Cloud Konsolidierung mit Oracle (RAC) – Wie viel ist zu viel?

Markus Michalewicz
Oracle Corporation
Oracle HQ, Redwood Shores, CA, USA

Key words: **cloud, consolidation, Oracle RAC, database, limits**

Introduction

"Grid", "virtualization" and last but not least "cloud consolidation" have one thing in common; at some point in time, they used to be ideas or approaches, intended to reduce IT costs by providing a simpler way of managing the data center and by utilizing the existing infrastructure in a more efficient way. The latter, in rather simple terms, means using the existing hardware more intensively. This leads to the question: "How much is too much?" In more Oracle specific terms and mainly considering the database, this means: "How many databases can one consolidate on a server? How many databases can one run in an Oracle Real Application Clusters (RAC) based cluster?" This article tries to answer these questions and gives some guidance on how respective limits can be determined.

Disclaimers

This article intends to provide practical guidance on how many databases can reasonably be consolidated on one server or on a shared (Oracle RAC based) clustered infrastructure. It is not the intention to find a limit as to how many (potentially unused) databases can be consolidated on a server overall, as this number would not have any practical significance.

On the other hand, this article does not provide any guidance on performance. The values, limits and resulting recommendations presented in this paper will not consider or ensure that the database that will be restricted as a result of the recommendations made in this paper will provide the performance required by the application using the database. The intention is to ensure that the system will remain stable if the consolidated databases will be utilized to their full extends within the boundaries given.

It also needs to be noted that although the author is an Oracle representative, this paper and the respective presentation do not claim to be a rule set or official Oracle guidance on this matter. They cannot be used to determine supportability or certification. The values, limits and resulting recommendations presented in this paper are based on years of experience, gathered and put together by the author based on research amongst Oracle RAC customers and members of the Oracle RAC development team. Most of these recommendations are not enforced by any part of the code at the moment of writing, which is why they are not a rule in general and unless explicitly stated.

Registered, Starting, Running

When considering limits regarding the maximum number of databases on a server, three states need to be distinguished: “registered”, “starting”, “running”, defined as follows:

Registered describes that *n* databases and f(*n*) DB instances are configured to potentially run on one server or a number of servers in the cluster. The term is based on cluster jargon, in which Oracle Databases are registered with the Oracle Clusterware (in the Oracle Cluster Registry, also known as OCR).

Starting describes the process of an Oracle Database instance coming up, which includes respective allocation of memory and processes on one server. Starting for this paper also describes the process of opening an Oracle Database by at least one instance and registering this database as well as respective processes with Oracle Clusterware for the purpose of fencing, if an Oracle RAC database is used.

Running describes the operational state of an Oracle Database with at least one instance started and the respective database opened while accepting or already serving database connections from clients.

Typically, any limit regarding the number of databases that can be consolidated on one server targets the limits for databases in running state, as expressed by the question: “How many databases can one run on a server?” If a cluster-based consolidation is used, starting databases need to be considered in addition, as databases will usually have to register with Oracle Clusterware soon after the startup process has completed. Too many databases starting at the same time may therefore lead to issues registering.

One can compare this fact with electric motor driven devices all being started simultaneously on the same fuse. As motor driven devices will consume most of their energy at the moment of startup, starting all devices at the same time may lead to a blown fuse, which otherwise could sustain a steady use of all these devices at the same time just fine. In a lot of ways, determining the limits of a given system is based on such “rules of common sense”. *Example*:

Based on a given cluster system, one customer had consolidated and registered 117 databases in a way that the equal amount of database instances were meant to start on one server of the cluster. Considering the yet to be discussed rules, the server is not suitable to run this number of databases. However, the customer complained that they were not even able to start this number of instances on the server, receiving error messages on Oracle Clusterware level, although the database instances were not being used at the time. On the other hand, another customer, having consolidated and registered 173 databases in a similar configuration with the server not being able to run all these database instances under load at the same time, did not experience any issues during startup. The difference between these customers was that the customer having no issues registered databases, but never started them at the same time. The system was a development and test system used across the world in a way that the use of the databases shifted with geographic regions and over time. Hence, new database instances were only started as a new geographic region started its shift, but not all at the same time. Note that both customers ultimately had to reduce the numbers of instances per server.

Database Consolidation

When planning to consolidate various databases on one system, first one database instance that does not share the system with any other database should be considered. Doing so, one will notice that the Oracle Database, especially the database instance, is basically designed based on the assumption that it is the only instance on a given server. The side effect of this assumption is that it will – by default – try to consume as many hardware resources as it can potentially get whenever it will need them. This basic assumption is obviously contradictory to the idea of consolidating more than one instance on a server. The consequence therefore is to restrict each instance from behaving this way.

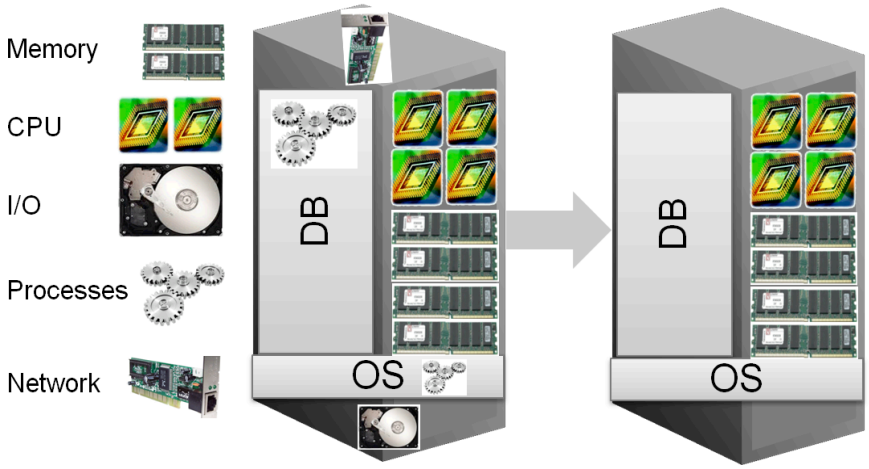

Figure 1: Hardware resources to consider when using an Oracle Database

The main hardware resources to consider when using an Oracle Database are memory, CPU, I/O, processes in general and network capacity. How these resources are used and how one can regulate their consumption depends on the resource itself.

When consolidating databases, the first two resources that need particular attention are memory and CPU. The Oracle Database has traditionally provided a variety of ways to manage memory allocation and consumption. These mechanisms remain when consolidating more than one instance on a server. Based on sga_target and pga_aggregated_target an Oracle Database instance can easily be domesticated regarding its memory consumption. Even for a single instance database that does not share the system with any other database, it is prudent to allocate only so much memory to the database that the system as a whole does not de-stabilize, which can easily happen due to intensive swapping for example.

Therefore, a lot of customers restrict the memory allocated to the Oracle Database to 70-90% of the physically available memory. These values may vary and depend on the actual system and the overall amount of memory available. In average, allocating around 80% of the physically available memory seems a good rule of thumb. One may also deduct a fixed amount of memory for the OS for example and as needed, as the OS memory consumption will not increase significantly per database instance added to the system. In consolidated environments the sum of all memory allocated to the Oracle instances by means of sga_target and pga_aggregated target should not exceed 80% of the physically available memory.

In addition to memory, CPU usage per database instance should be regulated in a consolidated environment, as otherwise, the same assumption applies; the database instance assumes to be the only one on the server and consequently would try to request as many CPU cycles as required and requested indirectly by the application. The initial idea is therefore to divide the available CPU power and distribute it over the instances running on one server at a time, for which one could use an OS based resource manager.

With Oracle Database 11g Release 2, however, Oracle announced a new feature called Instance Caging, which significantly simplifies the setup of consolidated environments as it makes OS based resource managers obsolete for the most parts and assuming that the only CPU consumption that needs to be regulated is the one for the Oracle Database instances. Instance Caging in very simple terms is based on modifying the CPU_COUNT parameter for an Oracle Database instance in order to prevent an instance from consuming more CPUs than specified by this parameter. (For more information on Instance Caging, please, see: http://www.oracle.com/technetwork/database/focus-areas/performance/instance-caging-wp-166854.pdf)

In simple terms, Oracle now allows adjusting the CPU_COUNT parameter of an instance, for which the rule formerly was that this parameter is not to be manually altered. Altering this parameter is, however, more or less required in consolidated environments in order to prevent exhausting CPU consumption by a specific instance. Ideally, the parameter is used in conjunction with Instance Caging, which in addition requires using a database resource manager plan (e.g. the default plan) to take effect.

Setting the CPU_COUNT parameter alone, however, already ensures a restricted CPU usage. It also prevents an exhausting allocation of additional database processes and resources, as a lot of other database resources are allocated based on a function of CPU_COUNT. Reducing the value for CPU_COUNT would therefore reduce the allocation of additional database resources as shown in figure 2 below.

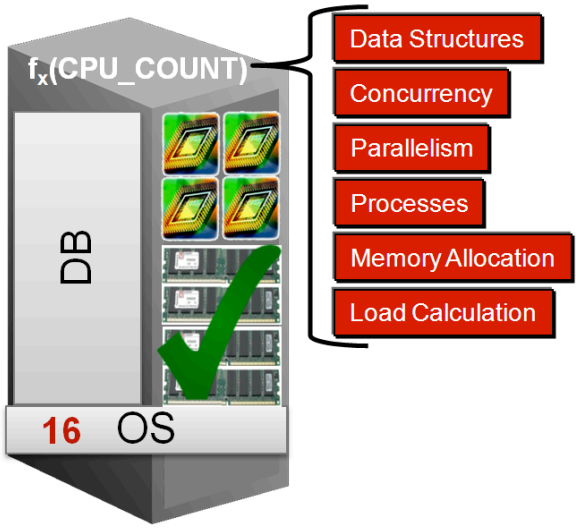

Figure 2: Additional database resources dependent on CPU_COUNT

Over-Provisioning

Following up on Instance Caging, one will find that Instance Caging allows caging instances using a partitioning as well as over-provisioning approach. Over-provisioning has become quite popular when consolidating either on physical hardware or in virtual machines. In either case, over-provisioning cannot be used to increase the amount of available hardware. The available hardware will still be the limit.

Over-provisioning describes an approach to distribute more CPU resources across processes on a particular system than physically available on the machine. Using virtual machines, one can set up the virtual machines in a way that the number of vCPUs allocated to the virtual machines exceeds the number of physical CPUs on the hosting machine. In case of Instance Caging over-provisioning is achieved by setting the sum of all CPU_COUNT parameters of all instances higher than the number of physical CPUs available in the machine, as figure 3, Over-Provisioning Approach, illustrates (the number of CPUs in the machine is 16).

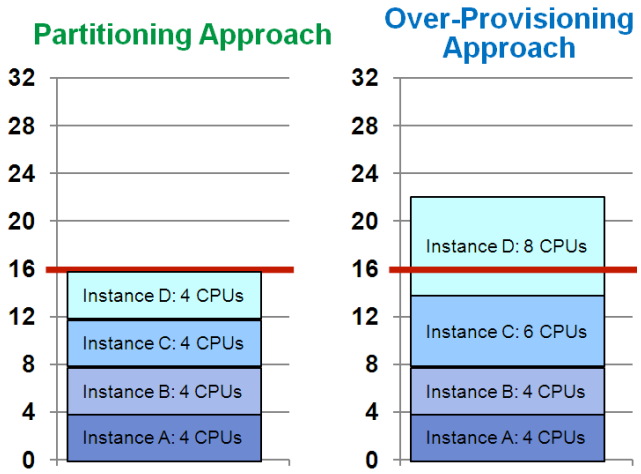

Figure 3: Instance Caging approaches based on a 16 CPUs server

While the Instance Caging white paper will provide more details on the over-provisioning approach, one needs to consider that a machine cannot run with 200% of its processing power for a longer period of time. Hence, purposely over-provisioning a machine will probably not result in the desired effect. CPU over-provisioning based on Instance Caging or virtual machines (hosting databases) for that matter is based on the idea that not all instances will run and require the full amount of assigned CPU power at the same time. Occasionally, all instances might run at the same time, but not for a longer period of time.

Based on this assumption, a limit as to how much one can overprovision the system is actually not required. However, as there may be periods of time when more CPU cycles are requested than physically available, it is a considerable approach to limit those requests, unless one is comfortable in letting the OS decide how to schedule those requests. As a rule of thumb, it is therefore recommended to overprovision with a maximum of three times the CPUs available in the system. For Instance Caging this means that the sum of all CPU_COUNT parameter values per instance for all instances on one server should not exceed 3x the amount of CPUs on this server.

Oracle RAC Based Consolidation

When consolidating using Oracle RAC databases the same principles as for the single instance database remain as per server limits. Per server limits (as opposed to cluster limits) are limits that one will encounter on a particular server. In most configurations server limits are reached before other limits are reached; with the exception of (LMS) real time processes.

Unlike single instance databases, Oracle RAC databases use and rely on (to a certain degree) real time (RT) processes in the stack. Oracle Clusterware in version 11g Rel. 2 and higher uses a number of real time processes, the Oracle Automatic Storage Management (ASM) instance uses one LMS real time process and each Oracle RAC database instance uses at least one LMS RT process (cpu < 4 → 1 LMS; cpu < 16 → 2 LMS, cpu >=16 → 2 LMS + 1 LMS for every 32 CPU – numbers may depend on the Oracle RAC version used).

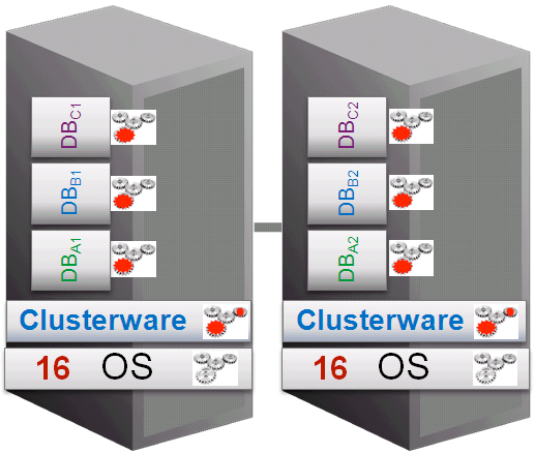

Figure 4: (LMS) RT processes used by Oracle RAC to be considered

The general rule of thumb with respect to those processes has been to limit the number of LMS RT processes per server to one less than the number of cores on the server (see My Oracle Support note 558185.1 for more details). Not only is this a conservative approach, but also a significant limit, as it could limit the number of instances significantly.

Considering the fact, however, that one RT process can only run on one CPU / core at a time, CPU contention due to RT processes should be avoided. LMS RT processes are the main concern in an Oracle RAC stack as they could potentially (depending on the usage of the database) remain on the CPU longer than any other RT process used in the stack. Other RT processes in the stack might get used frequently, but only for a short period of time per call.

Over-provisioning LMS RT processes is therefore strongly discouraged ("one cannot put 8 pounds of rice in a 4 pounds bag"). If more LMS processes than the available number of cores per server need to be started in order to satisfy a certain degree of instances per server density, additional LMS processes should be changed to time share (TS) from RT priority in order to observe the discussed limit.

Auto-Adjustment of LMS Process Priority in Oracle RAC 11.2.0.3

LMS processes will still run as RT processes per default with Oracle RAC 11.2.0.3. However, starting with this version, the database (instance) will take the number of cores per server into account and control the number of LMS RT processes for all 11.2.0.3 or later databases on the server accordingly. The goal is to keep the number of LMS RT processes equal to or lower compared to the number of cores on this server.

All LMS processes belonging to one instance will either run in RT or TS priority (without manual intervention). In the current implementation the startup order of the database instances on the server will determine the priority. In other words, adjustments to the LMS priority from RT to TS will begin with the database that was started last and continue in reverse startup order.

The number of LMS RT processes will be monitored every minute. Adjustments are triggered by instances starting or stopping. Oracle will allow changing the LMS priority for an instance manually in future releases (in 11.2.0.3, no manual alteration using sqlplus can be performed). Modifications to the LMS priority using respective instance parameters or OS commands remain unchanged and can be performed as for previous database versions.

Auto-adjustment of the LMS process priority does not apply to pre-11.2.0.3 databases. Pre-11.2.0.3 databases will not be monitored regarding their LMS process priority and will not be considered for the auto-adjustment of 11.2.0.3 databases on the same server. The same applies to the LMS process of the Oracle ASM instance. This LMS processes will not be considered when adjusting the LMS process priority of 11.2.0.3 databases and it will not be adjusted, even if Oracle Grid Infrastructure has been upgraded to 11.2.0.3, as adjustments are generally not required for this instance.

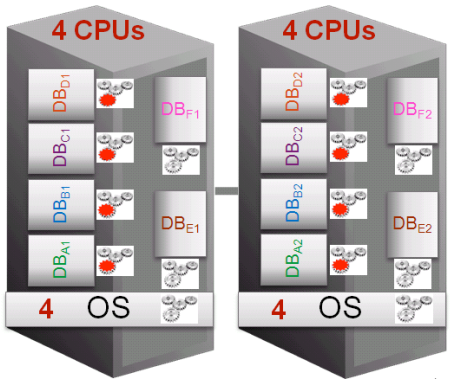

Figure 5: Auto-adjustment of LMS process priority with Oracle RAC 11.2.0.3

Additional Considerations

The numbers and recommendations given in the course of this paper consider an average use and provide recommendations based on a "rule of thumb" basis. Particular usage or certain access patterns may require adaptation of those numbers. Also, other database resources depend on CPU_COUNT as they are determined as a function of the same. Therefore, additional fine-tuning can be required and should be performed as needed. *Example*:

Using a data warehouse kind of workload, it might be a considerable approach to weigh the PGA when allocating and configuring memory as mentioned. The number of LMS processes can be reduced if the usage pattern of a database shows that LMS are not frequently used. PQ slaves follow a similar pattern. While they are generally allocated using a function of CPU_COUNT, they may be adjusted separately. Last but not least, other processes on the server may require additional adaptation.

Summary

Based on the discussion in this paper the following 5 recommendations can be made:

1. Manage Memory carefully (and dynamically)
   a. Set memory targets carefully
   b. Do not oversubscribe system resources such as shared memory identifiers and segments
2. Use CPU_COUNT and ideally Instance Caging to "cage instances"
3. Use DB Resource Manager to handle runaway queries
4. Over-provision only based on CPU_COUNT not on LMS RT processes
5. Consider additional, indirectly managed database resources as needed

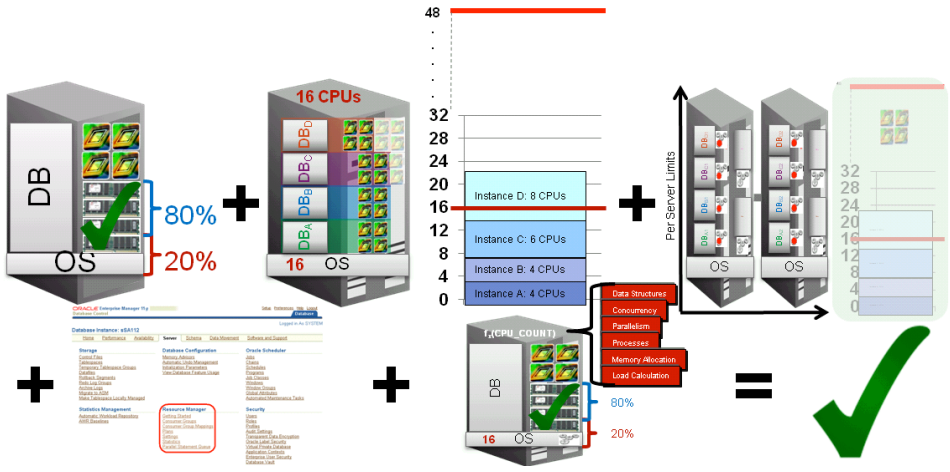

Figure 6: Summary

Markus Michalewicz
Oracle Corporation
500 Oracle Parkway, MS4OP840
USA – Redwood Shores, CA 94065

Telefon: +1(650)5065444
E-Mail Markus.Michalewicz@oracle.com
Internet: http://www.oracle.com/goto/rac