Ekspertyza

Analiza ograniczeń efektywności działania aplikacji MKP

wykonana na zlecenie Urzędu Komunikacji Elektronicznej (UKE)

ver. 2.0

PIOTR MACIĄG, ERYK WARCHULSKI, WALDEMAR GRABSKI, PIOTR GAWKOWSKI, JAROSŁAW ARABAS

29.10.2021

Instytut Informatyki, Wydział Elektroniki i Technik Informacyjnych, Politechnika Warszawska

2021

Historia wersji dokumentu

Data (dd.mm.rrrr)	Wersja	Uwagi
22.10.2021	1.0	Pierwsza wersja ekspertyzy dostarczona UKE
29.10.2021	2.0	Uszczegółowienie przetwarzania jednoczesnego na wielu instancjach aplikacji

Streszczenie

A. Identyfikacja fragmentów oprogramowania odpowiadających za największe zużycie czasu procesora, pamięci operacyjnej i największą liczbę operacji wejścia/wyjścia:

- Najbardziej czasochłonne jest wykonanie fazy trasowania dla paczek.
- Program przetwarza paczki sekwencyjnie – nie wykorzystuje możliwości zrównoleglenia poprzez jednoczesne trasowanie niezależnych paczek.
- Głównym problemem powodującym długi czas przetwarzania jest intensywne operowanie na tymczasowo tworzonej bazie danych – procesor i pamięć operacyjna są podczas tego procesu używane w marginalnym zakresie. Istotnym więc ograniczeniem jest wydajność systemu dyskowego na rzecz wykorzystania procesora i dostępnej pamięci operacyjnej.
- Co istotne, dla dostarczonych przez UKE danych, maksymalny rozmiar bazy danych to 1,9 GB (przy czym ok. 63% stanowiły indeksy, które okazały się zupełnie nieużywane!) przy jednoczesnym zapotrzebowaniu na pamięć operacyjną na poziomie ok. 6GB.
- Proces trasowania wykonuje się poprzez procedury składowane w tymczasowo utworzonej bazie danych. To one odpowiadają za całkowity czas trasowania.
- Istotna jest niska jakość kodu procedur składowanych, która generuje bardzo dużo operacji wejścia/wyjścia poprzez mechanizmy bazy danych.
- Konfiguracja bazy danych wymaga dostrojenia, ponieważ szereg parametrów nie jest ustawionych na optymalne wartości.

B. Analiza wpływu konfiguracji sprzętowej i softwarowej systemu, na którym jest posadowione oprogramowanie, na wydajność pracy oprogramowania:

- Głównym źródłem problemów jest wydajność pracy serwera bazodanowego ograniczona poprzez jego konfigurację oraz takie a nie inne wykorzystywanie mechanizmów przez procedury składowane, co powoduje bardzo duże obciążenie systemu dyskowego.
- Niezależnie od dostrojenia mechanizmów bazy danych, wysokowydajny system dyskowy będzie kluczowy, dla obecnej implementacji aplikacji, dla osiąganej wydajności przetwarzania.
- Baza danych w wersji 9.5 jest technologicznie przestarzała (nie jest już nawet oficjalnie wspierana) – nie posiada takich możliwości zrównoleglenia przetwarzania jak wersja aktualnie dostępna. Aktualnie wszystkie procedury i funkcje składowane uruchamiane w trakcie działania aplikacji są wykonywane z wykorzystaniem tylko jednego rdzenia procesora.
- System antywirusowy powinien być skonfigurowany tak, aby pomijać pliki serwera bazodanowego, tak aby nie wykonywać zbędnych skanowań plików związanych z jego pracą.

C. Weryfikacja elementów kodu źródłowego, wiążących się z największym zapotrzebowaniem na zasoby, z uwzględnieniem podziału na etapy analizy:

- Dostarczony kod źródłowy dotyczący modułu trasowania w Visual Basic nie dotyczy aktualnej wersji programu, niemniej jednak zidentyfikowano w nim szereg niepotrzebnych operacji na bazie danych (np. odczyty z tabel, których wyniki nie zostają w żaden sposób wykorzystywane).
- Program MKP podczas trasowania przetwarza paczki sekwencyjnie pomimo tego, że ich przetwarzanie nawet w obecnej wersji programu może odbywać się zupełnie niezależnie.
- Kod procedur składowanych zawiera wiele konstrukcji powodujących istotne problemy wydajnościowe – przykładem tego są zakładane i kasowane w każdej iteracji pętli tabele tymczasowe, które mogłyby być zupełnie wyeliminowane, a także nieefektywnie wykorzystywane mechanizmy indeksowania.

D. Rekomendacje dotyczące sposobu poprawy efektywności, w szczególności z uwzględnieniem wykorzystania możliwości sprzętu realizującego równoległe obliczenia:

- Biorąc pod uwagę zaobserwowane wykorzystanie pamięci operacyjnej oraz przestrzeni dyskowej przez roboczą bazę danych, najistotniejszą modyfikacją byłoby całkowite przeniesienie przetwarzania do pamięci operacyjnej i rezygnacja z tymczasowo tworzonej bazy danych. Wymagać to będzie jednak dużej inwestycji czasu i zasobów w związku z koniecznością odtworzenia całej funkcjonalności trasowania.
- W analizowanych danych nie napotkano na konflikty, które uniemożliwiłyby zrównoleglenia przetwarzania poszczególnych paczek – w sposób naturalny jest to więc droga na osiągnięcie istotnego przyspieszenia. Nawet w obecnej wersji programu możliwe jest jednoczesne uruchomienie jego wielu instancji i, rozdzielając odpowiednio paczki pomiędzy te instancje, przyspieszone w ten sposób wykonanie trasowania. Niestety, wąskim gardłem pozostaje wówczas system bazodanowy i wydajność systemu dyskowego na którym operuje.
- Aktualnie aplikacja MKP działa w oparciu o serwer bazy danych PostgreSQL 9.5, który nie wykorzystuje mechanizmów współbieżności oraz zrównoleglania operacji obliczeniowych. W związku z tym większość operacji przeprowadzanych przez bazę danych wykonywana jest z pełnym wykorzystaniem tylko jednego rdzenia procesora. Rekomendowane jest przejście na serwer PostgresSQL w wersji co najmniej 12 ze względu na możliwość wykorzystania mechanizmów wielowątkowego wykonywania operacji bazy danych, a także wydajniejsze wykonywanie zapytań i lepsze zarządzanie pamięcią dyskową. Niestety, obecna wersja programu MKP jest niekompatybilna z najnowszą aktualnie dostępną wersję serwera PostgresSQL, co wymaga dalszej analizy problemu.
- Pozostając przy obecnej wersji serwera bazy danych i programu MKP, możliwe jest wykonanie dostrojenia konfiguracji sprzętowej stacji roboczej, systemu operacyjnego a także serwera bazodanowego jako takiego. Wymagane jest jednak przeprowadzenie dalszych prac weryfikujących, czy zidentyfikowane i zastosowane w ramach tej ekspertyzy modyfikacje wpływają pozytywnie w sposób uniwersalny również dla innych zestawów danych, z którymi pracuje UKE.
- Na poziomie obecnej wersji procedur składowanych możliwe jest wykonanie ich refaktoryzacji i optymalizacji użycia tymczasowej bazy wykorzystywanej do trasowania – przede wszystkim poprzez pozbycie się zbędnych tabel tymczasowych, nieużywanych indeksów itd.

1 Cele ekspertyzy

Celem ekspertyzy jest zidentyfikowanie nieefektywnych elementów aplikacji MKP. W toku przeprowadzonych spotkań z przedstawicielami Urzędu Komunikacji Elektronicznej (UKE) ustalono, że najbardziej czasochłonnym elementem pracy z aplikacją jest tzw. proces *trasowania*. W związku z czym, analiza skupiła się głównie na odpowiedzialnym za ten proces module trasowania.

Główne cele ekspertyzy są następujące:

a. identyfikacja fragmentów oprogramowania odpowiadających za największe zużycie czasu procesora, pamięci operacyjnej i największą liczbę operacji wejścia/wyjścia,
b. analiza wpływu konfiguracji sprzętowej i softwarowej systemu, na którym jest posadowione oprogramowanie na wydajność pracy oprogramowania,
c. weryfikacja elementów kodu źródłowego wiążących się z największym zapotrzebowaniem na zasoby z uwzględnieniem podziału na etapy analizy,
d. opracowanie rekomendacji dotyczących sposobu poprawy efektywności, w szczególności z uwzględnieniem wykorzystania możliwości sprzętu realizującego równoległe obliczenia.

Przeprowadzenie ekspertyzy zostało poprzedzone kilkoma spotkaniami z przedstawicielami UKE, podczas których przedstawili oni szczegółowo typowy scenariusz użycia aplikacji MKP. Zamawiający dostarczył również:

- prekonfigurowaną maszynę, której szczegółowy opis zawarto w rozdziale 3,
- dokumentację użytkową aplikacji w wersji 4.15,
- pliki wykonywalne aplikacji MKP w wersji 4.15,
- kopię zapasową mkp_v415.backup oraz mkp_uke_netmodeller_v415.backup potrzebne do odtworzenia baz *mkp* oraz *mkp_uke_netmodeller* wymaganych do działania aplikacji,
- plik z kodem źródłowym w Visual Basic stanowiący interfejs odczytywania rezultatów trasowania z bazy danych przez aplikację MKP.
- Pliki testowe z danymi wejściowymi (tzw. *paczek*) aplikacji dla województwa:
    - lubuskiego,
    - mazowieckiego.

2 Charakterystyka aplikacji MKP

Na podstawie przeprowadzonych spotkań z przedstawicielami UKE oraz dostarczonej dokumentacji użytkowej aplikacji MKP ustalono, że składa się ona z szeregu modułów systemowych, których zadaniem jest, na podstawie zadanych parametrów wejściowych, przeanalizować i wyznaczyć obszary konkursowe dla sieci teleinformatycznych. W tym celu opracowane i zaimplementowane zostały następujące moduły: ładowania danych, wyznaczania modelu kosztowego sieci, wyznaczania modelu popytowego sieci, trasowania sieci oraz agregacji wyników z otrzymanych modeli.

Ze względu na architekturę systemową, aplikacja MKP składa się z następujących głównych części:

- aplikacji okienkowej MKP przygotowanej dla systemu Windows i zaimplementowanej (wedle przekazanej przez UKE informacji) w języku programowania Visual Basic. Do głównych zadań tej aplikacji należy umożliwienie użytkownikowi ładowania danych wejściowych oraz konfiguracja parametrów modułów, a także informowanie o przebiegu obliczeń oraz zwrócenie rezultatów wyjściowych.
- serwer bazy danych PostgreSQL w wersji 9.5. W trakcie działania aplikacji wykorzystywane są dwie instancje bazy danych o nazwach: *mkp* oraz *mkp_uke_netmodeller*. Możliwe jest odtworzenie struktur danych tych instancji oraz kodów źródłowych funkcji składowanych za pomocą plików *mkp_db_v415.sql* oraz *mkp-uke-netmodeller-db_v415.sql.* W szczególności baza danych *mkp_uke_netmodeller* składa się z szeregu schematów (jak np. ładowania i przetwarzania danych). Zdecydowanie największy wpływ na czas wykonania aplikacji mają uruchomienia funkcji i procedur schematu *nr*, który odpowiada za mechanizmy trasowania.
- tymczasowych instancji baz danych. W trakcie uruchomienia modułu trasowania aplikacji, dla każdej załadowanej paczki danych tworzona jest nowa instancja bazy danych, której nazwa odpowiada nazwie paczki danych. Struktura takiej instancji jest taka sama jak struktura instancji *mkp_uke_netmodeller.*

3 Środowisko testowe

Eksperymenty badające wydajność aplikacji MKP zostały przeprowadzone na dostarczonej przez Urząd Komunikacji Elektronicznej (UKE) maszynie testowej. Konfiguracja sprzętowa i programowa dostarczonej maszyny testowej w momencie jej dostarczenia była następująca:

- procesor Intel Xeon E-2136,
- 64 GB pamięci RAM,
- trzy dyski twarde (o oznaczeniach *C, D, E*), z których:
    - dysk *C* jest typu SSD (*Solid-State Drive*, model SKhynix SC401 SATA 512 GB),
    - natomiast dyski *D* oraz *E* to dyski mechaniczne (HDD - *Hard Disk Drive* – dyski mechaniczne, oba to modele ST1000LM049-2GH172),
- procesor graficzny Radeon Pro WX 4100,
- system Windows 10 Pro (wersja systemu 20H2),
- program antywirusowy McAfee w wersji 5.5.1.388,
- baza danych PostgreSQL 11 posadowiona na dysku C.

Z punktu widzenia przeprowadzonych analiz istotne jest, że wymieniony powyżej program antywirusowy pozostawał uruchomiony w trakcie wykonywania eksperymentów. Ponadto, w dostarczonej maszynie pamięć wirtualna została ustawiona na 9782 MB. Wszystkie testy zostały przeprowadzone dla domyślnie ustawionych parametrów aplikacji MKP.

4 Wstępne przygotowanie środowiska

Z przeprowadzonych spotkań z przedstawicielami UKE oraz z dostarczonej dokumentacji użytkowej aplikacji MKP wynikało jasno, iż dostarczona stacja robocza wymagała dodatkowych kroków przygotowawczych.

W celu uruchomienia aplikacji MKP został odinstalowany serwer PostgreSQL 11 początkowo zainstalowany na dostarczonej maszynie. Serwer ten został zastąpiony serwerem bazy danych w wersji PostgreSQL 9.5, który jest wymagany do poprawnego działania aplikacji MKP. Serwer PostgreSQL 9.5 został początkowo zainstalowany na dysku D (dysk HDD), a po przeprowadzeniu wstępnych eksperymentów (opisanych w Sekcji 7) został przeniesiony na dysk C (dysk SSD).

Przeprowadzona instalacja serwera bazy danych spowodowała konieczność odtworzenia prawidłowej konfiguracji środowiska programowego oraz wstępnych ustawień pozwalających na obserwację i analizę pracy aplikacji:

- W celu uruchomienia aplikacji, w pliku konfiguracyjnym *postgresql.conf* serwera bazy danych PostgreSQL 9.5 wartość parametru *max_locks_per_transactions* została ustawiona na 64000 (dla domyślnej wartości tego parametru w trakcie wykonywania aplikacji MKP pojawiał się błąd czasu wykonania). Wartość 64000 tego parametru została zasugerowana w przez administratora aplikacji MKP.
- Do każdego z testowanych plików konfiguracyjnych zostały dodane polecenia uruchomienia logowania i monitoringu działania bazy danych.
- W przypadku pozostałych parametrów pliku konfiguracyjnego *postgresql.conf* , testy działania aplikacji MKP zostały przeprowadzone zarówno dla standardowo ustawionych wartości parametrów tego pliku, jak i dla zoptymalizowanych wartości (co opisano w sekcji 8).

5 Wykorzystane narzędzia

W celu przeanalizowania istniejących problemów wydajnościowych aplikacji MKP zostały wykorzystane następujące narzędzia systemowe:

- Monitor zasobów systemu Windows do zebrania szeregu parametrów systemu operacyjnego w trakcie uruchomienia aplikacji MKP. W szczególności pozyskane zostały dane o czasie zużycia procesora, pamięci operacyjnej RAM oraz operacjach dyskowych.
- Monitor zadań systemu Windows do analizy wykorzystania zasobów maszyny testowej w czasie działania aplikacji.

Analizę i monitorowanie pracy serwera bazodanowego wykonano w oparciu o:

- Dane agregowane w widokach *pg_stat_statements*[1], *pg_stat_user_tables*[2], *pg_stat_user_indexes*[2], *pg_stat_database*[2] , które pozawalają na uzyskanie szeregu parametrów dotyczących zapytań wydanych przez aplikację MKP, struktury tabel, indeksów czy informacji dotyczących samej bazy danych. Dzięki widokowi *pg_stat_statements* możliwe jest uzyskanie informacji o takich parametrach zapytań, jak:
    - liczba wywołań danego zapytania,
    - całkowity czas wykonania danego zapytania,
    - całkowita liczba wierszy danych zwróconych przez zapytanie,
    - całkowita liczba odczytów i zapisów bloków współdzielonych bazy danych.

  W widokach *pg_stat_user_tables*, *pg_stat_user_indexes* zawarte są informacje dotyczące wykorzystywanych tabel w ramach danej bazy danych oraz dotyczące nałożonych na kolumny indeksów. Widok *pg_stat_database* agreguje informacje dotyczące bazy danych, tj. m.in. liczbę utworzonych plików tymczasowych, czy liczbę zapisanych oraz przeczytanych bloków danych z dysku.
- Program *pgbench*, który pozwala przeprowadzać testy wydajnościowe na serwerze bazodanowym PostgreSQL. Na podstawie wyników testów program określa średnie opóźnienie oraz średnią liczbę transakcji na sekundę.
- Otwarte oprogramowanie *pgBadger*[3] analizujące i wizualizujące logi operacji bazy danych w trakcie uruchomienia aplikacji MKP. Narzędzie *pgBadger* zostało wykorzystane do analizy następujących parametrów bazy danych:
    - Wizualizacji szeregu czasowego zapytań do bazy danych w trakcie uruchomienia aplikacji MKP.
    - Analizy liczby nawiązywanych połączeń z bazą danych przez aplikację MKP.
    - Analizy liczby zakleszczeń transakcji bazy danych.
    - Wizualizacji w postaci histogramów oraz szeregów czasowych parametrów zapytań do bazy danych, takich jak: najwolniejsze zapytania, najczęściej wykonywane zapytania, najdłużej wykonywane zapytania ze względu na znormalizowany czas wykonania.

6 Scenariusze testowe

W początkowym scenariuszu baza danych została zainstalowana na dysku twardym HDD (dysk *D*), jednakże, ze względu na długie czasy wykonywania trasowania, baza danych została następnie przeniesiona na dysk SSD (dysk *C*).

[1] https://www.postgresql.org/docs/9.4/pgstatstatements.html

[2] https://www.postgresql.org/docs/9.3/monitoring-stats.html

[3] https://github.com/darold/pgbadger

Dla domyślnych parametrów aplikacji MKP nie było możliwe wykonanie trasowania dla wszystkich załadowanych paczek dla woj. lubuskiego. W szczególności, podczas wykonywania obliczeń dla paczki 10861 występował komunikat o błędzie alokacji pamięci[4]. W związku z tym w przypadku woj. lubuskiego zostały zaprezentowane eksperymenty z pominięciem paczki 10861.

Ze względu na znaczne czasy wykonywania aplikacji w przypadku woj. mazowieckiego zostały zaprezentowane eksperymenty dla paczki nr 11403, która zawiera największą ilość danych.

Podsumowując, przedstawiony raport został przygotowany w oparciu o rezultaty eksperymentów modułu trasowania aplikacji MKP dla następujących scenariuszy:

1. wszystkich poza 10861 paczek danych dla woj. lubuskiego dla standardowej konfiguracji pliku *postgres.conf*.
2. wszystkich poza 10861 paczek danych dla woj. lubuskiego dla bazy danych PostgreSQL 9.5 dla zoptymalizowanych parametrów pliku *postgres.conf* .
3. paczki 11403 dla woj. mazowieckiego dla bazy danych PostgreSQL 9.5 dla standardowej konfiguracji pliku *postgres.conf.*
4. paczki 11403 dla woj. mazowieckiego dla bazy danych PostgreSQL 9.5 dla zoptymalizowanych parametrów pliku *postgres.conf.*

7 Wyniki analizy

Wstępne eksperymenty z aplikacją MKP dla różnego rozmiaru danych wejściowych pokazały, że serwer bazy danych w znacznym stopniu wykorzystuje operacje odczytu i zapisu danych na dysk. Dotyczy to w szczególności bazy danych *mkp_uke_netmodeller*, która zawiera procedury i funkcje składowane modułu trasowania. Dlatego też w kolejnych krokach analizy skoncentrowano się przede wszystkim na zbadaniu wydajności oraz mechanizmów działania bazy *mkp_uke_netmodeller.* Poniżej przedstawiamy wyniki przeprowadzonej analizy w odniesieniu do czterech celów wskazanych w rozdziale 1 i określonych w umowie na wykonanie ekspertyzy.

7.1 Identyfikacja wąskich gardeł z poziomu systemu operacyjnego

W celu zidentyfikowania fragmentów oprogramowania odpowiadających za największe zużycie zasobów maszyny testowej zebrano oraz przeanalizowano profil obciążenia systemu podczas realizacji scenariuszy testowych przedstawionych w rozdziale 6.

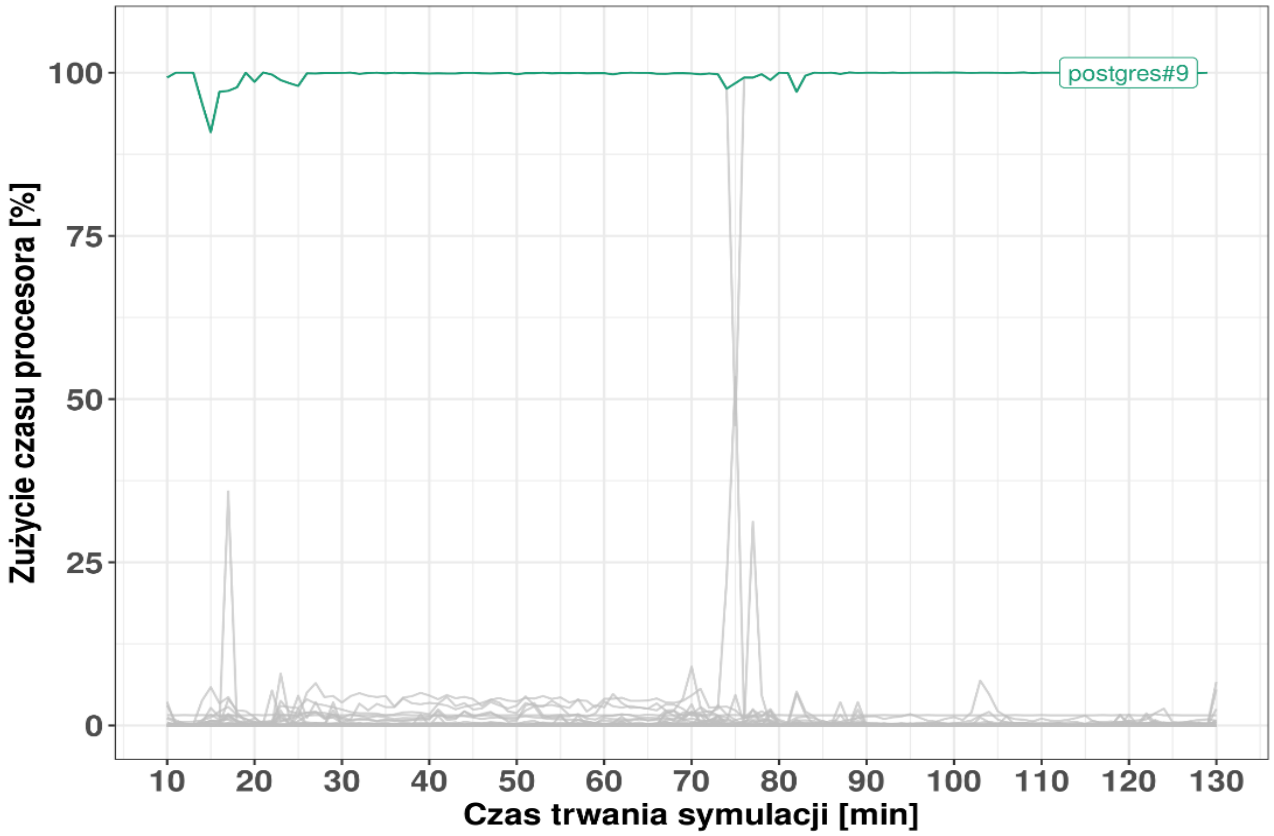
7.1.1 Czas procesora

W celu identyfikacji procesów, które w największym stopniu zużywają zasoby procesora, przeanalizowano logi zużycia procesorów logicznych. Na podstawie uzyskanych wyników stwierdzono, że procesem, który w największym stopniu zużywał czas jednego z procesorów logicznych był proces serwera bazodanowego PostgreSQL, tj. proces o identyfikatorze *postgres#9*. W procesie tym wykonywana była aplikacja MKP. Pozostałe procesy w niewielkim

[4] Dla paczki 10861 aplikacja MKP zwraca błąd *Error 53200 out of shared memory*.

stopniu zajmowały czas pozostałych procesorów logicznych (sumaryczne obciążenie całego procesora dochodziło do maksymalnie 25÷30%).

Na wykresie z Rysunek 1 przedstawione są przebiegi zużycia czasu procesora (poszczególnych rdzeni logicznych) przez wszystkie aktywne procesy w trakcie trwania symulacji dla paczki o ID 11403. Dla pozostałych eksperymentów, które obejmowały inne paczki, występował tożsamy efekt.

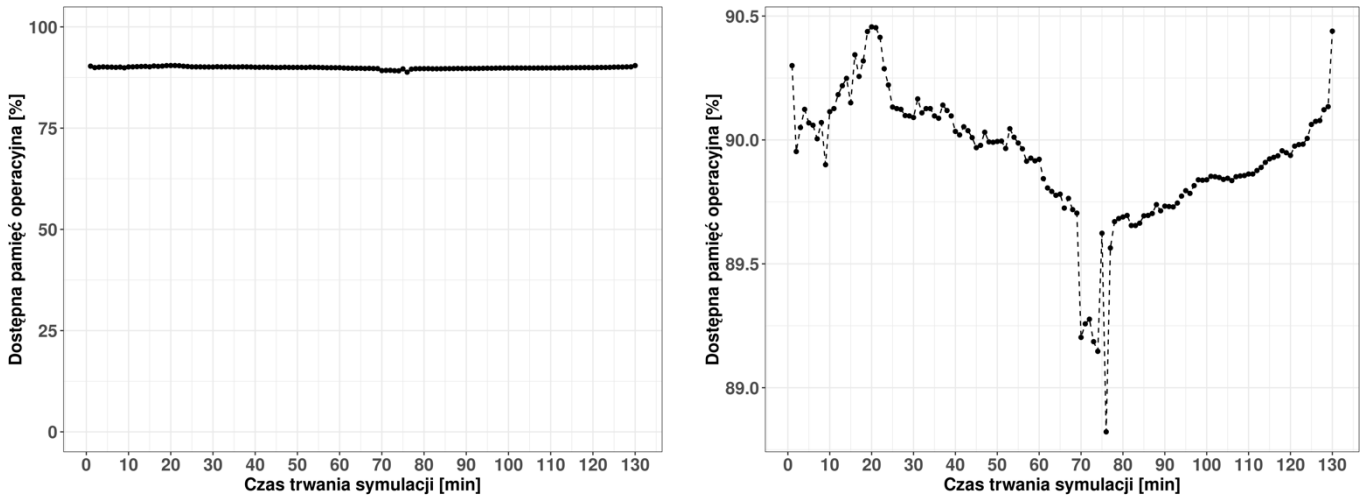


Rysunek 1 Zużycie czasu procesora przez aktywne procesy systemowe w trakcie trwania scenariusza testowego.

Podsumowując, aplikacja MKP przetwarza paczki sekwencyjnie i bez wykorzystania wielordzeniowowści (bo chociaż wykorzystuje ją do tego serwer bazy danych działający na wielu procesach i wątkach, to nie ma to odzwierciedlenia w obciążeniu poszczególnych rdzeni procesora – wykorzystywany jest de facto jeden).

7.1.2 Pamięć operacyjna

W ramach przeprowadzonych eksperymentów nie stwierdzono, aby pamięć operacyjna maszyny testowej została zajęta w znaczącym stopniu niezależnie od rozmiaru paczki czy liczby przetwarzanych paczek. Rysunek 2 przedstawia typowy poziom dostępnej pamięci operacyjnej (patrz konfiguracja sprzętowa stacji roboczej w rozdziale 3) podczas realizacji scenariuszy testowych. Przy standardowej konfiguracji bazy danych pamięć operacyjna w momencie rozpoczęcia eksperymentów była zajęta w 12.3%. Wartość ta utrzymywała się przez cały czas trwania eksperymentu. Większe zajęcie pamięci operacyjnej, tj. 15.5%, zauważono tylko w przypadku zmodyfikowanej konfiguracji serwera bazodanowego PostgreSQL. Jednakże wzrost zajętości pamięci był efektem zwiększenia wartości parametrów konfiguracyjnych, które bezpośrednio odpowiadają za zużycie pamięci operacyjnej, tj. *shared_buffers*, *temp_buffers* lub *work_mem*, co jest efektem zrozumiałym i pożądanym, bo ogranicza obciążenie systemu dyskowego.



Rysunek 2 Dostępna pamięć RAM w trakcie trwania symulacji.

Poczynione obserwacje świadczą o nikłym bezpośrednim wykorzystaniu pamięci operacyjnej przez aplikację MKP. **Niestety, w wykorzystywanej przez UKE konfiguracji serwera bazodanowego oraz przy obecnej implementacji przetwarzania, również on w sposób marginalny wykorzystuje pamięć operacyjną. Porównując czasy dostępu do pamięci operacyjnej i podsystemu dyskowego** (rozdział 7.1.3) **jasnym jest, że jest to rozwiązanie skrajnie nieefektywne**.

7.1.3 Operacje wejścia-wyjścia

W ramach przeprowadzonych eksperymentów zbadano intensywność stosowania operacji wejścia-wyjścia przez aktywne procesy. Zauważono, że przez większość czasu trwania symulacji liczba operacji wejścia-wyjścia przeprowadzana przez proces serwera bazodanowego PostgreSQL, w którym wykonywała się aplikacja MKP, była wielokrotnie większa od liczby takich operacji przeprowadzanych przez pozostałe aktywne procesy. Rysunek 3 przedstawia liczbę operacji wejścia-wyjścia na sekundę realizowanych przez aktywne procesy w trakcie trwania symulacji dla paczki o ID 14403 w konfiguracji dla dysku SSD.

Należy jednak dodać, że przez pierwsze 80 minut trwania symulacji skala wykonywanych operacji wejścia-wyjścia przez proces MKP była zbliżona do pozostałych procesów. Od 80 minuty trwania symulacji liczba operacji wejścia-wyjścia diametralnie zwiększyła się względem reszty -- średnio wykonywane było sto razy więcej operacji wejścia-wyjścia na sekundę. W dalszych analizach ustalono, że odpowiada za to wywołanie funkcji *f_rou_nodes_rank* bazy danych – o czym szerzej w rozdziale 7.3.

Rysunek 3 Liczba operacji wejścia-wyjścia na sekundę przez aktywne procesy systemowe w trakcie trwania symulacji.

Eksperymenty przeprowadzone na konfiguracji serwera bazy danych działającego na dysku mechanicznym ewidentnie wskazały na problemy z wydajnością operacji wejścia/wyjścia i były tożsame z obserwacjami ukazanymi na wykresie wyżej. W przypadku HDD efekt ten był jeszcze bardziej zauważalny z uwagi na różnice wydajności tradycyjnego HDD w zestawieniu z wydajnością typowego dysku SSD.

7.2 Analiza wpływu konfiguracji sprzętowej i softwarowej systemu

W przypadku konfiguracji, w której baza danych operowała na dysku mechanicznym, czas przetworzenia (trasowania) dla wszystkich paczek woj. lubuskiego trwało 3 godziny i 51minut. W przypadku konfiguracji bazy danych na dysku SSD, czas ten skrócił się do 2 godzin i 23 minut. Widać więc wyraźnie, że **wydajność systemu plików ma kluczowe znaczenie dla czasu przetwarzania**. Na potrzeby tej ekspertyzy uznano więc za bezsensowne prezentowanie wyników innych analiz przeprowadzonych z użyciem dysku mechanicznego (badania przeprowadzono, ale ich wyniki nie wnoszą nic niespodziewanego do wniosków końcowych).

W celu wstępnego sprawdzenia wydajności serwera bazy danych przeprowadzono serię eksperymentów przy użyciu narzędzia *pg_bench*. Program ten wykonuje ciąg zapytań do specjalnie utworzonej na rzecz testów bazy danych, a jednocześnie dokonuje pomiaru liczby wykonanych transakcji na sekundę. Ponadto na podstawie liczby transakcji trwających dłużej niż ustalony odgórnie limit wyznaczane jest średnie opóźnienie trwania transakcji.

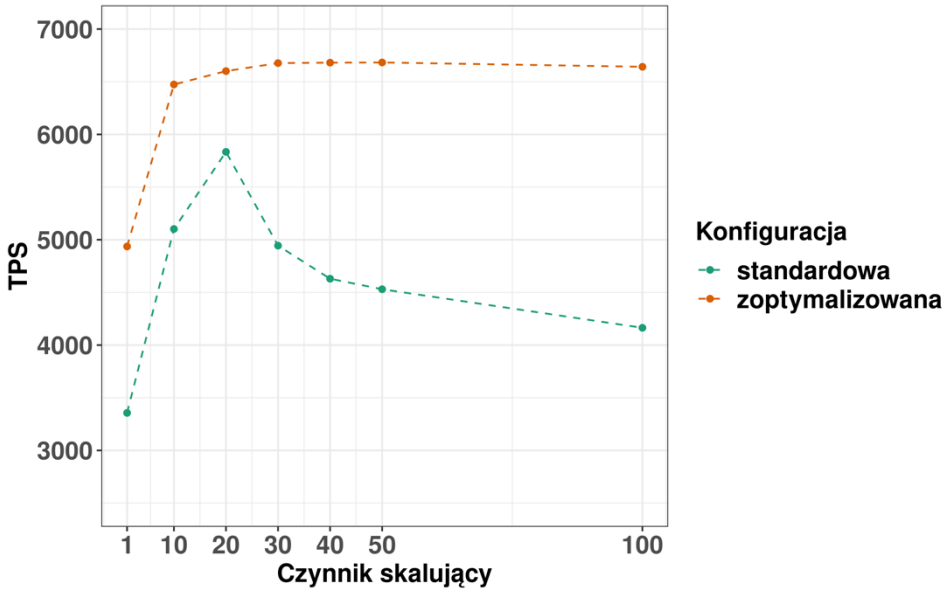
Wydajność serwera została przebadana dla dwóch konfiguracji:

- standardowej,
- zoptymalizowanej.

Konfiguracja standardowa jest równoważna konfiguracji używaną przez UKE i poza wartością parametru *max_locks_per_transaction* nie różni się od konfiguracji generowanej domyślnie przez serwer PostgreSQL.

Konfiguracja zoptymalizowana jest konfiguracją zmodyfikowaną przez Wykonawcę ekspertyzy, w której kluczowe parametry dotyczące szybkości działania serwera bazodanowego zostały dostosowane do konfiguracji sprzętowej używanego środowiska testowego. Ich uniwersalność dla innego środowiska pracy i zestawu przetwarzanych danych wymaga dalszych prac badawczych.

Wyniki eksperymentów znajdują się na Rysunek 4, na którym przedstawione jest liczba transakcji na sekundę w funkcji rozmiaru testowanej bazy danych (przyjęto czynniki skalujący, dla którego wartość 1 oznacza liczbę wierszy w tabelach od 1 do 100000). Na rzecz eksperymentów przyjęto stałą liczbę połączeń do bazy danych równą 10.



Rysunek 4 Liczba transakcji na sekundę dla konfiguracji serwera PostgreSQL.

Na podstawie otrzymanych wyników można zauważyć znaczący spadek średniego tempa transakcji wraz z przyrostem rozmiaru bazy danych w konfiguracji standardowej. Przyrost rozmiaru bazy danych między wartością czynnika skalującego równą 20 a 30 wynosi ok. 152 MB, co przekłada się na ponad 15% spadek średniego tempa transakcji. **Uzyskane wyniki sugerują, że standardowa konfiguracja serwera bazodanowego może nie być wystarczająca w przypadku większych rozmiarów bazy danych**. Jednakże należy wziąć pod uwagę, że średnie tempo transakcji nie jest jedyną wielkością, która odpowiada za szybkość działania aplikacji bazodanowych i utrzymanie wysokich wartości nie musi się przekładać jednoznacznie na poprawę szybkości.

Przeanalizowano również strukturę wykorzystywanych relacji przez aplikację MKP. W szczególności zwrócono uwagę na stosowane indeksy oraz próbowano wykryć relacje o dużym rozmiarze, które często były poddawane skanowi sekwencyjnemu w ramach realizowanych zapytań i jednocześnie nie posiadały nałożonych indeksów. Należy zaznaczyć, że stałe skanowanie sekwencyjne takich tabel może być przyczyną znacznego spadku wydajności.

Poniżej zostały przedstawione schematy tabel bazy danych, cechujące się dużą liczbą skanowań sekwencyjnych. Otrzymane wyniki zostały osiągnięte w ramach trasowania paczki 11403 dla woj. mazowieckiego przy standardowej konfiguracji bazy danych.

Tabela 1 Sposób wykonywania skanów przez serwer bazodanowy.

Schemat	Tabela	Liczba skanów sekw.	Liczba czytanych krotek sekw.	Liczba skanów ind.
nr	pa_zas	162402	3547790400	0
dict	edge_types	6595241	1866450671	709657

W przeprowadzonym eksperymencie zapytanie *nr.f_pa_zas_rap_split()* korzystające z tabeli *nr.pas_zas* okazało się być trzecim najwolniejszym zapytaniem. Tak znaczna liczba skanowań sekwencyjnych, a tym samym liczba wczytanych w ten sposób wierszy danych sugeruje, że założenie odpowiednich indeksów dla tabeli *nr.pas_zas* może przyśpieszyć wykonanie aplikacji. W przypadku relacji *dict.edge_types* liczba skanowań bazujących na indeksach jest niezerowa, ale nadal stanowi tylko 9% wszystkich wykonanych skanowań. Odwołania do tej tabeli również występują w jednych z najwolniej wykonywanych zapytań, tj. w *nr.f_initial_merge_edges_part()* czy *nr.f_essential_create_one_side_trenches()***.**

Z drugiej strony, niepotrzebne indeksy mogą być równie istotną przyczyną problemów z wydajnością działania aplikacji. Z tego względu przeanalizowano schematy bazy danych ze względu na nieefektywnie wykorzystywane indeksy. **Poniższa tabela zawiera przykłady schematów, w których indeksy nie były wykorzystywane lub były wykorzystywane rzadko przez zapytania do bazy danych, a zajmowały niepomijalny rozmiar.**

Tabela 2 Użycie indeksów w schematach i ich zajętość w bazie danych.

Schemat	Relacja	Liczba skanów ind.	Łączy rozmiar ind.
dict	Splitter_types	0	144 MB
stg	Road_vertices	8	137 MB
dict	Process_cfg	0	128 MB
nr	Rou_node_networks_paths	4	109 MB

W ramach przeprowadzonej analizy na maszynie testowej uruchomiono także wiele instancji aplikacji MKP, które łączyły się z tą samą bazą danych, ale przetwarzały inne paczki danych. Uzyskano w ten sposób pewną możliwość zrównoleglania działania aplikacji MKP ze względu na uruchomienie modułów obliczeniowych każdej z nich w osobnym procesie serwera PostgreSQL. Obserwacja monitora aktywności systemu Windows pokazała, że w takim przypadku istnieje możliwość wielowątkowego przeprowadzenia obliczeń (np. dla działania każdej z uruchomionych aplikacji i tworzonych dla nich procesów PostgreSQL przypisywany był inny rdzeń procesora). **Co ważne, rezultaty otrzymane dla każdej z aplikacji MKP z osobna nie są usuwane w przypadku uruchomienia kolejnej instancji aplikacji.**

Dzieje się tak, ponieważ dla każdej paczki tworzona jest osobna instancja tymczasowej bazy danych na potrzeby jej przetwarzania. Bazy takie dla poszczególnych paczek tworzone są niezależnie od siebie i posiadają unikalną nazwę (odpowiadającą identyfikatorowi paczki). Bazy te pozostają na serwerze bazodanowym do momentu załadowania danych poprzez przyciski *zaladuj informacje o paczkach* oraz *zaladuj pa do modułu popytowego*. Przy pierwszym z tych przycisków dostępna jest opcja (typowo załączona) powodująca usuwanie istniejących w danym momencie baz z wynikami trasowania paczek (pole opcji *kasuj bazy przy przeladowaniu*). W przypadku, gdy na każdej z jednocześnie uruchomionych instancji aplikacji MKP wybrane do trasowania będą wszystkie paczki, wówczas każda z tych instancji

aplikacji aktualizuje swój widok w momencie zakończenia trasowania przez inną z instancji określonej paczki. W przypadku, gdy poszczególne instancje aplikacji mają wybrane do trasowania rozłączne zbiory paczek, informacje o nie stanie trasowania paczki nie wybranej w danej instancji programu nie odświeżają się. Wystarczy jednak uruchomić aplikację na nowo i informacje o stanie trasowania dla wszystkich paczek są uaktualnione i widoczne dla użytkownika. Może się zdarzyć, że bardzo szybkie uruchamianie kolejnych instancji aplikacji spowoduje zakomunikowanie błędu związanego z niemożnością podłączenia się do serwera bazy danych. W celu uniknięcia takiej sytuacji zaleca się uruchamianie kolejnych instancji aplikacji oraz procesu trasowania z 2-3 minutowym odstępem.

Nie udało nam się niestety wykonać analizy modułu kosztowego, ponieważ w udostępnionym schemacie bazy danych brak jest funkcji *mk.f_mk_calc_net_sum()* . Niemniej jednak, należy podkreślić, że wyniki trasowania uzyskane przez pojedynczą instancję aplikacji (przy sekwencyjnym przetwarzaniu paczek) jak i te, uzyskane przez wiele równoległych instancji są identyczne. Wydaje się więc, że dalsze kroki pracy z równolegle uzyskanymi wynikami trasowania dla paczek powinny przebiegać tak samo jak przy standardowym, sekwencyjnym przetwarzaniu trasowania przez jedną instancję aplikacji.

Możliwe jest w ten sposób bardziej efektywne wykorzystanie dostępnych zasobów procesora. W szczególności **eksperymenty przeprowadzone dla 5 uruchomień aplikacji MKP pokazały, że możliwe jest wykorzystanie procesora niemal w 100%. Eksperyment przeprowadzony dla paczek 11401 – 11405 pokazał, że czas sekwencyjnego przetwarzania tych paczek przez pojedynczą instancję aplikacji MKP wynosi około 180 min., natomiast poprzez uruchomienie pięciu instancji MKP, z których każda przetwarza osobną paczkę danych możliwe jest przeprowadzenie obliczeń w maksymalnie 60 minut.** Brak pełnej skalowalności 5:1 (uzyskano stosunek redukcji czasu ~3:1) spowodowany jest ograniczeniami interfejsu wejścia/wyjścia systemu dyskowego, na którym operuje serwer bazodanowy.

7.3 Weryfikacja elementów kodu źródłowego

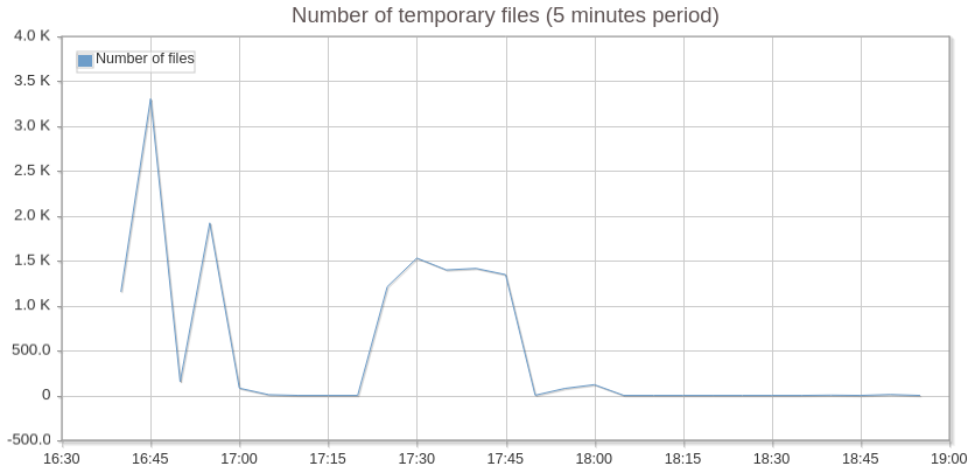
Na podstawie wyników eksperymentów i ich rezultatów zidentyfikowano najbardziej obciążające funkcje i procedury bazy danych *mkp-uke-netmodeller-db.* Poniżej przedstawiono analizę wyników eksperymentów oraz weryfikację kodów źródłowych oraz sugestie rozwiązań mogących przyspieszyć ich wykonanie.

Próbowano przeanalizować dostarczony kod źródłowy w języku Visual Basic zawarty w pliku *Kod aplikacji Trasowanie z raportu.vb*. Jednakże kod ten ewidentnie nie dotyczył wersji oprogramowania, która została nam finalnie dostarczona ze względu m.in. na brak zgodności nazw używanych schematów ze schematami, które występowały w bazie danych. Ponadto w trakcie analizy kodu pojawiła się wątpliwość czy dostarczona wersja stanowiła wersję produkcyjną oprogramowania – przykładowo, w kodzie występują niedomknięte łańcuchy znakowe, co powoduje, że plik ten jest niepoprawny składniowo. Ponadto, warto odnotować, że znajdowały się w nim fragmenty, w których realizowane były zapytania do bazy danych, a ich wyniki nie były w żaden sposób wykorzystywane w dalszych fragmentach. **Otwartym pozostaje więc pytanie, czy tego typu, zupełnie niepotrzebne operacje na bazie danych, nie są przypadkiem obecne również w kodzie aktualnej wersji aplikacji MKP.**

W celu zidentyfikowania fragmentów oprogramowania odpowiadających za największe zużycie zasobów maszyny testowej zebrano oraz przeanalizowano logi serwera bazodanowego PostgreSQL. Na podstawie zebranych logów utworzone zostały raporty przy pomocy oprogramowania *pgBadger*, których treść zostanie omówiona poniżej.

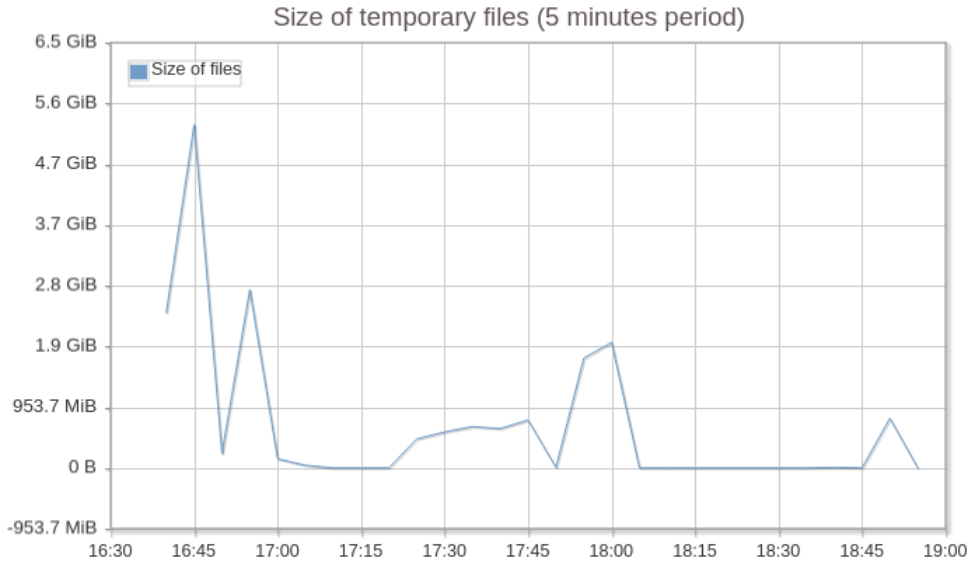
Na podstawie przeprowadzonych eksperymentów stwierdzono, że operacje wejścia/wyjścia są intensywnie stosowane w trakcie trwania symulacji. Pliki tymczasowe tworzone są w przypadku, gdy wykonywane zapytanie nie

mieści się w zarezerwowanej pamięci operacyjnej. Każde utworzenie pliku tymczasowego związane jest ze zwiększoną liczbą operacji wyjścia/wejścia i tym samym spowolnieniem działania aplikacji. Rysunek 5 przedstawiona liczbę plików tymczasowych tworzonych w trakcie działania symulacji dla paczki o ID 11403.



Rysunek 5 Liczba plików tymczasowych tworzonych w trakcie uruchomienia paczki 1140 dla woj. mazowieckiego.

W szczytowym momencie tworzonych było około 206 plików tymczasowych na sekundę, przy czym łączna ich zajętość wynosiła ponad 1 GB. Rysunek 6 przedstawia łączny rozmiar utworzonych plików tymczasowych w trakcie trwania symulacji.



Rysunek 6 Łączny rozmiar tworzonych plików tymczasowych dla paczki 11403.

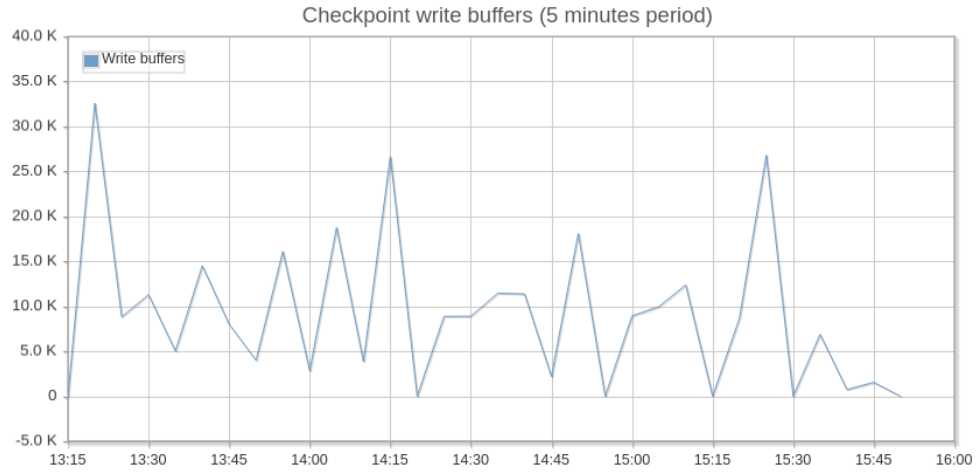
W poniższej tabeli przedstawione są zapytania, które charakteryzują się największą intensywnością operacji wejścia/wyjścia związaną z tworzeniem plików tymczasowych w przypadku symulacji dla paczki o ID 14403.

Tabela 3 Pliki tymczasowe tworzone przez zapytania generujące najintensywniejsze operacje wejścia-wyjścia.

Schemat	Relacja	Liczba plików	Łączny rozmiar [GB]
nr	f_mst_deisgnate_main_nodes_nets	6903	2.97
nr	f_initial_merge_edges_part1	4277	7.06
nr	f_rou_deisgnate_nodes	1721	2

Ponadto, w ramach eksperymentu zauważono, że duża liczba operacji wejścia/wyjścia związana jest z procesem zapisywania na dysk logu WAL (ang. *Write Ahead Log*), w którym rejestrowane są zdarzenia oraz zmiany odnotowywane przez serwer bazodanowy PostgreSQL.

Rysunek 7 przedstawia liczbę zapisanych buforów na pamięć dyskową w ramach omawianego procesu.



Rysunek 7 Liczba zapisanych buforów na pamięć dyskową dla paczki 11403.

Występowanie charakterystycznych pików na powyższym wykresie świadczy o dużej intensywności operacji wejścia/wyjścia. Przyczyną takiego stanu rzeczy może być zbyt mała wielkość pliku logów WAL (strojona parametrami *min_wal_size* oraz *max_wal_size)* lub wybór nieefektywnego sposobu w jaki serwer bazodanowy ma dokonywać zapisów (strojony parametrem *checkpoint_completion_target*).

Zidentyfikowane procedury i funkcje w największym stopniu obciążające zasoby maszyny testowej to m.in.:

- *nr.f_mst_designate_main_nodes_nets* (wraz z inną funkcją przez nią uruchamianą: *nr.f_rou_add_mn_path*),
- *nr.f_initial_merge_edges_part1,*
- *nr.f_rou_node_networks,*
- *nr.f_essential_create_one_side_trenches,*
- *nr.f_initial_merge_edges_part2,*
- *nr.f_clear_graph.*

Implementacja wymienionych funkcji i procedur została przygotowana w języku programowania PL/pgSQL, który pozwala na bezpośrednie odwoływanie się do struktur bazy danych, takich jak tabele i indeksy. Ponadto, wewnątrz funkcji i procedur języka PL/pgSQL możliwe jest wykorzystywanie zapytań SELECT pobierających dane z bazy danych oraz wykonywanie poleceń modyfikujących i dodających już istniejące dane. Wymienione wyżej funkcje i procedury stanowią część schematu *nr* bazy danych *mkp_uke_netmodeller,* który odpowiada za moduł trasowania.

Przeprowadzona analiza kodów źródłowych wskazała następujące potencjalnie nieefektywne elementy:

1. Działanie funkcji i procedur w znacznym stopniu opiera się na iteracyjnym odczytywaniu i zapisywaniu danych do bazy danych przy jednocześnie niewielkim wykorzystaniu pamięci operacyjnej maszyny testowej. W szczególności zauważalny jest schemat działania, w którym tymczasowe wyniki obliczeń nie są zapisywane do struktur dynamicznych języka PL/pgSQL (takich jak np. kursory), ale są tworzone dla nich tymczasowe tabele bazy danych. Podejście takie znacząco zwiększa konieczność wykonywania dodatkowych operacji dyskowych, których można uniknąć przechowują dane tymczasowe w pamięci operacyjnej. Przykładowo w funkcji *nr.f_rou_nodes_rank* operacja odczytu danych wykonywana jest wielokrotnie (dla paczki o ID 11403 ponad 12 tys. razy) i zajmuje ponad 60% czasu wykonywania się tej funkcji.

2. Dla tymczasowych tabel, w których przechowywane są rezultaty obliczeń często są tworzone bądź przebudowywane indeksy. Co ważniejsze ma to często miejsce w każdej iteracji pętli pojawiającej się we funkcji czy procedurze (jak np. w przypadku procedury *nr.f_initial_merge_edges_part1*). Indeks jest strukturą bazy danych, która z założenia ma przyspieszyć czas wyszukiwania danych w bazie (zwłaszcza w przypadku dużej liczby zapytań). Jednakże jego częste przebudowywanie lub tworzenie może wiązać się ze znacznym wykorzystaniem zasobów obliczeniowych komputera. Ponadto baza danych nie zawsze musi wykorzystać dany indeks w generowaniu planu zapytania i wykonywaniu zapytania. W związku z tym sugerowane jest zrezygnowanie z tworzenia części indeksów wewnątrz funkcji i procedur, w szczególności tych działających dla tabel tymczasowych.

3. W celu przyspieszenia działania wymienionych funkcji i procedur możliwe jest zastosowanie zagnieżdżonych zapytań zwłaszcza w odniesieniu do poleceń modyfikujących dane. Jest to możliwe np. w odniesieniu do funkcji *nr.f_initial_merge_edges_part1*. Funkcja ta uruchamia funkcję *nr.f_mst_designate*, która w każdej iteracji pętli wykonuje następujące operacje:

    a. Usunięcie tabeli tymczasowej na dane stworzonej w poprzedniej iteracji pętli.

    b. Stworzenie nowej tabeli tymczasowej dla danych za pomocą polecenia SELECT.

    c. Stworzenie indeksu dla tabeli tymczasowej.

    d. Wykonanie jednego polecenia uaktualniającego dane UPDATE wykorzystując tabelę tymczasową.

Optymalnym rozwiązaniem w tym przypadku byłoby przeniesienie zapytania definiującego tabelę tymczasową wprost do polecenia UPDATE. W takim przypadku możliwa byłaby całkowita rezygnacja z tworzenia tabeli tymczasowej oraz budowania dla niej indeksu.

8 Rekomendacje

W celu poprawy wydajności działania aplikacji rekomendowane są następujące rozwiązania:

1. Modyfikacja kodu źródłowego funkcji i procedur składowanych schematu *nr* w ten sposób, żeby pojedyncze zapytania SELECT nie tworzyły tabel tymczasowych, ale zostały przeniesione bezpośrednio do poleceń modyfikujących i uaktualniających dane (polecenia UPDATE oraz INSERT).

2. Bardziej efektywnym rozwianiem jest modyfikacja kodu źródłowego funkcji schematu *nr* instancji bazy danych *mkp_uke_netmodeller* w ten sposób, żeby rezultaty obliczeń oraz pomocnicze struktury danych przechowywane były w pamięci operacyjnej komputera, a nie w tabelach tymczasowych. W szczególności dotyczy to funkcji: *nr.f_initial_merge_edges_part1* oraz *nr.f_mst_designate_main_nodes_net.*

3. Rezygnację z indeksów tworzonych dla tabel tymczasowych, a w szczególności tych tworzonych w każdej iteracji pętli danej procedury lub funkcji oraz rewizje obecnie stosowanych indeksów (włączając zastosowanie tzw. wskazówek HINTS, które bezpośrednio wskażą, że dane zapytania mają zostać wykonane przy użyciu indeksów).

4. Dostosowanie parametrów konfiguracyjnych serwera do obciążenia pamięciowego ze strony aplikacji oraz do stosowanego sprzętu komputerowego. Serwer bazodanowy PostgreSQL generuje standardową konfigurację bazując na minimalnych wymaganiach sprzętowych niezbędnych do jego pomyślnego uruchomienia i stosowania. Z tego względu domyślne wartości parametrów nie są najczęściej dostosowane pod maksymalizację osiąganej wydajności działania, ani nie są dostosowane dla systemu komputerowego, na którym serwer jest uruchomiony. W celu zwiększenia wydajności działania serwera bazodanowego konieczna jest modyfikacja części parametrów. W Tabela 4 wymienione są najważniejsze parametry, których zmiana może pozytywnie wpłynąć na czas wykonania aplikacji MKP.

Tabela 4 Sugerowane zmiany parametrów konfiguracyjnych serwera PostgreSQL 9.5.

Parametr	Opis parametru	Sugerowana zmiana
shared_buffer	określa wielkość pamięci operacyjnej, którą serwer bazodanowy alokuje na bufory współdzielone	wartość domyślnie ustawiona jest na 128 MB, co dla współczesnych systemów komputerowych, które średnio posiadają 8 GB pamięci operacyjnej, jest stanowczo za mało. Sugeruje się, aby ta wartość stanowiła co najmniej 25% dostępnej pamięci operacyjnej, a na systemach z rodziny Windows wynosiła między 64 MB a 512 MB
effective_cache_size	określa wielkość pamięci podręcznej serwera bazodanowego z punktu widzenia planera zapytań	wartość domyślna wynosi 4 GB co przy większej ilości pamięci RAM (np. 64 GB) może być niewystarczające i skutkować wykonywaniem nieefektywnych zapytań przez planer. Sugeruje się zwiększenie tej wartości do co najmniej 24 GB

checkpoint_completion_target	określa rozpiętość w czasie procesu zapisywania danych z logu zdarzeń na dysk	wartość domyślna wynosi 0.5 i może skutkować tym, że serwer będzie zbyt często wykonywał operacje zapisu danych na dysk. Sugerowane jest podniesienie wartości tego parametru do 0.9
random_page_cost	określa estymator kosztu wykonania zapytania niesekwencyjnego przez planer zapytań	wartość domyślna wynosi 4.0 i jest dostosowana do dysków HDD. Dla dysków SSD koszt skanu losowego i sekwencyjnego jest znacznie mniejszy niż w przypadku dysku HDD, więc wartość tego parametru można obniżyć do wartości ok. 1.1
work_mem	określa wielkość pamięci operacyjnej poświęconej na operacje sortowania oraz na tabele haszujące przed zapisem ich do pliku tymczasowego	Wartość domyślna wynosi 4 MB i może być niewystarczająca w przypadku systemów komputerowych z dostępem do większych zasobów pamięci operacyjnej (np. 64 GB). Sugeruje się podniesienie tej wartości do jeden z wartości z zakresu od 185 MB do 350 MB
min_wal_size	określa minimalny rozmiar logu WAL	wartość domyślna wynosi 80 MB i tym samym może skutkować dużą częstotliwością zapisu stanu logu na dysk. Sugeruje się, aby podnieść tę wartość na co najmniej 2 GB
max_wal_size	określa maksymalny rozmiar logu WAL	wartość domyślna wynosi 1 GB i tym samym może skutkować dużą częstotliwością zapisu stanu logu na dysk. Sugeruje się, aby podnieść tę wartość na co najmniej 6 GB
synchronous_commit	określa sposób obsługi zatwierdzania transakcji przez serwer bazodanowy w zależności od stanu logu WAL	wartość domyślna jest ustawiona na *on.* W celu redukcji operacji wejścia-wyjścia sugeruje się, aby flaga ta ustawiona była na *off*. **UWAGA brak aktywnej flagi może prowadzić do utraty części wykonanych transakcji w przypadku, gdy nastąpi awaria systemu komputerowego lub serwera bazodanowego**

fsync	określa, czy serwer bazodanowy PostgreSQL będzie zapewniał persystencje wszelkich zmian i zdarzeń zaszłych w bazie danych na dysk twardy	wartość domyślna tej flagi jest ustawiona na *on*. W celu zredukowania operacji wejścia-wyjścia i uzyskania przyspieszenia działania aplikacji bazodanowej sugeruje się, aby wartość tej flagi była ustawiona na *off*. **UWAGA brak aktywnej flagi może prowadzić do niespójności bazy danych w przypadku, gdy nastąpi awaria systemu komputerowego lub serwera bazodanowego. Wyłączenie tej flagi sugerowane jest tylko, gdy możliwe jest odtworzenie stanu bazy danych z zewnętrznych źródeł (np. plików CSV)**
checkpoint_timeout	określa czas, po którym serwer bazodanowy zapisuje automatycznie stan logu WAL	wartość domyśla wynosi 5 minut; w celu zredukowania operacji wyjścia-wejścia można podnieść wartość tego parametru do wartości maksymalnej, tj. 60 minut. Należy jednak zaznaczyć, że im większa jest ta wartość, tym dłuższy jest proces odtworzenia stanu bazy danych w przypadku awarii

Aktualnie aplikacja MKP działa w oparciu o serwer bazy danych PostgreSQL 9.5, który nie wykorzystuje mechanizmów współbieżności oraz zrównoleglania operacji obliczeniowych. W związku z tym większość operacji przeprowadzanych przez bazę danych wykonywana jest z pełnym wykorzystaniem tylko jednego rdzenia procesora. Nowsze wersje serwera PostgreSQL (np. wersja 12) udostępniają więcej mechanizmów i parametrów konfiguracyjnych współbieżnego wykonywania operacji. Umożliwiają także oznakowanie funkcji zdefiniowanych przez użytkownika jako takich, które mogą być bezpiecznie wykonywane w sposób współbieżny (opcja ta nie jest dostępna dla serwera PostgreSQL 9.5) oraz domyślnie stosują mechanizm JIT (ang. Just-in-time compilation), polegający na kompilacji wywoływanych funkcji w momencie ich uruchomienia. **W związku z tym rekomendowane jest wykonanie migracji danych na serwer PostgreSQL w wersji co najmniej 12. Testy działania aplikacji MKP z serwerem PostgreSQL 12 przeprowadzone w ramach analizy pokazały, że w tym celu konieczne będzie dokonanie uaktualnienia typów danych niektórych kolumn dla tabel z bazy *mkp*.**

Przejście na nowszą wersję bazy danych jest zalecane także ze względu na możliwą poprawę wydajności wykonywania innych różnorakich operacji bazy danych, jak np. bardziej wydajne generowanie planów zapytań, czy lepsze zarządzanie pamięcią operacyjną.

9 Podsumowanie

Celem ekspertyzy było zidentyfikowanie nieefektywnych elementów aplikacji MKP. W toku przeprowadzonych spotkań z przedstawicielami Urzędu Komunikacji Elektronicznej (UKE) ustalono, że najbardziej czasochłonnym elementem pracy z aplikacją jest tzw. proces *trasowania* (co zostało potwierdzone w przeprowadzonych eksperymentach). Przeprowadzona analiza zidentyfikowała szereg nieefektywnych elementów aplikacji MKP. W szczególności są to: niska jakość kodu źródłowego procedur i funkcji składowanych, brak dostrojenia parametrów konfiguracyjnych serwera PostgreSQL 9.5 oraz przestarzała wersja serwera PostgreSQL nieobsługująca mechanizmów współbieżnego wykonywania zapytań do bazy danych (tzn. obliczenia procedur i funkcji składowanych są przeprowadzane z wykorzystaniem tylko jednego rdzenia procesora).

W celu zapewnienia poprawy wydajności działania aplikacji MKP sugerowane jest wprowadzenie szeregu zmian w implementacji aplikacji oraz konfiguracji serwera bazy danych. W szczególności dotyczą one zmiany implementacji struktur pomocniczych aktualnie wykorzystywanych przez mechanizm trasowania, tak aby w większym stopniu wykorzystywały pamięć operacyjną maszyny testowej, a w mniejszym operacje odczytu i zapisu danych na dysk twardy. Problem ten jest szczególnie istotny dla tabel tymczasowych, które aktualnie są bardzo intensywnie wykorzystywane przez aplikację. Z drugiej strony, dla tabel tymczasowych tworzone są indeksy typu B-drzewo, które, jak pokazały przeprowadzone eksperymenty, niekoniecznie muszą zostać wykorzystane przez serwer bazy danych, a ich budowanie w znacznym stopniu wykorzystuje zasoby aktualnie wykorzystywanego rdzenia procesora oraz operacje odczytu i zapisu danych. Przeanalizowaniu oraz przepisaniu powinien zostać poddany także kod źródłowy języka Visual Basic modułu trasowania aplikacji MKP. Aktualnie zawiera on np. operacje redundantne – zupełnie zbędne zapytania do bazy danych, których wyniki nie są w żaden sposób użyte.

Kolejna z proponowanych rekomendacji odnosi się do właściwego dostrojenia szeregu parametrów konfiguracji zasobów pamięciowych oraz dyskowych serwera bazy danych. W sekcji 8 zostały wskazane przykłady takich parametrów oraz sugerowane zmiany ich wartości.

Aktualnie aplikacja MKP działa w oparciu o serwer bazy danych PostgreSQL 9.5, który nie wykorzystuje mechanizmów współbieżności oraz zrównoleglania operacji obliczeniowych. W związku z tym, większość operacji przeprowadzanych przez bazę danych wykonywana jest z pełnym wykorzystaniem tylko jednego rdzenia procesora. Rekomendowane jest przejście na najnowszą dostępną wersję serwera PostgresSQL ze względu na możliwość wykorzystania mechanizmów wielowątkowego wykonywania operacji bazy danych (mechanizmy takie zostały po raz pierwszy wprowadzone dla serwera PostgreSQL 9.6).