The Oracle Database on HP Integrity servers

Executive summary

This white paper describes Oracle® Database deployment on the HP Integrity server platform with a strong focus on HP-UX. Even though the focus of this paper is Oracle 10*g*, much of it applies to Oracle 11*g* as well, even though there are some minor changes, especially in the area of dynamic memory management.

The integration of HP-UX workload management products with the resource management features provided by the database is described. While the ability of HP-UX to dynamically reconfigure containers can be useful, care must be taken when these workload management products are used in an Oracle deployment.

The paper also outlines the relationship of the cache-coherent Non-Uniform Memory Access (ccNUMA) architecture provided by the HP Integrity server platform with database components such as the System Global Area (SGA), private server processes, Oracle shared servers and parallel query slaves. Since tuning opportunities with Oracle on ccNUMA tend to focus on maximizing local memory accesses, a methodology for configuring an Oracle instance to maximize the number of Cell Local Memory (CLM) accesses is provided.

Intended audience: System Architects and Database Administrators responsible for Oracle deployments on HP-UX Integrity servers. Note that the paper describes current Oracle releases 10gR2 and 11gR1. Future releases may change and improve Oracle behavior in many aspects. This paper is not a best practices paper, nor a cookbook for setting up Oracle on Integrity servers. It attempts to provide some background to the Oracle features that are of interest on large Integrity servers.

Resource management with Oracle and HP-UX

This section focuses on how HP Virtual Server Environment (VSE) features work in an Oracle deployment – in particular, on the integration of HP workload management functionality with Oracle's database resource management products.

The following topics are covered

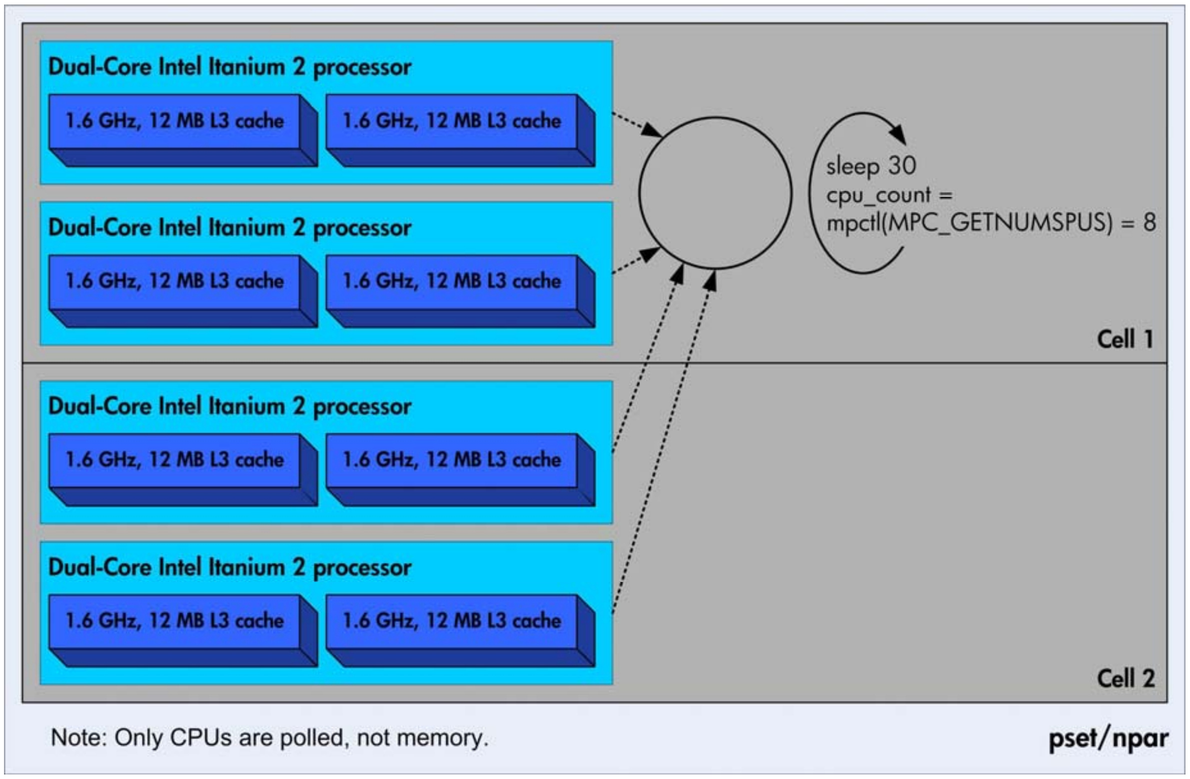
- Oracle's view of CPU resources
- Adding and deleting CPUs with HP-UX workload management products
- Oracle's memory management features
- Two different views of resource management – at the database and OS levels
- An overview of Oracle and HP-UX schedulers
- A sample integration of Oracle Database Resource Manager (DBRM) and HP-UX Workload Manager (WLM)

VSE integrates the HP virtualization products on Integrity servers. This paper looks at VSE from the Oracle point of view: resources (CPU, memory) being added to the container running the Oracle database and resources being deleted from the container. For more information about VSE, go to http://www.hp.com/go/vse. The container (or compartment) could be any of the supported compartments such as a Processor set (pset), a virtual partition (vPar), a hard partition (nPar) or a virtual machine (VM). Most of these containers types support dynamic adding and removing of resources.

Oracle's view of CPU resources

Figure 1 outlines an Oracle database instance's view of the available computing resources within a system. In this context, we assume that the database instance runs within a dynamically reconfigurable container, such as a pset or a vPar.

Figure 1. Sample container



One thread constantly monitors the number of CPUs in the container in which Oracle is running. Internally, Oracle uses the HP-UX 'mpctl' call to determine the number of CPUs, which works well for psets, vPars and nPars, since Oracle internal algorithms work with whole CPUs. However, this does not preclude the use of FSS (Fair Share Scheduler) with PRM (Process Resource Manager), as CPU_count is used to set up data structures for that number of threads executing in parallel at any point in time.

Adding and deleting CPUs

As indicated above, Oracle is prepared for the number of CPUs to change while the instance is running. On HP-UX, these changes would typically happen under the control of workload management products, Workload Manager (WLM) or Global Workload Manager (gWLM). Consult the web pages http://h71028.www7.hp.com/enterprise/cache/257279-0-0-225-121.html?jumpid=reg_R1002_USEN and http://www.hp.com/go/wlm for more information on these workload management products. These products allow you to specify limits that can trigger the addition or deletion of CPU resources. For example, resources can be added if CPU consumption increases above the limit for the container in which Oracle is running; alternatively, if consumption falls below the specified limit, CPU resources may be removed and, if needed, given to other applications.

Note that CPU consumption is not the only metric that can be used to control the movement of CPU resources between containers; for example, the number of active Oracle users can be used to trigger the re-allocation of CPU resources.

Oracle's view of available CPU resources is published through the Oracle parameter 'CPU_count'. This parameter can also be set by the DBA, setting this parameter in the Oracle init.ora or spfile will override any dynamic changes in the container that Oracle runs in, and will make those changes invisible to the Oracle database. Oracle is dependent on the view of CPUs provided by the operating system.

The CPU_count parameter could be used to isolate Oracle instances from each other when the Oracle database resource manager is in use; on a 8 processor system, one Oracle instance could be given 6 processors by setting CPU_count = 6, and an other instance could be given the remaining 2 processors by setting CPU_count = 2. HP-UX provides its own methods of isolating Oracle instances running on the same server, for example, through the use of psets or vPars.

It is important, that Oracle has the correct view of available processor resources, as Oracle adjusts quite a few internal parameters based on CPU_count:

- parallel_max_servers: The degree of parallelism for parallel queries.
- Fast_start_parallel_rollback: This parameter controls the degree of parallelism for recovery of DML or DDL when you have a system crash.
- db_block_lru_latches: A too small value may result in contention on LRU latches.
- log_buffer: The size of the log buffer depends on the number of processors.

How many CPUs can be added to or deleted from an instance? Note that some of Oracle's internal data structures have dependencies based on the number of CPUs. Since these structures get allocated at database startup, adding too many CPUs to a running instance may provide enough resources for an excessive number of threads to run simultaneously, causing contention in the database. While there is no absolute rule, the following rule-of-thumb is offered:

> **The number of CPUs should increase by no more than three times the number of CPUs at database startup.**

It is still possible to add more processors, but the Oracle parameter CPU_count will not increase beyond this limit.

It is important to make sure, that an Oracle container does not loan out all but one processor (when under control by gWLM, for instance), when Oracle is not running in that container. If this is the case, Oracle will only have one processor at startup, and this will put severe limits on how many processors that container can grow to.

In a ccNUMA aware deployment adding and deleting processors is a more complex task. Processors supporting a ccNUMA aware Oracle instance should run in a locality domain belonging to the Oracle instance. Processors from remote cells will not see the performance benefits of being close to a NUMA pool, which further restricts the choice of processors to add to a running Oracle instance.

Hyper-threading and Oracle

Intel® Itanium® processors support hyper-threading at the hardware level, hyper-threading can be enabled in the container running Oracle (pset, vPar or nPar). Oracle will make use of hyper-threads; they are visible to Oracle as normal processors. Enabling hyper-threads in a container running Oracle will double the number of processors in that container; this is still below the 3x upper limit of processors to add to a live Oracle instance. Thus hyper-threads can be enabled and disabled on-line, while Oracle is running. The performance impact of using hyper-threads is application dependent; some applications may see a substantial performance gain some see less or no gain. An OLTP application with a large number of active running threads may see a benefit of using hyper-threads.

Parallel query slaves and Dynamic processor resources

Above, we described the fact that deleting CPUs from an Oracle instance and adding CPUs to an instance will be observed by Oracle. If CPUs are added, Oracle processes will start running in the new CPUs. However, if CPUs are added while a parallel query is running, the number of query slaves will not be increased for the running query. Only when the query has finished, and a new parallel query starts, will the new CPUs have effect on the degree of parallelism (DOP). The default DOP is 2 times the number of CPUs. This should be taken into consideration, especially when HP-UX workload management products (gWLM) are used to control the size of the container Oracle runs in. Typically, the use of these tools are reactive in nature, CPUs will be added to an instance only after the increased load is observed as increased CPU utilization.

Oracle memory management and dynamic memory movement

Oracle dynamic memory management.

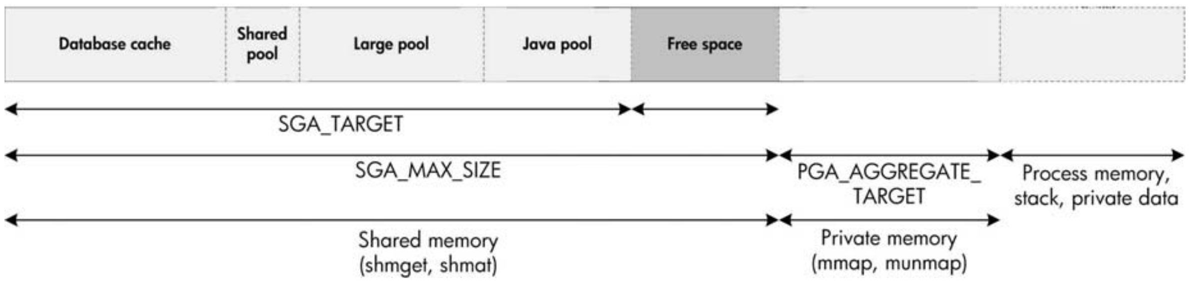
This section gives an overview of Oracle's dynamic memory management features and their relation to the resource management features of HP-UX.

As of HP-UX 11i v1, the kernel implements Process Resource Manager (PRM) Memory Resource Groups (MRG) that allows you to partition memory into logical units. Each group is allocated a fixed amount of memory and has the option to borrow or lend memory, as needed. The latest version of PRM allows you to group shared memory as well, which makes it more suitable for Oracle deployments. The following web page provides more information about Oracle and MRGs: http://h20338.www2.hp.com/hpux11i/downloads/PRM.Oracle.SharedMemory.pdf.

On HP-UX 11i v3, physical memory can be migrated between vPars as well, and Oracle can potentially take advantage of added memory in a vPar. For more about this feature, please consult the following website: http://docs.hp.com/en/9832/vParsMemMigration.pdf.

Oracle's memory management is outlined in Figure 2.

Figure 2. Oracle memory management



The following discussion applies to Oracle running in a non-ccNUMA system, or Oracle running with ccNUMA features disabled. A later section in this paper discusses the memory layout in a ccNUMA environment.

Oracle memory usage is split between the following:

- **Shared memory**
  The size of the shared memory – System Global Area (SGA) – segment used by Oracle is defined by SGA_MAX_SIZE. This parameter defines the total size of the shared memory segment that Oracle will allocate and to which it will attach at instance startup. On HP-UX, Oracle uses the 'shmget(), shmat()' application processing interface (API) to manage this memory. On HP-UX, Oracle will allocate a SGA with the size of SGA_MAX_SIZE at database startup.

  SGA_TARGET controls how much memory is actually being used by Oracle and can be dynamically changed up to SGA_MAX_SIZE. Oracle automatically resizes the different pools within the SGA based on load and Automatic Workload Repository (AWR) advisories.

  On HP-UX, the SGA will often be pinned in memory in many cases to improve performance: The administrator might have set the Oracle parameter 'LOCK_SGA' to true, and/or asynchronous I/O on raw devices or volumes may be in use. The asynchronous driver /dev/async may also pin the SGA in memory.

  If the SGA is not pinned, the pages allocated for "Free space" in Figure 2 is available for other use until SGA_TARGET is increased giving Oracle access to more shared memory; a shared memory segment sized at SGA_MAX_SIZE bytes is still allocated at startup, but the area in the virtual space above SGA_TARGET will not be used. Swap space is reserved for the whole segment, but since the area above SGA_TARGET is not used, thus it is "swapped" out, and does not occupy physical memory. The madvise(2) call is used by Oracle to tell HP-UX to free physical memory pages above SGA_TARGET

- **Private memory**
  PGA_AGGREGATE_TARGET defines how much private memory – Private Global Area (PGA) – Oracle is allowed to use for all its active server processes and/or threads. On HP-UX, this memory is managed with 'mmap/munmap' calls, which means that memory actually gets released back to the OS when not needed.

  PGA_AGGREGATE_TARGET is defined at database startup. It can by changed with 'ALTER DATABASE set PGA_AGGREGATE_TARGET' while the database is running. This could be done for instance, if memory has been dynamically added to the partition Oracle runs in.

- **Other**
  Remaining memory usage consists of the normal overhead associated with running many processes (data segment, process stacks, and more).

Oracle 11*g* uses an enhanced memory scheme, where the user only has to specify an upper bound for all Oracle memory including private memory and shared memory. This upper bound is specified with a dynamic parameter, 'MEMORY_TARGET', which is allowed to grow up to 'MEMORY_MAX_TARGET'. The parameters SGA_TARGET and PGA_AGGREGATE_TARGET are automatically tuned to fit the requirements of the application, when MEMORY_TARGET is set.

Adding memory to a running Oracle instance would have the following impact:

- It would automatically contribute to the area marked "process memory" in Figure 2, there would be space for more processes to run without paging.
- The DBA could increase SGA_TARGET allowing for more shared memory to be used by Oracle without paging out the SGA. Note that this will work only in the cases, where the SGA not pinned. The increased availability of memory is anticipated at startup by setting SGA_MAX_SIZE high (> SGA_TARGET).
- The DBA could increase PGA_AGGREGATE_TARGET allowing for more PGA use without increased paging

- In Oracle 11*g*, the DBA can increase MEMORY_TARGET to give Oracle more memory, and let Oracle size SGA_TARGET and PGA_AGGREGATE_TARGET automatically. The increased availability of memory is anticipated at database startup time by setting MEMORY_MAX_TARGET high (> MEMORY_TARGET).

If there is a need to decrease memory, the SGA_TARGET and/or PGA_AGGREGATE_TARGET parameter may need to be adjusted to meet the decrease in available memory as well (or MEMORY_TARGET, if in use on Oracle 11*g*).

It is important to understand that Oracle allocates all its shared memory (SGA) at startup; it will not allocate more shared memory segments, if more memory becomes available. But, it may be able to start using more of the memory allocated (but not reserved) at startup without introducing excessive paging on the system. The parameters SGA_TARGET, PGA_AGGREGATE_TARGET and MEMORY_TARGET can be changed for other reasons as well. For instance, an application no longer running, could have freed up memory for Oracle to use.

Asynchronous I/O and memory management

On HP-UX, asynchronous database I/O is done through an asynchronous I/O driver, /dev/async. This driver can be used under the following conditions:

- All or some of the database files are set up on raw devices and/or raw logical volumes.
- The Oracle database files are set up on ASM (Automatic Storage Management).
- A third party libODM (Oracle Device Management) library is used to access Oracle database files.

The default behavior of Oracle with /dev/async is to pin Oracle's SGA into memory permanently. With current Oracle releases (10.2.0.4, 11.1.07) this will happen when /dev/async is present in the system, the oracle/dba user has read/write access to it and with the MLOCK privilege. Even if Oracle is never going to use /dev/async to perform I/O, this pinning will happen. This is optimal for performance, but it makes any kind of Oracle automated memory management impossible. When memory is pinned, the upper bound of the Oracle SGA size is defined by the amount of lockable memory available at the time the database starts.

The user can also force the memory to be pinned by setting the Oracle parameter LOCK_SGA='TRUE'.

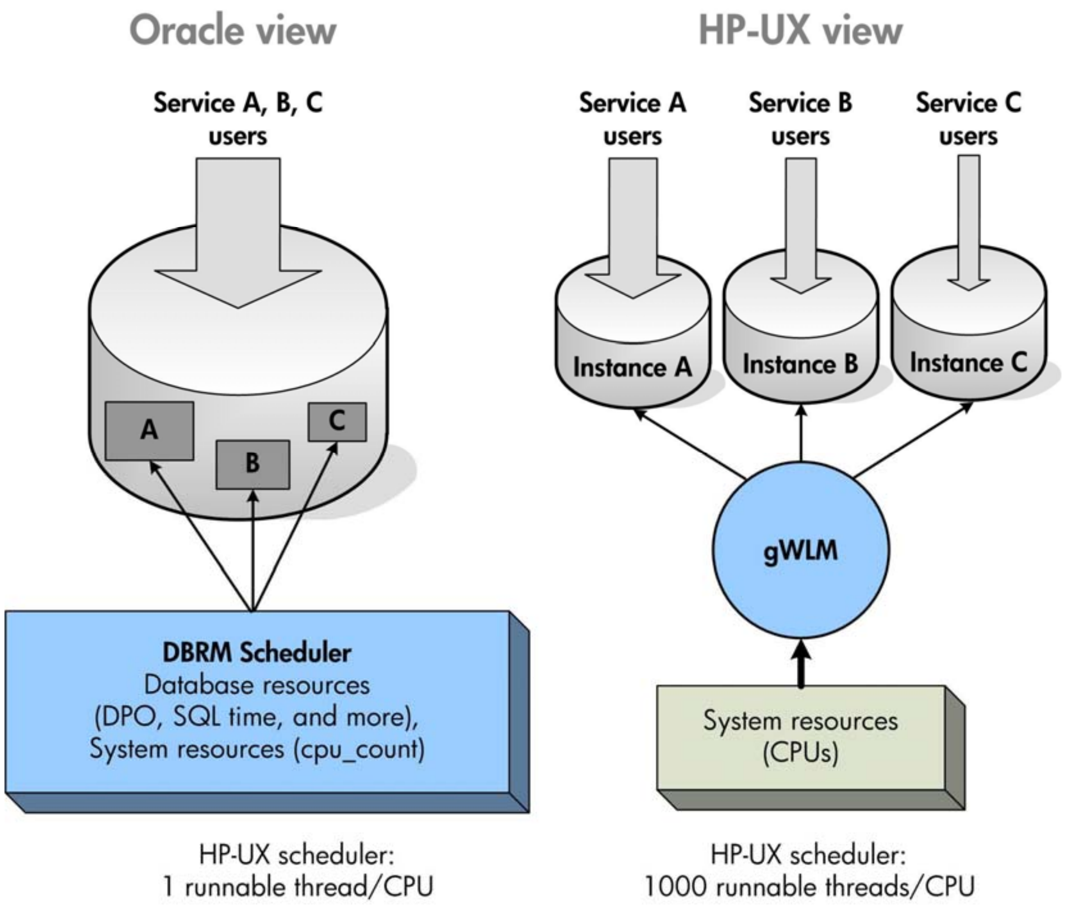
By setting the 'deferred locking' bit (0x100) in the minor number of /dev/async, only memory that is the target of an I/O operation will be pinned. The pin is released upon completion of the I/O operations. Thus, there will be extra overhead to the I/O operations, when memory is pinned and released. Oracle's automated memory management features will work with the 0x100 setting in the /dev/async minor number, as it allows Oracle to free pages in the SGA. The startup of the database will be quicker with this setting as well, since there is no pinning of memory at startup time.

For maximum performance /dev/async should not have the 0x100 bit set, and all SGA should be pinned into memory. However, this will disable Oracle's automatic memory management features (MEMORY_TARGET). This is a tradeoff between high performance and ease of use. Many databases are static in nature, and do not need memory configuration at runtime, in these cases pinning the SGA works well and provides optimal performance.

Two different views of resource management

Resource management in an Oracle database environment can be implemented at the database level, the OS level or both. Figure 3 provides two different views of resource management as provided by Oracle's Database Resource Manager (DBRM) and the HP-UX approach.

Figure 3. Different views of resource management



Oracle approach

Oracle's resource management approach is based on a single database instance running many services. DBRM is aware of database resources and system resources (such as CPU_count) and is used to allocate resources to the different consumer groups.

A consumer group may have a one-to-one relationship with a service; in Figure 3, for example, Consumer Group A may consist entirely of users running Service A. However, consumer groups are purely DBRM entities that are not necessarily tied to specific services.

Benefits of the Oracle approach include:

- DBRM controls the allocation of database resources and can, for example, be used to restrict the degree of parallelism for table scan for lower-priority users.
- The time a query may consume can be restricted.
- Available CPU resources can be re-allocated between the different consumer groups, by giving 75% of CPU to Consumer Group A, for example.
- There is only a single database to administer.

HP-UX approach

HP-UX resource management tools can only allocate resources effectively if the services (A, B and C in Figure 3) are deployed on separate database instances. In this case, WLM/gWLM can move CPUs between the instances based on CPU utilization.

Oracle scheduler

When activated, DBRM takes on the role of scheduling Oracle threads/processes. This is entirely user-space scheduling, with threads regularly asking DBRM for permission to run.

Figure 4 outlines this architecture.

Figure 4. The Oracle DBRM scheduler

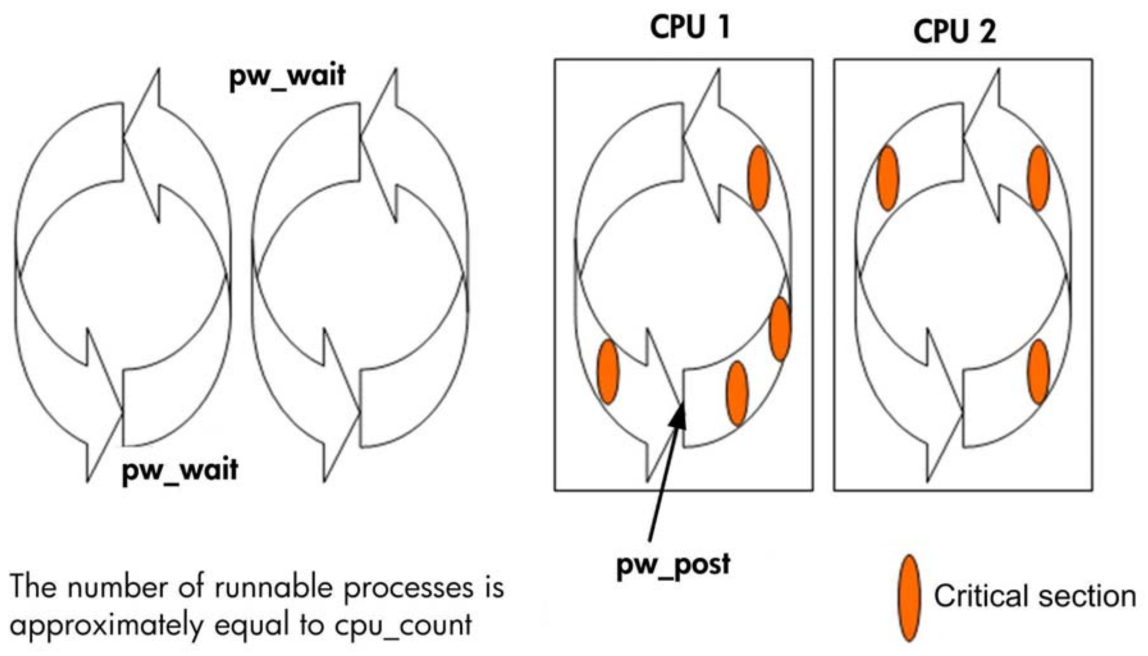

pw_wait
pw_wait
CPU 1
CPU 2
pw_post
The number of runnable processes is approximately equal to cpu_count
Critical section

Figure 4 shows four Oracle processes running in a two-CPU container. Since DBRM tries to keep as many processes runnable as there are processors in the container, two processes are runnable; two are waiting for permission to run.

At regular intervals the running processes ask for permission to continue; DBRM may choose to let the process run or place it in a wait state depending on CPU usage and the resource plans in effect.

The benefits of this scheduling approach include:

- An Oracle process/thread never gets interrupted by another process/thread while executing in a critical section holding a spinlock or latch, for example (marked in orange in Figure 4).
- Oracle makes scheduling decisions based on events that are only known to the database and not visible to any OS scheduler.
- Even on a very large system, the number of runnable processes is kept small, making the job of the OS scheduler easier.

In a production environment, the Oracle scheduler has to adapt to its processes yielding for reasons other than giving up time to other Oracle processes. Oracle server processes rarely keep the CPU 100% busy and may spend time blocked on I/O and/or inter-process communication (IPC); however, Oracle tries to keep the CPU as busy as possible busy under all conditions.

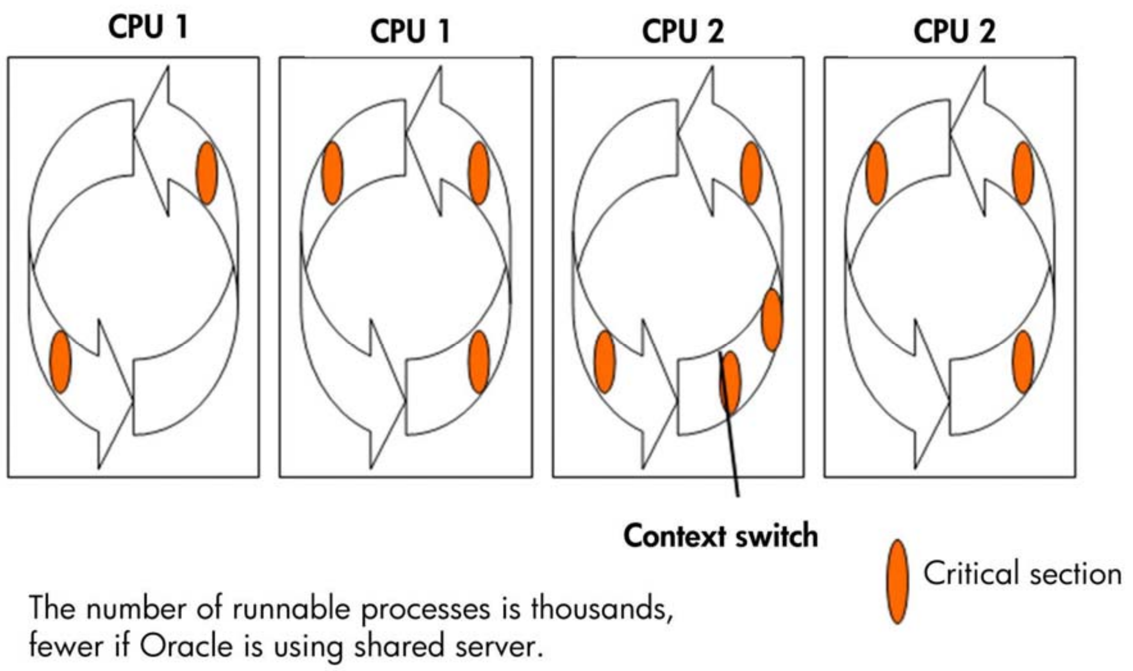
The DBRM uses the cpu_count parameter to decide how many threads to run in parallel. In fact, CPU_count is used to calculate lower and upper thresholds for the number of runnable processes.

The Oracle scheduler probably works best in a container where Oracle has ownership of all CPUs. A pset is a very good match for the DBRM scheduler. In a FSS group, Oracle thinks it has access to all visible CPUs in the system and makes its scheduling decisions accordingly, However, Oracle threads will be under control of the FSS scheduler, potentially interrupting the execution in critical sections. It is also a good practice to not let any other application occupy the same container as Oracle, to have the least possible impact on Oracle's scheduler. If processor sets are used, it is important to contain Oracle in one single processor set only, and not split up the database in several processor sets. If more than one processor sets are used, Oracle will have the incorrect view of available processors, and this may cause unnecessary contention in the database.

HP-UX scheduler

In practice, many HP-UX deployments still rely on the HP-UX scheduler, allowing HP-UX to schedule all running Oracle threads with good success. The HP-UX scheduler controls the running of Oracle threads with normal scheduling policies.

Figure 5. The HP-UX scheduler


CPU 1
CPU 1
CPU 2
CPU 2
Context switch
Critical section
The number of runnable processes is thousands,
fewer if Oracle is using shared server.

With the HP-UX scheduler, as shown in Figure 5, a large number of Oracle threads may be runnable at any point time. These processes keep running without needing to ask for permission. As a result, any process may be context-switched-out at any time, even when executing in a critical section. If the process were holding a latch when context switched-out, the release would take much longer, since that process would have to be re-scheduled before the latch could be released. This could potentially have a serious impact on overall system performance.

Different OSs deal with problems like context switches in different ways. On HP-UX, you should select a scheduler that can minimize risk by making sure that the priorities of processes do not degrade. Such degradation would extend the delay before the process could resume after being context switched out. Set the **hpux_sched_noage** parameter in Oracle's 'init.ora' file or 'spfile' as follows:

```
HPUX_SCHED_NOAGE = 178
```

178 is the highest **sched_noage** priority. This setting works fine, if there is only one Oracle instance running in the OS instance. If the same OS instance is running more than one Oracle instance or other applications in addition to the Oracle instance, setting hp_ux_sched_noage may cause scheduling problems. A CPU intensive database instance may starve an I/O intensive instance. Note, that the hp_ux_sched_noage setting is the default on Oracle 11gR1. If there are multiple Oracle instances and/or applications running in the same container, some kind of partitioning may be required to take advantage of hp_ux_sched_noage, for example the instances could be separated by running them in different psets.

Note that the 'dba' group needs to have the RTPRIO privilege, to be able to set HPUX_SCHED_NOAGE.

The HP-UX scheduler is well integrated with the underlying hardware, and has all the knowledge about cores and hyper-threads to make intelligent scheduling decisions. This information is not visible to the Oracle user space scheduler; it only has a view of "logical" processors.

Integration between HP-UX workload management and DBRM

Since HP-UX workload management products are unaware of Oracle's internal behavior, the integration of WLM/(g)WLM and DBRM to guarantee the best possible intelligent distribution of resources may present a challenge. For instance, if an Oracle instance is CPU-saturated by low-priority users, you may not want to activate an instant capacity (iCAP) CPU to that instance and pay the extra price just to support low-priority users. On the other hand, if CPU is being used by high-priority users, you may indeed want to allocate iCAP resources to that instance so as to temporarily provide enough resources to support the users.

Oracle does impose some restrictions on the use of both the database resource manager and an OS resource manager at the same time. These restrictions are documented in http://download.oracle.com/docs/cd/B19306_01/server.102/b14231/dbrm.htm#i1010600

However, the same document indicates that dynamic reconfiguration of CPUs is supportable.

Currently, there is no simple integrated method for exposing Oracle database resource management information to the HP workload management products; however, it can be achieved through mechanisms such as the Oracle toolkit for WLM. You can execute a SQL script to find out the numbers of high- and low-priority users logged on, then set up a (g)WLM policy to react to the number of high-priority users and add CPU resources as needed. For example, you can set up a policy so that, if there are more than two high-priority users per CPU in the Oracle instance, add another CPU.

Note:
It is beyond the scope of this paper to provide detailed instructions for setting up this policy.

You can set up the following simple Oracle resource plan to give 75% of CPU resources to high-priority users ('orabm_priv') and the remaining 25% of users to low-priority users ('orabm_low'):

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(SIMPLE_PLAN => 'use_case_plan',
CONSUMER_GROUP1 => 'orabm_priv', GROUP1_CPU => 75,
CONSUMER_GROUP2 => 'orabm_low', GROUP2_CPU => 25);
END;
/
```

Now create a script to regularly survey and return the number of high-priority users and, through the 'wlmsend' utility, pass this metric to (g)WLM:

```
SELECT ACTIVE_SESSIONS
FROM V$RSRC_CONSUMER_GROUP
WHERE NAME = 'orabm_priv'
```

If (g)WLM policies have been set up to react appropriately, a CPU is added if there are more than or equal to two high-priority users per CPU in the Oracle instance, as shown in Figure 6.

Figure 6. DBRM/(g)WLM sample integration

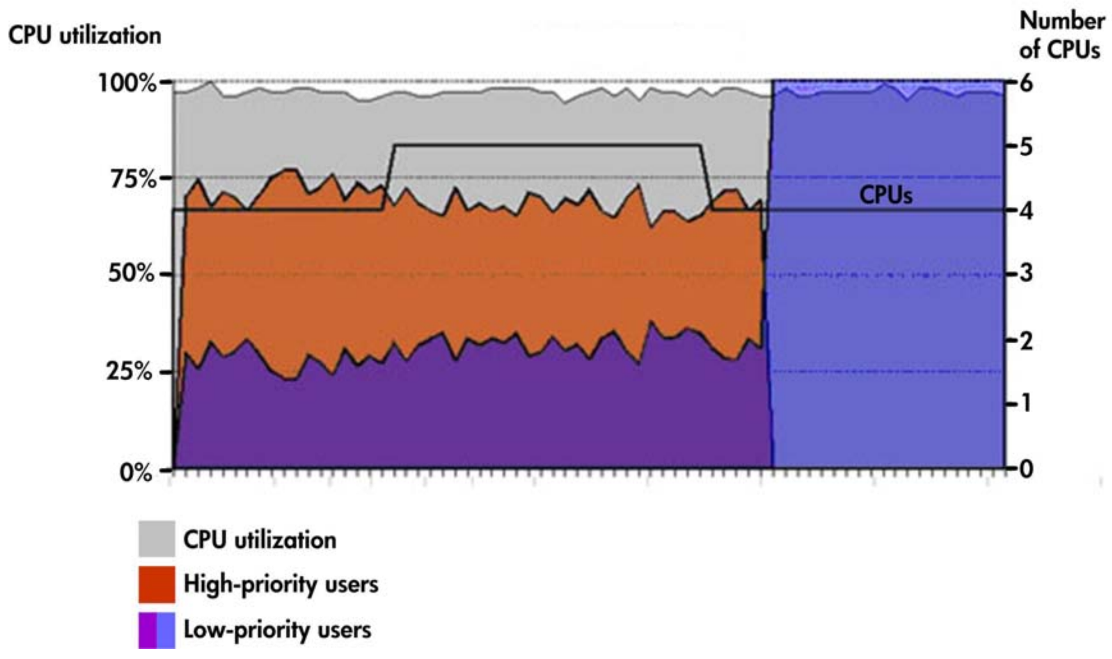


Figure 6 shows that, in this example, an Oracle instance (grey) is running at close to 100% CPU utilization with high-priority users (brown) consuming 75% and low-priority users (purple) consuming 25%.

With 7 high priority users, the number of high-priority users per CPU does not exceed two; four CPUs can accommodate the workload for these users. When an eighth high-priority user becomes active, the number of high-priority users per CPU now equals two; gWLM adds a CPU. With five CPUs, high-priority users continue to consume 75% of CPU resources per the resource plan.

When the eighth high-priority user again becomes inactive, the number of CPUs returns to four.

After the remaining high-priority users have completed their work, all available CPU resources are given to the low-priority users (blue). This is an important feature of the Oracle DBRM; low priority users are allowed to consume all resources not being used by high priority users.

A more elaborate DBRM resource plan, which is outside the scope of this white paper, could also restrict the number of low-priority users to a certain maximum number.

Oracle and ccNUMA

Starting with the 10*g*R2 release, Oracle has begun building some ccNUMA-awareness into the database engine. This section outlines the impact of ccNUMA on database components such as the System Global Area (SGA), private server processes, Oracle shared servers and parallel query slaves. The ccNUMA optimizations are work in progress, and very little documentation exists. The goal of NUMA optimizations is to localize memory accesses as much as possible, making sure that process private data and shared data is allocated out of the same locality domain as the one the process executes in. Some applications see a significant performance boost running with ccNUMA optimizations enabled, some see no performance improvements.

Current releases of Oracle will handle placement of shared memory (SGA) and Oracle background processes. For foreground processes, Oracle relies on the operating system for optimal placement. On HP-UX, the default process placement on ccNUMA systems allows processes to migrate between cells. The following sections describe various methods to give processes launch policies that keep them locked in the locality domain they are started in. This will optimize memory accesses and improve performance. Note that the applicability of these methods are highly application dependent, and may not work in all cases. In most cases running Oracle in its default mode will give more than enough performance, and provide for a more seamless integration with HP-UX workload management tools.

The following document describes recommendations for ccNUMA optimization on HP-UX in more detail: http://docs.hp.com/en/14655/ENW-LORA-TW.pdf.

SGA and background processes

The Oracle ccNUMA architecture is outlined in Figure 7.

Figure 7. Oracle ccNUMA architecture

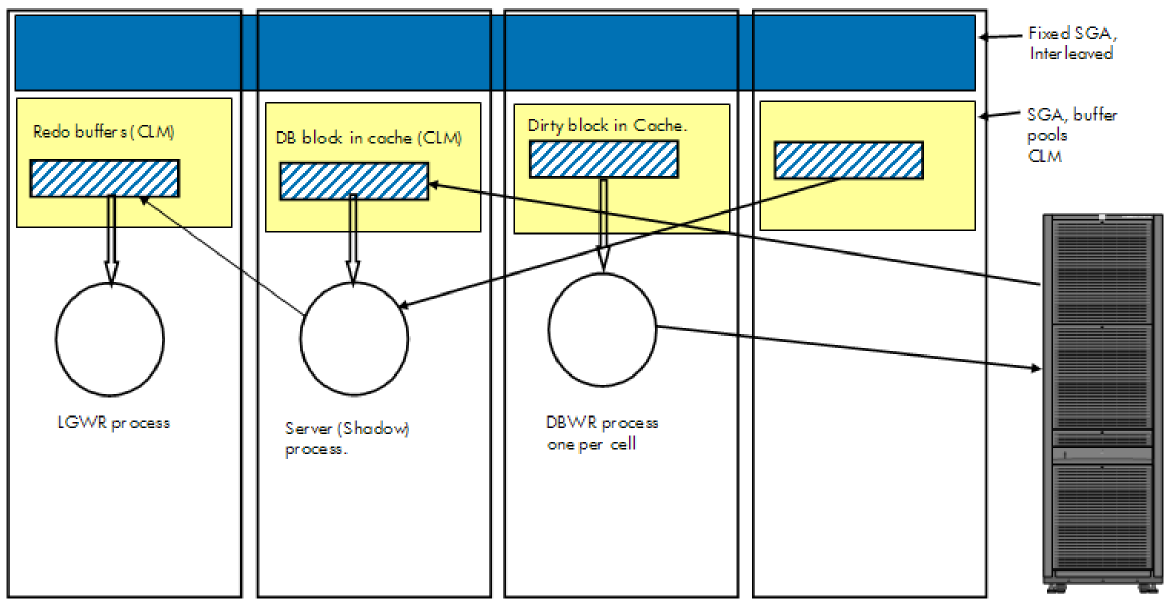


Figure 7 shows examples of areas that get allocated out of ILM (Interleaved Memory) and CLM (Cell Local Memory). Oracle requests placement of memory (CLM) by supplying hints in memory allocations calls to the OS. There is no guarantee, that the OS is able to provide memory in the desired locations, a misconfigured system could seriously skew Oracle's memory allocations.

The architecture includes the following components:

- **SGA**
  The fixed System Global Area (SGA) is laid out in interleaved memory across all locality domains running the Oracle instance. Since this area contains data structures shared by all Oracle processes, it makes sense to store them in interleaved memory that does not favor any particular process. Current Oracle releases do not explicitly place the fixed SGA in ILM, but depends on the default policies of HP-UX for placement. This default may depend on the OS release. The latest release of HP-UX 11i v3 uses a kernel parameter, numa_policy to control the default placement. This parameter should be set to '0' for current Oracle releases to allocate the fixed SGA out of interleaved memory.

  The remainder of the SGA (the most important being the DB block caches) is evenly split up between locality domains in Cell Local Memory (CLM).
- **DBWRs**
  There is one database writer (DBWR) process per locality domain, each being responsible for flushing and coalescing blocks within its own locality domain.
- **LGWR**
  The log writer (LGWR) process runs in a single locality domain and is never allowed to migrate outside that domain. The log buffers are contained in the same locality domain for fast access by the lgwr process.

It is up to the administrator to make sure that the system is configured with at least enough CLM to hold the SGA:

CLM = SGA size / number of cells

This formula is a raw estimate only; enough CLM should be configured to hold the private memory of Oracle processes as well. This is especially important, when process placement is controlled with launch policies locking processes to locality domains, not allowing process migration between locality domains. Use of Oracle shared servers and parallel query slaves also benefit from CLM for process private data, they are always locked into the locality domain they are started in automatically by Oracle. The amount of CLM is always application dependent, but a reasonable guideline is to configure 7/8 of all memory as CLM.

Oracle's ccNUMA features are enabled by default in a cell-based system regardless of the memory configuration; this means that NUMA optimizations are enabled, even when no cell local memory is configured. ccNUMA can be disabled by setting the following static parameters:

```
_enable_NUMA_optimization = FALSE
_db_block_numa = 1
```

Normally, setting "underscore" parameters like these is not supported without permission from Oracle support, and should be avoided on production systems. However, Oracle has published metalink 761065.1 to support the setting of these parameters. To take advantage of the ccNUMA optimizations, cell local memory should be configured on the database server, as servers ship with memory 100% interleaved by default. Oracle will work with ccNUMA optimizations on a system with no Cell Local Memory, but the Oracle optimizations are wasted, since all memory is interleaved. It is good practice to disable Oracle's NUMA optimizations on a system with only interleaved memory.

Background processes

The LGWR and DBWR processes shown in Figure 7 are examples of Oracle background processes, not actual server processes. Oracle controls placement of its background processes by locking them into the locality domain they are started in. This will guarantee that the process private memory gets allocated out of the same locality domain for best possible performance. Oracle also spawns one database writer process per locality domain, each DBWR process being responsible for flushing out dirty blocks from its local NUMA pool only.

Shadow processes

Foreground processes (shadow processes) are created and terminated as users connect to and disconnect from the database. Shadow processes are launched with the default launch policy; Oracle does not control placement for these processes.

Oracle localizes memory accesses to shared data as much as possible. This applies both to shared data used by Oracle to manage the processes as well as database data blocks accessed by the shadow processes. Under the default launch policy, the shadow process may migrate to another domain, especially under high loads. This has the advantage of keeping the load evenly distributed across the domains, but the disadvantage is that memory accesses become remote for processes migrating out of their home locality.

It is possible to evenly distribute shadow processes across locality domains, and locking them into domains by giving them the appropriate launch policies, as shown in an example in the Tuning a ccNUMA architecture by localizing memory accesses section. However, preventing shadow processes from migrating to another locality domain may not always be a good choice in that some cells may become underutilized if some users exit quickly while others are running longer jobs. If a process does migrate to a remote locality domain, it will loose its close proximity to its memory, both the SGA data in the local NUMA pool, and its private memory if it is allocated in CLM. As the process accesses database blocks not yet in memory, those shared memory accesses will become local, but the private data will not become local, until the process migrates back to its original domain. Note that the private memory of a process is allocated in CLM, unless no CLM is available in the locality domain, in which case the memory is allocated in ILM. If no ILM is available, HP-UX is going to look in remote domains for cell local memory.

Oracle shared servers

Oracle allows database users to share a single server through a mechanism known as a shared server (or multi-threaded server), which can be very useful in an environment where users log on, execute a short query, and log off. Since they are pre-spawned, using shared servers can eliminate the overhead needed to create new processes to serve connections.

Oracle shared servers are started by the database at database startup, not by the listener process, when connections to the database are created.

Since the number of shared servers is fixed and all are started at database startup, they are good candidates for a ccNUMA environment. Oracle does spread the shared servers evenly across all locality domains and makes sure they do not migrate between these domains by locking them into the locality domain they are started in. It is also good practice to configure enough CLM to hold the private memory of the shared servers.

Parallel query slaves

Parallel query slaves are processes spawned to parallelize full table scans. The scan is split into pieces and each piece given to a query slave; these slaves work in parallel, resulting in much shorter query times.

Parallel Query Slaves communicate through shared memory, and in the case of a complex query there may be a significant amount of messages passed between query slaves. Depending on the degree of parallelism, it could make sense to have a query run all its slaves in the same locality domain to make this shared memory access local. For queries with a very large degree of parallelism and a huge amount of query slaves running, it makes sense to spread out the slaves across all locality domains, and this is what Oracle does. Oracle will use the same placement strategy for query slaves as for other background processes: The query slaves will be evenly distributed across the available locality domains. This placement strategy will be used for slaves started at database startup and for slaves started by a query coordinator; the slaves are locked in the locality domains they are started in.

Oracle ccNUMA and dynamic partitions

Oracle NUMA optimizations are based on localizing memory accesses to locality domains (cells in an Integrity server) as much as possible. Shared memory (SGA) will be split up in equally sized NUMA pools, one for each locality domain. These pools will be allocated out of CLM. Background processes will also be placed and locked into locality domains. Current versions (10gR2, 11gR1) of Oracle queries the complex configuration (number of cells, and cell numbers) at startup time, and does not react to changes in this configuration. The following requirement must be met:

The localities (cells) present at database startup have to be always available to Oracle, and therefore cannot be deactivated during the lifetime of the Oracle instance. A "CPU_less" locality is not supported, there must be at least one CPU visible to Oracle in all localities at all times.

This is a reasonable requirement, since Oracle's ccNUMA optimizations are based on having processes running in the same locality as the shared memory associated with those processes.

Typically, the user would add and delete CPUs using HP-UX workload management tools, such as gWLM. These tools give little control over what CPUs are moved. The performance of NUMA optimized Oracle instance is dependent on preserving the localization created at startup. Moving CPUs in and out of an Oracle instance may defeat the purpose of this optimization since processes may have to move to remote CPUs, increasing latencies of memory accesses as a result.

On any ccNUMA platform, including HP-UX, a static container provides the best possible environment for Oracle with ccNUMA optimizations enabled. If Cell Local Memory is used, a static nPar or vPar should be set up with equally sized localities (same amount of memory and CPUs) to guarantee a well balanced environment under all conditions. For instance: A large parallel query distributes the query slaves evenly across all localities, if CPUs are deleted from one locality while the query is running, it is possible that this locality becomes a CPU bottleneck and slows down the whole query significantly.

The following HP-UX partitioning features are available when running Oracle with ccNUMA optimizations enabled. Before deploying any kind of dynamic partitioning with Oracle, consult the Oracle certification data to make sure that support for the Oracle release in question exists. The metalink note 761065.1 and the white paper titled "Considerations when running Oracle Database 10*g* or 11*g* in a ccNUMA environment under HP-UX" provide more information about Oracle and dynamic reconfiguration.

- nPars: The nPar should be set up with a minimum amount of equally sized (memory, CPU) cells to meet Oracle's CPU, I/O and memory requirements. The cells should be as close to each other in the complex to minimize memory latencies.
  - Removing a cell would violate the rule of having all localities visible to Oracle at all times.
  - A floating cell may be added to an Oracle instance, and that same cell can later be removed without shutting down the Oracle database. The CPUs in the added cell will be used by Oracle, and the memory in the cell will be used by newly created Oracle processes as private memory.

Private and shared memory of processes migrating into the new cell will not migrate with the process. Migrated processes will therefore see increases in memory latencies running in the new cell. Floating cells should not be added or removed while the database is starting up or shutting down.

  - Oracle does not allocate a shared NUMA pool in the new cell, nor are new background processes (database writer) started in the new cell.
  - TiCAP CPUs in the same nPar can be used to temporary increase processing power. They are local to the Oracle instance, meeting the Oracle requirement of not causing CPU-less localities when activated and de-activated.
  - If there are other applications running in the same nPar as Oracle, and these applications are dependent on floating cells, Oracle needs to be isolated from cell reconfigurations by running in a static pset.

- vPars (Virtual partitions): vPars can be set up with CLM memory to support Oracle with NUMA optimizations enabled. The vPar has to be configured with at least one non-floating CPU per locality to make sure localities do not disappear from Oracle's view. To guarantee non-floating CPUs, the vPar has to be set up with a number of Cell Local Processors per locality. There should be at least one cell local processor, but it is good practice to have a larger number of cell local processors to keep the variability at a minimum. The HP-UX Global Workload Manager (gWLM) supports the use of cell local processors, and will not remove these from a partition. gWLM is the preferred solution for controlling resources in a dynamic vPar running Oracle in ccNUMA mode. The latest release of the vPars software (A.05.04) is ccNUMA aware, trying to keep resources balanced between localities as much as possible, and the variability as small as possible.
- Psets (Processor sets): If CLM is available, Oracle can run in a pset with NUMA optimizations enabled. The pset has to be static, no CPU movement is allowed as it may result in a 'CPU-less" locality. Workload management tools give very little control over what physical CPUs are moved in a pset environment.
- PRM and FSS: Using the Process Resource Manager with the Fair Share Scheduler is supported, as this does not cause any changes in the CPU and cell configuration.

For a more complete description of configuring nPars and vPars according to HP's Locality-Optimized Resource Alignment (LORA) recommendations, please consult the following white paper: http://docs.hp.com/en/14655/ENW-LORA-TW.pdf.

It is also possible to set up a cell based system as an Oracle RAC cluster, where one cell is one node in the cluster. This type of configuration will allow cells to be removed and added on demand, following Oracle's standard procedures for adding and removing nodes in a cluster. A cell disappearing would be recovered by Oracle's standard recovery mechanism in a RAC environment.

Oracle databases heavily dependent on dynamic reconfiguration should be configured to run with NUMA optimizations disabled. This will guarantee a much smoother integration of the dynamic features of both HP-UX and the Oracle database itself.

HP-UX 11i v3 special considerations

On HP-UX 11i v3 release 0809, a kernel parameter, 'numa_policy', can be used to control the allocation of memory. Oracle explicitly allocates memory into CLM, but relies on default OS policies for some of its memory areas (fixed SGA). This parameter should be set to '0', if Oracle runs without ccNUMA optimizations in a partition or system with 100% ILM (Interleaved Memory). If Oracle runs with ccNUMA optimizations in a system or partition with CLM (Cell Local Memory), this parameter should still be set to '0' to make sure that the Oracle fixed SGA gets allocated out of interleaved memory.

Future releases of HP-UX will change the semantics of the numa_policy parameter, and Oracle may be affected by this.

HP-UX 11i v2 special considerations

On HP-UX 11i v2, a large number of shared memory segments can cause a performance problem, as there is only a fixed number of protection ID registers for shared memory segments. On a large system (more than 12 cells), Oracle, in ccNUMA mode, will allocate more shared memory segments than available protection ID registers, causing extra software overhead in the operating system for maintaining protection IDs. HP-UX 11i v3 provides the capability to share protection IDs (IPC_RELAXED_ISOLATION); Oracle makes use of this feature, and the number of shared memory segment will not cause a performance problem.

Tuning a ccNUMA architecture by localizing memory accesses

This section discusses methods for improving locality of database applications. As these methods are based on localizing memory accesses to private memory by locking shadow processes into locality domains, they may not be suitable for all Oracle applications. Most databases will be well served by running in Oracle's default NUMA mode, letting the operating system freely schedule the Oracle shadow processes. Oracle shared servers are good candidates for a NUMA environment, there is a fixed amount of servers running, and they are all started and placed into locality domains at startup time automatically. The focus of this section is on how to localize shadow processes by scheduling them with appropriate launch policies.

The Database Resource Manager

The scheduler in the database resource manager restricts the number of runnable processes to the number of CPUs visible to Oracle. This makes it more likely for processes to stay in the locality they are started in. The DBRM is ccNUMA aware, if the scheduler decides to yield a process, the process allowed to run will be selected from the same locality as the yielding process.

Tuning with multiple listeners

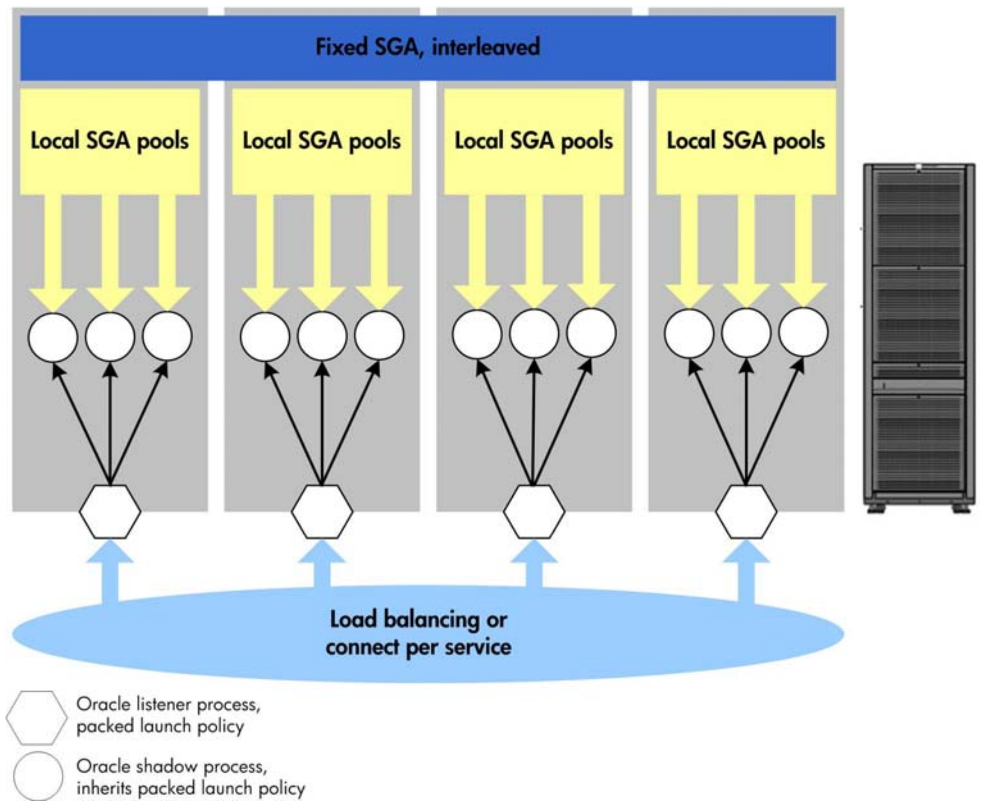
Since tuning opportunities with Oracle on ccNUMA tend to focus on maximizing local memory accesses, this section describes how to configure an Oracle instance to improve performance by maximizing the amount of CLM accesses.

As noted earlier, no current version of Oracle does placement of server (shadow) processes no HP-UX launch policies are set by Oracle, nor are those processes explicitly locked into locality domains.

The drawback of the methods described in this section is that Oracle processes will be locked into the locality domains they are started in. This is good from a memory access point of view, since all memory accesses to process private data will be local, but the processes are not allowed to migrate out from their locality domains. This could obviously result in an unbalanced system as users log out and log in.

Figure 8 shows an example of achieving maximum memory locality in a ccNUMA-based server.

Figure 8. Oracle ccNUMA tuning


Fixed SGA, interleaved
Local SGA pools
Local SGA pools
Local SGA pools
Local SGA pools
Load balancing or connect per service
Oracle listener process, packed launch policy
Oracle shadow process, inherits packed launch policy

To make sure that Oracle shadow processes never migrate between locality domains, you must set appropriate HP-UX launch policies. Normally, this could be a difficult task as potentially thousands of processes will be executing and terminating under Oracle control; moreover, Oracle does not set launch policies. However, since launch polices are inherited, you simply need to set the appropriate policy for the Oracle listener process, which is responsible for spawning the shadow processes on incoming connections. Enough Cell Local Memory should be configured to hold the private data for all the shadow processes, the 7/8 rule of thumb for CLM is appropriate for this type of configuration. Some interleaved memory is still needed for the Oracle fixed SGA.

This configuration assumes the use of private Oracle server processes, which is the most common way of connecting to a database, especially in benchmarking environments. There is no way of efficiently controlling the placement of shared servers, as these will be started by the database at database startup.

The following steps provide an example of configuring an Oracle instance to improve performance by maximizing the amount of CLM accesses:

1. Configure the Oracle network configuration file, 'listener.ora,' for multiple listeners. In a four-cell system you would configure four listeners (listener1 – listener4, for instance). A sample 'listener.ora' file follows:

Note:
Consult the appropriate Oracle documentation for information on setting up the Oracle network configuration file.

```
LISTENER1 =
(DESCRIPTION_LIST =
   (DESCRIPTION =
      (ADDRESS_LIST =
          (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
       )
      (ADDRESS_LIST =
          (ADDRESS = (PROTOCOL = TCP)(HOST = host1)(PORT = 1521))
       )
    )
 )

# SID list of the listener listener1
SID_LIST_LISTPROD1 =
   (SID_LIST =
      (SID_DESC =
         (SID_NAME = PLSExtProc)
         (ORACLE_HOME = /oracle/ora10g)
         (PROGRAM = extproc)
       )
      (SID_DESC =
         (GLOBAL_DBNAME = orcl10g)
         (ORACLE_HOME = /oracle/ora10g)
         (SID_NAME = orcl10g)
       )
    )

# listprod2 is the name of the second  listener
LISTENER2 =
(DESCRIPTION_LIST =
   (DESCRIPTION =
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
       )
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = TCP)(HOST = host1)(PORT = 1526))
      )
   )
)

# SID list of the listener listprod1
SID_LIST_LISTPROD2 =
   (SID_LIST =
      (SID_DESC =
         (SID_NAME = PLSExtProc)
         (ORACLE_HOME = /oracle/ora10g)
         (PROGRAM = extproc)
       )

     (SID_DESC =
        (GLOBAL_DBNAME = orcl10g)
        (ORACLE_HOME = /oracle/ora10g)
        (SID_NAME = orcl10g)
      )

<< repeat for listener3 and listener4 >>
)
```

2. Start the listener processes, giving them the packed launch policy. This example assumes four cells:

```
mpsched –l 1 –P PACKED lsnrctl start listener0
mpsched –l 2 –P PACKED lsnrctl start listener1
mpsched –l 3 –P PACKED lsnrctl start listener3
mpsched –l 4 –P PACKED lsnrctl start listener4
```

Use either Oracle load balancing or a transaction monitor to evenly distribute connections to the listener processes. The listeners spawn off shadow processes that inherit the packed policy and, thus, can only run within the locality domain in which they started, with optimal access to memory.

This methodology can be used to distribute different services within the same database instance, providing a useful mechanism for setting up services. If listeners were set up to handle different services, then each service would be restricted to its own locality domain(s), achieving good separation between the services. In addition, since Oracle tries to keep the database block buffers close to the shadow processes, there would be minimal buffer cache contention between services.

If the application spawns parallel query slaves, Oracle's placement strategy will override the PACKED policy given to the query coordinator, and slaves will be evenly distributed across the locality domains.

Note, that not all applications lend themselves to a multi-listener configuration like this, and can be run only in a single listener configuration.

Tuning with a single listener

In some cases, it is not practical to use many listeners. The listener process is still the only way of controlling placement of server processes. In the case of a single listener process (or less listener processes than locality domains in the system), you can still control placement by giving the listener either the Round Robin launch policy (RR) or the Least Loaded launch policy (LL):

```
mpsched –l 3 –P RR lsnrctl start
mpsched –l 4 –P LL lsnrctl start
```

The Round Robin policy will round robin the processes to be started through the locality domains, the LL policy will look for the "least loaded" locality domain to start the server process in. Processes started with either of these launch policies will never migrate out of the domain they are started in, thus providing maximum locality for their memory accesses.

Parallel query slaves will be launched using Oracle's default placement strategy, evenly distributed and locked into the locality domains available.

This method also works best, when Oracle runs in a static container, as the load is evenly distributed across locality domains with the round robin (RR) launch policy.

Tuning with psets

It may be tempting to use processor sets (psets) to control placement of Oracle processes, splitting up the Oracle instance in several psets. For instance, there could be a pset per locality domain, or even smaller psets making sure all processors in the pset share the same memory busses for optimal performance. However this method is not recommended, for the following reasons:

- Oracle would not know the correct amount of processors it has access to. The user would have to force the correct setting for CPU_count by setting this parameter every time the number of CPUs changes in the psets running the Oracle instance.
- Oracle may have the incorrect view of the amount of locality domains it has access to.

- Oracle background processes may be pushed out of the locality domains Oracle wants to run them in.

Oracle automatic memory management and ccNUMA

Oracle 11*g* has a new memory management feature that allows users to set one upper bound for Oracle's memory use. Oracle will automatically tune the sizes of the different memory pools within the SGA as well as the amount of shared memory (SGA) and private memory (PGA). The user can specify the memory limit through the parameters 'MEMORY_TARGET' and 'MEMORY_MAX_TARGET'. 'MEMORY_TARGET' is a dynamic parameter, and it can be bumped up to the value of 'MEMORY_MAX_TARGET'.

This feature does not work if memory is pinned, the parameter 'LOCK_SGA' cannot be set, if 'MEMORY_TARGET' is set. The use of asynchronous I/O through '/dev/async' may also have an impact on automatic memory management. See the section '/dev/async and memory management' for more details.

This is an example of Oracle's memory allocations, when starting in automatic memory management mode, with MEMORY_TARGET set to 54GB (this is on a 3-cell server):

```
[13159] shmget(0, 0x230000000, IPC_CREAT|IPC_EXCL|0x8000|0x80000|0660) = 4980744
[13159] shmget(0, 0x631000, IPC_CREAT|IPC_EXCL|0x8000|0x80000|0660) = 229385
[13159] shmget(0, 0x230000000, IPC_CREAT|IPC_EXCL|0x8000|0x80000|0660) = 229386
[13159] shmget(0, 0x230000000, IPC_CREAT|IPC_EXCL|0x8000|0x80000|0660) = 229387
[13159] shmget(0, 0x206000, IPC_CREAT|IPC_EXCL|0x80000|0660) = 229388
[13159] shmget(0, 0x670000000, IPC_CREAT|IPC_EXCL|0x80000|0660) = 229389
[13159] shmget(0, 0xea4a000, IPC_CREAT|IPC_EXCL|0x80000|0660) = 229390
[13159] shmget(0, 0x2000, IPC_CREAT|IPC_EXCL|0x80000|0660) = 229391
```

If MEMORY_TARGET is specified, Oracle will allocate 60% of this as NUMA pools in each locality, and 40% as interleaved memory. This 40% is marked for later potential use, if the SGA needs to grow. Oracle's automatic memory management features have thus not been optimized for ccNUMA, and applications heavily dependent on these features are better served by running Oracle with ccNUMA optimizations disabled. If fact, on some platforms, setting MEMORY_TARGET will automatically disable NUMA optimizations.

Verifying process placement and memory layout

Oracle's memory requirement requests (CLM vs. ILM) to the OS, are hints only. HP-UX does not guarantee that Oracle gets the memory it requests, in fact Oracle has no way of knowing whether enough CLM and ILM was configured on the system to meet its memory allocation requests. HP-UX will always give the requested amount of memory. For example, if enough ILM is not available, memory will be given from CLM instead.

Unfortunately, there is no simple way of querying Oracle's memory configuration. A system call trace (tusc) as the one in the previous section shows what Oracle requests, but it does not give any clue to where the memory is actually allocated. Furthermore, since some of the Oracle memory is not yet used, or the physical pages have been released with madvise() it may be hard to capture the real memory configuration with any existing tools. The HP-UX man page for the pstat_getlocality() call has an example that can be used to query the memory use on a system.

As far as process placement goes, the 'mpsched –q' command can be used to query the placement of Oracle processes.

Oracle and ccNUMA, Summary

ccNUMA is an evolving architecture both at the system level, OS level and at the Oracle level. The architecture leaves the Oracle user with many alternatives for deploying Oracle on a large server.

Running Oracle with ccNUMA optimizations enabled, will attempt to use the ccNUMA architecture to its advantage by localizing memory accesses as much as possible. A static nPar is the best choice for running Oracle with ccNUMA optimizations enabled, but other types of ccNUMA aware containers will work as well.

Oracle Database running with ccNUMA optimizations will realize the best potential performance gains in a static environment where CPUs and memory are not moved dynamically in to and out from the Oracle instance. If dynamic reconfiguration is used, the variation should be kept to a minimum, otherwise the ccNUMA optimizations may be voided to some extent. Oracle's automatic memory management features (MEMORY_TARGET) also work best, when ccNUMA optimizations have been disabled.

Performance can be further improved by setting launch policies for server processes, guaranteeing that a server process never migrates from the locality domain it was started in. The 'PACKED' or 'RR' launch policies can be used to control the placement of server processes. Launch policies are inherited by process children, but Oracle has a placement strategy of its own for parallel query slaves, making sure that slaves are distributed evenly across locality domains.

HP further recommends that the static nPar (or static vPar) is configured according to the following rules:

- Minimize the number of locality domains needed, and use domains as close to each other as possible.
- Configure a symmetric amount of memory in each locality.
- Configure a symmetric amount of processors in each locality domain.
- Configure CLM. When Oracle is running with NUMA optimizations enabled, enough CLM has to be configured to hold Oracle's NUMA pools. The amount of CLM is application dependent; the size of the Oracle SGA, the number of Oracle background processes, shared servers and parallel query slaves should be taken into account. The 7/8 CLM configuration is probably a good rule of thumb. On HP-UX, CLM is configured with the parmodify command.

In many cases, turning off Oracle's ccNUMA optimizations by setting the two parameters discussed earlier, will provide enough performance and is the easiest way of adapting to the dynamic features of HP-UX and Oracle. In this case, the server should be configured with a minimum amount of cell local memory, (10% is likely a good rule of thumb to let the operating system take advantage of its ccNUMA optimizations) or left at the default 100% ILM, allowing Oracle to run in interleaved mode without ccNUMA optimizations enabled.

Conclusion

The 10*g* and 11*g* versions of the Oracle database have included features to optimize performance in systems with non-uniform memory access, such as the HP Integrity servers. Utilizing this architecture can provide performance gains for performance critical applications. The architecture can be further boosted by letting HP-UX control launch policies for Oracle shadow processes. In many cases this kind of optimization is not needed, and applications are well served by Oracle's default mode of operation. Oracle can also be executed in interleaved mode, simulating a large SMP machine; this will provide lots of flexibility in dynamically reconfiguring partitions running Oracle.

The Oracle DBRM (Database resource manager) was also described, and it was shown, that some integration with the HP workload management product is possible, letting the Oracle DBRM take care of the Oracle internal resources.

For more information

HP-UX, www.hp.com/go/hpux

HP ActiveAnswers, www.hp.com/solutions/activeanswers

HP Integrity servers, www.hp.com/go/integrity

Locality-Optimized Resource Alignment, http://docs.hp.com/en/14655/ENW-LORA-TW.pdf

Oracle Database http://www.oracle.com/database/index.html

To help us improve our documents, please provide feedback at
http://h20219.www2.hp.com/ActiveAnswers/us/en/solutions/technical_tools_feedback.html.

Technology for better business outcomes

4AA2-0547ENW, Revision 2, February 2009