Rozdział 1. Hadoop Distributed File System (HDFS)

Rozproszony system plików Hadoop (HDFS) to rozproszony, skalowalny i przenośny system plików oparty na języku Java, przeznaczony do obsługi dużych klastrów serwerów standardowych. Projekt HDFS opiera się na GFS, systemie plików Google, który jest opisany w artykule (mogę udostępnić zainteresowanym) opublikowanym przez Google. Podobnie jak wiele innych rozproszonych systemów plików, HDFS przechowuje duże ilości danych i zapewnia przejrzysty dostęp do wielu klientów rozproszonych w sieci. Tam, gdzie HDFS przoduje, jest zdolność do przechowywania bardzo dużych plików w niezawodny i skalowalny sposób.

HDFS jest przeznaczony do przechowywania dużej ilości informacji, zazwyczaj petabajtów (dla bardzo dużych plików), gigabajtów i terabajtów. Osiąga się to za pomocą systemu plików o strukturze blokowej. Poszczególne pliki są dzielone na bloki o stałym rozmiarze, które są przechowywane na komputerach w klastrze. Pliki złożone z kilku bloków zazwyczaj nie mają wszystkich swoich bloków przechowywanych na jednym komputerze.

HDFS zapewnia niezawodność poprzez replikację bloków i dystrybucję replik w klastrze. Domyślny współczynnik replikacji wynosi trzy, co oznacza, że każdy blok istnieje w klastrze trzy razy. Replikacja na poziomie bloków zapewnia dostępność danych nawet w przypadku awarii maszyn.

Ten rozdział rozpoczyna się od wprowadzenia podstawowych koncepcji HDFS i wyjaśnia, jak współdziałać z systemem plików za pomocą natywnych wbudowanych poleceń. Po kilku przykładach wprowadzono bibliotekę klienta Python, która umożliwia programowy dostęp do HDFS z poziomu aplikacji Python.

# Rozdział 2. MapReduce z Pythonem

MapReduce to model programowania, który umożliwia przetwarzanie i generowanie dużych ilości danych poprzez podział pracy na niezależne zadania i wykonywanie zadań równolegle w klastrze maszyn. Styl programowania MapReduce został zainspirowany konstrukcjami programowania funkcjonalnego map i reduce, które są powszechnie używane do przetwarzania list danych. Na wysokim poziomie, każdy program MapReduce przekształca listę elementów danych wejściowych w listę elementów danych wyjściowych dwukrotnie, raz w fazie mapy i drugi raz w fazie redukcji.

Ten Rozdział rozpoczyna się od wprowadzenia modelu programowania MapReduce i opisania sposobu, w jaki dane przepływają przez różne fazy modelu. Przykłady pokazują następnie, jak zadania MapReduce można napisać w Pythonie.

# Data Flow

Struktura MapReduce składa się z trzech głównych faz: mapowania, mieszania i sortowania oraz redukcji. W tej sekcji szczegółowo opisano każdą fazę.

## Map

Pierwsza faza aplikacji MapReduce to faza mapy. W fazie mapy funkcja (zwana mapowaniem) przetwarza serię par klucz-wartość. Moduł mapujący sekwencyjnie przetwarza każdą parę klucz-wartość osobno, tworząc zero lub więcej wyjściowych par klucz-wartość (Wykres 2-1).

Diagram

Description automatically generated

###### Wykres 2-1. Mapper jest stosowany do każdej wejściowej pary klucz-wartość, tworząc wyjściową parę klucz-wartość

Jako przykład rozważ mapera, którego celem jest przekształcenie zdań w słowa. Dane wejściowe do tego programu mapującego byłyby ciągami zawierającymi zdania, a funkcją programu mapującego byłoby podzielenie zdań na słowa i wyprowadzenie słów (Wykres2-2).

Diagram

Description automatically generated

###### Wykres 2-2. Dane wejściowe programu mapującego to łańcuch, a funkcją programu mapującego jest podzielenie danych wejściowych na spacje; wynik to pojedyncze słowa z wejścia mapującego

## Tasowanie i Sortowanie

Druga faza MapReduce to tasowanie i sortowanie. Gdy mapery zaczynają kończyć, pośrednie wyjścia z fazy mapowania są przenoszone do reduktorów. Ten proces przenoszenia danych wyjściowych z maperów do reduktorów jest znany jako tasowanie.

Tasowanie jest obsługiwane przez funkcję partycji, znaną jako partycjonator. Program partycjonujący służy do kontrolowania przepływu par klucz-wartość od programów mapujących do reduktorów. Partycjoner otrzymuje klucz wyjściowy programu mapującego i liczbę reduktorów i zwraca indeks zamierzonego reduktora. Partycjonowanie zapewnia, że wszystkie wartości dla tego samego klucza są wysyłane do tego samego reduktora. Domyślny program do partycjonowania jest oparty na hashowaniu. Oblicza wartość skrótu klucza wyjściowego programu mapującego i na podstawie tego wyniku przypisuje partycję.

Ostatnim etapem przed rozpoczęciem przetwarzania danych przez reduktory jest proces sortowania. Klucze pośrednie i wartości dla każdej partycji są sortowane przez platformę Hadoop przed przedstawieniem ich reduktorowi.

## Redukcja

Trzecia faza MapReduce to faza redukcji. W fazie reduktora iterator wartości jest dostarczany do funkcji znanej jako reduktor. Iterator wartości to nieunikalny zestaw wartości dla każdego unikalnego klucza z danych wyjściowych fazy mapy. Reduktor agreguje wartości dla każdego unikalnego klucza i generuje zero lub więcej wyjściowych par klucz-wartość (Wykres 2-3).

Diagram

Description automatically generated

###### Wykres 2-3. Reduktor iteruje po wartościach wejściowych, tworząc wyjściową parę klucz-wartość

Jako przykład rozważmy reduktor, którego celem jest zsumowanie wszystkich wartości klucza. Dane wejściowe tego reduktora są iteratorem wszystkich wartości klucza, a reduktor sumuje wszystkie wartości. Reduktor następnie wyprowadza parę klucz-wartość, która zawiera klucz wejściowy i sumę wartości klucza wejściowego (Wykres 2-4).

Diagram

Description automatically generated

###### Wykres 2-4. Ten reduktor sumuje wartości klawiszy „kot” i „mysz”

Następna sekcja opisuje prostą aplikację MapReduce i jej implementację w Pythonie.

# Hadoop Streaming

Przesyłanie strumieniowe Hadoop to narzędzie, które jest dostarczane z dystrybucją Hadoop i umożliwia tworzenie zadań MapReduce z dowolnym plikiem wykonywalnym jako maperem i/lub reduktorem. Narzędzie do przesyłania strumieniowego Hadoop umożliwia używanie języka Python, skryptów powłoki lub dowolnego innego języka jako mapera, reduktora lub obu.

## Jak to działa?

Mapper i reduktor są plikami wykonywalnymi, które odczytują dane wejściowe wiersz po wierszu ze standardowego wejścia (stdin) i zapisują dane wyjściowe na standardowe wyjście (stdout). Narzędzie do przesyłania strumieniowego Hadoop tworzy zadanie MapReduce, przesyła zadanie do klastra i monitoruje jego postęp do momentu zakończenia.

Po zainicjowaniu programu mapującego każde zadanie mapy uruchamia określony plik wykonywalny jako osobny proces. Mapper odczytuje plik wejściowy i prezentuje każdy wiersz w pliku wykonywalnym przez standardowe wejście. Po tym, jak plik wykonywalny przetworzy każdy wiersz danych wejściowych, program mapujący zbiera dane wyjściowe ze standardowego wyjścia i konwertuje każdy wiersz na parę klucz-wartość. Klucz składa się z części wiersza przed pierwszym znakiem tabulacji, a wartość składa się z części wiersza po pierwszym znaku tabulacji. Jeśli wiersz nie zawiera znaku tabulacji, cały wiersz jest uważany za klucz, a wartość jest pusta.

Po zainicjowaniu reduktora każde zadanie redukcji uruchamia określony plik wykonywalny jako osobny proces. Reduktor konwertuje wejściową parę klucz-wartość na wiersze, które są prezentowane w pliku wykonywalnym przez standardowe wejście. Reduktor zbiera wyniki plików wykonywalnych ze standardowego wyjścia i konwertuje każdy wiersz na parę klucz-wartość. Podobnie jak mapper, plik wykonywalny określa pary klucz-wartość, oddzielając klucz i wartość za pomocą znaku tabulacji

## Przykład w Pythonie

Aby zademonstrować, jak narzędzie do przesyłania strumieniowego Hadoop może uruchamiać język Python jako aplikację MapReduce w klastrze Hadoop, aplikację WordCount można zaimplementować jako dwa programy w języku Python: *mapper.py* i *reducer.py*.

*mapper.py* to program w Pythonie, który implementuje logikę w fazie mapy WordCount. Czyta dane z stdin, dzieli wiersze na słowa i wyprowadza każde słowo z jego pośrednią liczbą na standardowe wyjście. Kod w Przykładzie 2-1 implementuje logikę *mapper.py*.

##### Przykład 2-1. python/MapReduce/HadoopStreaming/mapper.py

#!/usr/bin/env python

import sys

# Read each line from stdin

for line in sys.stdin:

# Get the words in each line

words = line.split()

# Generate the count for each word

for word in words:

# Write the key-value pair to stdout to be processed by

# the reducer.

# The key is anything before the first tab character and the

#value is anything after the first tab character.

print('{0}\t{1}'.format(word, 1))

*reducer.py* is program w Pythonie, który implementuje logikę w fazie redukcji WordCount. Czyta wyniki *mapper.py* ze standardowego wejścia (stdin), sumuje wystąpienia każdego słowa i zapisuje wynik na standardowe wyjście (stdout). Kod w Przykładzie 2-2 implementuje logikę *reducer.py*.

##### Przykład 2-2. python/MapReduce/HadoopStreaming/reducer.py

#!/usr/bin/env python

import sys

curr\_word = None

curr\_count = 0

# Process each key-value pair from the mapper

for line in sys.stdin:

# Get the key and value from the current line

word, count = line.split('\t')

# Convert the count to an int

count = int(count)

# If the current word is the same as the previous word,

# increment its count, otherwise print the words count

# to stdout

if word == curr\_word:

curr\_count += count

else:

# Write word and its number of occurrences as a key-value

# pair to stdout

if curr\_word:

print('{0}\t{1}'.format(curr\_word, curr\_count))

curr\_word = word

curr\_count = count

# Output the count for the last word

if curr\_word == word:

print('{0}\t{1}'.format(curr\_word, curr\_count))

Przed próbą wykonania kodu upewnij się, że pliki *mapper.py* i *reducer.py* mają uprawnienia do wykonywania. Następujące polecenie umożliwi to dla obu plików:

$ chmod a+x mapper.py reducer.py

Upewnij się również, że pierwszy wiersz każdego pliku zawiera prawidłową ścieżkę do Pythona. Ta linia umożliwia plikom *mapper.py* i *reducer.py* wykonywanie jako samodzielne pliki wykonywalne. Wartość #!/usr/bin/env python powinna działać w większości systemów, ale jeśli nie, wymień /usr/bin/env python ze ścieżką do pliku wykonywalnego Pythona w twoim systemie.

Aby przetestować programy Pythona lokalnie przed uruchomieniem ich jako zadania MapReduce, można je uruchomić z poziomu powłoki za pomocą poleceń echo i sort. Zdecydowanie zaleca się przetestowanie wszystkich programów lokalnie przed uruchomieniem ich w klastrze Hadoop.

$ echo 'jack be nimble jack be quick' | python3 ./mapper.py | sort -t 1 | python3 ./reducer.py

be 2

jack 2

nimble 1

quick 1

Gdy programy mapowania i reduktora pomyślnie wykonają testy, można je uruchomić jako aplikację MapReduce za pomocą narzędzia do przesyłania strumieniowego Hadoop. Polecenie do uruchamiania programów Pythona *mapper.py* i *reducer.py* w klastrze Hadoop wygląda następująco:

UWAGA: plik tekstowy należy umieścić na HDFS w folderze user albo o innej nazwie..

TO JEST JEDNA KOMENDA:

/usr/local/hadoop/bin/hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.4.jar -file ~/Desktop/mapper.py -mapper "python3 mapper.py" -file ~/Desktop/reducer.py -reducer "python3 reducer.py" -input /user/short\_story.txt -output /ścieżka\_usera/nazwa\_pliku\_output.txt

Opcje używane z narzędziem do przesyłania strumieniowego Hadoop są wymienione w Tabeli 2-1.

| Opcja | Opis |
| --- | --- |
| -files | Rozdzielona przecinkami lista plików do skopiowania do klastra MapReduce |
| -mapper | Polecenie do uruchomienia jako maper |
| -reducer | Polecenie do uruchomienia jako reduktor |
| -input | Ścieżka wejściowa DFS dla kroku Map |
| -output | Katalog wyjściowy DFS dla kroku Reduce |
| Tabela 2-1. Opcje do przesyłania strumieniowego Hadoop | |