An Oracle White Paper
September 2009

Database Instance Caging: A Simple Approach to Server Consolidation

Introduction

The goal of server consolidation is to use servers more efficiently in order to minimize the number of servers, particularly under-utilized servers. Server consolidation is a common theme for today's database administrators. With the steady increase in processing power, dedicating a server to a single database instance often results in inefficient resource usage.

When multiple database instances share a single server, they must share its CPU, memory, and I/O bandwidth. A dearth of any of these resources results in contention so that one resource-intensive database instance can significantly degrade the performance of the other instances. Today's market offers many tools for managing server resources, such as virtualization and O/S workload managers. A DBA may be reluctant to use these options because of licensing costs, administration costs, or performance overhead.

Oracle offers a simple and effective approach to server consolidation called "Instance Caging". Instance Caging focuses on managing CPU by limiting the CPU usage of an Oracle database instance. This white paper describes how to configure and use this database feature.

This white paper is written for Oracle database administrators. It contains references to Oracle terminology and concepts. For a full description of the Oracle terminology and concepts, please refer to the Oracle Database Concepts and Database Administrator's Guide.

Scenarios for Server Consolidation

Instance Caging limits the amount of CPU an Oracle database instance consumes, using the Oracle Database Resource Manager and the cpu_count parameter:

- The Oracle Database Resource Manager limits the amount of CPU that the database instance consumes.
- The cpu_count parameter specifies the limit.

Step-by-step instructions for configuring Instance Caging are given in the subsequent section. In this section, we describe how Instance Caging can be used to address two common scenarios.

Scenario 1: Non-Critical Databases

While performance-critical, production databases dominate much of the database literature, the majority of databases in most companies are actually test, development, or low-load, non-critical production databases. These databases are by far the most common candidates for server consolidation because DBAs are willing to expose them to the issues inherent to a shared, non-isolated environment. Furthermore, the DBA can usually place the database instances so that for the vast majority of the time, they can share a server without appreciably degrading each other's performance. The DBA does not need a CPU management tool that dedicates CPU resources to each instance. Such an approach would result in idle CPUs. Rather, he needs a tool to control the instances when there is a spike in one of their workloads.

One effective solution is to impose a CPU usage limit on each instance. For example, if 4 database instances are sharing a 4 CPU server, each database instance can be limited from using more than 3 CPUs at any time.

If all 4 database instances experience a simultaneous surge in database load, each database will not be able to use its 3 CPU limit since there is a total of only 4 CPUs. That is, the CPU limit specifies the maximum, not actual amount of CPU the database instance can use at any time.

When the sum of the database instances' CPU limits exceeds the total number of CPUs, the server is "over-provisioned", as shown in Figure 1.

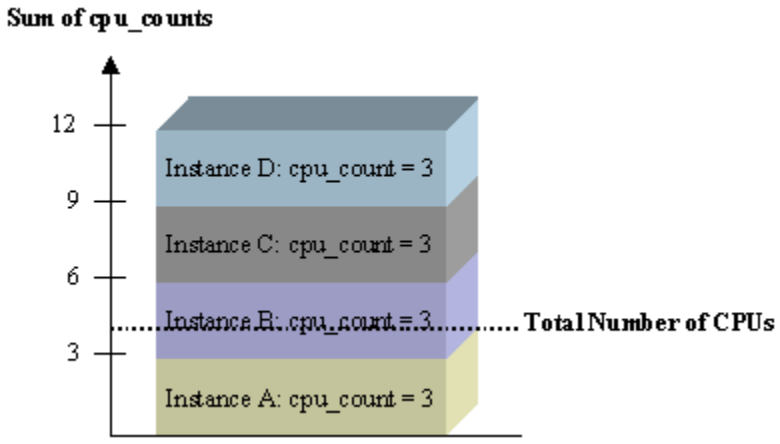

Figure 1: Configuring Instance Caging to Over-Provision the Server

The "Over-Provisioning" Approach

When a server is over-provisioned, the database instances can impact each other's performance. However, Instance Caging limits their impact and helps to provide predictable performance.

In general, if all database instances use Instance Caging, the maximum percentage of CPU that a database instance can consume at any point in time is its own limit divided by the sum of the limits for all active databases. This is because Instance Caging limits the number of runnable processes for each instance and the operating system allocates the CPU in proportion to the runnable processes for that instance.

For the previous scenario, if all 4 database instances are active and CPU-bound, then one instance will be able to consume 3 / (3 + 3 + 3 + 3) = 25% of the CPU. If only two instances are active and CPU-bound, then one instance will be able to consume 3 / (3 + 3) = 50% of the CPU.

Therefore, with over-provisioning, the DBA can predict the amount of CPU an instance will receive in the worst case, when all other database instances are fully loaded. In addition, if all other database instances are idle, the amount of unused CPU is also minimized.

Scenario 2: Performance-Critical Databases

In the second scenario, multiple performance-critical database instances share a server. The DBA would like to partition the CPUs among the database instances, thus shielding them from interfering with each other.

The "Partitioning" Approach

The DBA can use Instance Caging to partition the CPU resources by ensuring that the sum of all CPU limits does not exceed the total number of CPUs. For example, if 4 database instances share a 32 CPU server, then their limits can be set to 16, 8, 4, and 4, as shown in Figure 2.

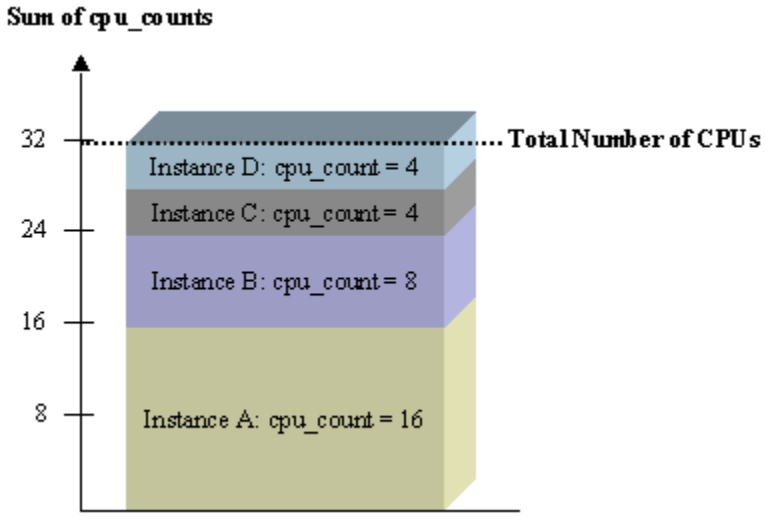

Figure 2: Configuring Instance Caging to Partition the Server

By dedicating CPU resources to a database instance, partitioning provides two advantages: (1) one database instance's CPU load cannot affect another's and (2) each database instance's CPU resources is fixed, leading to more predictable performance.

Alternatives to Instance Caging

In this section, we describe alternatives to Instance Caging. In general, most tools for managing CPU support one or more of the following controls. Instance Caging can be configured to support each of these controls.

1. Minimum CPU allocation. For each database instance, the DBA can configure the minimum amount of CPU that the database instance should be able to consume.
2. Maximum CPU allocation. For each database instance, the DBA can configure the maximum amount of CPU that the database instance can consume.
3. CPU shares. CPU shares specify the relative amount of CPU allocated to a database instance. At any moment in time, the CPU allocation for a database instance can be computed by dividing its share by the total number of shares from database instances that need CPU resources.

Hard Partitions

Many operating systems, such as Solaris, AIX, and HP-UX, allow the hardware to be partitioned. Hard partitions are often called "processor sets". The system administrator creates a processor set by selecting CPUs to be dedicated to the processor set. Only applications associated with this processor set can use its CPUs.

Using hard partitions is similar to configuring Instance Caging in the "partition" mode. However, while hard partitions restrict the database instance to running on the partition's CPUs, Instance Caging allows the database instance to run on any CPU.

Hard partitions are not available on all operating systems.

O/S Workload Managers

An O/S workload manager provides the ability to manage multiple types of resources, such as CPU and memory. These tools typically support all 3 types of CPU resource controls: minimum CPU, maximum CPU, and CPU shares. They may also offer controls to manage CPU, based on service-level objectives specified by the administrator. Examples of O/S workload managers include the AIX Workload Manager, the HP-UX Workload Manager, and the Solaris Resource Manager.

These workload managers offer a rich set of controls. However, they are not available on all platforms and differ widely in terms of feature set and configuration steps.

Virtualization

Virtualization is a prevalent technology used to help multiple database instances share a server. Since each database instance can reside in its own virtual machine, virtualization is a good way to provide isolation. The hypervisor or virtual machine monitor can typically be configured to allocate CPU for each virtual machine using all three types of controls discussed above.

While virtualization offers a rich set of controls, it has several issues:

1. Of all the options described here, it has the most administrative overhead. Not only do the resource controls need to be configured, the virtual machine for each database instance must be separately administered.
2. For some virtualization options, a virtualization license must be purchased from the vendor.
3. A database running on a virtual machine will likely suffer a loss in performance, particularly for I/O. While para-virtualization platforms help alleviate this issue, they too add some overhead to I/O.

Resource Manager and Server Consolidation

If Resource Manager is used to manage the CPU usage by workloads within a database instance, it can only effectively enforce the Resource Plan if it knows how many CPUs the database instance can use at any point in time. By default, Resource Manager assumes that the database instance can use all CPUs on the server.

With Instance Caging, the number of CPUs is specified by cpu_count. Because the number of CPUs is known, Resource Manager works well with Instance Caging.

For all alternative server consolidation tools, Resource Manager can determine the number of CPUs allocated to the database instance only if (a) the CPU allocation to the database instance is static (e.g. hard partitions) and (b) Oracle is provided this number when it checks for the number of CPUs. The number of CPUs that a database instance believes it is running on can be obtained by checking the value of the "cpu_count" parameter.

Enabling Instance Caging

Enabling Instance Caging is a simple, two-step process.

Step 1: Setting "cpu_count"

"cpu_count" is a dynamic parameter that is not set by default. It should be set to the maximum number of CPUs that the database instance should utilize at any time. For example, to limit the database instance to 4 CPUs:

```
alter system set cpu_count = 4;
```

For CMT CPUs like the UltraSPARC T1 and UltraSPARC T2, each thread should be counted as a CPU. For hyper-threaded CPUs like the Intel Xeon and Nehalem, each logical CPU should be counted as a CPU.

Although cpu_count is a dynamic parameter, Oracle does not recommend frequent or large modifications. As a best practice, Oracle recommends using a minimum value of 2.

Step 2: Enabling Resource Manager

The second step for Instance Caging is to enable a resource plan that manages CPU. A resource plan describes how CPU resources should be allocated to processes within the database instance.

Any resource plan with CPU directives can be used. For Oracle Database 11*g*, any of the out-of-the-box plans such as "DEFAULT_PLAN" or "DEFAULT_MAINTENANCE_PLAN" can be used.

Resource Manager can be enabled by setting the resource_manager_plan parameter:

```
alter system set resource_manager_plan = 'default_plan';
```

Enabling a Resource Plan and setting the cpu_count parameter turns on Instance Caging. For a more detailed description of using and configuring Resource Manager, please consult the Oracle Database Administrator's Guide.

Monitoring Instance Caging

Monitoring "cpu_count"

The following query can be issued to obtain the value of the cpu_count parameter value if it has been set. If 'cpu_count' has not been set, then no value will be returned.

```
select value from v$parameter where name = 'cpu_count' and (isdefault =
'FALSE' or ismodified != 'FALSE');
```

Monitoring Resource Manager

The following query can be issued to determine the status of Resource Manager.

```
select name from v$rsrc_plan where is_top_plan = 'TRUE' and cpu_managed =
'ON';
```

If no rows are returned, Resource Manager is off or Resource Manager is on but not managing CPU (i.e. the mgmt_p1-8 directives are not set in the Resource Plan). If a row is returned, it indicates the plan being used.

Monitoring Throttling

Instance Caging limits the CPU consumption of the database instance processes by throttling them. An Oracle process may be throttled if it is a foreground process or a non-critical background process. An Oracle process is being throttled when it is waiting on the "resmgr:cpu quantum" wait event. If the instance is being heavily throttled because of Instance Caging, this wait event will probably show up as one of the top wait events.

The amount of throttling can be monitored in two ways.

The v$rsrcmgrmetric_history view shows the amount of CPU consumption and throttling for each minute in the past hour. For each consumer group, "cpu_consumed_time" specifies the number of milliseconds of CPU consumed and "cpu_wait_time" specifies the number of milliseconds that processes were throttled.

```
select begin_time, consumer_group_name, cpu_consumed_time, cpu_wait_time
from v$rsrcmgrmetric_history order by begin_time;
```

The v$rsrc_consumer_group view shows the amount of CPU consumption and throttling since CPU Resource Management was enabled. 'consumed_cpu_time' and 'cpu_wait_time' measure time in milliseconds.

```
select name, consumed_cpu_time, cpu_wait_time from v$rsrc_consumer_group;
```

Case Studies

Case Study 1: Limiting CPU Utilization Using Instance Caging

The goal of this case study is to show that Instance Caging is regulating CPU usage by monitoring the CPU utilization.

The workload for this case study was an OLTP application called Order Entry, provided via Swingbench. See http://www.dominicgiles.com/swingbench.php for more details. This workload's database was run on a Linux server with 4 CPUs using Oracle Database 11*g* Release 2. The case study ran Swingbench without Instance Caging and then with Instance Caging with cpu_count set to 2. Table 1 below shows the results.

Without Instance Caging, the CPU utilization was 90% in user time and 6% in system time, resulting in busy time of 90% + 6% = 96%. This high busy time value indicates that the Oracle database instance was CPU bound and the server was almost fully utilized.

When Instance Caging was enabled with cpu_count set to 2, we expect the user time to be 50%, since the Oracle database instance was limited to using 2 out of 4 CPUs at any time.

As shown in the CPU Utilization row of the table, Instance Caging decreased the user time from 90% to 51%, almost exactly matching the target of 50%. This successful result demonstrates that Instance Caging is able to control the CPU utilization of an instance, based on the value of cpu_count.

We expect that the drop in CPU utilization would affect the application's performance. As shown in the blue row of the table, without Instance Caging, the application sustained about 32K transactions per minute. Since the drop in user time when Instance Caging was enabled was (90 – 51)/90 = 43%, we expected a similar drop in performance. With Instance Caging, the application sustained about 19K transactions per minute, or (32 – 19)/32 = 41% of the performance. Therefore, the drop in performance mirrored the drop in user time.

Table 1: Results for Case Study 1

	No Instance Caging	Instance Caging with cpu_count = 2 (out of 4 CPUs total)
CPU Utilization	user 90% sys 6% idle 4%	**user 51%** sys 4% idle 45%
Transactions Per Minute	32K	**19K: 41% drop in performance**

Case Study 2: Partitioning a Server Using Instance Caging

The goal of this case study is to see if Instance Caging can be used to partition the server. For a CPU-bound application, we expect to see that the performance of two databases is directly proportional to their cpu_count settings. Furthermore, we expect that Instance Caging does not cause any appreciable overhead; the total rate of performance should remain constant and should be equivalent to a single database running without Instance caging.

The workload for this case study was an OLTP application called Sysbench. See http://sysbench.sourceforge.net for further details. We ran two database instances on the same 6 CPU Linux server using Oracle Database 11*g* Release 2. We ran an instance of the sysbench application against each database instance. For both database instances, Instance Caging was enabled. We partitioned the server by varying the cpu_count for both instances so that they always summed to 6 CPUs.

In Figure 3, the Instance 1 and Instance 2 graphs show that varying the cpu_count for each instance results in a corresponding change to the transaction rate. The yellow line shows the total transaction rate. The constant total transaction rate indicates that Instance Caging is successfully shifting CPU resources from one database instance to another. The turquoise line shows a single instance running by itself without Instance Caging. The data points for <0,6> and <6,0> show that for a single instance, the performance is the same, whether Instance Caging is on or off, indicating that Instance Caging is not introducing any overhead. This successful result indicates that Instance Caging can be used to partition CPU resources between the database instances.

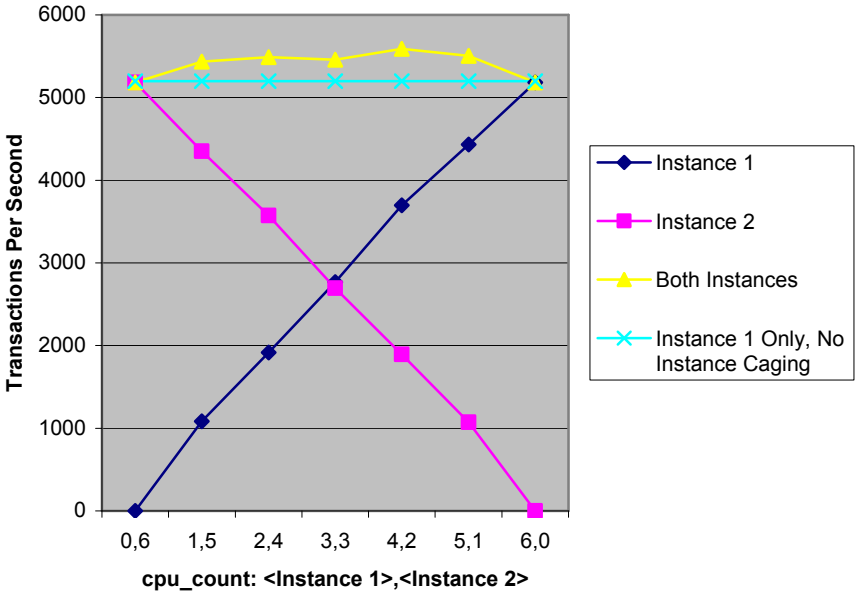

Figure 3: Performance of Partitioned Server

Supported Releases

Oracle Database 11*g* Release 2 is the first release for which Instance Caging is documented and supported. Instance Caging is only available with Oracle Database Enterprise Edition.

Conclusion

Instance Caging is a simple and effective feature for DBAs who want to consolidate the servers in their data center. Instance Caging can be configured in just two steps: setting the cpu_count parameter and enabling Resource Manager.

Many servers are shared by test, development, and non-critical production databases. The DBA can use Instance Caging to prevent one database from using too much CPU, thus limiting the impact of one database on another and preventing runaway activity from wreaking havoc on the entire server. The DBA can also use Instance Caging for performance-sensitive databases to partition the CPU.

Because Instance Caging is simple to configure and does not require any new software to be licensed or installed, it is an excellent alternative to other server consolidation tools, such as virtualization and O/S workload managers.

Database Instance Caging:
A Simple Approach to Server Consolidation
September 2009
Author: Sue K. Lee, Vikram Kumar
Contributing Authors: Dechang Gu

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Oracle is committed to developing practices and products that help protect the environment

0109