STUDIA INFORMATICA 2015
Volume 36 Number 1 (119)

Andrzej BARCZAK, Dariusz ZACHARCZUK, Damian PLUTA
Uniwersytet Przyrodniczo-Humanistyczny, Instytut Informatyki

METODY OPTYMALIZACJI ROZPROSZONYCH BAZ DANYCH W SYSTEMIE ORACLE

Streszczenie. Artykuł przedstawia zabiegi, jakie należy podjąć w celu maksymalnego dostrojenia wydajności rozproszonej bazy danych. Zostały tu opisane poszczególne metody i ich wpływ na wydajność w zależności od środowiska czy też konfiguracji. Na zakończenie przedstawiono ścieżkę, którą należy przejść oraz dane, jakie należy zebrać, aby cały proces dał pozytywny rezultat.

Słowa kluczowe: optymalizacja, rozproszone bazy danych, Oracle

OPTIMIZATION METHODS IN DISTRIBUTED DATABASES IN ORACLE DBMS

Summary. The article presents the procedures, which need be taken in order to maximize the performance of distributed database. Various methods are describe and their impact on performance depending on the environment and configuration. At the end of publication, author shows the path that one should take and data which should be collected, so that the whole process has given a positive result.

Keywords: optimization, distributed data base, Oracle

1. Wstęp

Można śmiało stwierdzić, że nie ma dnia, w którym nie wdajemy się w interakcję z rozproszoną bazą danych. Niezależnie od tego, czy wypłacamy gotówkę z bankomatu, rezerwujemy bilet do kina, czy umawiamy się na wizytę w serwisie samochodowym. Problem podniesienia wydajności dotyka w pewnym momencie każdej bazy danych i jest zagadnieniem złożonym, zwłaszcza w przypadku rozproszonej bazy danych. Optymalizacja rozproszonej bazy danych musi być wykonana całościowo – w celu uzyskania optymalnej wydajności. nie

można skoncentrować się tyko na jednym obszarze rozproszonej bazy danych lub jedynie na jednym jej węźle. Konieczne jest zbadanie takich obszarów, jak: komunikacja pomiędzy zdalnymi węzłami, działanie mechanizmów replikacji, czy wreszcie sposób wybierania danych ze zdalnych węzłów. Wszystkie powyższe aspekty zostaną przedstawione w niniejszym artykule.

Autor zakłada, że czytelnik zaznajomiony jest z architekturą rozproszonej bazy danych oraz w podstawowym stopniu z zagadnieniami z nią związanymi. Omówienie pojęć zostanie przeprowadzone dość krótko, nakreślając jedynie obszar prezentowanego zagadnienia. Niektóre z mechanizmów (np. partycjonowanie) mogą wymagać bazy w wersji Enterprise.

1.1. Podstawowe definicje

Kluczowe elementy wchodzące w skład architektury rozproszonych baz danych to:

- nazwa usługi bazy danych – określa jedną lub kilka nazw, z którymi klienci mogą się łączyć do instancji bazy danych. Instancja rejestruje te nazwy w procesie zwanym LISTENER'em,
- łącznik baz danych – łączy dwie fizyczne bazy danych i umożliwia użytkownikowi dostęp do nich jak do jednej logicznej bazy danych,
- widoki zmaterializowane, zwane również migawką – w kontekście rozproszonych baz danych są w najprostszym ujęciu kopią zdalnej tabeli z pewnego momentu w czasie. W systemach rozproszonych widoki zmaterializowane są wykorzystywane do synchronizacji aktualizacji danych pochodzących z węzłów zdalnych, włączając w to metody rozwiązywania konfliktów,
- synonimy – mogą być aliasami każdej tabeli, perspektywy, widoku zmaterializowanego, sekwencji, procedury, funkcji, pakietu, obiektu zdefiniowanego przez użytkownika lub innego synonimu,
- perspektywa zwana inaczej widokiem – jest logiczną reprezentacją danych zawartych w jednej lub kilku tabelach bazowych,
- procesy usługowe (dedykowane – domyślne, współdzielone) – tworzone są przez bazę danych w celu obsługi żądań procesów użytkowników podłączonych do instancji.

1.2. Zaawansowana replikacja

Mechanizmowi replikacji należy poświęcić trochę więcej uwagi. Zaawansowana replikacja wykorzystuje technologię rozproszonych baz danych do współdzielenia danych pomiędzy wieloma węzłami. Należy zauważyć, że zreplikowana baza danych nie jest tym samym co rozproszona baza danych. W rozproszonej bazie danych dane są dostępne w wielu lokaliza-

cjach, ale dana tabela znajduje się tylko w jednej lokalizacji. Natomiast replikacja oznacza, że te same dane znajdują się w wielu lokalizacjach. Głównymi powodami, dla których warto korzystać z replikacji, są:

- Dostępność i wydajność – replikacja zapewnia szybki lokalny dostęp do współdzielonych danych, ponieważ obciążenie rozdzielane jest pomiędzy wieloma węzłami.
- Przetwarzanie offline – widok zmaterializowany jest kompletną lub częściową kopią (repliką) tabeli źródłowej z pewnego momentu w czasie, umożliwiającą użytkownikowi pracę na podzbiorze bazy danych, która może być chwilowo nieosiągalna.
- Zmniejszenie obciążenia sieciowego – replikacja może być wykorzystana do dystrybucji odpowiednich podzbiorów danych do regionalnych węzłów systemu.

1.2.1. Replikacja multimaster

Środowisko replikacji multimaster składa się z wielu równoważnych węzłów nadrzędnych uczestniczących w procesie uaktualniania danych. Zmiany dokonane w jednym węźle nadrzędnym są propagowane do wszystkich innych węzłów nadrzędnych uczestniczących w procesie replikacji. Zastosowanie replikacji multimaster pozwala na zwiększenie dostępności danych oraz zapewnia równomierny rozkład obciążenia. Istnieją dwa typy replikacji multimaster: asynchroniczny i synchroniczny. Replikacja asynchroniczna przechwytuje lokalne zmiany, zapisuje je w kolejce i w regularnych odstępach czasu propaguje je do zdalnych węzłów. Replikacja synchroniczna nanosi wszystkie zmiany w danych na zasadzie transakcji. Ten typ replikacji zapewnia spójność danych we wszystkich węzłach w czasie rzeczywistym.

1.2.2. Replikacja migawkowa

Widok zmaterializowany jest repliką tabeli źródłowej z określonego momentu w czasie. Źródłem może być zarówno tabela zlokalizowana w węźle nadrzędnym, jak i widok zmaterializowany w węźle migawkowym. Widoki zmaterializowane są aktualizowane przez węzły migawkowe, poprzez nadchodzące od nich z określoną częstotliwością indywidualne aktualizacje, zwane odświeżeniami. Zastosowanie replikacji migawkowej pozwala osiągnąć następujące korzyści:

- redukcję obciążenia sieciowego,
- możliwość podziału danych na podzbiory,
- możliwość pracy offline.

W systemie Oracle dostępnych jest kilka typów widoków zmaterializowanych, które znajdują zastosowanie w różnych środowiskach replikacji:

- widoki zmaterializowane typu Primary Key – są domyślnym typem widoku i mogą być aktualizowane, jeśli zostały utworzone jako część grupy widoków zmaterializowanych i w ich definicji została wyspecyfikowana klauzula FOR UPDATE,

- widoki zmaterializowane typu ROWID – bazują na fizycznym identyfikatorze wiersza w tabeli źródłowej. Mogą być użyte na tabeli źródłowej nieposiadającej klucza podstawowego lub w sytuacji, gdy widok zmaterializowany nie uwzględnia wszystkich kolumn, które wchodzą w skład klucza podstawowego,
- widoki zmaterializowane typu złożonego – nie mogą być aktualizowane przyrostowo. Widok zmaterializowany uznawany jest za złożony, jeśli definiujące go zapytanie zawiera:
  - Klauzulę CONNECT BY.
  - Operacje INTERSECT, MINUS, UNION, czy UNION ALL.
  - Słowa kluczowe DISTINCT lub UNIQUE.
  - Funkcje agregujące (chociaż są od tego wyjątki): COUNT, AVG, SUM, VARIANCE i STDEV.
  - Złączenia.

W celu określenia, czy widok zmaterializowany spełnia wszystkie ograniczenia wymagane do tego, aby mógł być odświeżany w trybie przyrostowym, najprościej jest zdefiniować taki widok z opcją odświeżania przyrostowego (REFRESH FAST). Oracle zwróci błąd, jeśli widok zmaterializowany narusza jakiekolwiek ograniczenia widoków zmaterializowanych typu prostego.

W celu pełnego odświeżenia widoku zmaterializowanego wykonywane jest zapytanie, definiujące widok zmaterializowany, które po prostu ponownie tworzy ten widok. Jeśli zostało wykonane pełne odświeżanie widoku zmaterializowanego, na którym bazują inne widoki zmaterializowane, wtedy wszystkie widoki zależne również muszą zostać odświeżone w trybie pełnym.

Odświeżanie przyrostowe jest bardziej wydajne od pełnego odświeżania, gdy liczba zmian w obiekcie bazowym jest niewielka oraz kiedy każda kolumna wymieniona w warunku złączenia w zapytaniu definiującym widok zmaterializowany ma założony na niej indeks.

2. Optymalizacja rozproszonych baz danych

Rozdział ten porusza zagadnienia, które mają bezpośredni wpływ na wydajność rozproszonej bazy danych, poczynając od mechanizmów komunikacji pomiędzy węzłami rozproszonej bazy danych i obsługi żądań użytkowników, kończąc na mechanizmach dostępu do danych zlokalizowanych w węzłach rozproszonej bazy danych.

2.1. Optymalizacja procesów usługowych

Punktem wyjściowym dla optymalizacji procesów usługowych może być podejrzenie, że system działa niewydajnie lub zaobserwowano spadek wydajności. W przypadku współdzielonych procesów usługowych, na samym początku należy zbadać, czy ekspedytorzy radzą sobie z liczbą nadchodzących żądań i czy procesy usługowe są w stanie obsłużyć ten wolumen żądań. Natomiast jeśli system oparty jest na dedykowanych procesach usługowych, w pierwszej kolejności należy zweryfikować, czy liczba procesów usługowych jest wystarczająca oraz czy maszyna ma wystarczającą liczbę zasobów sprzętowych do ich obsługi.

Strojenie współdzielonych procesów usługowych polega na ustaleniu optymalnej liczby ekspedytorów i procesów usługowych wymaganej do stabilnego i wydajnego działania systemu. Aby określić, czy liczba procesów ekspedytorów jest wystarczająca do obsłużenia danego poziomu żądań, można posłużyć się dynamiczną perspektywą systemową V$DISPATCHE oraz V$QUEUE, wybierając z nich następujące dane:

```
D.NAME  AS DISPATCHER_NAME,
ROUND((D.BUSY/(D.BUSY+D.IDLE))*100, 2) AS TOTAL_BUSY_RATE,
ROUND(Q.WAIT/Q.TOTALQ, 2) AS AVERAGE_WAIT
…
WHERE Q.TYPE = 'DISPATCHER' AND Q.PADDR = D.PADDR;
```

W przypadku gdy obciążenie bazy danych zmienia się w zależności np. od pory dnia, należy zbadać obciążenie bazy danych w tym właśnie przedziale czasowym. Na podstawie zwróconych wyników można zaobserwować, czy działające procesy ekspedytora nie są nadmiernie obciążone. Gdyby zajętość (TOTAL_BUSY_RATE) przekraczała 50% lub czasy oczekiwania (AVERAGE_WAIT) przyjmowały znaczące wartości, zalecane byłoby zwiększenie liczy ekspedytorów. np.

```
ALTER SYSTEM SET DISPATCHERS="(PROTOCOL=tcp)(DISPATCHERS=5)";
```

W przypadku zauważenia zbyt wielu zadań w kolejce procesów do obsługi (informacja dostępna w widoku systemowym V$QUEUE.ITEMS_QUEUED) i długiego czasu oczekiwania (V$QUEUE.AVERAGE_WAIT), należy rozważyć zwiększenie liczby procesów usługowych, wykorzystując do tego parametry inicjalizacyjne SHARED_SERVERS i MAX_SHARED_SERVERS i dalej monitorować zawartości kolejki zadań do obsługi.

Wart sprawdzenia jest również stan wirtualnych pętli (widok V$CIRCUIT), za pomocą których użytkownicy łączą się z bazą danych poprzez ekspedytorów i procesy usługowe. Wartości kolumny QUEUE przechowują informacje o tym, w jakim stanie znajduje się aktualnie wirtualna pętla:

- COMMON – oczekiwanie w kolejce zadań na podjęcie przez proces serwera;
- DISPATCHER – oczekiwanie na ekspedytora, zbyt wysoka wartość może oznaczać przeciążenie serwera;

- SERVER – aktualnie trwa obsługa przez proces serwera;
- NONE – stan bezczynności.

Aby definitywnie stwierdzić, czy obciążenie procesów serwera jest zbyt duże, należy posłużyć się kolejnym zapytaniem wykorzystującym widok V$SHARED_SERVER oraz następującą instrukcją:

```
ROUND((SS.BUSY/(SS.BUSY+SS.IDLE))*100, 2)||'%' AS TOTAL_BUSY_RATE
```

W sytuacji zajętości powyżej 50% rekomendowane jest zwiększenie liczby procesów usługowych i dalsze monitorowanie tego aspektu wydajności bazy danych.

2.2. Współdzielenie połączeń

Rozważmy system, który jest w stanie obsłużyć 950 połączeń przez każdy z procesów ekspedytora i sytuację, w której wymagana jest obsługa 4000 użytkowników jednocześnie podłączonych za pośrednictwem protokołu TCP/IP i 2500 użytkowników jednocześnie podłączonych za pośrednictwem protokołu TCP/IP z SSL. W tym przypadku wymaganych jest minimum pięciu ekspedytorów dla protokołu TCP/IP i minimum trzech ekspedytorów dla protokołu TCP/IP z SSL:

```
DISPATCHERS="(PROTOCOL=tcp)(DISPATCHERS=5)"
DISPATCHERS="(PROTOCOL=tcps)(DISPATCHERS=3)"
```

Współdzielenie połączeń pozwala każdemu z ekspedytorów na obsłużenie więcej niż 950 sesji. Załóżmy, że w tym przypadku każdy z ekspedytorów będzie w stanie obsłużyć 4000 sesji TCP/IP i 2500 sesji TCP/IP z SSL. Taka konfiguracja zredukuje liczbę wymaganych ekspedytorów do jednego dla każdego z protokołów i pozwoli na bardziej efektywne wykorzystanie procesów ekspedytorów:

```
DISPATCHERS="(PROTOCOL=tcp)(DISPATCHERS=1)(POOL=on)(TICK=1)
(CONNECTIONS=950)(SESSIONS=4000)"
DISPATCHERS="(PROTOCOL=tcps)(DISPATCHERS=1)(POOL=on)(TICK=1)
(CONNECTIONS=950)(SESSIONS=2500)"
```

Jednak zanim zdecydujemy się na włączenie współdzielenia połączeń, należy decyzję tę poprzeć analizą obciążenia współdzielonych procesów usługowych i procesów ekspedytorów. Współdzielenie połączeń okaże się skuteczne, gdy mamy do czynienia z sytuacją, w której podłączeni klienci przez większość czasu znajdują się w stanie bezczynności, dzięki czemu ekspedytor będzie w stanie obsłużyć zwiększoną liczbę sesji.

2.3. Strojenie łączników baz danych

Mamy dwa typu łączników: współdzielony lub nie. Czynnikiem przemawiającym za zastosowaniem współdzielonych łączników baz danych będzie znacznie większa liczba użyt-

kowników, korzystających z łączników baz danych od liczby połączeń do lokalnej bazy danych. Wyjątek: współdzielony łącznik baz danych w scenariuszu z jednym użytkownikiem prowadzi do większej liczby połączeń sieciowych, dlatego należy korzystać z niego w sytuacjach, kiedy wielu użytkowników będzie wymagało dostępu do tego samego łącznika.

2.4. Optymalizacja zapytań

W celu optymalizacji zapytań SQL należy posłużyć się dostępnymi narzędziami, tj. EXPLAIN PLAN, SQL Trace czy TKPROF1[1], aby dowiedzieć się, w jaki sposób zapytania zostają rozbite i rozproszone do poszczególnych węzłów.

Najbardziej efektywnym sposobem optymalizacji zapytań rozproszonych jest dostęp do zdalnych baz danych w minimalnym zakresie, umożliwiającym uzyskanie wymaganych danych. Przykładem może być sytuacja, w której w zapytaniu rozproszonym odwołujemy się do dwóch zdalnych tabel zlokalizowanych na tym samym zdalnym węźle. Wydajność takiego zapytania może być poprawiona przez napisanie zapytania w taki sposób, aby odwoływało się ono tylko raz do zdalnej bazy danych oraz przez założenie filtra po stronie zdalnej bazy danych, co pozwoli na przesłanie do lokalnej bazy danych niezbędnego minimum danych. Napisanie zapytania we wspomniany powyżej sposób jest możliwe z wykorzystaniem połączonego widoku wbudowanego. W celu przedstawiania istoty tej konstrukcji wymagane jest wyjaśnienie kilku pojęć:

- widok wbudowany – zapytanie umieszczone w klauzuli FROM innego zapytania, które zastępuje tabelę w zapytaniu nadrzędnym:

```
SELECT … FROM (SELECT empno, ename FROM emp@orc1.world) e … WHERE …;
```

- Połączony widok wbudowany – widok wbudowany, który wybiera dane z wielu tabel zlokalizowanych w tej samej bazie danych. Rozwiązanie to redukuje liczbę dostępów do bazy danych, poprawiając tym wydajność zapytania rozproszonego.

Należy zaznaczyć, że optymalizator kosztowy Oracle potrafi wiele zapytań rozproszonych przepisać do postaci wykorzystującej połączony widok wbudowany. Można także do zapisu zapytań wykorzystać technikę subquery factoring, która umożliwi ponowne użycie zapytania oraz poprawi też jego czytelność.

2.4.1. Wykorzystanie optymalizacji kosztowej

Głównym zadaniem optymalizacji jest przepisanie zapytania rozproszonego do postaci wykorzystującej połączony widok wbudowany. Optymalizacja ta odbywa się w trzech krokach:

[1] Narzędzia i metody optymalizacji baz danych w Oracle 10g cz. 2

1) Wszystkie możliwe do scalenia widoki są scalane.
2) Optymalizator przeprowadza test bloku połączonego widoku wbudowanego.
3) Optymalizator przepisuje zapytanie do postaci wykorzystującej połączony widok wbudowany.

Optymalizacja kosztowa optymalizuje zapytania rozproszone, zgodnie ze statystkami zebranymi dla tabel, do których odwołuje się zapytanie oraz obliczeniami wykonywanymi przez optymalizator. Przykładowo, poniższe zapytanie zostało poddane optymalizacji kosztowej:

```
SELECT … FROM local l, remote1 r1, remote2 r2
WHERE l.c = r.c AND r1.c = r2.c AND r1.e > 300;
```

I po optymalizacji przyjmuje postać:

```
SELECT … FROM (  SELECT … FROM remote1 r1, remote2 r2
       WHERE r1.c = r2.c AND r1.e > 300) v, local l
WHERE l.c = r1.c;
```

Wykorzystanie konstrukcji połączonego widoku wbudowanego ogranicza wolumen danych wymagany do przetworzenia w zdalnym węźle, a tym samym redukuje ilość danych przesyłanych do węzła lokalnego, ograniczając kosztowny ruch sieciowy. Wzrost wydajności zapytania możliwy jest dzięki przeniesieniu warunku r1.e > 300, co powoduje zawężenie zbioru danych już w zdalnej bazie danych.

2.4.2. Wskazówki optymalizatora

W przypadku gdy zapytanie nie jest wystarczająco zoptymalizowane, możliwe jest zastosowanie wskazówek optymalizatora w celu poprawy efektu optymalizacji kosztowej:

- NO_MERGE: używamy w sytuacji, gdy napisane zapytanie rozproszone bazuje na wiedzy eksperckiej i wykorzystuje już widok wbudowany. Wskazówka jest również przydatna, jeśli zapytanie rozproszone zawiera agregacje, podzapytania lub złożony SQL, ponieważ zapytanie rozproszone zawierające któryś z tych elementów i tak nie zostanie przepisane przez optymalizator. Szybciej więc będzie, jeśli pominiemy optymalizację.
- DRIVING_SITE: wymusza wykonanie zapytania w innym węźle niż węzeł, który standardowo zostałby wybrany przez Oracle, np. w wytypowanym na podstawie wiedzy eksperckiej węźle zdalnym.
- ORDERED: instruuje optymalizator o tym, że złączenie tabel wymienionych w klauzuli FROM powinno nastąpić zgodnie z kolejnością, jaka została ustalona w tej klauzuli.
- LEADING: instruuje optymalizator, aby jako pierwsze tabele w porządku złączenia wykorzystał zestaw tabel wymieniony w tej wskazówce. Oracle rekomenduje używanie tej wskazówki zamiast ORDERED.

2.5. Strojenie widoków zmaterializowanych

Widoki zmaterializowane są bardzo przydatnym mechanizmem, umożliwiającym zredukowanie powtarzających się operacji wejścia-wyjścia, a w pewnych sytuacjach są wręcz niezastąpione. Niestety, problemem jest zapewnienie, aby widoki zmaterializowane były możliwie jak najświeższym odbiciem danych z ich tabel źródłowych, ponieważ widoki zmaterializowane potrafią bazować na wielu tabelach. Mechanizm widoków zmaterializowanych w bazie danych Oracle został już zoptymalizowany przez producenta bazy, dlatego bezpośrednia optymalizacja nie jest możliwa. Istnieje jednak kilka sposobów na pośrednie podniesienie wydajności tego mechanizmu.

Pierwszym krokiem, jaki powinien zostać podjęty w celu przyśpieszenia odświeżania widoku zmaterializowanego, jest analiza i optymalizacja zapytania definiującego widok zmaterializowany.

Zastosowanie partycjonowania tabel bazowych może w znaczy sposób wpłynąć na skrócenie czasu odświeżania widoków zmaterializowanych. W przypadku gdy klucz, na podstawie którego tabela została podzielona na partycje, odpowiada kluczowi wykorzystanemu w klauzuli WHERE zapytania definiującego widok zmaterializowany, a dodatkowo odświeżanie odbywa się w trybie całościowym.

Kolejnym sposobem na przyśpieszenie odświeżania widoków zmaterializowanych jest założenie indeksów na tabelach bazowych, które w miarę możliwości będą pokrywały się z warunkami w klauzuli WHERE zapytania definiującego widok zmaterializowany. Korzyści z wykorzystania indeksów będą szczególnie widoczne podczas odświeżania w trybie całościowym, w sytuacji gdy widok zmaterializowany prezentuje jedynie niewielką część danych tabeli bazowej, tj. mniejszą niż 15%. W przeciwnym przypadku wykorzystanie indeksów może być nieefektywne.

Równoległe wykonanie odświeżania umożliwia wielu procesom jednoczesne odświeżanie widoku zmaterializowanego, co w rezultacie przyśpiesza proces odświeżania. W sytuacji idealnej rozbicie procesu odświeżania widoku zmaterializowanego pomiędzy 4 równoległe procesy powinno spowodować 4-krotne skrócenie czasu odświeżanie tego widoku. Niestety, biorąc pod uwagę podsystem dyskowy, który może nie być w stanie obsłużyć czterech równoległych procesów z taką samą szybkością co jeden, czas ten może ulec wydłużeniu. Jednak i tak uzysk ze zrównoleglonego odświeżania będzie znaczny. Przed podjęciem decyzji o zrównolegleniu procesu odświeżania widoków zmaterializowanych konieczna jest analiza utylizacji zasobów sprzętowych bazy danych.

2.5.1. Porównanie prostych i złożonych widoków zmaterializowanych

W zależności od charakterystyki widoku zmaterializowanego w niektórych przypadkach należy rozważyć wykorzystanie kilku prostych widoków zmaterializowanych, które mogą być odświeżane przyrostowo, zamiast jednego złożonego widoku zmaterializowanego, którego odświeżanie przyrostowe jest niemożliwe ze względu na niespełnienie kryteriów dla widoków prostych. Weźmy pod uwagę dwa warianty bazujące na widoku w bazie lokalnej i jednej bazie zdalnej z dwiema tabelami bazowymi:

a) Widok zmaterializowany typu złożonego: bazujący na złożonym widoku zmaterializowanym, który w lokalnej bazie danych zapewnia wysoką wydajność zapytań, ponieważ operacja złączenia została wykonana podczas odświeżania widoku zmaterializowanego. Jednak z powodu złożoności widoku odświeżanie widoku następuje w trybie pełnym przez co dłużej niż w przypadku odświeżania przyrostowego.
b) Widok zmaterializowany typu prostego: bazuje na dwóch widokach zmaterializowanych typu prostego w bazie lokalnej, w której wykonywane jest także ich złączenie. W przypadku tej metody wydajność zapytań nie będzie już tak dobra jak w przypadku widoku złożonego w wariancie A. Jednakże proste widoki zmaterializowane mogą być odświeżane bardziej efektywnie z wykorzystaniem dziennika widoku zmaterializowanego i odświeżania przyrostowego.

Podsumowując, jeśli dane mają być odświeżane rzadko i wymagana jest większa wydajność zapytań, to lepszym rozwiązaniem jest widok zmaterializowany typu złożonego.

2.6. Rozmiar grupy odświeżania

Oracle jest zoptymalizowany pod kątem dużych grup odświeżania, dlatego odświeżanie tej samej liczby widoków zmaterializowanych w dużej grupie odświeżania zakończy się szybciej niż odświeżenie tej samej liczby widoków zmaterializowanych w małych grupach, zakładając, że chodzi o te same widoki. Podczas odświeżania takiej grupy każdy widok zmaterializowany w grupie jest blokowany w węźle migawkowym na czas wymagany do odświeżenia wszystkich widoków z grupy odświeżania. Dlatego też mniejsze grupy odświeżania oznaczają blokadę wszystkich widoków w grupie na krótszy odcinek czasu wymagany do jej odświeżenia. Podczas odświeżania widoków połączenie sieciowe musi być stale utrzymywane, w przeciwnym wypadku wszystkie zmiany są wycofywane w celu zachowania spójności bazy danych. Zatem w przypadkach, gdy trudno jest utrzymać połączenie sieciowe przez dłuższy czas, również zalecane jest stosowanie mniejszych grup odświeżania.

3. Przebieg procesu testowania

Testowanie wydajności rozproszonej bazy danych, opierając się na omówionych metodach optymalizacyjnych, nie jest zadaniem prostym, i co ważniejsze, wyniki będą dedykowane tylko i wyłącznie tej konkretnej topologii i konfiguracji, na której test zostanie dokonany. Czynniki takie, jak wiedza ekspercka nt. modelu logicznego optymalizowanej bazy, rozłożenia danych w tabelach czy chociażby znajomość warunków technicznych sprawiają, że nie jesteśmy w stanie wypracować i poddać testom ogólnego rozwiązania. Zmiana któregokolwiek czynnika wymaga przeprowadzania ponownych badań. Zadaniem każdego administratora rozproszonej bazy danych jest przejście przez poszczególne etapy i metody indywidualnie, a analiza efektów pracy doprowadzi do przyjęcia najbardziej odpowiedniego rozwiązania.

Proces testowania rozproszonej bazy danych powinien wyglądać następująco:

1) Testy dedykowanych i współdzielonych procesów usługowych: powinny polegać na symulacji obciążenia bazy danych za pośrednictwem wspomnianych procesów. Taki test powinien dostarczyć informacji nt: wykorzystania CPU, wykorzystania obszaru pamięci SGA i PGA, liczby logicznych i fizycznych operacji I/O, liczby sesji, poziomu trafień w bufor, średnich czasów wykonania zapytań, statusu wirtualnych pętli.
2) Testy publicznych i współdzielonych łączników baz danych dostarczą informacji nt.: obciążenia procesów ekspedytorów, obciążenia procesów serwera, czasu wykonywania zapytań, wysokości zdarzeń oczekiwania sesji na sieć.
3) Testy zapytań rozproszonych: korzystając z narzędzi SQL Trace i TKPROF, otrzymamy plany i rzeczywiste czasy wykonywania zapytań, do tych testów należą:
   - Test połączonych widoków wbudowanych i optymalizacja kosztowa.
   - Test wskazówek optymalizatora.
4) Test widoków zmaterializowanych:
   - Test partycjonowania tabel bazowych: korzystamy z narzędzi SQL Trace oraz TKPROF i poddajemy analizie plan i czas wykonywania zapytania.
   - Testy indeksów na tabeli bazowej: jw.
   - Test prostych i złożonych widoków zmaterializowanych: w zależności od rodzaju odświeżania należy wziąć pod uwagę ilość modyfikowanych danych w tabelach bazowych, a ostatecznej analizie poddać otrzymane czasy pobierania danych.
   - Test „małych” i „dużych” grup odświeżania: analizie poddajemy czasy odświeżania grup w trybie sekwencyjnym i równoległym oraz czas odświeżania kompletu widoków zmaterializowanych.

4. Podsumowanie

Posiadając pełną kolekcję wyników z całej ścieżki testowania, dostrojenie rozproszonego środowiska pozostanie już tylko formalnością. Jednak należy mieć świadomość, że optymalizacja baz danych jest bardzo szeroką dziedziną i artykuł ten z pewnością nie wyczerpuje tematu. Dodatkowo, jak już zostało to powiedziane w poprzednim rozdziale, kluczowy może być przypadek, nad optymalizacją którego pracujemy i do którego dobieramy rozwiązania. Obrana ścieżka postępowania w tym artykule bazowała właśnie na konkretnym, praktycznym problemie. Dzięki temu na zakończenie przedstawione zostaną częściowe wyniki testów, dokonanych podczas optymalizacji realnego problemu. Przedstawienie pełnych rezultatów wraz z ich analizą niestety nie jest możliwe z uwagi na ich obszerność.

Wachlarz dostępnych możliwości najnowszych baz danych jest oczywiście dużo większy, niż przedstawiony w artykule. Można np. użyć mechanizmów ADDM (Automatic Database Diagnostic Monitor), które służą do analizy danych zawartych w AWR (Automatic Workload Repository), aby zidentyfikować potencjalne wąskie gardła. Z pewnością można napisać jeszcze wiele artykułów badawczych, wykorzystujących te lub inne narzędzia, gdyż temat jest non stop aktualny.

4.1. Przykładowy test dedykowanych i współdzielonych procesów usługowych

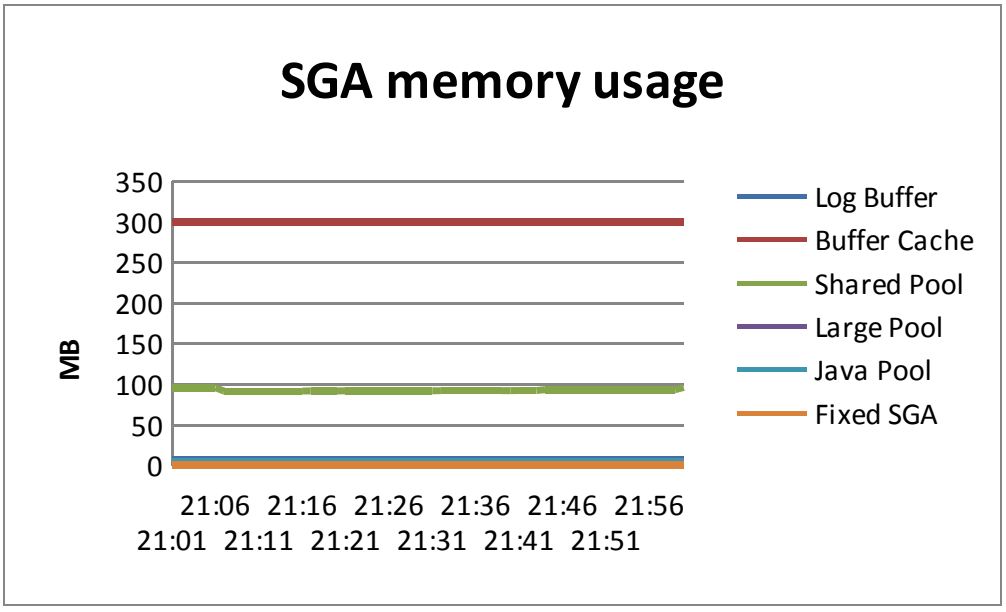
W procesie testowania wydajności bazy danych dla dedykowanych i współdzielonych procesów usługowych został wykorzystany główny węzeł rozproszonej bazy danych ORADB1, na którym znajdują się wszystkie tabele testowej bazy danych. W celu zasymulowania obciążenia bazy danych, generowanego przez 100 podłączonych klientów, powstała dedykowana aplikacja, zaimplementowana w języku Java. Wykonuje ona co określony interwał czasu jedno z czterech losowo wybranych zapytań, z również losowo wybranymi parametrami.

Przykład jednego z wykorzystywanych zapytań:

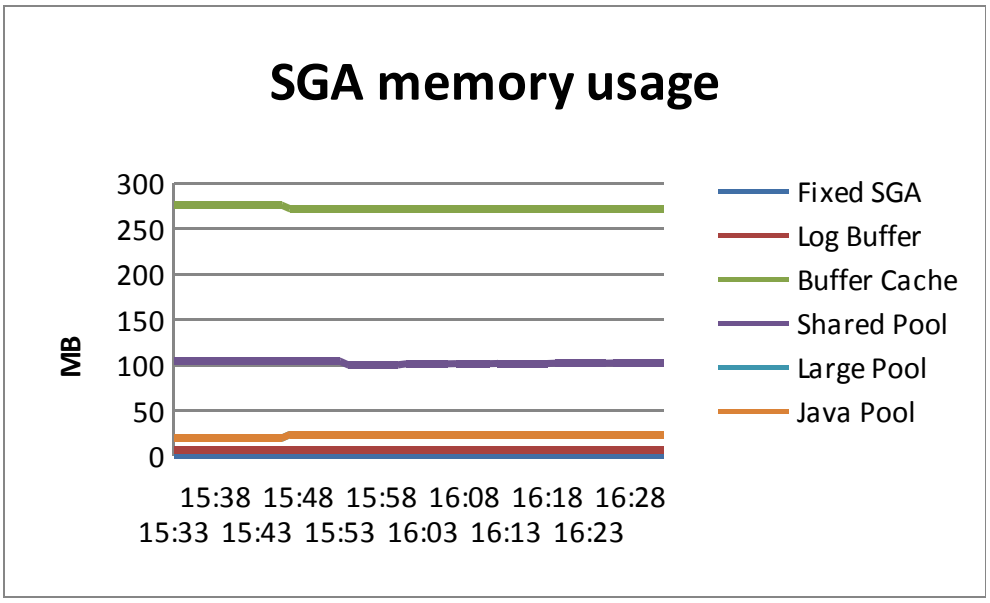
```
SELECT
    A.PLATNIK_ID, A.CYKL_BILINGOWY, A.MSISDN, PT.NAZWA AS PLAN_TARYFOWY,
   S.OPIS AS STATUS, A.DATA_AKTYWACJI
FROM
    (SELECT
        P.PLATNIK_ID, P.CYKL_BILINGOWY, K.MSISDN, K.PLAN_TARYFOWY_ID,
       K.STATUS, K.DATA_AKTYWACJI
    FROM
        TELEKOM.PLATNICY P, TELEKOM.KONTRAKTY K
    WHERE
        P.PLATNIK_ID = K.PLATNIK_ID AND
        P.PLATNIK_ID = (SELECT CEIL(DBMS_RANDOM.VALUE(0,200000)) FROM DUAL)) A,
        TELEKOM.SLO_PLANY_TARYFOWE PT,
        TELEKOM.SLO_STATUSY S
WHERE
    A.STATUS = S.STATUS AND
```

```
A.PLAN_TARYFOWY_ID = PT.PLAN_TARYFOWY_ID;
```

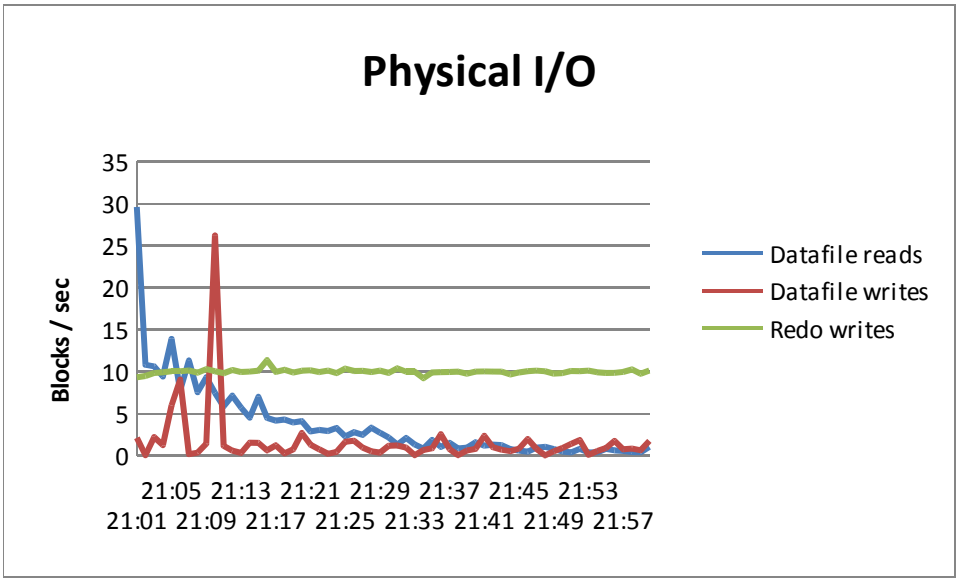
Oto wybrane zrzuty ekranów dla monitorowanych obszarów.



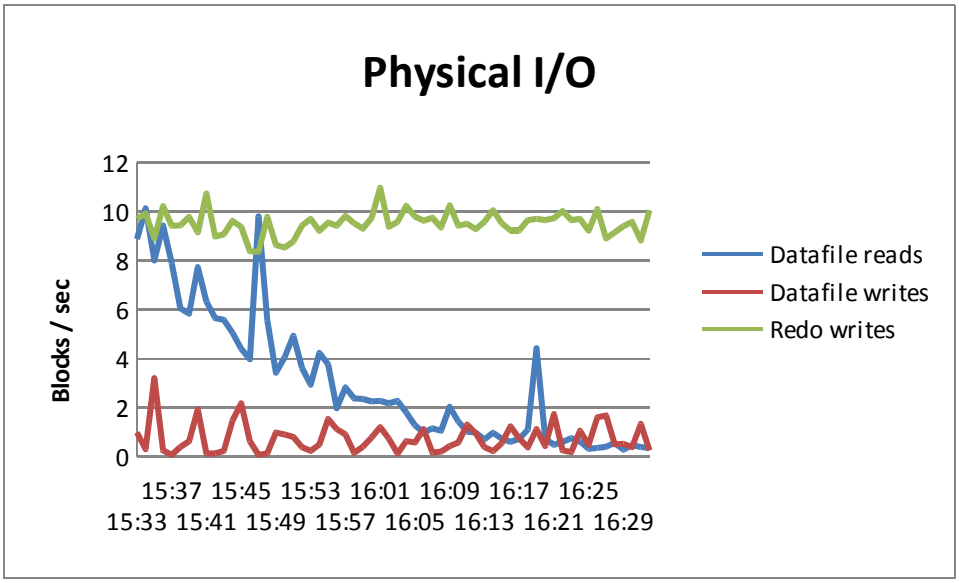
Rys. 1. Wykorzystanie obszaru SGA w teście dedykowanych procesów
Fig. 1. SGA usage for dedicated precesses



Rys. 2. Wykorzystanie obszaru SGA w teście współdzielonych procesów
Fig. 2. SGA usage for common precesses



Rys. 3. Liczba fizycznych operacji wejścia-wyjścia dla procesów dedykowanych
Fig. 3. Number of physical I/O for dedicated processes



Rys. 4. Liczba fizycznych operacji wejścia-wyjścia dla procesów współdzielonych
Fig. 4. Number of physical I/O for common processes

Dodatkowo w poniższych tabelach przedstawiono porównanie parametrów uzyskanych w testach.

Tabela 1

Wyniki testów

Monitor	Procesy dedykowane	Procesy współdzielone
Średnie użycie procesora [%]	~45%	~40%
Poziom trafień w bufor [%]	99,13	99,95
Średnie czasy zapytań [s]	0,41 0,58 0,11 0,12	0,98 1,22 0,20 0,29
Obciążenie współdzielonych procesów serwera	-	S000 - 53,97%

Tabela 2

Zajętość obszaru pamięci PGA podczas testów

SCHEMA	PGA_USED_MEMORY [MB]	PGA_ALLOCATED_MEM [MB]
Procesy dedykowane		
TELEKOM	64,81	101,59
ALL_SCHEMA	84,74	137,13
Procesy współdzielone		
TELEKOM	1,01	1,74
ALL_SCHEMA	26,82	47,91

Tabela 3

Obciążenie procesów ekspedytorów przy testach procesów współdzielonych

DISPATCHER_NAME	TOTAL_BUSY_RATE [%]	AVERAGE_WAIT [sec.]
D000	1,12	0,04

Tabela 4

Obciążenie kolejki zadań do obsługi procesów współdzielonych.

QUEUE_NAME	TYPE	ITEMS_QUEUED	AVERAGE_WAIT [sec.]
00	COMMON	4	0,21

Tabela 5

Status wirtualnych pętli

QUEUE_STATUS	ITEMS
SERVER	1
NONE	95
COMMON	4

Na podstawie tych oraz pozostałych wyników testów oraz przy założeniach, jakie miał spełniać system, została podjęta decyzja o użyciu 2 współdzielonych procesów, gdyż przy 1 procesie czasy wykonywania zapytań nie były zadowalające.

BIBLIOGRAFIA

1. Alapati S. R.: Expert Oracle Database 10g Administration. Apress, 2005.
2. Antognini Ch.: Troubleshooting Oracle Performance. Apress, 2008.
3. Barczak A., Florek J., Sydoruk T.: Bazy danych. Wydawnictwo Akademii Podlaskiej, 2007.
4. Burleson D. K.: Oracle Tuning: The Definitive Reference. Rampant TechPress, 2010.
5. Dye Ch.: Oracle Distributed Systems. O'Reilly, 1999.
6. Greenwald R., Stackowiak R., Stern J.: Oracle Essentials: Oracle Database 10g. 3rd Edition. O'Reilly, 2004.

7. Kyte T.: Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming Techniques and Solutions. Apress, 2010.
8. Lonley K., Bryla B.: Oracle Database 10g Podręcznik administrator baz danych. Helion, Gliwice 2008.
9. Powell G.: Oracle Performance Tuning for 10gR2. Elsevier Digital Press, 2007.
10. Price J.: SQL i Programowanie, Oracle Database 11g. Helion, Gliwice 2009.
11. Tow D.: SQL. Optymalizacja. Helion, Gliwice 2004.
12. Urman S., Hardman R., McLaughlin M.: Oracle Database 10g. Programowanie w języku PL/SQL. Helion, Gliwice 2007.
13. Whalen E., Schroeter M.: Oracle Optymalizacja wydajności. Helion, Gliwice 2003.
14. Wrembel R., Bębel B.: Oracle Projektowanie rozproszonych baz danych. Helion, Gliwice 2003.
15. Oracle PL/SQL Database Code Library and Resources, http://psoug.org/reference/library.html.
16. Oracle: Dokumentacja techniczna.

Abstract

Database optimization problem sooner or later is the theme of each major system. In the case of distributed databases, the situation is quite complicated and hard methods, ie. expansion of the hardware, in large diffuse environment are expensive and short-term solution. The only reasonable solution is to thorough analysis and tuning of each of the components of BD. The article discusses the path that must be overcome and methods that must be applied to so that the optimization of distributed database was successful. The tuning process in a distributed environment should be done as a whole. Because of that, presented methods apply to: dedicated and shared servers processes, shared and public database connectors, simple and collocated inline views, materialized views. Also the ways to optimize sql queries was shown. Using it will allow to shorten the time required to download the data, which is our main concern. At the end the it was specified what information tests should provide us. On this basis can successfully optimize the distributed database.

Adresy

Andrzej BARCZAK: Uniwersytet Przyrodniczo-Humanistyczny, Instytut Informatyki,
ul. 3 maja 54, 08-110 Siedlce, Polska, andrzej.barczak@neostrada.pl.

Dariusz ZACHARCZUK: Uniwersytet Przyrodniczo-Humanistyczny, Instytut Informatyki, ul. 3 maja 54, 08-110 Siedlce, Polska, dzariusz@dzariusz.pl.

Damian PLUTA: Uniwersytet Przyrodniczo-Humanistyczny, Instytut Informatyki, ul. 3 maja 54, 08-110 Siedlce, Polska.