednio w kodzie źródłowym (standardowe komentarze dokumentacji), dzięki czemu podczas pracy z kodem możliwe jest uzyskanie informacji pomocniczych bezpośrednio w edytorze kodu. Skrypt ten jest wymagany do pracy z Apache Pig, co zresztą zaprezentowano w poprzednim podrozdziale.

Ponadto główny katalog repozytorium zawiera kilka skryptów, które weryfikują poprawność działania poszczególnych funkcjonalności:

- fequal_test.py – sprawdza działanie funkcji fequal,
- linguistic_test.py – zawiera testy sprawdzające obsługę wartości rozmytych,
- membership_test.py – przechowuje przykładowe testy, które weryfikują poprawność zaimplementowanych funkcji przynależności,
- operators_test.py – sprawdza działanie rozmytych operatorów AND, OR i NOT.

Powyższe skrypty należy uruchomić z poziomu konsoli systemowej w podobnym sposób, jak pokazano na listingu 2.3.1.

```
python operators_test.py
```

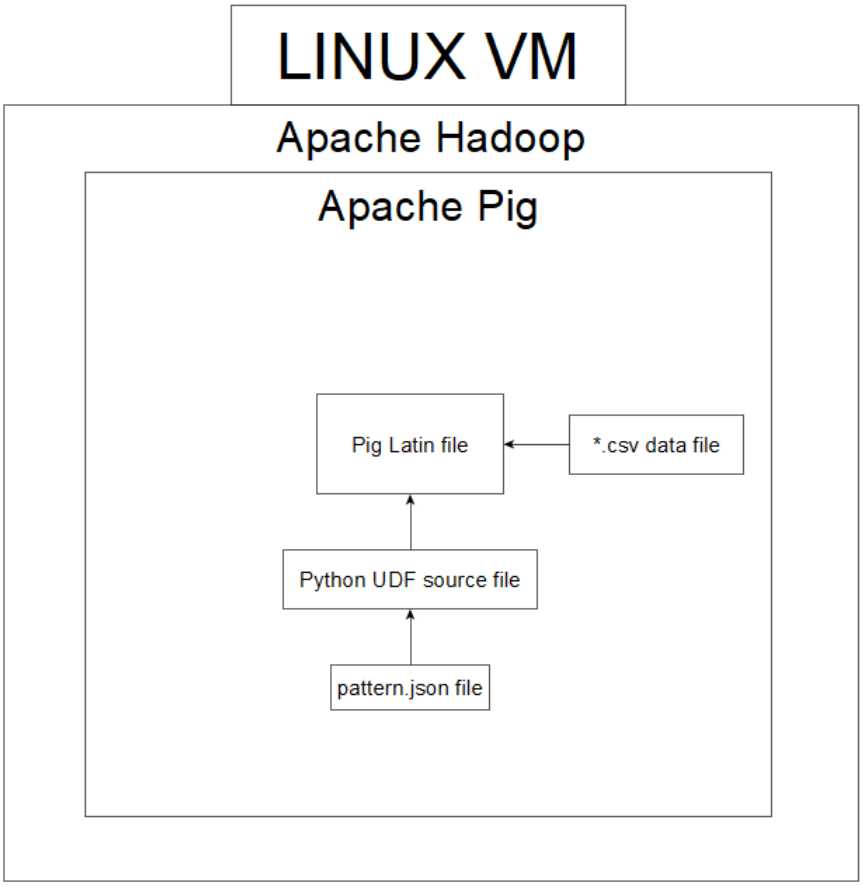
Listing 2.3.1. Uruchamianie skryptu Python z poziomu konsoli systemowej

Wynik działa skryptu operators_test.py zaprezentowany został na listingu 2.3.2.

```
Tests: F_OR, F_AND, F_NOT operators
========== test 1 ==========
values[0]: 1
values[1]: 0.5
result: {'F_OR': 1, 'F_AND': 0.5, 'F_NOT': {'values[0]': 0, 'values[1]': 0.5}}

========== test 2 ==========
values[0]: 0.75
values[1]: 0.1
result: {'F_OR': 0.75, 'F_AND': 0.1, 'F_NOT': {'values[0]': 0.25, 'values[1]': 0.9}}

========== test 3 ==========
values[0]: 0.1
values[1]: 0.4
result: {'F_OR': 0.4, 'F_AND': 0.1, 'F_NOT': {'values[0]': 0.9, 'values[1]': 0.6}}

========== test 4 ==========
values[0]: 0.6
values[1]: 1
result: {'F_OR': 1, 'F_AND': 0.6, 'F_NOT': {'values[0]': 0.4, 'values[1]': 0}}

========== test 5 ==========
values[0]: 0.2
values[1]: 0.3
result: {'F_OR': 0.3, 'F_AND': 0.2, 'F_NOT': {'values[0]': 0.8, 'values[1]': 0.7}}
```

Listing 2.3.2. Wynik działania testów ze skryptu operators_test.py

Na rysunku 2.3.1. zaprezentowane zostało ułożenie modułu w stosunku do całego systemu. Do działania modułu konieczne jest środowisko z systemem operacyjnym Linux (np. Ubuntu) oraz posiadanie aplikacji Hadoop i Apache Pig.



Rysunek 2.3.1. Schemat ułożenia modułu „PyPork”

3. Interfejs Pig Latin

Rozdział zawiera opis interfejsu dostępnego w języku Pig Latin z wykorzystaniem kilku przykładów. Do każdego scenariusza użycia dołączone zostały instrukcje i otrzymane wyniki. Wyniki operacji znajdują się w komentarzach (dwa znaki - -).

3.1. Ładowanie danych z pliku CSV

Listing 3.1.1. prezentuje sposób, w jaki można załadować przykładowe dane z pliku CSV. Za pomocą polecenia DUMP możliwe jest wypisanie wartości zmiennej w standardowym strumieniu wyjścia.

```
--Ładowanie danych z pliku user_data:
users = LOAD 'user_data.csv' USING PigStorage(',') AS (firstname: chararray, lastname:chararr
ay,wzrost:int, wiek:int);
DUMP users;
-- (zosia,samosia,150,12)
-- (ala,makota,160,32)
-- (ALICJA,makota,160,18)
-- (harry,potter,194,89)
-- (ola,makota,175,8)
```

Listing 3.1.1. Ładowanie danych z pliku CSV

3.2. Zmiana wartości na zmienną lingwistyczną

Listing 3.2.1. prezentuje wykorzystanie funkcji one_to_linguistic, dzięki której można przeprowadzić konwersję wartości numerycznej na odpowiadającą jej wartość zmiennej lingwistycznej. Ponadto w przykładzie umieszczono stopień zgodności dla poszczególnych wierszy.

```
--
Zamiana wartości na zmienną lingwistyczną(one_to_lingustic) wraz z wyliczeniem stopnia zgodno
ści(fuzzy_level)
fuzzy_users = FOREACH users GENERATE firstname, lastname, wzrost, fuzzy.one_to_lingustic('wzr
ost', wzrost) as fuzzy_wzrost, fuzzy.fuzzy_level('wzrost',wzrost,fuzzy.one_to_lingustic('wzro
st', wzrost)) as level, wiek, fuzzy.one_to_lingustic('wiek', wiek) as fuzzy_wiek;
DUMP fuzzy_users;
-- (zosia,samosia,150,niski,0.8,12,mlody)
-- (ala,makota,160,niski,0.5,32,stary)
-- (ALICJA,makota,161,niski,0.4,18,mlody)
-- (harry,potter,194,gigant,0.7,89,emeryt)
-- (ola,makota,175,wysoki,0.75,8,dzieciak)
```

Listing 3.2.1. Wykorzystanie funkcji one_to_linguistic

3.3. Grupowanie danych po zmiennej lingwistycznej

Na listingu 3.3.1. przedstawiono przykład obrazujący grupowanie danych po zmiennej lingwistycznej.

```
--Groupowanie danych po zmiennej lingwistycznej:
grouped_fuzzy_users = GROUP users BY fuzzy.one_to_lingustic('wzrost', wzrost) ;
DUMP grouped_fuzzy_users;
--
 (niski,{(ALICJA,makota,160,niski,18,mlody),(ala,makota,160,niski,32,stary),(zosia,samosia,15
0,niski,12,mlody)})
-- (gigant,{(harry,potter,194,gigant,89,emeryt)})
-- (wysoki,{(ola,makota,175,wysoki,8,dzieciak)})
```

Listing 3.3.1. Grupowanie danych po zmiennej lingwistycznej

3.4. Złączenie danych po zmiennej lingwistycznej

Na listingu 3.4.1. zaprezentowane zostało wykonanie operacji złączenia po zmiennej lingwistycznej. Obie instrukcje (jedna napisana w komentarzu) zwracają ten sam wynik do zmiennej self_joined_fuzzy_users.

```
users2 = LOAD 'user_data.csv' USING PigStorage(',') AS (firstname: chararray, lastname:charar
ray,wzrost:int, wiek:int, zajecie:chararray);
fuzzy_us2 = FOREACH users2 GENERATE fuzzy.one_to_lingustic('wzrost', wzrost) as fw, zajecie;

--złączenie dwóch PigStorage po zmiennej lingwistycznej
-- oba join daja ten sam wynik
-- self_joined_fuzzy_users = JOIN fuzzy_users BY fuzzy_wzrost, fuzzy_us2 BY fw;
self_joined_fuzzy_users = JOIN users BY fuzzy.one_to_lingustic('wzrost', wzrost), users2 BY f
uzzy.one_to_lingustic('wzrost', wzrost);

DUMP self_joined_fuzzy_users;
-- (ALICJA,makota,160,niski,18,mlody,niski,prawnik)
-- (ALICJA,makota,160,niski,18,mlody,niski,prawnik)
-- (ALICJA,makota,160,niski,18,mlody,niski,uczen)
-- (ala,makota,160,niski,32,stary,niski,prawnik)
-- (ala,makota,160,niski,32,stary,niski,prawnik)
-- (ala,makota,160,niski,32,stary,niski,uczen)
-- (zosia,samosia,150,niski,12,mlody,niski,prawnik)
-- (zosia,samosia,150,niski,12,mlody,niski,prawnik)
-- (zosia,samosia,150,niski,12,mlody,niski,uczen)
-- (harry,potter,194,gigant,89,emeryt,gigant,nauczyciel)
-- (ola,makota,175,wysoki,8,dzieciak,wysoki,pacjent)
```

Listing 3.4.1. Złączenie danych po zmiennej lingwistycznej

3.5. Zastosowanie funkcji fuzzy_level

Listing 3.5.1. przedstawia użycie funkcji fuzzy_level, która wylicza stopień zgodności dla podanej wartości do zmiennej lingwistycznej wywołując odpowiednią funkcję przynależności.

```
--
Funkcja fuzzy_level wylicza stopień zgodności dla podanej wartości do zmiennej lingwistycznej
 wywołując odpowiednią funkcję przynależności.
filtered = FILTER users BY fuzzy.fuzzy_level('wzrost',wzrost,'wysoki') > 0.5;
DUMP filtered;
-- (ola,makota,175,8)
```

Listing 3.5.1. Zastosowanie funkcji fuzzy_level

3.6. Zastosowanie funkcji F_AND, F_OR, F_NOT

Listingi 3.6.1. prezentuje sposób użycia funkcji realizujących działanie rozmytych operatorów logicznych F_AND, F_OR i F_NOT.

```
-- Przykłady łączenia funkcji za pomocą F_AND, F_OR i F_NOT.

filtered_fand = FILTER users2 BY fuzzy.F_AND(fuzzy.triangle_membership(wzrost, 140.0, 150.0,
160.0), fuzzy.trapezoid_membership(wiek,10.0,12.0,20.0,30.0)) > 0.5;
DUMP filtered_fand;
-- -- (zosia,samosia,150,12,uczen)

filtered_for = FILTER users2 BY fuzzy.F_OR(fuzzy.triangle_membership(wzrost, 140.0, 150.0, 16
0.0), fuzzy.trapezoid_membership(wiek,10.0,12.0,20.0,30.0)) > 0.5;
DUMP filtered_for;
-- (zosia,samosia,150,12,uczen)
-- (ALICJA,makota,160,18,prawnik)

filtered_font = FILTER users2 BY fuzzy.F_NOT(fuzzy.trapezoid_membership(wiek,10.0,12.0,20.0,3
0.0)) > 0.5;
DUMP filtered_font;
-- (ala,makota,160,32,prawnik)
-- (harry,potter,194,89,nauczyciel)
-- (ola,makota,175,8,pacjent)

filtered_fand2 = FILTER users BY fuzzy.F_AND(fuzzy.fuzzy_level('wzrost',wzrost,'wysoki'), fuz
zy.fuzzy_level('wiek',wiek,'dzieciak')) > 0.1;
DUMP filtered_fand2;
-- (ola,makota,175,8)

filtered_for2 = FILTER users BY fuzzy.F_OR(fuzzy.fuzzy_level('wzrost',wzrost,'wysoki'), fuzzy
.fuzzy_level('wiek',wiek,'emeryt')) > 0.5;
DUMP filtered_for2;
-- (harry,potter,194,89)
-- (ola,makota,175,8)
```

Listing 3.6.1. Zastosowanie funkcji F_AND, F_OR, F_NOT

3.7. Zastosowanie funkcji *_membership („około”)

Na listingu 3.7.1. przedstawiono sposób użycia funkcji *_membership, która realizuje założenie funkcji około, gdyż zwraca stopień zgodności dla podanej funkcji przynależności. W bibliotece „PyPork” dostępnych jest kilka implementacji funkcji przynależności.

```
--
  Funkcja fuzzy.*_membership realizuje założenie funkcji około, gdyż zwraca stopień zgodności
 dla podanej funkcji przynależności.
--
 Przykładem takiego filtrowania byłoby wywołanie fuzzy.*_membership(zmienna, parametry) > min
imalne_ktyterium_zgodności).
--
 Dostępne są funkcje trojkatna (triangle_membership), trapezowa (trapezoid_membership), l cla
ss (l_class_membership), y class (y_class_membership) oraz gaussowska (gaussian_membership)
-- Poniżej przykład dla funkcji trójkątnej:
tmp = FILTER users2 BY fuzzy.triangle_membership(wzrost, 140.0, 150.0, 160.0) >0.5;
DUMP tmp;
-- (zosia,samosia,150,12,uczen)
```

Listing 3.7.1. Zastosowanie funkcji około

3.8. Zastosowanie funkcji fequal

Na listingu 3.8.1. zaprezentowano działanie funkcji fequal, która zwraca stopień zgodności obliczając punkty przecięcia rozmytych wartości. Ograniczenia spowodowane funkcjami UDF sprawiają, że poprawne użycie funkcji fequal do realizacji docelowej funkcji fjoin wydaje się niemożliwe.

```
-- Funkcja fequal zwraca stopien zgodności obliczając punkty przecięcia rozmytych wartości.
--
 Ze względu na ograniczenia Pig Latin dotyczące wywoływania udf(user defined functions) popra
wne jej wykorzystanie do implementacji funkcji fjoin wydaje się niemożliwe.
tmp = FOREACH users2 GENERATE firstname, fuzzy.fequal(wzrost, 10, wzrost+5, 5);
DUMP tmp;
-- (zosia,0.6666666666666667)
-- (ala,0.6666666666666667)
-- (ALICJA,0.6666666666666667)
-- (harry,0.6666666666666667)
-- (ola,0.6666666666666667)
```

Listing 3.8.1. Zastosowanie funkcji fequal

4. Interfejs Python

Niniejszy rozdział dotyczy dokumentacji technicznej interfejsu dostępnego w języku Python. Opisane zostały funkcje z pliku main_fuzzy.py.

4.1. Rozmyte operatory logiczne

Dostępne są trzy rozmyte operatory logiczne.

4.1.1. Funkcja F_OR

Funkcja realizuje działanie rozmytego operatora OR.

Wejście: Wartości numeryczne, które mają zostać przetestowane. Każda wartość musi być z zakresu od 0 do 1 (włącznie).

Wyjście: Maksymalna wartość z podanych na wejściu wartości.

Przykład użycia w Python: `F_OR(1, 0.5)` lub `F_OR(*[1, 0.5])`.

```
def F_OR(*values):
    return max(values)
```

Listing 4.1.1.1. Funkcja F_OR

4.1.2. Funkcja F_AND

Funkcja realizuje działanie rozmytego operatora AND.

Wejście: Wartości numeryczne, które mają zostać przetestowane. Każda wartość musi być z zakresu od 0 do 1 (włącznie).

Wyjście: Minimalna wartość z podanych na wejściu wartości.

Przykład użycia w Python: `F_AND(1, 0.5)` lub `F_AND(*[1, 0.5])`.

```
def F_AND(*values):
    return min(values)
```

Listing 4.1.2.1. Funkcja F_AND

4.1.3. Funkcja F_NOT

Funkcja realizuje działanie rozmytego operatora NOT.

Wejście: Wartość numeryczna, która ma zostać przetestowana. Wartość musi być z zakresu od 0 do 1 (włącznie).

Wyjście: Wynik wykonania operacji rozmytego operatora NOT.

Przykład użycia w Python: `F_NOT(1)`.

```
def F_NOT(value):
    return 1 - value
```

Listing 4.1.3.1. Funkcja F_NOT

4.2. Funkcje przynależności

Dostępnych jest pięć funkcji przynależności.

4.2.1. Funkcja triangle_membership

Funkcja realizuje działanie trójkątnej funkcji przynależności.

Wejście: Wartość numeryczna x oraz parametry funkcji: a, b, c.

Wyjście: Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: `triangle_membership(4, 5, 10, 12)`.

```python
def triangle_membership(x, a, b, c):
    if x <= a:
        return 0
    elif x <= b:
        return (x - a) / (b - a)
    elif x <= c:
        return (c - x) / (c - b)
    else:
        return 0
```

Listing 4.2.1.1. Funkcja triangle_membership

4.2.2. Funkcja trapezoid_membership

Funkcja realizuje działanie trapezowej funkcji przynależności.

Wejście: Wartość numeryczna x oraz parametry funkcji: a, b, c, d.

Wyjście: Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: `trapezoid_membership(0.5, 1, 11, 14, 15)`.

```python
def trapezoid_membership(x, a, b, c, d):
    if x <= a:
        return 0
    elif x <= b:
        return (x - a) / (b - a)
    elif x <= c:
        return 1
    elif x <= d:
        return (d - x) / (d - c)
    else:
        return 0
```

Listing 4.2.2.1. Funkcja trapezoid_membership

4.2.3. Funkcja l_class_membership

Funkcja realizuje działanie funkcji przynależności klasy L.

Wejście: Wartość numeryczna x oraz parametry funkcji: a, b.

Wyjście: Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: `l_class_membership(5, 24, 30)`.

```
def l_class_membership(x, a, b):
    if x <= a:
        return 1
    elif x <= b:
        return (b - x) / (b - a)
    else:
        return 0
```

Listing 4.2.3.1. Funkcja l_class_membership

4.2.4. Funkcja y_class_membership

Funkcja realizuje działanie funkcji przynależności klasy Y.

Wejście: Wartość numeryczna x oraz parametry funkcji: a, b.

Wyjście: Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: `y_class_membership(5, 24, 30)`.

```
def y_class_membership(x, a, b):
    if x <= a:
        return 0
    elif x <= b:
        return (x - a) / (b - a)
    else:
        return 1
```

Listing 4.2.4.1. Funkcja y_class_membership

4.2.5. Funkcja gaussian_membership

Funkcja realizuje działanie gaussowskiej funkcji przynależności.

Wejście: Wartość numeryczna x oraz parametry funkcji: mean, sd.

Wyjście: Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: `gaussian_membership(1, 2, 5)`.

```
def gaussian_membership(x, mean, sd):
    return exp(((x - mean) ** 2) / 2 * (sd ** 2))
```

Listing 4.2.5.1. Funkcja gaussian_membership

4.3. Funkcje pomocnicze

Dostępnych jest kilka funkcji pomocniczych, które używane są przez inne funkcje.

4.3.1. Funkcja _callProperFun

Funkcja odpowiedzialna za wywołanie wskazanej funkcji przynależności. Jest to funkcja pomocnicza, która nie powinna być bezpośrednio używana.

Wejście: Wartość obiektu JSON z pliku konfiguracyjnego JSON. Obiekt JSON musi zawierać klucz „function” oraz inne odpowiednie klucze, które opisują parametry dla wskazanej funkcji przynależności. Przykładowo dla trójkątnej funkcji przynależności muszą zostać zdefiniowane klucze: „a”, „b” i „c”.

Wyjście: Wartość zwrócona przez wskazaną funkcję przynależności. Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: `_callProperFun(v, value)`.

```
def _callProperFun(v, value):
    if v['function'] == 'triangle':
        return triangle_membership(value, float(v['a']), float(v['b']), float(v['c']))
    elif v['function'] == 'trapezoid':
        return trapezoid_membership(value, float(v['a']), float(v['b']), float(v['c']), float
(v['d']))
    elif v['function'] == 'l_class':
        return l_class_membership(value, float(v['a']), float(v['b']))
    elif v['function'] == 'y_class':
        return y_class_membership(value, float(v['a']), float(v['b']))
    elif v['function'] == 'gaussian':
        return gaussian_membership(value, float(v['a']), float(v['b']))
```

Listing 4.3.1.1. Funkcja _callProperFun

4.3.2. Funkcja _line_intersection

Funkcja odpowiedzialna za sprawdzenie, czy podane na wejściu linie przecinają się. Jest to funkcja pomocnicza, która nie powinna być bezpośrednio używana.

Wejście: Pierwsza linia, druga linia.

Wyjście: Wartość określający, czy linie przecinają się.

Przykład użycia w Python: `_line_intersection(line1_1, line2_1)`.

```
def _line_intersection(L1, L2):
    Ax1 = L1[0][0]
    Ay1 = L1[0][1]
    Ax2 = L1[1][0]
    Ay2 = L1[1][1]
    Bx1 = L2[0][0]
    By1 = L2[0][1]
    Bx2 = L2[1][0]
    By2 = L2[1][1]
    d = (By2 - By1) * (Ax2 - Ax1) - (Bx2 - Bx1) * (Ay2 - Ay1)
    if d:
        uA = ((Bx2 - Bx1) * (Ay1 - By1) - (By2 - By1) * (Ax1 - Bx1)) / d
        uB = ((Ax2 - Ax1) * (Ay1 - By1) - (Ay2 - Ay1) * (Ax1 - Bx1)) / d
    else:
        return 0
    if not(0 <= uA <= 1 and 0 <= uB <= 1):
        return 0
    x = Ax1 + uA * (Ax2 - Ax1)
    y = Ay1 + uA * (Ay2 - Ay1)
    return y
```

Listing 4.3.2.1. Funkcja _line_intersection

4.4. Funkcje operacji rozmytych

Dostępnych jest kilka funkcji dotyczących operacji rozmytych.

4.4.1. Funkcja one_to_lingustic

Funkcja odpowiedzialna za konwersję wartości numerycznej do wartości lingwistycznej.

Wejście: Zmienna lingwistyczna, wartość numeryczna.

Wyjście: Wartość lingwistyczna odpowiadającą wartości numerycznej.

Przykład użycia w Python: `one_to_lingustic('wzrost', 170)`.

```
def one_to_lingustic(pattern_name, value):
    value_patter = pattern[pattern_name]
    max_val = 0
    max_key = ''
    for k, v in value_patter.items():
        res = _callProperFun(v, value)
        if res > max_val:
            max_val = res
            max_key = k
    return max_key
```

Listing 4.4.1.1. Funkcja one_to_lingustic

4.4.2. Funkcja list_to_lingustic

Funkcja odpowiedzialna za konwersję wartości numerycznych do wartości lingwistycznych.

Wejście: Zmienna lingwistyczna, lista wartości numerycznych.

Wyjście: Lista wartości lingwistycznych odpowiadających wartościom numerycznym.

Przykład użycia w Python: `list_to_lingustic('wzrost', [110, 170])`.

```
def list_to_lingustic(pattern_name, values):
    results = []
    for value in values:
        res = one_to_lingustic(pattern_name, value)
        results.append(res)
    return results
```

Listing 4.4.2.1. Funkcja list_to_lingustic

4.4.3. Funkcja fuzzy_level

Funkcja odpowiedzialna za wykonywanie operacji sprawdzającej stopień przynależności.

Wejście: Zmienna lingwistyczna, wartość numeryczna, wartość zmiennej lingwistycznej.

Wyjście: Wartość zwrócona przez wskazaną funkcję przynależności. Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: `fuzzy_level('wzrost', 170, 'wysoki')`.

```
def fuzzy_level(pattern_name, value, key):
    v = pattern[pattern_name][key]
    return _callProperFun(v, value)
```

Listing 4.4.3.1. Funkcja fuzzy_level

4.4.4. Funkcja fequal

Funkcja odpowiedzialna za wykonanie operacji rozmycia wartości. Zwraca stopień zgodności obliczając punkty przecięcia rozmytych wartości.

Wejście: Wartości numeryczne, poziom rozmycia dla wartości numerycznych.

Wyjście: Stopień zgodności dwóch zmiennych rozmytych.

Przykład użycia w Python: `fequal(3, 2, 10, 2)`.

```
def fequal(value1, fuzzy_level1, value2, fuzzy_level2):
    value1 = float(value1)
    value2 = float(value2)
    fuzzy_level1 = float(fuzzy_level1)
    fuzzy_level2 = float(fuzzy_level2)
    line1_1 = [[value1 - fuzzy_level1, 0.0], [value1, 1.0]]
    line1_2 = [[value1, 1.0], [value1 + fuzzy_level1, 0.0]]
    line2_1 = [[value2 - fuzzy_level2, 0.0], [value2, 1.0]]
    line2_2 = [[value2, 1.0], [value1 + fuzzy_level2, 0.0]]
    max_level = 0
    levels = [
        _line_intersection(line1_1, line2_1),
        _line_intersection(line1_1, line2_2),
        _line_intersection(line1_2, line2_1),
        _line_intersection(line1_2, line2_2)
    ]
    for lvl in levels:
        if lvl > max_level and lvl <= 1:
            max_level = lvl
    return max_level
```

Listing 4.4.4.1. Funkcja fequal

4.4.5. Funkcja add_new_pattern

Funkcja odpowiedzialna za dodawanie nowej zmiennej lingwistycznej. Zmienna zostaje dodawana do obiektu w aplikacji (przechowywanym w pamięci RAM) i nie jest dopisywana do pliku konfiguracyjnego JSON, dzięki czemu możliwe jest dynamiczne zdefiniowanie zmiennych lingwistycznych w trakcie wykonywania skryptu.

Wejście: Zmienna lingwistyczna, słownik przechowujący definicję zmiennej lingwistycznej.

Wyjście: Funkcja nic nie zwraca.

Przykład użycia w Python: `add_new_pattern('wzrost', {'niski': {'function': 'triangle', 'a': 130, 'b': 155, 'c': 165},'wysoki': {'function': 'triangle', 'a': 160, 'b': 180, 'c': 190}})`.

```
def add_new_pattern(pattern_name, new_pattern):
    pattern[pattern_name] = new_pattern
```

Listing 4.4.5.1. Funkcja add_new_pattern

4.4.6. Funkcja get_patterns_names

Funkcja odpowiedzialna za zwrócenie wszystkich zmiennych lingwistycznych z pliku konfiguracyjnego JSON.

Wejście: Funkcja nie przyjmuje żadnych argumentów.

Wyjście: Lista zmiennych lingwistycznych.

Przykład użycia w Python: `get_patterns_names()`.

```
def get_patterns_names():
    patterns_names = []
    for k, v in pattern.items():
        patterns_names.append(k)
    return patterns_names
```

Listing 4.4.6.1. Funkcja get_patterns_names

4.5. Plik konfiguracyjny pattern.json

W pliku tym możliwe jest zdefiniowanie zmiennych lingwistycznych. Dla każdej zmiennej lingwistycznej należy utworzyć obiekt, którego klucze przechowują informacje o możliwych wartościach przyjmowanych przez daną zmienną. Ponadto każda z wartości musi składać się z klucza „function" definiującego używaną funkcję przynależności (np. triangle) oraz kluczy opisujących używane przez tę funkcję parametry (np. a, b, c).

```
{
    "wzrost": {
        "bardzoniski": {
            "function": "triangle",
            "a": 80,
            "b": 110,
            "c": 135
        },
        "niski": {
            "function": "triangle",
            "a": 130,
            "b": 155,
            "c": 165
        },
        // pozostale wartosci
    },
    "wiek": {
        "dzieciak": {
            "function": "triangle",
            "a": 1,
            "b": 5,
            "c": 10
        },
        "mlody": {
            "function": "triangle",
            "a": 10,
            "b": 15,
            "c": 30
        },
        // pozostale wartosci
    }
}
```

Listing 4.4.7.1. Plik konfiguracyjny pattern.json

5. Podsumowanie

Wykonanie projektu pt. *„PyPork”* pozwoliło autorom zapoznać się z platformą Apache Pig, która znajduje zastosowanie w analizie i przetwarzaniu dużych zbiorów danych. Platforma ta udostępnia proceduralny język Pig Latin, dzięki któremu możliwe jest pisanie zarówno bardzo prostych, jak i bardziej zaawansowanych skryptów umożliwiających przetwarzanie danych. Pig Latin oprócz udostępniania typowych mechanizmów charakterystycznych dla języków programowania tj. różne typy danych (tuple, bag, map), czy funkcje (np. LOAD, STORE), daje również możliwość definiowania własnych funkcji, co pozwala wprowadzić dodatkowe operacje w potoku przetwarzania. Opcja rozbudowy istniejących rozwiązań niewątpliwie jest jedną z kluczowych zalet platform Hadoop i Apache Pig, ponieważ dzięki temu standardowy zestaw narzędzi (który swoją drogą i tak jest obszerny) może być rozbudowany. W kontekście projektu *„PyPork”* Apache Pig wzbogacono o wybrane elementy logiki rozmytej, która jak się okazuje może być niezwykle przydatna w ciekawej dziedzinie informatyki jaką jest Big Data.

Zaletą technologii Apache Pig jest przede wszystkim to, iż do definiowanie UDF możliwe jest z wykorzystaniem kilku języków programowania. Implementacja biblioteki w języku Python okazała się bardzo wygodna, aczkolwiek ma swoje ograniczenia, ponieważ język ten nie jest głównym językiem wspieranym przez Apache Pig, o czym zresztą można było się przekonać podczas próby implementacji operacji fjoin. Okazuje się, że z poziomu języka Pig Latin niemożliwe jest przekazanie dwóch kolekcji, w związku z czym potrzebne jest obejście tego ograniczenia np. poprzez wywołanie skryptu .pig z poziomu języka Python.

Autorzy modułu są zadowoleni z uzyskanych wyników i osiągniętego celu, choć nie da się ukryć, że aktualne rozwiązanie mogłoby zostać bardziej usprawnione. Podczas pracy nad finalną implementacji pojawiły się pewne trudności, co zasygnalizowano na kilku spotkaniach konsultacyjnych z prowadzącym. Niemniej jednak zarówno po wskazówkach, jak i kilku testach zrealizowanych podczas oddawania projektu udało się dojść do pozytywnych wniosków. Przy okazji jeśli dodatkowym celem projektu byłoby udostępnienie biblioteki *„PyPork”* szerzej publiczności, wtedy koniecznym krokiem jest usprawnienie publicznego repozytorium w serwisie GitHub. Ponadto konieczna byłaby rozbudowana istniejącej dokumentacji biblioteki. Tego typu zmiana z pewnością podniosłaby ogólną jakość projektu.

Podsumowując, realizacja projektu *„PyPork”* poruszającego dosyć niestandardową tematykę w ramach kursu *„Chmura Obliczeniowa i Technologie Big Data”*, zdecydowanie poszerzyła wiedzę dotyczącą technik pracy z językiem Python, zbiorów rozmytych i co najważniejsze specjalistycznych narzędzi służących do analizy, gromadzenia i przetwarzania dużych zbiorów danych. Nie da się ukryć, że wiedza o istnieniu narzędzi tj. Hadoop, czy Apache Pig jest niezwykle przydatna podczas projektowania komercyjnych produktów informatycznych, które główny nacisk kładą m.in. na sprawne przekazywanie i analizowanie dużych zbiorów informacji.