ORACLE DATABASE 12c

An Oracle White Paper
April 2014

Oracle Multitenant on SuperCluster T5-8: Scalability Study

ORACLE

Executive Overview

Consolidation in the data center is the driving factor in reducing capital and operational expense in IT today. This is particularly relevant as customers invest more in cloud infrastructure and associated service delivery. Database consolidation is a strategic component in this effort. In order to minimize capital and operational cost, an **efficient and elastic sharing of resources** is essential to **increase the density** (the number of supported databases) in such a consolidated environment.

Data centers today implement database consolidation predominantly using one of the following three options:

- databases in virtual machines (VMs) with typically one database per VM
- multiple *stacked* single-instance databases (non-CDBs) on the same operating system
- schema- or table-based consolidation inside the same database

With databases in VMs, only the physical server is shared, but each database still requires its own operating system (OS) and database instance, resulting in poor resource utilization due to mostly static allocation of resources (i.e. CPU, memory). Consequently, both capital costs (as a result of low density) as well as operational costs (many OS and DB instances to manage) are high for such deployments. non-CDBs share a common OS, but still require dedicated database instances, which prevents the sharing of database structures such as buffer cache or background processes between them. It also does not reduce management costs significantly as databases are still administered individually (i.e. with respect to backup, restore, patching, and disaster recovery). Schema- and table-based consolidation, on the other hand maximizes sharing, but lacks isolation in terms of namespace, security, lifecycle, independent patching/upgrades, and performance and is in many cases not a suitable consolidation option.

Oracle 12*c* introduces **Oracle Multitenant**, a new database consolidation model in which multiple Pluggable Databases (PDBs) are consolidated within a Container Database (CDB). While keeping many of the isolation aspects of single databases, it allows PDBs to share the system global area (SGA) and background processes of a common CDB. This paper analyzes and quantifies savings in compute resources, efficiencies in transaction processing, and consolidation density of Oracle Multitenant compared to consolidated single instance databases (non-CDBs) running in a bare-metal environment. This paper documents that **Oracle Multitenant on SuperCluster T5-8 gives 80% better performance and allows to consolidate 50% more active OLTP databases. It reduces overall resource requirements to support 252 databases by 64 CPU cores, 368 GB of memory, and 225,000 storage IOPS.** Since databases running in virtual machines have even less opportunity for sharing, the results presented in this paper also apply to VM-based database deployments, which – depending on the overhead of the hypervisor – will at best give similar performance as non-CDBs, and even worse performance as a result of virtualization overhead.

Introduction

Oracle Database Consolidation Models

Oracle databases are primarily deployed using three different consolidation models: Oracle databases deployed within virtual machines (VMs), *stacked* databases deployed on a single server or across clustered servers (non-CDBs), or databases deployed as multiple schemas consolidated in a single database[1]. Each of these models has advantages and disadvantages with respect to isolation, performance, density, and manageability. Oracle 12*c* introduces a new **in-database virtualization** consolidation model called **Oracle Multitenant**[2] which improves the consolidation density ratio and scalability as compared to the VM and *stacked* database consolidation deployment models while preserving a high level of isolation between databases. Oracle Multitenant also removes the complexity and management constraints related to each of the consolidation models depicted in **Figure 1**.

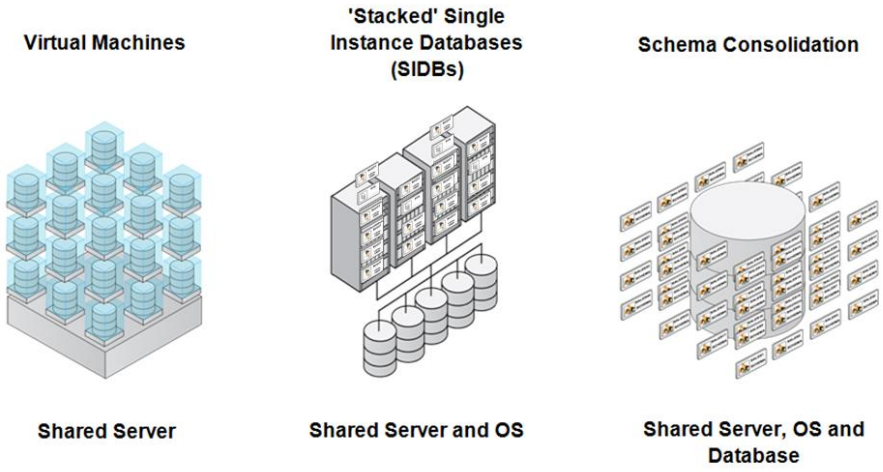

Figure 1. Most prominent consolidation models prior to Oracle Multitenant 12*c*

[1] Oracle Multitenant provides similar efficiencies and consolidation density as Schema Consolidation without the complexity of disparate schema management in patching, namespace management, granular recovery and isolation.

[2] See http://www.oracle.com/us/products/database/options/multitenant/overview/index.html
http://www.oracle.com/technetwork/database/multitenant-wp-12c-1949736.pdf
http://www.oracle.com/technetwork/database/multitenant/overview/index.html for further information.

Oracle Multitenant

In Oracle Multitenant, a common container database (CDB) provides core database functionality for all consolidated pluggable databases (PDBs). Each pluggable database is a portable, self-contained database with its own schemas, schema objects, and non-schema objects, stored in the individual PDB SYSTEM, SYSAUX, USER tablespaces and associated PDB data files. The root container database (CDB$ROOT) provides shared database dictionary structures, SGA memory and background processes.

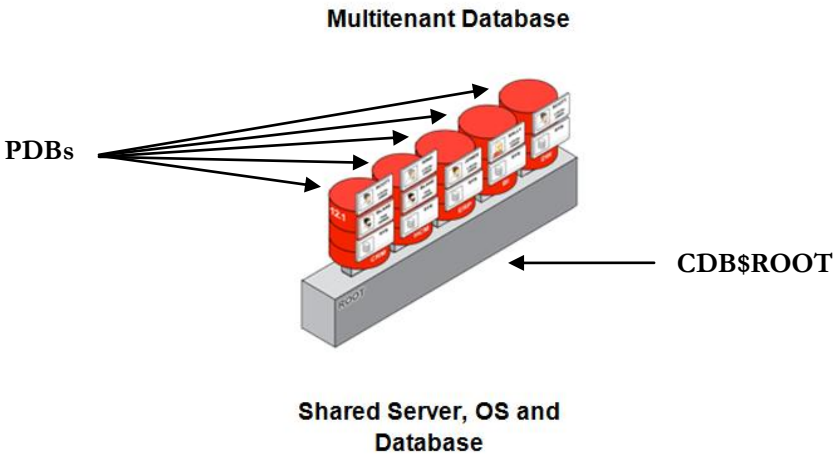

Figure 2. Oracle Multitenant Database with distinct PDBs plugged in to a single root container.

Oracle SuperCluster T5-8

Oracle SuperCluster systems are Oracle's fastest and most scalable engineered systems and are ideal for consolidation of databases and applications, and implementation of database as-a-service (DBaaS) for public and private cloud deployments. They deliver extreme performance, the highest consolidation density, availability, and efficiency, and eliminate the risks inherent in build-it-yourself architectures[3]. Oracle SuperCluster T5-8 is the latest engineered system using SPARC T series 5th generation processors.

[3] See http://www.oracle.com/us/products/servers-storage/servers/sparc/supercluster/overview/index.html for further information.

Consolidation Efficiency, Density, and Elasticity

This study provides a technical analysis of the consolidation efficiency, density, and elasticity of Oracle 12*c* Multitenant pluggable databases (PDBs) compared to *stacked* single instance databases (non-CDBs) running Online Transaction Processing (OLTP) workloads.

The first experiment establishes the baseline performance of pluggable databases by comparing a traditional single instance database to a single PDB inside a CDB, and demonstrates that in-database virtualization with Oracle Multitenant comes at measurably insignificant **overhead**. The second experiment compares the **efficiency** at large scale by consolidating 252 databases using PDBs vs. non-CDBs, analyzing in detail how savings in background processes and memory use contribute to 80% higher aggregate throughput of PDBs. For the third experiment, the per-database throughput is kept the same between PDBs and non-CDBs, demonstrating that savings in processing efficiency allow for 50% higher **density** in consolidation of databases with Oracle Multitenant than with single instances for given requirements. In real-world deployments, the workload is not typically static, but changes dynamically over time, requiring on-demand allocation of resources. The last experiment therefore demonstrates the **elasticity** of Oracle Multitenant in dynamically allocating buffer cache to pluggable databases as needed, whereas the buffer cache size for non-CDBs is static. This last test illustrates the Oracle Multitenant performance advantage over non-CDBs even at a smaller scale of consolidated databases. In all of the experiments, this study shows significant savings in compute resource consumption and measured gains in performance for the Oracle Multitenant consolidation model, from small scale consolidation as in the last test of 8 consolidated databases up to a denser scale of 252 databases.

Practical Benefits of Oracle Multitenant

This study highlights the advantages of Oracle Multitenant at varying consolidation densities through the efficient use of shared database infrastructure, specifically shared memory and a common set of background processes. Additional practical benefits of Oracle Multitenant, as a result of efficient consolidation, include:

- Reduced management complexity through Manage Many-as-One
  Database management can be executed from common or local users where a common user can manage all, or a subset of, PDBs. If necessary, management isolation can be restricted to the individual PDB through local users.
- Agile data mobility through database unplug/plug
  In-database virtualization allows for distinct PDB tablespaces and data files. Through simple SQL command line execution, PDBs can unplug from one container and plug in to another.
- Simple database patch/upgrade management.
  The database version of all tenants is determined by the version of the CDB. An upgrade of the CDB automatically upgrades all its tenants. Alternatively, using the unplug/plug operations, individual PDBs can be upgraded as needed by moving to a CDB with the desired version.

- Fast thin provisioning at the SQL command line
  Oracle Multitenant supports thin provisioning of tenants using copy-on-write, enabling users to snapshot-clone databases using SQL commands.
- Self-Service Provisioning
  DBAs and application developers are empowered through self-provisioning interfaces, using SQL or the free PDB Self-Service application to create their own databases.

Oracle Multitenant is an innovative database consolidation technology for Oracle databases, introduced in Oracle 12*c*, which reduces capital expense by achieving efficient, dense database consolidation without sacrificing performance. The Multitenant architecture minimizes operational expense by leveraging a manage many-as-one framework and provides intuitive self-service interfaces for fast, autonomous, database provisioning.

Oracle Database Memory Structures

The Oracle Database memory structures most relevant for this study are depicted in **Figure 3**. The main memory structures of interest are the System Global Area (SGA) and the Process Global Area (PGA). The SGA is a large area of shared memory which all Oracle processes attach to and includes the buffer cache, shared pool, large pool, Java pool, shared IO pool, log buffer, and others. The PGA consists of private memory allocated by each database process individually and is used for sorting of result sets, parallel query execution, and other operations.

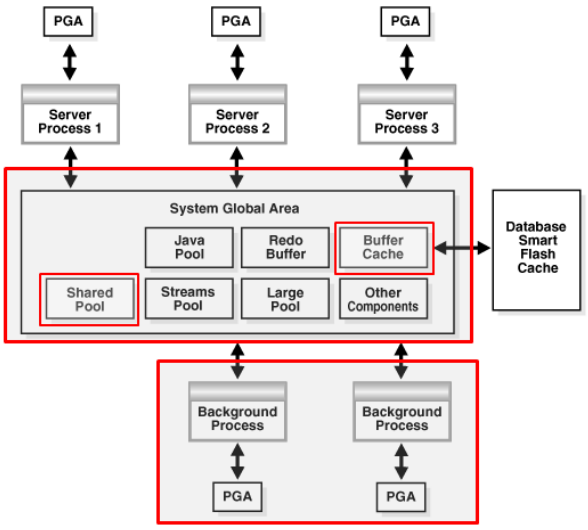

Figure 3. The Oracle Database Memory Structures

In the traditional non-CDBs consolidation model, each database instance has its own SGA, while in Oracle Multitenant, only a single SGA allocated by the CDB is shared among all PDBs. Within the SGA, the study mainly examines the consolidation efficiencies resulting from relative sizing of the database buffer cache and various other pools including shared pool, while savings in PGA memory

allow the use of a larger SGA in case of Oracle Multitenant. Combined, the savings in PGA memory and the efficient use of the shared pool allow a much larger allocation of memory to the buffer cache for PDBs than for non-CDBs using the same amount of total physical memory.

Oracle Database Processes Architecture

Oracle Database Server Processes

Oracle Database **server processes**, or also called **foreground processes**, are the processes created on behalf of user applications, and perform the following actions:

- Parse and run SQL statements issued through the application
- Read necessary data blocks from data files on disk into the shared database buffers of SGA if the blocks are not already present in the SGA
- Return results in such a way that the application can process the information

The application's performance, including transaction throughput and response time, is directly related to the behavior and efficiency of foreground processes.

Oracle Database Background Processes

The Oracle Database uses **background processes**[4] to implement supplementary database functionality which is not directly related to individual user sessions, such as asynchronous flushing of data blocks, writing and archiving of redo logs, and monitoring of other Oracle Database processes. Oracle background processes may be singleton processes that exist exactly once per database instance, or be scaled according to the number of CPUs, SQL directives, or initialization parameter settings.

The background processes most relevant for this study are:

- Log writer (*lgwr*) and log writer slaves (*lgnn*, new in Oracle 12*c*)
  Log writer and log writer slaves are responsible for synchronously flushing redo log to disk when transactions commit.
- Database writer (*dbwn* and *bwnn*)
  Database writers are responsible for asynchronously scanning the buffer cache for modified (dirty) buffers and flushing them to disk in the background.

[4] See http://docs.oracle.com/cd/E16655_01/server.121/e17615/bgprocesses.htm#REFRN104 for a list of Oracle background processes.

While the foreground processes directly relate to the application performance, the activity of background processes can also have a significant impact as foreground processes may compete with them for CPU cycles, and often rely on background activities, for example on log writers to flush redo buffers to disks (*log file sync*).

In this study, the number of Oracle Database background processes is one of the dominating factors for consolidation efficiency. In non-CDB consolidation, each database instance has its own set of background processes which only service work from their own database instance, while with Oracle Multitenant, a common set of background processes associated with the CDB performs work on behalf of all hosted PDBs.

Test Configurations

All experiments quoted in this white paper have been conducted on an Oracle SuperCluster T5-8, running Solaris 11 Update 1 and the latest Oracle 12.1.0.2 database software.

System Configuration:

Hardware - Oracle SuperCluster T5-8[5]:

- **Database Server:**

  Hardware: 1 T5-8, 8 sockets, 16-cores/socket, 3.6 GHz SPARC T5 processor (128 cores total). 2 TB Memory. L1 16KB data + 16KB instruction cache per core, L2 128K shared data and instruction cache per core, L3 8M cache per socket shared among cores.

  Operating System: Solaris 11 Update 1 SRU 16.

- **Storage Servers:**

  8 X3-2 Exadata Storage Servers, Software version OSS 11.2.3.3.0 connected with the compute node via InfiniBand.

Software - Oracle Database:

- Oracle 12.1.0.2.0 (release candidate)
  - Automatic Storage Management (ASM) with normal redundancy for diskgroups
  - Shared ORACLE_HOME for non-CDBs

[5] A full Rack SuperCluster T5-8 has one additional T5-8 server and a ZFS Storage Appliance, which have not been used in the experiments presented in this paper.

Workload:

The workload used in this study is a private OLTP workload with 5 different types of transactions and the following features:

- 72% write (insert/update/delete) transactions, which generate 5800 byte redo per transaction
- 28% read-only transactions including light-weight selects and mid-weight scans

Database Sizes:

While the data model is identical for all tests, the size of the consolidated databases has been varied to account for requirements of different tenants found in typical deployments. For the large-scale consolidation tests, a mix of 33% small, 33% medium, and 33% large databases with a size of 1 GB, 5 GB, and 15 GB per database has been provisioned. In these tests, the target transaction rate for each database is 97, 485, and 970 transactions per second for small, medium, and large databases. For the remaining tests with only 1 or 8 tenants, they use a much larger database size and transaction rate as described later.

Test Methodology:

The load is generated by two client machines, connected to the database server via 10 GbE. The load generator implements an open workload model using a connection pool and generates database requests at a configurable rate.

- Connection Pooling

  Connection pools are widely used by OLTP applications to eliminate the cost of establishing connections during runtime. In this study, one connection pool per database is used, sized twice as large as the average number of connections actually needed. This configuration better resembles real-world application deployments which require some headroom to accommodate load bursts or response time outliers.

- Open Workload Model

  Many performance tests and benchmarks use closed workload models in which the request arrival rate depends on the request completion rate and is therefore influenced by the performance of the system under test (SUT). This is unrealistic for real OLTP applications in which the request rate is solely triggered by the activity of external individuals. This study therefore uses an open workload model in which the load generator maintains a previously configured request arrival rate. Connections are acquired from the connection pool on demand to serve transactions. This model is more representative for OLTP workloads as it decouples the load generation from load consumption.

- System Utilization

  In all experiments, the transaction rate has been chosen such that the database server CPU utilization is close to 70% instead of being fully saturated, which is again more representative for real-world OLTP workloads that need headroom to accommodate sudden changes in load. Free memory on the server is kept around 500 GB for all tests (25% of total physical memory).

Metrics:

The study uses transaction processing throughput per second (reported as tps) and response time (reported in ms) as the primary metrics. For the detailed analysis, further database metrics such as *log file sync*, *log file parallel write*, buffer cache hit ratio, physical reads, as well as many system-level statistics such as CPU utilization, (involuntary) context switches, and I/O operations per second (IOPS) are used.

Monitoring Tools:

This study utilizes a large number of monitoring tools to analyze database and system performance and behavior, including:

- Oracle database monitors and reporters: AWR and ASH reports, and various v$ views
- Solaris monitoring utilities: iostat, mpstat, vmstat, prstat, DTrace, and Solaris Studio

Experiments:

This study evaluates and analyzes the consolidation efficiency, density, and elasticity of Oracle Multitenant pluggable databases (PDBs) in comparison to stacked single-instance databases (non-CDBs) in a series of four experiments.

- **Test 1: *Cost* - 1 non-CDB vs. 1 PDB**

  This test establishes the baseline performance of pluggable databases by comparing a traditional single instance database to a single PDB inside a CDB, showing that in-database virtualization with Oracle Multitenant imposes virtually no overhead.

- **Test 2: *Efficiency* - 252 non-CDBs vs. 252 PDBs**

  This test compares the efficiency of PDBs at large scale by consolidating 252 databases using PDBs vs. non-CDBs, analyzing in detail how savings in background processes and memory consumption contribute to 80% higher aggregate throughput of PDBs.

- **Test 3: *Density* – 168 non-CDBs vs. 252 PDBs**

  This test compares the consolidation density that is achievable for non-CDBs vs. PDBs if identical per-database throughput requirements must be supported in each deployment, showing that

Oracle Multitenant allows to consolidate 50% more active OLTP databases than is possible with consolidated single instances.

- **Test 4: *Elasticity* – 8 non-CDBs vs. 8 PDBs**

  This test demonstrates the elasticity of Oracle Multitenant in dynamically allocating buffer cache to pluggable databases as needed, whereas the buffer cache size for non-CDBs is static, which gives Oracle Multitenant a performance advantage over non-CDBs even at a smaller scale of only 8 consolidated databases.

Test 1: *Cost* – 1 non-CDB vs. 1 PDB

Many virtualization technologies impose an overhead that in some cases can be significant. This test demonstrates that in-database virtualization with Oracle Multitenant introduces nearly no overhead for the virtualized database. While transactions inside a PDB might have to take additional steps to access shared data in the CDB, the extra code path introduced by this is so insignificant that the cost of it is hardly measurable. For this test, the performance of a traditional (non-CDB) single instance database is compared to a single PDB running inside a CDB. Both databases run the same transaction rate on a single T5-8 socket.

Deployment	Throughput	Avg. Response Time	CPU Utilization
1 non-CDB	16,000 tps	4.8 ms	52%
1 PDB	16,000 tps	5.0 ms	54%

Table 1. Performance comparison of a non-CDB and a single PDB in a CDB

Key Take-Aways for Oracle Multitenant

- Minimum overhead compared to non-CDB (3~4%) introduced by extra calls of *kpdbSwitch()*

Result Summary

For some workloads, there can be a very small performance overhead for a PDB in terms of CPU consumption to perform the same amount of work as a non-CDB. It is primarily caused by an extra call to the *kpdbSwitch()* function whenever a session needs to switch between the PDB and CDB context to access shared data, which introduces a small overhead for OLTP workloads with high transaction rates. In this test, at a transaction rate of 18,000 tps, this function is responsible for a 4% increase (from 52% to 54%) in CPU utilization, while the difference in transaction response time is negligible. For workloads with lower transaction rates, such as DSS workloads, tests not further described in this paper have shown that this function is only called very infrequently, resulting in no measurable overhead for PDBs.

Test 2: *Efficiency* - 252 non-CDBs vs. 252 PDBs

This test compares the efficiency of Oracle Multitenant for large-scale database consolidation deployments of 252 databases, and provides a detailed analysis of savings in background processes and memory consumption and their effect on overall efficiency.

Deployment	Aggregate Throughput	Avg. Response Time	CPU Utilization	Memory Footprint per DB [6]	Storage IOPS
252 non-CDBs	72,600 tps	6.7 ms	68%	1702 MB	271,400
252 PDBs	130,300 tps	9.9 ms	68%	208 MB	131,200
PDBs vs. non-CDBs	**+80%**	**+3 ms**	**identical**	**-8x**	**-2x**

Table 2. Key performance metrics for efficiency

Key Take-Aways for Oracle Multitenant

- 80% higher throughput and 2 times fewer storage IOPS: efficiency gain from aggregating work (mainly log writing and database buffer flushing) from multiple databases in shared background processes
- 1.5 GB memory savings per database from sharing of SGA pools and background processes
- 92 times fewer background processes

Result Summary

At identical CPU utilization, PDBs achieve an 80% higher aggregate throughput than non-CDBs with slightly elevated response times and save 1.5 GB memory per database. The sharing of background processes allows PDBs to aggregate work – mainly log writing and flushing of database blocks – more efficiently, freeing CPU resources that can now be used for query processing instead, which results in higher throughput. While response times for non-CDBs are lower, they come at an enormous cost of not only loss in throughput, but also a 19 times higher redo log write rate. This requires the storage to sustain more than 100,000 IOPS for log writing alone (compared to 5,000 IOPS for all PDBs together), which – depending on the workload – might result in lack of storage performance for buffer cache misses or other storage operations (i.e. backup and restore). Considering this cost, an increase of response times by only 3 ms appears to be more than tolerable for an 80% throughput gain.

[6] The memory footprint of a database includes all private memory of background processes as well as shared memory for all SGA pools except buffer cache.

Analysis

Shared and Private Memory

Deployment	Shared Pool	Large Pool	Java Pool	Shared IO Pool	Total
per small non-CDB	260 MB	4 MB	4 MB	48 MB	26 GB
per medium non-CDB	752 MB	16 MB	16 MB	208 MB	81 GB
per large non-CDB	1376 MB	32 MB	64 MB	512 MB	163 GB
total 252 non-CDBs	**195.9 GB**	**4.3 GB**	**6.9 GB**	**63.0 GB**	**270 GB**
252 PDBs	**44.5 GB**	**0.5 GB**	**3.5 GB**	**0.5 GB**	**49 GB**
PDBs Saving	**151.4 GB**	**3.8 GB**	**3.4 GB**	**62.5 GB**	**221 GB**

Table 3. SGA memory pools

The SGA contains many memory pools that serve the database in different ways, including the shared pool, large pool, Java pool, and shared I/O pool. The largest of those, the shared pool, caches various types of program data including parsed SQL, PL/SQL code, system parameters, and data dictionary information. While these pools are critical for database functionality and to some extent performance, the most important part of the SGA is the buffer cache – a memory cache for data blocks (buffers). The sharing of SGA in Oracle Multitenant reduces the memory needs of the first-mentioned pools by storing many sharable contents in the root container (e.g. parent cursors, metadata for data dictionary tables and views, etc.). This results in a total of 220 GB memory previously allocated to these non-CDBs' SGA pools, and which can now be dedicated as shared memory to the CDB's buffer cache For PDB use.

Additional memory savings in Oracle Multitenant result from the reduced number of background processes. Each Oracle instance, regardless of whether non-CDB or CDB, requires a certain number of background processes, some of which are singletons, and others which typically scale with the number of CPUs. A small non-CDB as used in this test has between 30 and 35 background processes, while the large CDB has 91 of them (not including PQ slaves), including 32 database writers and 8 log writer slaves. Each background process requires private memory for process heap, stack, and anonymous memory. While most individual processes only need 20-50 MB of memory, large-scale consolidation of hundreds of non-CDBs leads to thousands of background processes which in total consume hundreds of GB of memory. For this test, the deployment of non-CDBs leads to 92 times more background processes than PDBs, which consume 150 GB of additional memory, 65 times more than for PDBs.

Deployment	Number of Backgrounds not including PQs	Private Memory of Backgroundss not including PQs
252 non-CDBs	8387	148.9 GB
252 PDBs	91	2.3 GB
PDBs Saving	**92 x fewer**	**146.6 GB**

Table 4. Background processes memory

Combining the savings from SGA and background processes, PDBs in total save 368 GB of memory compared to non-CDBs, which is a memory saving of 1.5 GB per database. Considering the database sizes of 1, 5, and 15 GB in this test, this saving is more than significant (or in other words, the per-database memory overhead for non-CDBs is very large). Given the same amount of physical memory in the server, this allows to increase the buffer cache by an equivalent amount while keeping free memory in the server constant. A larger buffer cache reduces cache misses and thus lowers the amount of necessary disk reads, as will be shown in the next sections.

Background CPU Time

The 92-fold increase in background processes for non-CDBs compared to PDBs does not only lead to wastage of memory, but also to increased CPU consumption. More CPU consumed by background processes results in fewer CPU cycles available for foreground processes. Since transaction throughput mainly depends on SQL processing performance of foreground processes, the increase in background CPU usage results in a decrease in foreground CPU (at identical overall CPU utilization) and thus in a loss of throughput.

Deployment	Background CPU Utilization	Foreground CPU Utilization	System CPU Utilization
252 non-CDBs	26% (7% usr, 19% sys)	35% (33% usr, 2% sys)	68% (40% usr, 28% sys)
252 PDBs	3 % (2% usr, 1% sys)	60% (56% usr, 4% sys)	68% (58% usr, 10% sys)

Table 5. CPU for background and foreground processes

Table 5 shows that background processes of non-CDBs consume 26% of the entire system's CPU capacity, while for PDBs they only consume 3%. With overall CPU utilization at 68% in both cases, this leaves, in the case of non-CDBs, only 35% of overall CPU time to foreground processes, while PDBs have 60% of the CPU available to their foreground processes. The remaining 7% (non-CDB) and 5% (PDB) CPU time are consumed by the kernel (slightly higher in case of non-CDBs because of increased storage I/O). Relative to non-CDBs, the foreground processes of PDBs have 71% more CPU time available to them, which allows them to achieve 80% higher throughput.

With so many more active processes for non-CDBs, the overall system behaves less efficient, resulting in twice as many involuntary context switches (*icsw*) and a higher thread migration rate (*migr*), which has an additional (though not significant) negative impact on CPU efficiency, in the case of non-CDBs (table 6). The increased involuntary context switches are mainly a result of additional I/O activity caused by preemptions of foreground processes, mostly from log writers (67%) and RDS kernel threads[7] (27%).

Deployment	icsw	migr
252 non-CDBs	95,200	603,000
252 PDBs	44,800	520,000
PDBs vs. non-CDBs	**-53%**	**-14%**

Table 6. Involuntary context switches and thread migrations per second

Among the many background processes, log writers and database writers contribute most to the high background CPU utilization and increased I/O requests for non-CDBs.

Log Writers (LGWRs)

When a session modifies a data block, it inserts a redo entry into the redo log buffer in the SGA. Once it commits, it needs to wait for the log writer to flush the redo buffer with all its records (and potentially those of other, concurrent sessions) to disk. For each of the sessions waiting for the log writer, their wait time is captured in the wait event *log file sync*. The log writer accumulates redo records of all sessions and flushes them in so-called *log writes* to disk as quickly as possible, where each log write can potentially consist of multiple physical writes. Each time, the log writer and all its slaves count the I/O time for each log write in the wait event *log file parallel write*.

Deployment	Number of LGWR (and slaves)	Log File Parallel Write		Log File Sync		Transactions per Log Write
		Waits per Second	Avg. Wait Time	Waits per Second	Avg. Wait Time	
252 non-CDBs	252	50,700	0.86 ms	52,700	1.67 ms	1
252 PDBs	1 + 8	3,300	1.54 ms	124,200	5.74 ms	38

Table 7. Redo log writing

[7] RDB (Reliable Datagram Sockets) is the protocol used in SuperCluster between the database server and the storage servers via InfiniBand. Solaris has special kernel threads to handle the RDS I/O requests.

Table 7 shows that the number of *log file sync* waits is roughly proportional to the transaction throughput. In the case of non-CDBs, 252 individual log writer processes flush redo to disk as quickly as possible. Since transaction concurrency per non-CDB is relatively low, they can hardly aggregate work from multiple concurrent transactions, which leads to one log write for each individual transaction. While this results in very small write sizes and therefore a low write time (0.86 ms) and session wait time (1.67 ms), it comes at a cost of a 15 times[8] higher log write rate resulting in more than 100,000 IOPS only for log writing (see below). This can be sustained with excellent response times by the Exadata Storage Servers used in SuperCluster T5-8, but might not be possible on other storage hardware. As described before, this write rate also comes at a high CPU cost. Oracle 12*c* introduced a scalable log writer architecture which allows for parallel log writing for large databases using log writer slaves. With pluggable databases, a single log writer and eight slaves efficiently aggregate work from all 252 PDBs and their sessions, which allows them to submit combined redo entries from (on average) 38 sessions in a single log write. Although the size of these writes is about 38 times larger than for non-CDBs, they only takes twice as long and results in only 4 ms additional *log file sync* wait time for sessions.

The transaction response time increase of 3 ms for PDBs compared to non-CDBs is exclusively caused by this 4 ms increase in *log file sync* time, which only affects write transactions. Due to the gains of CPU efficiency, the SQL processing of both read-only as well as write transactions is *faster* for PDBs. Workloads that are less write intensive than the one tested here will see an even smaller increase in transaction response times. Also workloads with more complex SQL processing, where SQL execution times dominate commit times, will be relatively less impacted by an increase in *log file sync*, even if the absolute increase is identical. For the workload used in this study, 4 ms *log file sync* times account for 40% of the transaction's total 10 ms response times. SQL processing only constitutes 60% of the elapsed time. That might not be typical for many applications.

Database Writers (DBWRs)

The effect from database writers is similar to that of log writers. In the non-CDB deployment, each instance requires at least 1 database writer, resulting in a total of 252 processes in this test. PDBs share the database writer processes of the CDB, which uses 32 of them for a container database of this size, and allows them to aggregate more work each time they flush dirty buffers to disk (*db file parallel write*). This results in 2.8 times fewer flushes (table 8) and – as more blocks are aggregated in one flush – 25% fewer disk writes (table 10)[9], and contributes to the savings in background CPU time described above.

[8] One *log file parallel write* can cause one or multiple, or in some cases zero, writes to the log device, that's why in later part the IOPS on log device is 19 times higher for NON-CDBs than PDBs while there *log file parallel write* is 15 times higher.

[9] One *db file parallel write* can contain one or multiple physical writes to the disk.

Deployment	Number of DBWRs	Db File Parallel Write	
		Waits per second	Avg. wait time
252 non-CDBs	252	46,900	0.94 ms
252 PDBs	32	16,900	0.94 ms

Table 8. Buffer cache flushing

Buffer Cache

The memory savings of private memory and other SGA pools described earlier allow more dedicates buffer cache memory to PDBs than to non-CDBs. Table 9 shows how the larger amount of buffer cache for PDBs translate into fewer buffer cache misses than for non-CDBs, which miss cache more often and require physical reads from the storage to bring missing buffers into the cache. Single instances face a 2-3 times higher buffer cache miss rate than PDBs. Since their aggregate transaction rate is much lower and the storage is very fast, these misses only cause a 6% increase in I/O wait time (table 9), but lead to a 45% increase in disk reads (table 10).

Deployment	Buffer Cache	Buffer Cache Miss Rate	I/O Wait Time per Second
per small non-CDB	0.73 GB	0.59%	27 ms
per medium non-CDB	3.22 GB	0.38%	119 ms
per large non-CDB	7.53 GB	0.46%	365 ms
252 non-CDBs	**964 GB**	**--- ---**	**42,900 ms**
252 PDBs	**1,250 GB**	**0.19%**	**40,600 ms**

Table 9. Buffer cache misses and I/O wait time

Storage IOPS

Exadata Storage Servers used in SuperCluster T5-8 deliver highest storage performance specially designed for the Oracle database. Each storage server is equipped with hard drives and flash, which is used as cache for data files as well as an accelerator for redo writes.

The reduced buffer cache misses and increased efficiencies of shared database writer and especially log writer processes allow PDBs to achieve an 80% higher transaction throughput at less than half the number of storage IOPS (table 10). Normalized per transaction, non-CDBs generate 3.7 times more IOPS than PDBs. With a less powerful storage than the storage used in these tests, non-CDBs' performance would have suffered much more. Log writing contributes most to the high IOPS rate of non-CDBs with 19 times more writes (100,000 more IOPS) than for PDBs.

Deployment	Type	Read IOPS	Write IOPS	KB per Read	KB per Write
252 non-CDBs	Data File	36,900	133,100	8.1	13.5
	Redo Log	0	101,400	n/a	8.6
	Total IOPS		271,400	n/a	n/a
252 PDBs	Data File	25,400	100,400	8.3	17.1
	Redo Log	0	5,400	n/a	205.2
	Total IOPS		131,200	n/a	n/a

Table 10. Storage I/O

Test 3: *Density* – 168 non-CDBs vs. 252 PDBs

While the previous test explained the efficiency gains of Oracle Multitenant by focusing on the achievable aggregate throughput for the same number of databases in both deployments, this test translates these savings into real life benefits. Rather than fixing the number of databases, it assumes a given throughput requirement per database and answers the question of how many databases can be consolidated on a fixed amount of hardware resources. This translates into hardware cost savings for Oracle Multitenant by reducing the amount of hardware needed to support all tenants in a data center.

Tenant	Database Size	Target Throughput	Number of Tenants
Small	1 GB	97 tps	33%
Medium	5 GB	485 tps	33%
Large	15 GB	970 tps	33%

Table 11. Tenant database sizes and transaction throughput requirements

Target throughput and database sizes for this test are identical to those of the efficiency tests, which resulted in 68% CPU utilization for 252 PDBs. For this test, the results for Oracle Multitenant are therefore identical to those of the previous test. The objective of this test is to maximize the number of supported non-CDBs given their above described throughput requirements, without exceeding 68% CPU utilization and using more than 75% of the system's total memory. Table 11 shows the database sizes and throughput requirements for small, medium, and large tenants, which each account for 33% of the deployed tenants for both PDBs and non-CDBs.

Deployment	Consolidated Tenants	Aggreate Throughput	Avg. Response Time	CPU Utilization	Cores	Storage IOPS
non-CDBs	168	86,900 tps	7.3 ms	68%	128	237,000
PDBs	252	130,300 tps	9.9 ms	68%	128	131,200
PDBs vs. non-CDBs	**+50%**	**+50%**	**+2.6 ms**	**identical**	**identical**	**-2x**

Table 12. Consolidation density on T5-8 with non-CDBs vs. with PDBs

Key Take-Aways for Oracle Multitenant

- 50% more tenants consolidated on the same hardware
- 33% fewer cores needed (allows to consolidate all tenants in a single server)
- 3x reduction in storage IOPS to support the same number of tenants

Result Summary

Because of the overhead from background processes and memory, non-CDBs only allow to consolidate 168 databases while satisfying the throughput requirements of each tenant. Using Oracle Multitenant, the gains in efficiency described before allow a total of 252 databases to be consolidated on the same hardware, using the same amount of CPU and memory resources, and additionally reducing storage IOPS by a factor of 2 (table 12). By converting these results into hardware requirements for consolidating the *same* number of tenants also with non-CDBs, 50% more cores (and one additional server) as well as 3 times more storage I/O capacity would be needed if databases were consolidated with single instances. Oracle Multitenant therefore not only reduces operational costs through simplified management, but achieves further cost savings though improved consolidation density, which reduces both capital and operational expenditures even more.

Deployment	CPU Cores	Storage IOPS
non-CDBs	192	355,500
PDBs	128	131,200
PDBs vs. non-CDBs	**-33%**	**-3x**

Table 13. Hardware requirements for 252 databases as specified in table 11.

Test 4: *Elasticity* – 8 non-CDBs vs. 8 PDBs

The previous tests have shown performance characteristics of consolidated databases under a steady (constant) load. In most real deployments, load fluctuates over time, which creates additional opportunities for efficiency if resources can be dynamically shared between databases. The non-CDBs deployments used in this test already allows instances to share CPU (which is not always the case for VM-based configurations), but buffer cache is still statically allocated per non-CDB. This test therefore focuses on the elasticity of Oracle Multitenant in dynamically allocating buffer cache to pluggable databases as needed. While previous tests demonstrated gains in efficiency at a large scale, this test shows that even at a smaller scale of only 8 consolidated databases, the elasticity of pluggable databases allows on-demand resource assignments, which gives PDBs superior performance over non-CDBs. In both deployments, load is generated using an identical sinus curve for all 8 databases, which is slightly shifted between databases so that their peaks do not fully align but still partially overlap. The configured average and peak loads for non-CDBs and PDBs are identical.

Key Take-Aways for Oracle Multitenant

- Dynamic allocation of shared buffer cache leads to 2.4 ms better response times, 20% fewer storage IOPS, and 38% reduced I/O Wait Time
- Dynamic resource sharing allows PDBs to allocate enough buffer cache to peak consumer, while non-CDBs fail to reach peak throughput due to static allocation

Result Summary

Deployment	Aggregate Throughput	Avg. Response Time	Disk I/O		Wait Events	
			Read IOPS	Write IOPS	I/O Wait Time (per second)	I/O Wait Time (per transaction)
8 non-CDBs	80,340 tps	13.8 ms	150,000	390,400	541 s	6.73 ms
8 PDBs	80,440 tps	11.4 ms	139,400	294,000	336 s	4.18 ms
PDBs vs. non-CDBs	**non-CDB unsteady** **PDB steady**	**-2.4 ms**	**-7%**	**-25%**	**-38%**	**-38%**

Table 14. Key performance metrics for elasticity

In this test, 8 databases with a size of 500 GB each are consolidated, using an aggregate buffer cache of 1192 GB (non-CDBs) and 1250 GB (PDBs). Even though the savings in private memory and other memory pools are not that significant at a scale of 8 databases, PDBs benefit additionally from the sharing of a common buffer cache. While overall throughput over time is similar between both deployments (the load generator queues requests and eventually issues them at a later point in time if they cannot be served immediately), it is very unsteady for non-CDBs where individual databases fail to reach their configured peak throughput (figure 4).

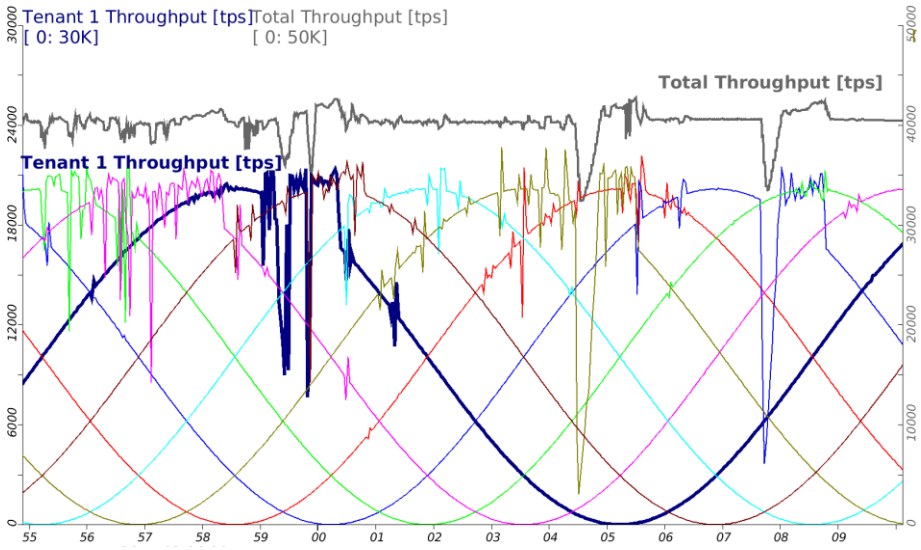

Figure 4. Per-Tenant and aggregate throughput for 8 non-CDBs

PDBs efficiently share the common buffer cache of PDBs, which reduces both pressure on flushing of dirty buffers (25% fewer write IOPS) as well as physical reads (reduced by 7%). This reduced I/O wait time for PDBs by 38%, resulting in 2.5 ms lower response times for PDBs, which all successfully reach their individual target throughput (figure 5).

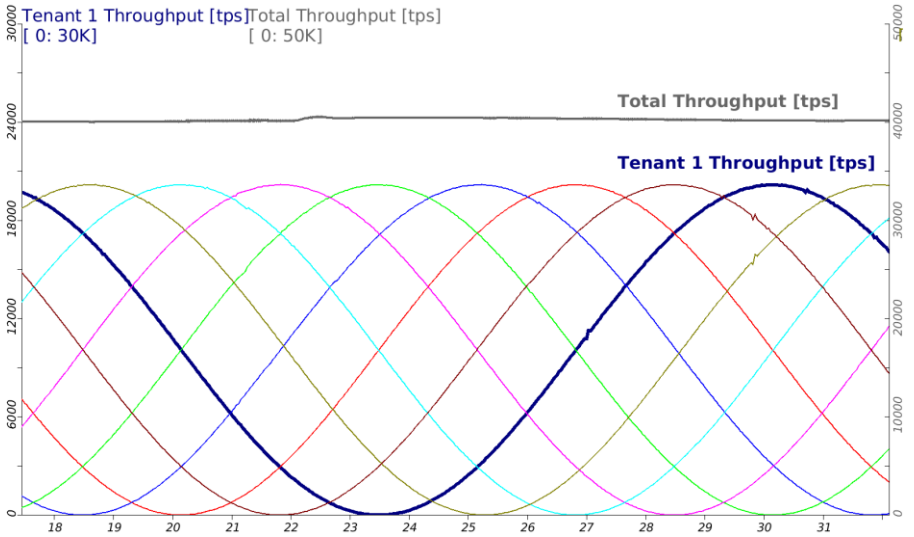

Figure 5. Per-Tenant and aggregate throughput for 8 PDBs

Overall Summary

The tests presented in this white paper demonstrate the efficiency gains from sharing of background processes and memory in Oracle Multitenant. The aggregation of work in background processes across pluggable databases saves a significant amount of CPU time and allows to reduce the number of cores needed to support 252 active OLTP databases from 196 to 128 cores. Aggregation of redo log and data block flushing across tenants further reduces the requirement on storage I/O from 356,000 IOPS to 131,000 IOPS. The sharing of SGA pools and background processes reduces the memory footprint of a database significantly and saves a total of 368 GB memory, which either translates into hardware cost savings, or can be *reinvested* by increasing the buffer cache of the CDB. Dynamic workloads benefit further from the on-demand allocation of buffer cache, which helps them to accommodate load peaks that they might not be able to satisfy with a static resource allocation.

The analysis of the test results allows to estimate savings also for other workloads than the one used in this white paper:

Write-intensive workloads will benefit from the aggregation of work in database writer and log writer processes, which results in lower CPU consumption for background processes and fewer disk writes. The saved CPU resources are available for more SQL processing instead. While individual log writer processes in non-CDBs might achieve lower *log file sync* times, they only accomplish this at the cost of a high rate of disk writes, which requires a very high-performing storage. With a slower storage, those wait times could be similar or even higher for non-CDBs than for PDBs.

Read-intensive workloads benefit from the reduced memory footprint of PDBs, which allows to consolidate more of them on the same amount of hardware. Since they commit less frequently, their response time is mostly affected by SQL processing and buffer cache efficiency rather than *log file sync*, and can be even better with PDBs than with non-CDBs due to an overall large buffer cache.

Workloads with more complex SQL, even if highly transactional, will be more impacted by SQL execution time instead of commit time. Since SQL execution can be more efficient in Oracle Multitenant with less disturbance from background processes, any potential increase in *log file sync* times might be more than compensated for by faster SQL processing.

Workloads with dynamic behavior can largely benefit from on-demand allocation of buffer cache, CPU time, and other resources in Oracle Multitenant, whereas single instances or virtualized deployments allocate many of these resources statically, which can lead to large over-provisioning and low resource utilization.

Conclusions

Oracle Multitenant is a new and innovative technology for in-database virtualization, where multiple pluggable databases inside a common container database are hosted. Compared to other virtualization technologies, it introduces nearly no overhead. Oracle Multitenant improves efficiency, density, and elasticity of database consolidation through sharing of memory structures and background processes compared to deployments of individual database instances on bare-metal or within virtual machines. The tests presented in this paper demonstrate that Oracle Multitenant not only reduces administrative costs through a simplified management model, but also allows consolidation of a much larger number of databases on the same (or even less) amount of hardware resources. Oracle SuperCluster T5-8 is the ideal consolidation platform for large-scale database consolidation. It provides powerful compute resources, a vast amount of physical memory, as well as database-optimized storage to easily support a large number of highly active OLTP databases consolidated with Oracle Multitenant.

In Oracle Multitenant, great savings in CPU consumption arise from the sharing of background processes among databases, which can significantly reduce the amount of cores needed to support a given number of databases. By aggregating redo log and data block flushing across tenants, Oracle Multitenant further reduces the requirement on storage I/O. Pluggable databases share both processes and SGA of a single container database, which greatly reduces the memory footprint of a database. Further benefits arise from the dynamic allocation of buffer cache, allowing Oracle Multitenant to quickly react to a tenant's change in demand, whereas single database instances have a static allocation of buffer cache that cannot be dynamically reassigned.

The combination of Oracle Multitenant and SuperCluster T5-8 provide unmatched efficiency, density and elasticity for database consolidation and lowers capital and operational costs through reduced management complexity and hardware resources.

Oracle Multitenant on SuperCluster T5-8:
Study of Database Consolidation Efficiency
April 2014
Authors: Yixiao Shen, Nicolas Michael
Contributing Authors: John McHugh

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

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

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

Hardware and Software, Engineered to Work Together