Implementacja operacji relacyjnych w Apache Spark z wykorzystaniem MapReduce

Streszczenie

W niniejszym dokumencie przedstawiono implementację podstawowych operacji relacyjnych w środowisku Apache Spark z wykorzystaniem modelu MapReduce. Analizie poddano zbiory danych z plików `relation.txt` i `join.txt`. Zaprezentowano procesy przetwarzania i wyniki operacji takich jak suma zbiorów, część wspólna oraz złączenia relacji, wzbogacone o odpowiednie fragmenty kodu w języku Python.

Spis treści

1 Wstęp

Przetwarzanie dużych zbiorów danych wymaga wydajnych i skalowalnych narzędzi. Apache Spark, oparty na modelu MapReduce, umożliwia efektywne wykonywanie operacji na danych rozproszonych. W niniejszym dokumencie skupimy się na implementacji podstawowych operacji relacyjnych z użyciem Spark, ilustrując procesy na przykładach.

2 Operacje na zbiorach z pliku `relation.txt`

Plik `relation.txt` zawiera pary składające się z nazwy relacji (A lub B) oraz wartości liczbowej. Celem jest wykonanie podstawowych operacji na zbiorach:

- Suma zbiorów $A \cup B$
- Część wspólna zbiorów $A \cap B$
- Różnica zbiorów $A - B$ oraz $B - A$

2.1 Tworzenie zbiorów A i B

Z pliku `relation.txt` wyodrębniamy wartości przypisane do relacji A i B:

$$A = \{0, 1, 4, 8, 9, 10, 17, \ldots, 98\}$$
$$B = \{0, 2, 3, 5, 8, 9, 10, \ldots, 99\}$$

2.2 Implementacja w Apache Spark

Poniżej przedstawiono implementację powyższych operacji z wykorzystaniem języka Python i biblioteki PySpark.

```python
from pyspark import SparkContext

sc = SparkContext.getOrCreate()
data = sc.textFile('relation.txt')

# Mapowanie każdej linii na parę (RelationName, Value)
lines = data.map(lambda line: line.strip().split())

# Utworzenie RDD dla relacji A i B
rddA = lines.filter(lambda x: x[0] == 'A').map(lambda x: int(x[1]))
rddB = lines.filter(lambda x: x[0] == 'B').map(lambda x: int(x[1]))

# Mapowanie elementów w A i B do par (Value, RelationName)
mapA = rddA.map(lambda x: (x, 'A'))
mapB = rddB.map(lambda x: (x, 'B'))

# Połączenie zmapowanych RDD
unionMap = mapA.union(mapB)

# Grupowanie po kluczu (Value)
grouped = unionMap.groupByKey().mapValues(list)

# Operacje na zbiorach
unionValues = grouped.map(lambda x: x[0])  # Suma zbiorów A i B
intersectionValues = grouped.filter(lambda x: 'A' in x[1] and 'B' in
    x[1]).map(lambda x: x[0])  # A     B
differenceAB = grouped.filter(lambda x: 'A' in x[1] and 'B' not in
    x[1]).map(lambda x: x[0])  # A - B
differenceBA = grouped.filter(lambda x: 'B' in x[1] and 'A' not in
    x[1]).map(lambda x: x[0])  # B - A

# Zbieranie i wyświetlanie wyników
print("Suma zbiorów A i B:")
print(unionValues.collect())

print("\nCzęść wspólna A i B:")
print(intersectionValues.collect())

print("\nRóżnica A - B:")
print(differenceAB.collect())

print("\nRóżnica B - A:")
print(differenceBA.collect())
```

Listing 1: Przetwarzanie pliku `relation.txt`

2.3 Wyniki operacji

Na podstawie powyższego kodu otrzymujemy następujące wyniki:

- **Suma zbiorów** $A \cup B$:
$$A \cup B = \{0, 1, 2, 3, 4, 5, \ldots, 99\}$$
- **Część wspólna zbiorów** $A \cap B$:
$$A \cap B = \{0, 8, 9, 10, 17, \ldots, 98\}$$
- **Różnica zbiorów** $A - B$:
$$A - B = \{1, 4, 18, \ldots, 96\}$$
- **Różnica zbiorów** $B - A$:
$$B - A = \{2, 3, 5, \ldots, 99\}$$

3 Złączenie relacji z pliku `join.txt`

3.1 Opis problemu

Celem jest wykonanie złączenia (*join*) między relacjami `Customers` i `Orders` na podstawie wspólnego atrybutu `CustomerID`, wykorzystując model MapReduce.

3.2 Dane wejściowe

Klienci (`Customers`):

CustomerID	Informacje o kliencie
1	**CompanyName**: Alfreds Futterkiste, **ContactName**: Maria Anders, **Country**: Germany
2	**CompanyName**: Ana Trujillo Emparedados y helados, **ContactName**: Ana Trujillo, **Country**: Mexico
3	**CompanyName**: Antonio Moreno Taquería, **ContactName**: Antonio Moreno, **Country**: Mexico
4	**CompanyName**: NO ORDERS, **ContactName**: NO ORDERS, **Country**: NO ORDERS

Zamówienia (`Orders`):

OrderID, CustomerID	OrderDate
10308, 1	'2016-09-18'
10309, 2	'2016-09-19'
10310, 3	'2016-09-20'
10311, 3	'2016-09-20'
10312, 3	'2016-09-20'
10313, 5	'2016-09-20'

3.3 Implementacja w Apache Spark

```
# Wczytanie danych z pliku join.txt
order_data = sc.textFile('join.txt')
# Podzielenie każdej linii na pola
order_lines = order_data.map(lambda line: line.strip().split('\t'))
# Funkcja przetwarzająca każdą linię i mapująca ją na parę (CustomerID, Record)
def process_order_line(line):
    if line[0] == 'Orders':
        # Mapowanie Orders na (CustomerID, ('Orders', OrderID, OrderDate))
        return (line[2], ('Orders', line[1], line[3]))
    elif line[0] == 'Customers':
        # Mapowanie Customers na (CustomerID, ('Customers', CompanyName,
            ContactName, Country))
        return (line[1], ('Customers', line[2], line[3], line[4]))
    else:
        return None

# Mapowanie linii i filtrowanie wartości None
order_mapped = order_lines.map(process_order_line).filter(lambda x: x is not None)
# Grupowanie po CustomerID
order_grouped = order_mapped.groupByKey().mapValues(list)
# Funkcja łącząca informacje o kliencie z zamówieniami
def join_orders(customers_orders):
    customer_id, records = customers_orders
    customer_info = None
    orders = []
    for record in records:
        if record[0] == 'Customers':
            customer_info = record  # ('Customers', CompanyName, ContactName,
                Country)
        elif record[0] == 'Orders':
            orders.append(record)  # ('Orders', OrderID, OrderDate)
    # Jeśli są dostępne informacje o kliencie, połącz z każdym zamówieniem
    if customer_info:
        return [
            (customer_id, customer_info[1:], order[1:])
            for order in orders
        ]
    else:
        # Brak informacji o kliencie, nie można wykonać złączenia
        return []

# Wykonanie operacji złączenia
joined_orders = order_grouped.flatMap(join_orders)
# Zbieranie i wyświetlanie wyników
print("\nPołączone zamówienia i klienci:")
for record in joined_orders.collect():
    customer_id = record[0]
    company_name, contact_name, country = record[1]
    order_id, order_date = record[2]
    print(f"CustomerID: {customer_id}, CompanyName: {company_name}, "
          f"ContactName: {contact_name}, Country: {country}, "
          f"OrderID: {order_id}, OrderDate: {order_date}")
```

Listing 2: Złączenie relacji z pliku `join.txt`

3.4 Wyniki złączenia

W wyniku wykonania powyższego kodu otrzymujemy następujące połączone rekordy:

- **CustomerID**: 1
  - **CompanyName**: Alfreds Futterkiste
  - **ContactName**: Maria Anders
  - **Country**: Germany
  - **OrderID**: 10308
  - **OrderDate**: 2016-09-18
- **CustomerID**: 2
  - **CompanyName**: Ana Trujillo Emparedados y helados
  - **ContactName**: Ana Trujillo
  - **Country**: Mexico
  - **OrderID**: 10309
  - **OrderDate**: 2016-09-19
- **CustomerID**: 3 (trzykrotnie dla różnych zamówień)
  - **CompanyName**: Antonio Moreno Taquería
  - **ContactName**: Antonio Moreno
  - **Country**: Mexico
  - **OrderID**: 10310, 10311, 10312
  - **OrderDate**: 2016-09-20

3.5 Analiza wyników

- Klient o `CustomerID` = 4 nie posiada zamówień, w związku z czym nie został uwzględniony w wynikach złączenia.
- Zamówienie o `OrderID` = 10313 z `CustomerID` = 5 nie zostało uwzględnione, ponieważ brak jest informacji o kliencie o `CustomerID` = 5 w relacji `Customers`.
- Operacja złączenia działa jako *inner join*, łącząc tylko te rekordy, dla których klucz `CustomerID` występuje w obu relacjach.

4 Wnioski

Zaprezentowane implementacje podstawowych operacji relacyjnych w środowisku Apache Spark pokazują efektywność modelu MapReduce w przetwarzaniu dużych zbiorów danych. Wykorzystanie RDD (Resilient Distributed Datasets) umożliwia łatwe i skalowalne operacje na danych, co jest kluczowe w analizie big data.

Literatura

[1] Apache Spark – *https://spark.apache.org/*

[2] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, Communications of the ACM, 51(1):107-113, 2008.