

XIV Konferencja PLOUG  
Szczyrk  
Październik 2008

# Pomiar obciążenia bazy danych Oracle 10g RAC

Jarosław Przybyszewski  
Accenture Services Sp. z o.o.

*e-mail: jarek.przybyszewski@gmail.com*

Piotr Węśław  
Roche Polska Sp. z o.o.

*e-mail: piotr.weslaw@roche.com*

**Abstrakt.** Pomiar i zarządzanie obciążeniem serwera bazodanowego jest jednym z kluczowych zadań, które umożliwiają świadczenie usług informatycznych w stabilny sposób. Na poziomie bazy danych oraz systemu operacyjnego istnieje szereg narzędzi mierzących pewne statystyki poszczególnych procesów, jednak nie umożliwiają one pomiaru obciążenia serwera z podziałem na poszczególne aplikacje bazodanowe. Niniejsze opracowanie koncentruje się na przedstawieniu sposobu, który wykorzystując dostępne w bazie danych statystyki, umożliwia zmierzenie obciążenia serwera wygenerowanego przez badaną aplikację.

**Informacja o autorach.** Piotr Węśław jest aktualnie pracownikiem farmaceutycznej firmy Roche Polska Sp. z o.o. Jeszcze podczas studiów na Wydziale Elektroniki i Technik Informatycznych Politechniki Warszawskiej zajmował się zagadnieniami związanymi z bazami danych ze szczególnym uwzględnieniem technologii Oracle. Od początku kariery zawodowej skupia się na tematyce dotyczącej współbieżności i strojenia wydajności dużych baz danych Oracle, zarówno hurtowni danych, jak i systemów transakcyjnych.

Jarosław Przybyszewski jest obecnie administratorem baz danych Oracle w firmie Accenture Services. Jego przygoda z Oracle rozpoczęła się w trakcie studiów na Politechnice Warszawskiej, a praca zawodowa prawie od początku była ściśle związana z szeroko rozumianym administrowaniem bazami danych Oracle. Aktualnie zajmuje się przede wszystkim zagadnieniami związanymi z bazami danych działającymi w konfiguracji klastrowej, w tym także optymalizacją zapytań.



## Wstęp

Problem planowania obciążenia generowanego przez hurtownie danych pojawił się razem z ich powstaniem. Ilości danych mogą dochodzić do dziesiątek, a nawet setek terabajtów, a ich analiza nie ogranicza się do pojedynczych rekordów (tak jak w systemach transakcyjnych), a raczej do całych zbiorów danych. Długie czasy przetwarzania czynią tego typu aplikacje wrażliwymi na wszelkie zmiany obciążenia systemu.

W systemach komercyjnych czas przeznaczony na wykonanie danej analizy danych jest ograniczony, a jego przekroczenie powoduje niedotrzymanie warunków świadczenia usług. Ponadto, wydłużenie jednego zadania zwiększa prawdopodobieństwa nieplanowego pokrycia się kilku przetwarzania, co negatywnie wpływa na działanie aplikacji. Aby utrzymać względnie krótkie czasy przetwarzania przy rosnących zbiorach danych, często zwiększane są możliwości serwera (np. przez dostawianie kolejnych procesorów) oraz wprowadzane jest zrównoleglenie przetwarzania przez dołączenie kolejnych maszyn, np. stosując bazę danych Oracle w konfiguracji RAC.

Celem planowania obciążenia serwera jest z jednej strony maksymalne wykorzystanie dostępnego sprzętu, a z drugiej strony dostarczenie określonych usług w określonym czasie. W przypadku hurtowni danych jest to przetworzenie kolejnych paczek informacji w odpowiednim czasie, przy czym czas ten powinien być powtarzalny podczas kolejnych uruchomień. Planowanie obciążenia jest konieczne, ponieważ wraz z jego wzrostem szybkość przetwarzania zadań może zacząć drastycznie spadać po przekroczeniu określonego progu. Aby móc w umiejętny sposób zaplanować obciążenie serwera, potrzebna jest metoda umożliwiająca dokładne zmierzenie obciążenia generowanego przez poszczególne aplikacje. Jedną z możliwych metod została przedstawiona w niniejszym opracowaniu.

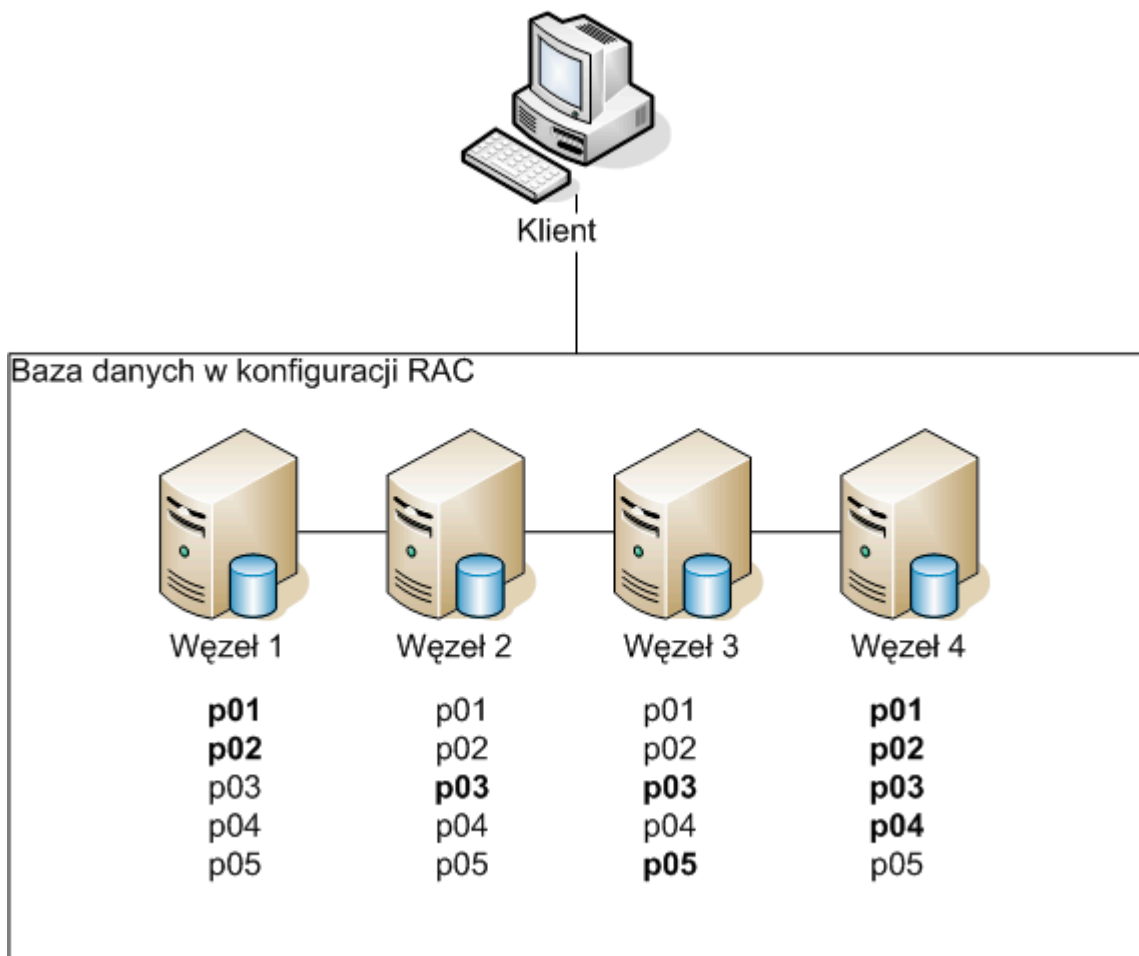
## Podstawowe problemy podczas pomiaru obciążenia

Pomiar obciążenia generowanego przez aplikację w rzeczywistym komercyjnym środowisku może być trudny lub wręcz niemożliwy bez usystematyzowanego podejścia. Wiąże się to z szeregiem problemów, jakie pojawiają się właśnie w tego typu środowiskach, a nie są obecne w warunkach laboratoryjnych. Są one związane ze złożonością samego systemu, dużą liczbą użytkowników ulokowanych w różnych strefach czasowych oraz jednoczesnym wykorzystaniu systemu do różnych celów.

### 2.1. Identyfikacja sesji

Podstawowym problemem jest znalezienie odpowiedzi na pytanie: „Jaki proces lub zbiór procesów serwera wykonuje pracę na potrzeby badanej aplikacji?”

Założmy, że badane środowisko składa się z serwera RAC z 4 węzłami. Każdy z węzłów działa na serwerze wyposażonym w 64 procesory. Dokładna liczba procesorów nie odgrywa większego znaczenia, ale istotne jest, że na serwerze swobodnie może być uruchomiona duża liczba zapytań równoległych.



Rys. 1. Zapytanie równoległe w środowisku RAC

W ogólnym przypadku zapytanie wykonywane jest przez szereg procesów serwera zlokalizowanych na różnych węzłach. Dobór węzłów jest podyktowany bieżącym obciążeniem systemu i nie można go w prosty sposób przewidzieć ani kontrolować. Co więcej, każde zapytanie wykonywane w danej sesji będzie prawdopodobnie wykonywane przy użyciu innego zestawu procesów serwera. Rys.1 przedstawia taką sytuację. Procesy serwera, oznaczone jako p01-p05, znajdują się na każdym z węzłów. Wykonanie konkretnego zapytania realizowane jest przez procesy zaznaczone jako pogrubione. Przy kolejnym zapytaniu zestaw procesów użytych do jego wykonania będzie już inny. Gdyby zapytanie było wykonywane tylko przez jeden proces serwera, możliwe byłoby zidentyfikowanie tego właśnie procesu, który jest tworzony w momencie zestawiania połączenia. Natomiast procesy serwera równoległego typowo nie są tworzone na początku połączenia, a istnieją przez większość czasu, w którym instancja bazy danych jest uruchomiona.

Z powyższych powodów identyfikator sesji klienta (SID) nie nadaje się do zidentyfikowania pojedynczego procesu serwera pracującego na jej potrzeby, ponieważ zadania sesji wykonywane są przez zmienny zbiór procesów serwera. W przypadku zapytań równoległych należałoby znaleźć metodę, która dynamicznie identyfikowałaby procesy serwera, które w danej chwili pracują na potrzeby sesji klienta.

## 2.2. Współdzielone środowisko testowe

Przy tak rozbudowanym i kosztownym rozwiązaniu jak serwer bazy danych pracujący w konfiguracji RAC, całe środowisko jest typowo dzielone przez wiele projektów i aplikacji. Nie ma ekonomicznego uzasadnienia, aby każda aplikacja posiadała dedykowany serwer, który jeszcze dodat-

kowo musiałby być powielony tyle razy, ile środowisk bierze udział w cyklu tworzenia oprogramowania. Najczęściej są to przynajmniej trzy środowiska: deweloperskie, akceptacyjne i produkcyjne. Zakładając cztery węzły na środowisko, dałoby to 12 serwerów dla pojedynczej aplikacji.

Podczas pomiaru obciążenie generowanego przez aplikację, bardzo istotna jest dokładność. Bez usystematyzowanego podejścia łatwo jest wprowadzić błąd poprzez dołączenie obciążenia generowanego przez inne aplikacje lub użytkowników. Pewnym rozwiązaniem byłoby tymczasowe wyłączenie wszystkich aplikacji. Niemniej, przy odpowiednio dużej liczbie projektów prowadzonych jednocześnie jest to niemożliwe, gdyż w takim przypadku na serwerze ciągle byłyby wykonywane pomiary obciążenia i nie byłoby czasu na pozostałe czynności związane z projektem. Dodatkowo zakłócenia wprowadzają użytkownicy, którzy niezależnie od działających aplikacji wykonują od czasu do czasu różne zapytania potrzebne do ich codziennej pracy.

Z powyższych powodów mierzenie obciążenia przy użyciu narzędzi dostarczanych przez system operacyjny nie jest wystarczające, gdyż praktycznie nie jest możliwe odseparowanie obciążenia generowanego tylko przez badaną aplikację. Punktem wyjścia powinien być moment zestawiania połączenia aplikacji z bazą danych, a następnie śledzenie wszelkich aktywności związanych z wykonywaniem zadań tej sesji.

## Metoda pomiaru obciążenia generowanego przez aplikację

Aby móc określić w jakim stopniu dana aplikacja wykorzystuje serwer, trzeba dysponować następującymi informacjami:

- Jakie jest obciążenie wygenerowane przez badaną aplikację ?
- Jakie jest obciążenie wygenerowane przez wszystkie aplikacje?
- Jakie jest całkowite obciążenie wygenerowane przez system operacyjny?
- Jaki jest czas bezczynności?

Wszystkie te informacje powinny być dostępne w funkcji czasu. Umożliwi to stworzenie charakterystyki danej aplikacji, która może następnie zostać użyta do planowania i zarządzania pojemnością serwera.

### 3.1. Pomiar czasu procesora wykorzystanego przez badaną aplikację

Pierwszą czynnością jaką należy wykonać, aby zmierzyć obciążenie generowane przez aplikację jest jednoznaczna identyfikacja sesji i procesów serwera, które są skojarzone z tą właśnie aplikacją. Zapytanie wykonywane jest w trybie równoległym, a poszczególne procesy rozrzucone są po poszczególnych węzłach serwera RAC.

Potrzebny jest więc mechanizm, który będzie oznaczał procesy serwera pracujące na potrzeby danej aplikacji tylko w okresie, w którym rzeczywiście wykonywane jest zapytanie. Procedura `DBMS_SESSION.SET_IDENTIFIER()` umożliwia oznaczenie wszystkich sesji (w tym serwera równoległego) pracujących nad wykonaniem zapytania.

```
SQL> select distinct sid from v$mystat;
      SID
-----
      148

SQL> exec dbms_session.set_identifier('APPLICATION1');
PL/SQL procedure successfully completed.

SQL> select sid,client_identifier from v$session where sid=148;
```

```
SID CLIENT_IDENTIFIER
```

```
-----
148 APPLICATION1
```

W powyższym przykładzie bieżąca sesja (SID=148) została oznaczona identyfikatorem APPLICATION1. W przypadku, gdyby zostało uruchomione zapytanie równoległe, wszystkie procesy serwera równoległego zostałyby oznaczone identyfikatorem na czas wykonania zapytania.

Drugim krokiem jest uruchomienie zbierania różnorodnych statystyk dla sesji oznaczonych konkretnym identyfikatorem. Umożliwia to procedura DBMS\_MONITOR.CLIENT\_ID\_STAT\_ENABLE(). Statystyki składają się z podzbioru statystyk sesji dostępnych standardowo poprzez widok V\$SESSTAT.

```
SQL> exec dbms_monitor.client_id_stat_enable('APPLICATION1');
```

```
PL/SQL procedure successfully completed.
```

Statystyka, która jest ważna z punktu widzenia mierzenia obciążenia serwera to “DB CPU”. Określa ona czas procesora wykorzystany przez wszystkie procesy (również serwera równoległego zlokalizowane na różnych węzłach serwera RAC) realizujące zapytania aplikacji z danym identyfikatorem. Jest to czas procesora wyrażony w mikrosekundach, który można obejrzeć przy użyciu widoku V\$CLIENT\_STATS. Należy podkreślić, że wartości statystyk są kumulacyjne i nie są niestety zapisywane przez AWR do repozytorium. Z tego powodu trzeba regularnie (np. raz na godzinę) kopiować je do tabeli tymczasowej, aby móc później obliczyć ile czasu procesora zostało wykorzystane przez aplikacje w określonym przedziale czasowym.

```
SQL> select client_identifier, stat_name, value
2 from v$client_stats where stat_name='DB CPU';
```

| CLIENT_IDENTIFIER | STAT_NAME | VALUE |
|-------------------|-----------|-------|
| APPLICATION1      | DB CPU    | 71936 |

Zarówno włączenie identyfikacji sesji należących do danej aplikacji, jak i zbieranie statystyk można łatwo zaimplementować przy pomocy wyzwalacza uruchamianego podczas logowania do bazy danych, a regularne zapisywanie wartości statystyki “DB CPU” jako zadanie przy użyciu pakietu DBMS\_JOB lub DBMS\_SCHEDULER.

Na tym etapie dostępna jest informacja mówiąca o tym, ile czasu procesora zostało wykorzystane przez daną aplikację, uwzględniając zapytania równoległe działające w środowisku serwera RAC.

### 3.2. Pomiar czasu wykorzystanego przez wszystkie aplikacje

Aby móc mówić o procentowym wykorzystaniu serwera przez aplikację, potrzeba jest informacja o czasie wykorzystanym przez wszystkie aplikacje działające na badanej bazie danych oraz system operacyjny. Dzięki AWR informacje takie dostępne są bezpośrednio w bazie danych przez następujące widoki:

- DBA\_HIST\_SYSSSTAT – dane historyczne szeregu statystyk na poziomie bazy danych, m.in. całkowity czas procesora wykorzystany przez bazę danych.
- DBA\_HIST\_OSSTAT – dane historyczne szeregu statystyk na poziomie systemu operacyjnego, m.in. czas rzeczywisty wykorzystany przez wszystkie aplikacje i system operacyjny.
- DBA\_HIST\_SNAPSHOT – dane o migawkach bazy danych znajdujących się w repozytorium AWR. Zawierają one identyfikator migawki oraz przedział czasowy, którego ona dotyczy. Standardowo jest to jedna godzina.

We wszystkich powyższych widokach dostępna jest kolumna INSTANCE\_NUMBER, która określa numer węzła serwera RAC. Jeśli baza danych współpracuje tylko z jedną instancją, wartość ta będzie zawsze wynosiła 1.

Czas procesora wykorzystany przez bazę danych i wszystkie działające na niej aplikacje można uzyskać przy użyciu poniższych zapytań. Nazwa statystyki „CPU used by this session” jest myląca i w tym wypadku oznacza czas procesora dla całej instancji bazy danych, a nie tylko bieżącej sesji. Jednostka czasu procesora jest inna niż w przypadku widoku V\$CLIENT\_STATS i jest wyrażona dziesiątkach milisekund. Wartości statystyki są kumulacyjne, więc ważne jest, aby wiedzieć w jakich momentach w przeszłości zostały one zapisane – można to sprawdzić korzystając z widoku DBA\_HIST\_SNAPSHOT.

```
SQL> select snap_id, instance_number, value from dba_hist_sysstat
2  where stat_name='CPU used by this session'
3  order by snap_id;
```

| SNAP_ID | INSTANCE_NUMBER | VALUE |
|---------|-----------------|-------|
| 62      | 1               | 978   |
| 63      | 1               | 1876  |
| 64      | 1               | 2816  |
| 65      | 1               | 3587  |
| 66      | 1               | 4359  |
| 67      | 1               | 1536  |
| 68      | 1               | 3455  |
| 69      | 1               | 4736  |
| 70      | 1               | 5952  |

9 rows selected.

```
SQL> select snap_id,begin_interval_time, end_interval_time
2  from DBA_HIST_SNAPSHOT where snap_id=65;
```

| SNAP_ID | BEGIN_INTERVAL_TIME   | END_INTERVAL_TIME     |
|---------|-----------------------|-----------------------|
| 65      | 08/07/26 18:00:54,921 | 08/07/26 19:00:10,328 |

Na tym etapie dostępna jest informacja zarówno o czasie procesora wykorzystanym przez badaną aplikację (punkt 3.1), jak i bazę danych wraz ze wszystkimi aplikacjami, z podziałem na poszczególne węzły serwera RAC. Można więc łatwo policzyć jaki procent obciążenia wygenerowanego przez całą bazę należy do badanej aplikacji (w danym okresie czasu – w tym przypadku jest to jedna godzina). Wartości statystyk są kumulacyjne, więc należy operować na ich przyrostach w danym okresie czasu.

$$P_{\text{aplikacji}} = \frac{\text{czas\_procesora\_badanej\_aplikacji}}{\text{czas\_procesora\_całej\_bazy\_danych}}$$

Należy zaznaczyć, że statystyki dotyczące bazy danych pochodzące z widoku DBA\_HIST\_SYSSTAT wyrażone są w jednostkach czasu procesora, natomiast miary dotyczące systemu operacyjnego pochodzące z widoku DBA\_HIST\_OSSTAT w jednostkach czasu rzeczywistego. Z tego powodu nie można obu tych wartości porównać bezpośrednio.

### 3.3. Pomiar czasu na poziomie systemu operacyjnego

Kolejnym krokiem jest porównanie czasu rzeczywistego wykorzystanego przez całą bazę danych do całkowitego czasu rzeczywistego wykorzystanego przez serwer oraz czasu bezczynności (na poziomie systemu operacyjnego). Jak zostało już powiedziane wcześniej w niniejszym opracowaniu, w wersji 10g wprowadzony został dostęp do statystyk systemu operacyjnego z poziomu bazy danych przez zestaw widoków, które można uzyskać przy pomocy poniższego zapytania:

```
SQL> select snap_id, instance_number, stat_name, value
2   from dba_hist_osstat
3   where stat_id in (1,2,3)
4   order by snap_id, stat_id;
```

| SNAP_ID | INSTANCE_NUMBER | STAT_NAME | VALUE   |
|---------|-----------------|-----------|---------|
| 62      | 1               | IDLE_TIME | 116319  |
| 62      | 1               | BUSY_TIME | 11178   |
| 62      | 1               | USER_TIME | 5186    |
| 63      | 1               | IDLE_TIME | 672801  |
| 63      | 1               | BUSY_TIME | 28719   |
| 63      | 1               | USER_TIME | 10641   |
| 64      | 1               | IDLE_TIME | 1376627 |
| 64      | 1               | BUSY_TIME | 48592   |
| 64      | 1               | USER_TIME | 16519   |
| 65      | 1               | IDLE_TIME | 2073499 |
| 65      | 1               | BUSY_TIME | 62779   |
| 65      | 1               | USER_TIME | 22443   |
| 66      | 1               | IDLE_TIME | 2784701 |
| 66      | 1               | BUSY_TIME | 73475   |
| 66      | 1               | USER_TIME | 25921   |

15 rows selected.

Znaczenie poszczególnych statystyk jest następujące:

- IDLE\_TIME – czas rzeczywisty (wyrażony w setnych sekundy), w którym nie były wykonywane żadne zadania
- BUSY\_TIME – czas rzeczywisty (wyrażony w setnych sekundy) wykorzystany przez bazę danych oraz system operacyjny.
- USER\_TIME – czas rzeczywisty (wyrażony w setnych sekundy) wykorzystany tylko i wyłącznie przez bazę danych. We wstępie zostało założone, że na serwerze oprócz systemu operacyjnego i instancji bazy danych nie ma uruchomionych innych aplikacji, co jest standardową konfiguracją w przypadku serwera RAC.

Procent czasu wykorzystanego przez bazę danych do całkowitego czasu, który upłynął można obliczyć przy pomocy poniższego wzoru. W tym wypadku wartości statystyk są również kumulacyjne, więc należy operować na ich przyrostach w danym okresie czasu.

$$P_{bazy\_danych} = \frac{\text{czas\_rzeczywisty\_bazy\_danych}}{\text{całkowity\_czas\_rzeczywisty}} = \frac{\Delta USER\_TIME}{\Delta BUSY\_TIME + \Delta IDLE\_TIME}$$

### 3.4. Obciążenie serwera

Dysponując powyższymi wartościami łatwo jest obliczyć (w procentach możliwości serwera) obciążenie wygenerowane przez badaną aplikację:

$$P = P_{bazy\_danych} * P_{aplikacji} * 100\%$$

Uzyskana wartość określa obciążenie serwera przez badaną aplikację w danej jednostce czasu. Aby mieć pełny obraz charakterystyki aplikacji, należy podzielić cały okres jej działania na kwanty czasu i przeprowadzić przedstawione obliczenia dla każdego z nich. W ramach danego przedziału czasowego wartość obciążenia jest uśredniona. Przykładowo, jeśli  $p=10\%$  w przeciągu godziny, nie oznacza to, że aplikacja w sposób jednostajny generuje obciążenie na poziomie  $10\%$ . Prawdopodobnie są momenty ze znacznie większym i mniejszym obciążeniem, ale średnio jest to  $10\%$ . Aby zmniejszyć kwant czasu dla którego wykonywane są obliczenia, należy zmienić częstotliwość tworzenia migawek przez AWR przy użyciu procedury `DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS()`. Poniższe polecenie ustawia częstotliwość tworzenia migawek co 15 minut.

```
SQL> exec dbms_workload_repository.modify_snapshot_settings (interval => 15);
PL/SQL procedure successfully completed.
```

```
SQL> select snap_interval
       2 from dba_hist_wr_control;
```

```
SNAP_INTERVAL
-----
+000000 00:15:00.0
```

## Mierzenie zużycia zasobów w czasie rzeczywistym

W poprzednim rozdziale opisano infrastrukturę dostępną razem z dystrybucją Oracle'a, która może być wykorzystana jako baza do zbudowania systemu umożliwiającego mierzenie zużycia zasobów w czasie rzeczywistym. W skład całej infrastruktury wchodzi szereg kluczowych komponentów, które umożliwiają szerokie możliwości analizy. Najłatwiejszym sposobem przedstawienia całego rozwiązania będzie przeanalizowanie go na przykładzie. Na potrzeby niniejszego artykułu przyjęte zostaną następujące założenia:

- badanym systemem jest hurtownia danych działająca w środowisku Oracle 10g RAC w trybie 24x7 przez 365 dni w roku,
- w systemie można wyróżnić dwie grupy aktywności: „ładowanie” danych i generowanie raportów na żądanie użytkowników,
- ładowanie odbywa się za pośrednictwem bazodanowego użytkownika APP\_ETL,
- użytkownicy generują raporty korzystając z aplikacji, która łączy się do bazy danych za pośrednictwem użytkownika APP\_REPORT,
- celem pomiaru jest aktywność związana z ładowaniem danych,
- mierzonym zasobem jest procentowe zużycie procesora (procesorów) na poszczególnych węzłach systemu.

Założenia te wynikają z charakteru hurtowni danych, sposobu ich działania oraz problemów występujących w środowiskach, w których przez długie okresy czasu system jest silnie obciążony przez procesy ładujące dane, a ponadto cały czas działają użytkownicy korzystający z dostępnych

informacji. Mając na uwadze powyższe uwagi w następnej kolejności przedstawiono kluczowe elementy systemu umożliwiające pomiar zużycia zasobów.

#### 4.1. Przygotowanie aplikacji i systemu

Korzystając z dostępnych w Oracle’u 10g pakietów należy odpowiednio zmodyfikować parametry AWR’a tak, aby pomiar był dokładniejszy oraz żeby możliwa była analiza długoterminowa:

- częstotliwość wykonywania migawek – im częściej są wykonywane migawki tym dokładniejsze pomiary można uzyskać. Oczywiście jest to pewien kompromis pomiędzy dokładnością pomiaru a zajętością przestrzeni dyskowej koniecznej do przechowywania większej ilości danych.
- okres przechowywania migawek – zależnie od potrzeb można go wydłużyć lub skrócić, co wpływa na zakres czasu, który będzie później dostępny do analizy. Także i w tym przypadku należy wziąć pod uwagę dostępną ilość przestrzeni dyskowej i dostosować parametry do konkretnych możliwości i wymagań.

W przypadku testowanego systemu ustawiono parametry w następujący sposób:

- Godzinny okres wykonywania migawek (jest to ustawienie domyślne AWR’a):

```
SQL> exec dbms_workload_repository.modify_snapshot_settings(interval=>60)
```

- Piętnasto dniowy okres przechowywania migawek:

```
SQL> exec dbms_workload_repository.modify_snapshot_settings(retention=>21600)
```

Wartości ustawionych parametrów można sprawdzić w widoku DBA\_HIST\_WR\_INTERVAL:

```
SQL> SELECT SNAP_INTERVAL, RETENTION FROM DBA_HIST_WR_INTERVAL;
```

Następnym koniecznym elementem do ustawienia jest identyfikator klienta, czyli w opisywanym przypadku części aplikacji odpowiedzialnej za ładowanie danych lub generowanie raportów. Istnieją tutaj dwie możliwości – albo w trakcie tworzenia aplikacji są dodawane odpowiednie wywołania procedur dostępnych w bazie danych, albo konieczne jest stworzenie odpowiedniego wyzwalacza. W większości wypadków będzie to druga opcja, gdyż programiści rzadko korzystają z tej możliwości w trakcie tworzenia oprogramowania. Przykładowy kod wyzwalacza wykorzystanego w trakcie testów znajduje się poniżej:

```
SQL> CREATE OR REPLACE TRIGGER ON_LOGON
 2 AFTER LOGON ON SCHEMA
 3 BEGIN
 4   DBMS_SESSION.SET_IDENTIFIER('APP_ETL');
 5 END;
```

Kluczowym elementem tutaj jest wybór identyfikatora – z oczywistych względów musi on być inny dla każdej aplikacji lub każdego modułu, który ma być mierzony. Po dokonaniu powyższych modyfikacji można przystąpić do testowania.

#### 4.2. Testowanie aplikacji

Po przygotowaniu aplikacji można rozpocząć właściwe pomiary. Oracle domyślnie nie zbiera danych dotyczących wydajności aplikacji. Włączanie zbierania danych następuje przez wywołanie następującej procedury:

```
SQL> exec dbms_monitor.client_id_stat_enable('APP_ETL')
```

a wyłączanie zbierania danych za pomocą procedury:

```
SQL> exec dbms_monitor.client_id_stat_disable('APP_ETL');
```

Samo włączenie zbierania danych dotyczących danej aplikacji powoduje jedynie możliwość ich przeglądania w widoku GV\$CLIENT\_STATS. Dlatego konieczne było stworzenie dodatkowych elementów umożliwiających praktyczne wykorzystanie informacji dostępnych w tym widoku. Na początek należy stworzyć dodatkowe repozytorium danych z tego widoku na przykład w następujący sposób:

```
SQL> CREATE TABLE STATS$CLIENT_STATS
 2 AS
 3 SELECT sysdate time_stamp, v.*
 4 FROM GV$CLIENT_STATS v
 5 WHERE rownum < 1;
```

Dane do tej tabeli należy regularnie ładować. W tym celu wystarczy zdefiniować własną procedurę oraz zadanie. Procedura powinna ładować dane do repozytorium, a zadanie powinno wywoływać tę procedurę. Całość kodu przedstawiona jest poniżej:

```
SQL> CREATE OR REPLACE procedure client_stats
 2 is
 3 BEGIN
 4     execute immediate 'insert into stats$client_stats
 5                       select sysdate time_stamp, v.*
 6                       from gv$client_stats v';
 7     commit;
 8 END;

SQL> begin
 2     DBMS_JOB.SUBMIT(:jobno, 'begin client_stats; end;',
 3                       to_date(SYSDATE+10, 'DD/MM/YYYY HH24:MI:SS'),
 4                       'TRUNC(SYSDATE + 10/1440, 'MI')');
 5     commit;
 6* end;
```

Oczywiście po zakończonych testach można wyłączyć lub skasować zadanie a dane z repozytorium przenieść do archiwum. Zadanie można także zdefiniować przy użyciu pakietu DBMS\_SCHEDULER dostępnego począwszy od wersji 10g. Tak przygotowana „infrastruktura” testowa umożliwi zebranie informacji niezbędnych do dalszej analizy.

### 4.3. Analiza danych

W celu analizy danych stworzono trzy widoki, których głównym celem jest zebranie w jednym wierszu danych dotyczących zużycia zasobów z aktualnie przetwarzanego i poprzedniego okresu (migawki). Dzięki temu w następnym kroku można policzyć różnicę między tymi wartościami, a co za tym idzie procentowe zużycie czasu procesora przez aplikację w danej jednostce czasu. Kod widoków znajduje się w sekcji załączników, a poniżej znajduje się opis kolumn każdego z widoków i jego krótki opis.

#### OS\_STATS\_VW

Widok OS\_STATS\_VW jest oparty na widoku udostępnianym przez Oracle 10g DBA\_HIST\_OSSTAT. Jego głównym celem jest zebranie danych z dwóch kolejnych migawek w jednym wierszu, co umożliwia policzenie różnicy między wartościami na początku i na końcu badanego przedziału czasu. Widok ten umożliwia policzenie ile procent czasu procesora zostało

zużyte przez bazę danych, ile czasu procesor był beczynny i jaki był całkowity czas procesora zużyty przez wszystkie aplikacje działające na serwerze.

| OS_STATS_VW     |        |                                                                                     |
|-----------------|--------|-------------------------------------------------------------------------------------|
| Nazwa           | Typ    | Opis                                                                                |
| SNAP_ID         | NUMBER | Identyfikator migawki                                                               |
| INSTANCE_NUMBER | NUMBER | Numer instancji                                                                     |
| USER_TIME       | NUMBER | Czas procesora zużyty przez właściciela bazy danych                                 |
| BUSY_TIME       | NUMBER | Całkowity czas procesora zużyty przez wszystkich użytkowników                       |
| IDLE_TIME       | NUMBER | Niewykorzystany czas procesora                                                      |
| USER_TIME_PREV  | NUMBER | Czas procesora zużyty przez właściciela bazy danych z poprzedniej migawki           |
| BUSY_TIME_PREV  | NUMBER | Całkowity czas procesora zużyty przez wszystkich użytkowników z poprzedniej migawki |
| IDLE_TIME_PREV  | NUMBER | Niewykorzystany czas procesora z poprzedniej migawki                                |

### SYS\_STATS\_VW

Widok SYS\_STATS\_VW jest oparty na widokach DBA\_HIST\_SYSSTAT i DBA\_HIST\_SNAPSHOT. Zawiera informacje, których źródłem jest widok V\$SYSSTAT dotyczące zużycia procesora przez bazę danych od startu instancji. Widok SYS\_STATS\_VW zawiera w każdym wierszu informacje z dwóch kolejnych migawek, co umożliwia policzenie zużycia procesora w danej jednostce czasu.

| SYS_STATS_VW      |              |                                                                                  |
|-------------------|--------------|----------------------------------------------------------------------------------|
| Nazwa             | Typ          | Opis                                                                             |
| INSTANCE_NUMBER   | NUMBER       | Numer instancji                                                                  |
| SYS_STAT_NAME     | VARCHAR2(64) | Nazwa statystyki                                                                 |
| SNAP_ID           | NUMBER       | Identyfikator migawki                                                            |
| END_INTERVAL_TIME | DATE         | Czas na koniec wykonywania migawki, czas wykonania migawki                       |
| TOTAL_CPU_MILI    | NUMBER       | Ilość czasu procesora zużyta przez instancję od jej startu                       |
| PREV_SYS_MILI     | NUMBER       | Ilość czasu procesora zużyta przez instancję od jej startu z poprzedniej migawki |

### CLIENT\_STATS\_VW

Ostatnim widokiem jest CLIENT\_STATS\_VW, którego źródłem danych jest repozytorium zawierające informacje dotyczące zużycia procesora przez badaną aplikację. Podobnie jak w przypadku poprzednich dwóch widoków każdy wiersz zawiera dane z dwóch kolejnych migawek, co umożliwia policzenie zużycia procesora przez aplikację w jednostce czasu.

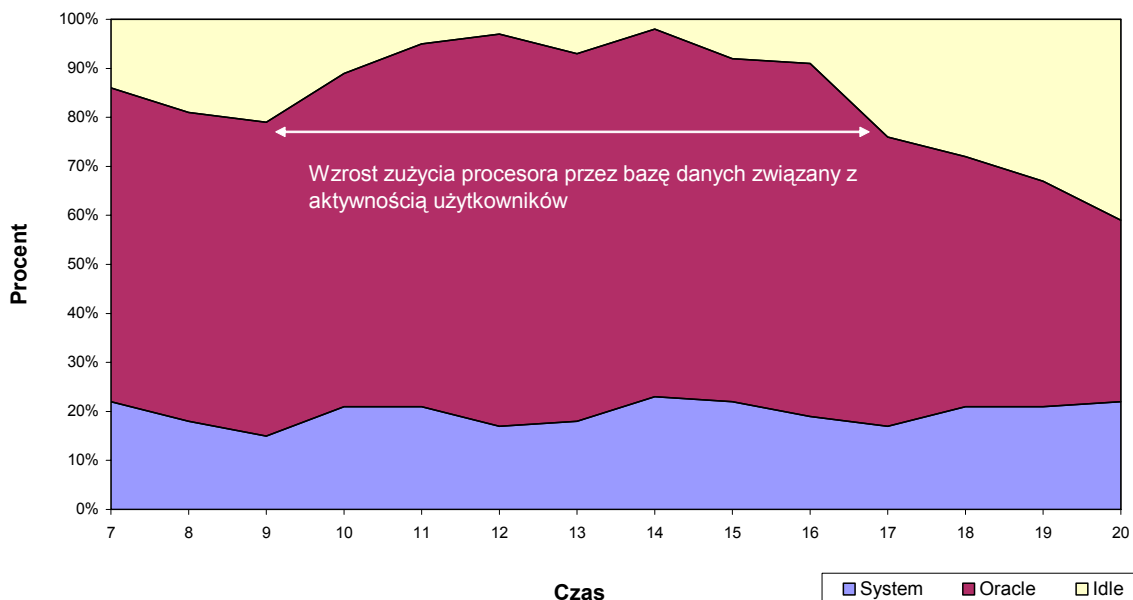
| CLIENT_STATS_VW   |              |                                                                                                  |
|-------------------|--------------|--------------------------------------------------------------------------------------------------|
| Nazwa             | Typ          | Opis                                                                                             |
| TIME_STAMP        | DATE         | Data wykonania migawki (przez zadanie zdefiniowane przez administratora)                         |
| INST_ID           | NUMBER       | Numer instancji                                                                                  |
| CLI_STAT_NAME     | VARCHAR2(64) | Nazwa statystyki                                                                                 |
| CPU_UTIL_CLI_MILI | NUMBER       | Ilość czasu procesora zużyta przez aplikację klienta od początku pomiaru                         |
| PREV_CLI_MILI     | NUMBER       | Ilość czasu procesora zużyta przez aplikację klienta od początku pomiaru (z poprzedniej migawki) |

Mając do dyspozycji te widoki można za pomocą jednego zapytania uzyskać następujące informacje:

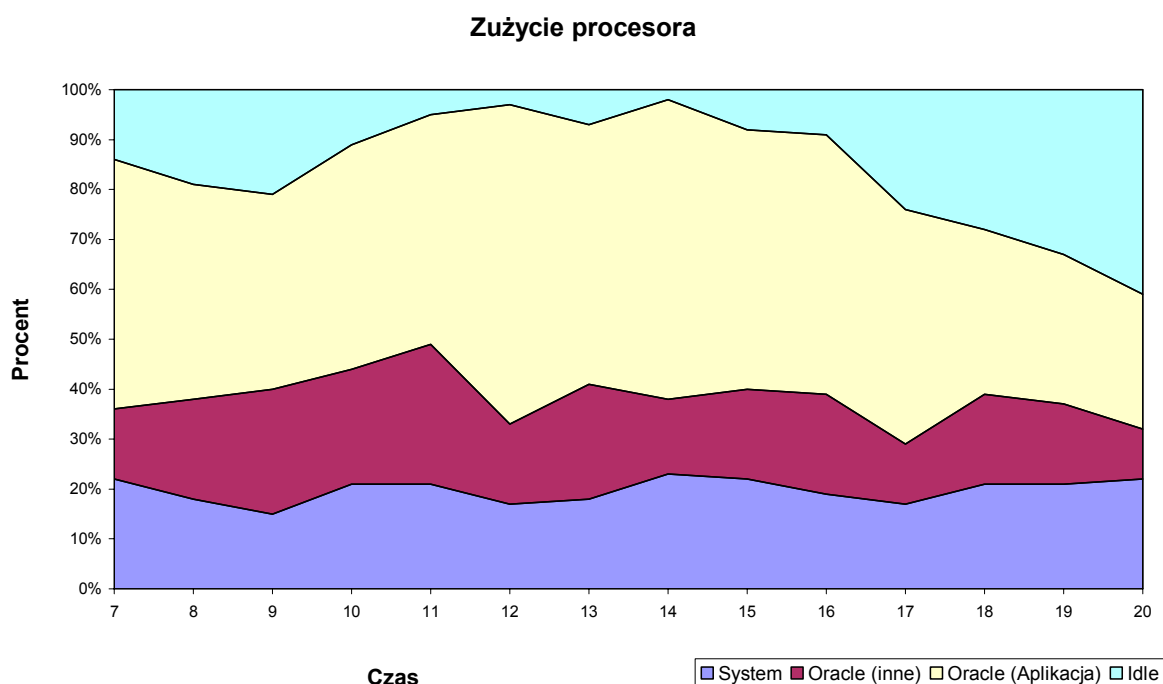
- Całkowite zużycie procesora w systemie przez wszystkie aplikacje
- Zużycie procesora przez bazę danych
- Zużycie procesora przez badaną aplikację lub aplikacje

Zapytanie, które umożliwi uzyskanie tych informacji znajduje się w sekcji załączników i nie będzie przedmiotem dokładnej analizy, gdyż w całości opiera się na prostych obliczeniach, które były dokładniej dyskutowane w poprzednim rozdziale. W omówionym przykładzie badanym za sobem był czas procesora, ale przy użyciu tej metody (i przy drobnych modyfikacjach widoków) jest możliwe zastosowanie jej do mierzenia innych parametrów działania systemu, np. ilość operacji dyskowych, liczbę sortowań. Przykładowe wykresy możliwe do uzyskania za pomocą opisanej metody znajdują się poniżej.

### Zużycie procesora



Rys. 2. Zużycie procesora na serwerze bazodanowym



Rys. 3. Zużycie procesora na serwerze bazodanowym z wyróżnieniem badanej aplikacji

#### 4.4. Błędy pomiaru

W przypadku stosowania tej metody należy pamiętać, że jest ona obarczona pewnym błędem i może być wykorzystana tylko w ściśle określonych warunkach, o których była mowa. Błędy wynikają przede wszystkim z tego, że parametry działania systemu są zbierane okresowo, ponadto w trakcie obliczeń podlegają kolejnym przekształceniom – uśrednianiu w jednostce czasu, co może powodować trudności w uchwyceniu krótkotrwałych „skoków” zużycia badanego zasobu. Skracanie czasu próbkowania powoduje wzrost dokładności w obliczeniach, ale ze względów technicznych nie jest możliwe tworzenie migawek w dowolnie krótkich odstępach czasu.

### Podsumowanie

Metoda opisana w niniejszym opracowaniu sprawdza się idealnie w połączeniu z innymi narzędziami służącymi do mierzenia wydajności i zużycia zasobów systemu dostępnych wraz z każdym systemem operacyjnym, np.: perfmon (Windows), sar (Linux/Unix), perfview (HP-UX). Zastosowanie tej metody pozwala dużo dokładniej szacować zużycie zasobów serwera i lepiej estymować jego wzrost, co w efekcie prowadzi do stabilnego dostarczania usług. W opracowaniu przedstawiono sposób pomiaru zużycia procesora przez aplikację, ale przy niewielkich modyfikacjach możliwe jest zbieranie danych na temat innych zasobów np. ilość operacji dyskowych. Dzięki temu możliwe jest jej zastosowanie do bardziej kompleksowych pomiarów wykorzystania serwera, gdyż samo zużycie procesora nie jest wystarczające do określenia całkowitego wpływu aplikacji na system.

Stosując opisaną metodę należy wziąć pod uwagę jej ograniczenia i warunki w jakich może ona być zastosowana. Nie jest możliwe jej zastosowanie na serwerach, na których oprócz bazy danych działają jeszcze inne aplikacje, gdyż może to w znaczący sposób zniekształcić uzyskiwane rezultaty. Jednakże mimo pewnych niedoskonałości może ona być z pewnością idealnym uzupełnieniem lub narzędziem weryfikacji innych metod stosowanych w trakcie tworzenia aplikacji.

## Załączniki

W sekcji załączników znajduje się kod widoków oraz przykładowe zapytanie umożliwiające pomiary zużycia procesora przez badaną aplikację.

### SYS\_STATS\_VW

```

CREATE OR REPLACE VIEW CLIENT_STATS_VW
AS
  SELECT TRUNC(TIME_STAMP-1/24,'HH24') TIME_STAMP,
         INST_ID                      ,
         STAT_NAME CLI_STAT_NAME      ,
         TRUNC(VALUE      /1000) CPU_UTIL_CLI_MILI      ,
         LAG(TRUNC(VALUE/1000),1,0) OVER
         (ORDER BY INST_ID,TIME_STAMP) AS PREV_CLI_MILI
         FROM STATS$CLIENT_STATS
         WHERE VALUE > 0
         AND STAT_NAME = 'DB CPU'
         AND TO_CHAR(TIME_STAMP,'HH24:MI') LIKE ('__:0_')
ORDER BY INST_ID,
         TRUNC(TIME_STAMP,'HH24') CREATE
OR REPLACE VIEW SYS_STATS_VW AS
  SELECT HSYS.INSTANCE_NUMBER          ,
         HSYS.STAT_NAME SYS_STAT_NAME  ,
         HSYS.SNAP_ID                  ,
         TRUNC(SNAP.END_INTERVAL_TIME,'HH24') END_INTERVAL_TIME,
         HSYS.VALUE *10 TOTAL_CPU_MILI      ,
         LAG(HSYS.VALUE*10,1,0) OVER
         (ORDER BY HSYS.INSTANCE_NUMBER,END_INTERVAL_TIME) AS PREV_SYS_MILI
         FROM DBA_HIST_SYSSTAT HSYS,
         (SELECT SNAP_ID                ,
              END_INTERVAL_TIME          ,
              INSTANCE_NUMBER
              FROM DBA_HIST_SNAPSHOT
              WHERE END_INTERVAL_TIME >=
                 TO_DATE('2008-07-16 10:32:00','YYYY-MM-DD HH24:MI:SS')
         ) SNAP
         WHERE STAT_NAME                  = 'CPU USED BY THIS SESSION'
         AND HSYS.SNAP_ID                  = SNAP.SNAP_ID
         AND HSYS.INSTANCE_NUMBER          = SNAP.INSTANCE_NUMBER
ORDER BY HSYS.SNAP_ID,
         INSTANCE_NUMBER;

```

### OS\_STATS\_VW

```

CREATE OR REPLACE VIEW OS_STATS_VW
AS
  SELECT SNAP_ID,
         INSTANCE_NUMBER,
         USER_TIME,
         BUSY_TIME,
         IDLE_TIME,
         LAG(USER_TIME,1,0) OVER
         (ORDER BY INSTANCE_NUMBER,SNAP_ID) AS USER_TIME_PREV,
         LAG(BUSY_TIME,1,0)
         OVER (ORDER BY INSTANCE_NUMBER,SNAP_ID) AS BUSY_TIME_PREV,

```

```

LAG (IDLE_TIME,1,0)
  OVER (ORDER BY INSTANCE_NUMBER,SNAP_ID) AS IDLE_TIME_PREV
FROM
(SELECT SNAP_ID,
  INSTANCE_NUMBER,
  MAX(
    CASE WHEN STAT_NAME = 'USER_TIME' THEN VALUE ELSE 0 END) USER_TIME,
  MAX(
    CASE WHEN STAT_NAME = 'BUSY_TIME' THEN VALUE ELSE 0 END) BUSY_TIME,
  MAX(
    CASE WHEN STAT_NAME = 'IDLE_TIME' THEN VALUE ELSE 0 END) IDLE_TIME
  FROM DBA_HIST_OSSTAT
  WHERE STAT_ID IN (1,2,3)
  GROUP BY INSTANCE_NUMBER, SNAP_ID)
ORDER BY INSTANCE_NUMBER, SNAP_ID;

```

## CLIENT\_STATS\_VW

```

CREATE OR REPLACE VIEW CLIENT_STATS_VW
AS
  SELECT TRUNC (TIME_STAMP-1/24,'HH24') TIME_STAMP,
  INST_ID,
  STAT_NAME CLI_STAT_NAME,
  TRUNC (VALUE /1000) CPU_UTIL_CLI_MILI,
  LAG (TRUNC (VALUE/1000),1,0) OVER
    (ORDER BY INST_ID,TIME_STAMP) AS PREV_CLI_MILI
  FROM STATS$CLIENT_STATS
  WHERE VALUE > 0
  AND STAT_NAME = 'DB CPU'
  AND TO_CHAR (TIME_STAMP, 'HH24:MI') LIKE ('__:0_')
  ORDER BY INST_ID,
  TRUNC (TIME_STAMP, 'HH24');

```

## Zapytanie

```

SELECT TO_CHAR (CLI.TIME_STAMP-1/24, 'DD/MM/YYYY HH24:MI') TIME_STAMP,
  CLI.INST_ID,
  (CLI.CPU_UTIL_CLI_MILI - CLI.PREV_CLI_MILI) AS INCR_CLI_MILI,
  (SYS.TOTAL_CPU_MILI - SYS.PREV_SYS_MILI) AS INCR_SYS_MILI,
  TO_CHAR ((OSST.BUSY_TIME - OSST.BUSY_TIME_PREV) /
    ((OSST.BUSY_TIME - OSST.BUSY_TIME_PREV) +
    (OSST.IDLE_TIME - OSST.IDLE_TIME_PREV)) * 100, '999.99') PERC_SERV_UTIL,
  TO_CHAR ((OSST.USER_TIME - OSST.USER_TIME_PREV) /
    ((OSST.BUSY_TIME - OSST.BUSY_TIME_PREV) +
    (OSST.IDLE_TIME - OSST.IDLE_TIME_PREV)) * 100, '999.99') PERC_ORCL_UTIL,
  TO_CHAR ((CLI.CPU_UTIL_CLI_MILI - CLI.PREV_CLI_MILI) /
    (SYS.TOTAL_CPU_MILI - SYS.PREV_SYS_MILI) *
    (OSST.USER_TIME - OSST.USER_TIME_PREV) /
    ((OSST.BUSY_TIME - OSST.BUSY_TIME_PREV) +
    (OSST.IDLE_TIME - OSST.IDLE_TIME_PREV)) * 100, '999.99') PERC_CLIENT_UTIL,
  TO_CHAR ((CLI.CPU_UTIL_CLI_MILI - CLI.PREV_CLI_MILI) /
    (SYS.TOTAL_CPU_MILI - SYS.PREV_SYS_MILI) * 100, '999.99') PERC_CLI_ORA_UTIL
FROM
  CLIENT_STATS_VW CLI,
  SYS_STATS_VW SYS,
  OS_STATS_VW OSST

```

---

```
WHERE CLI.TIME_STAMP = SYS.END_INTERVAL_TIME
      AND CLI.INST_ID      = SYS.INSTANCE_NUMBER
      AND (CLI.CPU_UTIL_CLI_MILI - CLI.PREV_CLI_MILI) /
          (SYS.TOTAL_CPU_MILI - SYS.PREV_SYS_MILI) * 100 BETWEEN 0 AND 100
      AND OSST.SNAP_ID      = SYS.SNAP_ID
      AND OSST.INSTANCE_NUMBER = SYS.INSTANCE_NUMBER
ORDER BY CLI.INST_ID, CLI.TIME_STAMP
```

