Implementacja i analiza wydajności algorytmu word count z użyciem MapReduce na klastrze EMR

Mateusz Łopaciński

19 maja 2025

1 Wprowadzenie

Celem ćwiczenia jest analiza porównawcza efektywności implementacji algorytmu word count w wersji sekwencyjnej oraz równoległej przy użyciu technologii Hadoop MapReduce. Eksperymenty przeprowadzono na klastrze Amazon EMR dla zbiorów danych o rozmiarach 1GB, 5GB oraz 10GB, utworzonych przez powielenie pliku źródłowego o rozmiarze 500MB.

2 Opis eksperymentu

Eksperyment przeprowadzono na klastrze EMR w dwóch konfiguracjach sprzętowych z identyczną łączną liczbą rdzeni (32 vCPU):

- **Mocniejsze węzły**: 2 węzły typu `m5.4xlarge` (po 16 vCPU każdy)
- **Słabsze węzły**: 4 węzły typu `m5.2xlarge` (po 8 vCPU każdy)

Implementację sekwencyjną uruchomiono na pojedynczym węźle Primary klastra EMR typu `m5.4xlarge` oraz `m5.2xlarge`, dopasowując sprzętowo wersję sekwencyjną do testów równoległych.

Każdy eksperyment przeprowadzano dwukrotnie, wynik końcowy to średnia z tych dwóch pomiarów.

3 Opis optymalnej implementacji sekwencyjnej

Sekwencyjna wersja algorytmu została zoptymalizowana poprzez użycie efektywnej struktury danych `Counter` z biblioteki standardowej języka Python. Dzięki zastosowaniu strumieniowego odczytu danych ograniczono użycie pamięci, zaś jednoprzebiegowe przetwarzanie zapewnia minimalizację liczby operacji wejścia-wyjścia. Złożoność algorytmu wynosi O(n), gdzie n to liczba słów w pliku wejściowym, co potwierdza teoretyczną optymalność zastosowanego podejścia.

3.1 Kod implementacji sekwencyjnej

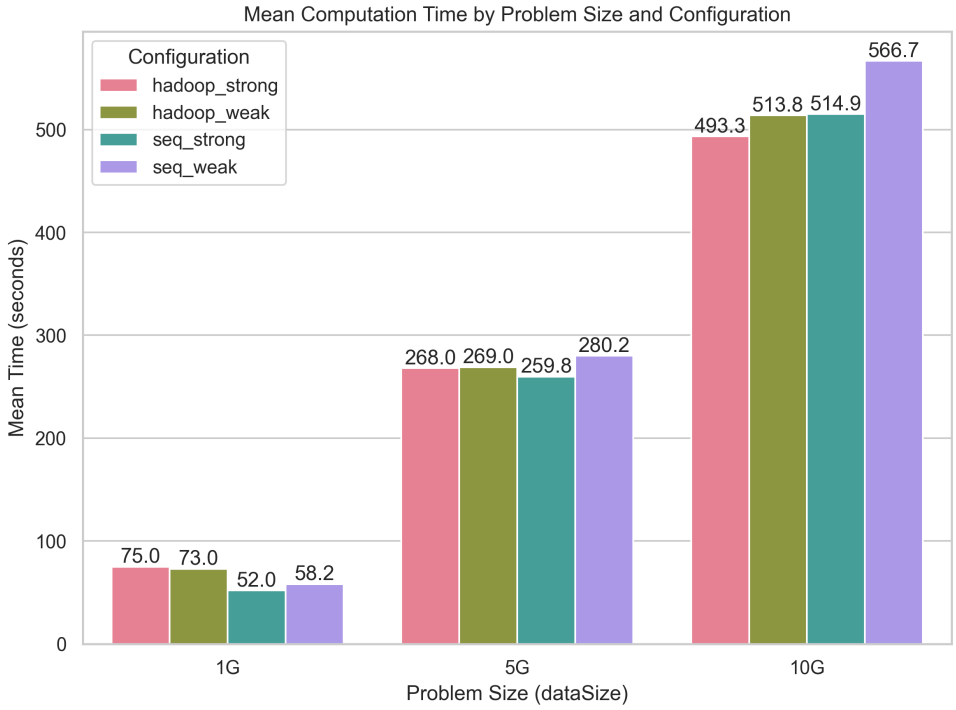
```python
#!/usr/bin/env python3
from collections import Counter
import sys
import time

def word_count(filename):
    word_counter = Counter()
    with open(filename, 'r', encoding='utf-8', errors='ignore') as f:
        for line in f:
            word_counter.update(line.strip().split())
    return word_counter

if __name__ == '__main__':
    start_time = time.perf_counter()
    result = word_count(sys.argv[1])
    for word, count in result.most_common():
        print(f"{word}\t{count}")
    elapsed = time.perf_counter() - start_time

    print(f"Elapsed time: {elapsed:.6f} seconds", file=sys.stderr)
```

4 Wyniki i analiza

Wyniki przedstawiono z użyciem identyfikatorów konfiguracji:

- **seq_strong**: Sekwencyjnie na węźle `m5.4xlarge` (16 vCPU).
- **seq_weak**: Sekwencyjnie na węźle `m5.2xlarge` (8 vCPU).
- **hadoop_strong**: Hadoop na 2 węzłach `m5.4xlarge` (32 vCPU).
- **hadoop_weak**: Hadoop na 4 węzłach `m5.2xlarge` (32 vCPU).

4.1 Czasy obliczeń



Rysunek 1: Czas obliczeń w zależności od rozmiaru problemu oraz konfiguracji

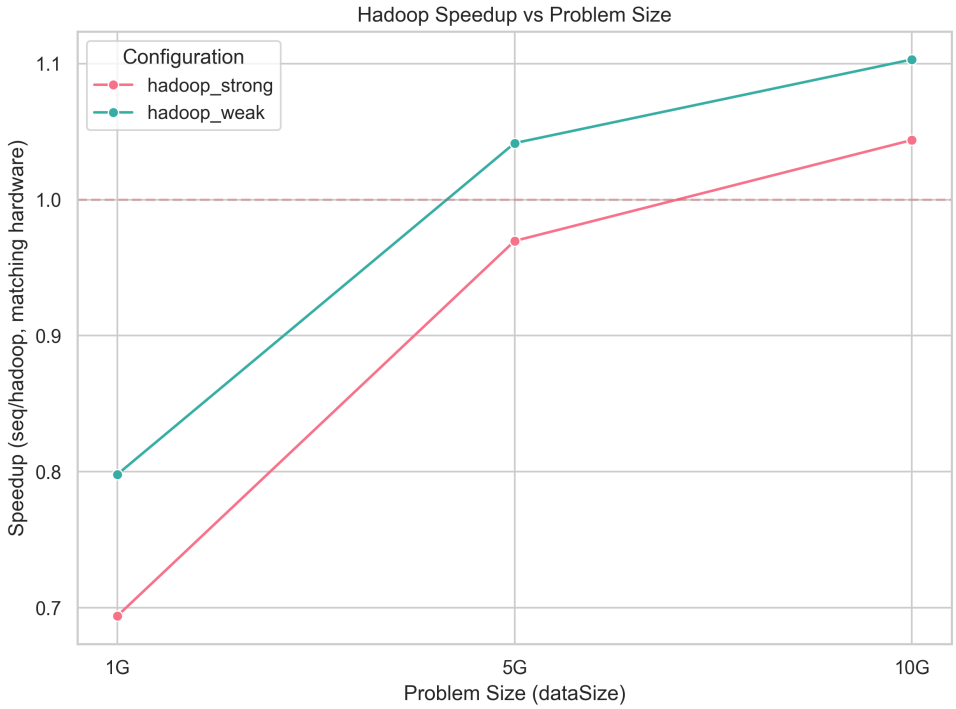
Na wykresie przedstawiono średnie czasy obliczeń algorytmu *word count* w zależności od konfiguracji sprzętowej oraz wielkości danych wejściowych (1GB, 5GB, 10GB).

Najważniejsze obserwacje:

- Dla najmniejszego zbioru danych (1GB):
  - Implementacja sekwencyjna jest zdecydowanie szybsza niż Hadoop.
  - Widoczne są znaczne narzuty Hadoop wynikające z rozproszenia zadań.
- Dla średniego zbioru danych (5GB):
  - Czasy obu implementacji zbliżają się do siebie, jednak implementacja sekwencyjna pozostaje nieco szybsza.
  - Narzuty komunikacyjne Hadoop wciąż dominują, ale różnica się zmniejsza.
- Dla największego zbioru danych (10GB):
  - Implementacja Hadoop staje się wydajniejsza od sekwencyjnej, co potwierdza, że równoległość rekompensuje dodatkowe koszty dystrybucji danych.

Podsumowując, wyniki wskazują na zalety Hadoop dopiero dla dużych zbiorów danych, co jest zgodne z oczekiwaniami teoretycznymi dotyczącymi technologii MapReduce.

4.2 Analiza przyspieszenia



Rysunek 2: Hadoop Speedup vs Problem Size

Wykres prezentuje przyspieszenie implementacji Hadoop względem implementacji sekwencyjnej dla różnych rozmiarów danych (1GB, 5GB, 10GB). Wartości przyspieszenia powyżej 1 wskazują na przewagę rozwiązania Hadoop.

Kluczowe obserwacje:

- Dla danych 1GB:
  - Przyspieszenie poniżej 1 (około 0.7-0.8), co wskazuje na wyraźnie wolniejszą realizację zadań przez Hadoop.
  - Dominują koszty związane z komunikacją i koordynacją procesu.
- Dla danych 5GB:
  - Przyspieszenie nadal poniżej 1, ale znacznie bliżej tej granicy.
  - Malejący wpływ kosztów komunikacji w stosunku do obliczeń.
- Dla danych 10GB:
  - Hadoop wykazuje przyspieszenie przekraczające 1, co potwierdza efektywność podejścia równoległego dla dużych zbiorów danych.
  - Konfiguracja `hadoop_weak` (więcej słabszych węzłów) wykazuje nieco lepszą skalowalność w porównaniu z `hadoop_strong` (mniej silniejszych węzłów).

Wnioski z tego wykresu jasno sugerują, że technologia Hadoop MapReduce najlepiej sprawdza się dla dużych zbiorów danych, gdzie zyski wynikające z równoległości przewyższają koszty komunikacji i zarządzania klastrem.

5 Analiza metryki COST

Metryka COST, definiowana jako iloczyn liczby rdzeni oraz czasu wykonania obliczeń, pozwala ocenić całkowity koszt zasobów obliczeniowych użytych przez implementację równoległą względem wersji sekwencyjnej:

$$\text{COST} = \text{liczba rdzeni} \times \text{czas obliczeń}$$

Na podstawie wyników uzyskanych w eksperymentach można stwierdzić:

- Dla mniejszych danych (1GB oraz 5GB), implementacja równoległa używa znacznie więcej zasobów (32 rdzenie), a czas wykonania jest dłuższy lub podobny jak w wersji sekwencyjnej, co przekłada się na znacznie wyższy COST.
- Dla największego badanego zbioru (10GB), mimo iż implementacja równoległa uzyskała nieznacznie krótszy czas wykonania, jej COST jest nadal wielokrotnie wyższy niż COST implementacji sekwencyjnej (ok. 15840 core-sekund vs. ok. 514 core-sekund).

Wynika z tego jednoznacznie, że w żadnym z badanych przypadków implementacja równoległa nie uzyskała korzystniejszej metryki COST niż rozwiązanie sekwencyjne. Hadoop może przyspieszyć obliczenia, ale kosztem znacznego zwiększenia całkowitego zużycia zasobów.

6 Podsumowanie i wnioski

Eksperyment wykazał, że technologia Hadoop MapReduce jest odpowiednia przede wszystkim dla dużych zbiorów danych. Przy mniejszych danych optymalna implementacja sekwencyjna pozostaje bardziej efektywna.

Dalsze prace mogłyby koncentrować się na analizie wydajności dla większych zbiorów danych oraz na testach różnych konfiguracji węzłów w celu lepszego określenia granicy opłacalności stosowania technologii Hadoop.