Problematyka masowego dostępu do baz danych – mity i fakty

Wojciech Karwowski, Piotr Kołodziej
Oracle Polska
e–mail: Wojciech.Karwowski@oracle.com, Piotr.Kolodziej@oracle.com

Abstrakt. W dobie internetowych systemów informatycznych coraz większą rolę odgrywa ich skalowalność. Masowy i nieskrępowany dostęp do danych może stwarzać poważne problemy wydajnościowe. Wbrew obiegowym opiniom, zwiększanie zasobów systemowych np. liczby procesorów, rozmiaru pamięci itp. może nie przynosić spodziewanych efektów. Konieczne staje się zrewidowanie konstrukcji aplikacji i struktury bazy danych. Referat ma na celu obalenie powszechnie spotykanych mitów dotyczących „poprawiania wydajności", przeciwstawiając im weryfikowalne fakty.

1. Skalowanie to nie zawsze to samo

„*Inność to powinność*" Jan I. Sztaudynger

Mówiąc o mitach dotyczących skalowania systemów i zarządzania wydajnością należy zacząć od spostrzeżenia, że już samo pojmowanie skalowania systemów obrosło licznymi mitami. Najbardziej pospolity z nich sprowadza się do ufności jaka jest potrzebna, by zadać ze śmiertelną powagą pytanie typu: „*Będę miał(a) aplikację na N użytkowników. Ilu na to potrzeba procesorów?"* Niestety, udzielenie wiarygodnej odpowiedzi na to pytanie bez znajomości charakteru aplikacji i planowanego obciążenia przekracza możliwości zwykłego śmiertelnika. Zaś przekonaniu, że można oszacować zapotrzebowanie na zasoby systemowe wyłącznie na podstawie liczby użytkowników należy przeciwstawić następujące fakty:

1. *Szacując obciążenie systemu należy wziąć pod uwagę charakter realizowanych zadań oraz rozłożenie zadań na poszczególnych użytkowników.*
2. *W systemach, w których równolegle wykonywane są zadania interaktywne i wsadowe, całkowite obciążenie systemu w sensie zużycia jego zasobów w największym stopniu zależy od charakteru i liczby uruchamianych zadań wsadowych i w stosunkowo niewielkim stopniu od liczby użytkowników.*

Kolejny mit dotyczy formułowania celu, do którego się dąży przy pracach nad wydajnością systemu. Spotyka się czasem cel sformułowany jako np. utrzymanie czasu jałowego procesorów na poziomie kilkudziesięciu procent. Tymczasem w systemie, w którym uruchomiona jest pewna liczba zadań wsadowych bądź raportów, obecność czasu jałowego procesorów jest raczej objawem niepokojącym, gdyż zadania zamiast się wykonywać, najwyraźniej na coś oczekują. Ponadto w systemie, w którym procesory nie wykonują cykli jałowych może wciąż istnieć duża rezerwa przepustowości.

Kryterium dobrej wydajności nie można sformułować wyłącznie za pomocą kategorii statystyk systemu operacyjnego czy serwera bazy danych. Najistotniejsze jest osiągnięcie wymaganego poziomu obciążenia wykonywanymi operacjami oraz akceptowalnego czasu ich realizacji.

2. Od przybytku głowa nie boli?

Szeroko rozpowszechniony jest pogląd, że rozbudowa konfiguracji sprzętowej jest najszybszym i najefektywniejszym sposobem skalowania systemu. Opiera się na słusznym skądinąd spostrzeżeniu, że modyfikowanie oprogramowania jest czynnością dużo bardziej złożoną niż wymiana sprzętu i grozi dodatkowo utratą funkcjonalności bądź stabilności ze względu na możliwość wprowadzenia nowych błędów w oprogramowaniu.

Tego rodzaju podejście jest słuszne w przypadku, gdy wąskim gardłem systemu jest rzeczywisty niedostatek zasobów: brak mocy procesorów, niedostatek pamięci bądź niewystarczająca wydajność urządzeń wejścia/wyjścia, zaś optymalizacja aplikacji nie pozostawia wiele do życzenia. Nie należy się jednak spodziewać dobrego skalowania czy rozwiązania problemów wydajnościowych w przypadku, gdy zapotrzebowanie na zasoby systemowe jest spowodowane przede wszystkim przez złą optymalizację, np. złe indeksowanie, nieefektywny plan wykonania zapytań SQL itd. Wzrost przepustowości bądź skrócenie czasu reakcji będą prawdopodobnie niewspółmiernie niskie w stosunku do wzrostu zasobów systemowych.

Zwiększenie wydajności może wcale nie nastąpić w przypadku, gdy dodawanie zasobów systemowych stanowi akt desperacji spowodowany tym, że tak naprawdę nie wiadomo, co jest przyczyną problemów wydajnościowych, np. nagłych i trudnych do wytłumaczenia spowolnień pracy. W większości tego rodzaju przypadków przyczyn należy dopatrywać się w problemach związanych z komunikacją sieciową oraz synchronizacją dostępu procesów do współdzielonych zasobów. Problemy synchronizacyjne mogą mieć miejsce na poziomie logicznym aplikacji (np. oczekiwanie na blokadach związanych z modyfikacją wierszy w tabelach) oraz wewnętrznym serwera bazy danych Oracle bądź systemu operacyjnego. W specyficznych przypadkach dodanie zasobów może wręcz pogorszyć wydajność systemu.

2.1. Spiesz się powoli?

Powszechnie się uważa, iż zwiększenie szybkości procesorów prowadzi zawsze do skrócenia czasu reakcji. Pogląd taki może być całkowicie słuszny w przypadku systemu, w którym występuje tylko jeden proces. Tymczasem, zależnie od algorytmu programu szeregującego oraz charakteru zadań, może w pewnych warunkach dojść do wydłużenia czasu reakcji pewnych procesów w przypadku zwiększenia prędkości procesorów. Jest to prawdopodobne w systemach, w których równolegle uruchamiane są zadania interaktywne oraz zadania wsadowe bądź duże raporty – zwiększenie mocy procesorów może doprowadzić do wydłużenia czasu obsługi procesów interaktywnych. Wyjaśnienie tego fenomenu na podstawie teorii kolejkowania można znaleźć np. w pracy [2].

Podobny efekt może być wywołany także przez mechanizm zarządzania pamięcią wirtualną (*Virtual Memory Manager*) w systemach operacyjnych, w których mechanizm ten jest zintegrowany z dynamicznym buforem plikowym. W przypadku pewnego niedoboru pamięci, uruchomienie zadań wsadowych intensywnie korzystających z systemu plików może spowodować „podkradanie" stron procesom interaktywnym z reguły oczekującym na zdarzenia. Intensywne zapotrzebowanie na pamięć potrzebną na buforowanie operacji wejścia/wyjścia prowadzi do zwolnienia ostatnio nie używanych stron pamięci (algorytm LRU). Wzrost szybkości procesora oznacza w takim przypadku zwiększenie aktywności zadań wsadowych, zwiększenie zapotrzebowania na wolną pamięć i tym samym prawdopodobieństwa wydłużenia reakcji procesów interaktywnych w związku ze zwiększonym stronicowaniem należących do nich obszarów.

Przykład z mechanizmem zarządzania pamięcią wirtualną stanowi ilustrację dobrze znanych zasad:

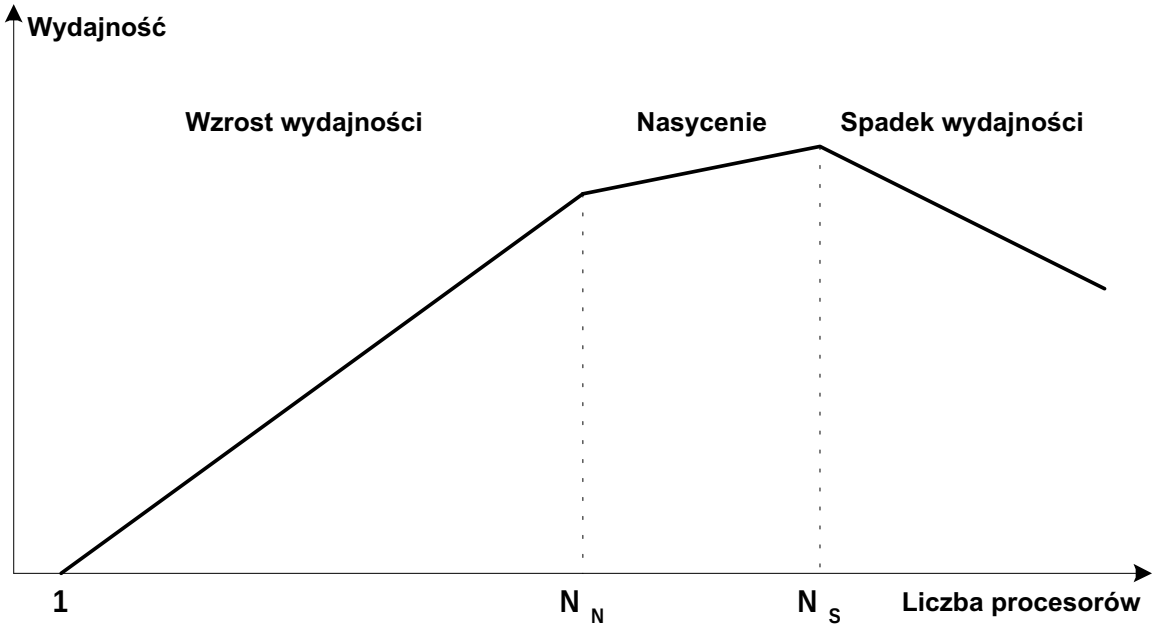
1. *Należy dodawać te zasoby, których najbardziej brakuje.*
2. *Dodanie zasobów może spowodować zwiększone zapotrzebowanie na zasoby innego rodzaju.*

2.2. Superkomputery dobre na wszystko?

Współczesne komputery oferują możliwości konfigurowania coraz większej liczby procesorów w architekturze SMP. Zastosowanie rozwiązań znanych wcześniej w technologiach superkomputerowych pozwala na uzyskanie bardzo wysokiej mocy obliczeniowej oraz wysokiej przepustowości przy transmisjach danych angażujących pamięć oraz urządzenia wejścia/wyjścia. Ww. cechy systemów wieloprocesorowych sprawiają, że słusznie uważa się je za doskonałą platformę dla systemów baz danych.

Z drugiej strony, zdarzają się sytuacje, kiedy po uruchomieniu niektórych aplikacji na maszynach kilkunastoprocesorowych problemy wydajnościowe się pogłębiają. Z reguły charakteryzują się one stopniowym pogarszaniem wydajności – wydłużaniem czasu reakcji aż do wystąpienia objawów zawieszenia.

Zjawisko to można tłumaczyć m.in. na podstawie modelu skalowalności aplikacji na maszynach wieloprocesorowych (rys. 1). Zakłada on istnienie trzech podstawowych obszarów – wzrostu wydajności, nasycenia oraz spadku wydajności. Chociaż przyjęcie liniowości charakterystyki we wszystkich trzech obszarach jest bardzo dużym uproszczeniem, to model ten jest w zupełności wystarczający dla ilustracji omawianych zjawisk. Inną niedoskonałością modelu jest oczywista trudność w jednoznacznym przełożeniu dość subiektywnie pojmowanej wydajności na wartości liczbowe[1].



Rys. 1. Model skalowalności aplikacji na maszynach wieloprocesorowych

Podstawowymi parametrami modelu są:

- N_N – liczba procesorów, powyżej której dodanie do systemu następnych procesorów nie przynosi zauważalnego wzrostu wydajności.
- N_S – liczba procesorów, powyżej której dodanie do systemu następnych procesorów powoduje spadek wydajności.

Wartości parametrów powyższego modelu oraz nachylenia charakterystyk w obszarze wzrostu bądź spadku wydajności zależą od konstrukcji sprzętu, systemu operacyjnego, serwera bazy danych i przede wszystkim, konstrukcji aplikacji oraz planowanego obciążenia systemu w czasie eksploatacji. W przypadku bardzo dobrze skonstruowanych aplikacji można się spodziewać dobrego skalowania serwera bazy Oracle do przynajmniej kilkudziesięciu procesorów.

W architekturach superkomputerowych nasycenie i spadek wydajności związane są przede wszystkim ze wzrastającym kosztem synchronizacji dostępu do wewnętrznych zasobów serwera bazy Oracle oraz systemu operacyjnego na poziomie zatrzasków (*latches, spinlocks*). W istocie zagadnienie synchronizacji dostępu do krytycznych zasobów serwera bazy Oracle w systemach wieloprocesorowych jest analogiczne, jak w przypadku systemów operacyjnych. W obu

[1] Od czego jednak są funkcje kryterialne? Zawsze można zmajstrować jakąś na poczekaniu.

przypadkach klasyczną metodę ochrony ścieżek krytycznych zastąpiono przez zorientowaną na strukturę danych ochronę zasobów krytycznych. Bez przebudowy mechanizmu szeregowania dostępu do krytycznych obszarów jądra nie byłoby możliwe efektywne skalowanie systemów nawet dla konfiguracji kilkuprocesorowej.

W przypadku serwera bazy danych Oracle, do dobrego skalowania na maszynach SMP istotny jest optymalny kod aplikacji nie powodujący zbyt częstych oczekiwań na synchronizację dostępu do krytycznych zasobów jądra w obszarze dzielonym SGA. Oczekiwania związane z synchronizacją mają miejsce wówczas, gdy z powodu modyfikowania obszarów krytycznych przez inne procesy niemożliwe byłoby zapewnienie spójności wykonywanej operacji. Zagadnienie synchronizacji nabiera wagi wraz ze wzrostem liczby procesów mogących równolegle uzyskiwać dostęp do pamięci – pośrednio zatem zależy od liczby użytkowników oraz od liczby procesorów.

Najczęściej spotykana problematyka synchronizacji dostępu do zasobów współdzielonych w warunkach pracy wielu sesji obejmuje m.in.:

1. Zajętość obszaru *library cache*, związaną z jego intensywnym zapełnianiem przez kursory, które nie są współdzielone. Tego rodzaju problemy są spowodowane przez m.in.:
   - Niewłaściwą metodykę parametryzowania dynamicznie generowanych instrukcji SQL polegającą np. na bezpośrednim wpisywaniu parametrów zamiast wartości wiązanych (*bind variables*)[2]. Jest to szczególnie dotkliwe, gdy problem jest umiejscowiony w często wykonywanej sekcji kodu. Jednym z najbardziej oczywistych jego przejawów jest obecność wielu niemal identycznych kursorów w tabeli dynamicznej V$SQLAREA[3].
   - Konieczność utrzymywania wielu wersji tego samego kursora (zapytania). Częstą przyczyną jest projekt bazy danych i aplikacji zakładający utrzymywanie w schemacie każdego z użytkowników „warstwy" obiektów pośrednich (synonimów, perspektyw, modułów PL/SQL). W takim przypadku w tabeli V$SQLAREA można zaobserwować wysoką wartość atrybutu VERSION_COUNT przy kursorach, które wykorzystują parametryzację przez wartości wiązane.
2. Konieczność utrzymywania wielu negatywnych zależności dla wykorzystywanych nazw w przestrzeniach nazewniczych (*namespaces*) użytkowników. Jest to typowy efekt bardzo częstego odwoływanie się do obiektów bazy danych za pośrednictwem synonimów publicznych. Nie stanowi to jednak istotnego problemu w przypadku, gdy sesje są uruchamiane na prawach użytkowników bazy Oracle (np. rozwiązania oparte o kartrydże PL/SQL).
3. Wysoką zajętość bufora bazowego (*buffer cache*) – spowodowaną przez istnienie rozległych „gorących obszarów" aplikacji (*hot spot area*). Obszary te charakteryzują się bardzo wysoką częstością odczytów i zapisów.

Powyższe zagadnienia prawie w całości należą do sfery projektu i konstrukcji schematu bazy danych i aplikacji. Znaczne ich nasilenie może spowodować, że charakterystyka skalowania aplikacji na maszynach SMP wejdzie w obszar nasycenia lub znacznego spadku wydajności już dla około 10 procesorów.

Wnioski:

1. *Jak na ironię, superkomputery wymagają dobrze zoptymalizowanych aplikacji. W szczególności konieczne jest zminimalizowanie liczby operacji wymagających wyłączności dostępu*

[2] Przekazywanie kryteriów do zapytania za pomocą literałów ma uzasadnienie np. w wypadku dużych zapytań *ad-hoc*, gdyż tylko wtedy optymalizator kosztowy może skorzystać z histogramów. Tego typu zapytania są wykonywane na tyle rzadko, że nie powinny wpływać na zajętość obszaru *library cache*.

[3] W serwerze bazy Oracle 8i pojawia się możliwość wymuszenia współdzielenia ww. zapytań przez ustawienie parametru CURSOR_SHARING=FORCE. Należy się jednak liczyć z pewnym wpływem tego ustawienia na pracę optymalizatora kosztowego.

do wewnętrznych struktur serwera bazy danych leżących w obszarze współdzielonym (shared pool).

2. *Decyzję o zastosowaniu superkomputera warto poprzedzić badaniem aplikacji pod kątem optymalizacji dostępu do zasobów dzielonych. W niektórych przypadkach pozwoli to na uniknięcie przykrego rozczarowania faktem, że charakterystyka skalowalności aplikacji na maszynach SMP wykazuje znaczny jej spadek dla posiadanej liczby procesorów.*

3. Kłamstwa, benchmarki i statystyki?

Wokół standardowych testów wydajnościowych (m.in. TPC-C i TPC-D) narosły dwa nawzajem sprzeczne nieporozumienia. Pierwsze polega na bezkrytycznym przenoszeniu uzyskanych rezultatów na sferę wdrażanego systemu, drugie z kolei polega na całkowitym negowaniu ich wartości.

Liczby, jak to liczby, wymagają interpretacji. Im bardziej charakterystyka rzeczywistej aplikacji byłaby zbliżona do aplikacji TPC, tym bardziej wyniki testów byłyby miarodajne dla określenia planowanego obciążenia. Niestety, rzeczywiste aplikacje istotnie różnią się między sobą i prawie w niczym nie chcą przypominać aplikacji używanych w testach TPC, choćby dlatego, że są z reguły dużo bardziej skomplikowane. Nie sposób też znaleźć wiarygodnej formuły pozwalającej na przełożenie obciążenia stosowanego w testach na obciążenie występujące w rzeczywistym systemie.

Pomimo to, wyniki uzyskane w testach TPC-C/TPC-D posiadają wartość istotną przy skalowaniu systemów i zarządzaniu wydajnością:

- Z reguły stanowią dość dobre oszacowanie górnych granic możliwości skalowania systemu.
- Pozwalają na porównanie możliwości sprzętu i oprogramowania systemowego. Do tego celu zostały stworzone.

Ze względu na fakt, że na podstawie wyników standardowych testów nie sposób określić wydajności rzeczywistych aplikacji, konieczne jest samodzielne prowadzanie badań wydajności. Najczęściej obserwowane jest obciążenie systemu i czasy reakcji wybranych funkcji aplikacji.

Najczęściej spotykane problemy związane z metodą badania wydajności rzeczywistej aplikacji obejmują m.in.:

1. Inny charakter obciążenia systemu przy rzeczywistej pracy jak w czasie testów.
2. Nieuwzględnienie charakterystyki obciążenia, związanej z uruchomieniem dużej liczby sesji na prawach różnych użytkowników (por. problemy opisane w punkcie 2.2)

Uzyskana charakterystyka wydajności, choćby obciążona niedokładnościami, stanowi najlepszą podstawę do określenia docelowych wymagań sprzętowych, systemowych oraz docelowej przepustowości systemu. Dodatkową korzyścią wynikającą z prowadzenia testów wydajnościowych aplikacji jest możliwość wychwycenia problemów aplikacji przed oddaniem jej do eksploatacji.

Wniosek:

Lepsze niedoskonałe badanie wydajności rzeczywistej aplikacji niż żadne.

4. Cudowne lekarstwa?

Naturalną reakcją na wystąpienie problemów wydajnościowych w fazie wdrożenia systemu bądź jego eksploatacji, jest poszukiwanie środków zaradczych. Niestety, z reguły nie istnieją łatwe rozwiązania, gdy problem wydajnościowy jest głęboko zakorzeniony w naturze aplikacji, np. projekcie struktury bazy danych, przyjętych konwencjach itd.

Przyjęcie do wiadomości, że dzieło, w które włożyło się wiele wysiłku i często wiele serca może wymagać czasochłonnych poprawek bądź gruntownej przebudowy z reguły jest bardzo trudne i budzi mniej lub bardziej świadomy opór. Jest to nieomal niemożliwe w atmosferze zagrożenia – np. w przypadkach niekonstruktywnego formułowania kwestii odpowiedzialności, napiętych harmonogramów, nierealistycznych wymagań bądź zobowiązań oraz ostrych konfliktów.

Tego rodzaju sytuacje sprzyjają zachowaniom ucieczkowym – w sferze technicznej przejawiają się skupieniem na poszukiwaniach sposobu, w wyniku którego problemy znikną jak za dotknięciem czarodziejskiej różdżki. Poszukiwanie udoskonaleń samo w sobie jest rzeczą bardzo pozytywną. Jednak znalezienie rozwiązania problemu jest mało prawdopodobne, gdy świadomie bądź nieświadomie ucieka się przed poznaniem jego genezy oraz związanych z nim mechanizmów.

Przykładem są decyzje o rozbudowie sprzętu w sytuacji, gdy natura problemu jest nie do końca znana. W punkcie 2 pokazano, że ich efekty mogą być przeciwne do zamierzonych. Innym nierzadko spotykanym przykładem są próby praktycznej realizacji poglądu, że wystarczy zbuforować całą (!) bazę danych w pamięci, by wyeliminować problemy związane długim czasem realizacji zapytań SQL. Pomijając fakt, że nawet dla niedużych systemów postulat ten jest wciąż trudny do spełnienia, to zwiększenie prędkości transmisji do procesorów dużych ilości danych (konsekwencja złej optymalizacji) powoduje znaczne zwiększenie ich zajętości oraz obciążenie szyn systemowych. W takim przypadku można się spodziewać, że w systemie powstanie wąskie gardło w postaci niedostatku mocy obliczeniowej. Innym problemem staną się liczne oczekiwania w związku z synchronizacją dostępu do współdzielonego bufora bazowego i zarządzaniem kolejkami LRU.

Jak każda ucieczka od problemu, zachowanie to nie przynosi pożądanych efektów – problem zazwyczaj powraca, często ze zdwojoną siłą. Sytuacja z reguły staje się dużo trudniejsza – nieefektywnie wykorzystano sporo cennego czasu, poniesione zostały nieefektywne nakłady, budżet zostaje nadszarpnięty, cierpliwość użytkownika też.

Wniosek:

Realną szansę na trwałe i satysfakcjonujące rozwiązanie problemu daje poznanie i oddziaływanie na jego przyczyny.

5. Uczniowie czarnoksiężnika

Rozwój sprzętu i oprogramowania daje możliwości tworzenia rozwiązań coraz bardziej wydajnych i skalowalnych. Nowe specyficzne cechy stanowią m.in. odpowiedź na znane z niektórych wcześniejszych doświadczeń bariery funkcjonalne i wydajnościowe.

Bardzo silnie zakorzenionym i często występującym zachowaniem (z całą pewnością wykracza to poza sferę mitu) jest wewnętrzny przymus do wykorzystania cech owianych fascynującym nimbem nowości bądź tajemnicy, niezależnie od tego, czy rzeczywiście istnieje potrzeba zastosowania nowej funkcjonalności. Daje się zauważyć syndrom ucznia czarnoksiężnika – łatwe wprawianie w ruch coraz bardziej złożonych mechanizmów, nad którymi łatwo traci się kontrolę wskutek braku wiedzy i umiejętności, do których zdobycia konieczny jest czas i odpowiedni zbiór doświadczeń. Nie mniej istotne wydaje się nabranie właściwego stosunku do nowych możliwości, pozwalające na dokonanie w pełni świadomego wyboru uwzględniającego założony cel, spodziewane zyski jak i możliwe koszty. Wbrew zamierzeniom, nieprzemyślane i nieskoordynowane zastosowanie nowej funkcjonalności może spowodować poważne problemy związane z wydajnością i stabilnością systemu.

Przykłady opisanych wyżej działań są często identyczne jak działania opisane w punktach 2 i 4. Ponadto do tej kategorii można zaliczyć np.:

- Uruchamianie *parallel query* i *parallel execution* na maszynach nie posiadających stosownych zasobów – np. jednoprocesorowych, w systemach z niedoborem mocy obliczeniowej, z niedostateczną wydajnością dysków.

- Próby zwiększenia wydajności systemu poprzez zastosowanie Oracle Parallel Server bez podjęcia prac w kierunku minimalizacji narzutu związanego z synchronizacją węzłów oraz w sytuacji, gdy problemy wydajnościowe wynikają przede wszystkim ze skali problemów w synchronizacji dostępu do współdzielonych obszarów serwera bazy – przede wszystkim *buffer cache* i *row cache*.
- Nadmierne partycjonowanie tabel – np. stosowanie dużej, nie dającej się niczym uzasadnić liczby partycji bądź subpartycji.
- Stosowanie MTS (*Multithreaded Server*) w warunkach dużej rezerwy pamięci bądź znacznej zajętości sesji przez wykonywane polecenia SQL.
- Stosowanie 64-bitowych wersji oprogramowania w celu skonfigurowania bardzo wielkich obszarów *buffer cache* i *shared pool* bez uprzedniego zbadania sposobu wykorzystania ich przez aplikację.
- Stosowanie mechanizmów typowych dla systemów DSS oraz *data warehouse* do optymalizowania zapytań występujących w aplikacjach OLTP – np. zapytania gwiaździste (*star queries*), złączenia typu hash (*hash joins*).
- Stosowanie indeksów bitmapowych w często modyfikowanych tabelach.

Omawiane działania nie są z reguły w stanie rozwiązać problemów wydajnościowych, gdyż zazwyczaj nie działają na ich przyczyny. Negatywnym skutkiem podejmowania tego rodzaju przedsięwzięć jest przede wszystkim stworzenie nowych problemów, odwrócenie uwagi od problemu podstawowego oraz pochłanianie sił i środków koniecznych do zrealizowania działań nie przynoszących w efekcie spodziewanej poprawy wydajności.

Wnioski:

1. *Konstruowanie złożonych i skalowalnych systemów w większości przypadków wymaga mistrzowskiego stosowania możliwie prostych środków.*
2. *Zastosowanie rozwiązań zaawansowanych powinno stanowić świadomą decyzję, mającą na celu osiągnięcie dobrze określonych celów, do których osiągnięcia proste i standardowe rozwiązania są niewystarczające.*
3. *Większa liczba „ciekawych" rozwiązań nie musi się przekładać na lepszą wydajność i większą stabilność.*
4. *Wiele problemów związanych z wydajnością może mieć bardzo proste podłoże leżące w sferze projektu lub konfiguracji systemu. Do ich rozwiązania nie jest koniecznie stosowanie zaawansowanych technologii.*

6. Większa skala wymaga wyższej kultury eksploatacji

Niezależnie od dostępnych zasobów, uzyskanie dobrej wydajności wymaga opracowania choćby najprostszego programu obserwacji i zarządzania obciążeniem systemu. Jego celem jest zapewnienie w miarę równomiernego rozłożenia obciążenia systemu w czasie jego eksploatacji oraz niedopuszczanie do wystąpienia spiętrzeń zadań prowadzących do wyczerpania rezerwy mocy obliczeniowej i zasobów systemowych. Nie mniej istotne jest unikanie sytuacji, gdy równoległe wykonanie zadań powoduje poważne problemy w synchronizowaniu dostępu do współdzielonych zasobów na poziomie logicznym bądź na poziomie jądra serwera bazy Oracle (por. punkt 6.1). W szczególności, konieczne jest wydzielenie cykli uruchamiania szczególnie obciążających zadań, oraz cykli utrzymania systemu – m.in. backup'u, konserwacji oprogramowania, utrzymania obiektów oraz przestrzeni bazy danych.

6.1. Niewinne z pozoru operacje

W przypadku dużej skali systemu np. w warunkach pracy bardzo wielu użytkowników, wykonanie pewnych, wyglądających na neutralne dla wydajności systemów operacji może spowodować dotkliwe konsekwencje.

Przykładowo, szczególnej uwagi wymagają aplikacje intensywnie korzystające z kodu PL/SQL składowanego w bazie danych (np. serwisy internetowe oparte o kartrydże PL/SQL, niektóre aplikacje Oracle Forms). Istotne znaczenie ma unikanie kompilacji PL/SQL podczas eksploatowania systemu przez wielu użytkowników oraz sytuacji prowadzących do jego unieważnienia (*invalidation*) na skutek zarządzania zależnościami (*object dependency management*). Unieważnienie PL/SQL składowanego w bazie danych powoduje z kolei konieczność jego rekompilacji przy kolejnej próbie jego wykonania [3].

Kompilacja PL/SQL bądź rekompilacja wywołana jego unieważnieniem może być przyczyną bardzo poważnych problemów wydajnościowych związanych z synchronizowaniem dostępu do obszaru *library cache.* Występujące wówczas liczne zaśnięcia sesji przy oczekiwaniu na zdarzenia *'library cache pin'* oraz *'library cache lock'* mogą spowodować ich praktyczne zawieszenie.

Unieważnianie kodu PL/SQL powodowane jest przez instrukcje **ALTER**, **REVOKE**, **GRANT**, **CREATE OR REPLACE**, wykonane na obiektach bazowych, do których ten kod się odwołuje. Jest to opisany w [3] efekt kaskadowy, związany z mechanizmem zarządzaniem zależnościami między obiektami. Ww. problem może się ponadto objawiać występowaniem błędów ORA-4020 (zakleszczenie przy próbie uzyskania wyłączności dostępu do obiektu) , ORA-4021 (przekroczenie limitu czasu oczekiwania przy próbie uzyskania wyłączności dostępu do obiektu) oraz ORA-600, najczęściej z argumentem [17285], wraz z towarzyszącymi błędami ORA-4061 (stan obiektu PL/SQL został unieważniony), ORA-4065 (zmieniony lub usunięty podprogram PL/SQL) i ORA-6508 (brak możliwości wywołania podprogramu PL/SQL).

Przytoczony wyżej problem stanowi jeden z przykładów świadczących o konieczności wprowadzenia cykli utrzymania systemu. Ponadto konstrukcja, konfiguracja i praktyka bieżącej konserwacji systemu powinny możliwie skutecznie zapobiegać wykonaniu operacji związanych utrzymywaniem bazy danych w czasie jego większego obciążenia.

6.2. Dobra praktyka administrowania

Wraz ze wzrostem skali systemu coraz większego znaczenia nabiera także stosowanie praktyk administrowania bazą danych pozwalających na zminimalizowanie narzutu serwera bazy Oracle przy zarządzaniu przestrzenią. Redukowanie liczby rekursywnych operacji SQL oraz operacji wejścia/wyjścia pozwala na udostępnienie maksymalnych zasobów systemowych na potrzeby wykonywanych aplikacji. Zalecane praktyki administrowania bazą danych, mające na celu uniknięcie dynamicznego zarządzania przestrzenią obejmują m.in.:

1. Ustawianie stosownie dużych wartości parametrów INITIAL, NEXT dla segmentów[4].
2. Prealokację ekstentów w intensywnie modyfikowanych tabelach w chwilach ich mniejszego obciążenia.
3. Prealokację przestrzeni tabel w trakcie mniejszego obciążenia systemu.

Celem jest zarówno uniknięcie narzutu związanego z dynamicznym zarządzaniem przestrzenią jak i możliwości wystąpienia błędów związanych z niemożnością rozszerzenia segmentów (np. tabel i indeksów) o kolejne ekstenty.

[4] Grzechem dość powszednim biorąc pod uwagę jego częstość, zaś ciężkim biorąc pod uwagę jego skutki jest stosowanie bardzo dużej liczby drobnych ekstentów do składowania segmentów, np. konfigurując parametry typu: STORAGE (INITIAL 10K NEXT 10K PCTINCREASE 0 MAXEXTENTS UNLIMITED).

Następnym przykładem jest konieczność prawidłowej konfiguracji segmentów wycofania w systemach charakteryzujących się dużym rozmiarem wykonywanych zmian w danych oraz uruchamianiem długotrwałych zapytań, np. w systemach obciążonych równoczesną pracą o charakterze OLTP, zadaniami wsadowymi oraz raportowaniem. Zbyt mały rozmiar przestrzeni zaalokowanej w segmentach wycofania powoduje częste występowanie błędów ORA-1555 („przestarzała migawka") przy wykonywaniu długotrwałych i kosztownych zapytań. Wspomniany błąd jest spowodowany brakiem możliwości uzyskania spójnego obrazu danych, które ulegały zmianom od chwili uruchomienia trwającego wiele godzin zapytania (mechanizm *read consistency*). Zmiany mogą być wykonywane przez zatwierdzone jak i wycofane transakcje.

Wpływ błędów ORA-1555 oraz błędów związanych z dynamicznym zarządzaniem przestrzenią na wydajność systemu, szczególnie dotkliwy w przypadku napiętego harmonogramu przetwarzania, polega przede wszystkim na znaczącym wydłużeniu czasu potrzebnego do pomyślnego zakończenia operacji. Konieczność wznawiania kosztownych operacji, np. zadań wsadowych, długotrwałych zapytań, powoduje znaczące zwiększenie obciążenia systemu w stosunku do rzeczywistych potrzeb, czego wtórnym efektem bywa postępujące pogarszanie efektywności pozostałych operacji.

Wnioski:

1. *Wraz ze wzrostem skali systemu, każdy problem konfiguracyjny bądź związany z konserwacją systemu może spowodować narastającą lawinę niekorzystnych efektów – włącznie z rozległą utratą możliwości efektywnego wykonywania przetwarzania(efekt domina).*
2. *Warunkiem koniecznym do osiągnięcia wymaganej wydajności jest stworzenie konfiguracji i zapewnienie konserwacji systemu, które są dostosowane do planowanego obciążenia.*

7. Słowo końcowe chociaż nie ostatnie

Uruchomienie systemu, zestrojenie, osiągnięcie wymaganej wydajności – czy to wszystko? Nic z tego. Środowiska rzeczywiste podlegają ciągłym zmianom – jest to reakcja na zmieniające się potrzeby i wymagania. Następuje wzrost rozmiaru bazy danych, do aplikacji są wprowadzane zmiany poszerzające funkcjonalność, zmienia się charakterystyka obciążenia systemu przez wykonywane operacje. Zmiana warunków pracy systemu prędzej czy później pociąga za sobą konieczność ponownego strojenia. Strojenie i zarządzanie wydajnością nie są czynnościami jednorazowymi, lecz procesami [1]. Ich zakończenie może nastąpić wraz z wycofaniem systemu z eksploatacji.

Zapewnienie stałego poziomu wydajności wymaga stałego obserwowania i zarządzania obciążeniem. Konieczne jest ciągłe zarządzanie zasobami, diagnozowanie i rozwiązywanie problemów najlepiej przed wystąpieniem poważniejszych skutków.

Czy tym razem to już naprawdę wszystko, co można powiedzieć na temat skalowania i zarządzania wydajnością systemów? Opracowaniu z pewnością nie przyświecał cel zaprezentowania syntezy, choćby mikroskopijnego wydania „*Summy wydajnościowej w pigułce"*, celem raczej było zasygnalizowanie pewnej problematyki.

Przede wszystkim, sfera technologii jest na tyle złożona, że bezkrytyczne przyjmowanie obiegowych poglądów, jedynie powierzchowna jej znajomość i zaniechanie stałego zgłębiania jej tajników łatwo może sprowadzić na manowce. Technologia jest jednak jedną z wielu warstw omawianej problematyki – jest ona kształtowana przez ludzi, podobnie jak realizowane za jej pomocą projekty i eksploatowane systemy. Projektom i systemom kształt nadają wszyscy zaangażowani ludzie – począwszy od operatorów, poprzez administratorów, programistów i projektantów, na szefach projektów czy firm skończywszy. Kształt nadają ich pragnienia i emocje, ambicje i problemy, motywacje oraz cele, jakie realizują, niezależnie od tego, czy dotyczą one zbiorowości, czy pojedynczych osób. Nie sposób tych sfer rozdzielić i nie sposób którejkolwiek pominąć. Warto im poświęcić od czasu do czasu choćby chwilę głębszej refleksji.

Bibliografia

1. Aronoff E., Loney K., Sonawalla N., Oracle8 Advanced Tuning & Administration, Osborne McGraw-Hill, 1998, ISBN 0-07-882534-2
2. Gunther N., The Practical Performance Analyst, McGraw-Hill, 1998
3. Oracle8i Release 8.1.5 Application Developer's Guide