The Oracle Database Resource Manager: Scheduling CPU Resources at the Application Level

Ann Rhee, Sumanta Chatterjee, Tirthankar Lahiri
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
ann.rhee@oracle.com, sumanta.chatterjee@oracle.com, tirthankar.lahiri@oracle.com

Introduction

Modern Database Management Systems have numerous mechanisms for allowing administrators to manage the usage of system resources such as disk space and memory, yet they have typically not included mechanisms for controlling the usage of the all-important system resource of CPU-time. Traditionally, this functionality has been left to the host Operating System scheduler. Because of this, the only recourse for Oracle to control users' CPU usage was to impose hard limits on CPU consumption. If a user session exceeded its CPU limit, it was terminated. In Oracle8*i* we introduced the Oracle Database Resource Manager, a novel DBMS CPU management mechanism that allows a database administrator to delineate logically distinct units of a workload and to partition CPU resources between these units. Performance investigations show that running with the Database Resource Manager imposes no measurable overheads on a workload, and the use of the Resource Manager actually improves performance with large user-populations.

Several commercial Operating Systems now support CPU resource management via a fair-share scheduling mechanism. We believe that for large, complex applications such as Oracle, CPU resource management is better handled within the application itself. Since the scheduling is done within the Oracle application and the application is portable across numerous platforms, the Oracle Resource Manager is able to consistently enforce its scheduling policies, independent of OS platform.

The Need for Database CPU Management

In an attempt to consume all the available CPU bandwidth, DBAs often tend to over-configure the number of server-processes (e.g. ten times the number of CPUs). Over-scheduling can be inefficient: scheduling too many processes increases context switch overheads and increases the probability of preempting processes within critical sections, leading to performance bottlenecks. On the other hand, under-scheduling can lead to under-utilized systems. The simple approach of allowing the OS to treat all server processes uniformly may be adequate when the workload is homogenous and response time requirements are the same for all users. However, workloads within a single instance of a database are becoming exceedingly complex and heterogeneous, consisting of many different classes of users and activities as illustrated in the following examples:

- **Mixed-mode workloads***:* Workloads may consist of an OLTP portion with large user groups issuing frequent short transactions and a DSS/batch portion consisting of complex data-intensive queries. It may be necessary to allow the OLTP users a greater share of system bandwidth and to limit the intrusive effects of the DSS queries during normal business hours. It may also be necessary to control the number of active DSS sessions to conserve sparse resources, such as temporary disk space required for sort operations. DSS sessions may be allowed a greater share of the system after close of business, when OLTP activity is likely to be lighter.
- **Consolidation within the enterprise**: Even within a single enterprise, the trend has been to consolidate business data management into fewer and larger database systems. Response time requirements for different groups within a company (Accounts Payable, Sales, etc.) may be quite different.
- **Application hosting**: For economies of scale, companies may out-source database administration. A third-party administered large-scale database may host applications from a number of different client companies who pay for varying proportions of system usage. Different clients should see throughput proportional to their investment.

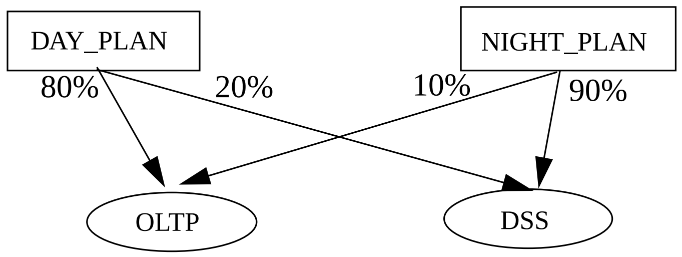
All of the above scenarios require the server to have more explicit control over the amount of CPU resources consumed by the different pieces of a workload.

Overview of Database Resource Manager

The Oracle Database Resource Manager includes a scheduling mechanism that keeps track of CPU time consumption and performs scheduling decisions at fixed time-intervals. It also contains a mechanism to the control the number of concurrently active sessions. Once all active session slots are filled, subsequent sessions attempting execution will be queued and let into the system as other active sessions finish their work. The Resource Manager also allows the administrator to set scheduling policies based on the predicted execution time of a query (e.g. only run very long queries in a low priority group, or prevent very long queries from even entering the system). The Resource Manager has a set of PL/SQL based packages that allow the administrator to define the scheduling policy for a workload. The key components of the Resource Manager interface are as follows:

- **Resource consumer group**: A resource consumer group is a group of active user sessions and defines a logical component of a workload. This is a concept specified by the administrator. Resource consumer groups may be defined at the granularity of the user so that all sessions created by a particular user are assigned to the same group. Group association for a user session may be changed dynamically from the user's default so that more fine-grained division of CPU time can be achieved across user-sessions.
- **Resource plan***:* A resource plan is a description of how CPU time should be allocated between different consumer groups. Conceptually, a resource plan is a node in a directed acyclic graph of CPU allocations known as the *resource plan schema*. A child of a resource plan is either another resource plan (non-leaf node in the plan schema) or a consumer group (a leaf node).
- **Resource plan directive**: Weighted directed arcs within a resource plan are known as *Resource Plan Directives*. A child of a resource plan directive is either a resource consumer group or another resource plan.

Example: A mixed-mode workload with an OLTP/DSS component (described earlier). can be run according to two different plans depending on the time of day. This can be described by the following single-level plan schema.



The Database Resource Manager allows the administrator to dynamically switch between DAY_PLAN and NIGHT_PLAN as needed.

How Does the Database Resource Manager Work?

The Database Resource Manager controls the number of concurrently running Oracle processes at any given time. The number of concurrently running processes roughly equals the number of CPUs on the host machine (or Oracle's partition of the host machine). By only having a small number of OS runnable processes at any given time, the Database Resource Manager takes scheduling decisions away from the OS scheduler; the OS scheduler has no choice but to run the processes which are runnable. Running processes calculate their CPU usage and yield to other Oracle processes (as chosen by the Database Resource Manager) when their quantum expires.

The Database Resource Manager is a module, not a process. Each running Oracle process or thread must call into the Resource Manager scheduling code periodically. This code determines whether the running process can continue to run or must yield to another Oracle process. If it must yield, the Resource Manager code determines which process can run in its place. It then signals this process and the process whose quantum had just expired simply puts itself to sleep. Using this method, the Database Resource Manager can portably adhere to an administrator specified CPU scheduling plan.

Existing Operating System Resource Managers

Most large operating systems provide functionality to support user-supplied scheduling plans, similar to those described in the previous section. Some examples are Sun Microsystems' Solaris Resource Manager (SRM) [3], Hewlett-Packard's HP/UX Process Resource Manager (PRM) [2], and IBM's OS/390 / AIX Workload Manager (WLM) [1]. The fair-share scheduling component of these OS resource managers are all fairly similar. The resource managers allow the system administrator to group processes or threads into different scheduling classes. The administrator can then allocate either percentages of CPU time or CPU shares to the different scheduling classes. The system administrator can set up "rules" which specify which processes belong to which scheduling classes. These rules may be based on, but are not limited to, the owner of the process, name of the process, and which executable the process is running. For example, the administrator could specify that all processes owned by "Bob" belong to Scheduling_Class_1. Or, any process which runs the executable /usr/games/fun belongs to Scheduling_Class_2.

Once all processes have been assigned a scheduling class, the OS scheduler takes over. The algorithms used to implement fair share scheduling differ between OS resource managers, yet the goal is always the same: to adhere to the scheduling policies set forth by the system administrators. For example, say the system administrator specified 75 shares of CPU to Scheduling_Class_1, and 25 shares to Scheduling_Class_2. The OS scheduler would roughly run processes in Scheduling_Class_1 three times as often as those in Scheduling_Class_2.

Benefits of Managing CPU at the Application Level

Since most large operating systems have implemented CPU resource management a common question is: Why not simply propagate database administrator specified Oracle CPU resource requirements to the OS and let the OS scheduler manage CPU resources between Oracle processes? This is a bad idea for several reasons:

- **Knowledge of Oracle resources** - Since it runs within the Oracle kernel, the Database Resource Manager is aware of what Oracle processes are executing and when shared Oracle resources such as latches (lightweight mutual exclusion primitives) and enqueues (database locks) are held. Therefore, the Database Resource Manager is able to do intelligent scheduling - e.g. it would allow lower priority processes to run if they hold shared resources that are needed by higher priority processes. This avoids priority inversion problems. There is no way for the OS to know this type of information and thus schedule intelligently.
- **Process Grouping** - Most rules used by OS resource managers to group processes into different scheduling classes are insufficient for differing Oracle processes. For example, all Oracle processes belong to the same OS userid and run the same executable. Yet, since the Database Resource Manager runs in the Oracle kernel, it knows which process is running on behalf of which Oracle user (note that the concept of an Oracle user is different from the concept of an OS user). This gets even more complicated since some Oracle processes may run on behalf of different Oracle users at different times. For example, using Oracle's shared-server feature a single server process can service multiple database users. Similarly, parallel slaves which are spawned on behalf of an Oracle coordinator process should inherit the Oracle userid of the originating coordinator, but only for the duration of that particular parallel operation. The Database Resource Manager is able to associate these processes with the correct Oracle users.
- **Integration with other resource management features** - The Database Resource Manager is currently closely tied with Oracle's parallel execution feature. The parallel execution code uses Database Resource Manager information to intelligently decide how many parallel slaves to spawn and on which Oracle instances to spawn them. In the future, the Database Resource Manager CPU management code may be tied directly with Oracle's memory management code and/or temporary space management code. It would be quite difficult and complicated to share information back and forth between Oracle kernel components and an outside OS CPU resource manager.

- **Portability and consistency** - Since the Database Resource Manager runs in the Oracle kernel, it is completely portable on all platforms on which Oracle is supported. Also, the behavior of the Database Resource Manager is consistent across all platforms. If the OS resource managers were to take over Oracle resource management, Oracle would have to ensure that this worked across all OS platforms (impossible since not all OS's have resource managers) as well as ensure that the behavior was consistent on different operating systems.
- **Statistics gathering** - The Database Resource Manager currently gathers many statistics while running - e.g. average run queue length per consumer group, average wait time per group, total CPU consumption per group. This information is gathered and can by displayed using the standard Oracle system views. The OS may not be able to give the same, consistent statistics to the administrator.
- **Stability** - Oracle will most likely be adding new features to the Database Resource Manager and possibly changing existing features based on user feedback. If Oracle hands off implementation of CPU resource management to the OS, it will be quite difficult, if not impossible, to change or add functionality. To accomplish this, we would need *all* OS vendors to agree to changes and implement them. This would prove to be quite a difficult task.
- **Dynamic reconfiguration and adaptability** - Many operating systems support dynamic reconfiguration of system resources - e.g. CPU, memory. The Database Resource Manager is able to adapt to changing system configurations without having to bring down the database instance, thus increasing availability.

Conclusion

The Oracle Database Resource Manager has a powerful DBMS CPU management mechanism allowing fine-grained control over the utilization of processing time among differing Oracle processes. Many operation system resource managers have similar functionality, allowing system administrators control over how CPU resources are allocated among generic OS processes. We believe OS resource managers are very valuable for certain workloads - e.g. a set of smaller, more homogenous applications running on a single large machine. Yet, due to the size and complexity of an application such as Oracle (which contains a Virtual Operating System running on top of a commercial operating system) letting a fair share OS scheduler schedule amongst differing Oracle processes was not a feasible option. The best solution was to implement a portable, consistent, internal CPU scheduler within our own Database Resource Manager.

References

1. Gfroerer, D., Castro, C., *AIX 5L Workload Manager*, IBM Corporation, 2000.
2. Hewlett-Packard Corporation, *http://www.software.hp.com*
3. McDougall, R., A. Cockcroft, E. Hoogendoorn, E. Vargas, *Sun Blueprints Resource Management*, Sun Microsystems Press, 1999.
4. Rhee, A., *Functional Specification for the Database Resource Manager Oracle8i*, Oracle Corporation, May, 1998.