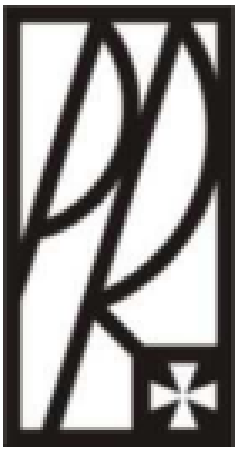


POLITECHNIKA RZESZOWSKA im. Ignacego Łukasiewicza

WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

PRACA POJEKTOWA

Usługi sieciowe w biznesie

„Cassandra vs OracleDB – porównanie wydajności”

Filip Kosiorowski 173161

Inżynieria i analiza danych, semestr 6, grupa L2

Rzeszów 2025

1. Cel projektu

Celem projektu jest porównanie wydajności dwóch systemów zarządzania bazami danych: relacyjnej bazy danych **Oracle Database Express Edition (Oracle XE)** oraz nierelacyjnej bazy danych **Apache Cassandra**. Obie technologie reprezentują odmienne podejścia do przechowywania i przetwarzania danych. Oracle to klasyczny system RDBMS oparty na języku SQL i modelu relacyjnym, a Cassandra to rozproszona, kolumnowa baza NoSQL stworzona z myślą o skalowalności i wysokiej dostępności.

2. Specyfikacje techniczne

Testy były przeprowadzane na laptopie LENOVO IdeaPad Gaming 3 z roku 2021, jego specyfikacje to procesor AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz, 16 GB pamięci RAM, karta graficzna NVIDIA GeForce GTX 1650, 512 dysk SSD z systemem operacyjnym Windows 11 Home.

3. Użyte środowisko

Plan pierwotny zakładał użycie dwóch maszyn wirtualnych z systemem Windows 10. Pomysł ten został niedługo później odrzucony z racji na niedostateczną wydajność środowiska. Maszyna na Windowsie pochłaniała zbyt dużo zasobów komputera, do tego stopnia, że niemożliwe byłoby operowanie na niej w umożliwiającym prace tempie. Ponadto instalacja Cassandry na Windowsie przekroczyła moje umiejętności i po wielu próbach i paru godzinach spędzonych przy planie pierwotnym porzuciłem go.

Drugi plan zakładał użycie dwóch maszyn wirtualnych z Linuxem. Rozwiązywało to niedostateczną wydajność i problem z instalacją Cassandry, ponieważ jest ona wysoce kompatybilna z tym systemem. Niestety pojawiły się kolejne komplikacje natury technicznej, częste wieszanie się maszyny, niezależnie od użytego programu, czy to VirtualBox czy Vmware. Wiązały się z tym jeszcze częste crashe i bluescreeny zarówno na maszynie wirtualnej jak i na maszynie głównej. Po naprawieniu problemów z jedną maszyną, występowały problemy w drugiej.

By nie poświęcać idei projektu, co można byłoby osiągnąć robiąc prawdopodobnie dwie różne maszyny, nie mając też możliwości zaopatrzenia się w mocniejszy sprzęt postanowiłem

wykorzystać Dockera. Oficjalny obraz Cassandry dostępny na Dockerze umożliwia szybkie i elastyczne uruchamianie środowiska w kontenerze. Dzięki temu użytkownicy mogą łatwo przetestować działanie klastra bazy danych lub zintegrować Cassandrę z aplikacjami w środowisku deweloperskim. Kontener oparty jest na systemie Debian i dostępny jest w wielu wersjach, co pozwala na wybór odpowiedniej wersji Cassandry do konkretnego projektu. Do uruchomienia kontenera wystarczy jedno polecenie docker run, które tworzy i startuje pojedynczy węzeł bazy danych. Istnieje również możliwość tworzenia wielu kontenerów w celu skonfigurowania klastra Cassandra, wykorzystując zmienne środowiskowe, takie jak CASSANDRA_SEEDS, CASSANDRA_LISTEN_ADDRESS
czy CASSANDRA_BROADCAST_ADDRESS. Użytkownik może także dostosować konfigurację klastra, nazwę klastra (CASSANDRA_CLUSTER_NAME), topologię fizyczną (data center i rack), a także ustawić parametry dotyczące komunikacji z innymi węzłami lub klientami (np. adres RPC). Domyślnie dane przechowywane są w katalogu /var/lib/cassandra, jednak można łatwo podłączyć wolumen lub katalog z systemu hosta, aby zapewnić trwałość danych po restarcie kontenera (w moim przypadku /tmp). Cassandra w kontenerze obsługuje również narzędzie cqlsh, które pozwala na bezpośrednią interakcję z bazą danych za pomocą języka CQL (Cassandra Query Language). Dzięki wsparciu dla różnych architektur procesora i szerokim możliwościom konfiguracji, Cassandra uruchomiona z oficjalnego obrazu Dockera jest wygodnym i efektywnym rozwiązaniem do budowy oraz testowania skalowalnych baz danych w środowisku kontenerowym.

Oracle Database XE (Express Edition) w postaci kontenera gvenzl/oracle-xe, dostępny na Dockerze, pozwolił na pracę w dostatecznych warunkach, choć system Oracle Linux nie jest bardzo kooperatywny.

Do testów użyto dwóch zestawów danych o różnej zasobności. Plik Auta.csv zawiera dane odnośnie samochodów elektrycznych, ma 250 tysięcy rekordów i waży 56,8 MB. Drugi plik to produkty.csv, zawiera opisy pewnych produktów. Ilość rekordów to równy milion a zapis waży 155MB. Pliki niepozbawione są błędów, pustych wpisów, niepoprawnie zapisanych rekordów, więc musiały zostać poprawione.

Import danych jest jedną z form testów wydajności bazy danych. Zostało to wykonane w następujący sposób:

1. Skopiowanie pliku csv do odpowiedniego kontenera
2. Utworzenie odpowiedniej tabeli

3. Skopiowanie zawartości pliku csv do tabeli

Podczas przystępowania do importu danych do Cassandry występowały następujące błędy:

ParseError - Failed to parse <wartość Size> : [<class 'decimal.ConversionSyntax'>]

- Cassandra próbowała wstawić dane tekstowe jako liczby całkowite.
- Rozwiązanie: w kolumnie size występowały różne wartości, np. 30x40 cm lub S,M,XL itp. Należało umieścić te wartości w cudzysłowie by nie były odczytywane jako liczba

Cannot rename non PRIMARY KEY column cafv_eligibility

- Cassandra nie pozwala na zmianę nazw kolumn niebędących kluczami głównymi.

Improper COPY command

A**wk nie zwracał wyniku (brak działania komendy)**

- Błąd w numerze kolumny podczas przetwarzania CSV ($12="\""$11"\"" zamiast odpowiedniej kolumny size).
- Skutek: plik nie był poprawnie modyfikowany, a błędne dane trafiały do COPY.

Niezgodność kolejności kolumn między tabelą a CSV

- Przykład: COPY używało innej kolejności niż nagłówki pliku, co prowadziło do błędnego przypisywania danych i błędów typu.

Podczas przystępowania do importu danych do Oracla występowały następujące błędy:

ORA-04036: PGA memory used by the instance exceeds PGA_AGGREGATE_LIMIT

- Oznacza że przekroczono przydzielony limit pamięci dla operacji procesora (PGA). Należy zalogować się na sysdba i użyć komendy:
  ALTER SYSTEM SET PGA_AGGREGATE_LIMIT = 6G CONTAINER=ALL; (mi udało się dla 6Gb, przy lepszym sprzęcie można zwiększyć)

ORA-03114: not connected to ORACLE

- Sesja SQL została zerwana.

ORA-03113: end-of-file on communication channel

- Błąd komunikacyjny z instancją Oracle, np. przez restart lub crash.

ORA-12899: value too large for column

- tekst dłuższy niż maksymalna długość kolumny (np. VARCHAR2(100) dostaje 105 znaków).

ORA-01861: literal does not match format string

- Występowało przy kopiowaniu pliku produkty.csv. Oracle używa ‘ jako znaku który kończy i zaczyna string. Niestety niektóre produkty miały np. nazwę Men’s clothing, co powodowało duże problemy w imporcie. Naprawa musiała nastąpić manualnie w Visual Studio Code za pomocą ctrl+H.

4. Testy

a) Test wydajności importu

Celem testu jest sprawdzenie, jak długo trwa wstawienie (import) danych z plików CSV do odpowiednich tabel w obu bazach danych, przy porównywalnym rozmiarze danych i strukturze tabel.

W tym celu stworzono odpowiednie tabele

Dla OracleXE:

```
CREATE TABLE vehicles (
    VIN VARCHAR2(200),
    County VARCHAR2(200),
    City VARCHAR2(200),
    State VARCHAR2(200),
    Postal_Code VARCHAR2(200),
    Model_Year VARCHAR2(200),
    Make VARCHAR2(200),
    Model VARCHAR2(200),
    Electric_Vehicle_Type VARCHAR2(200),
    CAFV_Eligibility VARCHAR2(200),
    Electric_Range NUMBER,
    Base_MSRP NUMBER,
    Legislative_District VARCHAR2(200),
    DOL_Vehicle_ID VARCHAR2(200),
    Vehicle_Location VARCHAR2(200),
    Electric_Utility VARCHAR2(200),
    Census_Tract VARCHAR2(200)
);
```

```
CREATE TABLE produkty (
    index_id          NUMBER PRIMARY KEY,
    name              VARCHAR2(100),
    description       VARCHAR2(500),
    brand             VARCHAR2(100),
    category          VARCHAR2(100),
    price             NUMBER(10, 2),
    currency          VARCHAR2(10),
    stock             NUMBER,
    ean               VARCHAR2(20),
    color             VARCHAR2(50),
    size              VARCHAR2(50),
    availability      VARCHAR2(30),
    internal_id       NUMBER
);
```

Dla Cassandry

```
CREATE TABLE vehicles (
    vin TEXT PRIMARY KEY,
    county TEXT,
    city TEXT,
    state TEXT,
    postal_code TEXT,
    model_year INT,
    make TEXT,
    model TEXT,
    electric_vehicle_type TEXT,
    cafv_eligibility TEXT,
    electric_range INT,
    base_msrp INT,
    legislative_district INT,
    dol_vehicle_id TEXT,
    vehicle_location TEXT,
    electric_utility TEXT,
    census_tract TEXT
);
```

```
CREATE TABLE sklep.produkty (
    index_id text PRIMARY KEY,
    name text,
    description text,
    brand text,
    category text,
    price decimal,
    currency text,
    stock int,
    ean text,
    color text,
    size text,
    availability text,
    internal_id bigint
);
```

Import do tabeli Vehicles przebiegał bezproblemowo, w przypadku OracleXE wyglądało to tak następująco:

- Skopiowanie pliku csv do katalogu tmp w kontenerze
- Napisanie skryptu piszącego Inserty, które później zostały uruchomione (nie miałem żadnej możliwości użycia lub instalacji SQL*Loadera)

```
#!/bin/bash

START=$(date +%s)

# Wygeneruj import.sql z CSV
awk -F',' 'BEGIN {
  print "SET DEFINE OFF;";
}
NR > 1 {
  printf "INSERT INTO vehicles VALUES (";
  for (i = 1; i <= NF; i++) {
    gsub(/'\''/, "'\''", $i);
    printf "'\''%s'\''", $i;
    if (i < NF) printf ", ";
  }
  print ");"
}
END {
  print "COMMIT;";
}' /tmp/auta.csv > /tmp/import.sql

# Uruchom import
docker exec -i 31d85b2c5a69 sqlplus -s system/oracle@localhost:1521/XEPDB1 @/tmp/import.sql
```

Skrypt dla vehicles: operacja generowania insertów trwała 4 min i 30 sekund, czas procesowania importów wyniósł 3 minuty 1 sekundę.

insert_produkty.ps1

```
# Ścieżki do plików
$csvFilePath = "C:\Users\kosio\Desktop\Studia\Semestr6\ProjektCassandra\produkty.csv"  # Ścieżka do pliku CSV
$sqlFilePath = "C:\Users\kosio\Desktop\Studia\Semestr6\ProjektCassandra\import_produkty.sql"  # Ścieżka do pliku SQL
# Start pomiaru czasu
$StartTime = Get-Date
# Otwórz plik CSV
$csvContent = Import-Csv -Path $csvFilePath
$header = "SET DEFINE OFF;"
Add-Content -Path $sqlFilePath -Value $header
# Dodaj instrukcje INSERT dla każdego wiersza
foreach ($row in $csvContent) {
    $values = @(
    "'$($row.Index)'",
    "'$($row.Name)'",
    "'$($row.Description)'",
    "'$($row.Brand)'",
    "'$($row.Category)'",
    "'$($row.Price)'",
    "'$($row.Currency)'",
    "'$($row.Stock)'",
    "'$($row.EAN)'",
    "'$($row.Color)'",
    "'$($row.Size)'",
    "'$($row.Availability)'",
    "'$($row.'Internal ID')'"
)
    $insertQuery = "INSERT INTO produkty VALUES (" + ($values -join ", ") + ");"
    Add-Content -Path $sqlFilePath -Value $insertQuery
}
Add-Content -Path $sqlFilePath -Value "COMMIT;"
$EndTime = Get-Date
$ElapsedTime = $EndTime - $StartTime
Write-Host "Cała operacja trwała $($ElapsedTime.TotalSeconds) sekund."
```

Skrypt dla produktów był dużo bardziej skomplikowany i miał o mnóstwo problemów, prostsze rozwiązania nie działy a ilość rekordów stawiała wyzwanie dla systemu.

```
Przetworzono 1000000 rekordÃ³w...

Wygenerowano plik SQL z 1000000 rekordami w 191.68 minut.
```

Po wygenerowaniu plik z insertami wymagał paru obróbek, po wielu próbach przeprocesowania insertów (tu występowały problemy ze zbyt małą pamięcią oraz częste crashe Dockera) udało się prawie poprawnie umieścić dane w tabeli (w produktach występowało dużo wartości pustych (jak na milion rekordów to nie aż tak dużo jednak blokowało to znacznie proces insertowania). Czas procesowania insertów dla miliona wartości wyniósł 18 minut i 37 sekund.

Import do tabel w Cassandrze przebiegał bezproblemowo, wyglądało to tak następująco:

- Skopiowanie plików csv do katalogu tmp

```
- docker cp C:\Users\kosio\Desktop\Studia\Semestr6\ProjektCassandra\produkty.csv cassandra:/tmp/produkty.csv
- docker cp C:\Users\kosio\Desktop\Studia\Semestr6\ProjektCassandra\auta.csv cassandra:/tmp/auta.csv
```

- w kontenerze napisanie odpowiedniej komendy

```
- cqlsh -e "COPY sklep.vehicles FROM '/tmp/auta.csv' WITH HEADER = true;"
- cqlsh -e "COPY sklep.produkty FROM '/tmp/produkty.csv' WITH HEADER = true;"
```

Import vehicles: Processed: 246137 rows; Rate: 47091 rows/s; Avg. rate: 24255 rows/s

246137 rows imported from 1 files in 0 day, 0 hour, 0 minute, and 10.148 seconds (0 skipped).

Import produkty:

```
Failed to process 1 rows; failed rows written to import_sklep_produkty.err
Processed: 1000001 rows; Rate:   19598 rows/s; Avg. rate:   31788 rows/s
1000000 rows imported from 1 files in 0 day, 0 hour, 0 minute, and 31.459 seconds (0 skipped).
cqlsh:sklep>
```

Cassandra przeprocesowała inserty vehicles 27 razy szybciej a produktów 36 razy szybciej od OracleXE.

b) Select * from

Celem testu jest porównanie czasu potrzebnego na wykonanie pełnego zapytania pobierającego wszystkie dane z tabeli w dwóch różnych systemach bazodanowych. Test pozwala ocenić, jak obie bazy radzą sobie z prostym, ale potencjalnie kosztownym zapytaniem przy dużych wolumenach danych.

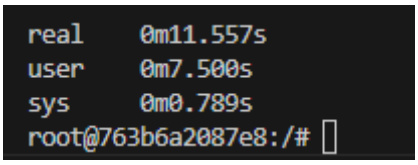
Dla OracleXE

Produkty

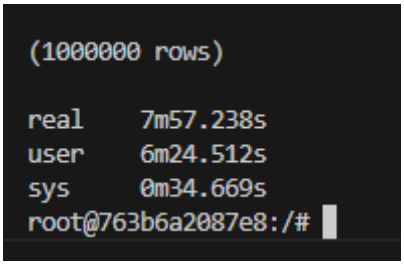
Niestety w tym momencie wystąpił dziwny crash, uniemożliwiający dokończenie kwerendy, ładowała się ona przez ponad 7 minut. Ponadto cała tabela z produktami została usunięta. Kolejne testy na OracleXE będą tylko przeprowadzane na vehicles ze względów wydajnościowych.

Dla Cassandry

Vehicles



Produkty



c) Czas trwania SELECT z filtrowaniem po 3 kolumnach

Dla Oracle

SELECT * FROM vehicles

WHERE state = 'WA' AND make = 'TESLA' AND model_year = 2020;

Dla Cassandry

time cqlsh -e "SELECT * FROM sklep.vehicles WHERE state = 'WA' AND make = 'TESLA' AND model_year = 2020 ALLOW FILTERING;"

```
real    0m0.656s
user    0m0.374s
sys     0m0.080s
root@763b6a2087e8:/#
```

time cqlsh -e "SELECT * FROM sklep.produkty WHERE brand = 'Perkins Inc' AND category = 'Fitness Equipment' AND currency = 'USD' ALLOW FILTERING;"

```
 index_id | availability | brand       | category          | color         | currency | description                 | ean           | internal_id | name                                | price | size       | stock
----------+--------------+-------------+-------------------+---------------+----------+-----------------------------+---------------+-------------+-------------------------------------+-------+------------+-------
   213531 |   100x200 mm | Perkins Inc | Fitness Equipment |  AntiqueWhite |      USD |  Building indeed camera high. | 6027482545588 |          73 | Clean Trimmer Sense Go Rechargeable |   803 | 100x200 mm |   739
   862086 |           XL | Perkins Inc | Fitness Equipment | PaleGoldenRod |      USD | Edge skill old international. | 5941280536610 |          82 |  Ultra Fan Lamp Wireless Clean One |   799 |         XL |   217

(2 rows)

real    0m3.179s
user    0m0.349s
sys     0m0.073s
root@763b6a2087e8:/#
```

d) Test SELECT z filtrowaniem i limitowaniem

Dla Oracle

SELECT * FROM vehicles

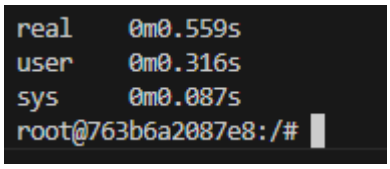
WHERE state = 'CA' AND make = 'TESLA'

FETCH FIRST 100 ROWS ONLY

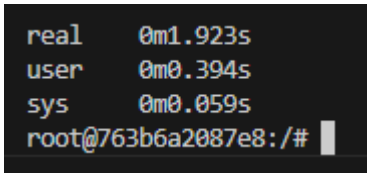
```
75 rows selected.

Elapsed: 00:00:01.04
SQL>
```

Dla Cassandry

time cqlsh -e "SELECT * FROM sklep.vehicles WHERE state = 'CA' AND make = 'TESLA' LIMIT 100 ALLOW FILTERING;"



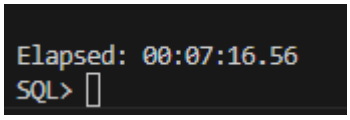
time cqlsh -e "SELECT * FROM sklep.produkty WHERE category = 'Fitness Equipment' AND currency = 'USD' AND color = 'LemonChiffon' LIMIT 100 ALLOW FILTERING;"



e) Przeszukiwanie z warunkiem zakresu

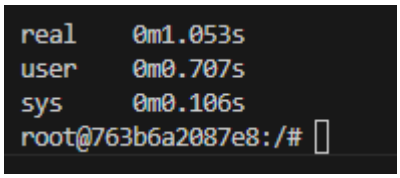
Oracle

SELECT * FROM vehicles WHERE electric_range > 200;



Cassandra

time cqlsh -e "SELECT * FROM sklep.vehicles WHERE electric_range > 200 ALLOW FILTERING;"



Test	Cassandra	OracleXE	Cassandra szybsza o
Import 250k (vehicles)	10s	4:30min	~27x
Import 1M (products)	31s	18:37 min	~36x
Select * from vehicles	11s	>7 min	~38x
Select * from produkty	7:57 min	-	Obsługuje
SELECT z 3 filtrami	0,5s	1s	~5x
Przeszukiwanie z zakresem	1s	7:16 min	~436x

5. Krytyka i wnioski

W tej części chciałem się skupić na tym co można było przeprowadzić lepiej by testy dały dużo lepsze wartości.

- użycie od razu dockera jak w finalnej wersji

- zaopatrzenie się w lepszy sprzęt, by udźwignął dwie maszyny wirtualne lub by operacje wykonywały się w lepszym czasie

- znalezienie lepszych danych, bez błędów, które trzeba było naprawiać

- poświęcenie znacznie większej liczby czasu na łatki i optymalizację

- automatyzacja oczyszczania danych, np. w Pythonie

- dołączenie więcej metryk wydajności, nie tylko czas ale także CPU i RAM. Nie udało się to z racji na okazjonalne zamrażanie się programu podczas wykonywania kwerend

Idealny projekt to dwie maszyny wirtualne na Linuxie. Z odpowiednią mocą obliczeniową komputera i lepszymi danymi można przygotować bardzo dokładne porównanie obu baz danych. Jednakże już istnieją odpowiednie benchmarki, służące tylko i wyłącznie do mierzenia wydajności OracleDB i Cassandry, służą temu odpowiednio Oracle SQL Developer i cassandra-stress. Z perspektywy użytkownika obie bazy są bardzo dobrze opisane i łatwo znaleźć informacje o tym, która jest w czym lepsza. Z perspektywy badawczej, przygotowany

specjalnie po to benchmark zawsze będzie lepszym miernikiem niż przygotowane przez badacza środowisko, chyba że badacz chce skorzystać z niezależnego narzędzia do mierzenia. Wtedy powinien zrobić je samodzielnie, najlepiej na dwóch maszynach wirtualnych i bardzo silnym komputerze.

Wnioski

Projekt, mimo licznych trudności technicznych oraz ograniczeń środowiskowych, pozwolił na uzyskanie szeregu cennych obserwacji dotyczących praktycznego wykorzystania dwóch różnych podejść do zarządzania danymi – relacyjnego (OracleXE) i nierelacyjnego (Cassandra). Testy wykazały, że Cassandra zdecydowanie lepiej radzi sobie z operacjami masowego importu danych oraz prostymi zapytaniami odczytu, szczególnie przy dużych zbiorach danych. Jej architektura rozproszona i szybkość działania czynią ją dobrym wyborem dla aplikacji wymagających skalowalności i wysokiej dostępności.

Z drugiej strony, efektywne wykorzystanie Cassandry wymaga bardzo starannego zaprojektowania modelu danych – pod konkretne zapytania – oraz świadomości jej ograniczeń (m.in. brak natywnych agregacji, ograniczone filtrowanie bez indeksów, brak JOIN-ów). OracleXE, choć znacznie bardziej zasobożerny i wolniejszy w przypadku dużych operacji na danych, oferuje bogaty zestaw funkcji – pełną kontrolę nad transakcjami, spójność, możliwość używania zaawansowanych zapytań, agregacji i złączeń. Jest więc lepszym wyborem tam, gdzie priorytetem są precyzja, złożoność relacji i niezawodność w przetwarzaniu danych.

Wnioski z projektu jednoznacznie pokazują, że wybór bazy danych powinien być uzależniony nie tyle od samej wydajności, co od charakteru danych i wymagań biznesowych konkretnego projektu. Cassandra będzie optymalna dla systemów Big Data, telemetrii czy rejestrów zdarzeń, natomiast Oracle sprawdzi się w aplikacjach transakcyjnych, finansowych i wszędzie tam, gdzie ważna jest integralność relacji między danymi.

6. Źródła

https://hub.docker.com/_/cassandra

https://hub.docker.com/r/gvenzl/oracle-xe

https://catalog.data.gov/dataset/electric-vehicle-population-data

https://github.com/datablist/sample-csv-files?tab=readme-ov-file