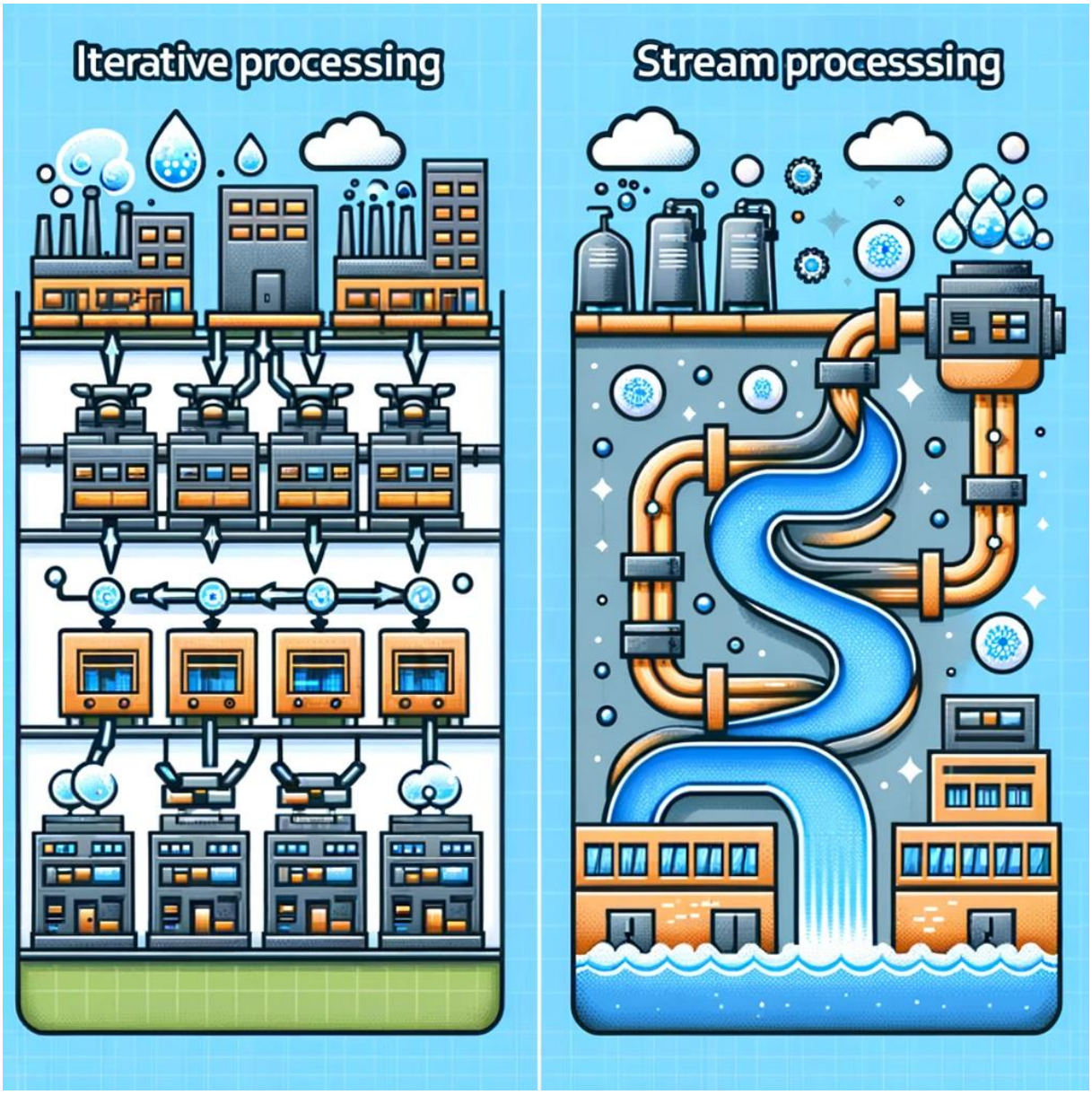
RAPORT

Przetwarzanie strumieniowe przy zapisie plików Big Data do bazy danych



Adam Szałański

69597

02.01.2024

PODSUMOWANIE

CEL RAPORTU

Celem raportu było zbadanie i porównanie efektywności przetwarzania strumieniowego w kontekście zapisywania dużych zbiorów danych (Big Data) do bazy danych. Raport skupiał się na analizie wydajności przetwarzania danych za pomocą tradycyjnego przetwarzania iteracyjnego w porównaniu do operacji strumieniowych.
W ramach raportu przygotowano aplikację odwzorowującą sukces wspominanej w kontekście biznesowym firmy.

WYKORZYSTANE TECHNOLOGIE

Podczas tworzenia raportu utworzono aplikację opartą o język programowania Java z użyciem Spring Boot, a także z wykorzystaniem bibliotek Lombok, Hibernate JPA, jdbcTemplate oraz MapStruct. Do profilowania wydajności aplikacji użyto Java Flight Recorder (JFR), co umożliwiło dokładną analizę zużycia zasobów i wydajności.

PRZEBIEG TWORZENIA PROJEKTU, O KTÓRY OPARTO RAPORT

Projekt tworzony był zaczynając od podstawowej implementacji, inspirując się popularnymi wynikami z wyszukiwarki Google (głównie platforma Baeldung), a następnie rozwijany o rozwiązania stosowane w pracy komercyjnej autora raportu. W ramach logiki aplikacji utworzono dwie implementacje przetwarzania danych z pliku, mapowania tych danych oraz zapisu ich do bazy danych, różniące się wykorzystaniem przetwarzania iteracyjnego i strumieniowego.

KLUCZOWE WNIOSKI

- Wykorzystanie strumieni wiąże się z koniecznością dostosowania logiki z ukierunkowaniem na przetwarzanie strumieniowe. Może to prowadzić zarówno do uproszczenia, jak i utrudnienia implementacji niektórych rozwiązań.
- W trakcie testów wykazano, że zastosowanie strumieni prowadzi do znacznego zmniejszenia zużycia zasobów komputera.
- Przy większych zbiorach danych, przetwarzanie iteracyjne prowadziło do wyczerpania pamięci operacyjnej, co nie miało miejsca przy przetwarzaniu strumieniowym
- Korzyści z przetwarzania strumieniowego wzrastały wraz z rozmiarem danych testowych

ZALECENIA

- Na podstawie analiz zaleca się stosowanie przetwarzania strumieniowego mimo potencjalnie bardziej wymagającej implementacji
- W związku z koniecznością zapisywania w bazie danych nie wczytanych w pełni, zaleca się zapis danych do tabeli tymczasowej, a następnie, po wczytaniu wszystkich danych, przeniesienie ich do tabeli docelowej prostym poleceniem języka SQL (czas trwania takiego polecenia jest niezauważalny)
- Należy pamiętać, aby podczas przetwarzania strumieniowego nie wykonywać akcji wymagających użycia wszystkich elementów strumienia naraz, jak np. zliczanie elementów strumienia celem wyświetlenia ich liczby

WPROWADZENIE

KONTEKST BIZNESOWY

Jeden z klientów firmy Software Mind prowadzi działalność w branży telekomunikacyjnej. W związku z prowadzoną formą działalności, musi obsługiwać on cały ruch swojej sieci komórkowej. W praktyce oznacza to, że obsłużona musi zostać każda, nawet najmniejsza akcja każdego z klientów końcowych (użytkowników numerów telefonów zarejestrowanych w sieci klienta firmy Software Mind). W ciągu każdej minuty do systemów klienta wpadają setki milionów akcji, które mogą oznaczać coś trywialnego jak np. przełączenie między nadajnikiem sieci komórkowej, a także sprawy mniej trywialne, jak np. doładowanie konta karty prepaid, czy próba wykonania połączenia. Akcje te zbierane są w plikach tekstowych, a następnie ładowane do bazy danych przez oprogramowanie dostarczane przez firmę Software Mind. Jednym z najważniejszych wyzwań z jakim Software Mind musiało się zmierzyć było procesowanie plików w sposób jak najbardziej efektywny, ponieważ implementowana logika ma bezpośredni wpływ na jakość użytkowania usług telekomunikacyjnych przez klientów końcowych.

DEFINICJA PROBLEMU

Problemem, z którym firma musiała się zmierzyć, było znalezienie metody na szybkie i efektywne przetwarzanie rosnących zbiorów danych, szczególnie w zakresie zapisywania tych danych do bazy danych. To wyzwanie jest istotne w kontekście aplikacji wymagających bieżącej analizy i interpretacji danych.

TYP ANALITYKI

Raport skupił się na analityce preskryptywnej i deskryptywnej. Analityka preskryptywna została użyta do określenia najlepszych metod przetwarzania danych, podczas gdy analityka deskryptywna pomogła zrozumieć i opisać charakterystykę danych oraz ich przetwarzanie.

UZASADNIENIE WYBORU ROZWIĄZANIA

Wybór przetwarzania strumieniowego przez firmę Software Mind wynikał z potrzeby efektywniejszego zarządzania dużymi zbiorami danych dostarczanych przez klienta z branży telekomunikacyjnej. Przetwarzanie strumieniowe oferuje lepszą wydajność i elastyczność w porównaniu do tradycyjnych metod przetwarzania iteracyjnego, co jest kluczowe w środowiskach Big Data.

WADY I ZALETY

Rozważono zalety przetwarzania strumieniowego, takie jak większa wydajność i niższe zużycie zasobów, oraz wady, w tym złożoność implementacji i potencjalne ograniczenia w zastosowaniach.

METODOLOGIA

OGÓLNY ZARYS

W ramach projektu zastosowano metodologię skoncentrowaną na praktycznym porównaniu dwóch różnych podejść do przetwarzania Big Data: tradycyjnego przetwarzania iteracyjnego oraz przetwarzania strumieniowego. Cel ten osiągnięto poprzez opracowanie, implementację i testowanie dwóch wariantów aplikacji przetwarzającej duże zbiory danych.

PROCES TWORZENIA APLIKACJI

Projektowanie Aplikacji: Projekt został rozpoczęty od zaprojektowania architektury aplikacji, która pozwoliłaby na łatwe porównanie dwóch podejść przetwarzania.

Implementacja: Utworzono dwie główne ścieżki przetwarzania danych: jedną wykorzystującą przetwarzanie iteracyjne oparte o standardowe listy Javy, a drugą korzystającą z przetwarzania strumieniowego. Obie implementacje wykorzystywały wspólne komponenty dla operacji wejścia/wyjścia oraz interakcji z bazą danych, aby zapewnić spójność testów.

TESTOWANIE WYDAJNOŚCI

Zbiory Danych: Testy wydajności zostały przeprowadzone na zbiorach danych o różnych rozmiarach, od 1 miliona do 101 milionów wierszy.

Parametry Pomiarowe: Głównymi mierzonymi parametrami były czas trwania przetwarzania, maksymalne obciążenie procesora oraz zużycie pamięci operacyjnej.

Narzędzia do Testów: Wykorzystano Java Flight Recorder (JFR) do monitorowania wydajności aplikacji i gromadzenia szczegółowych danych na temat zużycia zasobów.

ANALIZA WYNIKÓW

Porównanie Metod: Wyniki z obu metod przetwarzania zostały zebrane, zestawione w tabelach i porównane pod kątem wydajności i efektywności zużycia zasobów komputera.

Interpretacja Danych: Dane zostały przeanalizowane, aby zrozumieć, jak różne podejścia wpływają na wydajność w kontekście różnych rozmiarów zbiorów danych.

OPIS WYBRANYCH TECHNOLOGII

JAVA WRAZ Z FRAMEWORKIEM SPRING BOOT

Wybór Technologii: Java, w połączeniu ze Spring Boot, została wybrana jako podstawa projektu ze względu na jej rozpowszechnienie oraz udowodnioną skuteczność w budowaniu skalowalnych i wydajnych aplikacji.

Zastosowanie: Te technologie zapewniły stabilną i wszechstronną platformę do zarządzania skomplikowanymi operacjami i przepływami danych, umożliwiając efektywną realizację wymaganej logiki aplikacji. Spring Boot szczególnie przyczynił się do uproszczenia konfiguracji oraz integracji różnych komponentów aplikacji.

LOMBOK

Wybór Technologii: Lombok został wykorzystany do uproszczenia struktury kodu aplikacji.

Zastosowanie: Dzięki tej bibliotece możliwe było automatyczne generowanie metod takich jak gettery, settery oraz konstruktory, co znacząco przyspieszyło proces tworzenia aplikacji i zredukowało ilość kodu, zapewniając jednocześnie większą czytelność i łatwość w utrzymaniu.

MAPSTRUCT

Wybór Technologii: MapStruct został wybrany do efektywnego mapowania obiektów DTO (Data Transfer Object) na encje bazodanowe.

Zastosowanie: Biblioteka ta umożliwiła automatyczne i precyzyjne przekształcanie struktur danych, co było kluczowe przy przetwarzaniu i przygotowywaniu danych do zapisu w bazie danych. Dzięki MapStruct, konwersja danych z formatu pliku CSV na format zgodny z modelem bazy danych była szybka i niezawodna.

HIBERNATE JPA

Wybór Technologii: Hibernate JPA został wykorzystany jako główny ORM (Object-Relational Mapping) do interakcji z bazą danych.

Zastosowanie: Hibernate JPA ułatwił zarządzanie encjami bazodanowymi oraz wykonywanie operacji bazodanowych, takich jak wstawianie, odczyt, aktualizacja i usuwanie danych. Jego zastosowanie pozwoliło na łatwiejsze i bardziej efektywne zarządzanie danymi w bazie danych, choć wymagało dodatkowych optymalizacji dla operacji wsadowych.

JDBC TEMPLATE

Wybór Technologii: JdbcTemplate zostało wybrane jako narzędzie do bezpośrednich operacji na bazie danych, aby przeciwdziałać niektórym ograniczeniom Hibernate JPA, zwłaszcza w kontekście przetwarzania dużej ilości danych.

Zastosowanie: JdbcTemplate pozwoliło na bardziej kontrolowane i wydajne operacje bazodanowe. W szczególności było używane do przeprowadzania testów zapisu danych, gdzie jego zdolność do obsługi skomplikowanych zapytań i wsadowego przetwarzania była kluczowa dla osiągnięcia oczekiwanej wydajności.

NAPOTKANE PROBLEMY I ZASTOSOWANE ROZWIĄZANIA

PROBLEM 1: WYDAJNOŚĆ I EFEKTYWNOŚĆ METODY SAVEALL W HIBERNATE JPA

Wykrycie Problemu: Wstępnie zamierzano opierać przetwarzanie danych na metodzie `saveAll` z repozytorium Hibernate JPA. Jednakże, napotkano na kilka kluczowych problemów. W modelu wykorzystującym sekwencje do generowania ID, Hibernate JPA wykonywał najpierw zapytanie `SELECT` w celu pobrania wartości ID z sekwencji, a dopiero potem realizował operację `INSERT` z tym ID. Taki proces znacząco spowalniał operacje zapisu.

Dodatkowe Problemy: Ustalono, iż metoda `saveAll` faktycznie przetwarza każdy rekord indywidualnie, co było wysoce nieefektywne, szczególnie przy obsłudze większych zbiorów danych. Co więcej, `saveAll` nie była kompatybilna z przetwarzaniem strumieniowym, co ograniczało jej użyteczność w tym kontekście.

Rozwiązanie: Jako odpowiedź na te wyzwania, zaimplementowano alternatywne podejście wykorzystujące `jdbcTemplate`. Rozwinięto własną implementację polecenia `INSERT`, umożliwiającą jednoczesny zapis wszystkich danych w jednej operacji. To rozwiązanie znacząco usprawniło proces zapisu danych do bazy, eliminując problemy związane z nadmiernymi zapytaniami do sekwencji ID i zwiększając efektywność przetwarzania większych zbiorów danych.

PROBLEM 2: WYDAJNOŚĆ I ZARZĄDZANIE PAMIĘCIĄ PRZY TWORZENIU ZŁOŻONYCH POLECEŃ INSERT

Wykrycie Problemu: W trakcie implementacji własnego rozwiązania do wstawiania danych do bazy danych za pomocą złożonych poleceń `INSERT`, napotkano na problem z wydajnością i zarządzaniem pamięcią. Mechanizm konstruowania zapytań SQL do wsadowego wstawiania rekordów, szczególnie przy obsłudze znacznych zbiorów danych, prowadził do znacznego zwiększenia zużycia pamięci operacyjnej.

Rozwiązanie: W celu optymalizacji procesu i zarządzania pamięcią wprowadzono limit na liczbę rekordów przetwarzanych w ramach jednego polecenia `INSERT`. Ustanowiono maksymalną liczbę rekordów na poziomie 5000. Po osiągnięciu tego limitu, aplikacja wykonuje skumulowane polecenie `INSERT`, a następnie inicjalizuje proces konstruowania nowego zapytania dla kolejnych rekordów. To podejście pozwoliło na efektywne zarządzanie zasobami i uniknięcie przeciążenia systemu.

PROBLEM 3: ODCZYTYWANIE PLIKÓW CSV BEZ FAWORYZOWANIA METOD PRZETWARZANIA

Wykrycie Problemu: Wstępny plan zakładał użycie biblioteki `commons-csv` z Apache Commons do odczytywania plików CSV. Ta biblioteka oferowała prostotę w odczycie plików zawierających nagłówki i wartości, ułatwiając selekcję danych. Jednakże, `commons-csv` nie wspierała przetwarzania strumieniowego, co oznaczało konieczność

najpierw tworzenia kolekcji, a dopiero później jej strumieniowania. Taki sposób działania mógłby pozbawić przetwarzanie strumieniowe jednej z jego głównych zalet, czyli odporności na błędy związane z brakiem pamięci operacyjnej.

Rozwiązanie: Aby zapewnić równość warunków dla obu testowanych metod przetwarzania, konieczne było opracowanie własnych implementacji metod odczytu danych. Stworzono dwa dedykowane mechanizmy odczytu: jeden dla przetwarzania iteracyjnego i drugi dla przetwarzania strumieniowego. Takie podejście pozwoliło na sprawiedliwe porównanie obu metod, bez wprowadzania sztucznych ograniczeń dla metody strumieniowej.

PROBLEM 4: OKREŚLENIE WIELKOŚCI I GENEROWANIE PLIKÓW TESTOWYCH

Wykrycie Problemu: Kluczowym wyzwaniem było określenie odpowiedniej wielkości plików testowych oraz ich wygenerowanie, aby spełniały kryteria Big Data. Wymagane było utworzenie plików o liczbie wierszy zaczynających się od miliona, aby zapewnić wiarygodność testów.

Rozwiązanie: Rozwiązanie tego problemu zapewnił skrypt napisany w języku Python, który umożliwiał dynamiczne generowanie danych testowych o różnych wielkościach. Skrypt wykorzystywał bibliotekę pandas do tworzenia i zapisywania danych do plików CSV, oferując elastyczność w konfiguracji wielkości i struktury danych.

Dodatkowe Wyzwanie: Dobór Odstępów między wielkościami plików

Jak Napotkano Problem: Początkowe podejście polegające na stosowaniu dużych odstępów między kolejnymi plikami (o 100 milionów rekordów) okazało się zbyt ambitne, prowadząc do problemów z brakiem pamięci, nawet na komputerach z dużą ilością RAM.

Rozwiązanie: Przeprowadzono serię testów z mniejszymi plikami, co umożliwiło dokładniejsze obserwacje różnic w wydajności między przetwarzaniem iteracyjnym a strumieniowym. Ostatecznie, utworzono pliki testowe o zróżnicowanych wielkościach: od 1 miliona do 901 milionów rekordów, co pozwoliło na precyzyjne zbadanie momentu, w którym przetwarzanie strumieniowe zaczyna wyraźnie przeważać nad przetwarzaniem iteracyjnym.

PROBLEM 5: IMPLEMENTACJA I ANALIZA DANYCH Z PROFILERA JFR

Wykrycie Problemu: Ze względu na długotrwałe procesowanie plików, nie było możliwe manualne monitorowanie zużycia zasobów przez różne implementacje procesowania. Wymagane było zastosowanie narzędzia profilującego, które można zintegrować z kodem, aby monitorowanie odbywało się automatycznie i skoncentrowane tylko na testowanym kodzie.

Wybór Narzędzia: Wybrano Java Flight Recorder (JFR) jako narzędzie profilujące, które oferowało wymaganą funkcjonalność oraz jako jedno z niewielu było dostępne za darmo.

Jednakże, jego skomplikowana dokumentacja początkowo utrudniała właściwą konfigurację i integrację z aplikacją.

Rozwiązanie: Po pokonaniu trudności związanych z konfiguracją, udało się zintegrować JFR z testowanym kodem, umożliwiając automatyczne zbieranie danych o maksymalnym obciążeniu procesora i zużyciu pamięci operacyjnej.

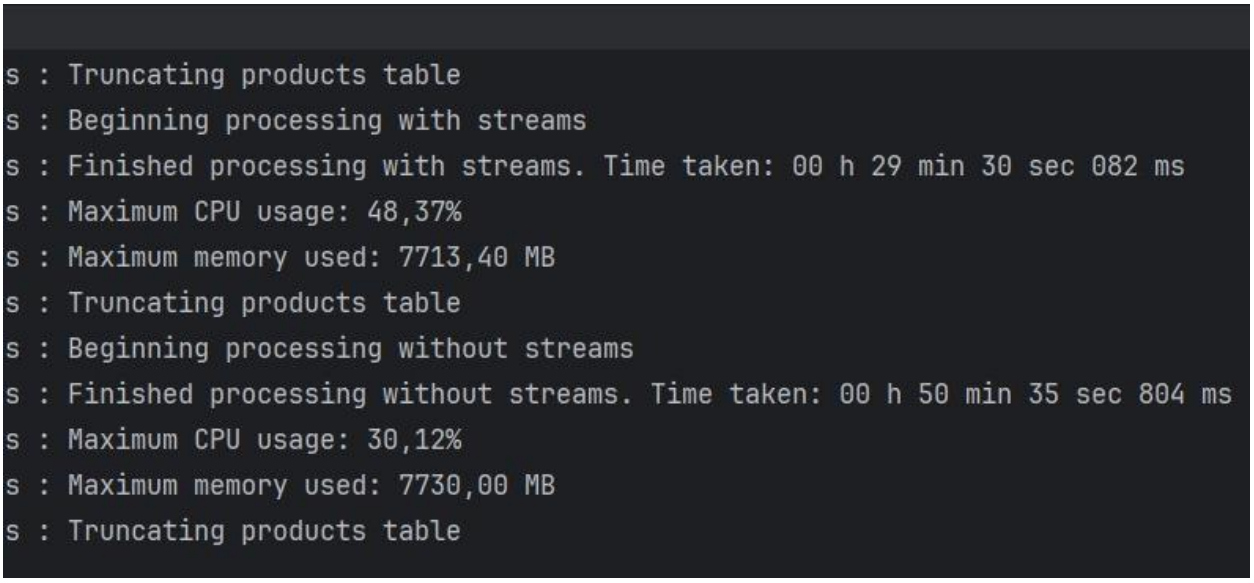
Dodatkowe Wyzwanie: Analiza Danych: Kolejnym wyzwaniem było odczytanie i analiza danych zebranych przez JFR. Dane profilera wymagały najpierw zapisu do pliku, a następnie ręcznego wczytania i analizy. Zaimplementowano mechanizm analizy, który skupiał się na zbieraniu i interpretowaniu maksymalnych wartości obciążenia procesora i zużycia pamięci, co pozwoliło na rzetelne porównanie wydajności różnych metod przetwarzania.

PREZENTACJA DZIAŁANIA TESTÓW APLIKACJI

Działanie testów polega na kilku etapach:

- Wyczyszczenie bazy danych: baza danych jest wstępnie czyszczona z użyciem polecenia truncate, aby rosnąca jej wielkość nie miała negatywnego wpływu na wyniki testów
- Przygotowanie profilera dla przetwarzania strumieniowego: profiler należy przygotować z użyciem odpowiednich parametrów oraz wystartować proces profilowania
- Uruchomienie przetwarzania strumieniowego: uruchomienie zwykłego procesu przetwarzania strumieniowego
- Zatrzymanie profilera i analiza wyników: profiler należy zatrzymać aby nie rejestrował w dalszym ciągu wyników, kiedy test już się zakończył
- Ponowne wyczyszczenie bazy: przygotowanie bazy przed testem z użyciem procesowani iteracyjnego
- Przygotowanie nowego profilera dla przetwarzania iteracyjnego: podobnie jak poprzednio, prfiler należy przygotować odpowiednimi parametrami i uruchomić profilowanie
- Uruchomienie przetwarzania iteracyjnego: uruchomienie zwykłego procesu przetwarzania iteracyjnego
- Zatrzymanie profilera i analiza wyników

Efekty działania można zobaczyć w postaci logów w konsoli aplikacji. Logi te prezentują czas trwania danego etapu testu, maksymalne użycie procesora podczas testu, maksymalne użycie pamięci operacyjnej podczas testu.



Rysunek 1 - prezentacja działania testu w przypadku pliku z 21 mln rekordów

Na powyższym zrzucie ekranu zaprezentowano wyniki podane przez aplikację podczas testowania przetwarzania dla pliku zawierającego 21 mln rekordów.

WYNIKI TESTÓW

Testy zostały przeprowadzone na laptopie Dell 5401 o następującej specyfikacji:

- **CPU**: Intel Core i7 9850H 2.60GHz
- **Ram**: 32GB – 1x16GB Hynix DDR4 2666MHz + 1x16GB G.Skill Ripjaws 2400MHz
- **SSD**: Toshiba 512GB KXG60ZNV512G NvME

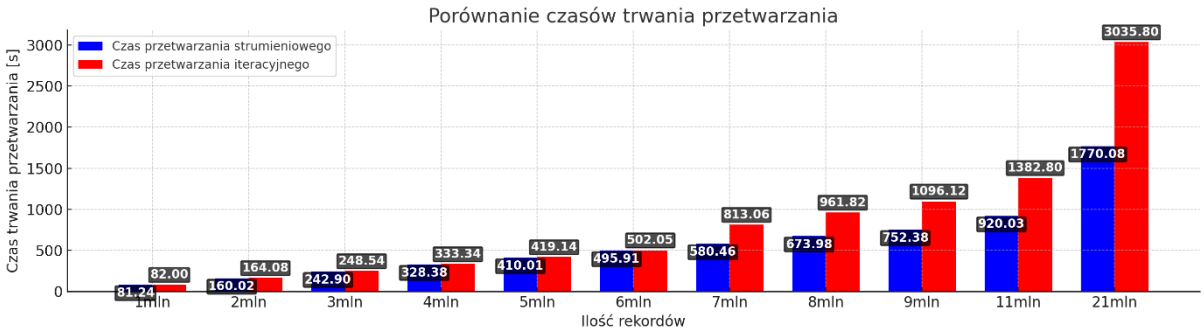
Poniższa tabela prezentuje zbiorczo wyniki uzyskane z testów

Ilość rekordów	Czas trwania przetwarzania strumieniowego [HH:MM:SS.sss]	Czas trwania przetwarzania iteracyjnego [HH:MM:SS.sss]	Maksymalne obciążenie CPU przetwarzania strumieniowego [%]	Maksymalne obciążenie CPU przetwarzania iteracyjnego [%]	Maksymalne użycie pamięci przetwarzania strumienowego [MB]	Maksymalne użycie pamięci przetwarzania iteracyjnego [MB]
1000000	00:01:21.244	00:01:22.001	6,72%	6,86%	843,81	2423,78
2000000	00:02:40.023	00:02:44.084	6,86%	6,63%	2421,61	3244,98
3000000	00:04:02.903	00:04:08.541	6,63%	6,85%	3245,16	4315,75
4000000	00:05:28.379	00:05:33.344	6,85%	6,80%	4315,62	5327,34
5000000	00:06:50.009	00:06:59.137	6,80%	6,86%	5056,26	6164,33
6000000	00:08:15.912	00:08:22.049	6,86%	6,84%	5573,29	6463,51
7000000	00:09:40.464	00:13:33.056	6,84%	22,09%	6205,79	7282,02
8000000	00:11:13.976	00:16:01.823	36,24%	23,67%	7187,01	7575,23
9000000	00:12:32.385	00:18:16.122	48,47%	24,48%	7511,29	7716,69
11000000	00:15:20.033	00:23:02.799	23,11%	25,18%	7716,2	7717,95
21000000	00:29:30.082	00:50:35.804	48,37%	30,12%	7713,4	7730

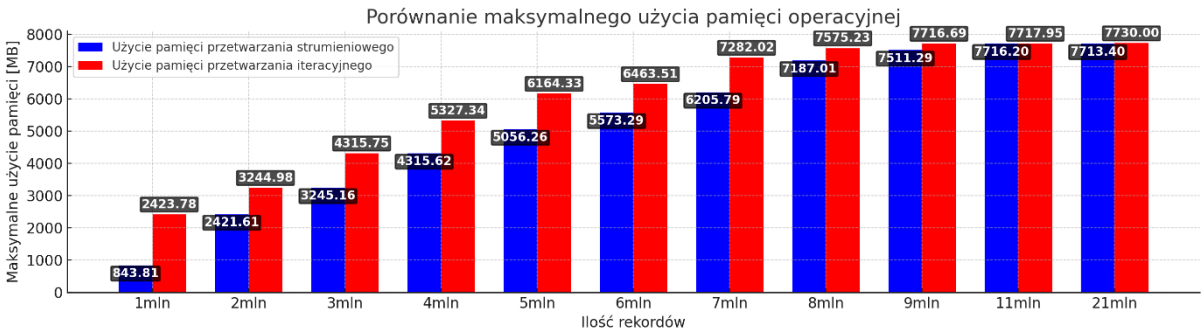
Tabela 1 - prezentacja wyników pomiarów

Ponieważ plik posiadający 21 mln rekordów był ostatnim, jaki udało się przetworzyć przez obie implementacje przetwarzania, wyniki zostały ograniczone do tego pliku. Z plikami większymi poradzić mogło sobie jedynie przetwarzanie strumieniowe.

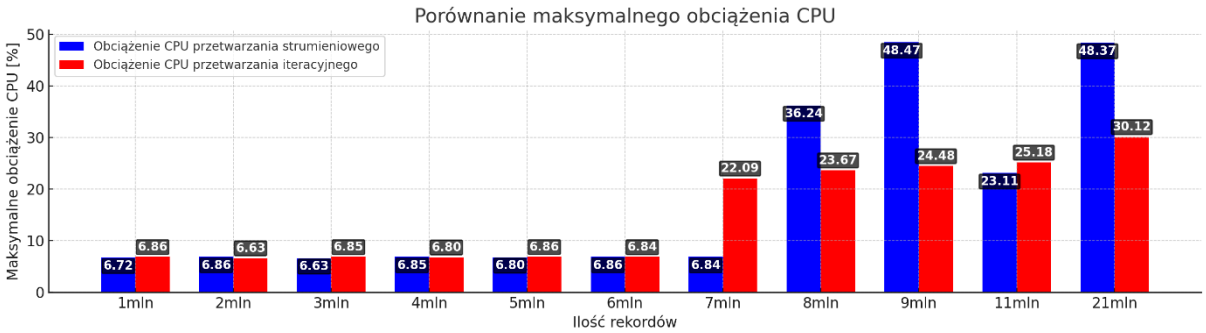
Zwizualizujmy teraz wyniki pomiarów na wykresach:



Rysunek 2 - wykres przedstawiający porównanie czasów trwania procesów przetwarzania pliku



Rysunek 3 - wykres przedstawiający porównanie maksymalnego użycia pamięci operacyjnej podczas przetwarzania pliku



Rysunek 4 - wykres przedstawiający porównanie procentowego maksymalnego obciążenia CPU podczas przetwarzania pliku

Analizując zebrane wyniki można zauważyć, że zarówno w kontekście czasu przetwarzania plików, jak i w kontekście maksymalnego użycia pamięci operacyjnej przetwarzanie strumieniowe miało znaczną przewagę nad przetwarzaniem iteracyjnym. Nie tylko trwało ono krócej, ale także zużywało mniej pamięci operacyjnej.

Możemy także zauważyć, że w przypadku porównania czasów trwania, przewaga procesowania strumieniowego rosła wraz ze wzrostem pliku, co sugeruje, że jest to idealne rozwiązanie dla plików typu big data. W przypadku zużycia pamięci sytuacja ta jest odwrotna i przewaga procesowania strumieniowego maleje wraz ze wzrostem pliku. Patrząc jednak na zużycie pamięci przez przetwarzanie iteracyjne, wykazuje ono podejrzaną tendencję do unikania granicy 8000MB. Warto w przyszłości zweryfikować, czy ten wynik nie został zaburzony przez odgórnie narzucony limit 8GB dostępnej dla aplikacji pamięci i czy w związku z tym reszta pliku nie była wczytywana z użyciem pliku wymiany.

Na koniec, analizując maksymalne obciążenie CPU możemy zauważyć, że początkowo różnica między przetwarzaniem iteracyjnym i przetwarzaniem strumieniowym była nieznaczna, rzędu kilku setnych procenta. Natomiast od pliku wielkości 7 mln rekordów, użycie procesora drastycznie wzrosło. Początkowo tendencję tą wykazało tylko przetwarzanie iteracyjne, jednak już przy kolejnym pliku przetwarzanie strumieniowe także ją wykazało i to w jeszcze większej skali, niż przetwarzanie iteracyjne. Jest to jednak dobry objaw, ponieważ sugeruje, że przetwarzanie strumieniowe potrafiło efektywniej wykorzystać dostępne zasoby komputera.

WNIOSKI

Projekt stworzony na potrzeby raportu w prosty sposób pokazał przewagę przetwarzania z użyciem strumieni nad przetwarzaniem iteracyjnym w kontekście plików o rozmiarach Big Data. Potwierdza to także wspomniany kontekst biznesowy, w którym sukcesem firmy było efektywne zarządzanie zasobami komputera z wykorzystaniem przetwarzania strumieniowego. Zastosowanie przetwarzania strumieniowego nie jest jedynym z czynników decydujących o sukcesie rozwiązania oferowanego przez firmę, lecz jest on jednak czynnikiem niesamowicie ważnym, ponieważ mógłby on upośledzić wszystkie inne rozwiązania zastosowane w produkcie firmy.

Podsumowując zastosowanie przetwarzania strumieniowego możemy śmiało powiedzieć że jest ono efektywniejsze od zwykłego przetwarzania iteracyjnego i pozwala na dużo więcej operacji w porównaniu do standardowego przetwarzania iteracyjnego. Warto jednak zaznaczyć, że wiele bibliotek, które próbowano wykorzystać podczas tego projektu, nie było przygotowanych do współpracy ze strumieniami. Oznacza to, że implementacja przetwarzania strumieniowego często wiąże się z koniecznością własnoręcznej re-implementacji już gotowych rozwiązań.