Implementing Instance Caging (Doc ID 1177593.1)

In this Document

APPLIES TO:

Oracle Database - Enterprise Edition
Information in this document applies to any platform.

PURPOSE

This bulletin includes a feature called Instance Caging. Instance Caging allows the DBA to limit the CPU usage of an Oracle instance by setting the CPU_COUNT initialization parameter and enabling CPU resource management. With Instance Caging, users can partition CPU resources among multiple instances running on a server to ensure predictable performance.

SCOPE

This document is intended for all Oracle DBAs and database support staff.

DETAILS

Using instance caging requires only two simple configurations:

* Enable the resource manager by assigning a resource plan (e.g. DEFAULT_PLAN)

o enable instance caging.

Do the following for each instance on the server:

1.Enable the Resource Manager by assigning a resource plan, and ensure that the resource plan has CPU directives, using the mgmt_p1 through mgmt_p8 parameters.

2.Set the cpu_count initialization parameter.

This is a dynamic parameter, and can be set with the following statement:

ALTER SYSTEM SET CPU_COUNT = 4;

To illustrate how it works, here is a test using a small test server with 4 cores.

To burn all the available CPU resources, We started four sessions executing the following PL/SQL block:

```
DECLARE
n NUMBER;
BEGIN
WHILE (TRUE)
LOOP
n:= dbms_random.random();
END LOOP;
END;
```

With the four sessions up and running, the output of vmstat(8) looks like the following:

```
procs ------------memory------------ --swap-- ----io---- --system-- -----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa
5 0 0 6037164 71568 5476620 0 0 66 198 1033 823 100 0 0 0
5 0 0 6037164 71572 5476616 0 0 0 158 1021 801 100 0 0 0
5 0 0 6037164 71572 5476616 0 0 0 48 1016 750 100 0 0 0
```

Notice that there is no idle time and that the number of processes waiting for run time is 5.
Now, to show what instance caging can do to limit the CPU utilization, I started the following PL/SQL block:

```
DECLARE
l_sql VARCHAR2(100) := 'ALTER SYSTEM SET cpu_count = ';
BEGIN
EXECUTE IMMEDIATE l_sql || '4';
dbms_lock.sleep(10);
EXECUTE IMMEDIATE l_sql || '3';
dbms_lock.sleep(10);
EXECUTE IMMEDIATE l_sql || '2';
dbms_lock.sleep(10);
EXECUTE IMMEDIATE l_sql || '1';
dbms_lock.sleep(10);
EXECUTE IMMEDIATE l_sql || '0';
END;
```

This time the output of vmstat(8) looks like the following:

```
procs -----------memory---------- --swap-- ----io---- --system-- -----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa
5 0 0 3695176 80856 7780352 0 0 0 96 1019 783 100 0 0 0
5 0 0 3695176 80856 7780352 0 0 0 170 1023 795 100 0 0 0
5 0 0 3695184 80856 7780352 0 0 0 156 1028 795 100 0 0 0
5 0 0 3695184 80860 7780348 0 0 0 120 1021 795 100 0 0 0
5 0 0 3694928 80860 7780348 0 0 2 168 1029 810 100 0 0 0
3 0 0 3694928 80860 7780348 0 0 0 214 1029 939 77 0 23 0
3 0 0 3694928 80864 7780344 0 0 0 118 1020 961 75 0 25 0
3 0 0 3694928 80864 7780344 0 0 2 152 1026 961 75 0 25 0
3 0 0 3694596 80868 7780340 0 0 2 158 1027 975 75 0 25 0
3 0 0 3694612 80868 7780340 0 0 0 142 1031 979 75 0 25 0
3 0 0 3694612 80868 7780340 0 0 2 164 1024 963 75 0 25 0
3 0 0 3694616 80872 7780336 0 0 0 358 1079 961 52 0 49 0
2 0 0 3694616 80872 7780336 0 0 0 120 1021 909 50 0 50 0
2 0 0 3697312 80872 7780336 0 0 0 162 1025 952 50 0 50 0
2 0 0 3694744 80876 7780332 0 0 0 142 1027 948 50 0 50 0
1 0 0 3694744 80876 7780332 0 0 0 120 1021 954 40 0 60 0
1 0 0 3694748 80876 7780332 0 0 0 234 1034 953 26 0 74 0
1 0 0 3694748 80876 7780332 0 0 0 134 1021 921 26 0 74 0
1 0 0 3696484 80876 7780332 0 0 0 120 1020 954 26 0 74 0
1 0 0 3696476 80880 7780328 0 0 2 196 1035 996 26 0 74 0
5 0 0 3696476 80880 7780328 0 0 0 112 1020 643 96 0 4 0
6 0 0 3696484 80880 7780328 0 0 0 216 1040 778 100 0 0 0
5 0 0 3696484 80884 7780324 0 0 0 160 1021 763 100 0 0 0
5 0 0 3696484 80884 7780324 0 0 0 112 1020 775 100 0 0 0
5 0 0 3696468 80888 7780320 0 0 0 156 1026 785 100 0 0 0
```

Notice that not only the idle time but also the number of processes waiting for run time dynamically changes according to the value specified for the CPU_COUNT initialization parameter. Great!

The only problem I faced during my tests is that the management does not work as expected when lot of time is spent running kernel code. On my server I am able to reproduce such a problem by running the following PL/SQL block:

```
BEGIN
WHILE (TRUE)
LOOP
FOR i IN (SELECT * FROM t)
LOOP
NULL;
END LOOP;
END LOOP;
END;
```

When four sessions run that code, the output of vmstat(8) looks like the following:

```
procs -----------memory---------- --swap-- ----io---- --system-- -----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa
4 0 0 3690112 81032 7780652 0 0 0 138 1025 786 65 35 0 0
4 0 0 3690112 81036 7780648 0 0 0 182 1024 775 66 34 0 0
4 0 0 3690112 81036 7780648 0 0 0 84 1018 779 66 34 0 0
4 0 0 3690112 81036 7780648 0 0 0 138 1024 822 66 34 0 0
4 0 0 3690112 81040 7780644 0 0 0 204 1032 809 66 34 0 0
4 0 0 3690112 81044 7780640 0 0 0 120 1027 798 63 36 1 0
4 0 0 3690112 81044 7780640 0 0 0 146 1021 833 61 37 2 0
4 0 0 3690112 81044 7780640 0 0 0 160 1023 785 63 35 1 0
5 0 0 3690112 81044 7780640 0 0 0 98 1021 782 62 36 1 0
4 0 0 3690112 81044 7780640 0 0 0 146 1021 818 64 36 1 0
4 0 0 3690048 81044 7780640 0 0 2 180 1031 803 63 36 1 0
```

```
3 0 0 3690048 81044 7780640 0 0 0 126 1027 921 53 29 18 0
3 0 0 3690048 81044 7780640 0 0 0 146 1022 988 46 25 29 0
3 0 0 3690044 81048 7780636 0 0 0 180 1032 976 48 25 28 0
3 0 0 3690044 81048 7780636 0 0 0 212 1058 974 48 25 27 0
2 0 0 3690052 81048 7780636 0 0 0 144 1028 950 47 23 30 0
2 0 0 3690052 81048 7780636 0 0 0 204 1035 971 34 14 51 0
2 0 0 3691404 81048 7780636 0 0 0 106 1020 939 32 13 55 0
2 0 0 3691404 81048 7780636 0 0 0 124 1021 972 34 14 53 0
2 0 0 3691404 81048 7780636 0 0 0 202 1032 986 33 14 53 0
4 0 0 3691404 81048 7780636 0 0 0 80 1024 850 51 29 21 0
4 0 0 3691404 81048 7780636 0 0 0 172 1028 790 65 35 0 0
4 0 0 3691404 81048 7780636 0 0 0 220 1033 785 63 37 0 0
5 0 0 3691404 81048 7780636 0 0 0 62 1016 771 64 36 0 0
5 0 0 3691404 81048 7780636 0 0 0 170 1022 785 62 38 0 0
5 0 0 3691404 81048 7780636 0 0 0 18 1026 772 64 36 0 0
```

Notice that in this case about one third of the time is spent running kernel code. As a result, the CPU utilization (for the values of 3, 2 and 1) is always higher than expected.

1- Regarding the CPU_COUNT, The Oracle cpu_count is determined when you start Oracle. The cpu_count affects the Oracle cost-based optimizer through many calculated parameters that use cpu_count as their basis and are considered every time that Oracle creates an execution plan for a SQL statement. A doubled setting for cpu_count can result in changes to yhour explain plans when moving from a single to multiple CPUs or when you increase the number of CPUs in your system.

2- Oracle DOES NOT RECOMMEND changing CPU_COUNT. A number of other defaults for various configuration parameters are derived, by the database, based on the cpu_count and hence it is strongly recommended NOT to modify the cpu_count parameter

3- In 11g, you have a new feature "instance caging", as shown below, you can have more than one instance on the same server, and you can share the CPUs reported by the operating system kernell (select cpu_count_current,cpu_core_count_current from v$license;)

> 3.a) Over-provisioning—You would use this approach for non-critical databases such as development and test systems, or low-load non-critical production systems. In this approach, the sum of the CPU limits for each instance exceeds the actual number of CPUs on the system. For example, on a 4-CPU system with four database instances, you might limit each instance to three CPUs. When a server is over-provisioned in this way, the instances can impact each other's performance. However, instance caging limits the impact and helps provide somewhat predictable performance. On the other hand, if one of the instances has a period of high load, the CPUs are available to handle it. This is a reasonable approach for non-critical systems, because one or more of the instances may frequently be idle or at a very low load.
>
> 3.b) Partitioning—This approach is for critical production systems, where you want to prevent instances from interfering with each other. You allocate CPUs such that the sum of all allocations is equal to the number of CPUs on the server. For example, on a 16-server system, you might allocate 8 CPUs to the first instance, 4 CPUs to the second, and 2 each to the remaining two instances. By dedicating CPU resources to each database instance, the load on one instance cannot affect another's, and each instance performs predictably.

4- If you are running Development or Testing system and the OS reports 128 CPU then you can divide these CPUs against the instances and each instance will take the CPU based on the processing needs, the CPU usage can be controlled in each instance by enabling the Resource Manager and assigning a resource plan, and ensure that the resource plan has CPU directives

5- If you are running a production system then you should set the CPU count to the physical sockets you have and Oracle will detect the multi core and will utilize it supposing that the CPU's are running hyperthreaded and T5240 has this feature enabled

REFERENCES

NOTE:1484302.1 - Master Note: Overview of Oracle Resource Manager and DBMS_RESOURCE_MANAGER